





[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

<b>TRACE32 Documents</b> .....	
<b>ICD In-Circuit Debugger</b> .....	
<b>Processor Architecture Manuals</b> .....	
<b>ARM/CORTEX/XSCALE</b> .....	
<b>ARM Debugger</b> .....	<b>1</b>
<b>History</b> .....	<b>8</b>
<b>Warning</b> .....	<b>9</b>
<b>Introduction</b> .....	<b>10</b>
Brief Overview of Documents for New Users	10
Demo and Start-up Scripts	11
<b>Quick Start of the JTAG Debugger</b> .....	<b>13</b>
<b>Troubleshooting</b> .....	<b>15</b>
Communication between Debugger and Processor cannot be established	15
<b>FAQ</b> .....	<b>16</b>
ARM	16
ARM7	19
JANUS	20
ARM9	22
ARM10	22
ARM11	23
Cortex-A/R (ARMv7, 32-bit)	24
XSCALE	26
<b>Trace Extensions</b> .....	<b>27</b>
<b>Symmetric Multiprocessing</b> .....	<b>28</b>
<b>ARM Specific Implementations</b> .....	<b>29</b>
TrustZone Technology	29
Debug Permission	29
Checking Debug Permission	30
Checking Secure State	30
Changing the Secure State from within TRACE32	30
Accessing Memory	30

Accessing Coprocessor CP15 Register	31
Accessing Cache and TLB Contents	31
Breakpoints and Vector Catch Register	31
Breakpoints and Secure Modes	31
big.LITTLE	32
Debugger Setup	32
Consequence for Debugging	33
Requirements for the Target Software	33
big.LITTLE MP	33
Breakpoints	34
Software Breakpoints	34
On-chip Breakpoints for Instructions	34
On-chip Breakpoints for Data	34
Hardware Breakpoints (Bus Trace only)	35
Example for Standard Breakpoints	36
Complex Breakpoints	42
Direct ICE Breaker Access	42
Example for ETM Stopping Breakpoints	43
Access Classes	44
Coprocessors	52
Accessing Memory at Run-time	54
Semihosting	58
SVC (SWI) Emulation Mode	58
DCC Communication Mode (DCC = Debug Communication Channel)	60
Virtual Terminal	62
Large Physical Address Extension (LPAE)	63
Consequence for Debugging	63
Virtualization Extension, Hypervisor	64
Consequence for Debugging	64
Runtime Measurement	64
Trigger	64
<b>ARM specific SYStem Commands .....</b>	<b>65</b>
SYStem.CLOCK	Inform debugger about core clock 65
SYStem.CONFIG.state	Display target configuration 65
SYStem.CONFIG	Configure debugger according to target topology 66
<parameters> describing the “DebugPort”	73
<parameters> describing the “JTAG” scan chain and signal behavior	78
<parameters> describing a system level TAP “Multitap”	82
<parameters> configuring a CoreSight Debug Access Port “DAP”	84
<parameters> describing debug and trace “Components”	88
<parameters> which are “Deprecated”	98
SYStem.CONFIG SMMU	Internal use 102
SYStem.CPU	Select the used CPU 104

SYStem.CpuAccess	Run-time memory access (intrusive)	105
SYStem.JtagClock	Define JTAG frequency	106
SYStem.LOCK	Tristate the JTAG port	108
SYStem.MemAccess	Run-time memory access	109
SYStem.Mode	Establish the communication with the target	113
SYStem.Option	Special setup	116
SYStem.Option ABORTFIX	Do not access memory area from 0x0 to 0x1f	116
SYStem.Option AHBHPROT	Select AHB-AP HPROT bits	116
SYStem.Option AMBA	Select AMBA bus mode	116
SYStem.Option ASYNCBREAKFIX	Asynchronous break bugfix	117
SYStem.Option AXIACEEnable	ACE enable flag of the AXI-AP	117
SYStem.Option AXICACHEFLAGS	Select AXI-AP CACHE bits	117
SYStem.Option AXIHPROT	Select AXI-AP HPROT bits	118
SYStem.Option BUGFIX	Breakpoint bug fix	118
SYStem.Option BUGFIXV4	Asynch. break bug fix for ARM7TDMI-S REV4	119
SYStem.Option BigEndian	Define byte order (endianness)	120
SYStem.Option BOOTMODE	Define boot mode	120
SYStem.Option CINV	Invalidate the cache after memory modification	121
SYStem.Option CFLUSH	FLUSH the cache before step/go	121
SYStem.Option CacheParam	Define external cache	121
SYStem.Option DACR	Debugger ignores DACR access permission settings	122
SYStem.Option DAPDBGPWRUPREQ	Force debug power in DAP	122
SYStem.Option DAP2DBGPWRUPREQ	Keep forcing debug power in DAP2	123
SYStem.Option DAPSYSPWRUPREQ	Force system power in DAP	123
SYStem.Option DAP2SYSPWRUPREQ	Force system power in DAP2	124
SYStem.Option DAPNOIRCHECK	No DAP instruction register check	124
SYStem.Option DAPREMAP	Rearrange DAP memory map	125
SYStem.Option DBGACK	DBGACK active on debugger memory accesses	125
SYStem.Option DBGNOPWRDWN	DSCR bit 9 will be set in debug mode	125
SYStem.Option DBGUNLOCK	Unlock debug register via OSLAR	126
SYStem.Option DCDIRTY	Bugfix for erroneously cleared dirty bits	126
SYStem.Option DCFREEZE	Disable data cache linefill in debug mode	126
SYStem.Option DEBUGPORTOptions	Options for debug port handling	127
SYStem.Option DIAG	Activate more log messages	127
SYStem.Option DisMode	Define disassembler mode	128
SYStem.Option DynVector	Dynamic trap vector interpretation	129
SYStem.Option EnReset	Allow the debugger to drive nRESET (nSRST)	129
SYStem.Option ETBFIXMarvell	Read out on-chip trace data	129
SYStem.Option ETMFX	Shift data of ETM scan chain by one	130
SYStem.Option ETMFXWO	Bugfix for write-only ETM register	130
SYStem.Option ETMFX4	Use only every fourth ETM data package	130
SYStem.Option EXEC	EXEC signal can be used by bustrace	130
SYStem.Option EXTBYPASS	Switch off the fake TAP mechanism	131

SYStem.Option FASTBREAKDETECTION	Fast core halt detection	131
SYStem.Option HRCWOVerRide	Enable override mechanism	131
SYStem.Option ICEBreakerETMFiXMarvell	Lock on-chip breakpoints	132
SYStem.Option ICEPiCK	Enable/disable assertions and wait-in-reset	132
SYStem.Option IMASKASM	Disable interrupts while single stepping	132
SYStem.Option IMASKHLL	Disable interrupts while HLL single stepping	133
SYStem.Option INTDiS	Disable all interrupts	133
SYStem.Option IRQBREAKFiX	Break bugfix by using IRQ	133
SYStem.Option KEYCODE	Define key code to unsecure processor	134
SYStem.Option L2Cache	L2 cache used	134
SYStem.Option L2CacheBase	Define base address of L2 cache register	134
SYStem.Option LOCKRES	Go to 'Test-Logic Reset' when locked	135
SYStem.Option MACHINESPACES	Address extension for guest OSes	136
SYStem.Option MEMORYHPRoT	Select memory-AP HPROT bits	136
SYStem.Option MemStatusCheck	Check status bits during memory access	137
SYStem.Option MMUSPACES	Separate address spaces by space IDs	137
SYStem.Option MonitorHoldoffTime	Delay between monitor accesses	138
SYStem.Option MPU	Debugger ignores MPU access permission settings	138
SYStem.Option MultiplesFiX	No multiple loads/stores	138
SYStem.Option NODATA	No data connected to the trace	139
SYStem.Option NOiRCHECK	No JTAG instruction register check	139
SYStem.Option NoPRCRReset	Do not cause reset by PRCR	139
SYStem.Option NoRunCheck	No check of the running state	140
SYStem.Option NoSecureFix	Do not switch to secure mode	140
SYStem.Option OVERLAY	Enable overlay support	141
SYStem.Option PALLADIUM	Extend debugger timeout	141
SYStem.Option PC	Define address for dummy fetches	142
SYStem.Option PROTECTION	Sends an unsecure sequence to the core	142
SYStem.Option PWRCHECK	Check power and clock	142
SYStem.Option PWRCHECKFiX	Check power and clock	143
SYStem.Option PWRDWN	Allow power-down mode	143
SYStem.Option PWRDWNRecover	Mode to handle special power recovery	144
SYStem.Option PWRDWNRecoverTimeOut	Timeout for power recovery	144
SYStem.Option PWROVR	Specifies power override bit	144
SYStem.Option ResBreak	Halt the core after reset	145
SYStem.Option ResetDetection	Choose method to detect a target reset	146
SYStem.Option RESeTREGiSter	Generic software reset	146
SYStem.Option RESTARTFiX	Wait after core restart	147
SYStem.Option RisingTDO	Target outputs TDO on rising edge	147
SYStem.Option ShowError	Show data abort errors	148
SYStem.Option SOFTLONG	Use 32-bit access to set breakpoint	148
SYStem.Option SOFTQUAD	Use 64-bit access to set breakpoint	148
SYStem.Option SOFTWORD	Use 16-bit access to set breakpoint	149

SYStem.Option SPLIT	Access memory depending on CPSR	149
SYStem.Option StandByTraceDelaytime	Trace activation after reset	149
SYStem.Option STEPSOFT	Use software breakpoints for ASM stepping	149
SYStem.Option SYSPWRUPREQ	Force system power	150
SYStem.Option TIDBGEN	Activate initialization for TI derivatives	150
SYStem.Option TIETMFX	Bug fix for customer specific ASIC	150
SYStem.Option TIDEMUXFIX	Bug fix for customer specific ASIC	150
SYStem.Option TraceStrobe	Deprecated command	151
SYStem.Option TRST	Allow debugger to drive TRST	151
SYStem.Option TURBO	Speed up memory access	151
SYStem.Option WaitIDCODE	IDCODE polling after deasserting reset	152
SYStem.Option WaitReset	Wait with JTAG activities after deasserting reset	152
SYStem.Option.WATCHDOG	Disable watchdog while debugging	153
SYStem.Option ZoneSPACES	Enable symbol management for ARM zones	154
Overview of Debugging with Zones		155
Operation System Support - Defining a Zone-specific OS Awareness		158
SYStem.Option ZYNQJTAGINDEPENDENT	Configure JTAG cascading	160
SYStem.RESetOut	Assert nRESET/nSRST on JTAG connector	160
SYStem.state	Display SYStem window	161
<b>ARM Specific Benchmarking Commands .....</b>		<b>162</b>
BMC.EXPORT	Export benchmarking events from event bus	162
BMC.MODE	Define the operating mode of the benchmark counter	163
BMC.<counter>.EVENT	Configure the performance monitor	164
Functions		167
BMC.PRESCALER	Prescale the measured cycles	168
BMC.<counter>.RATIO	Set two counters in relation	168
BMC.TARA	Calibrate the benchmark counter	169
<b>ARM Specific TrOnchip Commands .....</b>		<b>170</b>
TrOnchip.A	Programming the ICE breaker module	170
TrOnchip.A.Value	Define data selector	170
TrOnchip.A.Size	Define access size for data selector	170
TrOnchip.A.CYcle	Define access type	171
TrOnchip.A.Address	Define address selector	172
TrOnchip.A.Trans	Define access mode	173
TrOnchip.A.Extern	Define the use of EXTERN lines	173
TrOnchip.AddressMask	Define an address mask	174
TrOnchip.MatchASID	Extend on-chip breakpoint/trace filter by ASID	174
TrOnchip.MatchZone	Extend on-chip breakpoint/trace filter by zone	175
TrOnchip.ContextID	Enable context ID comparison	176
TrOnchip.CONVert	Allow extension of address range of breakpoint	176
TrOnchip.Mode	Configure unit A and B	177
TrOnchip.RESet	Reset on-chip trigger settings	178
TrOnchip.Set	Set bits in the vector catch register	178

TrOnchip.TEnable	Define address selector for bus trace	179
TrOnchip.TCYcle	Define cycle type for bus trace	180
TrOnchip.VarCONVert	Convert breakpoints on scalar variables	181
TrOnchip.state	Display on-chip trigger window	182
<b>CPU specific MMU Commands</b>		<b>183</b>
MMU.DUMP	Page wise display of MMU translation table	183
MMU.List	Compact display of MMU translation table	187
MMU.SCAN	Load MMU table from CPU	189
<b>CPU specific SMMU Commands</b>		<b>191</b>
SMMU	Hardware system MMU (SMMU)	191
SMMU.ADD	Define a new hardware system MMU	195
SMMU.Clear	Delete an SMMU	196
SMMU.Register	Peripheral registers of an SMMU	197
SMMU.Register.ContextBank	Display registers of context bank	198
SMMU.Register.Global	Display global registers of SMMU	199
SMMU.Register.StreamMapRegGrp	Display registers of an SMRG	200
SMMU.RESet	Delete all SMMU definitions	201
SMMU.SSDtable	Display security state determination table	202
SMMU.StreamMapRegGrp	Access to stream map table entries	204
SMMU.StreamMapRegGrp.ContextReg	Display context bank registers	205
SMMU.StreamMapRegGrp.Dump	Page-wise display of SMMU page table	207
SMMU.StreamMapRegGrp.List	List the page table entries	209
SMMU.StreamMapTable	Display a stream map table	210
<b>Target Adaption</b>		<b>217</b>
Probe Cables		217
Interface Standards JTAG, Serial Wire Debug, cJTAG		217
Connector Type and Pinout		217
Debug Cable		217
CombiProbe		217
Preprocessor		218
<b>Support</b>		<b>219</b>
Available Tools		219
ARM7		219
ARM9		229
ARM10		238
ARM11		238
Cortex-A/R		240
Compilers		260
Target Operating Systems		261
Boot Loaders		262
3rd-Party Tool Integrations		263
<b>Products</b>		<b>264</b>

Product Information	264
ARM7	264
ARM9	266
ARM10	267
ARM11	268
Cortex-A/R	269
Order Information	270
ARM7	270
ARM9	271
ARM10	273
ARM11	275
Cortex-A/-R	277

## History

---

- 21-Sep-18 Added description for the command [TrOnchip.MatchZone](#).
- 27-Mar-18 New command [SYStem.Option.WaitIDCODE](#).
- 23-Jan-18 New chapter “[Accessing Memory at Run-time](#)”.
- 11-Jan-18 Added description for the new internal command [SYStem.CONFIG.SMMU](#).
- 12-Dec-17 Added description for [SYStem.Option WATCHDOG](#).
- 04-Oct-17 Added description for new options in chapter “[CPU specific MMU Commands](#)”.
- 19-Sep-17 Reworked section “[Access Classes](#)”.
- 07-Aug-17 New command [SYStem.Option MACHINESPACES](#).
- 22-May-17 Revised the descriptions of the commands [TrOnchip.CONVert](#) and [TrOnchip.VarCONVert](#).



**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.

This document describes the processor-specific settings and features for the Cortex-A/R (ARMv7, 32-bit) debugger.

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

---

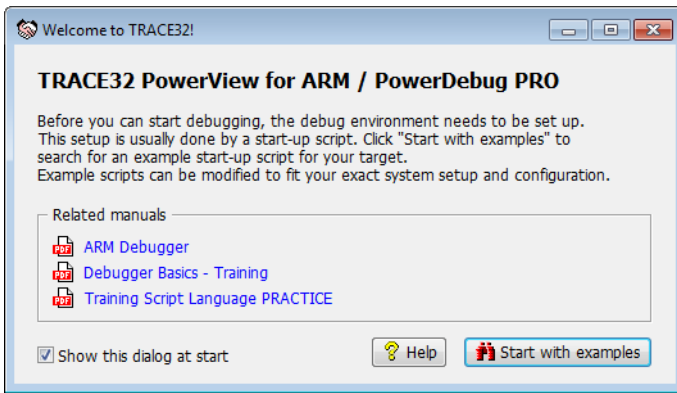
### Architecture-independent information:

- **“Debugger Basics - Training”** (training\_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app\_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general\_ref\_<x>.pdf): Alphabetic list of debug commands.

### Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos\_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.
- This manual does not cover the Cortex-A/R (ARMv8, 32/64-bit) cores. If you are using this processor architecture, please refer to **“ARMv8-A-R Debugger”** (debugger\_armv8a.pdf).
- This manual does not cover the Cortex-M processor architecture. If you are using this processor architecture, please refer to **“Cortex-M Debugger”** (debugger\_cortexm.pdf) for details.

To get started with the most important manuals, use the **WELCOME.view** dialog:



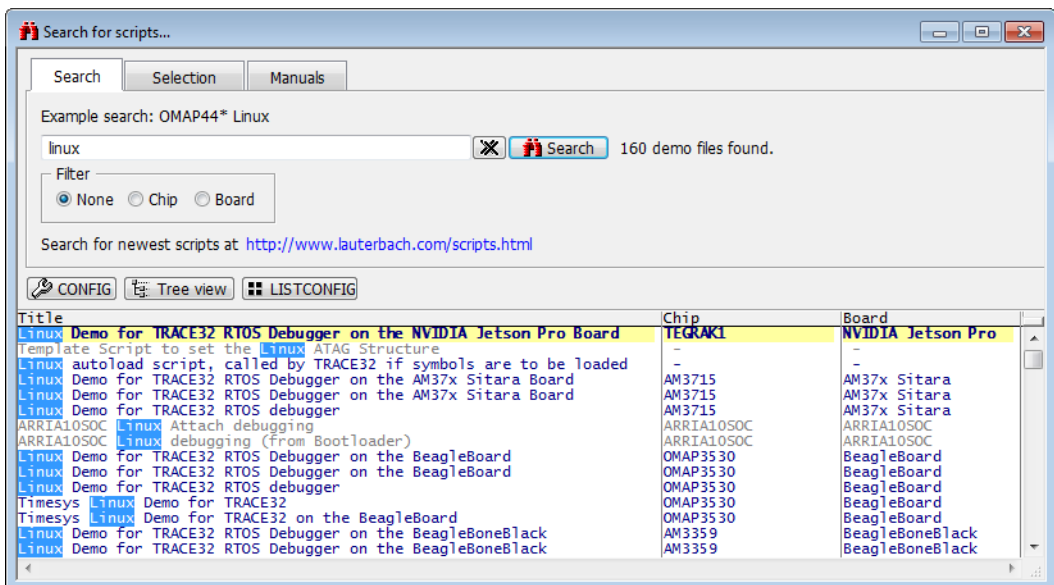
## Demo and Start-up Scripts

Lauterbach provides ready-to-run PRACTICE start-up scripts for public known architecture 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 inspect the demo folder manually in the root installation directory of TRACE32.

The `~/demo/arm/` folder contains:

<b>hardware/</b>	Ready-to-run debugging and flash programming demos for evaluation boards. <b>Recommended for getting started!</b>
<b>combiprobe/</b>	CombiProbe-specific examples.
<b>bootloader</b>	Examples for uboot, uefi and other bootloaders.
<b>compiler/</b>	Hardware independent compiler examples.
<b>etc/</b>	Various examples, e.g. data trace, terminal application, ...
<b>fdx/</b>	Example applications for the FDX feature.
<b>flash/</b>	Binaries for target based programming and example declarations for internal flash.
<b>kernel/</b>	Various OS Awareness examples.
<b>simul/</b>	Examples of peripheral simulation models, which extend the functionality of the TRACE32 Instruction Set Simulator.

# Quick Start of the JTAG Debugger

Starting up the debugger is done as follows:

1. Reset the debugger.

```
RESet
```

The **RESet** command ensures that no debugger setting remains from a former debug session. All settings get their default value. **RESet** is not required if you start the debug session directly after booting the TRACE32 development tool. **RESet** does not reset the target.

2. Select the chip or core you intend to debug.

```
SYStem.CPU <cputype>
```

Based on the selected chip the debugger sets the **SYStem.CONFIG** and **SYStem.Option** commands the way which should be most appropriate for debugging this chip. Ideally no further setup is required.

If you select a Cortex-A or Cortex-R core instead of a chip (e.g. "SYStem.CPU CortexR4") then you need to specify the base address of the debug register block:

```
SYStem.CONFIG.COREDEBUG.Base <address>
```

3. Connect to target.

```
SYStem.Up
```

This command establishes the JTAG communication to the target. It resets the processor and enters debug mode (halts the processor; ideally at the reset vector). After this command is executed, it is possible to access memory and registers.

Some devices can not communicate via JTAG while in reset or you might want to connect to a running program without causing a target reset. In this case use

```
SYStem.Mode Attach
```

instead. A "Break" will halt the processor.

4. Load the program you want to debug.

```
Data.LOAD armle.axf
```

This loads the executable to the target and the debug/symbol information to the debugger's host. If the program is already on the target then load with **/NoCODE** option.

A start sequence example is shown below. This sequence can be written to an ASCII file (script file) and executed with the command **DO** *<filename>*.

```
WinCLEAR                ; Clear all windows

SYStem.CPU ARM940T       ; Select the core type

MAP.BOnchip 0x100000++0xffff ; Specify where FLASH/ROM is

SYStem.Up               ; Reset the target and enter debug mode

Data.LOAD armle.axf      ; Load the application

Register.Set pc main     ; Set the PC to function main

Register.Set r13 0x8000   ; Set the stack pointer to address 8000

PER.view                ; Show clearly arranged peripherals
                        ; in window *)

List                    ; Open source code window *)

Register /SpotLight      ; Open register window *)

Frame.view /Locals /Caller ; Open the stack frame with
                        ; local variables *)

Var.Watch var1 var2      ; Open watch window for variables *)

Break.Set 0x1000 /Program ; Set software breakpoint to address
                        ; 1000 (address 1000 outside of BOnchip
                        ; range)

Break.Set 0x101000 /Program ; Set on-chip breakpoint to address
                        ; 101000 (address 101000 is within
                        ; BOnchip range)
```

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

## Communication between Debugger and Processor cannot be established

---

Typically the **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages like “debug port fail” or “debug port time out” while executing this command, this may have the reasons below. “target processor in reset” is just a follow-up error message. Open the **AREA.view** window to view all error messages.

- The target has no power or the debug cable is not connected to the target. This results in the error message “target power fail”.
- You did not select the correct core type **SYStem.CPU** <type>.
- There is an issue with the JTAG interface. See “**ARM JTAG Interface Specifications**” (app\_arm\_jtag.pdf) and the manuals or schematic of your target to check the physical and electrical interface. Maybe there is the need to set jumpers on the target to connect the correct signals to the JTAG connector.
- There is the need to enable (jumper) the debug features on the target. It will e.g. not work if nTRST signal is directly connected to ground on target side.
- The target is in an unrecoverable state. Re-power your target and try again.
- The target can not communicate with the debugger while in reset. Try **SYStem.Mode Attach** followed by “Break” instead of **SYStem.Up** or use **SYStem.Option EnReset OFF**.
- The default JTAG clock speed is too fast, especially if you emulate your core or if you use an FPGA based target. In this case try **SYStem.JtagClock 50kHz** and optimize the speed when you got it working.
- Your core needs adaptive clocking. Use the RTCK mode: **SYStem.JtagClock RTCK**.
- The core is used in a multicore system and the appropriate multicore settings for the debugger are missing. See for example **SYStem.CONFIG IRPRE**. This is the case if you get a value IR\_Width > 5 when you enter “DIAG 3400” and “AREA”. If you get IR\_Width = 4 (ARM7, ARM9, Cortex) or IR\_Width = 5 (ARM11), then you have just your core and you do not need to set these options. If the value can not be detected, then you might have a JTAG interface issue.
- The core has no clock.
- The core is kept in reset.
- There is a watchdog which needs to be deactivated.
- Your target needs special debugger settings. Check the directory ~\demo\arm\hardware if there is a suitable script file \*.cmm for your target.

<div>Debugging via VPN</div> <div>Ref: 0307</div>	<p><b>The debugger is accessed via Internet/VPN and the performance is very slow. What can be done to improve debug performance?</b></p> <p>The main cause for bad debug performance via Internet or VPN are low data throughput and high latency. The ways to improve performance by the debugger are limited:</p> <p>In PRACTICE scripts, use "SCREEN.OFF" at the beginning of the script and "SCREEN.ON" at the end. "SCREEN.OFF" will turn off screen updates. Please note that if your program stops (e.g. on error) without executing "SCREEN.OFF", some windows will not be updated.</p> <p>"SYStem.POLLING SLOW" will set a lower frequency for target state checks (e.g. power, reset, jtag state). It will take longer for the debugger to recognize that the core stopped on a breakpoint.</p> <p>"SETUP.URATE 1.s" will set the default update frequency of Data.List/Data.dump/Variable windows to 1 second (the slowest possible setting).</p> <p>prevent unneeded memory accesses using "MAP.UPDATEONCE [address-range]" for RAM and "MAP.CONST [address--range]" for ROM/FLASH. Address ranged with "MAP.UPDATEONCE" will read the specified address range only once after the core stopped at a breakpoint or manual break. "MAP.CONST" will read the specified address range only once per SYStem.Mode command (e.g. SYStem.Up).</p>
---	--



<p>Setting a Software Breakpoint fails</p> <p>Ref: 0276</p>	<p><b>What can be the reasons why setting a software breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to implement the wanted breakpoint.</p> <p>Possible reasons:</p> <p>The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller.</p> <p>Example: Read, write and access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</p> <p>TRACE32 can not change the memory.</p> <p>Example: ROM and Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All type of memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</p> <p>Contrary settings in TRACE32.</p> <p>Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</p> <p>RTOS and MMU:</p> <p>If the memory can be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</p>
<p>Data values onchip breakpoints</p> <p>Ref: 0369</p>	<p><b>Is it possible to set onchip breakpoints with data values?</b></p> <p>ARM7/9 support setting onchip breakpoints with data values. ARM11, CORTEX A/R does not support this capability. However, if the processor has an ETM logic, TRACE32 can provide this functionality by using two of the address and data comparators provided in the ETM. By setting the option ETM.StoppingBreakPoints, the resource management of TRACE32 is reconfigured so that two address/data comparators of the ETM can be used as standard read/write breakpoints. If the CPU does not support data values breakpoints and the ETM is not used, TRACE32 will stop the CPU when the data address is accessed, compare the data value with the condition and restart the CPU if the values are not equal.</p>
<p>Error Message Emulator Berr Error</p> <p>Ref: 0037</p>	<p><b>The message "emulator berr error" is displayed in some windows.</b></p> <p>This message indicates that the ARM has entered the ABORT mode as result of a system speed access from debug mode. The reason is that at least one memory access which was necessary to update the window was terminated with active ABORT (if AMBA: ERROR) signal.</p>

<p>Please highlight the options "-g -dual_debug -dwarf2" and /GHS</p> <p>Ref: 0432</p>	<p><b>How to generate and load debug information using a Greenhills Compiler?</b></p> <p>Depending on the version the following parameters must be passed to the compiler:</p> <p>-g -dual_debug -dwarf2</p> <p>Please note that not all options can be selected in the user interface and must be added manually in the compilers configuration file. Within TRACE32 it is recommended to load the files with option /GHS e.g.:</p> <p><b>Data.LOAD.Elf filename /GHS</b></p>
<p>Unstable Data</p> <p>Ref: 0038</p>	<p><b>Why do I have flickering data in some windows?</b></p> <p>Please make sure that the TURBO mode is off (SYStem.Option TURBO OFF). Another setting that may solve the problem is the reduction of the JTAG frequency (SYStem.JtagClock 5 MHz).</p>

Setting a Software Breakpoint fails	<p><b>What can be the reasons why setting a software-breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to realize the wanted breakpoint.</p> <p>Possible reasons:</p> <ul style="list-style-type: none"> <li>• The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller. Example: Read, Write and Access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</li> <li>• TRACE32 can not change the memory. Example: ROM. Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</li> <li>• Contrary settings in TRACE32. Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</li> <li>• RTOS and MMU: If the memory is able to be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</li> </ul>
Arm Dongle	<p><b>Modifications for ARM Debug Cable</b></p> <p><a href="http://www.lauterbach.com/faq/arm_dongle.pdf">http://www.lauterbach.com/faq/arm_dongle.pdf</a> Modifications ARM Dongle</p>

Debugging via  
VPN

Ref: 0307

**The debugger is accessed via Internet/VPN and the performance is very slow. What can be done to improve debug performance?**

The main cause for bad debug performance via Internet or VPN are low data throughput and high latency. The ways to improve performance by the debugger are limited:

In PRACTICE scripts, use "SCREEN.OFF" at the beginning of the script and "SCREEN.ON" at the end. "SCREEN.OFF" will turn off screen updates. Please note that if your program stops (e.g. on error) without executing "SCREEN.OFF", some windows will not be updated.

"SYStem.POLLING SLOW" will set a lower frequency for target state checks (e.g. power, reset, jtag state). It will take longer for the debugger to recognize that the core stopped on a breakpoint.

"SETUP.URATE 1.s" will set the default update frequency of Data.List/Data.dump/Variable windows to 1 second (the slowest possible setting).

prevent unneeded memory accesses using "MAP.UPDATEONCE [address-range]" for RAM and "MAP.CONST [address--range]" for ROM/FLASH. Address ranged with "MAP.UPDATEONCE" will read the specified address range only once after the core stopped at a breakpoint or manual break. "MAP.CONST" will read the specified address range only once per SYStem.Mode command (e.g. SYStem.Up).

## What can be the reasons why setting a software breakpoint fails?

Setting a software breakpoint can fail when the target HW is not able to implement the wanted breakpoint.

Possible reasons:

The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller.

Example: Read, write and access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data").

Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.

TRACE32 can not change the memory.

Example: ROM and Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All type of memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).

Contrary settings in TRACE32.

Like: MAP.BOnchip for this memory range. Break.SELect.<breakpoint-type> Onchip (HARD is only available for ICE and FIRE).

RTOS and MMU:

If the memory can be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.

Setting a Software Breakpoint fails	<p><b>What can be the reasons why setting a software-breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to realize the wanted breakpoint.</p> <p>Possible reasons:</p> <ul style="list-style-type: none"><li>• The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller. Example: Read, Write and Access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</li><li>• TRACE32 can not change the memory. Example: ROM. Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</li><li>• Contrary settings in TRACE32. Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</li><li>• RTOS and MMU: If the memory is able to be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</li></ul>
Arm Dongle	<p><b>Modifications for ARM Debug Cable</b></p> <p><a href="http://www.lauterbach.com/faq/arm_dongle.pdf">http://www.lauterbach.com/faq/arm_dongle.pdf</a> Modifications ARM Dongle</p>

Arm Dongle	<b>Modifications for ARM Dongle</b>
------------	-------------------------------------

Setting a Software Breakpoint fails	<p><b>What can be the reasons why setting a software-breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to realize the wanted breakpoint.</p> <p>Possible reasons:</p> <ul style="list-style-type: none"> <li>The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller. Example: Read, Write and Access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</li> <li>TRACE32 can not change the memory. Example: ROM. Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</li> <li>Contrary settings in TRACE32. Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</li> <li>RTOS and MMU: If the memory is able to be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</li> </ul>
Arm Dongle	<p><b>Modifications for ARM Debug Cable</b></p> <p><a href="http://www.lauterbach.com/faq/arm_dongle.pdf">http://www.lauterbach.com/faq/arm_dongle.pdf</a> Modifications ARM Dongle</p>

<p>Debugging via VPN</p> <p>Ref: 0307</p>	<p><b>The debugger is accessed via Internet/VPN and the performance is very slow. What can be done to improve debug performance?</b></p> <p>The main cause for bad debug performance via Internet or VPN are low data throughput and high latency. The ways to improve performance by the debugger are limited:</p> <p>In PRACTICE scripts, use "SCREEN.OFF" at the beginning of the script and "SCREEN.ON" at the end. "SCREEN.OFF" will turn off screen updates. Please note that if your program stops (e.g. on error) without executing "SCREEN.OFF", some windows will not be updated.</p> <p>"SYStem.POLLING SLOW" will set a lower frequency for target state checks (e.g. power, reset, jtag state). It will take longer for the debugger to recognize that the core stopped on a breakpoint.</p> <p>"SETUP.URATE 1.s" will set the default update frequency of Data.List/Data.dump/Variable windows to 1 second (the slowest possible setting).</p> <p>prevent unneeded memory accesses using "MAP.UPDATEONCE [address-range]" for RAM and "MAP.CONST [address--range]" for ROM/FLASH. Address ranged with "MAP.UPDATEONCE" will read the specified address range only once after the core stopped at a breakpoint or manual break. "MAP.CONST" will read the specified address range only once per SYStem.Mode command (e.g. SYStem.Up).</p>
---	--



## What can be the reasons why setting a software breakpoint fails?

Setting a software breakpoint can fail when the target HW is not able to implement the wanted breakpoint.

Possible reasons:

The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller.

Example: Read, write and access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data").

Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.

TRACE32 can not change the memory.

Example: ROM and Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All type of memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).

Contrary settings in TRACE32.

Like: MAP.BOnchip for this memory range. Break.SELect.<breakpoint-type> Onchip (HARD is only available for ICE and FIRE).

RTOS and MMU:

If the memory can be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.

Setting a Software Breakpoint fails	<p><b>What can be the reasons why setting a software-breakpoint fails?</b></p> <p>Setting a software breakpoint can fail when the target HW is not able to realize the wanted breakpoint.</p> <p>Possible reasons:</p> <ul style="list-style-type: none"> <li>• The wanted breakpoint needs special features that are only possible to realize by the trigger unit inside the controller. Example: Read, Write and Access (Read/Write) breakpoints ("type" in Break.Set window). Breakpoints with checking in real-time for data-values ("Data"). Breakpoints with special features ("action") like TriggerTrace, TraceEnable, TraceOn/TraceOFF.</li> <li>• TRACE32 can not change the memory. Example: ROM. Flash when no preparation with FLASH.Create, FLASH.TARGET and FLASH.AUTO was made. All memory if the memory device is missing the necessary control signals like WriteEnable or settings of registers and SpecialFunctionRegisters (SFR).</li> <li>• Contrary settings in TRACE32. Like: MAP.BOnchip for this memory range. Break.SELect.&lt;breakpoint-type&gt; Onchip (HARD is only available for ICE and FIRE).</li> <li>• RTOS and MMU: If the memory is able to be changed by Data.Set but the breakpoint doesn't work it might be a problem of using an MMU on target when setting the breakpoint to a symbolic address that is different than the writable and intended memory location.</li> </ul>
Arm Dongle	<p><b>Modifications for ARM Debug Cable</b></p> <p><a href="http://www.lauterbach.com/faq/arm_dongle.pdf">http://www.lauterbach.com/faq/arm_dongle.pdf</a> Modifications ARM Dongle</p>

There are two types of trace extensions available on the ARM:

- **ARM-ETM:** an Embedded Trace Macrocell or Program Trace Macrocell is integrated into the core. The Embedded Trace Macrocell provides program and data flow information plus trigger and filter features. The Program Trace Macrocell provide similar features but no data trace. The TRACE32 does not distinguish between ETM and PTM. The **ETM** command group is used for both.

Please refer to the online help books “**ARM-ETM Trace**” (trace\_arm\_etm.pdf) and “**ARM-ETM Programming Dialog**” (trace\_arm\_etm\_dialog.pdf) for detailed information about the usage of ARM ETM/PTM.

Please note that in case of CoreSight ETM/PTM you need to inform the debugger about the CoreSight trace system on the chip. If you can select the chip you are using (e.g. 'SYSstem.CPU OMAP4430') then this is automatically done. If you select a core (e.g. 'SYSstem.CPU CortexA9') then you need to configure the debugger in your start-up script by using commands like:

- **SYSstem.CONFIG.ETM.Base**
- **SYSstem.CONFIG.FUNNEL.Base**
- **SYSstem.CONFIG.TPIU.Base**
- **SYSstem.CONFIG.FUNNEL.ATBSource**
- **SYSstem.CONFIG.TPIU.ATBSource**

In case a HTM or ITM/STM module is available and shall be used you need also settings for that.

- **ARM7 Bus Trace:** the Preprocessor for ARM7 family samples the external address and data bus. The features for the Bus Trace are described in this book.

The commands for the ARM7 bus trace are:

- **SYSstem.Option AMBA**
- **SYSstem.Option NODATA**
- **TrOnchip.TEnable** and **TrOnchip.TCYcle**

# Symmetric Multiprocessing

---

A multi-core system used for **Asymmetric Multiprocessing (AMP)** has specialized cores which are used for specific tasks. To debug such a system you need to open separate TRACE32 graphical user interfaces (GUI) one for each core. On each GUI you debug the application which is assigned to this core and will never be executed on another core. The GUIs can be synchronized regarding program start and halt in order to debug the cores interaction.

ARM11 MPCore and Cortex-A9 MPCore are examples for multi-core architectures which allow **Symmetric Multiprocessing (SMP)**. The included cores of identical type are connected to a single shared main memory. Typically a proper SMP real-time operating system assigns the tasks to the cores. You will not know on which core the task you are interested in will be executed.

To debug an SMP system you start only one TRACE32 GUI.

The selection of the proper SMP chip (e.g. 'CNS3420' or 'OMAP4430') causes the debugger to connect to all included SMP-able cores on start-up (e.g. by 'SYStem.Up'). If you have an SMP-able core type selected (e.g. 'ARM11MPCore' or 'CortexA9MPCore') you need to specify the number of cores you intend to SMP-debug by **SYStem.CONFIG CoreNumber** *<number>*.

On a selected SMP chip (e.g. 'CNS3420' or 'OMAP4430') the CONFIG parameters of all cores are typically known by the debugger. For an SMP-able core type you need to set them yourself (e.g. DAPIRPRE, COREDEBUG.Base, ...). Where needed multiple parameters are possible (e.g. 'SYStem.CONFIG.COREDEBUG.Base 0x80001000 0x80003000').

System options and selected JTAG clock affect all cores.

All cores will be started, stepped and halted together. An exception is the assembler single-step which will affect only one core.

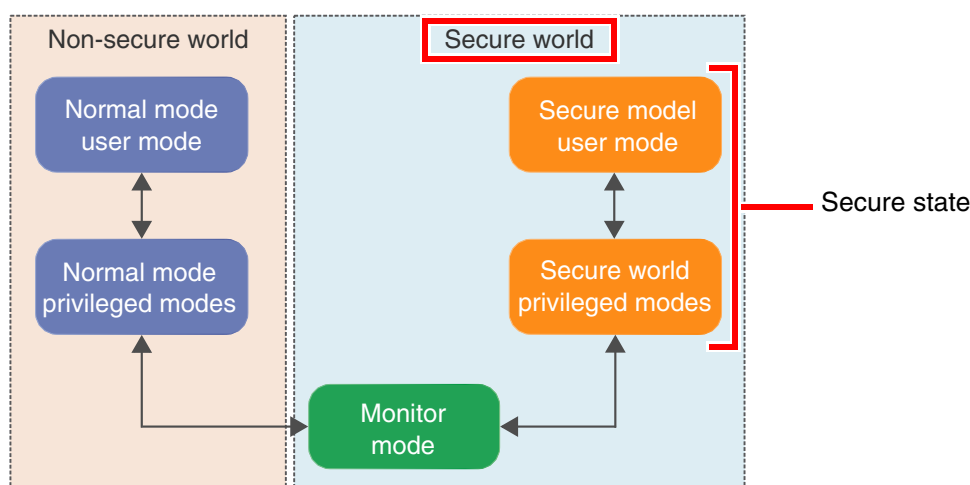
TRACE32 takes care that software and on-chip breakpoints will have effect on whatever core the task will run.

When the task halts, e.g. due to a breakpoint hit, the TRACE32 GUI shows the core on which the debug event has happened. The core number is shown in the state line at the bottom of the main window. You can switch the GUIs perspective to the other cores when you right-click on the core number there. Alternatively you can use the command **CORE.select** *<number>*.

## TrustZone Technology

The Cortex-A and ARM1176 processor integrate ARM's TrustZone technology, a hardware security extension, to facilitate the development of secure applications.

It splits the computing environment into two isolated worlds. Most of the code runs in the 'non-secure' world, whereas trusted code runs in the 'secure' world. There are core operations that allow you to switch between the secure and non-secure world. For switching purposes, TrustZone introduces a new secure 'monitor' mode. Reset enters the secure world:



Only when the core is in the secure world, core and debugger can access the secure memory. There are some CP15 registers accessible in secure state only, and there are banked CP15 registers, with both secure and non-secure versions.

## Debug Permission

Debugging is strictly controlled. It can be enabled or disabled by the SPIDEN (Secure Privileged Invasive Debug Enable) input signal and SUIDEN (Secure User Invasive Debug Enable) bit in SDER (Secure Debug Enable Register):

- SPIDEN=0, SUIDEN=0: debug in non-secure world, only
- SPIDEN=0, SUIDEN=1: debug in non-secure world and secure user mode
- SPIDEN=1: debug in non-secure and secure world

SPIDEN is a chip internal signal and its level can normally not be changed. The SUIDEN bit can be changed in secure privileged mode, only.

Debug mode can not be entered in a mode where debugging is not allowed. Breakpoints will not work there. A **Break** command or a **SYStem.Up** will work the moment a mode is entered where debugging is allowed.

## Checking Debug Permission

---

The DBGDSCR (Debug Status and Control Register) bit 16 shows the signal level of SPIDEN. In the SDER (Secure Debug Enable Register) you can see the SUIDEN flag assuming you are in the secure state which allows reading the SDER register.

## Checking Secure State

---

In the peripheral file, the DBGDSCR register bit 18 (NS) shows the current secure state. You can also see it in the [Register.view](#) window if you scroll down a bit. On the left side you will see 'sec' which means the core is in the secure state, 'nsec' means the core is in non-secure state. Both reflect the bit 0 (NS) of the SCR (Secure Control Register). However SCR is only accessible in secure state.

In monitor mode, which is also indicated in the [Register.view](#) window, the core is always in secure state independent of the NS bit (non-secure bit) described above. However, in monitor mode, you can access the secure CP15 register if NS=secure. And you can access the non-secure CP15 register if NS=non-secure.

## Changing the Secure State from within TRACE32

---

From the TRACE32 PowerView GUI, you can switch between secure mode (0) and non-secure mode (1) by toggling the 'sec', 'nsec' indicator in the [Register.view](#) window or by executing this command:

```
Register.Set NS 0 ;secure mode
Register.Set NS 1 ;non-secure mode
```

It sets or clears the NS (Non-Secure) bit in the SCR register. You will get a 'emulator function blocked by device security' message in case you are trying to switch to secure mode although debugging is not allowed in secure mode.

This way you can also inspect the register of the other world. Please note that a change in state affects program execution. Remember to set the bit back to its original value before continuing the application program.

## Accessing Memory

---

If you do not specify otherwise, the debugger shows you the memory of the secure state the core is currently in.

- The access class 'Z:' indicates secure mode ('Z' -> trustZone, 'S' -> Supervisor)
- The access class 'N:' indicates non-secure mode.

By preceding an address with the 'Z:' and 'N:' access class, you can force a certain memory view for all memory operations.

## Accessing Coprocessor CP15 Register

---

The peripheral file and 'C15:' access class will show you the CP15 register bank of the secure mode the core is currently in. When you try to access registers in non-secure world which are accessible in secure world only, the debugger will show you '????????'.

You can force to see the other bank by using access class "ZC15:" for secure, "NC15:" for non-secure respectively.

## Accessing Cache and TLB Contents

---

Reading cache and TLB (Translation Look-aside Buffer) contents is only possible if the debugger is allowed to debug in secure state. You get a 'function blocked by device security' message otherwise.

However, a lot of devices do not provide this debug feature at all. Then you get the message 'function not supported by this device'.

## Breakpoints and Vector Catch Register

---

Software breakpoints will be set in secure or non-secure memory depending on the current secure mode of the core. Alternatively, software breakpoints can be set by preceding an address with the access class "Z:" (secure) or "N:" (non-secure).

On-chip breakpoints will halt the core in any secure mode. Setting breakpoints for certain secure mode is not yet available.

Vector catch debug events ([TrOnchip.Set](#) ...) can individually be activated for secure state, non-secure state, and monitor mode.

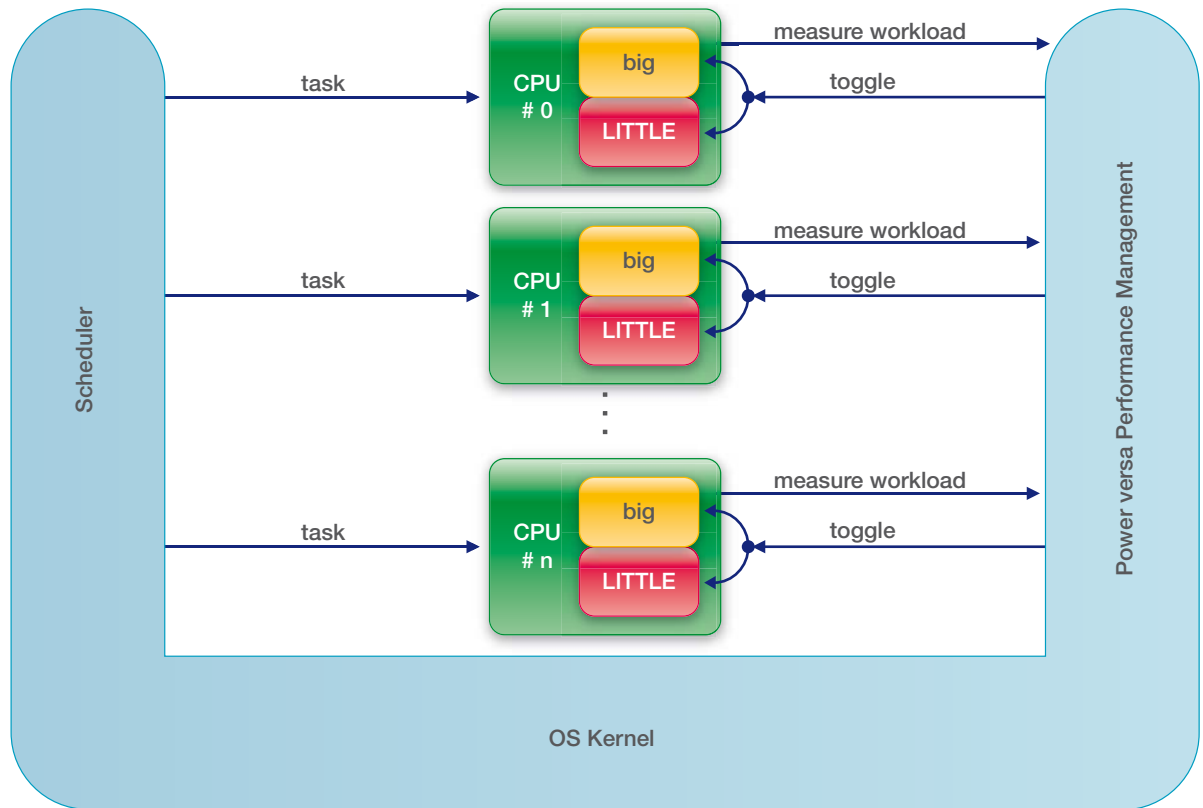
## Breakpoints and Secure Modes

---

The security concept of the ARMv8 architecture allows to specify breakpoints that cause a halt event only for a certain secure mode (secure/non-secure/hypervisor).

Please refer to the chapter about [secure, non-secure and hypervisor breakpoints](#) to get additional information.

ARM big.LITTLE processing is an energy savings method where high-performance cores get paired together in a cache-coherent combination. Software execution will dynamically be transitioned between these cores depending on performance needs.



The OS kernel scheduler sees each pair as a single virtual core. The big.LITTLE software works as an extension to the power-versa-performance management. It can switch the execution context between the big and the LITTLE core.

Qualified for pairing is Cortex-A15 (as 'big') and Cortex-A7 (as 'LITTLE').

## Debugger Setup

**Example** for a symmetric big.LITTLE configuration (2 Cortex-A15, 2 Cortex-A7):

```
SYStem.CPU CORTEXA15A7
SYStem.CONFIG CoreNumber 4.
CORE.ASSIGN BIGLITTLE 1. 2. 3. 4.
SYStem.CONFIG.COREDEBUG.Base <CA15_1> <CA7_2> <CA15_3> <CA7_4>
```



**Example** for a non-symmetric big.LITTLE configuration (1 Cortex-A15, 2 Cortex-A7):

```
SYStem.CPU CORTEXA15A7
SYStem.CONFIG CoreNumber 4.
CORE.ASSIGN BIGLITTLE 1. 2. NONE 4.
SYStem.CONFIG.COREDEBUG.Base <CA15_1> <CA7_2> <dummy_3> <CA7_4>
```

## Consequence for Debugging

---

The shown core numbers are extended by 'b' = 'big' or 'l' = 'LITTLE'.

The core status (active or powered down) can be checked with **CORE.SHOWACTIVE** or in the [state line](#) of the [TRACE32 main window](#), where you can switch between the cores.

The debugger assumes that one core of the pair is inactive.

The OS Awareness sees each pair as one virtual core.

The peripheral file respects the core type (Cortex-A15 or Cortex-A7).

## Requirements for the Target Software

---

The routine (OS on target) which switches between the cores needs to take care of (copying) transferring the on-chip debug settings to the core which wakes up.

This needs also to be done when waking up a core pair. In this case you copy the settings from an already active core.

## big.LITTLE MP

---

Another logical use-model is ('MP' = Multi-Processing). It allows both the big and the LITTLE core to be powered on and to simultaneously execute code.

From the debuggers point of view, this is not a big.LITTLE system in the narrow sense. There are no pairs of cores. It is handled like a normal multicore system but with mixed core types.

Therefore for the setup, we need **SYStem.CPU CORTEXA15A7**, but we use **CORE.ASSIGN** instead of **CORE.ASSIGN BIGLITTLE**.

**Example** for a symmetric big.LITTLE MP configuration (2 Cortex-A15, 2 Cortex-A7):

```
SYStem.CPU CORTEXA15A7
SYStem.CONFIG CoreNumber 4.
CORE.ASSIGN 1. 2. 3. 4.
SYStem.CONFIG.COREDEBUG.Base <CA15_1> <CA7_2> <CA15_3> <CA7_4>
```

# Breakpoints

---

## Software Breakpoints

---

If a software breakpoint is used, the original code at the breakpoint location is patched by a breakpoint code.

While software breakpoints are used one of the two ICE breaker units is programmed with the breakpoint code (on ARM7 and ARM9, except ARM9E variants). This means whenever a software breakpoint is set only one ICE unit breakpoint is remaining for other purposes. 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. For the ARM architecture the on-chip breakpoints are provided by the “ICEbreaker” unit. on-chip breakpoints are usually needed for instructions in FLASH/ROM.

With the command **MAP.BOnchip** *<range>* it is possible to tell the debugger where you have ROM / FLASH on the target. If a breakpoint is set into a location mapped as BOnchip one ICEbreaker unit is automatically programmed.

## 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 ARM notation these breakpoints are called watchpoints. A watchband may use one or two ICEbreaker units.

The number of on-chip breakpoints for data accesses can be extended by using the ETM Address and Data comparators. Refer to **ETM.StoppingBreakPoints**.

- **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 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

	On-chip Breakpoints	Instruction Breakpoints	Read/Write Breakpoints	Data Breakpoint
<b>ARM7 Janus</b>	2 (Reduced to 1 if software breakpoints are used)	2/1 Breakpoint ranges as bit masks	2/1 Breakpoint ranges as bit masks	2
<b>ARM9</b>	2 (Reduced to 1 if software breakpoints are used, except ARM9E)	2/1 Breakpoint ranges as bit masks	2/1 Breakpoint ranges as bit masks	2
<b>ARM10</b>	2-16 Instruction 2-16 Read/Write	2-16 single address	2-16 single address	—
<b>ARM11</b>	2-16 Instruction 2-16 Read/Write	2-16 single address	2-16 single address	—
<b>Cortex-A5</b>	3 instruction 2 read/write	3 single address	2 range as bit mask, break before make	—
<b>Cortex-A8</b>	6 instruction 2 read/write	6 range as bit mask	2 range as bit mask, break before make	—
<b>Cortex-A7/A9/A15</b>	6 instruction 4 read/write	6 single address	4 range as bit mask, break before make	—

## Hardware Breakpoints (Bus Trace only)

When a Preprocessor for ARM7 family is used, hardware breakpoints are available to filter the trace information. Refer to [TrOnchip.TEnable](#) for more information.

If a hardware breakpoint is used the resources to set the breakpoint are provided by the TRACE32 development tool.

## Example for Standard Breakpoints

Assume you have a target with

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

The command to configure TRACE32 correctly for this configuration is:

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

**The following standard breakpoint combinations are possible.**

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

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

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

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

3. Two breakpoints in ROM/FLASH

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

4. Two breakpoints on a read or write access

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

5. One breakpoint in ROM/FLASH and one breakpoint on a read or write access

```
Break.Set 0x100 /Program ; On-chip breakpoint 1  
Break.Set 0x108010 /Read ; On-chip breakpoint 2
```

TRACE32 will set any breakpoint to work in any secure and non-secure mode. As of build 59483, TRACE32 distinguishes between secure, non-secure, and hypervisor breakpoints. The support for these kinds of breakpoints is disabled per default, i.e. all breakpoints are set for all secure/non-secure modes.

### Enable and Use Secure, Non-Secure and Hypervisor Breakpoints

To make use of this feature, you have to enable the symbol management for ARM zones first with the **SYStem.Option ZoneSPACES** command:

```
SYStem.Option ZoneSPACES ON           ; Enable symbol management
                                       ; for ARM zones
```

Usually TRACE32 will then set the secure/non-secure breakpoint automatically if it has enough information about the secure/non-secure properties of the loaded application and its symbols. This means the user has to tell TRACE32 if a program code runs in secure/non-secure or hypervisor mode when the code is loaded:

```
Data.LOAD.ELF armf Z:      ; Load application, symbols for secure mode
Data.LOAD.ELF armf N:      ; Load application, symbols for non-secure mode
Data.LOAD.ELF armf H:      ; Load application, symbols for hypervisor mode
```

Please refer to the **SYStem.Option ZoneSPACES** command for additional code loading examples.

Now breakpoints can be used as usual, i.e. TRACE32 will automatically take care of the secure type when a breakpoint is set. This depends on how the application/symbols were loaded:

```
Break.Set main              ; Set breakpoint on main() function, Z:, N: or
                             ; H: access class is automatically set
Var.Break.Set struct1      ; Set Read/Write breakpoints to the whole
                             ; structure struct1. The breakpoint is either
                             ; a secure/non-secure or hypervisor type.
```

## Example 1 - Load Secure Application and Set Breakpoints

```
SYStem.Option ZoneSPACES ON      ; Enable symbol management

// Load demo application and tell TRACE32 that it is secure
Data.LOAD.ELF ~/demo/arm/compiler/arm/armle.axf Z:

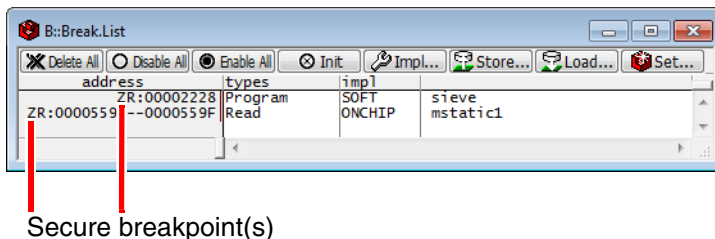
// Set a breakpoint on the sieve() function start
Break.Set sieve

// Set a read breakpoint to the global variable mstatic1
Var.Break.Set mstatic1 /Read

Break.List                        ; Show breakpoints
```

First the symbol management is enabled. An application is loaded and TRACE32 is advised by the access class “Z:” at the end of the **Data.LOAD.ELF** command that this application runs in secure mode.

As a next step, two breakpoints are set but the user does not need to care about any access classes. The **Break.List** window shows that the breakpoints are automatically configured to be of the secure type. This is shown by the “Z:” access class that is set at the beginning of the breakpoint addresses:



## Set Breakpoints and Enforce Secure Mode

TRACE32 allows the user to specify whether a breakpoint should be set for secure, non-secure or hypervisor mode. This means the user has to specify an access class when the breakpoint is set:

```
Break.Set Z:main      ; Enforce secure breakpoint on main()

Break.Set N:main      ; Enforce non-secure breakpoint on main()

Break.Set H:main      ; Enforce hypervisor breakpoint on main()
```

Breakpoints on variables need the variable name and the access class to be enclosed in round brackets:

```
Var.Break.Set (Z:struct1) ; Enforce secure read/write breakpoint

Var.Break.Set (N:struct1) ; Enforce non-secure read/write breakpoint

Var.Break.Set (H:struct1) ; Enforce hypervisor read/write breakpoint
```

```
SYStem.Option ZoneSPACES ON      ; Enable symbol management

// Load demo application and tell TRACE32 that it is secure
Data.LOAD.ELF ~/demo/arm/compiler/arm/armle.axf Z:

// Set secure breakpoint (auto-configured) on function main()
Break.Set main

// Explicitly set hypervisor breakpoint on function sieve()
Break.Set H:sieve

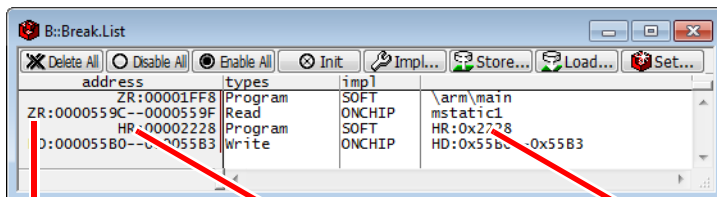
// Set secure read breakpoint (auto-configured) on variable mstatic1
Var.Break.Set mstatic1 /Read

// Explicitly set hypervisor write breakpoint on variable vtdef1
Var.Break.Set (H:vtdef1) /Write

Break.List                        ; Show breakpoints
```

First, the symbol management is enabled. An application is loaded and TRACE32 is advised by the “Z:” at the end of the **Data.LOAD.ELF** command that this application runs in secure mode.

As a next step, four breakpoints are set. Two of them do not have any access class specified, so TRACE32 will use the symbol information to make it a secure breakpoint. The other two breakpoints are defined as hypervisor breakpoints using the “H.” access class. In this case the symbol information is explicitly overwritten. The **Break.List** now shows a mixed breakpoint setup:



Secure breakpoint

Hypervisor breakpoint

No symbol information

### NOTE:

If a breakpoint is explicitly set in another mode, there might be no symbol information loaded for this mode. This means that the **Break.List** can only display the address of the breakpoint but not the corresponding symbol.





## Complex Breakpoints

---

To use the advanced features of the ICE breaker unit the **TrOnchip** command group is possible. These commands provide full access to both ICE breaker units called A and B in the TRACE32 system. For an example of complex breakpoint usage please refer to the chapter **TrOnchip Example**. Most features can also be used by setting advanced breakpoints (e.g. task selective breakpoints, exclude breakpoints). Ranged breakpoints use multiple breakpoint resources to better fit the range when the resources are available.

## Direct ICE Breaker Access

---

It is possible to program the complete ICE breaker unit directly, by using the access class ICE. E.g. the command `Data.Set ICE:10 %Long 12345678` writes the value 12345678 to the Watchpoint 1 Address Value Register. The following table lists the addresses of the relevant registers.

Address	Register
ICE:8	Watchpoint 0 Address Value
ICE:9	Watchpoint 0 Address Mask
ICE:0A	Watchpoint 0 Data Value
ICE:0B	Watchpoint 0 Data Mask
ICE:0C	Watchpoint 0 Control Value
ICE:0D	Watchpoint 0 Control Mask
ICE:10	Watchpoint 1 Address Value
ICE:11	Watchpoint 1 Address Mask
ICE:12	Watchpoint 1 Data Value
ICE:13	Watchpoint 1 Data Mask
ICE:14	Watchpoint 1 Control Value
ICE:15	Watchpoint 1 Control Mask

For more details please refer to the ARM data sheet. It is recommended to use the **Break.Set** or **TrOnchip** commands instead of direct programming, because then no special ICEbreaker knowledge is required.

## Example for ETM Stopping Breakpoints

The default on-chip breakpoints either allow you to just set an instruction breakpoint on a single address or to apply a mask to get a rough range. In case of a mask, the given range is extended to the next range limits that fit the mask, i.e. the breakpoint may cover a wider address range than initially anticipated.

ETM stopping breakpoints allow you to set a true address range for instructions, i.e. the end and the start address of the breakpoint really match your expectations. This only works if the CPU provides an ETM with the necessary resources, e.g. the address comparators.

### Prerequisites for ETM stopping breakpoints:

- Make sure that an ETM base address is configured. Otherwise TRACE32 will assume that there is no ETM.

```
SYStem.CONFIG ETM Base DAP:<etm_base>          ; Make ETM available
```

- If your CPU has its own CTI, it is recommended that you specify the CTI as well. Dependant on the specific core implementation, the CTI might be needed to receive the ETM stop events:

```
SYStem.CONFIG CTI Base DAP:<cti_base>
```

It's recommended to add both configuration commands to your PRACTICE start-up script (\*.cmm).

### To set ETM stopping breakpoints:

1. Activate the ETM Stopping breakpoints support:

```
ETM.StoppingBreakpoints ON
```

2. Set the instruction range breakpoints, e.g.:

```
Break.Set func10          ; Set address range breakpoint on  
                           ; the address range of function  
                           ; func10  
  
Break.Set 0xEC009008++0x58 ; Set address range breakpoint with  
                           ; precise start and end address
```

The [Break.List](#) window provides an overview of all set breakpoints.

For more information, see [ETM.StoppingBreakPoints](#) in “[ARM-ETM Trace](#)” (trace\_arm\_etm.pdf).

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 about the term [access class](#), see “[TRACE32 Glossary](#)” ([glossary.pdf](#)).

**In this section:**

- [Description of the Individual Access Classes](#)
- [Combinations of Access Classes](#)
- [How to Create Valid Access Class Combinations](#)
- [Access Class Expansion by TRACE32](#)

## Description of the Individual Access Classes

---

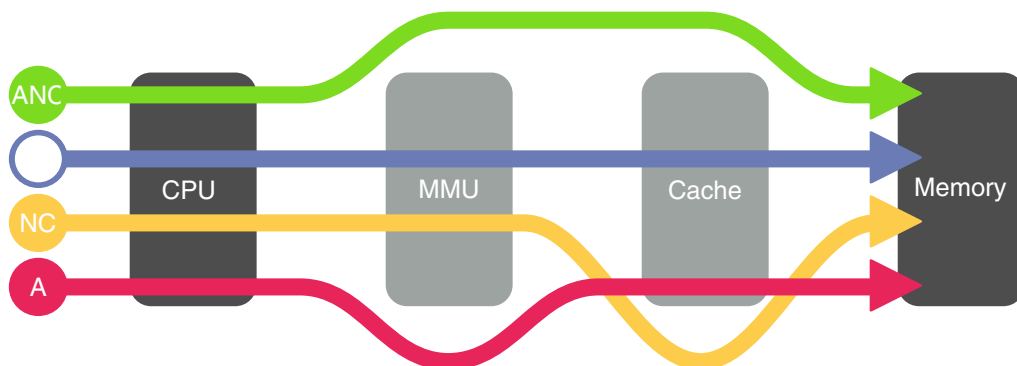
Access Class	Description
A	Absolute addressing (physical address)
AHB, AHB2	See DAP.
APB, APB2	See DAP.
AXI, AXI2	See DAP.
C14	Access to C14-Coprocessor register. Its recommended to only use this in AArch32 mode.
C15	Access to C15-Coprocessor register. Its recommended to only use this in AArch32 mode.
D	Data Memory

Access Class	Description
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). The DAP is a CoreSight component mandatory on Cortex based devices.</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 (AHB MEM-AP in case of a Cortex-M).</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.CpuAccess</a> and <a href="#">SYStem.MemAccess</a> )
M ARMv8-A only	EL3 Mode (TrustZone devices). This access class only refers to the 64-bit EL3 mode. It does not refer to the 32-bit monitor mode. If an ARMv8 based device is in 32-bit only mode, any entered "M" access class will be converted to a "ZS" access class.
H	EL2/Hypervisor Mode (devices having Virtualization Extension)
I	Intermediate physical address. Available on devices having Virtualization Extension.
J	Java Code (8-bit)
N	EL0/1 Non-Secure Mode (TrustZone devices)
P	Program Memory
R	AArch32 ARM Code (A32, 32-bit instr. length)
S	Supervisor Memory (privileged access)
SPR ARMv8-A only	Access to System Register, Special Purpose Registers and System Instructions. Its recommended to only use this in AArch64 mode.
T	AArch32 Thumb Code (T32, 16-bit instr. length)
U	User Memory (non-privileged access) not yet implemented; privileged access will be performed.
USR	Access to Special Memory via User-Defined Access Routines

Access Class	Description
VM	Virtual Memory (memory on the debug system)
X ARMv8-A only	AArch64 ARM64 Code (A64, 32-bit instr. length)
Z	Secure Mode (TrustZone devices)

## Combinations of Access Classes

Combinations of access classes are possible as shown in the example illustration below:



The access class “A” in the red path means “physical access”, i.e. it will only bypass the MMU but consider the cache content. The access class “NC” in the yellow path means “no cache”, so it will bypass the cache but not the MMU, i.e. a virtual access is happening.

If both access classes “A” and “NC” are combined to “ANC”, this means that the properties of both access classes are summed up, i.e. both the MMU and the cache will be bypassed on a memory access.

The blue path is an example of a virtual access which is done when no access class is specified.

The access classes “A” and “NC” are not the only two access classes that can be combined. An access class combination can consist of up to five access class specifiers. But any of the five specifiers can also be omitted.

**Three specifiers:** Let’s assume you want to view a secure memory region that contains 32-bit ARM code. Furthermore, the access is translated by the MMU, so you have to pick the correct CPU mode to avoid a translation fail. In our example it should be necessary to access the memory in ARM supervisor mode. To ensure a secure access, use the access class specifier “Z”. To switch the CPU to supervisor mode during the access, use the access class specifier “S”. And to make the debugger disassemble the memory content as 32-bit ARM code use “R”. When you put all three access class specifiers together, you will obtain the access class combination “ZSR”.

```
List.Mix ZSR:0x10000000 // View 32-bit ARM code in secure memory
```

**One specifier:** Let's imagine a physical access should be done. To accomplish that, start with the "A" access class specifier right away and omit all other possible specifiers.

```
Data.dump A:0x80000000 // Physical memory dump at address 0x80000000
```

**No specifiers:** Let's now consider what happens when you omit all five access class specifiers. In this case the memory access by the debugger will be a virtual access using the *current CPU context*, i.e. the debugger has the same view on memory as the CPU.

```
Data.dump 0xFB080000 // Virtual memory dump at address 0xFB080000
```

Using no or just a single access class specifier is easy. Combining at least two access class specifiers is slightly more challenging because access class specifiers cannot be combined in an arbitrary order. Instead you have to take the syntax of the access class specifiers into account.

If we refer to the above example "ZSR" again, it would not be possible to specify the access class combination as "SZR" or "RZS", etc. Instead you have to follow certain rules to make sure the syntax of the access class specifiers is correct. This will be illustrated in the next section.

## How to Create Valid Access Class Combinations

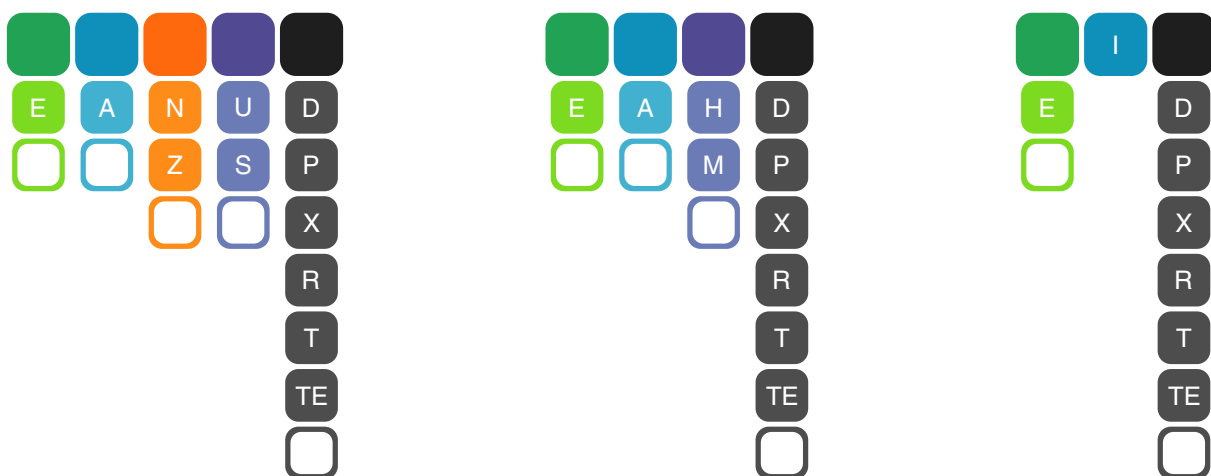
The illustrations below will show you how to combine access class specifiers for frequently-used access class combinations.

### Rules to create a valid access class combination:

- From each column of an illustration, select only one access class specifier.
- You may skip any column - but only if the column in question contains an empty square.
- Do not change the original column order. Recommendation: Put together a valid combination by starting with the left-most column, proceeding to the right.

### Memory Access through CPU (CPU View)

The debugger uses the CPU to access memory and peripherals like UART or DMA controllers. This means the CPU will carry out the accesses requested by debugger. Examples would be virtual, physical, secure, or non-secure memory accesses.



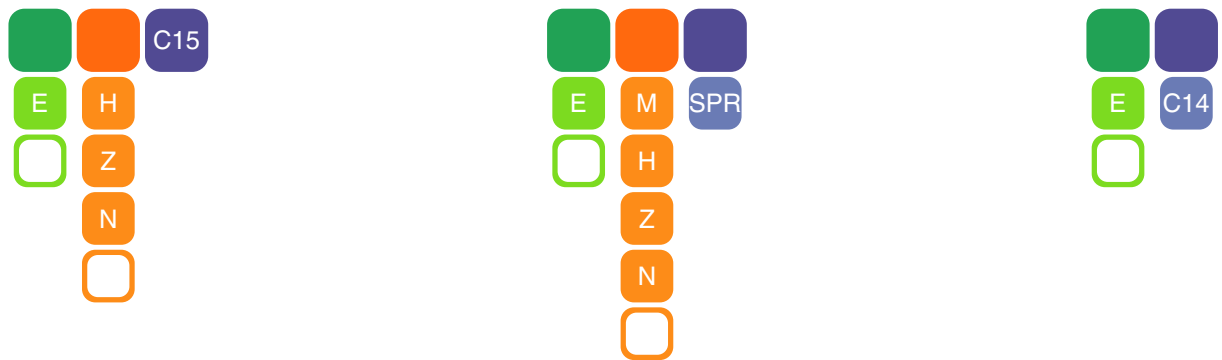
### Example combinations:

<b>AD</b>	View physical data (current CPU mode)
<b>AH</b>	View physical data or program code while CPU is in hypervisor mode
<b>ED</b>	Access data at run-time
<b>NUX</b>	View A64 instruction code at non-secure virtual address location, e.g. code of the user application.
<b>ZSD</b>	View data in secure supervisor mode at virtual address location



## Peripheral Register Access

This is used to access core ID and configuration/control registers.

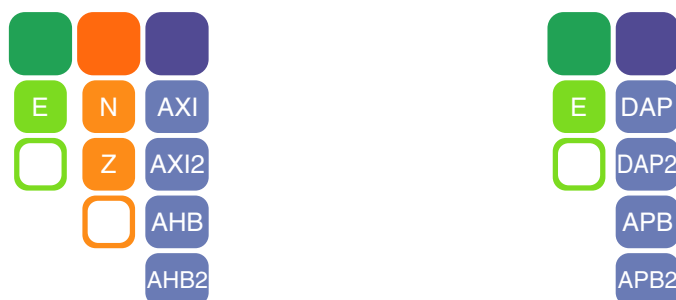


### Example combinations:

<b>NC15</b>	Access non-secure banked coprocessor 15 register (AArch32 mode)
<b>C15</b>	Access coprocessor 15 register in current secure mode (AArch32 mode)
<b>SPR</b>	Access system register (AArch64 mode)
<b>MSPR</b>	Access system registers in EL3 (AArch64) mode
<b>HSPR</b>	Access system registers in EL2 (AArch64) mode
<b>ZSPR</b>	Access system registers in secure EL1 (AArch64) mode

## CoreSight Access

These accesses are typically used to access the CoreSight busses APB, AHB and AXI directly through the DAP bypassing the CPU. For example, this could be used to view physical memory at run-time.



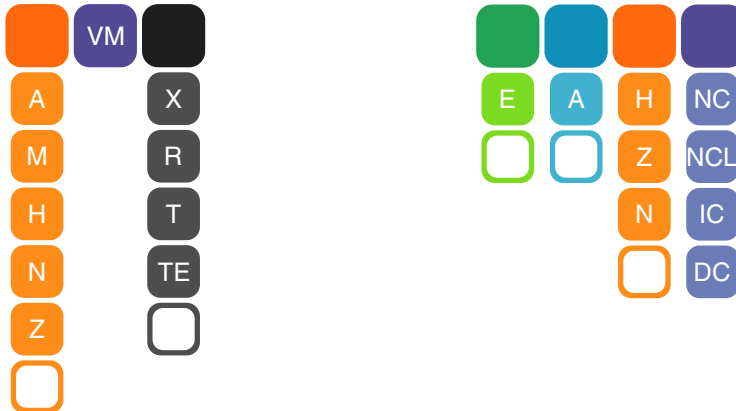
### Example combinations:

<b>EAXI</b>	Access memory location via AXI during run-time
<b>EZAXI</b>	Access secure memory location via AXI during run-time
<b>DAP</b>	Access debug access port (e.g. core debug registers)

### Cache and Virtual Memory Access

---

These accesses are used to either access the [TRACE32 virtual memory \(VM:\)](#) or to access data and instruction caches directly or to bypass them.



### Example combinations:

<b>VM</b>	Access virtual memory using current CPU context
<b>AVM</b>	Access virtual memory ignoring current CPU context
<b>HVMR</b>	Access virtual memory that is banked in hypervisor mode and disassemble memory content as 32-bit ARM instruction code
<b>NC</b>	Bypass all cache levels during memory access
<b>ANC</b>	Bypass MMU and all cache levels during memory access

Access Class Expansion by TRACE32

If you omit access class specifiers in an access class combination, then TRACE32 will make an educated guess to fill in the blanks. The access class is expanded based on:

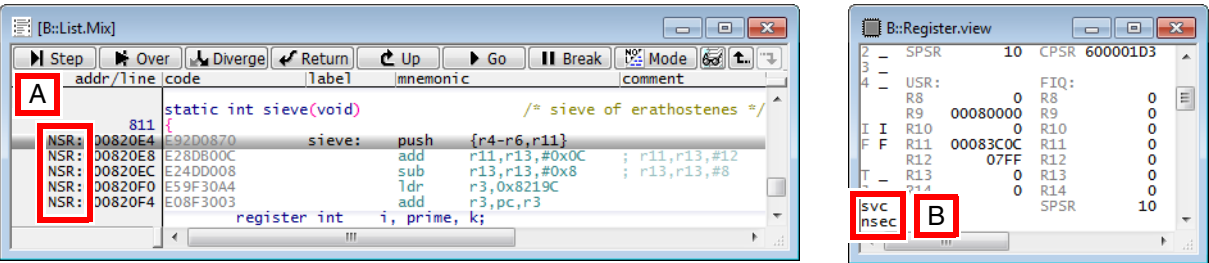
- The current CPU context (architecture specific)
- The used window type (e.g. **Data.dump** window for data or **List.Mix** window for code)
- Symbol information of the loaded application (e.g. combination of code and data)
- Segments that use different instruction sets
- Debugger specific settings (e.g. **SYStem.Option.\***)

Examples: Memory Access through CPU

Let’s assume the CPU is in non-secure supervisor mode, executing 32-bit code.

User input at the command line	Expansion by TRACE32	These access classes are added because...
List.Mix  (see also illustration below)	NSR:	N: ... the CPU is in non-secure mode. S: ... the CPU is in supervisor mode. R: ... code is viewed (not data) and the CPU uses 32-bit instructions.
Data.dump A:0x0	ANSD:0x0	N: ... the CPU is in non-secure mode. S: ... the CPU is in supervisor mode. D: ... data is viewed (not code).
Data.dump Z:0x0	ZSD:0x0	S: ... the CPU is in supervisor mode. D: ... data is viewed (not code).
<b>NOTE:</b> ‘E’ and ‘A’ are not automatically added because the debugger cannot know if you intended a run-time or physical access.		

Your input, here `List.Mix` at the TRACE32 command line, remains unmodified. TRACE32 performs an access class expansion and visualizes the result in the window you open, here in the **List.Mix** window.



- A TRACE32 makes an educated guess to expand your *omitted* access class to “NSR”.
- B Indicates that the CPU is in non-secure supervisor mode.

The following coprocessors can be accessed if available in the processor:

Coprocessor 14. Please refer to the chapter [Virtual Terminal](#) and to your ARM documentation for details. On Cortex-A and Cortex-R the debug register can be accessed by 'C14' access class and the address is the address offset in the debug register block divided by 4. Recommended is to use the 'DAP:' or 'EDAP:' access class, but then the address is the address offset plus the base address of the debug register block which is 0xd4011000.

Coprocessor 15, which allows the control of basic CPU functions. This coprocessor can be accessed with the access class C15. For the detailed definition of the CP15 registers, please refer to the ARM data sheet. The CP15 registers can also be controlled in the [PER](#) window.

The TRACE32 address is composed of the CRn, CRm, op1, op2 fields of the corresponding coprocessor register command

```
<MCR|MRC> p15, <op1>, Rd, CRn, CRm, <op2>
```

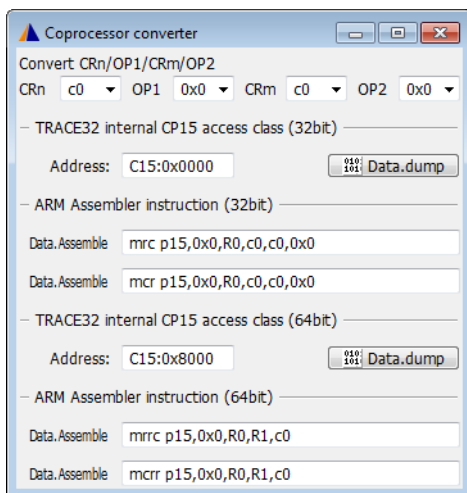
BIT0-3:CRn, BIT4-7:CRm, BIT8-10:<op2>, BIT12-14:<op1>, Bit16=0 (32-bit access)

```
<MCRR|MRRC> p15, <op1>, <Rd1>, <Rd2>, <CRm>
```

BIT0-3: -, BIT4-7:CRm, BIT8-10: -, BIT12-14:<op1>, Bit16=1 (64-bit access)

is the corresponding TRACE32 address (one nibble for each field). There is also a script-based dialog, which assists in calculating the C15 address class offsets. Simply run the following command to display this dialog:

```
DO ~/demo/arm/etc/coprocessor/coprocessor_converter.cmm
```



On Cortex-A/R or ARM11 you can access other available coprocessors by using the same addressing scheme. The access class is then e.g. 'C10:' instead of 'C15'. You need to secure that access to this coprocessor is permitted in the Coprocessor Access Control Register.

The "C15:" access class provides the view of the mode the core currently is in. On devices having "TrustZone" (ARM1176, Cortex-A) there are some banked CP15 register, one for secure and one for non-secure mode. With "ZC15:" and "NC15:" you can access the secure / non-secure bank independent of the current core mode. On devices having a "Hypervisor" mode (e.g. Cortex-A7, -A15) there are CP15 register which are only available in hypervisor mode or in monitor mode with NS bit set. With "HC15:" you can access these register independent of the current core mode.

# Accessing Memory at Run-time

This section describes how memory can be accessed at run-time. It gives an overview of all available methods for Arm based devices.

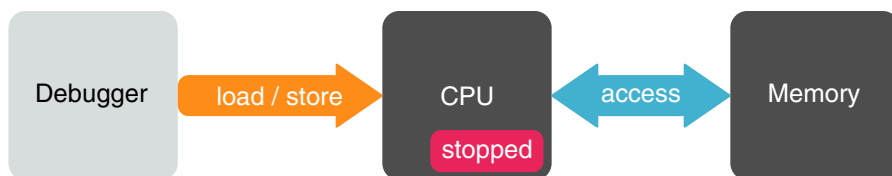
## In this section:

- [Intrusive and Non-intrusive Run-time Access](#)
- [Cache Coherent Non-intrusive Run-time Access](#)
- [Performing Intrusive and Non-intrusive Run-time Accesses with TRACE32](#)
- [Performing Cache Coherent Non-intrusive Run-time Accesses with TRACE32](#)
- [Additional Considerations](#)

## Intrusive and Non-intrusive Run-time Access

### Intrusive run-time access

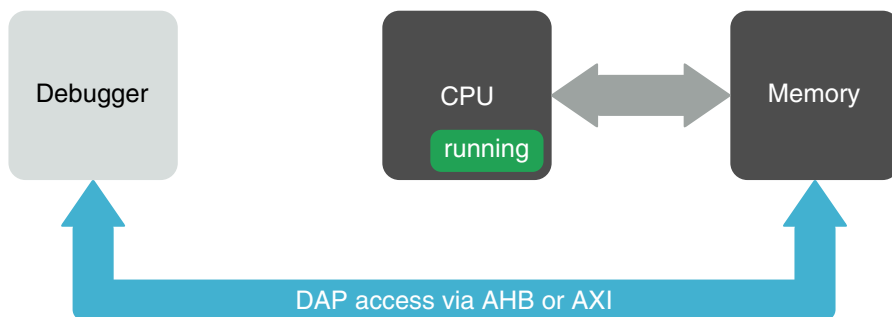
Intrusive means that the CPU is periodically stopped and restarted, so that the debugger can access the memory content through the CPU using load / store commands.



The debugger will see memory the same way the CPU does; however, real-time constraints may be broken.

### Non-intrusive run-time access

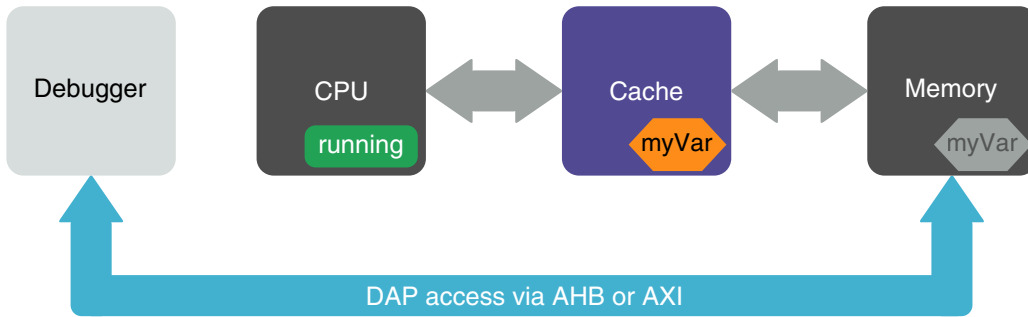
Non-intrusive means that the CPU is not stopped during the memory access.



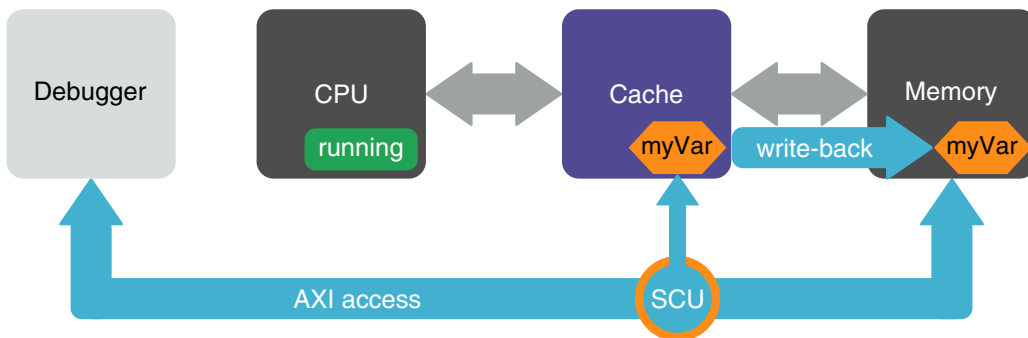
The debugger cannot read through the CPU while it is running and continuously accessing memory. Therefore the debugger has to use a DAP access, i.e. the AHB or AXI bus. The CPU is bypassed, which will equal a physical memory access. This way the real-time constraints are preserved. This access method only works if an AHB or AXI is present and if the busses are properly mapped to memory.

## Cache Coherent Non-intrusive Run-time Access

A non-intrusive run-time access through the AHB/AXI bus will bypass caches. In the example below, “myVar” is only updated in the cache but not in memory. Hence its current state is invisible to the debugger.



An example of such a cache would be a write-back cache. For the debugger to see the current value of “myVar”, a run-time access has to trigger a cache flush, so that “myVar” is written back to memory.



In this example, the cache coherency is maintained by the Snoop Control Unit (SCU). During an AXI access, the SCU can be instructed to trigger a write of “myVar” back to memory. This feature is not supported for the AHB. It is implementation-defined whether this is available for AXI transactions.

## Performing Intrusive and Non-intrusive Run-time Accesses with TRACE32

All of the previously mentioned access methods can be carried out in TRACE32.

To access memory at run-time, add the access class “E” as a prefix. “E” means run-time access and can be combined with most access classes that access memory. E.g. “Data.dump NSD:<addr>” can be extended to “Data.dump ENSD:<addr>”.

### Intrusive run-time access

To activate intrusive memory accesses, use the command **SYStem.CpuAccess Enable**.

```
SYStem.CpuAccess Enable      ; Intrusive run-time memory access, CPU
                             ; is periodically stopped / restarted
Data.dump E:0x100            ; Intrusive access via CPU. Prefix "E"
Var.view %E myVar            ; is required to read 0x100 or myVar
```

## Non-intrusive run-time access: Direct DAP access

You can directly specify an access to memory via the AHB or AXI bus using an access class. This requires that the AHB or AXI is defined as a valid access port. If you select a known chip with **SYSystem.CPU**, then TRACE32 configures this setting automatically. Please see the following example for the AXI:

```
SYSystem.CONFIG MEMORYACCESSPORT 1. ; Define memory access port and AXI
SYSystem.CONFIG AXIACCESSPORT 1.    ; access port (e.g. port number 1)

Data.dump EAXI:<addr>                ; Run-time access via AXI. Prefix "E"
Data.dump EAXI:myVar                 ; is required to read <addr> or myVar
```

## Non-intrusive run-time access: Indirect DAP access

It is not very convenient or even not always possible to use an AXI or AHB access class specifier. In most cases you should let the debugger decide which access to use. Use the command **SYSystem.MemAccess DAP** to activate non-intrusive run-time accesses via AHB or AXI. TRACE32 will then redirect access to the AHB or AXI bus. This requires that the AHB or AXI is defined as a valid access port.

```
SYSystem.CONFIG MEMORYACCESSPORT 1. ; Define memory access port and AHB
// SYSystem.CONFIG AHBACCESSPORT 1. ; or AXI access port
SYSystem.CONFIG AXIACCESSPORT 1.

SYSystem.MemAccess DAP                ; Non-intrusive access via AHB / AXI

Data.dump E:0x100                    ; Run-time access via DAP. Prefix "E"
Var.view %E myVar                    ; is required to read 0x100 or myVar
```

## Performing Cache Coherent Run-time Accesses with TRACE32

So far there is not guarantee that the run-time accesses via AHB / AXI will be coherent. This means, you might not see the current value of e.g. a variable because the value is in the cache but not updated in memory.

The AXI may allow you to select whether an access should be performed as a coherent transaction or not. To activate this feature, use **SYSystem.Option AXIACEEnable ON**

```
SYSystem.CONFIG MEMORYACCESSPORT 1. ; Define memory access port and AXI
SYSystem.CONFIG AXIACCESSPORT 1.    ; access port (e.g. port number 1)

SYSystem.Option AXIACEEnable ON      ; Enable cache coherent transactions
SYSystem.MemAccess DAP               ; Non-intrusive access via AXI

Data.dump E:0x100                    ; Run-time access via AXI. Prefix "E"
Var.view %E myVar                    ; is required to read 0x100 or myVar
```



**NOTE:**

- Support for cache coherent AXI transactions is implementation-defined. Therefore **SYStem.Option AXIACEEnable ON** may be without effect.
- The AHB does not provide such a coherency mechanism.

## Coherent cache accesses without AXI coherency support

The AXI may not provide cache coherent transactions or there may only be an AHB available. In this case you can still perform non-intrusive cache-coherent run-time memory accesses. But this requires that you change the configuration of your target application in one of the following ways:

- Configure the address range of interest as “non-cacheable”
- Configure the address range of interest as “write-through”
- Configure the entire cache as “write-through” (global setting)
- Make the CPU periodically flush the cache lines of interest
- Disable the cache
- Use a monitor program that accesses the memory address range of interest through the cache (CPU view) and provides the result to the debugger, e.g. via shared memory or DCC. This requires a code instrumentation of the target application.

## Additional Considerations

### Non-intrusive run-time access with active MMU

If the run-time access involves virtual addresses that do not directly map to physical addresses, the debugger has to be made aware of the proper virtual-to-physical address translations. For more information about address translations, refer to the descriptions of the following commands:

**TRANSlation.Create**

If the CPU has never stopped, set the translation manually.

**MMU.Scan**

Scan static page tables into the debugger while the CPU is stopped.

**TRANSlation.TableWalk**

Use if CPU stops and page tables are modified frequently (e.g. by OS).

Semihosting is a technique for an application program running on an ARM processor to communicate with the host computer of the debugger. This way the application can use the I/O facilities of the host computer like keyboard input, screen output, and file I/O. This is especially useful if the target platform does not yet provide these I/O facilities or in order to output additional debug information in `printf()` style.

A semihosting call from the application causes an exception by a SVC (SWI) instruction together with a certain SVC number to indicate a semihosting request. The type of operation is passed in R0. R1 points to the other parameters. On Cortex-M semihosting is implemented using the BKPT instead of SVC instruction.

Normally semihosting is invoked by code within the C library functions of the ARM RealView compiler like `printf()` and `scanf()`. The application can also invoke the operations used for keyboard input, screen output, and file I/O directly. The operations are described in the RealView Compilation Tools Developer Guide from ARM in the chapter “Semihosting Operations”.

The debugger which needs to interface to the I/O facilities on the host provides two ways to handle a semihosting request which results in a SVC (SWI) or BKPT exception:

### SVC (SWI) Emulation Mode

---

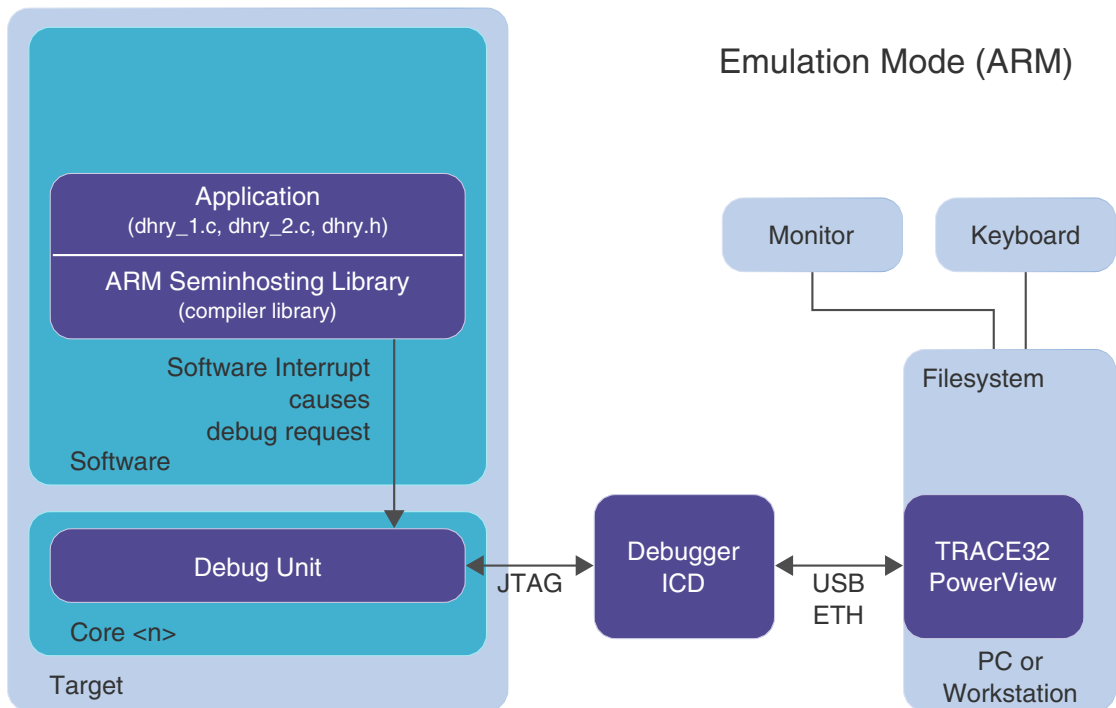
A breakpoint placed on the SVC exception entry stops the application. The debugger handles the request while the application is stopped, provides the required communication with the host, and restarts the application at the address which was stored in the link register R14 on the SVC exception call. Other as for the DCC mode the SVC parameter has to be 0x123456 to indicate a semihosting request.

This mode is enabled by **TERM.METHOD ARMSWI** [*<address>*] and by opening a **TERM.GATE** window for the semihosting screen output. The handling of the semihosting requests is only active when the **TERM.GATE** window is existing.

On ARM7 an on-chip or software breakpoint needs to be set at address 8 (SWI exception entry). On other ARM cores also the vector catch register can be used: **TrOnchip.Set SWI ON**. The Cortex-M does not need a breakpoint because it already uses the breakpoint instruction BKPT for the semihosting request.

When using the *<address>* option of the **TERM.METHOD ARMSWI** *<address>*, any memory location with a breakpoint on it can be used as a semihosting service entry instead of the SVC call at address 8. The application just needs to jump to that location. After servicing the request the program execution continues at that address (not at the address in the link register R14). You could for example place a 'BX R14' command at that address and hand the return address in R14. Since this method does not use the SVC command no parameter (0x123456) will be checked to identify a semihosting call.

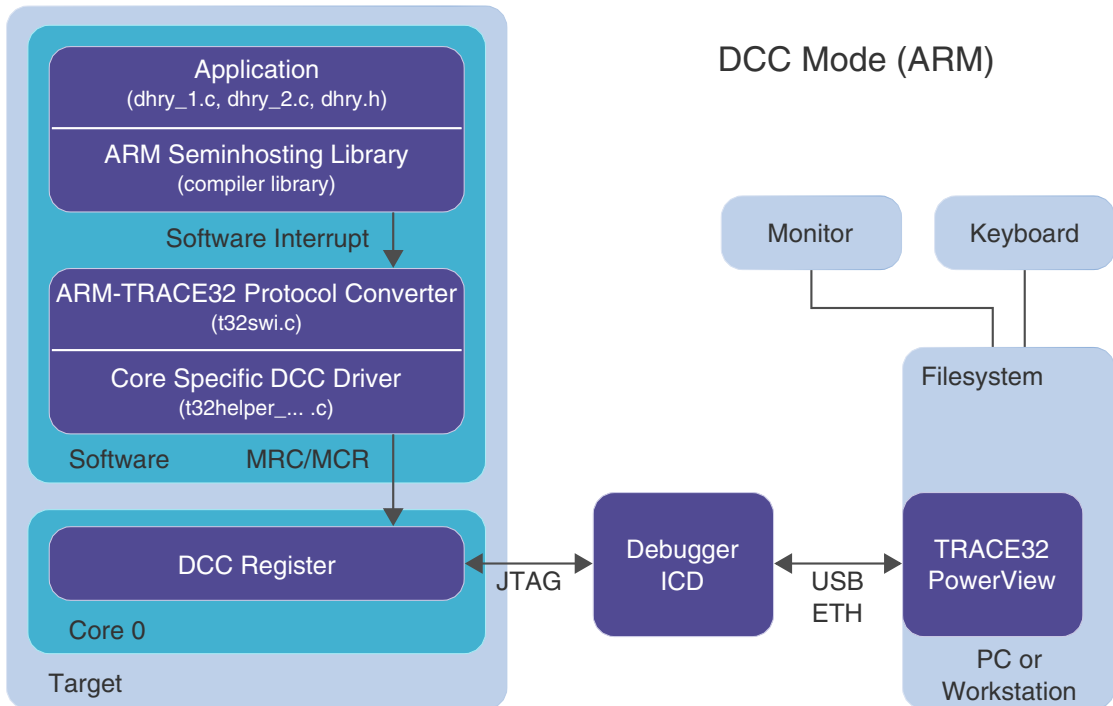
**TERM.HEAPINFO** defines the system stack and heap location. The C library reads these memory parameters by a SYS\_HEAPINFO semihosting call and uses them for initialization. An example can be found in ~/demo/arm/etc/semihosting\_arm\_emulation/swisoft\_<x>.cmm.



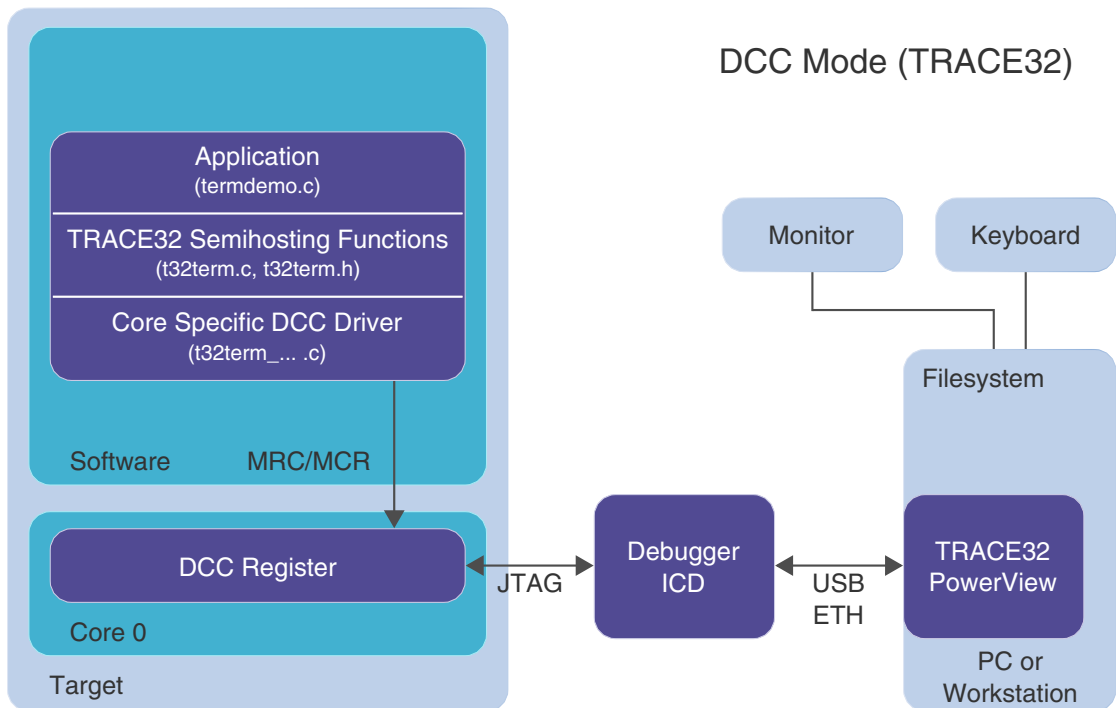
## DCC Communication Mode (DCC = Debug Communication Channel)

A semihosting exception handler will be called by the SVC (SWI) exception. It uses the Debug Communication Channel based on the JTAG interface to communicate with the host. The target application will not be stopped, but the semihosting exception handler needs to be loaded or linked to the application. The Cortex-M does not provide a DCC, therefore this mode can not be used.

This mode is enabled by **TERM.METHOD DCC3** and by opening a **TERM.GATE** window for the semihosting screen output. The handling of the semihosting requests is only active when the **TERM.GATE** window is existing. **TERM.HEAPINFO** defines the system stack and heap location. The ARM C library reads these memory parameters by a SYS\_HEAPINFO semihosting call and uses them for initialization. An example (swidcc\_x.cmm) and the source of the ARM compatible semihosting handler (t32swi.c, t32helper\_x.c) can be found in ~/demo/arm/etc/semihosting\_arm\_dcc



In case the ARM library for semihosting is not used, you can alternatively use the native TRACE32 format for the semihosting requests. Then the SWI handler (t32swi.c) is not required. You can send the requests directly via DCC. Find examples and source codes in `~/demo/arm/etc/semihosting_trace32_dcc`



# Virtual Terminal

The command **TERM** opens a terminal window which allows to communicate with the ARM core over the Debug Communications Channel (DCC). All data received from the comms channel are displayed and all data inputs to this window are sent to the comms channel. Communication occurs byte wide or up to four bytes per transfer. The following modes can be used:

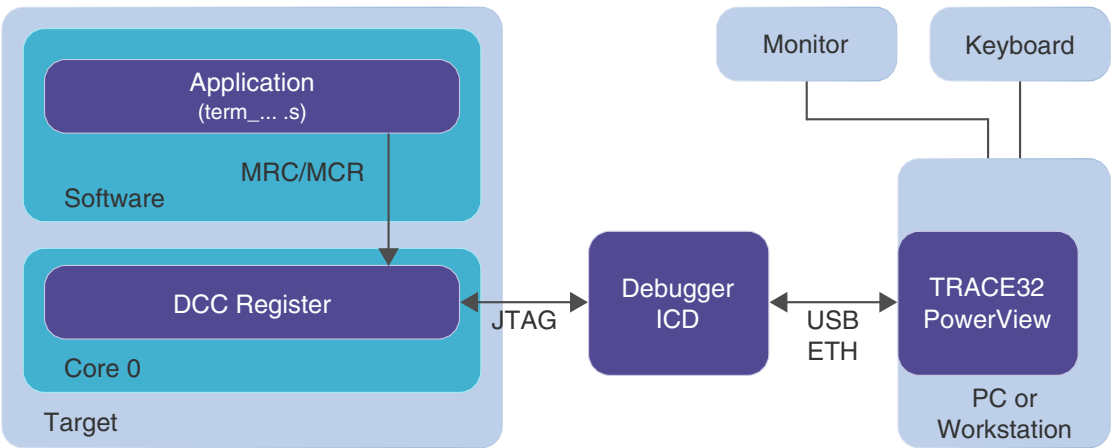
<b>DCC</b>	Use the DCC port of the JTAG interface to transfer 1 byte at once.
<b>DCC3</b>	Three byte mode. Allows binary transfers of up to 3 bytes per DCC transfer. The upper byte defines how many bytes are transferred (0 = one byte, 1 = two bytes, 2 = three bytes). This is the preferred mode of operation, as it combines arbitrary length messages with high bandwidth.
<b>DCC4A</b>	Four byte ASCII mode. Does not allow to transfer the byte 00. Each non-zero byte of the 32 bit word is a character in this mode.
<b>DCC4B</b>	Four byte binary mode. Used to transfer non-ASCII 32-bit data (e.g. to or from a file).

The **TERM.METHOD** command selects which mode is used (**DCC**, **DCC3**, **DCC4A** or **DCC4B**).

The communication mechanism is described e.g. in the ARM7TDMI data sheet in chapter 9.11. Only three move to/from coprocessor 14 instructions are necessary.

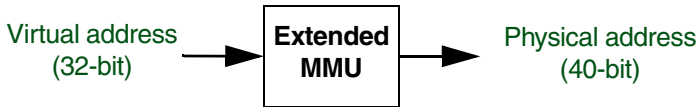
The TRACE32 ~/demo/arm/etc/virtual\_terminal directory contains examples for the different ARM families which demonstrate how the communication works.

## Virtual Terminal



# Large Physical Address Extension (LPAE)

LPAE is an optional extension for the ARMv7-AR architecture. It allows physical addresses above 32-bit. The instructions still use 32-bit addresses, but the extended memory management unit can map the address within a 40-bit physical memory range.



It is for example implemented on Cortex-A7 and Cortex-A15.

## Consequence for Debugging

We have extended only the physical address, because the virtual address is still 32-bit.

**Example:** Memory dump starting at physical address 0x0280004000.

“A:” = absolute address = physical address.

```
Data.dump A:02:80004000
```

Unfortunately the above command will result in a bus error (“????????”) on a real chip because the debug interface does not support physical accesses beyond the 4GByte. It will work on the TRACE32 Instruction Set Simulator and on virtual platforms.

In case the Debug Access Port (DAP) of the chip provides an AXI MEM-AP then the debugger can act as a bus master on the AXI, and you can access the physical memory independent of TLB entries.

```
Data.dump AXI:02:80004000
```

However this does not show you the cache contents in case of a write-back cache. For a cache coherent access you need to set:

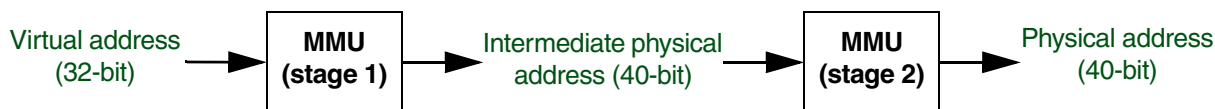
```
SYStem.Option AXIACEEnable ON
```

This requires that the CPU debug logic supports this setting. If the debug logic does not support coherent AXI accesses, this option is will be without effect.

# Virtualization Extension, Hypervisor

---

The 'Virtualization Extension' is an optional extension in ARMv7-A. It can for example be found on Cortex-A7 and Cortex-A15. It adds a 'Hypervisor' processor mode used to switch between different guest operating systems. The extension assumes **LPAAE** and **TrustZone**. It adds a second stage address translation.



## Consequence for Debugging

---

The debugger shows you the memory view of the mode the core is currently in. The address translation and therefore the view can/will be different for secure mode, non-secure mode, and hypervisor mode.

You can force a certain view/translation by switching to another mode or by using the access classes "Z:" (secure), "N:" (non-secure) or "H:" (hypervisor).

If you want to perform an access addressed by an intermediate physical address, you can use the 'I:' access class.

OS Awareness for multiple operating systems is under development. At the moment you can have only one OS Awareness at a time.

## Runtime Measurement

---

The command **RunTime** allows run time measurement based on polling the CPU run status by software. Therefore the result will be about a few milliseconds higher than the real value.

If the signal DBGACK on the JTAG connector is available, the measurement will automatically be based on this hardware signal which delivers very exact results. Please do not disable the option **SYStem.Option DBGACK**. The runtime of the debugger accesses while the CPU is halted would also be measured, otherwise.

## 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.



SYStem.CLOCK

Inform debugger about core clock

Format:

SYStem.CLOCK <frequency>

Informs the debugger about the core clock frequency. This information is used for analysis functions where the core frequency needs to be known. This command is only available if the debugger is used as front-end for virtual prototyping.

SYStem.CONFIG.state

Display target configuration

Format:

SYStem.CONFIG.state [/<tab>]

<tab>:

DebugPort | Jtag | MultiTap | DAP | COmponents

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

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

<tab>	Opens the <b>SYStem.CONFIG.state</b> window on the specified tab. For tab descriptions, refer to the tab descriptions below.
DebugPort	<p>The <b>DebugPort</b> tab (default) 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>DAP</b>	<p>The <b>DAP</b> tab informs the debugger about an ARM CoreSight Debug Access Port (DAP) and about how to control the DAP to access chip-internal memory busses (AHB, APB, AXI) or chip-internal JTAG interfaces.</p> <p>For descriptions of the commands on the <b>DAP</b> tab, see <a href="#">DAP</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> <i>&lt;parameter&gt;</i> <b>SYStem.MultiCore</b> <i>&lt;parameter&gt;</i> (deprecated)
<i>&lt;parameter&gt;</i> : <a href="#">(DebugPort)</a>	<b>CJTAGFLAGS</b> <i>&lt;flags&gt;</i> <b>CJTAGTCA</b> <i>&lt;value&gt;</i> <b>CONNECTOR</b> [MIPI34   MIPI20T] <b>CORE</b> <i>&lt;core&gt;</i> <i>&lt;chip&gt;</i> <b>CoreNumber</b> <i>&lt;number&gt;</i> <b>DEBUGPORT</b> [DebugCable0   DebugCableA   DebugCableB] <b>DEBUGPORTTYPE</b> [JTAG   SWD   CJTAG   CJTAGSWD] <b>NIDNTTRSTTORST</b> [ON   OFF]

<parameter>: (DebugPort cont.)	<b>NIDNTPSRISINGEDGE</b> [ON   OFF] <b>NIDNTRSTPOLARITY</b> [High   Low] <b>PortSHaRing</b> [ON   OFF   Auto] <b>Slave</b> [ON   OFF] <b>SWDP</b> [ON   OFF] <b>SWDPIDLEHIGH</b> [ON   OFF] <b>SWDPTargetSel</b> <value> <b>TriState</b> [ON   OFF]
<parameter>: (JTAG)	<b>CHIPDRLENGTH</b> <bits> <b>CHIPDRPATTERN</b> [Standard   Alternate <pattern>] <b>CHIPDRPOST</b> <bits> <b>CHIPDRPRE</b> <bits> <b>CHIPIRLENGTH</b> <bits> <b>CHIPIRPATTERN</b> [Standard   Alternate <pattern>] <b>CHIIRPOST</b> <bits> <b>CHIIRPRE</b> <bits>
<parameter>: (JTAG cont.)	<b>DAP2DRPOST</b> <bits> <b>DAP2DRPRE</b> <bits> <b>DAP2IRPOST</b> <bits> <b>DAP2IRPRE</b> <bits> <b>DAPDRPOST</b> <bits> <b>DAPDRPRE</b> <bits> <b>DAPIRPOST</b> <bits> <b>DAPIRPRE</b> <bits>
<parameter>: (JTAG cont.)	<b>DRPOST</b> <bits> <b>DRPRE</b> <bits> <b>ETBDRPOST</b> <bits> <b>ETBDRPRE</b> <bits> <b>ETBIRPOST</b> <bits> <b>ETBIRPRE</b> <bits> <b>IRPOST</b> <bits> <b>IRPRE</b> <bits>
<parameter>: (JTAG cont.)	<b>NEXTDRPOST</b> <bits> <b>NEXTDRPRE</b> <bits> <b>NEXTIRPOST</b> <bits> <b>NEXTIRPRE</b> <bits> <b>RTPDRPOST</b> <bits> <b>RTPDRPRE</b> <bits> <b>RTPIRPOST</b> <bits> <b>RTPIRPRE</b> <bits>
<parameter>: (JTAG cont.)	<b>Slave</b> [ON   OFF] <b>TAPState</b> <state> <b>TCKLevel</b> <level> <b>TriState</b> [ON   OFF]

<parameter>: (Multitap)	<b>CFGCONNECT</b> <code> <b>DAP2TAP</b> <tap> <b>DAPTAP</b> <tap> <b>DEBUGTAP</b> <tap> <b>ETBTAP</b> <tap> <b>MULTITAP</b> [NONE   IcepickA   IcepickB   IcepickC   IcepickD   IcepickBB   IcepickBC   IcepickCC   IcepickDD   STCLTAP1   STCLTAP2   STCLTAP3   <b>MSMTAP</b> <irlength> <irvalue> <drlength> <drvalue> <b>JtagSequence</b> <subcommands>] <b>NJCR</b> <tap> <b>RTPTAP</b> <tap> <b>SLAVETAP</b> <tap>
<parameter>: (DAP)	<b>AHBACCESSPORT</b> <port> <b>APBACCESSPORT</b> <port> <b>AXIACCESSPORT</b> <port> <b>COREJTAGPORT</b> <port> <b>DAP2AHBACCESSPORT</b> <port> <b>DAP2APBACCESSPORT</b> <port> <b>DAP2AXIACCESSPORT</b> <port> <b>DAP2COREJTAGPORT</b> <port>
<parameter>: (DAP cont.)	<b>DAP2DEBUGACCESSPORT</b> <port> <b>DAP2JTAGPORT</b> <port> <b>DAP2AHBACCESSPORT</b> <port> <b>DEBUGACCESSPORT</b> <port> <b>JTAGACCESSPORT</b> <port> <b>MEMORYACCESSPORT</b> <port>
<parameter>: (CComponents)	<b>ADTF.Base</b> <address> <b>ADTF.RESET</b> <b>AET.Base</b> <address> <b>AET.RESET</b> <b>BMC.Base</b> <address> <b>BMC.RESET</b> <b>CMI.Base</b> <address> <b>CMI.RESET</b>
<parameter>: (CComponents cont.)	<b>CMI.TraceID</b> <id> <b>COREDEBUG.Base</b> <address> <b>COREDEBUG.RESET</b> <b>CTI.Base</b> <address> <b>CTI.Config</b> [NONE   ARMV1   ARMPostInit   OMAP3   TMS570   CortexV1   QV1] <b>CTI.RESET</b> <b>DRM.Base</b> <address> <b>DRM.RESET</b>

**<parameter>:**  
(Components  
cont.)

**DTM.RESET**  
**DTM.Type** [None | Generic]  
**DWT.Base** <address>  
**DWT.RESET**  
**EPM.Base** <address>  
**EPM.RESET**  
**ETB2AXI.Base** <address>  
**ETB2AXI.RESET**

**<parameter>:**  
(Components  
cont.)

**ETB.ATBSource** <source>  
**ETB.Base** <address>  
**ETB.NoFlush** [ON | OFF]  
**ETB.RESET**  
**ETB.Size** <size>  
**ETF.ATBSource** <source>  
**ETF.Base** <address>  
**ETF.RESET**  
**ETM.Base** <address>

**<parameter>:**  
(Components  
cont.)

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

**<parameter>:**  
(Components  
cont.)

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

**<parameter>:**  
(Components  
cont.)

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

*<parameter>*:  
(Components  
cont.)

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

*<parameter>*:  
(Components  
cont.)

**SC.Base** *<address>*  
**SC.RESET**  
**SC.TraceID** *<id>*  
**STM.Base** *<address>*  
**STM.Mode** [NONE | XTiv2 | SDTI | STP | STP64 | STPv2]  
**STM.RESET**  
**STM.Type** [None | GenericARM | SDTI | TI]  
**TPIU.ATBSource** *<source>*  
**TPIU.Base** *<address>*  
**TPIU.RESET**  
**TPIU.Type** [CoreSight | Generic]

*<parameter>*:  
(Deprecated)

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

*<parameter>*:  
(Deprecated cont.)

**DTMETBFUNNELPORT** *<port>*  
**DTMFUNNEL2PORT** *<port>*  
**DTMFUNNELPORT** *<port>*  
**DTMTPIUFUNNELPORT** *<port>*  
**DWTBASE** *<address>*  
**ETB2AXIBASE** *<address>*  
**ETBBASE** *<address>*

*<parameter>*:  
(Deprecated cont.)

**ETBFUNNELBASE** *<address>*  
**ETFBASE** *<address>*  
**ETMBASE** *<address>*  
**ETMETBFUNNELPORT** *<port>*  
**ETMFUNNEL2PORT** *<port>*  
**ETMFUNNELPORT** *<port>*  
**ETMTPIUFUNNELPORT** *<port>*  
**FILLDRZERO** [ON | OFF]

<b>&lt;parameter&gt;:</b> (Deprecated cont.)	<b>FUNNEL2BASE</b> <address> <b>FUNNELBASE</b> <address> <b>HSMBASE</b> <address> <b>HTMBASE</b> <address> <b>HTMETBFUNNELPORT</b> <port> <b>HTMFUNNEL2PORT</b> <port> <b>HTMFUNNELPORT</b> <port> <b>HTMTPIUFUNNELPORT</b> <port>
<b>&lt;parameter&gt;:</b> (Deprecated cont.)	<b>ITMBASE</b> <address> <b>ITMETBFUNNELPORT</b> <port> <b>ITMFUNNEL2PORT</b> <port> <b>ITMFUNNELPORT</b> <port> <b>ITMTPIUFUNNELPORT</b> <port> <b>PERBASE</b> <address> <b>RAMBASE</b> <address> <b>RTPBASE</b> <address>
<b>&lt;parameter&gt;:</b> (Deprecated cont.)	<b>SDTIBASE</b> <address> <b>STMBASE</b> <address> <b>STMETBFUNNELPORT</b> <port> <b>STMFUNNEL2PORT</b> <port> <b>STMFUNNELPORT</b> <port> <b>STMTPIUFUNNELPORT</b> <port> <b>TIADTFBASE</b> <address> <b>TIDRMBASE</b> <address>
<b>&lt;parameter&gt;:</b> (Deprecated cont.)	<b>TIEPMBASE</b> <address> <b>TIICEBASE</b> <address> <b>TIOCPBASE</b> <address> <b>TIOCPTYPE</b> <type> <b>TIPMIBASE</b> <address> <b>TISCBASE</b> <address> <b>TISTMBASE</b> <address>
<b>&lt;parameter&gt;:</b> (Deprecated cont.)	<b>TPIUBASE</b> <address> <b>TPIUFUNNELBASE</b> <address> <b>TRACEETBFUNNELPORT</b> <port> <b>TRACEFUNNELPORT</b> <port> <b>TRACETPIUFUNNELPORT</b> <port> <b>view</b>

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

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

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

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

### Syntax Remarks:

---

The commands are not case sensitive. Capital letters show how the command can be shortened.

**Example:** "SYStem.CONFIG.DWT.Base 0x1000" -> "SYS.CONFIG.DWT.B 0x1000"

The dots after "SYStem.CONFIG" can alternatively be a blank.

**Example:** "SYStem.CONFIG.DWT.Base 0x1000" or "SYStem.CONFIG DWT Base 0x1000".



### **CJTAGFLAGS** <flags>

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

Default: 0

### **CJTAGTCA** <value>

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

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

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

Default: MIPI34 if CombiProbe is used, MIPI20T if uTrace is used.

### **CORE** <core> <chip>

The command helps to identify debug and trace resources which are commonly used by different cores. The command might be required in a multicore environment if you use multiple debugger instances (multiple TRACE32 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 will control the (same) trace port. Sometimes it does not hurt if such a module will be controlled twice. So even then it might work. But the correct specification which might be a must is to tell the debugger that these cores sharing resources are 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 they share the same resource if the control registers of the resource has the same address.

Default:

<core> depends on CPU selection, usually 1.

<chip> derived from `CORE=` parameter in the configuration file (config.t32), usually 1. If you start multiple debugger instances with the help of t32start.exe you will get ascending values (1, 2, 3,...).

**CoreNumber** <number>

Number of cores to be considered in an SMP (symmetric multiprocessing) debug session. There are core types like ARM11MPCore, CortexA5MPCore, CortexA9MPCore and Scorpion which can be used as a single core processor or as a scalable multicore processor of the same type. If you intend to debug more than one such core in an SMP debug session you need to specify the number of cores you intend to debug.

Default: 1.

**DEBUGPORT**

[**DebugCable0** | **DebugCableA** | **DebugCableB**]

It specifies which probe cable shall be used e.g. "DebugCableA" or "DebugCableB". At the moment only the CombiProbe allows to connect more than one probe cable.

Default: depends on detection.

**DEBUGPORTTYPE**

[**JTAG** | **SWD** | **CJTAG** | **CJTAGSWD**]

It specifies the used debug port type "JTAG", "SWD", "CJTAG", "CJTAG-SWD". It assumes the selected type is supported by the target.

Default: JTAG.

### What is NIDnT?

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

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

**NIDNTPSRISINGEDGE**  
[ON | OFF]

Send data on rising edge for NIDnT PS switching.

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

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

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

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

**NIDNTRSTPOLARITY**  
[High | Low]

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

When connecting via NIDnT to a target system, the reset line might be a high-active signal.

To configure TRACE32 to use a high-active reset signal, use `SYStem.CONFIG NIDNTRSTPOLARITY High`

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

**NIDNTTRSTTORST**  
[ON | OFF]

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

When connecting via NIDnT (e.g. a microSD card slot) to the target system, the reset line might not include a pull-up on the target system.

To circumvent problems, TRACE32 allows to drive the target reset line via the TRST signal of an ARM debug cable.

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

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

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

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

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

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

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

**Slave** [ON | OFF]

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

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

Default: OFF.

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

**SWDP** [ON | OFF]

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

Default: OFF.

**SWDPIdleHigh**  
[ON | OFF]

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

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

Default: OFF.

**SWDPTargetSel** <value>

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

Default: 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.

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.

... <b>DRPOST</b> <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.
... <b>DRPRE</b> <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.
... <b>IRPOST</b> <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.
... <b>IRPRE</b> <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.
<b>CHIPDRLLENGTH</b> <bits>	Number of Data Register (DR) bits which needs to get a certain BYPASS pattern.
<b>CHIPDRPATTERN</b> [Standard   Alternate] <pattern>	Data Register (DR) pattern which shall be used for BYPASS instead of the standard (1...1) pattern.
<b>CHIPIRLLENGTH</b> <bits>	Number of Instruction Register (IR) bits which needs to get a certain BYPASS pattern.
<b>CHIPIRPATTERN</b> [Standard   Alternate] <pattern>	Instruction Register (IR) pattern which shall be used for BYPASS instead of the standard pattern.

**Slave [ON | OFF]**

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

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

Default: OFF.

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

For CortexM: Please check also

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

**TAPState <state>**

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

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

Default: 7 = Select-DR-Scan.

**TCKLevel <level>**

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

Default: 0.

**TriState [ON | OFF]**

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

Please note:

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

Default: OFF.

## TAP types:

Core TAP providing access to the debug register of the core you intend to debug.

-> DRPOST, DRPRE, IRPOST, IRPRE.

DAP (Debug Access Port) TAP providing access to the debug register of the core you intend to debug. It might be needed additionally to a Core TAP if the DAP is only used to access memory and not to access the core debug register.

-> DAPDRPOST, DAPDRPRE, DAPIRPOST, DAPIRPRE.

DAP2 (Debug Access Port) TAP in case you need to access a second DAP to reach other memory locations.

-> DAP2DRPOST, DAP2DRPRE, DAP2IRPOST, DAP2IRPRE.

ETB (Embedded Trace Buffer) TAP if the ETB has its own TAP to access its control register (typical with ARM11 cores).

-> ETBDRPOST, ETBDRPRE, ETBIRPOST, ETBIRPRE.

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

-> NEXTDRPOST, NEXTDRPRE, NEXTIRPOST, NEXTIRPRE.

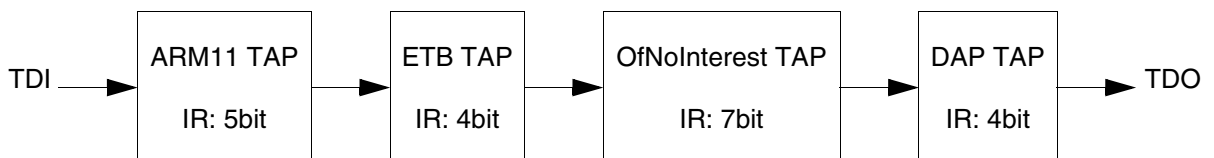
RTP (RAM Trace Port) TAP if the RTP has its own TAP to access its control register.

-> RTPDRPOST, RTPDRPRE, RTPIRPOST, RTPIRPRE.

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

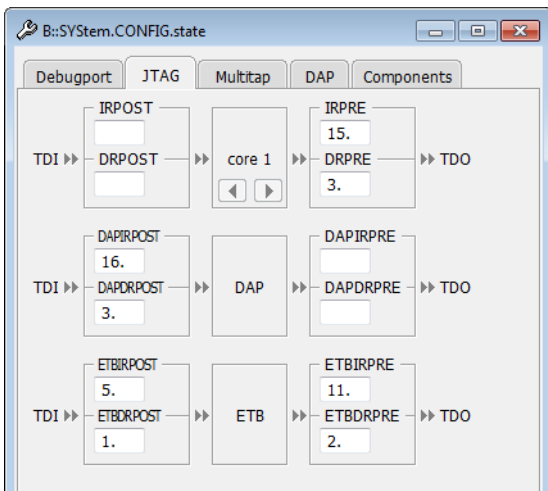
-> CHIPDRPOST, CHIPDRPRE, CHIIRPOST, CHIIRPRE.

## 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.
```



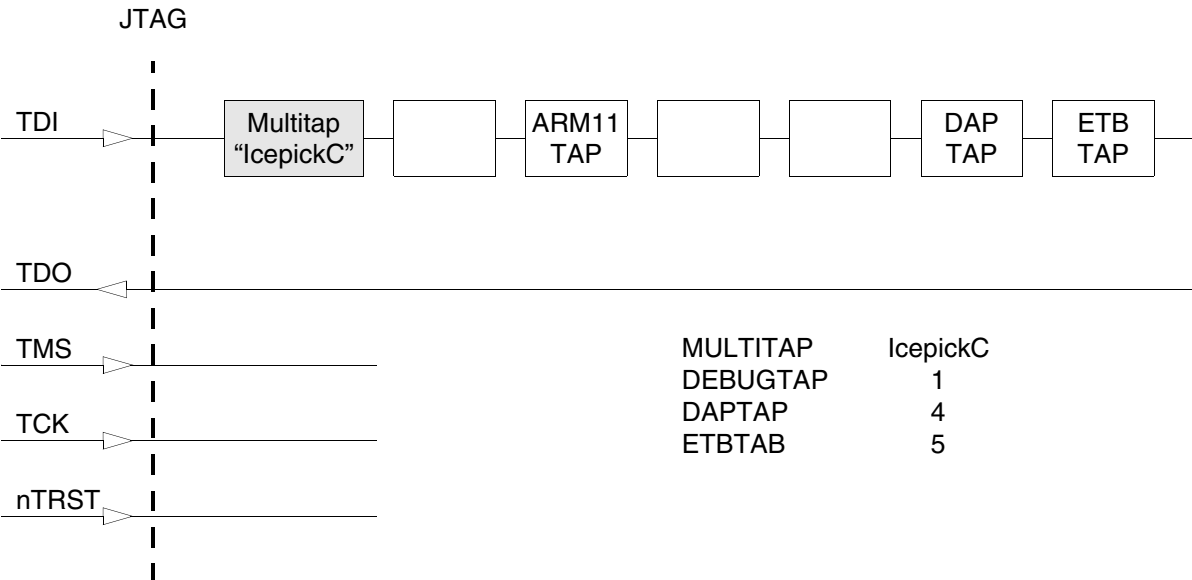


<parameters> describing a system level TAP “Multitap”

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:



CFGCONNECT <code>

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

Used if MULTITAP=STCLTAPx.

DAPTAP <tap>

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

Used if MULTITAP=Icepickx.

DAP2TAP <tap>

Specifies the TAP number which needs to be activated to get a 2nd DAP TAP in the JTAG chain.

Used if MULTITAP=Icepickx.

**DEBUGTAP** <tap>

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

Used if MULTITAP=Icepickx.

**ETBTAP** <tap>

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

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

**MULTITAP**

[NONE | IcepickA | IcepickB  
| IcepickC | IcepickD |  
IcepickBB | IcepickBC |  
IcepickCC | IcepickDD |  
STCLTAP1 | STCLTAP2 |  
STCLTAP3 | MSMTAP  
<irlength> <irvalue>  
<drlength> <drvalue>  
JtagSequence <subcmds>]

Selects the type and version of the MULTITAP.

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

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

For a description of the **JtagSequence** <subcommands>, see [SYStem.CONFIG.MULTITAP JtagSequence](#).

**NJCR** <tap>

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

Used if MULTITAP=Icepickx.

**RTPTAP** <tap>

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

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

**SLAVETAP** <tap>

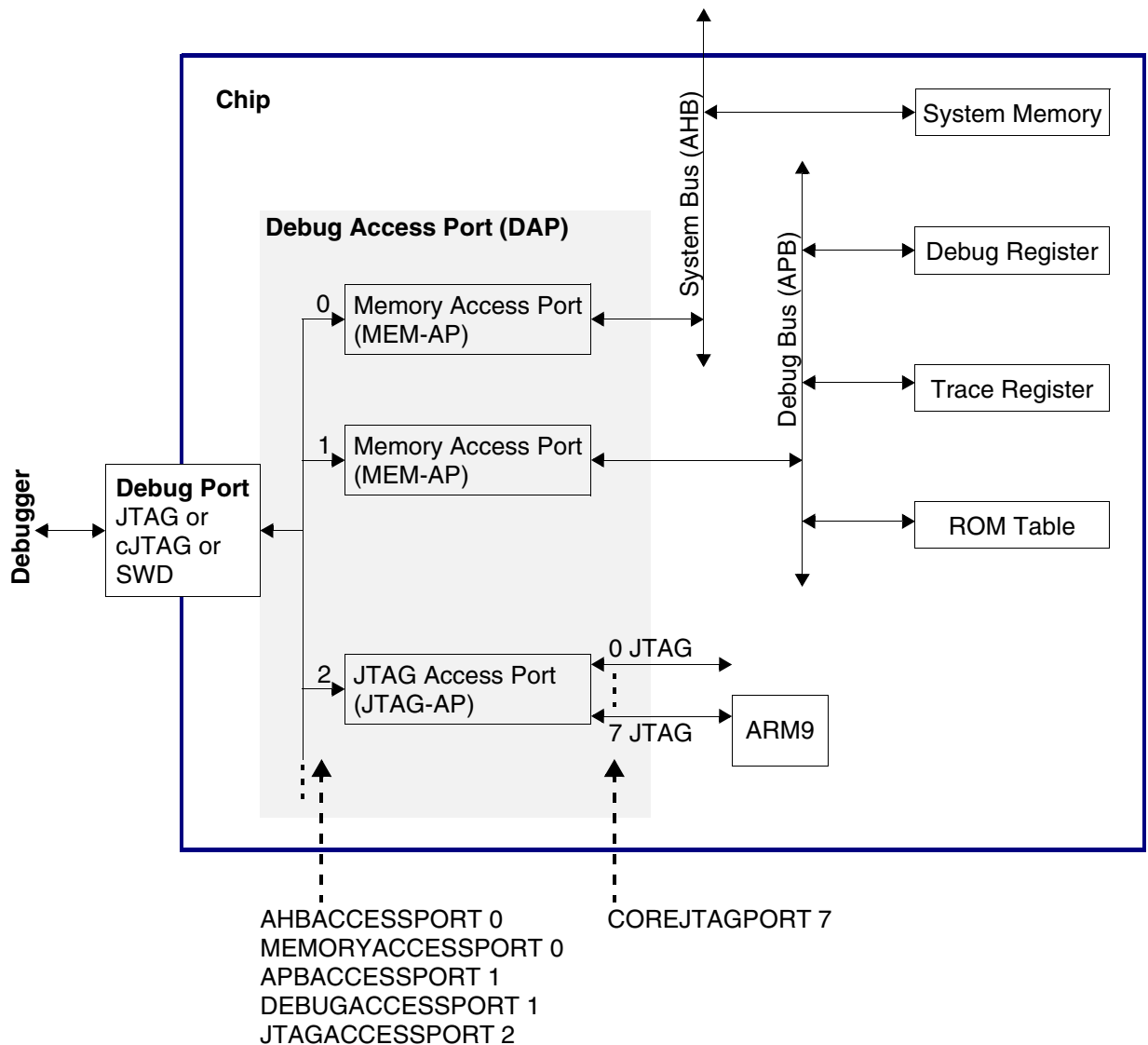
Specifies the TAP number to get the Icepick of the sub-system in the JTAG scan chain.

Used if MULTITAP=IcepickXY (two Icepicks).

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

1. Different memory busses (AHB, APB, AXI). This is especially important if the on-chip debug register needs to be accessed this way. You can access the memory buses by using certain access classes with the debugger commands: “AHB:”, “APB:”, “AXI:”, “DAP”, “E:”. The interface to these buses is called Memory Access Port (MEM-AP).
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number between 0 and 7 denotes the JTAG interfaces to be addressed.
3. At emulation or simulation system with using bus transactors the access to the busses must be specified by using the transactor identification name instead using the access port commands. For emulations/simulations with a DAP transactor the individual bus transactor name don't need to be configured. Instead of this the DAP transactor name need to be passed and the regular access ports to the busses.

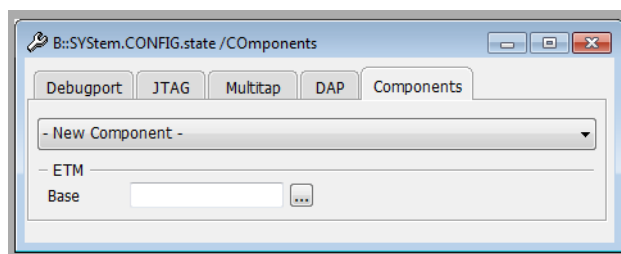
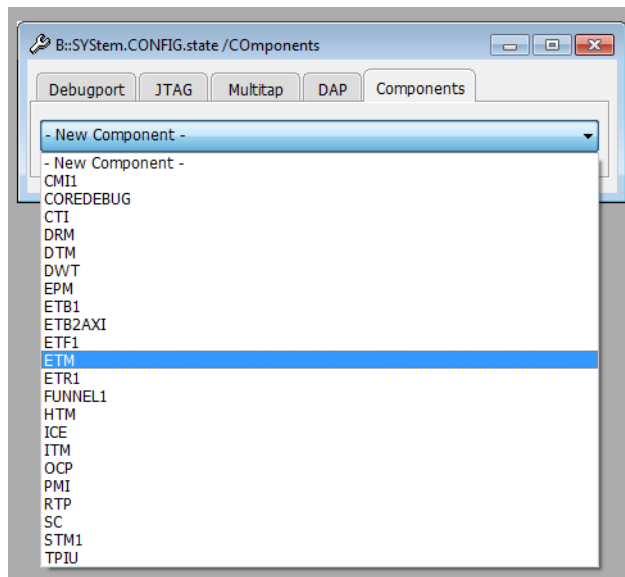
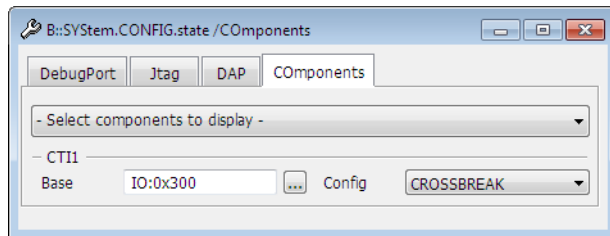
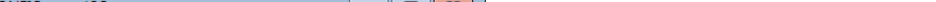
Example:



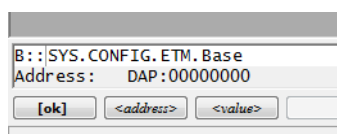
<b>AHBACCESSPORT</b> <port>	DAP access port number (0-255) which shall be used for "AHB:" access class. Default: <port>=0.
<b>APBACCESSPORT</b> <port>	DAP access port number (0-255) which shall be used for "APB:" access class. Default: <port>=1.
<b>AXIACCESSPORT</b> <port>	DAP access port number (0-255) which shall be used for "AXI:" access class. Default: port not available
<b>COREJTAGPORT</b> <port>	JTAG-AP port number (0-7) connected to the core which shall be debugged.

<b>DAP2AHBACCESSPORT</b> <port>	DAP2 access port number (0-255) which shall be used for “AHB2:” access class. Default: <port>=0.
<b>DAP2APBACCESSPORT</b> <port>	DAP2 access port number (0-255) which shall be used for “APB2:” access class. Default: <port>=1.
<b>DAP2AXIACCESSPORT</b> <port>	DAP2 access port number (0-255) which shall be used for “AXI2:” access class. Default: port not available
<b>DAP2DEBUGACCESS- PORT</b> <port>	DAP2 access port number (0-255) where the debug register can be found (typically on APB). Used for “DAP2:” access class. Default: <port>=1.
<b>DAP2COREJTAGPORT</b> <port>	JTAG-AP port number (0-7) connected to the core which shall be debugged. The JTAG-AP can be found on another DAP (DAP2).
<b>DAP2JTAGPORT</b> <port>	JTAG-AP port number (0-7) for an (other) DAP which is connected to a JTAG-AP.
<b>DAP2MEMORYACCESS- PORT</b> <port>	DAP2 access port number where system memory can be accessed even during runtime (typically on AHB). Used for “E:” access class while running, assuming “SYStem.MemoryAccess DAP2”. Default: <port>=0.
<b>DEBUGACCESSPORT</b> <port>	DAP access port number (0-255) where the debug register can be found (typically on APB). Used for “DAP:” access class. Default: <port>=1.
<b>JTAGACCESSPORT</b> <port>	DAP access port number (0-255) of the JTAG Access Port.
<b>MEMORYACCESSPORT</b> <port>	DAP access port number where system memory can be accessed even during runtime (typically on AHB). Used for “E:” access class while running, assuming “SYStem.MemoryAccess DAP”. Default: <port>=0.
<b>AHBNAME</b> <name>	AHB bus transactor name that shall be used for “AHB:” access class.
<b>APBNAME</b> <name>	APB bus transactor name that shall be used for “APB:” access class.
<b>AXINAME</b> <name>	AXI bus transactor name that shall be used for “AXI:” access class.
<b>DAP2AHBNAME</b> <name>	AHB bus transactor name that shall be used for “AHB2:” access class.

<b>DAP2APBNAME</b> <name>	APB bus transactor name that shall be used for “APB2:” access class.
<b>DAP2AXINAME</b> <name>	AXI bus transactor name that shall be used for “AXI2:” access class.
<b>DAP2DEBUGBUSNAME</b> <name>	APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP2:” access class.
<b>DAP2MEMORYBUSNAME</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 “SYStem.MemoryAccess DAP2”.
<b>DEBUGBUSNAME</b> <name>	APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP:” access class.
<b>MEMORYBUSNAME</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 “SYStem.MemoryAccess DAP”.
<b>DAPNAME</b> <name>	DAP transactor name that shall be used for DAP access ports.
<b>DAP2NAME</b> <name>	DAP transactor name that shall be used for DAP access ports of 2nd order.



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.



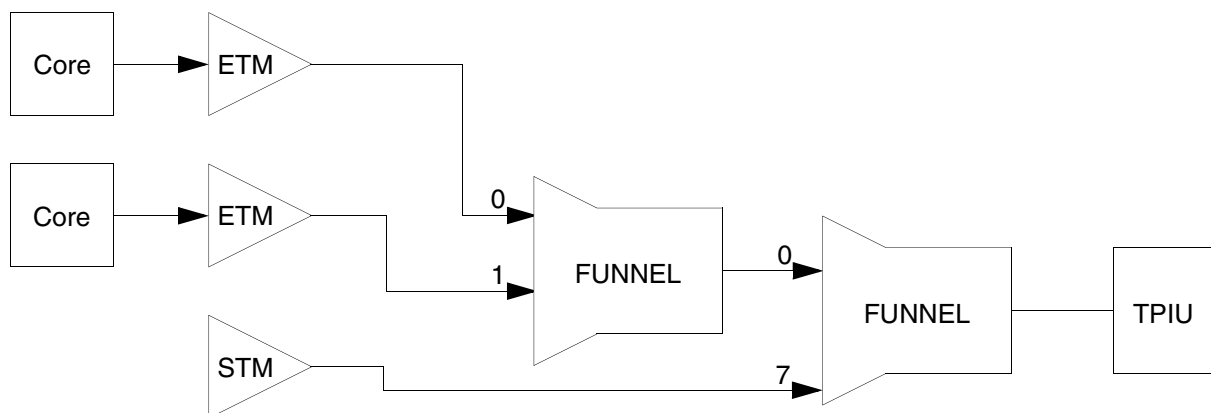


You can have several of the following components: CMI, ETB, ETF, ETR, FUNNEL, STM.

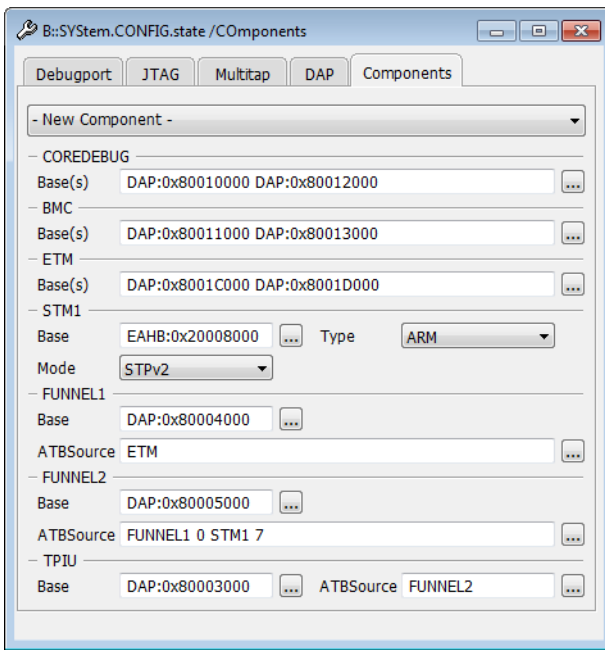
**Example:** FUNNEL1, FUNNEL2, FUNNEL3,...

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](#).

... **.RESET**

Undo the configuration for this component. This does not cause a physical reset for the component on the chip.

For a list of possible components including a short description see [Components and Available Commands](#).

... **.TraceID** <id>

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

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

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

For a list of possible components including a short description see [Components and Available Commands](#).

**CTI.Config** <type>

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

NONE: The CTI is not used by the debugger.

ARMV1: This mode is used for ARM7/9/11 cores which support synchronous halt, only.

ARMPostInit: Like ARMV1 but the CTI connection differs from the ARM recommendation.

OMAP3: This mode is not yet used.

TMS570: Used for a certain CTI connection used on a TMS570 derivative.

CortexV1: The CTI will be configured for synchronous start and stop via CTI. It assumes the connection of DBGRQ, DBGACK, DBGRESTART signals to CTI are done as recommended by ARM. The CTIBASE must be notified. “CortexV1” is the default value if a Cortex-A/R core is selected and the CTIBASE is notified.

QV1: This mode is not yet used.

ARMV8V1: Channel 0 and 1 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only.

ARMV8V2: Channel 2 and 3 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only.

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

Informs the debugger that a customer proprietary Data Trace Message (DTM) module is available. This causes the debugger to consider this source when capturing common trace data.

Trace data from this module will be recorded and can be accessed later but the unknown DTM module itself will not be controlled by the debugger.

**ETB.NoFlush** [ON | OFF]

Deactivates an ETB flush request at the end of the trace recording. This is a workaround for a bug on a certain chip. You will lose trace data at the end of the recording. Don't use it if not needed. Default: OFF.

**ETB.Size** <size>

Specifies the size of the Embedded Trace Buffer. The ETB size can normally be read out by the debugger. Therefore this command is only needed if this can not be done for any reason.

**ETM.StackMode** [**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**]

In case the funnel can not or may not be programmed by the debugger, this option needs to be OFF. Default is ON.

**HTM.Type** [**CoreSight** | **WPT**]

Selects the type of the AMBA AHB Trace Macrocell (HTM). CoreSight is the type as described in the ARM CoreSight manuals. WPT is a NXP proprietary trace module.

**L2CACHE.Type** [**NONE** | **Generic** | **L210** | **L220** | **L2C-310** | **AURORA** | **AURORA2**]

Selects the type of the level2 cache controller. L210, L220, L2C-310 are controller types provided by ARM. AURORAx are Marvell types. The 'Generic' type does not need certain treatment by the debugger.

**OCP.Type** <type>

Specifies the type of the OCP module. The <type> is just a number which you need to figure out in the chip documentation.

**RTP.PerBase** <address>

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

**RTP.RamBase** <address>

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

**STM.Mode** [**NONE** | **XTiv2** | **SDTI** | **STP** | **STP64** | **STPv2**]

Selects the protocol type used by the System Trace Module (STM).

**STM.Type** [None | Generic | ARM | SDTI | TI]

Selects the type of the System Trace Module (STM). Some types allow to work with different protocols (see STM.Mode).

**TPIU.Type** [CoreSight | Generic]

Selects the type of the Trace Port Interface Unit (TPIU).

CoreSight: Default. CoreSight TPIU. TPIU control register located at TPIU.Base <address> will be handled by the debugger.

Generic: Proprietary TPIU. TPIU control register will not be handled by the debugger.

## Components and Available Commands

---

See the description of the commands above. Please note that there is a common description for ... .ATBSource, ... .Base, , ... .RESET, ... .TraceID.

**ADTF.Base** <address>

**ADTF.RESET**

AMBA trace bus DSP Trace Formatter (ADTF) - Texas Instruments

Module of a TMS320C5x or TMS320C6x core converting program and data trace information in ARM CoreSight compliant format.

**AET.Base** <address>

**AET.RESET**

Advanced Event Triggering unit (AET) - Texas Instruments

Trace source module of a TMS320C5x or TMS320C6x core delivering program and data trace information.

**BMC.Base** <address>

**BMC.RESET**

Performance Monitor Unit (PMU) - ARM debug module, e.g. on Cortex-A/R

Bench-Mark-Counter (BMC) is the TRACE32 term for the same thing.

The module contains counter which can be programmed to count certain events (e.g. cache hits).

**CMI.Base** <address>

**CMI.RESET**

**CMI.TraceID** <id>

Clock Management Instrumentation (CMI) - Texas Instruments

Trace source delivering information about clock status and events to a system trace module.

**COREDEBUG.Base** <address>

**COREDEBUG.RESET**

Core Debug Register - ARM debug register, e.g. on Cortex-A/R

Some cores do not have a fix location for their debug register used to control the core. In this case it is essential to specify its location before you can connect by e.g. SYStem.Up.

**CTI.Base** <address>

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

**CTI.RESET**

Cross Trigger Interface (CTI) - ARM CoreSight module

If notified the debugger uses it to synchronously halt (and sometimes also to start) multiple cores.

**DRM.Base** <address>

**DRM.RESET**

Debug Resource Manager (DRM) - Texas Instruments

It will be used to prepare chip pins for trace output.

**DTM.RESET**

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

Data Trace Module (DTM) - generic, CoreSight compliant trace source module

If specified it will be considered in trace recording and trace data can be accessed afterwards.

DTM module itself will not be controlled by the debugger.

**DWT.Base** <address>

**DWT.RESET**

Data Watchpoint and Trace unit (DWT) - ARM debug module on Cortex-M cores

Normally fix address at 0xE0001000 (default).

**EPM.Base** <address>

**EPM.RESET**

Emulation Pin Manager (EPM) - Texas Instruments

It will be used to prepare chip pins for trace output.

**ETB2AXI.Base** <address>

**ETB2AXI.RESET**

ETB to AXI module

Similar to an ETR.

**ETB.ATBSource** <source>

**ETB.Base** <address>

**ETB.RESET**

**ETB.Size** <size>

Embedded Trace Buffer (ETB) - ARM CoreSight module

Enables trace to be stored in a dedicated SRAM. The trace data will be read out through the debug port after the capturing has finished.

**ETF.ATBSource** <source>

**ETF.Base** <address>

**ETF.RESET**

Embedded Trace FIFO (ETF) - ARM CoreSight module

On-chip trace buffer used to lower the trace bandwidth peaks.

**ETM.Base** <address>

**ETM.RESET**

Embedded Trace Macrocell (ETM) - ARM CoreSight module

Program Trace Macrocell (PTM) - ARM CoreSight module

Trace source providing information about program flow and data accesses of a core.

The ETM commands will be used even for PTM.

**ETR.ATBSource** <source>

**ETR.Base** <address>

**ETR.RESET**

Embedded Trace Router (ETR) - ARM CoreSight module

Enables trace to be routed over an AXI bus to system memory or to any other AXI slave.

**FUNNEL.ATBSource** <sourcelist>

**FUNNEL.Base** <address>

**FUNNEL.Name** <string>

**FUNNEL.PROGrammable** [ON | OFF]

**FUNNEL.RESET**

CoreSight Trace Funnel (CSTF) - ARM CoreSight module

Combines multiple trace sources onto a single trace bus (ATB = AMBA Trace Bus)

**HSM.Base** <address>

**HSM.RESET**

Hardware Security Module (HSM) - Infineon

**HTM.Base** <address>

**HTM.RESET**

**HTM.Type** [CoreSight | WPT]

AMBA AHB Trace Macrocell (HTM) - ARM CoreSight module

Trace source delivering trace data of access to an AHB bus.

**ICE.Base** <address>

**ICE.RESET**

ICE-Crusher (ICE) - Texas Instruments

**ITM.Base** <address>

**ITM.RESET**

Instrumentation Trace Macrocell (ITM) - ARM CoreSight module

Trace source delivering system trace information e.g. sent by software in printf() style.

**L2CACHE.Base** <address>

**L2CACHE.RESET**

**L2CACHE.Type** [NONE | Generic | L210 | L220 | L2C-310 | AURORA | AURORA2]

Level 2 Cache Controller

The debugger might need to handle the controller to ensure cache coherency for debugger operation.

**OCP.Base** <address>

**OCP.RESET**

**OCP.TraceID** <id>

**OCP.Type** <type>

Open Core Protocol watchpoint unit (OCP) - Texas Instruments

Trace source module delivering bus trace information to a system trace module.

**PMI.Base** <address>

**PMI.RESET**

**PMI.TraceID** <id>

Power Management Instrumentation (PMI) - Texas Instruments

Trace source reporting power management events to a system trace module.



**RTP.Base** <address>

**RTP.PerBase** <address>

**RTP.RamBase** <address>

**RTP.RESET**

RAM Trace Port (RTP) - Texas Instruments

Trace source delivering trace data about memory interface usage.

**SC.Base** <address>

**SC.RESET**

**SC.TraceID** <id>

Statistic Collector (SC) - Texas Instruments

Trace source delivering statistic data about bus traffic to a system trace module.

**STM.Base** <address>

**STM.Mode** [NONE | XTiv2 | SDTI | STP | STP64 | STPv2]

**STM.RESET**

**STM.Type** [None | Generic | ARM | SDTI | TI]

System Trace Macrocell (STM) - MIPI, ARM CoreSight, others

Trace source delivering system trace information e.g. sent by software in printf() style.

**TPIU.ATBSource** <source>

**TPIU.Base** <address>

**TPIU.RESET**

**TPIU.Type** [CoreSight | Generic]

Trace Port Interface Unit (TPIU) - ARM CoreSight module

Trace sink sending the trace off-chip on a parallel trace port (chip pins).

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

... **BASE** <address>

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

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

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

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

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

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

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

For a list of possible components including a short description see [Components and Available Commands](#).

... **PORT** <port>

Informs the debugger about which trace source is connected to which input port of which funnel. A CoreSight trace funnel provides 8 input ports (port 0-7) to combine the data of various trace sources to a common trace stream.

**Example:** SYStem.CONFIG STMFUNNEL2PORT 3

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

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

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

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

For a list of possible components including a short description see [Components and Available Commands](#).

**BYPASS** <seq>

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

**CTICONFIG** <type>

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

NONE: The CTI is not used by the debugger.

ARMV1: This mode is used for ARM7/9/11 cores which support synchronous halt, only.

ARMPostInit: Like ARMV1 but the CTI connection differs from the ARM recommendation.

OMAP3: This mode is not yet used.

TMS570: Used for a certain CTI connection used on a TMS570 derivative.

CortexV1: The CTI will be configured for synchronous start and stop via CTI. It assumes the connection of 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.

<b>DTMCONFIG [ON   OFF]</b>	<p> Informs the debugger that a customer proprietary Data Trace Message (DTM) module is available. This causes the debugger to consider this source when capturing common trace data. Trace data from this module will be recorded and can be accessed later but the unknown DTM module itself will not be controlled by the debugger. </p>
<b>FILLDRZERO [ON   OFF]</b>	<p> This changes the bypass data pattern for other TAPs in a multi-TAP JTAG chain. It changes the pattern from all "1" to all "0". This is a workaround for a certain chip problem. It is available on the ARM9 debugger, only. </p>
<b>TIOCPTYPE &lt;type&gt;</b>	<p> Specifies the type of the OCP module from Texas Instruments (TI). </p>
<b>view</b>	<p> Opens a window showing most of the SYStem.CONFIG settings and allows to modify them. </p>

### Deprecated and New Commands

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

#### SYStem.CONFIG <parameter>

<i>&lt;parameter&gt;</i> : (Deprecated)	<i>&lt;parameter&gt;</i> : (New)
<b>BMCBASE &lt;address&gt;</b>	<b>BMC.Base &lt;address&gt;</b>
<b>BYPASS &lt;seq&gt;</b>	<b>CHIPIRPRE &lt;bits&gt;</b>
	<b>CHIPIRLENGTH &lt;bits&gt;</b>
	<b>CHIPIRPATTERN.Alternate &lt;pattern&gt;</b>
<b>COREBASE &lt;address&gt;</b>	<b>COREDEBUG.Base &lt;address&gt;</b>
<b>CTIBASE &lt;address&gt;</b>	<b>CTI.Base &lt;address&gt;</b>
<b>CTICONFIG &lt;type&gt;</b>	<b>CTI.Config &lt;type&gt;</b>
<b>DEBUGBASE &lt;address&gt;</b>	<b>COREDEBUG.Base &lt;address&gt;</b>
<b>DTMCONFIG [ON   OFF]</b>	<b>DTM.Type.Generic</b>
<b>DTMETBFUNNELPORT &lt;port&gt;</b>	<b>FUNNEL4.ATBSource DTM &lt;port&gt; (1)</b>
<b>DTMFUNNEL2PORT &lt;port&gt;</b>	<b>FUNNEL2.ATBSource DTM &lt;port&gt; (1)</b>
<b>DTMFUNNELPORT &lt;port&gt;</b>	<b>FUNNEL1.ATBSource DTM &lt;port&gt; (1)</b>
<b>DTMTPIUFUNNELPORT &lt;port&gt;</b>	<b>FUNNEL3.ATBSource DTM &lt;port&gt; (1)</b>
<b>DWTBASE &lt;address&gt;</b>	<b>DWT.Base &lt;address&gt;</b>
<b>ETB2AXIBASE &lt;address&gt;</b>	<b>ETB2AXI.Base &lt;address&gt;</b>

<b>ETBBASE</b> <address>	<b>ETB1.Base</b> <address>
<b>ETBFUNNELBASE</b> <address>	<b>FUNNEL4.Base</b> <address>
<b>ETFBASE</b> <address>	<b>ETF1.Base</b> <address>
<b>ETMBASE</b> <address>	<b>ETM.Base</b> <address>
<b>ETMETBFUNNELPORT</b> <port>	<b>FUNNEL4.ATBSource ETM</b> <port> (1)
<b>ETMFUNNEL2PORT</b> <port>	<b>FUNNEL2.ATBSource ETM</b> <port> (1)
<b>ETMFUNNELPORT</b> <port>	<b>FUNNEL1.ATBSource ETM</b> <port> (1)
<b>ETMTPIUFUNNELPORT</b> <port>	<b>FUNNEL3.ATBSource ETM</b> <port> (1)
<b>FILLDRZERO</b> [ON   OFF]	<b>CHIPDRPRE</b> 0
	<b>CHIPDRPOST</b> 0
	<b>CHIPDRLENGTH</b> <bits_of_complete_dr_path>
	<b>CHIPDRPATTERN.Alternate</b> 0
<b>FUNNEL2BASE</b> <address>	<b>FUNNEL2.Base</b> <address>
<b>FUNNELBASE</b> <address>	<b>FUNNEL1.Base</b> <address>
<b>HSMBASE</b> <address>	<b>HSM.Base</b> <address>
<b>HTMBASE</b> <address>	<b>HTM.Base</b> <address>
<b>HTMETBFUNNELPORT</b> <port>	<b>FUNNEL4.ATBSource HTM</b> <port> (1)
<b>HTMFUNNEL2PORT</b> <port>	<b>FUNNEL2.ATBSource HTM</b> <port> (1)
<b>HTMFUNNELPORT</b> <port>	<b>FUNNEL1.ATBSource HTM</b> <port> (1)
<b>HTMTPIUFUNNELPORT</b> <port>	<b>FUNNEL3.ATBSource HTM</b> <port> (1)
<b>ITMBASE</b> <address>	<b>ITM.Base</b> <address>
<b>ITMETBFUNNELPORT</b> <port>	<b>FUNNEL4.ATBSource ITM</b> <port> (1)
<b>ITMFUNNEL2PORT</b> <port>	<b>FUNNEL2.ATBSource ITM</b> <port> (1)
<b>ITMFUNNELPORT</b> <port>	<b>FUNNEL1.ATBSource ITM</b> <port> (1)
<b>ITMTPIUFUNNELPORT</b> <port>	<b>FUNNEL3.ATBSource ITM</b> <port> (1)
<b>PERBASE</b> <address>	<b>RTP.PerBase</b> <address>
<b>RAMBASE</b> <address>	<b>RTP.RamBase</b> <address>
<b>RTPBASE</b> <address>	<b>RTP.Base</b> <address>
<b>SDTIBASE</b> <address>	<b>STM1.Base</b> <address>
	<b>STM1.Mode</b> SDTI
	<b>STM1.Type</b> SDTI
<b>STMBASE</b> <address>	<b>STM1.Base</b> <address>
	<b>STM1.Mode</b> STPV2
	<b>STM1.Type</b> ARM
<b>STMETBFUNNELPORT</b> <port>	<b>FUNNEL4.ATBSource STM1</b> <port> (1)
<b>STMFUNNEL2PORT</b> <port>	<b>FUNNEL2.ATBSource STM1</b> <port> (1)
<b>STMFUNNELPORT</b> <port>	<b>FUNNEL1.ATBSource STM1</b> <port> (1)
<b>STMTPIUFUNNELPORT</b> <port>	<b>FUNNEL3.ATBSource STM1</b> <port> (1)

TIADTFBASE <address>  
TIDRMBASE <address>  
TIEPMBASE <address>  
TIICEBASE <address>  
TIOCPBASE <address>  
TIOCTYPE <type>  
TIPMIBASE <address>  
TISCBASE <address>  
TISTMBASE <address>

TPIUBASE <address>  
TPIUFUNNELBASE <address>  
TRACEETBFUNNELPORT <port>  
TRACEFUNNELPORT <port>  
TRACETPIUFUNNELPORT <port>

view

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)  
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.CONFIG SMMU

Internal use

Format:

SYStem.CONFIG SMMU <x> <subcmd>

<x>:

1 ... 20

<subcmd>:

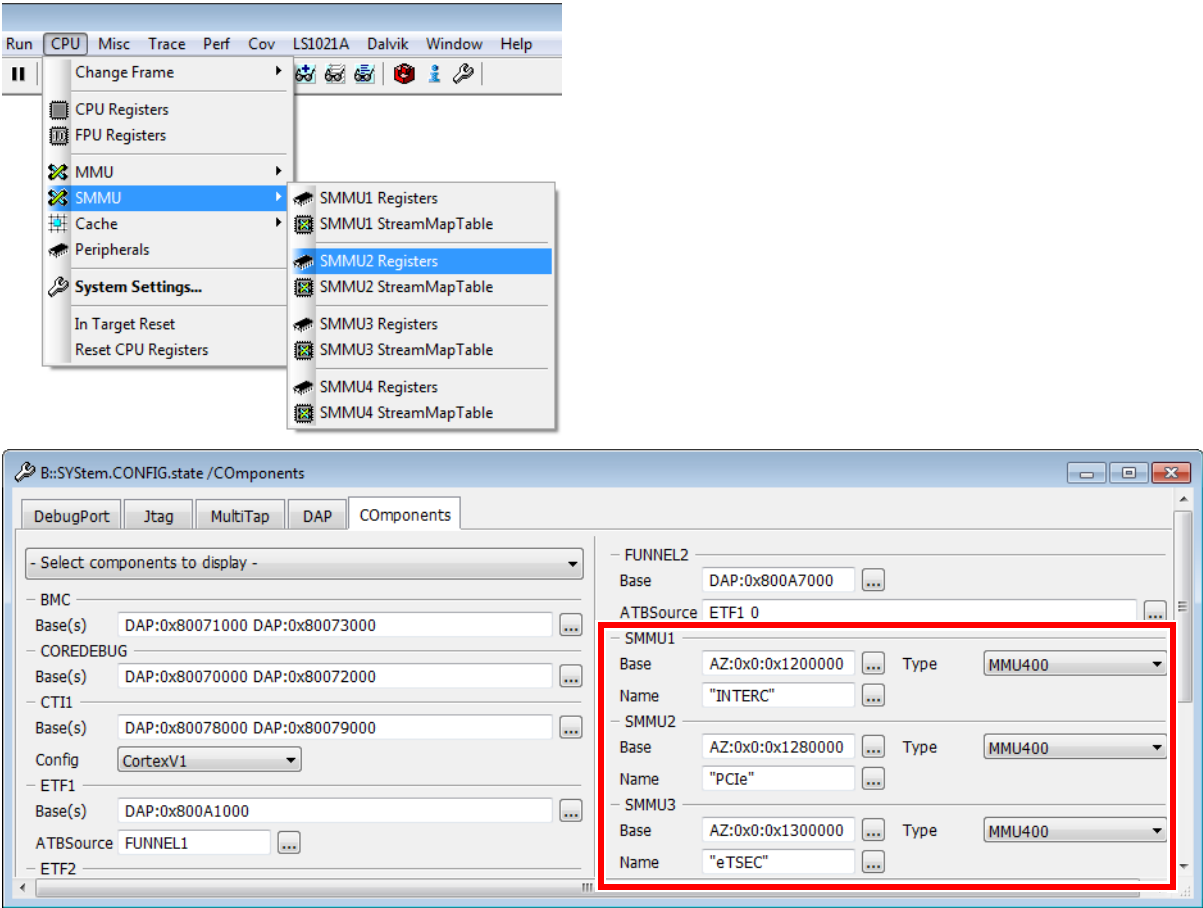
Base <base\_address>  
Type MMU400 | MMU401 | MMU500  
Name "<name>"  
RESET

For some CPUs with SMMUs, TRACE32 configures the SMMUs parameters *automatically* after you have selected a CPU with the **SYStem.CPU** command.

**NOTE:**

For a *manual* SMMU configuration, use the **SMMU.ADD** command.

You can access the automatically configured SMMUs through the **CPU** menu > **SMMU** submenu in TRACE32. The individual SMMU configurations can be viewed in the **SYStem.CONFIG.state /Component** window.



- <x> Serial number of the SMMU.
- Base Logical or physical base address of the memory-mapped SMMU register space.
- Type Defines the type of the ARM system MMU IP block: **MMU400**, **MMU401**, or **MMU500**.
- Name Assigns a user-defined name to an SMMU.
- RESET Resets the configuration of an SMMU specified with <x>.

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

*<cpu>*: **ARM7TDMI** | **ARM740TD** | ... (JTAG Debugger ARM7)  
**ARM9TDMI** | **ARM920T** | **ARM940T** |... (JTAG Debugger ARM9)  
**JANUS2** (JTAG Debugger Janus)  
**ARM1020E** | **ARM1022E** | **ARM1026EJ** |... (JTAG Debugger ARM10)  
**ARM1136J** | **ARM1136JF** |... (JTAG Debugger ARM11)  
**CORTEXA8** | **SCORPION** |... (JTAG Debugger Cortex-A)  
**CORTEXM3** |... (JTAG Debugger Cortex-M)

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

Default selection:

- ARM7TDMI if the JTAG Debugger for ARM7 is used.
- ARM9TDMI if the JTAG Debugger for ARM9 is used.
- JANUS2 if the JTAG Debugger for JANUS is used.
- ARM1020E if the JTAG Debugger for ARM10 is used.
- ARM1136J if the JTAG Debugger for ARM11 is used.
- CORTEXA8 if the JTAG Debugger for Cortex-A is used.
- CORTEXM3 if the JTAG Debugger for Cortex-M is used.



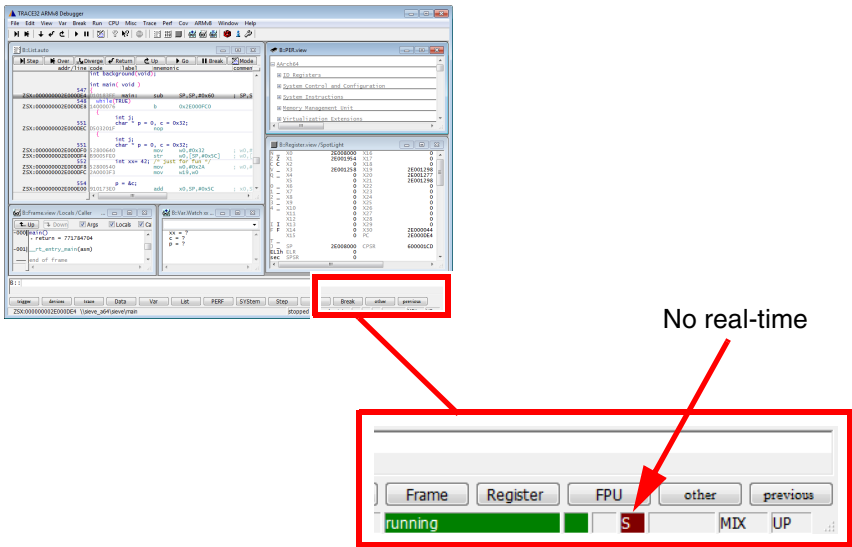
Format: **SYSystem.CpuAccess Enable | Denied | Nonstop**

Default: Denied.

Configures how memory access is handled during runtime.

- Enable** Allow intrusive run-time memory access.
- Denied** Lock intrusive run-time memory access.
- Nonstop** Lock all features of the debugger that affect the run-time behavior.

If **SYSystem.CpuAccess Enable** is set, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program. To make this possible, the program execution is shortly stopped by the debugger. Each stop takes some time depending on the speed of the JTAG port and the operations that should be performed. A white S against a red background in the TRACE32 **state line** warns you that the program is no longer running in real-time:



To update specific windows that display memory or variables while the program is running select the memory E: or the format option %E.

```
Data.dump E:0x100
Var.View %E first
```

Format:	<b>SYSystem.JtagClock</b> [ <i>&lt;frequency&gt;</i>   <b>RTCK</b>   <b>ARTCK</b> <i>&lt;frequency&gt;</i>   <b>CTCK</b> <i>&lt;frequency&gt;</i>   <b>CRTCK</b> <i>&lt;frequency&gt;</i> ]
	<b>SYSystem.BdmClock</b> (deprecated)
<i>&lt;frequency&gt;</i> :	<b>4 kHz...100 MHz</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.

<i>&lt;frequency&gt;</i>	<ul style="list-style-type: none"><li>The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the <b>SYSystem.state</b> window.</li><li>Besides a decimal number like “100000.” also short forms like “10kHz” or “15MHz” can be used. The short forms imply a decimal value, although no “.” is used.</li></ul>
<b>RTCK</b>	<p>The JTAG clock is controlled by the RTCK signal (<b>R</b>eturned <b>T</b>CK).</p> <p>On some processor derivatives (e.g. ARMxxxE-S) 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.</p> <p>In case you have a processor derivative requiring a synchronization of the processor clock and the JTAG clock, but your target does not provide an RTCK signal, you need to select a fix JTAG clock below 1/6 of the processor clock (ARM7, ARM9), below 1/8 of the processor clock (ARM11), respectively.</p> <p>When RTCK is selected, the frequency depends on the processor clock and on the propagation delays. The maximum reachable frequency is about 16 MHz.</p>

SYSystem.JtagClock RTCK

## **ARTCK**

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

The **RTCK** mode allows theoretical frequencies up to 1/6 (ARM7, ARM9) or 1/8 (ARM11) 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 1/3 of the processor clock speed. TDI and TMS will be delayed by 1/2 TCK clock cycle. TDO will be sampled with RTCK.

## **CTCK**

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

## **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, fix frequency.

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 ARM 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, EDBGREQ must have a pull-down resistor.

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

*<mode>*:  
**Cerberus**  
**CPU**  
**DAP**  
**NEXUS**  
**TSMON3**  
**TSMON**  
**PTMON3**  
**PTMON**  
**QMON**  
**UDMON3**  
**UDMON**  
**RealMON**  
**TrkMON**  
**GdbMON**  
**Denied**

Default: Denied.

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.

#### **Cerberus**

The memory access is done through an Infineon proprietary Cerberus module. This memory access is only available and selectable on a few Infineon processors and only by script or in the command line.

#### **CPU**

A run-time memory access is made without CPU intervention while the program is running. This is only possible on the instruction set simulator.

#### **DAP**

A run-time memory access is done via a Memory Access Port (MEM-AP) of the Debug Access Port (DAP). This is only possible if a DAP is available on the chip and if the memory bus is connected to it (Cortex, CoreSight). The debugger uses the AXI MEM-AP specified by **SYStem.CONFIG AXIACCESSPORT** if available, the MEM-AP (typically AHB) specified by **SYStem.CONFIG MEMORYACCESSPORT** otherwise.

#### **NEXUS**

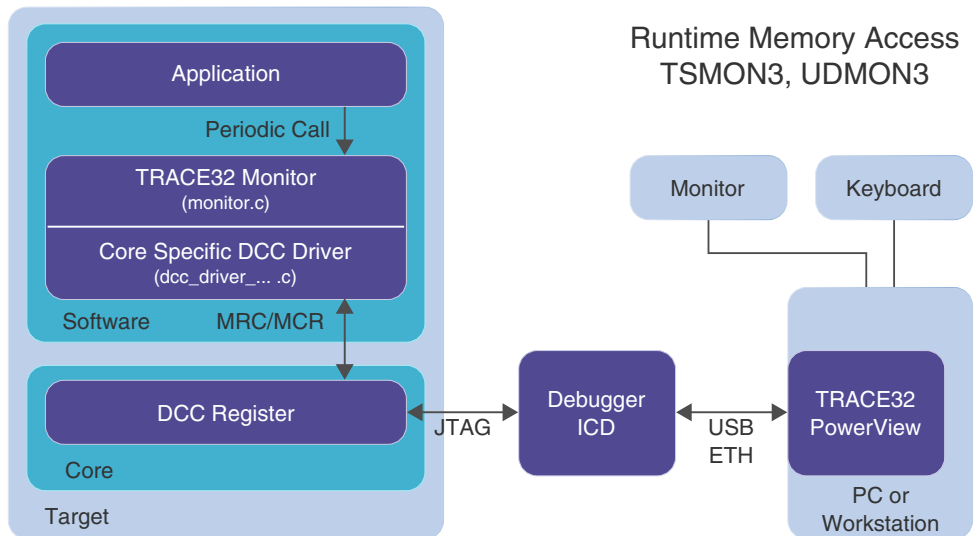
The memory access is done through the Nexus interface which is only available on MAC7xxx processors.

## TSMON3 TSMON

TSMON uses a data format which shall not be used anymore. It still works for compatibility reasons. TSMON3 shall be used.

A run-time memory access is done via a **Time Sharing Monitor**.

The application is responsible for calling the monitor code periodically. The call is typically included in a periodic interrupt or in the idle task of the kernel. See the example in the directory  
~~/demo/arm/etc/runtime\_memory\_access.



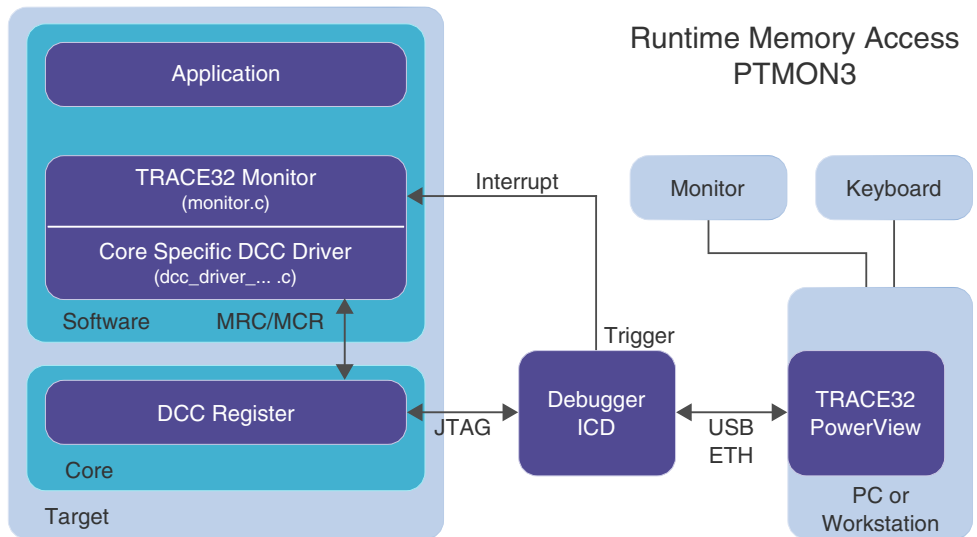
Besides runtime memory access TSMON3 would allow run mode debugging. But manual break is not possible with TSMON3 and could only be emulated by polling the DCC port. Therefore better use UDMON3 (or RealMON, TrkMON, GdbMON) for this purpose.

## PTMON3 PTMON

PTMON uses a data format which shall not be used anymore. It still works for compatibility reasons. PTMON3 shall be used.

A run-time memory access is done via a **Pulse Triggered Monitor**.

Whenever the debugger wants to perform a memory access while the program is running, the debugger generates a trigger for the trigger bus. If the trigger bus is configured appropriate (**TrBus**), this trigger is output via the TRIGGER connector of the TRACE32 development tool. The TRIGGER output can be connected to an external interrupt in order to call a monitor. See the example in the directory  
~~/demo/arm/etc/runtime\_memory\_access.



Besides runtime memory access PTMON3 would allow run mode debugging. But manual break is not possible with PTMON3 and could only be emulated by polling the DCC port. Therefore better use UDMON3 (or RealMON, TrkMON, GdbMON) for this purpose.

## QMON

Select QNX monitor (pdebug) for Run Mode Debugging of embedded QNX. Ethernet is used as communication interface. For more information, "[Run Mode Debugging Manual QNX](#)" (rtos\_qnx\_run.pdf).

## UDMON3 UDMON

UDMON uses a data format which shall not be used anymore. It still works for compatibility reasons. UDMON3 shall be used.

A run-time memory access is done via a **Usermode Debug Monitor**.

The application is responsible for calling the monitor code periodically. The call is typically included in a periodic interrupt or in the idle task of the kernel. For runtime memory access UDMON3 behaves exactly as TSMON3. See the example in the directory `~/demo/arm/etc/runtime_memory_access` and see the picture at TSMON3.

Besides runtime memory access UDMON3 allows run mode debugging. Handling of interrupts when the application is stopped is possible when the background monitor is activated. On-chip breakpoints and manual program break are only possible when the application runs in user (USR) mode. See also the example in the directory `~/demo/arm/etc/background_monitor`.

## RealMON

Run-time memory access and run mode debugging is done via the RealMonitor from ARM. The RealMonitor target software is supplied with ARM Firmware Suite.

## TrkMON

Select TRK for Run Mode Debugging of Symbian OS. DCC is used as communication interface.

## GdbMON

Select T32server (extended gdbserver) for Run Mode Debugging of embedded Linux. DCC is used as communication interface. For more information refer to [“Run Mode Debugging Manual Linux”](#) (`rtos_linux_run.pdf`).

## Denied

No memory access is possible while the CPU is executing the program.

If specific windows, that display memory or variables should be updated while the program is running select the memory class E: or the format option **%E**.

```
Data.dump E:0x100
```

```
Var.View %E first
```



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

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

Default: Down.

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

- Down** Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
- NoDebug** Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
- Prepare** Resets the target. This can be done via the reset line or CPU specific reset registers, see also [SYStem.Option RESetREGister](#). Afterwards direct access to the CoreSight DAP interface is provided. For a reset, the reset line has to be connected to the debug connector.

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

Typical use cases:

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

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

## Attach

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

## StandBy

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

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

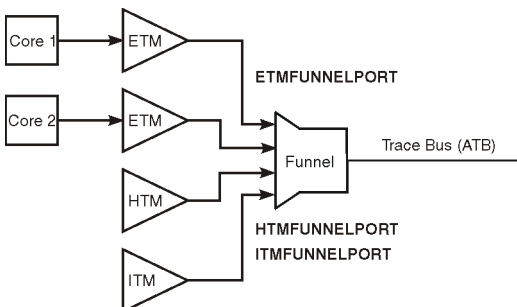
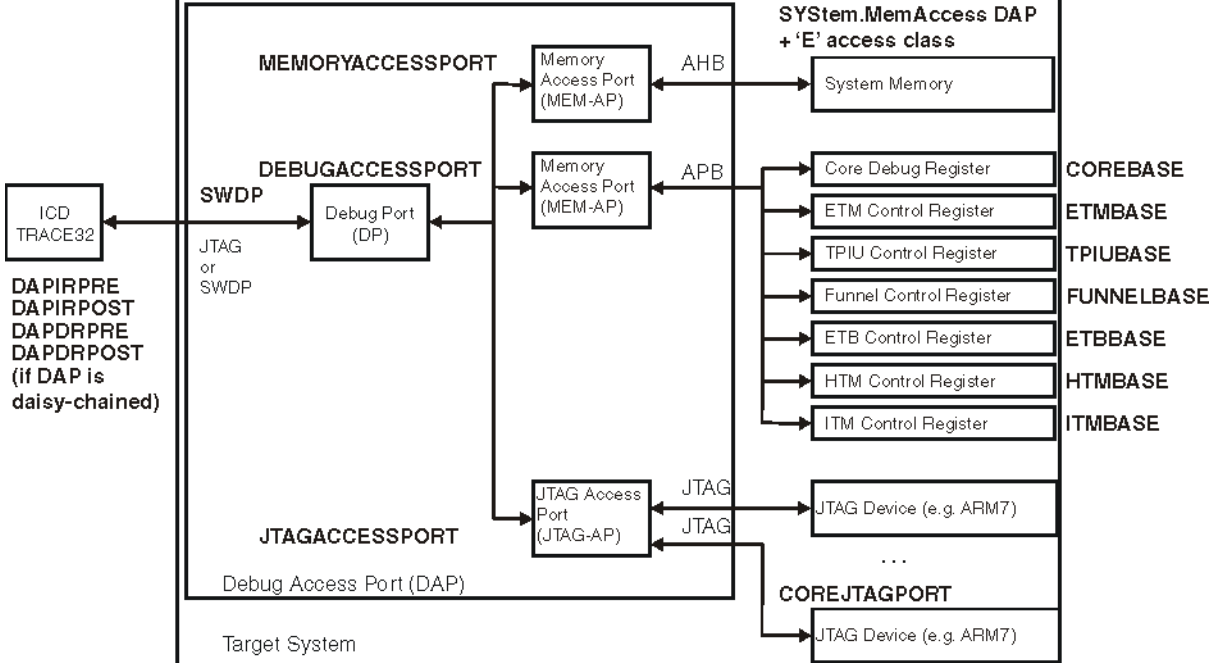
When a CPU power-down is detected, the debugger switches automatically back to the **StandBy** mode. This allows debugging of a power cycle because debug registers will be restored on power-up.

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

## Up

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

\_\_\_\_\_



The **SYStem.Option** commands are used to control special features of the debugger or emulator or to configure the target. It is recommended to execute the **SYStem.Option** commands **before** the emulation is activated by a **SYStem.Up** or **SYStem.Mode** command.

**SYStem.Option ABORTFIX** Do not access memory area from 0x0 to 0x1f

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

Default: OFF.

Workaround for a special customer configuration. It suppresses all debugger accesses to the memory area from 0x0 to 0x1f. This feature is only available on ARM7 family.

**SYStem.Option AHBHPROT** Select AHB-AP HPROT bits

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

Default: 0

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

**SYStem.Option AMBA** Select AMBA bus mode

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

This option is only necessary if a **ARM7 Bus Trace** is used.

Default: OFF.

This option should be set according to the bus mode of the ASIC.

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

This option is required for Cortex-A9, Cortex-A9MPCore r0p0, r0p1, r1p0, r1p1.

Default: OFF.

CPSR.T and CPSR.J bits can be corrupted on an asynchronous break. The fix causes the debugger to replace the asynchronous break by a synchronous break via breakpoint register. Breaks via external DBGRRQ signal e.g. from CTI still fail and may not be used.

**SYStem.Option AXIACEEnable**

ACE enable flag of the AXI-AP

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

Default: OFF.

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

**SYStem.Option AXICACHEFLAGS**

Select AXI-AP CACHE bits

Format:	<b>SYStem.Option AXICACHEFLAGS &lt;value&gt;</b>
<value>:	<b>DEVICENONSHAREABLE DEVICEINNERSHAREABLE DEVICEOUTERSHAREABLE DeviceSYStem NonCacheableNonShareable NonCacheableInnerShareable NonCacheableOuterShareable NonCacheableSYStem WriteThroughNonShareable WriteThroughInnerShareable WriteBackOuterShareable WRITETHROUGHSYSTEM</b>

Default: 0

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

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

SYStem.Option AXIHPROT

Select AXI-AP HPROT bits

Format:SYStem.Option AXIHPROT <value>

Default: 0

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

SYStem.Option BUGFIX

Breakpoint bug fix

Format:SYStem.Option BUGFIX [ON | OFF]

Default: OFF.

Breakpoint bug fix required on ARM7TDMI-S Rev2:

You need to activate this option when having an ARM7TDMI-S Rev2. The bug is fixed on Rev3 and following. With this option activated and ARM7TDMIS selected as CPU type, we enable the software breakpoint workaround as described in the ARM errata of ARM7TDMI-S Rev2 (“consecutive breakpoint” bug). Software breakpoints are set as undefined opcodes that cause the core to enter the undefined opcode handler. The debugger tries to set a breakpoint at the undef vector (either software or on-chip). When a breakpoint is reached the core will take the undefined exception and stop at the vector. The debugger detects this state and displays the correct registers and CPU state. This workaround is only suitable where undefined instruction trap handling is not being used.

Breakpoint bug fix required on ARM946E-S Rev0, Rev1 and ARM966E-S Rev0, Rev1:  
(This is a different bug fix as for the ARM7.) This option will automatically be activated by the TRACE32 software, since the core revision will be read out. On the above revisions the breakpoint code normally used for software breakpoints behave wrong. Having this option active an undefined opcode is used together with an on-chip comparator instead of the breakpoint code.

This option is available on ARM7 and on ARM9, but it has a different meaning.

## **SYStem.Option BUGFIXV4**      Asynch. break bug fix for ARM7TDMI-S REV4

Format: <b>SYStem.Option BUGFIXV4 [ON   OFF]</b>
--

Default: OFF.

This option is available on ARM7. You need to activate this option when having an ARM7TDMI-S Rev4.

With this option activated, we replace an asynchronous break, e.g. caused by the “break” command, by a break caused by an on-chip breakpoint range. If the bugfix is not activated when using an ARM7TDMI-S Rev4, the application might be restarted at a wrong address.

There is no known workaround to secure correct behavior of the external DBGRQ input and a program halt caused by an ETM trigger condition. Therefore do not use these features on an ARM7TDMI-S Rev4.

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

Default: OFF.

This option selects the byte ordering mechanism. For correct operation the following three settings must correspond:

- This option
- The compiler setting (-li or -bi compiler option)
- The level of the ARM BIGEND input pin (on ARM7x0T and ARM9x0T and JANUS2 the bit in the CP15 control register)

The endianness is auto-detected for the ARM10 and ARM11.

## SYStem.Option BOOTMODE

## Define boot mode

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

Default: 0.

This option selects a boot mode for the chip.

The command is only available on a few chips providing this feature.



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

Default: OFF.

If this option is ON the cache is invalidated after memory modifications even when memory is modified by the EPROM Simulator (ESI). This is necessary to maintain software breakpoint consistency.

## SYStem.Option CFLUSH

## FLUSH the cache before step/go

[SYStem.state window > CFLUSH]

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

Default: ON.

If this option is ON, the cache is invalidated automatically before each **Step** or **Go** command. This is necessary to maintain software breakpoint consistency.

## SYStem.Option CacheParam

## Define external cache

Only available for: ARM7

Format: **SYStem.Option CacheParam** <range> <size>

Define the <address\_range> and the <size> of an external cache.

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

Default: OFF.

Derivatives having a Domain Access Control Registers (DACR) do not allow the debugger to access memory if the location does not have the appropriate access permission. If this option is activated, the debugger temporarily modifies the access permission to get access to any memory location.

## SYStem.Option DAPDBGPWRUPREQ

## Force debug power in DAP

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

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.

<b>ON</b>	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.
<b>AlwaysON</b>	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.
<b>OFF</b>	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**.

This option is for target processors having a Debug Access Port (DAP) e.g., Cortex-A or Cortex-R.

## **SYStem.Option DAP2DBGPWRUPREQ**      Keep forcing debug power in DAP2

Format:	<b>SYStem.Option DAP2DBGPWRUPREQ [ON   AlwaysON]</b>
---------	--

Default: ON.

This option controls the DBGPWRUPREQ bit of the CTRL/STAT register of a second Debug Access Port (DAP2). Debug power will always be requested by the debugger on a debug session start. In case of ON this bit will be cleared at the end of the debug session, in case of AlwaysON this bit stays set.

This option is for target processors having a second Debug Access Port (DAP2).

## **SYStem.Option DAPSYSPWRUPREQ**      Force system power in DAP

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

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.

This option is for target processors having a Debug Access Port (DAP) e.g., Cortex-A or Cortex-R.

Format:	<b>SYStem.Option DAP2SYSPWRUPREQ</b> [AlwaysON   ON   OFF]
---------	--

Default: ON.

This option controls the SYSPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port 2 (DAP2) during and after the debug session

**AlwaysON**

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

**ON**

System power is requested by the debugger on a debug session start, and the control bit is set to 1.  
The system power is released at the end of the debug session, and the control bit is set to 0.

**OFF**

System power is **not** requested by the debugger on a debug session start, and the control bit is set to 0.

Format:	<b>SYStem.Option DAPNOIRCHECK</b> [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:	<b>SYStem.Option DAPREMAP</b> {<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

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

**SYStem.Option DBGACK**      DBGACK active on debugger memory accesses

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

Default: ON.

If this option is on the DBGACK signal remains active during memory accesses in debug mode. If the DBGACK signal is used to freeze timers or to disable other peripherals it is strictly recommended to enable this option.

Disabling of this option may be useful for triggering on memory accesses from debug mode (only useful for hardware developers).

This option is not available on the ARM10.

**SYStem.Option DBGNOPWRDWN**      DSCR bit 9 will be set in debug mode

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

Default: OFF.

If this option is on DSCR[9] will be set while the core is in debug mode and cleared while the user application is running. **SYStem.Option PWRDWN** will be ignored.

This option is normally not useful. It was implemented for a special customer design.

This option is available on the ARM11.

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

Default: ON.

This option allows the debugger to unlock the debug register by writing to the Operating System Lock Access Register (OSLAR) when a debug session will be started. If it is switched off the operating system is expected to unlock the register access, otherwise debugging is not possible.

This option is only available on the Cortex-R and Cortex-A.

## SYStem.Option DCDIRTY

## Bugfix for erroneously cleared dirty bits

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

Default: OFF.

This is a workaround for a chip bug which erroneously clears the dirty bits of a data cache line if there is any write-through forced by the debugger in this line. When the option is active the debugger does not use write-through mode in general. It only forces write through on a program memory write.

This option is only available on the ARM1176, Cortex-R, Cortex-A.

## SYStem.Option DCFREEZE

## Disable data cache linefill in debug mode

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

Default: ON.

This option disables the data cache linefill while the processor is in debug mode. This avoids that the data cache contents is altered on memory read accesses performed by the debugger. This is especially required if you want to inspect the data cache contents. You can disable this option if you want to cause a burst memory access (e.g. on a data.test command) which only occurs on a cache linefill.

This option is available on ARM11, only.

Format:	<b>SYStem.Option DEBUGPORTOptions</b> <option>
<option>:	<b>JTAGTOSWD</b> .[auto   CORTEX   LUMINARY   None] <b>SWDDORMANT</b> .[auto   WakeUp   NoWakeUp]

Default: auto.

This option is only required for expert users in case a non-standard SWD implementation is used.

**JTAGTOSWD** tells the debugger what to do when switching from JTAG to SWD mode:

<b>auto</b>	Auto detection. The debugger tries to detect the correct SWD to JTAG shift sequence.
<b>CORTEX</b>	Switching procedure like it should be done for Cortex cores.
<b>LUMINARY</b>	Switching procedure like it should be done for LuminaryMicro devices.
<b>None</b>	Do not send any SWD to JTAG sequence.

**SWDDORMANT** tells the debugger how to handle a dormant state when switching to SWD:

<b>auto</b>	Auto detection.
<b>WakeUp</b>	Wake up the debug port without prior checks by sending DORMANT to SWD sequence.
<b>NoWakeUp</b>	Do not wake up the debug port by suppressing the DORMANT to SWD sequence.

SYStem.Option DIAG

Activate more log messages

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

Default: OFF.

Adds more information to the report in the **SYStem.LOG.List** window.

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

*<option>*:  
**AUTO**  
**ACCESS**  
**ARM**  
**THUMB**  
**THUMBEE**

Default: AUTO.

This command specifies the selected disassembler.

<b>AUTO</b>	The information provided by the compiler output file is used for the disassembler selection. If no information is available it has the same behavior as the option <b>ACCESS</b> .
<b>ACCESS</b>	The selected disassembler depends on the T bit in the CPSR or on the selected access class. (e.g. <code>Data.List SR:0</code> for ARM mode or <code>Data.List ST:0</code> for THUMB mode).
<b>ARM</b>	Only the ARM disassembler is used (highest priority).
<b>THUMB</b>	Only the THUMB disassembler is used (highest priority).
<b>THUMBEE</b>	Only the THUMB disassembler is used which supports the Thumb-2 Execution Environment extension (highest priority).



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

This option is only available on XScale.

Default: OFF.

If this option is ON and a trap occurs the trap vector is read from memory and the trap vector is executed out of the memory.

The vector tables have been overloaded by the debugger to place the debug vector instead of the reset vector. If the application changes the vector during runtime the overloaded vector table in the mini instruction cache of the debugger remains active and a trap will jump to unintended position. With system option DynVector trap vector contents are read at runtime and the memory is executed. Executing an application with system option DynVector ON has disadvantage on runtime, so that it makes sense to switch off the option after the table has changed and afterwards remains unchanged. We have implemented this by an explicit option to be non intrusive on normal operation.

## 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 ETBFIXMarvell

## Read out on-chip trace data

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

Default: OFF

Bugfix for 88FR111 from Marvell. At least the first core revisions have an issue with the ETB read/write pointer. ON activates a different method to read out the on-chip trace data.

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

Default: OFF.

Bug fix for ETM7 implementations showing a wrong shift behavior. The ETM register data will be shifted by one bit otherwise. This feature is only available on the ARM7 family.

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

Default: OFF.

Bug fix for a customer device where ETM registers can not be read. This fix is only useful on this certain device.

Format: **SYStem.Option ETMFIx4 [ON | OFF]**

Default: OFF.

Bug fix for a customer device where each ETM data package was sent out four times.

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

Default: OFF.

Defines whether the EXEC line is available to the bustrace or not. The EXEC signal indicates if a fetched command has been executed. The bustrace can work without EXEC signal, but it is not possible to show the condition code pass/fail for conditional instructions. The option has no effect when no bustrace is available. This command has no meaning for the ETM trace.

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

Default: ON.

Bugfix for DB8500 V1. It allows you to switch off the fake TAP mechanism of the modem.

## SYStem.Option FASTBREAKDETECTION

## Fast core halt detection

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

Default: OFF.

It advises the debugger to do a permanent polling via JTAG to check if the core has halted. This allows a faster detection and generation of trigger signal for other tools like PowerIntegrator, especially if the hardware signal DBGACK is not available on the JTAG connector. It causes a high payload on the JTAG interface which will be a disadvantage e.g. if other debuggers use the same JTAG interface (multicore debugging).

This option is available on ARM9, only.

## SYStem.Option HRCWOVerRide

## Enable override mechanism

Format: **SYStem.Option HRCWOVerRide [ON | OFF] [/NONE | /PORESET]**

Default: OFF.

Enables the Hardcoded Reset Configuration Word override mechanism for NXP/Freescale Layerscape/QorIQ devices. The feature is required e.g. to program the flash in cases where the flash content is empty or corrupted.

In order to use this functionality, please contact Lauterbach for more details.

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

Default: OFF.

Bugfix for 88FR111 from Marvell. ON locks the usage of read-only/write-only on-chip breakpoints. They do not work on the 88FR111, at least not on the first core revisions.

SYStem.Option ICEPICK

Enable/disable assertions and wait-in-reset

Format:	<b>SYStem.Option ICEPICK</b> <option>
<option>:	<b>SystemReset</b> . [ON   OFF] <b>WaitInReset</b> . [ON   OFF]

Default: SystemReset.ON WaitInReset.ON may be preset with the correct parameters for known SoCs in TRACE32.

<b>SystemReset</b>	Enables/disables the assertions of <b>SystemReset</b> using the TI-ICEPick. <ul style="list-style-type: none"><li><b>ON</b>: Enables the assertion of <b>SystemReset</b>.</li><li><b>OFF</b>: Disables the assertion of <b>SystemReset</b>.</li></ul>
<b>WaitInReset</b>	Enables/disables the TI-ICEPick Wait-In-Reset functionality. This flag allows depending on the SoC implementation to hold a core on the reset vector. <ul style="list-style-type: none"><li><b>ON</b>: Enables the Wait-In-Reset.</li><li><b>OFF</b>: Disables the Wait-In-Reset.</li></ul>

SYStem.Option IMASKASM

Disable interrupts while single stepping

[SYStem.state window > IMASKASM]

Format:	<b>SYStem.Option IMASKASM</b> [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 a single step, the interrupt mask bits are restored to the value before the step.

## SYStem.Option IMASKHLL

## Disable interrupts while HLL single stepping

[\[SYStem.state window > IMASKHLL\]](#)

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

Default: OFF.

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

## SYStem.Option INTDIS

## Disable all interrupts

[\[SYStem.state window > INTDIS\]](#)

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

Default: OFF.

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

## SYStem.Option IRQBREAKFIX

## Break bugfix by using IRQ

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

The bug shows up on Cortex-A9, Cortex-A9MPCore r0p0, r0p1, r1p0, r1p1.

Default: 0 = OFF.

CPSR.T and CPSR.J bits can be corrupted on an asynchronous break. The bug fix is intended for an SMP multicore debug session where hardware based synchronous break is required. Instead causing an asynchronous break via CTI an IRQ is requested via CTI. There needs to be a breakpoint at the end of the IRQ routine handling this case. The fix causes the debugger to replace the program counter value by the IRQ link register R14\_irq - 4 and the CPSR register by SPSR\_irq if the core halts at <address>. Everything else like initializing the IRQ and CTI needs to be done by a user script.

Format: **SYStem.Option KEYCODE** <key>

Default: 0, means no key required.

Some processors have a security feature and require a key to unsecure the processor in order to allow debugging. The processor will use the specified key on the next debugger start-up (e.g. SYStem.Up) and forgets it immediately. For the next start-up the keycode must be specified again.

This option is available on ARM9.

## **SYStem.Option L2Cache**

L2 cache used

Format: **SYStem.Option L2Cache** [ON | OFF] (deprecated)  
**Use SYStem.CONFIG L2CACHE.Type instead.**

Default: OFF, means no L2 cache is used.

On certain Marvell derivatives the debugger can not detect if an (optional) level 2 cache is available and used. The information is needed to activate L2 cache coherency operations.

This option is available on Marvell ARM9, Cortex-A.

## **SYStem.Option L2CacheBase**

Define base address of L2 cache register

Format: **SYStem.Option L2CacheBase** <base\_address> (deprecated)  
**Use SYStem.CONFIG L2CACHE.Base instead.**

Default: 0, means no L2 cache implemented.

In case the L2 cache from ARM (L210, L220 and PL310) is available and active on the chip, then the debugger needs to flush and invalidate the L2 cache when patching the program e.g. when setting a software breakpoint. Therefore it needs to know the (physical) base address of the L2 register block.

This option is available on ARM9, ARM11, Cortex-R, Cortex-A.

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

This command is only available on obsolete ICD hardware. The state machine of the JTAG TAP controller is switched to Test-Logic Reset state (ON) or to Run-Test/Idle state (OFF) before a **SYStem.LOCK ON** is executed.

Format:	<b>SYStem.Option MACHINESPACES</b> [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.

### Machine IDs (0 and > 0)

- On ARM CPUs with hardware virtualization, guest machines are running in the non-secure zone (N:) and use machine IDs > 0.
- The hypervisor functionality is usually running in the hypervisor zone (H:) and uses machine ID 0.
- Software running in the secure monitor mode (Z: for ARM32) or EL3 mode (M: for ARM64) is also using machine ID 0.

Format:	<b>SYStem.Option MEMORYHPROT</b> <value>
---------	--

Default: 0

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



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

Default: OFF

Enables status flags check during a memory access. The debugger checks if the CPU is ready to receive/provide new data. Usually this is not needed. Only slow targets (like emulations systems) may need a status check.

## SYStem.Option MMUSPACES

## Separate address spaces by space IDs

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

Default: OFF.

Enables the use of [space IDs](#) for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of [address spaces](#) ([zone spaces](#), [MMU spaces](#), and [machine spaces](#)), see **“TRACE32 Glossary”** ([glossary.pdf](#)).

**NOTE:** **SYStem.Option MMUSPACES** should not be used if only one translation table is used on the target.

If a debug session requires space IDs, you must observe the following sequence of steps:

1. Activate **SYStem.Option MMUSPACES**.
2. Load the symbols with [Data.LOAD](#).

Otherwise, the internal symbol database of TRACE32 may become inconsistent.

## Examples:

```
;Dump logical address 0xC00208A belonging to memory space with  
;space ID 0x012A:  
Data.dump D: 0x012A:0xC00208A  
  
;Dump logical address 0xC00208A belonging to memory space with  
;space ID 0x0203:  
Data.dump D: 0x0203:0xC00208A
```

## SYStem.Option MonitorHoldoffTime

Delay between monitor accesses

Format: **SYStem.Option MonitorHoldoffTime** <time>

Default: 0.

It specifies the minimum delay between two access to the target debug client in case of run-mode debugging.

## SYStem.Option MPU

Debugger ignores MPU access permission settings

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

Default: OFF.

Derivatives having a memory protection unit do not allow the debugger to access memory if the location does not have the appropriate access permission. If this option is activated, the debugger temporarily modifies the access permission to get access to the memory location.

## SYStem.Option MultiplesFIX

No multiple loads/stores

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

Default: OFF.

Bug fix for derivatives (e.g. ARM946 V1.1) which do not handle multiple loads (LDM) and multiple store (STM) commands properly in debug mode. When activated only single loads/stores are used by the debugger.

## SYStem.Option NODATA

No data connected to the trace

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

This option is only necessary if a **Bus Trace** is used.

Default: OFF.

It should be ON, if a trace is connected and data information can not be recorded. Otherwise undefined data will be displayed in the trace records.

## SYStem.Option NOIRCHECK

No JTAG instruction register check

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

Default: OFF.

Bug fix for derivatives which do not return the correct pattern on a JTAG instruction register (IR) scan. When activated the returned pattern will not be checked by the debugger. On ARM7 also the check of the return pattern on a scan chain selection is disabled.

This option is only available on ARM7 and ARM9.

The option is automatically activated when using **SYStem.Option TURBO**.

## SYStem.Option NoPRCRReset

Do not cause reset by PRCR

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

Default: OFF.

It causes the debugger not to (additionally) use the soft reset via DBGPRCR register on functions like **SYStem.Up**, **SYStem.Mode Go**, **SYStem.RESetOut**.

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

Default: OFF.

If this option is ON, it advises the debugger not to do any running check. In this case the debugger does not even recognize that there will be no response from the processor. Therefore there always is the message “running”, independent of whether the core is in power down or not. This can be used to overcome power saving modes in case users know when a power saving mode happens and that they can manually deactivate and re-activate the running check.

**NOTE:** This command will affect the setting of **SYStem.POLLING** *<stopped\_mode>*.

## SYStem.Option NoSecureFix

## Do not switch to secure mode

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

Default: OFF.

This is a bugfix for customer specific devices which do not allow the debugger to temporarily switch to secure mode while the application is in non-secure mode.

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

Default: OFF.

- ON** Activates the overlay extension and extends the address scheme of the debugger with a 16 bit virtual overlay ID. Addresses therefore have the format `<overlay_id>:<address>`. This enables the debugger to handle overlaid program memory.
- OFF** Disables support for code overlays.
- WithOVS** Like option **ON**, but also enables support for software breakpoints. This means that TRACE32 writes software breakpoint opcodes both to the *execution area* (for active overlays) and to the *storage area*. In this way, it is possible to set breakpoints into inactive overlays. Upon activation of the overlay, the target's runtime mechanisms copies the breakpoint opcodes to the execution area. For using this option, the storage area must be readable and writable for the debugger.

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

## SYStem.Option PALLADIUM

## Extend debugger timeout

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

Default: OFF.

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

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

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

Default address: 0

After each load or store operation from debug mode the ARM core makes some instruction fetches from memory. These fetches are not necessary for the debugger, but it is not possible to suppress them.

This option allows to specify the base address of these fetches. The fetch address is anywhere within a 64 KByte block that begins at the specified base address. It is necessary to modify this option if these fetches go to aborted memory locations.

This option is not available/required on the ARM10 and ARM11. There are no dummy-fetches on ARM10 and ARM11.

## SYStem.Option PROTECTION

Sends an unsecure sequence to the core

Format: **SYStem.Option PROTECTION** <filename>

This option was made for certain ARM9 derivatives having a protected access to the debug features. It sends the key pattern in the file in a certain way to the core in order to gain the right to debug the core.

This option is available on ARM9.

## SYStem.Option PWRCHECK

Check power and clock

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

Default: ON.

In case of a chip level TAP (SYStem.CONFIG MULTITAP) this option decides if power, clock and secure state will be checked or not.

This option is only available on ARM11, Cortex-R, Cortex-A.

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

Default: OFF.

Fix for a certain chip bug: It uses the OSLK bit instead of the SPD bit of the PRSR register to detect power down.

This option is only available on Cortex-R, Cortex-A.

## SYStem.Option PWRDWN

Allow power-down mode

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

Default: OFF.

ARM11: If this option is OFF, the debugger sets the external signal **DBGNOPWRDWN** high in order to force the system power controller in emulate mode. Otherwise the communication to the debugger gets lost when entering power down state.

Some OMAPxxxx derivatives: If this option is OFF, the debugger forces the OMAP to keep clock and keep power.

Cortex-R, Cortex-A: Controls the PWRDWN bit in device power-down and reset control register (PRCR).

This option is only available on ARM11, Cortex-R, Cortex-A.

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

Default: OFF.

Assumes **SYStem.JtagClock RTCK** is selected.

When the target core is running and RTCK stops working for longer than specified by **SYStem.Option PWRDWNRecoverTimeout** it is assumed power is gone. In this case “running (power down)” will be shown. On power recovery the target logic ensures the core immediately enters debug mode by asserting DBGSRQ signal. The debugger detects the recovery, restores all debug register and restarts the program execution.

This option is only available on ARM9.

## **SYStem.Option PWRDWNRecoverTimeout**

## **Timeout for power recovery**

Format: **SYStem.Option PWRDWNRecoverTimeout** <time>

Specifies a timeout period as a limit to decide if just a sleep mode was entered (stopped RTCK) or a real power down happened which requires the debug registers to be restored on a power recovery. See command **SYStem.Option PWRDWNRecover**.

This option is only available on ARM9.

## **SYStem.Option PWROVR**

## **Specifies power override bit**

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

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

This option is only available on certain ARM9 and ARM11 derivatives.

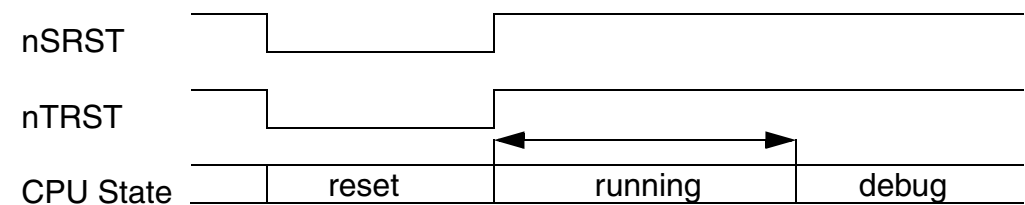


Format:

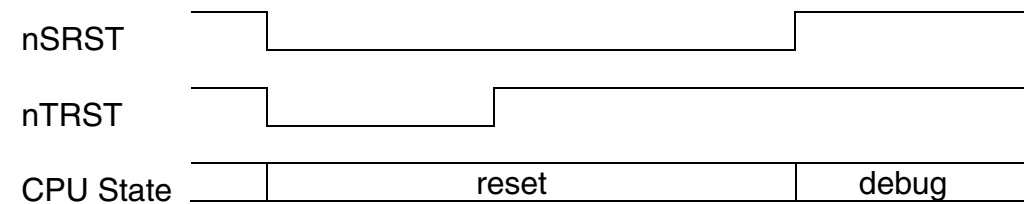
SYSystem.Option ResBreak [ON | OFF]

Default: ON.

This option has to be disabled if the nTRST line is connected to the nRESET / nSRST line on the target. In this case the CPU executes some cycles while the **SYSystem.Up** command is executed. The reason for this behavior is the fact that it is necessary to halt the core (enter debug mode) by a JTAG sequence. This sequence is only possible while nTRST is inactive. In the following figure the marked time between the deassertion of reset and the entry into debug mode is the time of this JTAG sequence plus a time delay selectable by **SYSystem.Option WaitReset** (default = 3 msec).



If nTRST is available and not connected to nRESET/nSRST it is possible to force the CPU directly after reset (without cycles) into debug mode. This is also possible by pulling nTRST fixed to VCC (inactive), but then there is the problem that it is normally not ensured that the JTAG port is reset in normal operation. If the ResBreak option is enabled the debugger first deasserts nTRST, then it executes a JTAG sequence to set the DBGRQ bit in the ICE breaker control register and then it deasserts nRESET/nSRST.



Format: **SYStem.Option ResetDetection** *<method>*

*<method>*: **nSRST | None**

Default: nSRST

Selects the method how an external target reset can be detected by the debugger.

<b>nSRST</b>	Detects a reset if nSRST (nRESET) line on the debug connector is pulled low.
<b>None</b>	Detection of external resets is disabled.

## SYStem.Option RESetREGister

## Generic software reset

Format: **SYStem.Option.RESetRegister NONE**  
**SYStem.Option.RESetRegister** *<address>*  
*<mask>*  
*<assert\_value>*  
*<deassert\_value>*  
*[/<width>]*

*<width>*: **Byte | Word | Long | Quad**

Specifies a register on the target side, which allows the debugger to assert a software reset, in case no nReset line is present on the JTAG header. The reset is asserted on **SYStem.Up**, **SYStem.Mode.Go**, **SYStem.Mode Prepare** and **SYStem.RESetOut**. The specified address needs to be accessible during runtime (for example E, DAP, AXI, AHB, APB).

<i>&lt;address&gt;</i>	Specifies the address of the target reset register.
<i>&lt;mask&gt;</i>	The <i>&lt;assert_value&gt;</i> and <i>&lt;deassert_value&gt;</i> are written in a read-modify-write operation. The mask specifies which bits are changed by the debugger. Bits of the mask value which are '1' are not changed inside the reset register.

<assert\_value> Value that is written to assert reset.

<deassert\_value> Value that is written to deassert reset.

**NOTE:** The debugger will not perform the default warm reset via the PRCR if this option is set.

## SYStem.Option RESTARTFIX

Wait after core restart

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

Default: OFF.

Bug fix for a certain customer derivative. When activated the debugger keeps the JTAG state machine on every restart for 10  $\mu$ s in Run-Test/Idle state before the JTAG communication will be continued. This option is available on ARM7 and will be ignored on other debuggers.

## SYStem.Option RisingTDO

Target outputs TDO on rising edge

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

Default: OFF.

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

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

Default: ON.

If the ABORT (if AMBA: BERROR) line becomes active during a system speed access the ARM core can change to ABORT mode. When this option is on this change of mode is indicated by the warning '**emulator berr error**'.

This option is not available on the ARM10 and ARM11 (always shown).

## SYStem.Option SOFTLONG

## Use 32-bit access to set breakpoint

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

Default: OFF.

Instructs the debugger to use 32-bit accesses to patch the software breakpoint code.

## SYStem.Option SOFTQUAD

## Use 64-bit access to set breakpoint

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

Default: OFF.

Activate this option if software breakpoints should be written by 64-bit accesses. This was implemented in order not to corrupt ECC.

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

Default: OFF.

Instructs the debugger to use 16-bit accesses to patch the software breakpoint code.

## **SYStem.Option SPLIT**

## **Access memory depending on CPSR**

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

Default: OFF.

If this option is ON, the debugger does privileged or non-privileged memory access depending on the current CPU mode (CPSR register). If this option is OFF, the debugger accesses the memory in privileged mode except another access mode is requested. This feature is only available if a DEBUG INTERFACE (LA-7701) is used for the ARM7.

## **SYStem.Option StandByTraceDelaytime**

## **Trace activation after reset**

Format: **SYStem.Option StandByTraceDelaytime <delay\_in\_us>**

Default: 0.

Only when standby mode is active you can specify a time delay where the debugger waits after reset is deasserted before it activates the trace. This option is available on ARM9 only.

## **SYStem.Option STEPSOFT**

## **Use software breakpoints for ASM stepping**

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

Default: OFF.

If set to ON, software breakpoints are used for single stepping on assembler level (advanced users only).

## **SYStem.Option SYSPWRUPREQ**

Force system power

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

Default: ON.

This option controls the SYSPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port (DAP). If the option is ON, system power will be requested by the debugger on a debug session start.

This option is for target processors having a Debug Access Port (DAP).

## **SYStem.Option TIDBGEN**

Activate initialization for TI derivatives

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

Default: OFF.

If this option is active the debugger sends a special initialization sequence, which is required for some derivatives from Texas Instruments (TI) to enable the on-chip debug support. When a TI CPU type (e.g. "OMAP1510") is selected, this option is automatically set.

This option is only available on ARM9.

## **SYStem.Option TIETMFX**

Bug fix for customer specific ASIC

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

## **SYStem.Option TIDEMUXFIX**

Bug fix for customer specific ASIC

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

Format: **SYStem.Option TraceStrobe** [CE | OE | CE+OE | STR | STR-] (deprecated)

## SYStem.Option TRST

## Allow debugger to drive TRST

[SYStem.state window > TRST]

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

Default: ON.

If this option is disabled, the nTRST line is never driven by the debugger (permanent high). Instead five consecutive TCK pulses with TMS high are asserted to reset the TAP controller which have the same effect.

## SYStem.Option TURBO

## Speed up memory access

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

Default: OFF.

If TURBO is disabled the CPU checks after each system speed memory access in debug mode if the CPU has finished the corresponding cycle. This check will significantly reduce the down- and upload speed (30-40%).

If TURBO is enabled the CPU will make no checks. This may result in unpredictable errors if the memory interface is slow. Therefore it is recommended to use this option only for a program download and in case you know that the memory interface is fast enough to take the data with the speed they are provided by the debugger.

This option is not available on the ARM10.

Format:SYSystem.Option WaitIDCODE [ON | OFF | <time>]

Default: OFF = disabled.

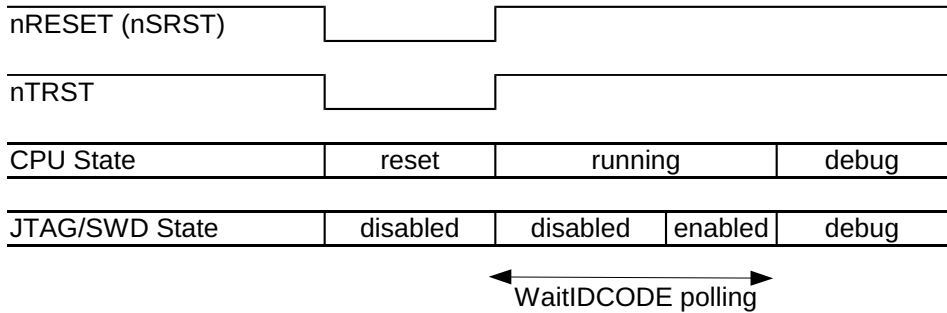
Allows to add additional busy time after reset. The command is limited to systems that use an ARM DAP.

If **SYSystem.Option WaitIDCODE** is enabled and **SYSystem.Option ResBreak** is disabled, the debugger starts to busy poll the JTAG/SWD IDCODE until it is readable. For systems where JTAG/SWD is disabled after RESET and e.g. enabled by the BootROM, this allows an automatic adjustment of the connection delay by busy polling the IDCODE.

After deasserting nSRST and nTRST the debugger waits the time configured by **SYSystem.Option WaitReset** till it starts to busy poll the JTAG/SWD IDCODE. As soon as the IDCODE is readable, the regular connection sequence continues.

ON	1 second busy polling
OFF	Disabled
<time>	Configurable polling time, max. 30 sec, use 'us', 'ms', 's' as units.

**Example:** The following figure shows a scenario with **SYSystem.Option ResBreak** disabled and **SYSystem.Option WaitIDCODE** enabled. The polling mechanism tries to minimize the delay between the JTAG/SWD disabled and debug state.



[\[SYSystem.state window > WaitReset\]](#)

Format:SYSystem.Option WaitReset [ON | OFF | <time>]

Default: OFF = 3 msec.

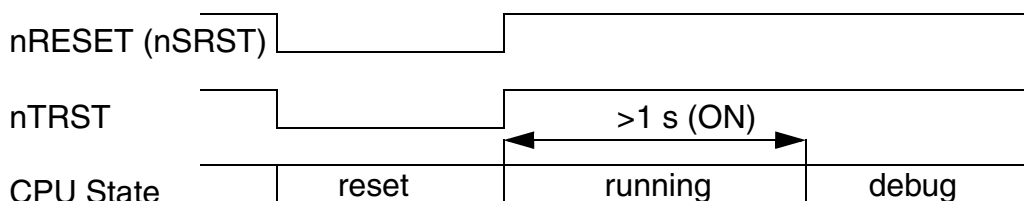


Allows to add additional wait time during reset.

<b>ON</b>	1 sec delay
<b>OFF</b>	3 msec delay
<b>&lt;time&gt;</b>	Selectable time delay, min. 50 usec, max. 30 sec, use 'us', 'ms', 's' as units.

If **SYStem.Option WaitReset** is enabled and **SYStem.Option ResBreak** is disabled, the debugger waits after the deassertion of nSRST and nTRST before the first JTAG activity starts (see picture below). It waits for at least 1 s, then it waits until nSRST is released from target side; the max. wait time is 35 s. During this time the core may execute some code, e.g. to enable the JTAG port.

If **SYStem.Option ResBreak** is enabled, the debugger waits the **<time>** specified with the command **SYStem.Option WaitReset**.



## SYStem.Option.WATCHDOG

Disable watchdog while debugging

Format: **SYStem.Option WATCHDOG [DEFAULT | OFF]**

Default: DEFAULT

Enables/disables the internal watchdog on some devices on connection time, e.g. during **SYStem.Up**. The option is available on some Spansion/Cypress S6J devices. Please refer to the example scripts if the option is available.

**DEFAULT** Does not modify the watchdog configuration.

**OFF** Disables the watchdog when connecting.

Format:	<b>SYSystem.Option ZoneSPACES</b> [ON   OFF]
---------	--

Default: OFF

The **SYSystem.Option ZoneSPACES** command is relevant if an ARM CPU with TrustZone or VirtualizationExtension is debugged. In these ARM CPUs, the processor has two or more CPU operation modes called:

- Non-secure mode
- Secure mode
- Hypervisor mode
- 64-bit EL3/Monitor mode (ARMv8-A only)

Within TRACE32, these CPU operation modes are referred to as [zones](#).

<b>NOTE:</b>	For an explanation of the TRACE32 concept of <a href="#">address spaces</a> ( <a href="#">zone spaces</a> , <a href="#">MMU spaces</a> , and <a href="#">machine spaces</a> ), see <b>“TRACE32 Glossary”</b> ( <a href="#">glossary.pdf</a> ).
--------------	--

In each CPU operation mode (zone), the CPU uses separate MMU translation tables for memory accesses and separate register sets. Consequently, in each zone, different code and data can be visible on the same logical addresses.

To ease debug-scenarios where the CPU operation mode switches between non-secure, secure or hypervisor mode, it is helpful to load symbol sets for each used zone.

<b>OFF</b>	TRACE32 does not separate symbols by access class. Loading two or more symbol sets with overlapping address ranges will result in unpredictable behavior. Loaded symbols are independent of ARM zones.
<b>ON</b>	Separate symbol sets can be loaded for each zone, even with overlapping address ranges. Loaded symbols are specific to one of the ARM zones - each symbol carries one of the access classes N:, Z:, H: or M: For details and examples, see <a href="#">below</a> .

If **SYStem.Option ZoneSPACES** is enabled (**ON**), TRACE32 enforces any memory address specified in a TRACE32 command to have an access class which clearly indicates to which zone the memory address belongs. The following access classes are supported:

<b>N</b>	Non-secure mode Example: Linux user application
<b>Z</b>	Secure mode Example: Secure crypto routine
<b>H</b>	Hypervisor mode Example: XEN hypervisor
<b>M</b> ARMv8-A only	64-bit EL3/Monitor mode Example: Trusted boot stage / monitor

If an address specified in a command is not clearly attributed to N: Z:, H: or M:, the access class of the current PC context is used to complete the addresses' access class.

Every loaded symbol is attributed to either non-secure (N:), secure (Z:), hypervisor (H:) or EL3/monitor (M:) zone. If a symbol is referenced by name, the associated access class (N:, Z:, H: or M:) will be used automatically, so that the memory access is done within the correct CPU mode context. As a result, the symbol's logical address will be translated to the physical address with the correct MMU translation table.

**NOTE:**

The loaded symbols and their associated access class can be examined with command **sYmbol.List** or **sYmbol.Browse** or **sYmbol.INFO**.

## Example: Symbols Loading

---

```
SYStem.Option ZONESPACES ON

; 1. Load the vmlinux symbols for non-secure mode (access classes N:, NP:
; and ND: are used for the symbols) with offset 0x0:
Data.LOAD.Elf vmlinux N:0x0 /NoCODE

; 2. Load the sysmon symbols for secure mode (access classes Z:, ZP: and
; ZD: are used for the symbols) with offset 0x0:
Data.LOAD.Elf sysmon Z:0x0 /NoCODE

; 3. Load the xen-syms symbols for hypervisor mode (access classes H:,
; HP: and HD: are used for the symbols) but without offset:
Data.LOAD.Elf xen-syms H: /NoCODE

; 4. Load the sieve symbols without specification of a target access
; class and address:
Data.LOAD.Elf sieve /NoCODE
; Assuming that the current CPU mode is non-secure in this example, the
; symbols of sieve will be assigned the access classes N:, NP: and ND:
; during loading.
```

## Example: Symbolic Memory Access

---

```
; dump the address on symbol swapper_pg_dir which belongs
; to the non-secure symbol set "vmlinux" we have loaded above:

Data.dump swapper_pg_dir

; This will automatically use access class N: for the memory access,
; even if the CPU is currently not in non-secure mode.
```

## Example: Deleting Zone-specific Symbols

---

To delete a complete symbol set belonging to a specific zone, e.g. the non-secure zone, use the following command to delete all symbols in the specified address range.

```
sYmbol.Delete N:0x0--0xffffffff ; non-secure mode (access classes N:)
```

## Example: Zone-specific Debugger Address Translation Setup

If the option **ZoneSPACES** is enabled and the debugger address translation is used (**TRANSlation** commands), a strict zone separation of the address translations is enforced. Also, common address ranges created with **TRANSlation.COMMON** will always be specific for a certain zone.

This script shows how to define separate translations for the zones **N:** and **H:**

```
SYStem.Option ZoneSPACES ON

Data.LOAD.Elf sysmon    Z:0 /NoCODE
Data.LOAD.Elf usermode N:0 /NoCODE /NoClear

; set up address translation for secure mode
TRANSlation.Create Z:0xC0000000++0xffffffff A:0x10000000

; set up address translation for non-secure mode
TRANSlation.Create N:0xC0000000++0x1ffffffff A:0x40000000

; enable address translation and table walk
TRANSlation.ON

; check the complete translation setup
TRANSlation.List
```

If the CPU's virtualization extension is used to virtualize one or more guest systems, the hypervisor always runs in the CPU's hypervisor mode (zone H:), and the current guest system (if a ready-to-run guest is configured at all by the hypervisor) will run in the CPU's non-secure mode (zone N:).

Often, an operation system (such as a Linux kernel) runs in the context of the guest system.

In such a setup with hypervisor and guest OS, it is possible to load both the hypervisor symbols to H: and all OS-related symbols to N:

A TRACE32 OS Awareness can be loaded in TRACE32 to support the work with the OS in the guest system. This is done as follows:

1. Configure the OS Awareness as for a non-virtualized system. See:
  - [“Training Linux Debugging”](#) (training\_rtos\_linux.pdf)
  - **TASK.CONFIG** command
2. Additionally set the default access class of the OS Awareness to the non-secure zone:

```
TASK.ACCESS N:
```

The TRACE32 OS Awareness is now configured to find guest OS kernel symbols in the **non-secure** zone.

**NOTE:**

This debugger setup, which is based on the option **ZoneSPACES**, allows work with only one guest system simultaneously.

If the hypervisor has configured more than one guest, only the guest that is active in the non-secure CPU mode is visible.

To work with another guest, the system must continue running until an inactive guest becomes the active guest.

With **SYStem.Option.MACHINESPACES** enabled, TRACE32 also supports concurrent debugging of a virtualized system with hypervisor and multiple guests.

the CPU specific zones N: Z: H: and M: will be extended by machine specific zones. Each of these zones is identified by a [machine ID](#). Each guest has its own zone because it uses a separate translation table and a separate register set.

## Example: Setup for a Guest OS and a Hypervisor

In this script, the hypervisor is configured to run in zone **H:** and a Linux kernel with OS Awareness as current guest OS in zone **N:**

```
SYStem.Option ZoneSPACES ON

; within the OS Awareness we need the space ID to separate address spaces
; of different processes / tasks
SYStem.Option MMUSPACES ON

; here we let the target system boot the hypervisor. The hypervisor will
; set up the guest and boot Linux on the guest system.
...

; load the hypervisor symbols
Data.LOAD.Elf xen-syms H:0 /NoCODE
Data.LOAD.Elf usermode N:0 /NoCODE /NoClear

; set up the Linux OS Awareness
TASK.CONFIG ~/demo/arm/kernel/linux/linux-3.x/linux3.t32
MENU.ReProgram ~/demo/arm/kernel/linux/linux-3.x/linux.men

; instruct the OS Awareness to access all OS-related symbols with
; access class N:
TASK.ACCESS N:

; set up the debugger address translation for the guest OS

; Note that the default address translation in the following command
; defines a translation of the logical kernel addresses range
; N:0xC0000000++0xFFFFFFFF to the intermediate physical address range
; starting at I:0x40000000
MMU.FORMAT linux swapper_pg_dir N:0xC0000000++0xFFFFFFFF I:0x40000000

; define the common address range for the guest kernel symbols
TRANSLation.COMMON N:0xC0000000--0xFFFFFFFF

; enable the address translation and the table walk
TRANSLation.TableWalk ON
TRANSLation.ON
```

### NOTE:

If **SYStem.Option MMUSPACES ON** is used, all addresses for all zones will show a **space ID** (such as **N:0x024A:0x00320100**), even if the OS Awareness runs only in one zone (as defined with command **TASK.ACCESS**).

Any task-related command, such as **MMU.List.TaskPageTable** <taskname>, will automatically refer to tasks running in the same zone as the OS Awareness.

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

Default: OFF

This option is for a Zynq Ultrascale+ device using JTAG Boot mode. There are two cases:

1. Device operates in cascaded mode. The ARM DAP and TAP controllers both use the PL JTAG interface, i.e. forming a JTAG daisy chain.
2. Device operates in independent mode. The TAP controller is accessed via the PL JTAG interface. The ARM DAP is connected to the MIO or EMIO JTAG interface.

This command controls whether the debugger connects to the device in independent or cascaded mode. This depends on the used JTAG interface.

<b>ON</b>	<p>The ARM DAP is accessed through the MIO or EMIO JTAG interface. No JTAG chain configuration is required by the debugger.</p> <p><b>NOTE:</b> Please set this option to <b>ON</b> if JTAG is connected via the independent JTAG (e.g. via MIO or EMIO via FPGA) lines.</p>
<b>OFF</b>	<p>The ARM DAP is accessed through the PL JTAG interface and has to be chained with the TAP controller by the debugger.</p>

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

If possible (nRESET/nSRST is open collector), this command asserts the nRESET/nSRST line on the JTAG connector. While the CPU is in debug mode, this function will be ignored. Use the **SYStem.Up** command if you want to reset the CPU in debug mode.



Format: **SYStem.state**

Displays the **SYStem.state** window for ARM.

# ARM Specific Benchmarking Commands

The **BMC (BenchMark Counter)** commands provide control of the on-chip performance monitor unit (PMU). The PMU consists of a group of counters that can be configured to count certain events in order to get statistics on the operation of the processor and the memory system.

The counters of Cortex-A/R cores can be read at run-time. The counters of ARM11 cores can only be read while the target application is halted. This group of counters is not available for ARM7 to ARM10 cores.

For information about *architecture-independent* **BMC** commands, refer to “**BMC**” (general\_ref\_b.pdf).

For information about *architecture-specific* **BMC** commands, see command descriptions below.

## BMC.EXPORT

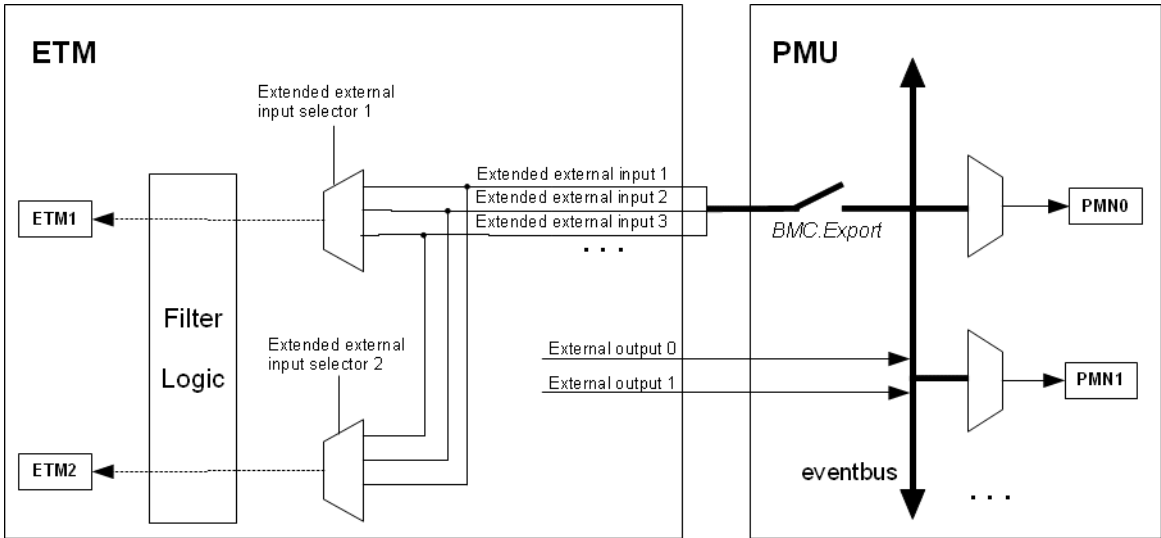
## Export benchmarking events from event bus

Format: **BMC.EXPORT [ON | OFF]**

Enable / disable the export of the benchmarking events from the event bus. If enabled, it allows an external monitoring tool, such as an ETM to trace the events. For further information please refer to the target processor manual under the topic performance monitoring.

Default: OFF

The figure below depicts an example configuration comprising the PMU and ETM:



In case ETM1 or ETM2 are selected for event counting, **BMC.EXPORT** will automatically be switched on. Furthermore the according extended external input selectors of the ETM will be set accordingly.

Format: **BMC.MODE** *<mode>*

*<mode>*:  
**OFF**  
**ICACHE**  
**DCACHE**  
**SYSIF**  
**CLOCK**  
**TIME**

This command only applies to some ARM9 based derivatives from Texas Instruments.

The Benchmark Counter - short BMC - is a hardware counter. It collects information about the throughput of the target processor, like instruction or data cache misses. This information may be helpful in finding bottlenecks and tuning the application.

<b>OFF</b>	Switch off the benchmark counter.
<b>ICACHE</b>	Counts Instructions CACHE misses, in relation to total instruction access.
<b>DCACHE</b>	Counts Data CACHE misses, in relation to total data access.
<b>SYSIF</b>	Counts if SYStem bus InterFace is busy, in relation to total system bus access.
<b>CLOCK</b>	Incremented for each CPU clock.
<b>TIME</b>	TIME is measured by counting CLOCK. The translation to TIME is done by using the CPU frequency. For this reason, the CPU frequency has to be entered with the command <b>BMC.CLOCK</b> .

Format:	<b>BMC.&lt;counter&gt;.EVENT &lt;event&gt;</b>
<counter>:	<b>PMN0 PMN1</b>
<event>:	<b>OFF   INST   BINST   BMIS   PC   ICMISS   ITLBMISS   ISTALL   DACCESS   DCACHE   DCMISS   DTBLMISS   DSTALL   DFULL   DCWB   WBDRAIN   TLBMISS   EMEM   ETMEXTOUT0   ETMEXTOUT1   Delta   Echo   CLOCK   TIME   NONE   ...</b>

The command is available on ARM1136, ARM1176 and Cortex cores. This description applies to ARM1136. All available events are described in detail in the technical reference guide of the ARM cores.

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

<event>	For a description of the <events>, refer to the <i>Technical Reference Manual</i> (TRM) of the respective core, chapter “ <i>Performance Monitor Unit</i> ” (PMU).  For a description of some selected <events>, see below.
OFF	Switch off the performance monitor.
INST	The selected counter counts executed instructions.
BINST	Counts executed branch instructions.
BMIS	Counts branches which were mispredicted by the core (for static) or prefetch unit (for dynamic) branch prediction. A branch misprediction causes the pipeline to be flushed, and the correct instruction to be fetched.
PC	Counts changes of the PC by the program e.g. as in a MOV or LDR instruction with PC as destination.
ICMISS	Counts instruction cache misses which requires a instruction fetch from the external memory.
ITLBMISS	Counts misses of the instruction MicroTLB.

<b>ISTALL</b>	ISTALL increments the counter by 1 for every cycle the condition is valid. The CPU is stalled when the instruction buffer cannot deliver an instruction. This happens as a result of an instruction cache miss or an instruction MicroTLB miss.
<b>DACCESS</b>	DACCESS is incremented by 1 for every nonsequential data access, regardless of whether or not the item is cached or not.
<b>DCACHE</b>	DCACHE is incremented for each access to the data cache.
<b>DCMISS</b>	DCMISS counts for missing data in the data cache.
<b>DTBLMISS</b>	Counts misses in the data MicroTLB.
<b>DSTALL</b>	In a data dependency conflict the CPU is stalled. DSTALL increments the counter by one for every cycle the stall persists.
<b>DFULL</b>	If the pipeline of load store unit is full, the counter will be incremented by one for each clock the condition is met.
<b>DCWB</b>	Data cache write back occurs for each half line of four words that are written back from cache to memory.
<b>WBDRAIN</b>	Write buffer drains force all buffered data writes to be written to external memory. WBDRAIN will count all that drains which are done because of a data synchronization barrier or strongly ordered operations.
<b>TBLMISS</b>	Counts main TLB misses.
<b>EMEM</b>	Incremented for each explicit external data access. That includes cache refills, non-cashable and write-through access. It does not include instruction cache fills or data write backs.
<b>ETMEXTOUT0</b>	The counter is incremented, if the ETMEXTOUT0-signal is asserted for a cycle. The ETM can be programmed to rise that signal on behalf / as result of certain events, like a counter overflow or an address compare.
<b>EMTEXTOUT1</b>	The counter is incremented, if the ETMEXTOUT1-signal is asserted for a cycle. The ETM can be programmed to rise that signal on behalf of certain events, like a counter overflow or an address compare.
<b>Delta</b>	Counts hits of the Delta-Marker, if specified.
<b>Echo</b>	Counts hits of the Echo-Marker, if specified.
<b>CLOCK</b>	The counter is incremented for every cpu clock.

<b>TIME</b>	TIME is measured by counting CLOCK. The transaction to TIME is done by using the cpu frequency. For this reason, the CPU frequency has to be entered with the command <b>BMC.CLOCK</b> .
<b>INIT</b>	Reset the benchmark counter to zero.

**Example 1:** To count for branches taken, in relation to mispredicted branches, use the following commands:

BMC.RESet	; Reset the BMC settings
BMC.state	; Display the BMC window
BMC.PMN0.EVENT BINST	; Set the first (PMN0) performance counter ; to count all taken branches
BMC.PMN1.EVENT BMIS	; Set the second (PMN1) performance counter ; to mispredicted branches
BMC.PMN0.RATIO PMN1/PMN0	; Calculate the ratio between branches ; taken and branches mispredicted
Go sieve	; Go to the function sieve
BMC.Init	; Initialize the benchmark counter to start ; the measurement of function sieve
Go.Return	; Go to the last instruction of the function ; sieve

**Example 2:** To count for data access in relation to data cache misses:

BMC.RESet	; Reset the BMC settings
BMC.state	; Display the BMC window
BMC.PMN0.EVENT DCACCESS	; Set the first (PMN0) performance counter ; to count all data accesses
BMC.PMN1.EVENT DCMISS	; Set the second (PMN1) performance counter ; to count data cache misses
BMC.PMN0.RATIO PMN1/PMN0	; Calculate the ratio between data access ; and cache misses
Go sieve	; Go to the function sieve
BMC.Init	; Initialize the benchmark counter
Go.Return	; Go to the last instruction of the function ; sieve

Benchmark counter values can be returned with the function **BMC.COUNTER()**.

Format:	<b>BMC.PRESCALER [ON   OFF]</b>
---------	---------------------------------

If ON, the cycle counter register, which counts for the cpu cycles which is used to measure the elapsed time, will be divided (prescaled) by 64. The display of the time will be corrected accordingly.

BMC.<counter>.RATIO

Set two counters in relation

Format:	<b>BMC.&lt;counter&gt;.RATIO &lt;ratio&gt;</b>
<counter>:	<b>PMN0 PMN1</b>
<ratio>:	<b>OFF PMN0/PMN1 PMN1/PMN0 PMN0/PMNC PMN1/PMNC</b>

It might be useful to set two counter values in relation to each other, e.g. data cache accesses (DCACCESS) and data cache misses (DCMISS).

<b>PMN0/PMN1</b>	Calculate the ratio PMN0/PMN1.
<b>PMN1/PMN0</b>	Calculate the ratio PMN1/PMN0.
<b>PMN0/PMNC</b>	Calculate the ratio PMN0/PMNC.
<b>PMN1/PMNC</b>	Calculate the ratio PMN1/PMNC.

For an example, see [BMC.<counter>.EVENT](#).



Format:	<b>BMC.TARA</b>
---------	-----------------

Due to restricted technical feasibility, the benchmark counter will start counting before the application runs. To improve the exactness of the result you can perform **BMC.Init**, single step an assembler command and execute **BMC.TARA**. On following measurements the obtained result will be subtracted from the benchmark counter.

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

TrOnchip.A

Programming the ICE breaker module

Available for ARM7 and ARM9 family.

TrOnchip.A.Value

Define data selector

Format:

TrOnchip.A.Value

TrOnchip.B.Value

<hexmask> | <bitmask>

<hexmask> | <bitmask>

Defines the two data selectors of ICE breaker as hex or binary mask (x means don't care). If you want to trigger on a certain byte or word access you must specify the mask according to the address of the access. E.g. you make a byte access on address 2 and you want to trigger on the value 33, then the necessary mask is 0xx33xxx.

Available for ARM7 and ARM9 family.

TrOnchip.A.Size

Define access size for data selector

Format:

TrOnchip.A.Size

TrOnchip.B.Size

<size>

<size>

<size>:

OFF

Byte

Word

Long

Defines on which access size when ICE breaker stops the program execution.

Available for ARM7 and ARM9 family.

Format:           **TrOnchip.A.CYcle** *<cycle>*  
                  **TrOnchip.B.CYcle** *<cycle>*

*<cycle>*:           **OFF**  
                  **Read**  
                  **Write**  
                  **Access**  
                  **Execute**

Defines on which cycle the ICE breaker stops the program execution.

<b>OFF</b>	Cycle type does not matter.
<b>Read</b>	Stop the program execution on a read access.
<b>Write</b>	Stop the program execution on a write access.
<b>Access</b>	Stop the program execution on a read or write access.
<b>Execute</b>	Stop the program execution on an instruction is executed.

Available for ARM7 and ARM9 family.

Format:           **TrOnchip.A.Address** <selector>  
                  **TrOnchip.B.Address** <selector>

<selector>:       **OFF**  
                  **Alpha**  
                  **Beta**  
                  **Charly**

The address/range for an address selector can not be defined directly. Set an breakpoint of the type Alpha, Beta or Charly to the address/range.

#### Example 1:

```
Break.Set 1000 /Alpha           ; set an Alpha breakpoint to 1000
TrOnchip.A.Address Alpha       ; use Alpha breakpoint as address
                               ; selector for the unit A
```

#### Example 2:

```
Var.Break.Set flags[3] /Beta   ; set a Beta breakpoint to flags[3]
TrOnchip.B.Address Beta       ; use Beta breakpoint as address
                               ; selector for the unit B
```

Available for ARM7 and ARM9 family.

Format:           **TrOnchip.A.Trans** *<mode>*  
                   **TrOnchip.B.Trans** *<mode>*

*<mode>*:       **OFF**  
                   **User**  
                   **Svc**

Defines in which mode ICE breaker should stop the program execution.

- OFF**                      Mode doesn't matter.
- User**                    Stop the program execution only in user mode.
- Svc**                     Stop the program execution only in supervisor mode.

Available for ARM7 and ARM9 family.

## TrOnchip.A.Extern

## Define the use of EXTERN lines

Format:           **TrOnchip.A.Extern** *<mode>*  
                   **TrOnchip.B.Extern** *<mode>*

*<mode>*:       **OFF**  
                   **Low**  
                   **High**

Defines if the EXTERN lines are considered by unit A or unit B.

Available for ARM7 and ARM9 family.

Format:

TrOnchip.AddressMask <value> | <bitmask>

Format:

TrOnchip.MatchASID [ON | OFF]  
TrOnchip.ASID [ON | OFF] (deprecated)

OFF (default)	Stop the program execution at on-chip breakpoint if the address matches. Trace filters and triggers become active if the address matches.
ON	Stop the program execution at on-chip breakpoint if both the address and the ASID match. Trace filters and triggers become active if both the address and the ASID match.

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

<b>OFF</b>	Stop the program execution at on-chip breakpoint if the address matches. Trace filters and triggers become active if the address matches.
<b>ON</b> (default)	Stop the program execution at on-chip breakpoint if both the address and the <a href="#">zone</a> match. Trace filters and triggers become active if both the address and the zone match.

**NOTE:** **SYStem.Option ZoneSPACES** must be set to **ON** for **TrOnchip.MatchZone ON** to take effect.

However, the setting **TrOnchip.MatchZone ON** is *not* supported by all ARM cores nor by all ETMs.

**Example:** In these two demo code snippets, let's compare the setting **TrOnchip.MatchZone ON** and **OFF** for an on-chip breakpoint at address 0x100 in zone Z (= secure memory).

```
SYStem.Option ZoneSPACES ON
```

```
;create an on-chip breakpoint in secure memory
Break.Set ZSR:0x100 /Onchip
```

```
TrOnchip.MatchZone ON ;observe the zones for on-chip breakpoints
```

```
;--> application execution will stop at the on-chip breakpoint
; only if both conditions are fulfilled:
; a) the address is 0x100 and
; b) the zone is Z (= secure memory)
```

```
SYStem.Option ZoneSPACES ON
```

```
;create an on-chip breakpoint in secure memory
Break.Set ZSR:0x100 /Onchip
```

```
TrOnchip.MatchZone OFF ;ignore the zones for on-chip breakpoints
```

```
;--> now application execution will stop at address 0x100
; irrespective of the zone
```

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

If the debug unit provides breakpoint registers with ContextID comparison capability, **TrOnchip.ContextID** has to be set to ON in order to set task/process specific breakpoints that work in real-time.

```
TrOnchip.ContextID ON
```

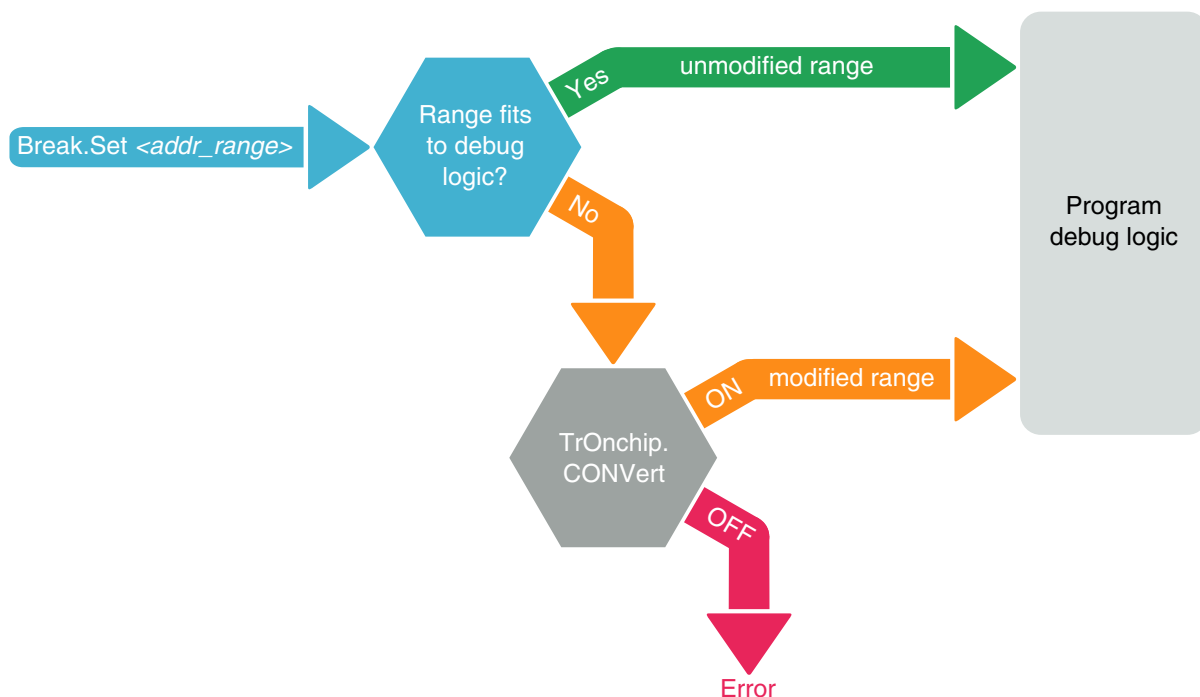
```
Break.Set VectorSwi /Program /Onchip /TASK EKern.exe:Thread1
```

## TrOnchip.CONVert

Allow extension of address range of breakpoint

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

Controls for all on-chip read/write breakpoints whether the debugger is allowed to change the user-defined address range of a breakpoint (see **Break.Set** <addr\_range> in the figure below).



The debug logic of a processor may be implemented in one of the following three ways:



1. The debug logic does not allow to set range breakpoints, but only single address breakpoints. Consequently the debugger cannot set range breakpoints and returns an error message.
2. The debugger can set any user-defined range breakpoint because the debug logic accepts this range breakpoint.
3. The debug logic accepts only certain range breakpoints. The debugger calculates the range that comes closest to the user-defined breakpoint range (see “modified range” in the figure above).

The **TrOnchip.CONVERT** command covers case 3. For case 3) the user may decide whether the debugger is allowed to change the user-defined address range of a breakpoint or not by setting **TrOnchip.CONVERT** to **ON** or **OFF**.

<b>ON</b> (default)	If <b>TrOnchip.Convert</b> is set to <b>ON</b> and a breakpoint is set to a range which cannot be exactly implemented, this range is automatically extended to the next possible range. In most cases, the breakpoint now marks a wider address range (see “modified range” in the figure above).
<b>OFF</b>	If <b>TrOnchip.Convert</b> is set to <b>OFF</b> , the debugger will only accept breakpoints which exactly fit to the debug logic (see “unmodified range” in the figure above). If the user enters an address range that does not fit to the debug logic, an error will be returned by the debugger.

In the **Break.List** window, you can view the requested address range for all breakpoints, whereas in the **Break.List /Onchip** window you can view the actual address range used for the on-chip breakpoints.

## TrOnchip.Mode

## Configure unit A and B

Format:	<b>TrOnchip.Mode</b> <mode>
<mode>:	<b>AORB</b> <b>AANDB</b> <b>BAFTERA</b> <b>WATCH</b>

Defines the way in which unit A and B are used together. See **TrOnchip.A**.

<b>AORB</b>	Stop the program execution if unit A or unit B match.
<b>AANDB</b>	Stop the program execution if both units match.
<b>BAFTERA</b>	Stop the program execution if first unit A and then unit B match.
<b>WATCH</b>	Cause assertion of the internal watchpoint signal on a match.

Available for ARM7 and ARM9 family.

Format:	<b>TrOnchip.RESet</b>
---------	-----------------------

Resets all TrOnchip settings.

TrOnchip.Set

Set bits in the vector catch register

Format:	<b>TrOnchip.Set StepVector [ON   OFF]</b>  ARM9, ARM11 also: <b>[FIQ   IRQ   DABORT   PABORT   SWI   UNDEF   RESET]</b>  Devices having TrustZone (ARM1176, Cortex-A) additionally: <b>[NFIQ   NIRQ   NDABORT   NPABORT   NSWI   NUNDEF   SFIQ   SIRQ   SDABORT   SPABORT   SSWI   SUNDEF   SRESET   MAFIC   MIRQ   MDABORT   MPABORT   MSWI]</b>  Devices having a Hypervisor mode (e.g. Cortex-A7, -A15) additionally: <b>[HFIQ   HIRQ   HDABORT   HPABORT   HSWI   HUNDEF   HENTRY]</b>
---------	---

Default: DABORT, PABORT, UNDEF, RESET ON, others OFF.

On devices having TrustZone you can specify for most exceptions if the vector catch shall take effect only in non-secure (N...), secure (S...) or monitor mode (M...), on devices having a Hypervisor mode also in hypervisor mode (H...).

If StepVector is activated a breakpoint range will be set on the trap vector table (e.g. 0x00--0x1f) when a single step is requested. This is helpful to check if a interrupt or trap occurs.

<b>FIQ, ... HENTRY</b>	Sets/resets the corresponding bits in the vector catch register of the core. If the bit of a vector is set and the corresponding exception occurs, the processor enters debug state as if there had been a breakpoint set on an instruction fetch from that exception vector.
----------------------------	---

Format: **TrOnchip.TEnable** *<mode>*

*<mode>*:  
**ALL**  
**Alpha**  
**Beta**  
**Charly**  
**Delta**  
**Echo**

Defines a filter for the trace. The Preprocessor for the ARM7 family (bus trace) provides 1 address comparator, that is implemented as a comparator (bit mask). Since this comparator is provided by the TRACE32 development tools, it is listed as a Hardware Breakpoint.

**Example 1:** Sample only entries to the function `sieve`.

```
Break.Set sieve /Charly
TrOnchip.TEnable Charly
TrOnchip.TCYcle Fetch
```

**Example 2:** Sample all read and write accesses to the variable `flags[3]`.

```
Var.Break.Set flags[3] /Alpha
TrOnchip.TEnable Alpha
TrOnchip.TCYcle Access
```

Format: **TrOnchip.TCYcle** *<cycle>*

*<cycle>*:  
**ANY**  
**Read**  
**Write**  
**Access**  
**Fetch**  
**Soft**

Defines the cycle type for the bus trace address selector.

<b>ANY</b>	Cycle type doesn't matter.
<b>Read</b>	Record only read accesses.
<b>Write</b>	Record only write accesses.
<b>Access</b>	Record only data accesses.
<b>Fetch</b>	Record only instruction fetches.
<b>Soft</b>	Not used now.

**Example:** Assume there is a byte variable called 'flag' and you want to trigger if the value 59 is written to the variable.

```
Break.Set flag /Alpha          ; set an alpha breakpoint to the address
                                ; of the variable flag

TrOnchip.A Address Alpha      ; enable alpha break for on-chip trigger

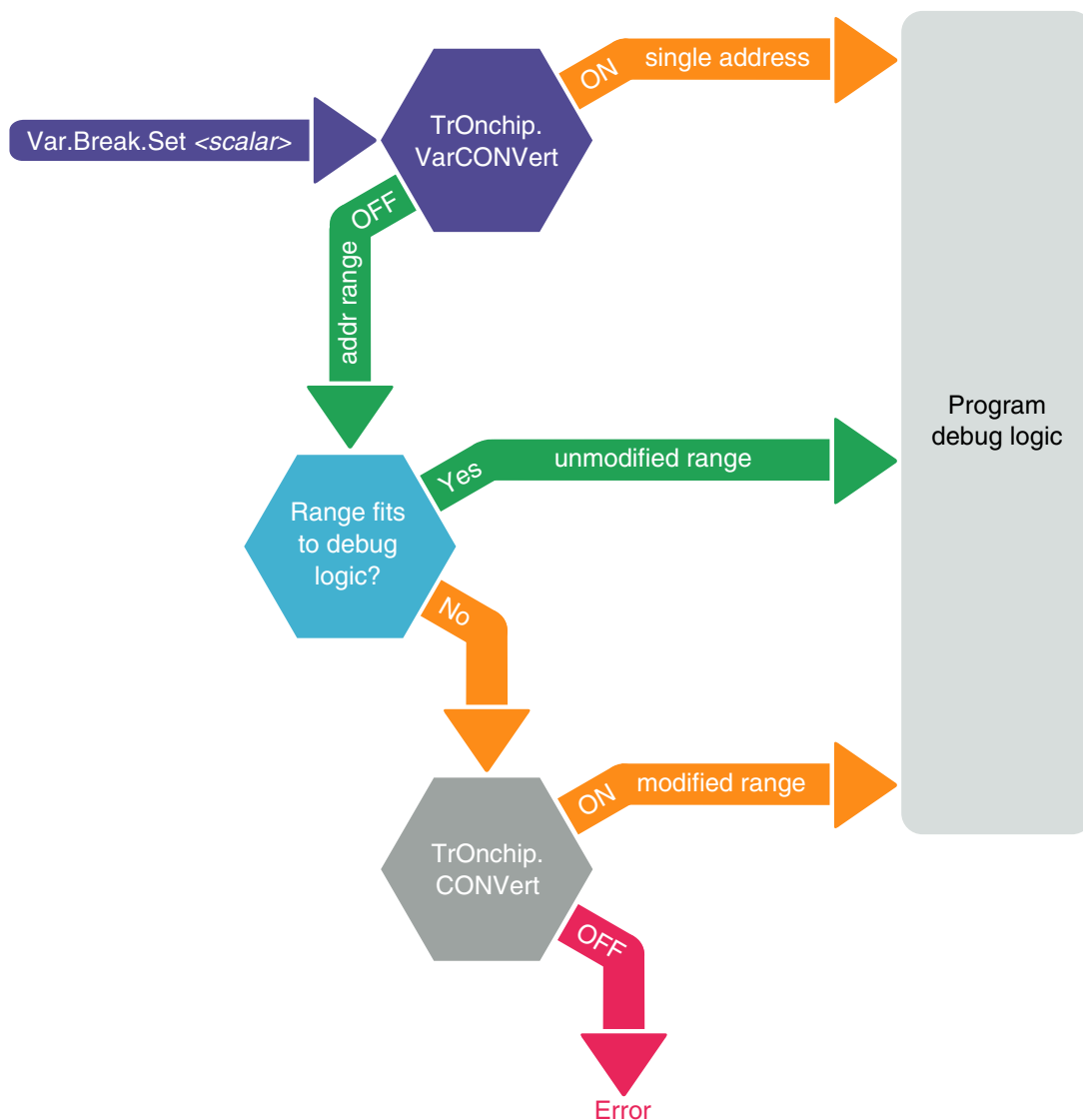
TrOnchip.A Value 0xxxxxx59    ; specify data pattern; this example
                                ; assumes that the address of flags is on
                                ; an address dividable by 4 and you have
                                ; little endian byte ordering (lowest byte
                                ; on data bus)

TrOnchip.A Cycle Write        ; specify that you want to trigger only on
                                ; a write access

TrOnchip.A Size Byte          ; specify that you want to trigger only on
                                ; byte access
```

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

Controls for all scalar variables whether the debugger sets an HLL breakpoint with **Var.Break.Set** only on the start address of the scalar variable or on the entire address range covered by this scalar variable.



<b>ON</b>	<p>If <b>TrOnchip.VarCONVert</b> is set to <b>ON</b> and a breakpoint is set to a scalar variable (int, float, double), then the breakpoint is set only to the start address of the scalar variable.</p> <ul style="list-style-type: none"><li>• Allocates only one single on-chip breakpoint resource.</li><li>• Program will not stop on accesses to the variable's address space.</li></ul>
<b>OFF</b> (default)	<p>If <b>TrOnchip.VarCONVert</b> is set to <b>OFF</b> and a breakpoint is set to a scalar variable (int, float, double), then the breakpoint is set to the entire address range that stores the scalar variable value.</p> <ul style="list-style-type: none"><li>• The program execution stops also on any unintentional accesses to the variable's address space.</li><li>• Allocates <b>up to two</b> on-chip breakpoint resources for a single range breakpoint.</li></ul> <p><b>NOTE:</b> The address range of the scalar variable may not fit to the debug logic and has to be converted by the debugger, see <a href="#">TrOnchip.CONVert</a>.</p>

In the [Break.List](#) window, you can view the requested address range for all breakpoints, whereas in the [Break.List /Onchip](#) window you can view the actual address range used for the on-chip breakpoints.

TrOnchip.state

Display on-chip trigger window

Format:	TrOnchip.state
---------	----------------

Opens the **TrOnchip.state** window.

MMU.DUMP

Page wise display of MMU translation table

Format:	<b>MMU.DUMP</b> <table> [<range>   <addr>   <range> <root>   <addr> <root>] [</option>] <b>MMU.&lt;table&gt;.dump</b> (deprecated)
<table>:	<b>PageTable</b> <b>KernelPageTable</b> <b>TaskPageTable</b> <task_magic>   <task_id>   <task_name>   <space_id>:0x0 <cpu_specific_tables>
<option>:	<b>MACHINE</b> <machine_magic>   <machine_id>   <machine_name> <b>Fulltranslation</b>

Displays the contents of the CPU specific MMU translation table.

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

<root>	The <root> argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
PageTable	Displays the current MMU translation table entries of the CPU. This command reads all tables the CPU currently uses for MMU translation and displays the table entries.
KernelPageTable	Displays the MMU translation table of the kernel. If specified with the <b>MMU.FORMAT</b> command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable <task_magic>   <task_id>   <task_name>   <space_id>:0x0	Displays the MMU translation table entries of the given process. Specify one of the <b>TaskPageTable</b> arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. <ul style="list-style-type: none"><li>• For information about the first three parameters, see <a href="#">“What to know about Task Magic Numbers, Task IDs and Task Names”</a> (general_ref_t.pdf).</li><li>• See also the appropriate <a href="#">OS Awareness Manuals</a>.</li></ul>

<b>MACHINE</b> <code>&lt;machine_magic&gt;  </code> <code>&lt;machine_id&gt;  </code> <code>&lt;machine_name&gt;</code>	<p>The following options are only available if <b>SYSTEM.Option MACHINESPACES</b> is set to <b>ON</b>.</p> <p>Dumps a page table of a virtual machine. The <b>MACHINE</b> option applies to <b>PageTable</b> and <b>KernelPageTable</b> and some <code>&lt;cpu_specific_tables&gt;</code>.</p> <p>The parameters <code>&lt;machine_magic&gt;</code>, <code>&lt;machine_id&gt;</code> and <code>&lt;machine_name&gt;</code> are displayed in the <b>TASK.List.MACHINES</b> window.</p>
<b>Fulltranslation</b>	<p>For page tables of guest machines both the <b>intermediate physical address</b> and the physical address is displayed in the <b>MMU.DUMP</b> window.</p> <p>The physical address is derived from a table walk using the guest's intermediate page table.</p>

## CPU-specific Tables in MMU.DUMP <table>

<b>ITLB</b>	Displays the contents of the Instruction Translation Lookaside Buffer. For column descriptions, click <a href="#">here</a> .
<b>DTLB</b>	Displays the contents of the Data Translation Lookaside Buffer. For column descriptions, click <a href="#">here</a> .
<b>TLB0</b>	Displays the contents of the Translation Lookaside Buffer 0. For column descriptions, click <a href="#">here</a> .
<b>TLB1</b>	Displays the contents of the Translation Lookaside Buffer 1. For column descriptions, click <a href="#">here</a> .
<b>NonSecPageTable</b>	Displays the translation table used if the CPU is in non-secure mode and in privilege level PL0 or PL1. This is the table pointed to by MMU registers TTBR0 and TTBR1 in non-secure mode. This option is only visible if the CPU has the TrustZone and/or Virtualization Extension.
<b>SecPageTable</b>	Displays the translation table used if the CPU is in secure mode. This is the table pointed to by MMU registers TTBR0 and TTBR1 in secure mode. This option is only visible if the CPU has the TrustZone Extension.
<b>HypPageTable</b>	Displays the translation table used by the MMU when the CPU is in HYP mode. This is the table pointed to by MMU register HTTBR. This table is only available in CPUs with Virtualization Extension.
<b>IntermedPageTable</b>	Displays the translation table used by the MMU for the second stage translation of a guest machine ( <b>intermediate physical address</b> to physical address). This is the table pointed to by MMU register VTTBR. This table is only available in CPUs with Virtualization Extension.



<b>Logical</b>	Logical address.
<b>Physical</b>	Physical address.
<b>Vmid</b>	Virtual machine ID.
<b>Asid</b>	Address space ID.
<b>Glb</b>	Global flag.
<b>Sec</b>	Non-secure identifier for physical address.
<b>idx</b>	Index of the TLB entry.
<b>pagesize</b>	Page size.
<b>Hyp</b>	Hypervisor entry flag.
<b>V</b>	Valid flag.
<b>L</b>	Locked flag.
<b>I</b>	Inner shareability flag.
<b>O</b>	Outer shareability flag.
<b>M</b>	Indicates if the line was brought in when MMU was enabled.
<b>D</b>	Domain ID.
<b>Attributes</b>	Memory Attributes (check design manual of respective architecture for the format).
<b>Tablewalk</b>	Table walk information.

### Example 1:

```
SYStem.Option MACHINESPACES ON

; your code to load Hypervisor Awareness and define guest machine setup.

;                                     <machine_id>
MMU.DUMP.PageTable /MACHINE          2.

;                                     <machine_name>
MMU.DUMP.PageTable /MACHINE          "Dom0 "
```

### Example 2:

```
SYStem.Option MACHINESPACES ON

; your code to load Hypervisor Awareness and define guest machine setup.

;                                     <machine_name>:::<task_name>
MMU.DUMP.TaskPageTable                "Dom0::swapper"
```

### Example 3:

```
SYStem.Option MACHINESPACES ON

;your code to load Hypervisor Awareness and define guest machine setup.

;a) dumps the current guest page table of the current machine, showing
;   the intermediate physical addresses.
;   Without the option /Fulltranslation the column "physical" is hidden.
MMU.DUMP.PageTable 0x400000

;b) With the option /Fulltranslation the intermediate physical addresses
;   are translated to physical addresses and shown in column "physical"
MMU.DUMP.PageTable 0x400000 /Fulltranslation

;c) dumps the current page table of machine 2
;                                     <machine_id>
MMU.DUMP.PageTable /MACHINE          2.          /Fulltranslation
```

## Results for 3 a) and 3 b)

logical	intermediate physical	sec	d	size	permissions
N:2::0000:00400000--00400FFF	I:2::411E8000--411E8FFF	ns		00001000	P:readonly
N:2::0000:00401000--00401FFF	I:2::411EC000--411ECFFF	ns		00001000	P:readonly
N:2::0000:00402000--00402FFF	I:2::411ED000--411EDFFF	ns		00001000	P:readonly

logical	intermediate physical	physical	sec	d	size	permissions
N:2::0000:00400000--00400FFF	I:2::411E8000--411E8FFF	AH:7F7EB000--7F7EBFFF	ns		00001000	P:readonly
N:2::0000:00401000--00401FFF	I:2::411EC000--411ECFFF	AH:7F7EC000--7F7ECFFF	ns		00001000	P:readonly
N:2::0000:00402000--00402FFF	I:2::411ED000--411EDFFF	AH:7F7ED000--7F7EDFFF	ns		00001000	P:readonly

## MMU.List

## Compact display of MMU translation table

Format:	<b>MMU.List</b> <i>&lt;table&gt;</i> [ <i>&lt;range&gt;</i>   <i>&lt;addr&gt;</i>   <i>&lt;range&gt;</i> <i>&lt;root&gt;</i>   <i>&lt;addr&gt;</i> <i>&lt;root&gt;</i> ] [ <i>&lt;option&gt;</i> ] <b>MMU.&lt;table&gt;.List</b> (deprecated)
<i>&lt;table&gt;</i> :	<b>PageTable</b> <b>KernelPageTable</b> <b>TaskPageTable</b> <i>&lt;task_magic&gt;</i>   <i>&lt;task_id&gt;</i>   <i>&lt;task_name&gt;</i>   <i>&lt;space_id&gt;</i> :0x0 <i>&lt;cpu_specific_tables&gt;</i>
<i>&lt;option&gt;</i> :	<b>MACHINE</b> <i>&lt;machine_magic&gt;</i>   <i>&lt;machine_id&gt;</i>   <i>&lt;machine_name&gt;</i> <b>Fulltranslation</b>

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

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

<i>&lt;root&gt;</i>	The <i>&lt;root&gt;</i> argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<b>PageTable</b>	Lists the current MMU translation of the CPU. This command reads all tables the CPU currently uses for MMU translation and lists the address translation.

<b>KernelPageTable</b>	Lists the MMU translation table of the kernel. If specified with the <a href="#">MMU.FORMAT</a> command, this command reads the MMU translation table of the kernel and lists its address translation.
<b>TaskPageTable</b> <task_magic>   <task_id>   <task_name>   <space_id>:0x0	Lists the MMU translation of the given process. Specify one of the <b>TaskPageTable</b> arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. <ul style="list-style-type: none"> <li>For information about the first three parameters, see “<a href="#">What to know about Task Magic Numbers, Task IDs and Task Names</a>” (general_ref_t.pdf).</li> <li>See also the appropriate <a href="#">OS Awareness Manuals</a>.</li> </ul>
<option>	For description of the options, see <a href="#">MMU.DUMP</a> .

## CPU-specific Tables in MMU.List <table>

<b>NonSecPageTable</b>	Displays the translation table used if the CPU is in non-secure mode and in privilege level PL0 or PL1. This is the table pointed to by MMU registers TTBR0 and TTBR1 in non-secure mode. This option is only visible if the CPU has the TrustZone and/or Virtualization Extension. This option is only enabled if Exception levels EL0 or EL1 use AArch32 mode.
<b>SecPageTable</b>	Displays the translation table used if the CPU is in secure mode. This is the table pointed to by MMU registers TTBR0 and TTBR1 in secure mode. This option is only visible if the CPU has the TrustZone Extension. This option is only enabled if the Exception level EL1 uses AArch32 mode.
<b>HypPageTable</b>	Displays the translation table used by the MMU when the CPU is in HYP mode. This is the table pointed to by MMU register HTTBR. This table is only available in CPUs with Virtualization Extension.
<b>IntermedPageTable</b>	Displays the translation table used by the MMU for the second stage translation of a guest machine ( <a href="#">intermediate physical address</a> to physical address). This is the table pointed to by MMU register VTTBR. This table is only available in CPUs with Virtualization Extension.

Format:	<b>MMU.SCAN</b> <table> [<range> <address>] [/<option>] <b>MMU.&lt;table&gt;.SCAN</b> (deprecated)
<table>:	<b>PageTable</b> <b>KernelPageTable</b> <b>TaskPageTable</b> <task_magic>   <task_id>   <task_name>   <space_id>:0x0 <b>ALL</b> <cpu_specific_tables>
<option>:	<b>MACHINE</b> <machine_magic>   <machine_id>   <machine_name> <b>Fulltranslation</b>

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

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

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

<b>PageTable</b>	Loads the current MMU address translation of the CPU. This command reads all tables the CPU currently uses for MMU translation, and copies the address translation into the debugger-internal static translation table.
<b>KernelPageTable</b>	Loads the MMU translation table of the kernel. If specified with the <a href="#">MMU.FORMAT</a> command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.
<b>TaskPageTable</b> <task_magic>   <task_id>   <task_name>   <space_id>:0x0	Loads the MMU address translation of the given process. Specify one of the <b>TaskPageTable</b> arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. <ul style="list-style-type: none"><li>• For information about the first three parameters, see <a href="#">“What to know about Task Magic Numbers, Task IDs and Task Names”</a> (general_ref_t.pdf).</li><li>• See also the appropriate <a href="#">OS Awareness Manual</a>.</li></ul>

<b>ALL</b>	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate <a href="#">OS Awareness Manual</a> .
<i>&lt;option&gt;</i>	For description of the options, see <a href="#">MMU.DUMP</a> .

## CPU-specific Tables in MMU.SCAN *<table>*

<b>OEMAddressTable</b>	Loads the OEM Address Table from the CPU to the debugger internal translation table.
<b>NonSecPageTable</b>	Displays the translation table used if the CPU is in non-secure mode and in privilege level PL0 or PL1. This is the table pointed to by MMU registers TTBR0 and TTBR1 in non-secure mode. This option is only visible if the CPU has the TrustZone and/or Virtualization Extension. This option is only enabled if Exception levels EL0 or EL1 use AArch32 mode.
<b>SecPageTable</b>	Displays the translation table used if the CPU is in secure mode. This is the table pointed to by MMU registers TTBR0 and TTBR1 in secure mode. This option is only visible if the CPU has the TrustZone Extension. This option is only enabled if the Exception level EL1 uses AArch32 mode.
<b>HypPageTable</b>	Loads the translation table used by the MMU when the CPU is in HYP mode. This is the table pointed to by MMU register HTTBR. This table is only available in CPUs with Virtualization Extension.
<b>IntermedPageTable</b>	Loads the translation table used by the MMU for the second stage translation of a guest machine ( <a href="#">intermediate physical address</a> to physical address). This is the table pointed to by MMU register VTTBR. This table is only available in CPUs with Virtualization Extension.

Using the **SMMU** command group, you can analyze the current setup of up to 20 system MMU instances. Selecting a CPU with a built-in SMMU activates the **SMMU** command group.

```
SYStem.CPU ARMv8 ;for example, the 'ARMv8' CPU is SMMU-capable

SMMU.ADD ...      ;you can now define an SMMU, e.g. an SMMU for a
                  ;graphics processing unit (GPU)
```

The TRACE32 SMMU support visualizes the most important configuration settings of an SMMU. These visualizations include:

- The context type defined for each stream map register group (SMRG). This visualization shows the translation context associated with the SMRG such as:
  - The stage 1 context bank and the stage 1 page table type
  - The stage 2 context bank and the stage 2 page table type
  - The information whether the SMRG context is a HYPC and MONC context type
- The stream matching register settings, if supported by the SMMU
- The associated context bank index, the page table format and the MMU-enable/disable state for stage 1 and/or stage 2 address translation contexts
- Page table dumps for stage 1 and/or stage 2 address translation contexts
- A quick indication of contexts where a fault has occurred or contexts that are stalled.
- A quick indication of the global SMMU fault status
- Peripheral register view:
  - Global Configuration Registers of the SMMU
  - Stream Matching and Mapping Registers
  - Context Bank Registers

A good way to familiarize yourself with the **SMMU** command group is to start with:

- The [SMMU.ADD](#) command
- The [SMMU.StreamMapTable](#) command
- [Glossary - SMMU](#)
- [Arguments in SMMU Commands](#)

The **SMMU.StreamMapTable** command and the window of the same name serve as your SMMU command and control center in TRACE32. The right-click popup menu in the **SMMU.StreamMapTable** window allows you to execute all frequently-used SMMU commands through the user interface TRACE32 PowerView.

The other SMMU commands are designed primarily for use in PRACTICE scripts (\*.cmm) and for users accustomed to working with the command line.

## Glossary - SMMU

This figure illustrates a few SMMU terms. For explanations of the illustrated SMMU terms and other important SMMU terms not shown here, see below.

stream map visibility	reg. grp index	stream ref. id	matching id mask	valid	context type	stage 1 pagetbl. fmt	cbndx	state	stage 2 pagetbl. fmt	cbndx	state
sec/nsec	0x06	0x0B4C	0x7000	yes	bypass mode						
sec/nsec	0x07	0x0000	0x0000	no	fault						
sec/nsec	0x08	0x0000	0x0000	no	fault						
sec/nsec	0x09	0x0341	0x7000	yes	s1 trsl - s2 byp	AArch32 Shrt	0x12	on			
sec/nsec	0x0A	0x0EB5	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x14	on	AArch64 Long	0x15	on
sec/nsec	0x0B	0x0464	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x16	on	AArch64 Long	0x17	on
sec/nsec	0x0C	0x00FB	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x18	on	AArch64 Long	0x19	on
sec/nsec	0x0D	0x0587	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x1A	on	AArch64 Long	0x1B	on
sec only	0x0E	0x0000	0x0000	no	fault						
sec only	0x0F	0x0000	0x0000	no	fault						
sec only	0x10	0x0000	0x0000	no	fault						
sec only	0x11	0x0000	0x0000	no	fault						

**A** See [stream mapping table](#).

**B** Each row stands for a [stream map register group \(SMRG\)](#).

**C** Index of a [translation context bank](#).

**D** Data from stream matching registers, see [stream matching](#).

## Memory Transaction Stream

A stream of memory access transactions sent from a device through the SMMU to the system memory bus. The stream consists of the address to be accessed and a number of design specific memory attributes such as the privilege, cacheability, security attributes or other attributes.

The streams carry a stream ID which the SMMU uses to determine a translation context for the memory transaction stream. As a result, the SMMU may or may not translate the address and/or the memory attributes of the stream before it is forwarded to the system memory bus.

## Security State Determination Table (SSD Table)

If the SMMU supports two security states (secure and non-secure) an SSD index qualifies memory transactions sent to the SMMU. The SSD index is a hardware signal which is used by the SMMU to decide whether the incoming memory transaction belongs to the secure or the non-secure domain.

The information whether a SSD index belongs to the secure or to the non-secure domain is contained in the SMMU's SSD table.



## Stream ID

---

Peripheral devices connected to an SMMU issue memory transaction streams. Every incoming memory transaction stream carries a Stream Identifier which is used by the SMMU to associate a translation context to the transaction stream.

## Stream Map Register Group (SMRG)

---

A group of SMMU registers determining the translation context for a memory transaction stream, see [stream mapping table](#).

## Stream Mapping Table (short: Stream Map Table)

---

An SMMU table which describes what to do with an incoming memory transaction stream from a peripheral device. In particular, this table associates an incoming memory transaction stream with a translation context, using the stream ID of the stream as selector of a translation context.

Each stream mapping table entry consists of a group of registers, called *stream map register group*, which describe the translation context.

In case an SMMU supports *stream matching*, TRACE32 also displays the *stream matching registers* associated with an entry's stream map register group. The stream mapping table is the central table of the SMMU. See [SMMU.StreamMapTable](#).

## Stream Matching

---

In an SMMU which supports stream matching, the stream ID of an incoming memory transaction stream undergoes a matching process to determine which entry of the stream mapping table will be used to specify the translation context for the stream.

TRACE32 displays the reference ID and the bit mask used by the SMMU to perform the stream ID matching process in the [SMMU.StreamMapTable](#) window.

## Translation Context

---

A translation context describes the translation process of an incoming memory transaction stream. An incoming memory transaction stream may undergo a stage 1 address translation and/or a stage 2 address translation. Further, the memory attributes of the incoming memory transaction stream may be changed. It is also possible that an incoming memory transaction stream is rendered as fault.

The stream mapping table determines which translation context is applied to an incoming memory transaction stream.

## Translation Context Bank (short: Context Bank)

---

A group of SMMU registers specifying the translation context for an incoming memory transaction stream. The registers carry largely the same names and contain the same information as the core's MMU registers describing the address translation process.

The registers of a translation context bank describe the translation table base address, the memory attributes to be used during the translation table walk and translation attribute remapping.

This table provides an overview of frequently-used arguments in SMMU commands. Arguments that are only used in one **SMMU** command are described together with that **SMMU** command.

<code>&lt;name&gt;</code>	User-defined name of an SMMU. Use the <b>SMMU.ADD</b> command to define an SMMU and its name. This name will be used to identify an SMMU in all other <b>SMMU</b> commands.
<code>&lt;smrg_index&gt;</code>	Index of a stream map register group, e.g. 0x04. The indices are listed in the <b>index</b> column of the <b>SMMU.StreamMapTable</b> .
<code>&lt;cbndx&gt;</code>	Index of a translation context bank.
<code>&lt;address&gt;   &lt;range&gt;</code>	Logical address or logical address range describing the start address or the address range to be displayed in the SMMU page table list or dump windows.
<b>IntermediatePT</b>	Used to switch between stage 1 and stage 2 page table or register view: <ul style="list-style-type: none"><li>• Omit this option to view the translation table entries or registers of stage 1.</li><li>• Include this option to view the translation table entries or registers of stage 2.</li></ul>

Format:	<b>SMMU.ADD</b> "<name>" <smmu_type> <base_address>
<smmu_type>:	<b>MMU400   MMU401   MMU500</b>

Defines a new SMMU (a hardware system MMU). A maximum of 20 SMMUs can be defined.

<b>NOTE:</b>	<p>For some CPUs with SMMUs, TRACE32 will automatically configure the SMMU parameters, so that you can immediately work with the SMMUs and do not need to manually configure them.</p> <p>After selecting the CPU type, check one of the following locations in TRACE32 to see if there are any pre-configured SMMUs:</p> <ul style="list-style-type: none"><li>• The <b>CPU</b> menu &gt; <b>SMMU</b> popup menu</li><li>• The <b>SYStem.CONFIG.state /COmponents</b> window</li></ul>
--------------	---

Arguments:

<base_address>	<p>Logical or physical base address of the memory-mapped SMMU register space.</p> <p><b>NOTE:</b> If the SMMU supports two security states (secure and non-secure), not all SMMU registers are visible from the non-secure domain.</p> <ul style="list-style-type: none"><li>• If you specify a <b>secure</b> address as the SMMU base address, you will be able to see <b>all</b> SMMU information.</li><li>• If you specify a <b>non-secure</b> address as the SMMU base address, you will only see the SMMU information which is visible from the non-secure domain.</li></ul> <p>To specify a <b>secure</b> address, precede the base address with an <b>access class</b> such as <b>AZ:</b> or <b>ZD:</b></p> <p>The <b>SMMU.ADD</b> command interprets access classes with an ambiguous security status as secure access classes:</p> <ul style="list-style-type: none"><li>• Physical access class A: becomes <b>AZ:</b></li><li>• Logical access classes like D: or C: become <b>ZSD:</b></li></ul> <p>The <b>SMMU.ADD</b> command leaves access classes with a distinct security status unchanged, e.g. the access classes <b>NSD:</b>, <b>NUD:</b>, <b>HD:</b> etc.</p>
----------------	---

<code>&lt;name&gt;</code>	<p>User-defined name of an SMMU. The name must be unique and can be max. 9 characters long.</p> <p><b>NOTE:</b></p> <ul style="list-style-type: none"> <li>For the <b>SMMU.ADD</b> command, the name must be quoted.</li> <li>For <i>all other</i> <b>SMMU</b> commands, omit the quotation marks from the name identifying an SMMU. See also PRACTICE script example below.</li> </ul>
<code>&lt;smmu_type&gt;</code>	<p>Defines the type of the ARM system MMU IP block: <b>MMU400</b>, <b>MMU401</b>, or <b>MMU500</b>.</p>

Example:

```

;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU500 AZ:0x50000000

;display the stream map table named myGPU
SMMU.StreamMapTable myGPU

```

SMMU.Clear

Delete an SMMU

Format:

SMMU.Clear `<name>`

Deletes an SMMU definition, which was created with the **SMMU.ADD** command of TRACE32. The **SMMU.Clear** command does not affect your target SMMU.

To delete all SMMU definitions created with the **SMMU.ADD** command of TRACE32, use **SMMU.RESet**.

Argument:

<code>&lt;name&gt;</code>	For a description of <code>&lt;name&gt;</code> , click <a href="#">here</a> .
---------------------------	---

Example:

```

SMMU.Clear myGPU      ;deletes the SMMU named myGPU

```

Using the **SMMU.Register** command group, you can view and modify the peripheral registers of an SMMU. The command group provides the following commands:

<b>SMMU.Register.ContextBank</b>	Display the registers of a context bank
<b>SMMU.Register.Global</b>	Display the global registers of an SMMU
<b>SMMU.Register.StreamMapRegGrp</b>	Display the registers of an SMRG

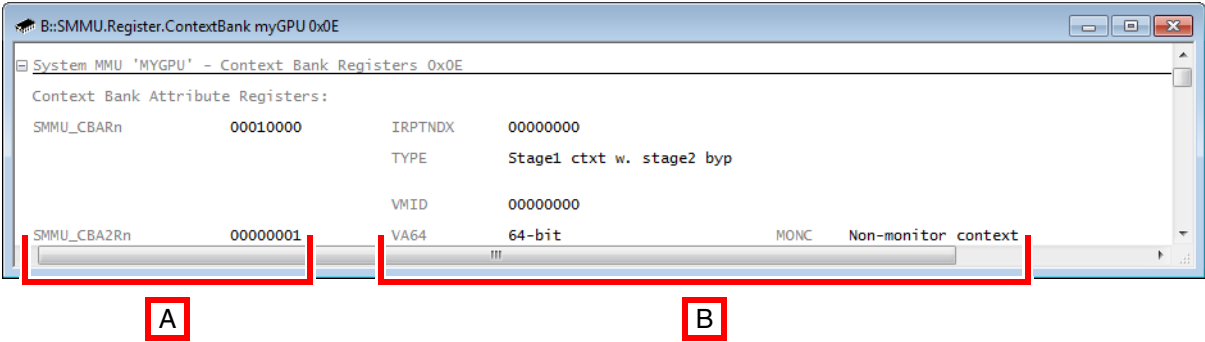
**Example:**

```
;open the SMMU.Register.StreamMapRegGrp window
SMMU.Register.StreamMapRegGrp    myGPU    0x0A

;highlight changes in any SMMU.Register.* window in orange
SETUP.Var %SpotLight.on
```

Format: **SMMU.Register.ContextBank** <name> <cbndx>

Opens the peripheral register window **SMMU.Register.ContextBank**. This window displays the registers of the specified context bank. These are listed under the section heading **Context Bank Registers**.



- A Register name and content.
- B Names of the register bit fields and bit field values.

NOTE:

The commands **SMMU.Register.ContextBank** and **SMMU.StreamMapRegGrp.ContextReg** are similar.

The difference between the two commands is:

- The first command expects a <cbndx> as an argument and allows to view an arbitrary context bank.
- The second command expects an <smrg\_index> with an optional **IntermediatePT** as arguments and displays either a stage 1 or stage 2 context bank associated with the <smrg\_index>.

Argument:

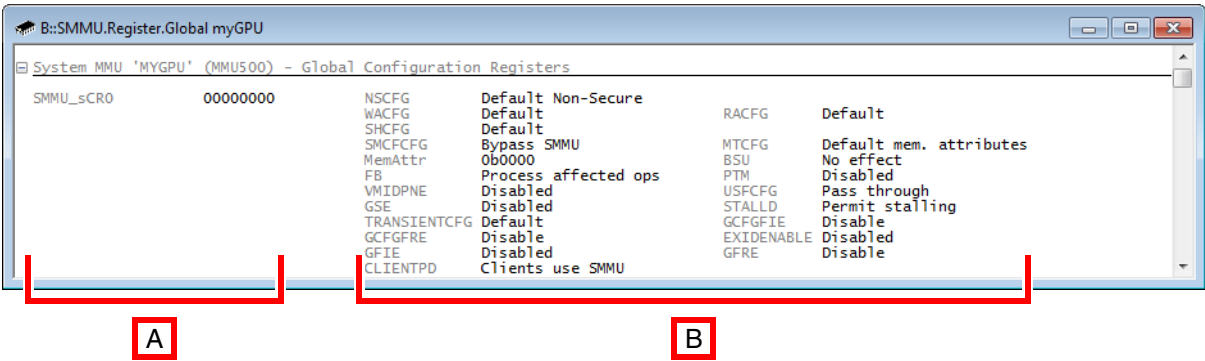
<name>	For a description of <name>, etc., click <a href="#">here</a> .
--------	---

Example:

```
SMMU.Register.ContextBank myGPU 0x16
```

Format: **SMMU.Register.Global** <name>

Opens the peripheral register window **SMMU.Register.Global**. This window displays the global registers of the specified SMMU. These are listed under the section heading **Global Configuration Registers**.



- A Register name and content.
- B Names of the register bit fields and bit field values.

Argument:

<name>	For a description of <name>, click <a href="#">here</a> .
--------	---

Example:

```
SMMU.Register.Global myGPU
```

To display the global registers of an SMMU via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click an SMRG, and then select **Peripherals > Global Configuration Registers** from the popup menu.

Format:

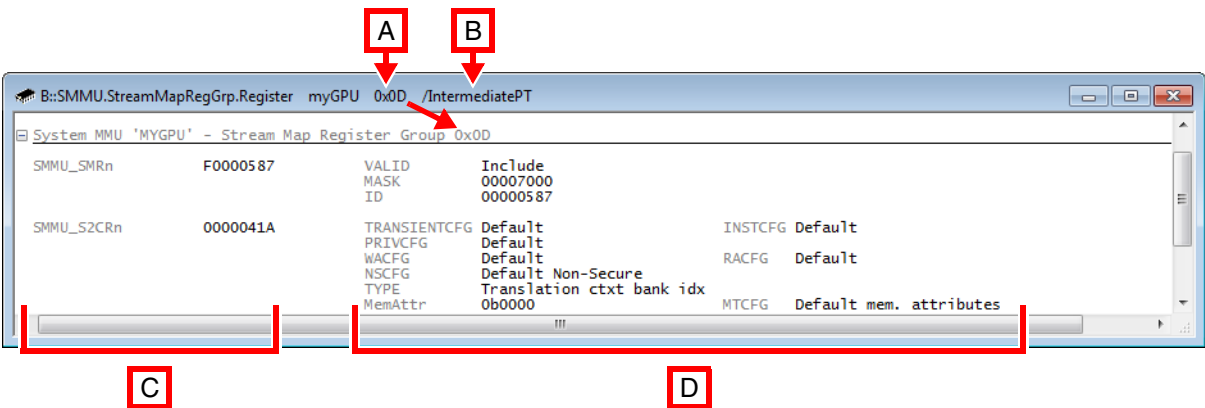
SMMU.Register.StreamMapRegGrp <args>

SMMU.StreamMapRegGrp.Register <args> (as an alias)

<args>:

<name> <smrg\_index> [/IntermediatePT]

Opens the peripheral register window **SMMU.Register.StreamMapRegGrp**. This window displays the registers of the specified SMRG. These are listed under the gray section heading **Stream Map Register Group**.



- A 0x0D is the <smrg\_index> of the selected SMRG.
  - B The option **IntermediatePT** is used to display the context bank registers of stage 2.
  - C Register name and content.
  - D Names of the register bit fields and bit field values.
- Compare also to [SMMU.StreamMapRegGrp.ContextReg](#).

Arguments:

<name>	For a description of <name>, etc., click <a href="#">here</a> .
--------	---

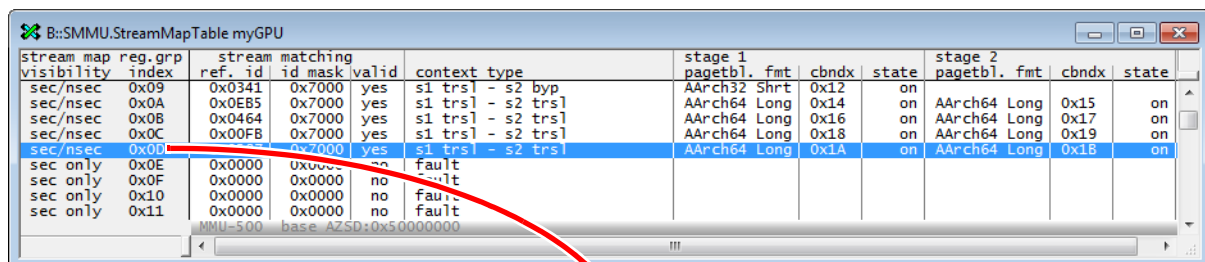
Example:

```
SMMU.StreamMapRegGrp.Register    myGPU    0x0D    /IntermediatePT
```

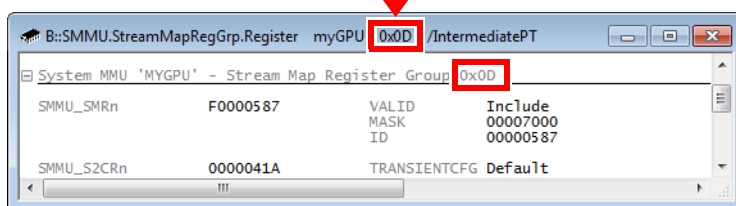


To view the registers of an SMRG via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click an SMRG, and then select **Peripherals > Stream Mapping Registers** from the popup menu.



stream map visibility	reg. grp index	stream ref. id	matching id mask	valid	context type	stage 1 pagetbl. fmt	cbndx	state	stage 2 pagetbl. fmt	cbndx	state
sec/nsec	0x09	0x0341	0x7000	yes	s1 trsl - s2 byp	AArch32 Shrt	0x12	on	AArch64 Long	0x15	on
sec/nsec	0x0A	0x0EB5	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x14	on	AArch64 Long	0x17	on
sec/nsec	0x0B	0x0464	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x16	on	AArch64 Long	0x19	on
sec/nsec	0x0C	0x00FB	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x18	on	AArch64 Long	0x1B	on
sec/nsec	0x0D	0x0000	0x0000	yes	s1 trsl - s2 trsl	AArch64 Long	0x1A	on	AArch64 Long	0x1B	on
sec only	0x0E	0x0000	0x0000	no	fault						
sec only	0x0F	0x0000	0x0000	no	fault						
sec only	0x10	0x0000	0x0000	no	fault						
sec only	0x11	0x0000	0x0000	no	fault						



B::SMMU.StreamMapRegGrp.Register myGPU 0x00 /IntermediatePT			
System MMU 'MYGPU' - Stream Map Register Group 0x00			
SMMU_SMRn	F0000587	VALID MASK ID	Include 00007000 00000587
SMMU_S2CRn	0000041A	TRANSIENTCFG	Default

## SMMU.RESet

## Delete all SMMU definitions

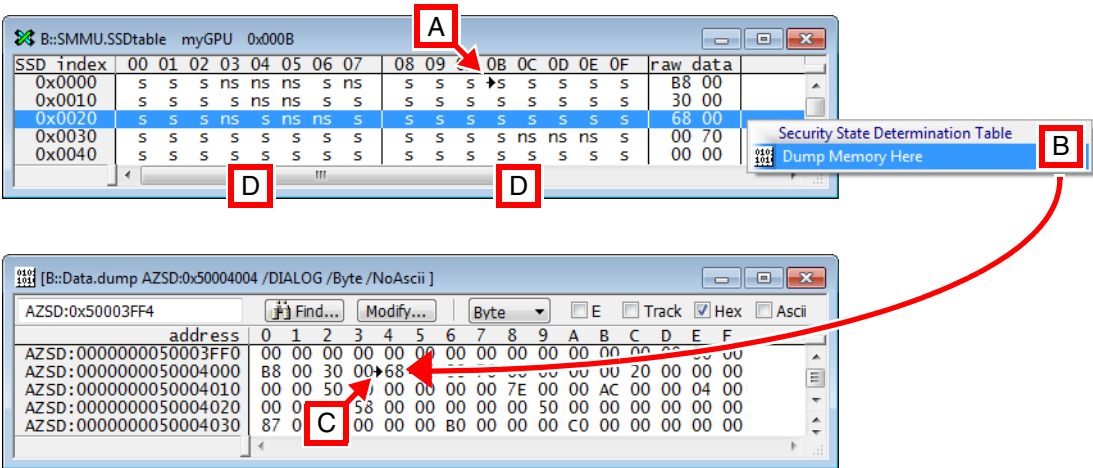
Format: **SMMU.RESet**

Deletes all SMMU definitions created with **SMMU.ADD** from TRACE32. The **SMMU.RESet** command does not affect your target SMMU.

To delete an individual SMMU created with **SMMU.ADD**, use **SMMU.Clear**.

Format: **SMMU.SSDtable** <name> [<start\_index>]

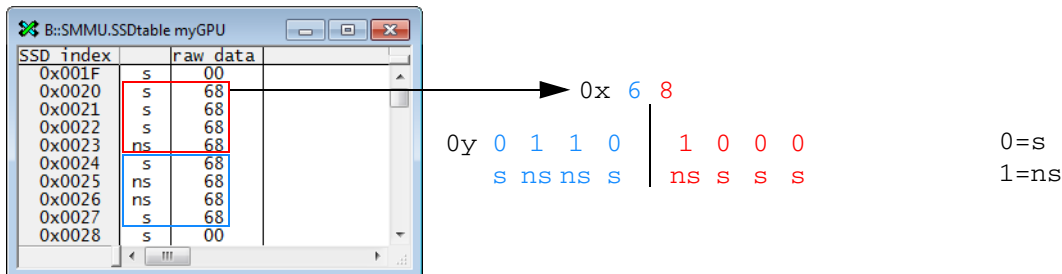
Displays the security state determination table (SSD table) as a bit field consisting of **s** (secure) or **ns** (non-secure) entries. If the SMMU has no SSD table defined, you receive an error message in the **AREA** window.



- A** In the SSD table, the black arrow indicates the <start\_index>, here 0x00B
- B** Right-click to dump the SSD table raw data in memory.

For each SSD index of an incoming memory transaction stream, the SSD table indicates whether the outgoing memory transaction stream accesses the secure (**s**) or non-secure (**ns**) memory domain.

You may find the SSD table easier to interpret by reducing the width of the **SMMU.SSDtable** window. Example for the raw data 0x68 in the SSD table:



- C** In the **Data.dump** window, the black arrow indicates the dumped raw data from the SSD table.
- D** The 1st white column (00 to 07) relates to the 1st **raw data** column.  
The 2nd white column (08 to 0F) relates to the 2nd **raw data** column, etc.

Arguments:

<name>	For a description of <name>, click <a href="#">here</a> .
<start_index>	Starts the display of the SSD table at the specified SSD index. See <b>SSD index</b> column in the <b>SMMU.SSDtable</b> window.

Example:

```
;display the SSD table starting at the SSD index 0x000B
SMMU.SSDtable      myGPU      0x000B
```

To view the SSD table via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click any SMRG, and then select **Security State Determination Table (SSD)** from the popup menu.

<b>NOTE:</b>	The menu option is grayed out if the SMMU does not support the two security states <b>s</b> (secure) or <b>ns</b> (non-secure).
--------------	---

The **SMMU.StreamMapRegGrp** command group allows to view the details of the translation context associated with stage 1 and/or stage 2 of an SMRG. Every SMRG is identified by its [<smrg\\_index>](#).

The **SMMU.StreamMapRegGrp** command group provides the following commands:

<b>SMMU.StreamMapRegGrp.ContextReg</b>	Shows the registers of the context bank associated with the stage 1 and/or stage 2 translation.
<b>SMMU.StreamMapRegGrp.Dump</b>	Dumps the page table associated with the stage 1 and/or stage 2 translation page wise.
<b>SMMU.StreamMapRegGrp.List</b>	Lists the page table entries associated with the stage 1 and/or stage 2 translation in a compact format.

Format:

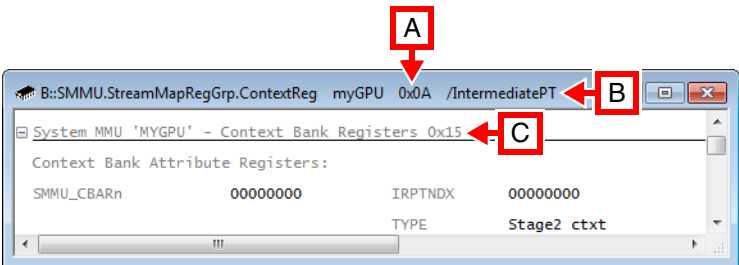
SMMU.StreamMapRegGrp.ContextReg <args>

<args>:

<name> <smrg\_index> [/IntermediatePT]

Opens the peripheral register window **SMMU.StreamMapRegGrp.ContextReg**, displaying the context bank registers of stage 1 or stage 2 of the specified <smrg\_index> [A]. The context bank index (cbndx) of the shown context bank registers is printed in the gray section heading **Context Bank Registers** [C].

The **cbndx** columns in the **SMMU.StreamMapTable** window tell you which context bank is associated with stage 1 or stage 2: If there is no context bank defined for stage 1 or stage 2, then the respective **cbndx** cell is empty. In this case, the peripheral register window **SMMU.StreamMapRegGrp.ContextReg** does not open.



- A 0x0A is the <smrg\_index> of the selected SMRG.
- B The option **IntermediatePT** is used to display the context bank registers of stage 2.
- C 0x15 is the index from the **cbndx** column of a stage 2 context bank. See [example](#) below.
- Compare also to **SMMU.StreamMapRegGrp.Register**.

NOTE:

The commands **SMMU.Register.ContextBank** and **SMMU.StreamMapRegGrp.ContextReg** are similar.

The difference between the two commands is:

- The first command expects a <cbndx> as an argument and allows to view an arbitrary context bank.
- The second command expects an <smrg\_index> with an optional **IntermediatePT** as arguments and displays either a stage 1 or stage 2 context bank associated with the <smrg\_index>.

Arguments:

<name>	For a description of <name>, etc., click <a href="#">here</a> .
--------	---

## PRACTICE Script Example and Illustration of the Context Bank Look-up:

```
SMMU.StreamMapRegGrp.ContextReg myGPU 0x0A /IntermediatePT
```

The top window, titled "System MMU 'MYGPU' - Context Bank Registers 0x15", displays the "Context Bank Attribute Registers". The registers shown are:

Register	Value	IRPTNDX	Value	TYPE	Value
SMMU_CBARN	00000000		00000000		
				Stage2	txt

The bottom window, titled "B::SMMU.StreamMapTable myGPU", displays a table of stream map entries. The entry for "SMMU-500 base AZSD:0x50000000" is highlighted in blue. The table columns are:

stream map visibility	reg. grp index	stream map ref. id	stream map mask	stream map valid	context type	stage 1 pagetbl. fmt	cbndx	state	stage 2 pagetbl. fmt	cbndx	state
sec/nsec	0x05	0x0099	0x7000	yes	s1 trsl - s2 byp	AArch32 Long	0x0A	on			
sec/nsec	0x06	0x009C	0x7000	yes	bypass mode						
sec/nsec	0x07	0x0000	0x0000	no	fault						
sec/nsec	0x08	0x0000	0x0000	no	fault						
sec/nsec	0x09	0x0341	0x7000	yes	s1 trsl - s2 byp						
sec/nsec	0x0A	0x0EB5	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x14	on	AArch64 Long	0x15	on
sec/nsec	0x0	0x0464	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x16	on	AArch64 Long	0x17	on
sec/nsec	0x0	0x00FB	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x18	on	AArch64 Long	0x19	on
sec/nsec	0x0	0x0587	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x1A	on	AArch64 Long	0x1B	on

To display the context bank registers via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click an SMRG, and then select **Peripherals > Context Bank Registers of Stage 1 or 2** from the popup menu.

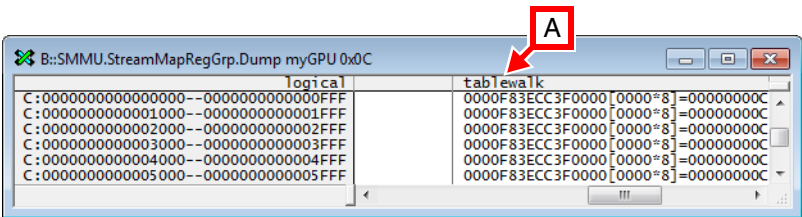
Format:

SMMU.StreamMapRegGrp.Dump <args>

<args>:

<name> <smrg\_index> [<address> | <range>] [/<option>]

Opens the **SMMU.StreamMapRegGrp.Dump** window for the specified SMRG, displaying the page table entries of the SMRG page wise. If no valid translation context is defined, the window displays the error message “registerset undefined”.



**A** To view the details of the page table walk, scroll to the right-most column of the window.  
For a description of the columns in the **SMMU.StreamMapRegGrp.Dump** window, click [here](#).

Arguments:

<name>	For a description of <name>, etc., click <a href="#">here</a> .
IntermediatePT	<p>Omit this option to view translation table entries of stage 1.</p> <p>Include this option to view translation table entries of stage 2.</p> <p>In SMMUs that support only stage 2 page tables, this option can be omitted.</p>

Example:

```
SMMU.StreamMapRegGrp.Dump myGPU 0x0C
```

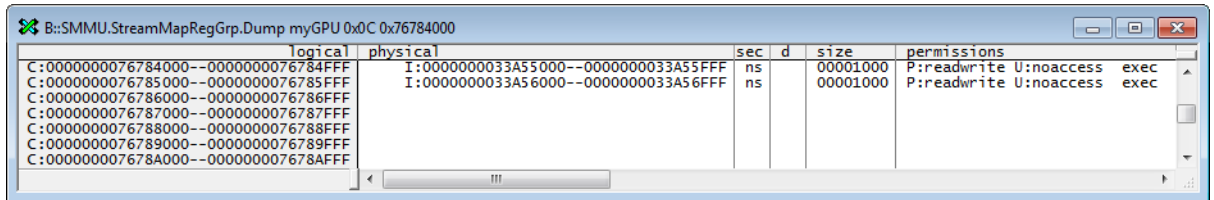
To display an SMMU page table page-wise via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click an SMRG, and then select from the popup menu:
  - **Stage 1 Page Table > Dump** or
  - **Stage 2 Page Table > Dump**

## Description of Columns

This table describes the columns of the following windows:

- [SMMU.StreamMapRegGrp.List](#)
- [SMMU.StreamMapRegGrp.Dump](#)



The screenshot shows a window titled "B::SMMU.StreamMapRegGrp.Dump myGPU 0xC0x76784000". It displays a table with the following columns: logical, physical, sec, d, size, and permissions. The data rows show memory mappings for various address ranges, with security states (ns, ns) and domains (ns, ns). The permissions column shows "P:readwrite U:noaccess exec" for the first two rows and "P:readwrite U:noaccess" for the last two rows.

logical	physical	sec	d	size	permissions
C:0000000076784000--0000000076784FFF	I:0000000033A55000--0000000033A55FFF	ns	ns	00001000	P:readwrite U:noaccess exec
C:0000000076785000--0000000076785FFF	I:0000000033A56000--0000000033A56FFF	ns	ns	00001000	P:readwrite U:noaccess exec
C:0000000076786000--0000000076786FFF					
C:0000000076787000--0000000076787FFF					
C:0000000076788000--0000000076788FFF					
C:0000000076789000--0000000076789FFF					
C:000000007678A000--000000007678AFFF					

Column	Description
logical	Logical page address range
physical	Physical page address range
sec	Security state of entry (s=secure, ns=non-secure, sns=non-secure entry in secure page table)
d	Domain
size	Size of mapped page in bytes
permissions	Access permissions (P=privileged, U=unprivileged, exec=execution allowed)
glb	Global page
shr	Shareability (no=non-shareable, yes=shareable, inn=inner shareable, out=outer shareable)
pageflags	Memory attributes (see <a href="#">Description of the memory attributes.</a> )
tablewalk	Only for <a href="#">SMMU.StreamMapRegGrp.Dump</a> : <ul style="list-style-type: none"><li>• Details of table walk for logical page address (one sub column for each table level, showing the table base address, entry index, entry width in bytes and value of table entry)</li></ul>



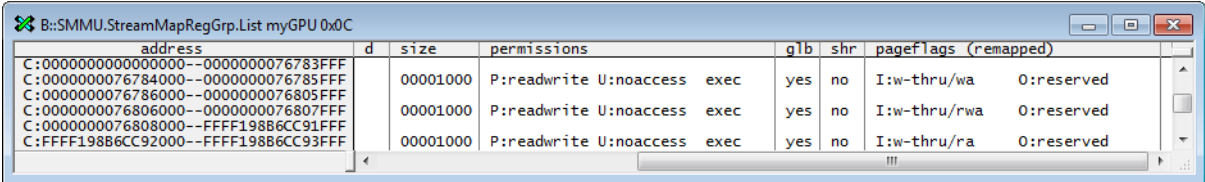
Format:

SMMU.StreamMapRegGrp.List <args>

<args>:

<name> <smrg\_index> [<address> | <range>] [/IntermediatePT]

Opens the **SMMU.StreamMapRegGrp.List** window for the specified SMMU, listing the page table entries of a stream map group. If no valid translation context is defined, the window displays an error message.



For a description of the columns in the **SMMU.StreamMapRegGrp.List** window, click [here](#).

Arguments:

<name>	For a description of <name>, etc., click <a href="#">here</a> .
IntermediatePT	<p>Omit this option to view translation table entries of stage 1. Include this option to view translation table entries of stage 2.</p> <p>In SMMUs that support only stage 2 page tables, this option can be omitted.</p>

Example:

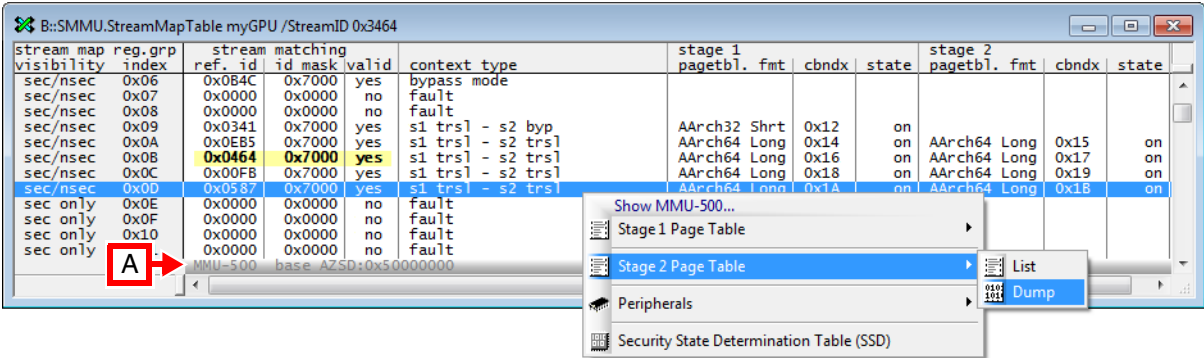
```
SMMU.StreamMapRegGrp.List myGPU 0x0C
```

To list the page table entries via the user interface TRACE32 PowerView:

- In the **SMMU.StreamMapTable** window, right-click an SMRG, and then select from the popup menu:
  - Stage 1 Page Table > List** or
  - Stage 2 Page Table > List**

Format: **SMMU.StreamMapTable** <name> [/StreamID <value>]

Opens the **SMMU.StreamMapTable** window, listing all stream map register groups of the SMMU that has the specified <name>. The window provides an overview of the SMMU configuration.



**A** The gray window status bar displays the <smmu\_type> and the SMMU <base\_address>. In addition, the window status bar informs you of [global faults](#) in the SMMU, if there are any faults.

Arguments

<name>	For a description of <name>, click <a href="#">here</a> .
<b>StreamID</b> <value>	<p>Only available for SMMUs that support stream ID matching. The <b>StreamID</b> option highlights all SMRGs in <b>yellow</b> that match the specified stream ID &lt;value&gt;. SMRGs highlighted in yellow help you identify incorrect settings of the stream matching registers.</p> <p>For &lt;value&gt;, specify the stream ID of an incoming memory transaction stream.</p> <ul style="list-style-type: none"><li>The highlighted SMRG indicates which stream map table entry will be used to translate the incoming memory transaction stream.</li><li>More than one highlighted row indicates a potential, global SMMU fault called stream match conflict fault.</li></ul> <p>The stream ID matching algorithm of TRACE32 mimics the SMMU stream matching on the real hardware.</p> <p>The reference ID, mask and validity fields of the stream match register are listed in the <b>ref. id</b>, <b>id mask</b> and <b>valid</b> columns.</p>

This PRACTICE script example shows how to define an SMMU with the **SMMU.ADD** command. Then the script opens the SMMU in the **SMMU.StreamMapTable** window, searches for the `<stream_id> 0x3463` and highlights the matching SMRG `0x0464` in yellow.

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU500 A:0x50000000

;open the window and highlight the matching SMRG in yellow
SMMU.StreamMapTable myGPU /StreamID 0x3464
```

stream map visibility	reg. grp index	stream ref. id	stream id mask	matching id mask	valid	context type
sec/nsec	0x09	0x0341	0x7000		yes	s1 trs1 - s2 byp
sec/nsec	0x0B	0x0464	0x7000	yes		s1 trs1 - s2 trs1
sec/nsec	0x0D	0x0587	0x7000	yes		s1 trs1 - s2 trs1
sec only	0x0E	0x0000	0x0000	no		fault
sec only	0x0F	0x0000	0x0000	no		fault

MMU-500 base AZSD:0x50000000

**NOTE:** At first glance, the **StreamID** `0x3464` does not seem to match the SMRG `0x0464`. However, if you take the ID mask `0x7000` (= `0y0111_0000_0000_0000`) into account, the match is correct.

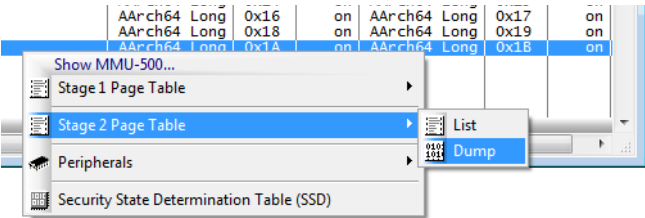
The row highlighted in yellow in the **SMMU.StreamMapTable** window is a correct match for the **StreamID** `0x3464` we searched for.

See also function **SMMU.StreamID2SMRG()** in “[General Function Reference](#)” (`general_func.pdf`).

## About the SMMU.StreamMapTable Window

By right-clicking an SMRG or double-clicking certain cells of an SMRG, you can open additional windows to receive more information about the selected SMRG.

- Right-clicking opens the **Popup Menu**.
- Double-clicking an SMRG in the columns **ref. id**, **id mask**, **valid**, or **context type** opens the **SMMU.StreamMapRegGrp.Register** window.
- Double-clicking an SMRG in the two columns **pagetbl. fmt** opens the **SMMU.StreamMapRegGrp.List** window, displaying the page table for stage 1 or stage 2.
- Double-clicking an SMRG in the two **cbndx** columns or the two **state** columns opens the **SMMU.StreamMapRegGrp.ContextReg** window, displaying the context bank registers for stage 1 or stage 2.



The popup menu in the **SMMU.StreamMapTable** window provides convenient shortcuts to the following commands:

Popup Menu	Command
Stage 1 Page Table > Stage 2 Page Table >	(--)
<ul style="list-style-type: none"> <li>List</li> <li>Dump</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">SMMU.StreamMapRegGrp.List</a></li> <li><a href="#">SMMU.StreamMapRegGrp.Dump</a></li> </ul>
Peripherals >	(--)
<ul style="list-style-type: none"> <li>Global Configuration Registers</li> <li>Stream Mapping Registers</li> <li>Context Bank Registers of Stage 1 and Context Bank Registers of Stage 2</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">SMMU.Register.Global</a></li> <li><a href="#">SMMU.Register.StreamMapRegGrp</a></li> <li><a href="#">SMMU.Register.ContextBank</a></li> </ul>
Security State Determination Table (SSD)	<a href="#">SMMU.SSDtable</a>

Column Name	Description
<b>stream map reg. grp</b>	<ul style="list-style-type: none"> <li><b>visibility:</b> The column is only visible if the SMMU supports the two security states <i>secure</i> and <i>non-secure</i>.  The label <b>sec/nsec</b> indicates that the SMRG is visible to secure and non-secure accesses.  The label <b>sec only</b> indicates that the SMRG is visible to secure accesses only.</li> <li><b>index:</b> The index numbers start at <b>0x00</b> and are incremented by 1 per SMRG.</li> </ul>
<b>stream matching</b>	See description of the columns <b>ref. id</b> , <b>id mask</b> , and <b>valid</b> below.
<b>ref. id, id mask, and valid</b>	If the SMMU supports <i>stream matching</i> , then the following columns are visible: <b>ref. id</b> , <b>id mask</b> , and <b>valid</b> . Otherwise, these columns are hidden.
<b>context type</b>	Depending on the translation context of a stream mapping register group, the following values are displayed <a href="#">[Description of Values]</a> : <ul style="list-style-type: none"> <li>• s2 translation only</li> <li>• s1 trsl - s2 trsl</li> <li>• s1 trsl - s2 fault</li> <li>• s1 trsl - s2 byp</li> <li>• fault (s1 trsl-s2 trsl)</li> <li>• fault (s1 trsl-s2 flt)</li> <li>• fault (s1 trsl-s2 byp)</li> <li>• fault</li> <li>• bypass mode</li> <li>• reserved</li> <li>• HYPIC or MONC</li> </ul>
<b>stage 1 pagetbl. fmt</b> or <b>stage 2 pagetbl. fmt</b>	Displays the page table format of <b>stage 1</b> or <b>stage 2</b> <a href="#">[Description of Values]</a> : <ul style="list-style-type: none"> <li>• Short descr. (32-bit ARM architecture only)</li> <li>• Long descr. (32-bit ARM architecture only)</li> <li>• AArch32 Shrt (64-bit ARM architecture only)</li> <li>• AArch32 Long (64-bit ARM architecture only)</li> <li>• AArch64 Long (64-bit ARM architecture only)</li> </ul>
<b>cbndx</b>	Displays the context bank index (cbndx) associated with the translation context of <b>stage 1</b> or <b>stage 2</b> .

Column Name	Description
<b>state</b>	<p>Displays whether the MMU of <b>stage 1</b> or <b>stage 2</b> is enabled (ON) or disabled (OFF) and whether a fault has occurred in a translation context bank:</p> <ul style="list-style-type: none"> <li>• <b>F</b>: any single fault</li> <li>• <b>M</b>: multiple faults</li> <li>• <b>S</b>: the SMMU is stalled</li> </ul> <p>The letters F, M, and S are highlighted in red in the <b>SMMU.StreamMapTable</b> window (<a href="#">example</a>).</p> <p>The information about the faults is derived from the register SMMU_CbN_FSR (fault status register of the context bank).</p> <p>Double-click the respective <b>state</b> cell to open the <a href="#">SMMU.StreamMapRegGrp.ContextReg</a> window. The register SMMU_CbN_FSR provides details about the fault.</p>

Values in the Column “context type”	Description
<b>s2 translation only</b>	Context defines a stage 2 translation only
<b>s1 trsl - s2 trsl</b>	Context defines a stage 1 translation, followed by a stage 2 translation (nested translation)
<b>s1 trsl - s2 fault</b>	Context defines a stage 1 translation followed by a stage 2 fault
<b>s1 trsl - s2 byp</b>	Context defines a stage 1 translation followed by a stage 2 bypass
<b>fault (s1 trsl-s2 trsl)</b>	Context defines a stage 1 translation followed by a stage 2 translation, but SMMU has no stage 1 (SMMU configuration fault)
<b>fault (s1 trsl-s2 flt)</b>	Context defines a stage 1 translation followed by a stage 2 fault, but SMMU has no stage 1 (SMMU configuration fault)
<b>fault (s1 trsl-s2 byp)</b>	Context defines a stage 1 translation followed by a stage 2 bypass, but SMMU has no stage 1 (SMMU configuration fault)
<b>fault</b>	Context defines a fault
<b>bypass mode</b>	Context defines bypass mode
<b>reserved</b>	Context type is improperly defined
<b>HYPC</b>	Is displayed on the right-hand side of the column if the context is a hypervisor context.
<b>MONC</b>	Is displayed on the right-hand side of the column if the context is a monitor context.

Values in the Columns “stage 1 pagetbl. fmt” “stage 2 pagetbl. fmt”	Description
<b>Short descr.</b>	Page table uses the 32-bit short descriptor format (32-bit targets only)
<b>Long descr.</b>	Page table uses the 32-bit long descriptor (LPAE) format (32-bit targets only)
<b>AArch32 Shrt</b>	Page table uses the 32-bit short descriptor format (64-bit targets only)
<b>AArch32 Long</b>	Page table uses the 32-bit long descriptor (LPAE) format (64-bit targets only)
<b>AArch64 Long</b>	Page table uses the 64-bit long descriptor (LPAE) format (64-bit targets only)

Codes in the gray window status bar at the bottom of the [SMMU.StreamMapTable](#) window indicate the current global fault status of the SMMU. These codes for the global faults are MULTI, UUT, PF, EF, CAF, UCIF, UCBF, SMCF, USF, ICF [A].

## To view the descriptions of the global faults:

1. Double-click the gray window status bar to open the [SMMU.Register.Global](#) window [A].
2. Search for this register: SMMU\_sGFSR [B]  
The global faults are described in the column on the right [C].

stream map visibility	reg. grp index	stream ref. id	matching id mask	valid	context type	stage 1 pagetbl. fmt	cbndx	state	stage 2 pagetbl. fmt	cbndx	state
sec/nsec	0x06	0x0B4C	0x7000	yes	bypass mode						
sec/nsec	0x07	0x0000	0x0000	no	fault						
sec/nsec	0x08	0x0000	0x0000	no	fault						
sec/nsec	0x09	0x0341	0x7000	yes	s1 trsl - s2 byp	AArch32 Shrt	0x12	on			
sec/nsec	0x0A	0x0E55	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x14	on			
sec/nsec	0x0B	0x0464	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x16	<b>F</b> on	AArch64 Long	0x15	on
sec/nsec	0x0C	0x00FB	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x18	on	AArch64 Long	0x19	on
sec/nsec	0x0D	0x0587	0x7000	yes	s1 trsl - s2 trsl	AArch64 Long	0x1A	on	AArch64 Long	0x1B	on
sec only	0x0E	0x0000	0x0000	no	fault						
sec only	0x0F	0x0000	0x0000	no	fault						
sec only	0x10	0x0000	0x0000	no	fault						
sec only	0x11	0x0000	0x0000	no	fault						

MMU-500 base AZSD:0x500 **A** MULTI UUT PF EF CAF UCIF UCBF SMCF USF ICF

SMMU_IDR7	MAJOR	MINOR
SMMU_sGFAR	00000000	0
<b>SMMU_sGFSR</b>	800001FF	

**B**

**C** Multiple faults occurred  
Unsupported upstream transaction fault recorded  
Permission fault  
External fault caused by an external abort  
Configuration access fault  
Unimplemented context interrupt fault  
Unimplemented context bank fault  
Stream match conflict fault  
Unidentified stream fault  
Invalid context fault

**A** Codes of global faults.

**B** The information about the global faults is derived from the register SMMU\_sGFSR (secure global fault status register).

**C** Descriptions of the global faults in the [SMMU.Register.Global](#) window.

### NOTE:

A red letter in a **state** column of the [SMMU.StreamMapTable](#) window indicates a fault in a context bank. For descriptions of these faults, see [state](#) column.



# 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 SWDP [ON | OFF]** alternatively 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

---

Adaption for ARM Debug Cable: See <http://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 JTAG Interface Specifications**” (app\_arm\_jtag.pdf).

### CombiProbe

---

Adaption for ARM CombiProbe: See <http://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

---

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

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

Adaption for ARM ETM Preprocessor HSSTP: See <http://www.lauterbach.com/adetmhsstp.html>.

## Available Tools

### ARM7

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
AD6522			YES				YES
AD6526			YES				YES
AD6528			YES				YES
AD6529			YES				YES
AD6532			YES				YES
ADUC7020			YES				YES
ADUC7021			YES				YES
ADUC7022			YES				YES
ADUC7023			YES				YES
ADUC7024			YES				YES
ADUC7025			YES				YES
ADUC7026			YES				YES
ADUC7027			YES				YES
ADUC7028			YES				YES
ADUC7029			YES				YES
ADUC7030			YES				YES
ADUC7032			YES				YES
ADUC7033			YES				YES
ADUC7034			YES				YES
ADUC7036			YES				YES
ADUC7039			YES				YES
ADUC7060			YES				YES
ADUC7061			YES				YES
ADUC7121			YES				YES
ADUC7122			YES				YES
ADUC7124			YES				YES
ADUC7128			YES				YES
ADUC7129			YES				YES
ADUC7229			YES				YES
ARM710T			YES		YES		YES
ARM720T			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
ARM740T			YES		YES		YES
ARM7DI			YES		YES		YES
ARM7TDMI			YES		YES		YES
ARM7TDMI-S			YES		YES		YES
AT75C220			YES				YES
AT75C310			YES				YES
AT75C320			YES				YES
AT76C501			YES				YES
AT76C502			YES				YES
AT76C502A			YES				YES
AT76C503			YES				YES
AT76C503A			YES				YES
AT76C510			YES				YES
AT76C551			YES				YES
AT76C901			YES				YES
AT78C1501			YES				YES
AT91CAP7E			YES				YES
AT91CAP7S250A			YES				YES
AT91CAP7S450A			YES				YES
AT91F40416			YES				YES
AT91F40816			YES				YES
AT91FR40162			YES				YES
AT91FR4042			YES				YES
AT91FR4081			YES				YES
AT91M40100			YES				YES
AT91M40400			YES				YES
AT91M40403			YES				YES
AT91M40800			YES				YES
AT91M40807			YES				YES
AT91M42800A			YES				YES
AT91M43300			YES				YES
AT91M55800A			YES				YES
AT91M63200			YES				YES
AT91R40008			YES				YES
AT91R40807			YES				YES
AT91RM3400			YES				YES
AT91SAM7A1			YES				YES
AT91SAM7A2			YES				YES
AT91SAM7A3			YES				YES
AT91SAM7L128			YES				YES
AT91SAM7L64			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
AT91SAM7S128			YES				YES
AT91SAM7S256			YES				YES
AT91SAM7S32			YES				YES
AT91SAM7S321			YES				YES
AT91SAM7S512			YES				YES
AT91SAM7S64			YES				YES
AT91SAM7SE256			YES				YES
AT91SAM7SE32			YES				YES
AT91SAM7SE512			YES				YES
AT91SAM7X128			YES				YES
AT91SAM7X256			YES				YES
AT91SAM7X512			YES				YES
AT91SAM7XC128			YES				YES
AT91SAM7XC256			YES				YES
AT91SAM7XC512			YES				YES
AT91SC321RC			YES				YES
BC6911			YES				YES
BERYLLIUM			YES				YES
BU7611AKU			YES				YES
CBC32XXA			YES				YES
CDC3207G			YES		YES		YES
CDC3272G			YES		YES		YES
CDC32XXG			YES				YES
CDMAX			YES				YES
CEA32XXA			YES				YES
CL-PS7110			YES				YES
CL-PS7111			YES				YES
CL-PS7500FE			YES				YES
CL-SH8665			YES				YES
CL-SH8668			YES				YES
CLARITY			YES				YES
CS22210			YES				YES
CS22220			YES				YES
CS22230			YES				YES
CS22250			YES				YES
CS22270			YES				YES
CS89712			YES				YES
CSM5000			YES				YES
CSM5200			YES				YES
CX81210			YES				YES
CX81400			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
D5205			YES				YES
D5313			YES				YES
D5314			YES				YES
EASYCAN1			YES				YES
EASYCAN2			YES				YES
EASYCAN4			YES				YES
EP7209			YES				YES
EP7211			YES				YES
EP7212			YES				YES
EP7309			YES				YES
EP7311			YES				YES
EP7312			YES				YES
EP7339			YES				YES
EP7407			YES				YES
GMS30C7201			YES				YES
GP4020			YES				YES
HELIUM_100			YES				YES
HELIUM_200			YES				YES
HELIUM_210			YES				YES
HMS30C7202			YES				YES
HMS31C2816			YES				YES
HMS39C70512			YES				YES
HMS39C7092			YES				YES
IXP220			YES				YES
IXP225			YES				YES
KS17C40025			YES				YES
KS17F80013			YES				YES
KS32C61100			YES				YES
KS32P6632			YES				YES
L64324			YES				YES
L7200			YES				YES
L7205			YES				YES
L7210			YES				YES
LH75400			YES				YES
LH75401			YES				YES
LH75410			YES				YES
LH75411			YES				YES
LH77790			YES				YES
LH79520			YES				YES
LITHIUM			YES				YES
LOGIC_CBP3.0			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
LOGIC_CBP4.0			YES				YES
LOGIC_L64324			YES				YES
LPC2101			YES		YES		YES
LPC2102			YES		YES		YES
LPC2103			YES		YES		YES
LPC2104			YES		YES		YES
LPC2105			YES		YES		YES
LPC2106			YES		YES		YES
LPC2109			YES		YES		YES
LPC2112			YES		YES		YES
LPC2114			YES		YES		YES
LPC2119			YES		YES		YES
LPC2124			YES		YES		YES
LPC2129			YES		YES		YES
LPC2131			YES		YES		YES
LPC2131/01			YES		YES		YES
LPC2132			YES		YES		YES
LPC2132/01			YES		YES		YES
LPC2134			YES		YES		YES
LPC2134/01			YES		YES		YES
LPC2136			YES		YES		YES
LPC2136/01			YES		YES		YES
LPC2138			YES		YES		YES
LPC2138/01			YES		YES		YES
LPC2141			YES		YES		YES
LPC2142			YES		YES		YES
LPC2144			YES		YES		YES
LPC2146			YES		YES		YES
LPC2148			YES		YES		YES
LPC2194			YES		YES		YES
LPC2210			YES		YES		YES
LPC2212			YES		YES		YES
LPC2214			YES		YES		YES
LPC2220			YES		YES		YES
LPC2290			YES		YES		YES
LPC2292			YES		YES		YES
LPC2294			YES		YES		YES
LPC2364			YES		YES		YES
LPC2365			YES		YES		YES
LPC2366			YES		YES		YES
LPC2367			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
LPC2368			YES		YES		YES
LPC2377			YES		YES		YES
LPC2378			YES		YES		YES
LPC2387			YES		YES		YES
LPC2388			YES		YES		YES
LPC2458			YES		YES		YES
LPC2460			YES		YES		YES
LPC2468			YES		YES		YES
LPC2470			YES		YES		YES
LPC2478			YES		YES		YES
LPC2880			YES		YES		YES
LPC2888			YES		YES		YES
M4641			YES				YES
MAC7101			YES				YES
MAC7111			YES				YES
MAC7116			YES				YES
MAC7121			YES				YES
MAC7131			YES				YES
MAC7141			YES				YES
MKY-82A			YES				YES
MKY-85			YES				YES
ML670100			YES				YES
ML671000			YES				YES
ML674000			YES				YES
ML674001			YES				YES
ML674080			YES				YES
ML675001			YES				YES
ML675200			YES				YES
ML675300			YES				YES
ML67Q2300			YES				YES
ML67Q2301			YES				YES
ML67Q4002			YES				YES
ML67Q4003			YES				YES
ML67Q4100			YES				YES
ML67Q5002			YES				YES
ML67Q5003			YES				YES
ML67Q5200			YES				YES
ML67Q5300			YES				YES
ML70511LA			YES				YES
ML7051LA			YES				YES
MN1A7T0200			YES				YES



CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MODEM			YES				YES
MSM3000			YES				YES
MSM3100			YES				YES
MSM3300			YES				YES
MSM5000			YES				YES
MSM5100			YES				YES
MSM5105			YES				YES
MSM5200			YES				YES
MSM5500			YES				YES
MSM6000			YES				YES
MSM6050			YES				YES
MSM6200			YES				YES
MSM6600			YES				YES
MSP1000			YES				YES
MT1020A			YES				YES
MT92101			YES				YES
MTC-20276			YES				YES
MTC-20277			YES				YES
MTC-30585			YES				YES
MTK-20141			YES				YES
MTK-20280			YES				YES
MTK-20285			YES				YES
NET+15			YES				YES
NET+20			YES				YES
NET+40			YES				YES
NET+50			YES				YES
NITROGEN			YES				YES
NS7520			YES				YES
OMAPV2230			YES		YES		YES
PBM_990_90			YES				YES
PCC-ISES			YES				YES
PCD80703			YES		YES		YES
PCD80705			YES		YES		YES
PCD80708			YES		YES		YES
PCD80715			YES		YES		YES
PCD80716			YES		YES		YES
PCD80718			YES		YES		YES
PCD80720			YES		YES		YES
PCD80721			YES		YES		YES
PCD80725			YES		YES		YES
PCD80727			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
PCD80728			YES		YES		YES
PCF26002			YES				YES
PCF26003			YES				YES
PCF87750			YES				YES
PCI2010			YES				YES
PCI3610			YES				YES
PCI3620			YES				YES
PCI3700			YES				YES
PCI3800			YES				YES
PCI5110			YES				YES
PCI9501			YES				YES
PH21101			YES				YES
PMB7754			YES				YES
PS7500FE			YES				YES
PUC3030A			YES				YES
PUC303XA			YES				YES
S3C3400A			YES				YES
S3C3400X			YES				YES
S3C3410X			YES				YES
S3C44A0A			YES				YES
S3C44B0X			YES				YES
S3C4510B			YES				YES
S3C4520A			YES				YES
S3C4530A			YES				YES
S3C4610D			YES				YES
S3C4620D			YES				YES
S3C4640X			YES				YES
S3C4650D			YES				YES
S3C46C0			YES				YES
S3C46M0X			YES				YES
S3C4909A			YES				YES
S3C49F9X			YES				YES
S3F401F			YES				YES
S3F441FX			YES				YES
S3F460H			YES				YES
S5N8946			YES				YES
S5N8947			YES				YES
SC100			YES				YES
SC110			YES				YES
SIRFSTARII			YES				YES
SJA2020			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
SOCLITE+			YES				YES
ST30F7XXA			YES		YES		YES
ST30F7XXC			YES		YES		YES
ST30F7XXZ			YES		YES		YES
STA2051			YES				YES
STR710			YES				YES
STR711			YES				YES
STR712			YES				YES
STR715			YES				YES
STR720			YES		YES		YES
STR730			YES				YES
STR731			YES				YES
STR735			YES				YES
STR736			YES				YES
STR750FV			YES				YES
STR751FR			YES				YES
STR752FR			YES				YES
STR755FR			YES				YES
STR755FV			YES				YES
STW2400			YES				YES
TA7S05			YES				YES
TA7S12			YES				YES
TA7S20			YES				YES
TA7S32			YES				YES
TMS320VC5470			YES				YES
TMS320VC5471			YES				YES
TMS470PVF241			YES				YES
TMS470PVF341			YES				YES
TMS470PVF344			YES				YES
TMS470PVF345			YES				YES
TMS470PVF346			YES				YES
TMS470PVF347			YES				YES
TMS470PVF348			YES				YES
TMS470Q			YES				YES
TMS470R1A128			YES				YES
TMS470R1A256			YES				YES
TMS470R1A288			YES				YES
TMS470R1A384			YES				YES
TMS470R1A64			YES				YES
TMS470R1B1M			YES				YES
TMS470R1B512			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
TMS470R1B768			YES				YES
TMS470R1VC336A			YES				YES
TMS470R1VC338			YES				YES
TMS470R1VC346A			YES				YES
TMS470R1VC348			YES				YES
TMS470R1VC688			YES				YES
TMS470R1VF288			YES				YES
TMS470R1VF334A			YES				YES
TMS470R1VF336			YES				YES
TMS470R1VF336A			YES				YES
TMS470R1VF338			YES				YES
TMS470R1VF346			YES				YES
TMS470R1VF346A			YES				YES
TMS470R1VF348			YES				YES
TMS470R1VF356A			YES				YES
TMS470R1VF37A			YES				YES
TMS470R1VF448			YES				YES
TMS470R1VF45A			YES				YES
TMS470R1VF45AA			YES				YES
TMS470R1VF45B			YES				YES
TMS470R1VF45BA			YES				YES
TMS470R1VF478			YES				YES
TMS470R1VF48B			YES				YES
TMS470R1VF48C			YES				YES
TMS470R1VF4B8			YES				YES
TMS470R1VF55B			YES				YES
TMS470R1VF55BA			YES				YES
TMS470R1VF67A			YES				YES
TMS470R1VF688			YES				YES
TMS470R1VF689			YES				YES
TMS470R1VF76B			YES				YES
TMS470R1VF7AC			YES				YES
UPD65977			YES				YES
UPLAT_CORE			YES				YES
VCS94250			YES				YES
VMS747			YES				YES
VWS22100			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
VWS22110			YES				YES
VWS23112			YES				YES
VWS23201			YES				YES
VWS23202			YES				YES
VWS26001			YES				YES

## ARM9

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
88AP128			YES				YES
88AP162			YES				YES
88AP166			YES				YES
88AP168			YES				YES
88E6208			YES				YES
88E6218			YES				YES
88F5082			YES				YES
88F5180N			YES				YES
88F5181			YES				YES
88F5181L			YES				YES
88F5182			YES				YES
88F5281			YES				YES
88F6082			YES				YES
88F6180			YES				YES
88F6183			YES				YES
88F6183L			YES				YES
88F6190			YES				YES
88F6192			YES				YES
88F6280			YES				YES
88F6281			YES				YES
88F6282			YES				YES
88F6283			YES				YES
88F6321			YES				YES
88F6322			YES				YES
88F6323			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
88F6601			YES				YES
88FR101			YES		YES		YES
88FR102			YES		YES		YES
88FR111			YES		YES		YES
88FR131			YES				YES
88FR301			YES		YES		YES
88FR321			YES				YES
88FR331			YES				YES
88FR521			YES				YES
88FR531			YES				YES
88FR571			YES				YES
88I6745			YES				YES
AAEC-2000			YES				YES
AM1707			YES				YES
AM1808			YES				YES
AM1810			YES				YES
AM3872			YES				YES
AM3874			YES				YES
AM3892			YES				YES
AM3894			YES				YES
ARM7EJ-S			YES		YES		YES
ARM915T			YES		YES		YES
ARM920T			YES		YES		YES
ARM922T			YES		YES		YES
ARM926EJ-S			YES		YES		YES
ARM940T			YES		YES		YES
ARM946E-S			YES		YES		YES
ARM966E-S			YES		YES		YES
ARM968E-S			YES		YES		YES
ARM9E-S			YES		YES		YES
ARM9EJ-S			YES		YES		YES
ARM9TDMI			YES		YES		YES
AT91CAP9E			YES				YES
AT91CAP9EC			YES				YES
AT91CAP9S250A			YES				YES
AT91CAP9S500A			YES				YES
AT91CAP9SC250A			YES				YES
AT91CAP9SC500A			YES				YES
AT91RM9200			YES		YES		YES
AT91SAM9260			YES				YES
AT91SAM9261			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
AT91SAM9263			YES		YES		YES
AT91SAM9G10			YES				YES
AT91SAM9G20			YES				YES
AT91SAM9G45			YES				YES
AT91SAM9M10			YES				YES
AT91SAM9R64			YES				YES
AT91SAM9RL64			YES				YES
AT91SAM9XE128			YES				YES
AT91SAM9XE256			YES				YES
AT91SAM9XE512			YES				YES
CN9414			YES				YES
CX22490			YES				YES
CX22491			YES				YES
CX22492			YES				YES
CX22496			YES				YES
CX82100			YES				YES
CYUSB2014			YES				YES
CYUSB3011			YES				YES
CYUSB3012			YES				YES
CYUSB3013			YES				YES
CYUSB3014			YES				YES
CYUSB3031			YES				YES
CYUSB3033			YES				YES
CYUSB3035			YES				YES
DB5500			YES		YES		YES
DRA6XX			YES		YES		YES
DRA71X			YES		YES		
DRA72X			YES		YES		
DRA74X			YES		YES		
DRA74XP			YES		YES		
DRA75X			YES		YES		
DRA75XP			YES		YES		
DRA76XP			YES		YES		
DRA77XP			YES		YES		
DRX401			YES				YES
DRX402			YES				YES
DRX403			YES				YES
DRX404			YES				YES
DRX406			YES				YES
DRX407			YES				YES
DRX414			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
DRX416			YES				YES
DRX440			YES				YES
DRX442			YES				YES
DRX443			YES				YES
DRX444			YES				YES
DRX445			YES				YES
DRX446			YES				YES
DRX447			YES				YES
DRX449			YES				YES
DRX453			YES				YES
DRX457			YES				YES
DRX459			YES				YES
ECONA_CNS1101			YES				YES
ECONA_CNS1102			YES				YES
ECONA_CNS1104			YES				YES
ECONA_CNS1105			YES				YES
ECONA_CNS1109			YES				YES
ECONA_CNS1202			YES				YES
ECONA_CNS1205			YES				YES
ECONA_CNS2131			YES				YES
ECONA_CNS2132			YES				YES
ECONA_CNS2133			YES				YES
ECONA_CNS2181			YES				YES
ECONA_CNS2182X			YES				YES
EP9301			YES				YES
EP9307			YES				YES
EP9312			YES				YES
EP9315			YES				YES
EPXA1			YES		YES		YES
EPXA10			YES		YES		YES
EPXA4			YES		YES		YES
ERTEC200			YES		YES		YES
ERTEC400			YES		YES		YES
FA526			YES				YES
FA606TE			YES				YES
FA616TE			YES				YES
FA626			YES				YES
FA626TE			YES				YES
HELIUM_500			YES				YES
IMX23			YES		YES		YES
IMX25			YES		YES		YES



CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
IMX27			YES		YES		YES
IMX27L			YES		YES		YES
IMX280			YES		YES		YES
IMX281			YES		YES		YES
IMX283			YES		YES		YES
IMX285			YES		YES		YES
IMX286			YES		YES		YES
IMX287			YES		YES		YES
INFOSTREAM			YES				YES
KIRA100			YES				YES
LH7A400			YES				YES
LH7A404			YES				YES
LH7A405			YES				YES
LPC2915			YES				YES
LPC2917			YES				YES
LPC2917/01			YES				YES
LPC2919			YES				YES
LPC2919/01			YES				YES
LPC2921			YES				YES
LPC2923			YES				YES
LPC2925			YES				YES
LPC2926			YES				YES
LPC2930			YES				YES
LPC2939			YES				YES
LPC3000			YES				YES
LPC3130			YES				YES
LPC3131			YES				YES
LPC3141			YES				YES
LPC3143			YES				YES
LPC3152			YES				YES
LPC3154			YES				YES
LPC3180			YES				YES
LPC3220			YES				YES
LPC3230			YES				YES
LPC3240			YES				YES
LPC3250			YES				YES
MB86R01			YES		YES		YES
MB86R02			YES		YES		YES
MB86R03			YES		YES		YES
MC9328MX1			YES		YES		YES
MC9328MX21			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MC9328MX21S			YES				YES
MC9328MXL			YES		YES		YES
MC9328MXS			YES		YES		YES
ML67Q2003			YES				YES
MSM6100_3G			YES		YES		YES
MSM6250			YES		YES		YES
MSM6300			YES		YES		YES
MSM6500			YES		YES		YES
MV76100			YES				YES
MV78100			YES				YES
MV78200			YES				YES
NETX100			YES		YES		YES
NETX50			YES		YES		YES
NETX500			YES		YES		YES
NETX51			YES		YES		YES
NEXPERIA			YES				YES
NS9210			YES				YES
NS9215			YES				YES
NS9360			YES				YES
NS9750			YES				YES
NS9775			YES				YES
OMAP-L137			YES				YES
OMAP-L138			YES				YES
OMAP1510			YES		YES		YES
OMAP1610			YES		YES		YES
OMAP1611			YES		YES		YES
OMAP1612			YES		YES		YES
OMAP1710			YES		YES		YES
OMAP310			YES		YES		YES
OMAP331			YES		YES		YES
OMAP3430			YES		YES		YES
OMAP3440			YES		YES		YES
OMAP3630			YES		YES		YES
OMAP3640			YES		YES		YES
OMAP4430			YES		YES		YES
OMAP4460			YES		YES		YES
OMAP4470			YES		YES		YES
OMAP5430			YES		YES		YES
OMAP5432			YES		YES		YES
OMAP5910			YES		YES		YES
OMAP5912			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
OMAP710			YES		YES		YES
OMAP730			YES		YES		YES
OMAP732			YES		YES		YES
OMAP733			YES		YES		YES
OMAP750			YES		YES		YES
OMAP850			YES		YES		YES
OMAPV1030			YES		YES		YES
OMAPV1035			YES		YES		YES
OMAPV2230			YES		YES		YES
PMB8870			YES		YES		YES
PMB8875			YES		YES		YES
PMB8876			YES		YES		YES
PMB8877			YES		YES		YES
PMB8878			YES		YES		YES
PMB8888			YES		YES		YES
PXA910			YES				YES
PXA920			YES				YES
S3C2400X			YES				YES
S3C2410			YES				YES
S3C2410X			YES				YES
S3C2416			YES				YES
S3C2440A			YES				YES
S3C2442B			YES				YES
S3C2443X			YES				YES
S3C2450			YES				YES
S3C2500A			YES				YES
S3C2510			YES				YES
S3C2800X			YES				YES
SC200			YES		YES		YES
SC210			YES		YES		YES
SCORPIO			YES				YES
SP2503			YES				YES
SP2506			YES				YES
SP2512			YES				YES
SPEAR300			YES		YES		YES
SPEAR310			YES		YES		YES
SPEAR320			YES		YES		YES
SPEAR320S			YES		YES		YES
SPEAR600			YES		YES		YES
STN8810			YES		YES		YES
STN8815			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
STN8820			YES		YES		YES
STR910FAM32			YES		YES		YES
STR910FAW32			YES		YES		YES
STR910FAZ32			YES		YES		YES
STR911FAM42			YES		YES		YES
STR911FAM44			YES		YES		YES
STR911FAM46			YES		YES		YES
STR911FAM47			YES		YES		YES
STR911FAW42			YES		YES		YES
STR911FAW44			YES		YES		YES
STR911FAW46			YES		YES		YES
STR911FAW47			YES		YES		YES
STR912FAW42			YES		YES		YES
STR912FAW44			YES		YES		YES
STR912FAW46			YES		YES		YES
STR912FAW47			YES		YES		YES
STR912FAZ42			YES		YES		YES
STR912FAZ44			YES		YES		YES
STR912FAZ46			YES		YES		YES
STR912FAZ47			YES		YES		YES
T6TC1XB-0001			YES				YES
T8300			YES				YES
T8302			YES				YES
TDA2EX			YES		YES		
TDA2PX			YES		YES		
TDA2X			YES		YES		
TMPA900			YES				YES
TMPA901			YES				YES
TMPA910			YES				YES
TMPA911			YES				YES
TMPA912			YES				YES
TMS320C6A8143			YES				YES
TMS320C6A8147			YES				YES
TMS320C6A8148			YES				YES
TMS320C6A8167			YES				YES
TMS320C6A8168			YES				YES
TMS320DA828			YES				YES
TMS320DA830			YES				YES
TMS320DM335			YES				YES
TMS320DM355			YES				YES
TMS320DM357			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
TMS320DM365			YES				YES
TMS320DM368			YES				YES
TMS320DM6441			YES				YES
TMS320DM6443			YES				YES
TMS320DM6446			YES				YES
TMS320DM6467			YES				YES
TMS320DM8147			YES				YES
TMS320DM8148			YES				YES
TMS320DM8165			YES				YES
TMS320DM8166			YES				YES
TMS320DM8167			YES				YES
TMS320DM8168			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
ARM1020E			YES	YES	YES		YES
ARM1022E			YES	YES	YES		YES
ARM1026EJ-S			YES	YES	YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
88SV581X-V6			YES				YES
ARM1136J-S			YES		YES		YES
ARM1136JF-S			YES		YES		YES
ARM1156T2-S			YES		YES		YES
ARM1156T2F-S			YES		YES		YES
ARM1176JZ-S			YES		YES		YES
ARM1176JZF-S			YES		YES		YES
ARM11MPCORE			YES		YES		YES
BCM2835			YES				YES
IMX31			YES		YES		YES
IMX35			YES		YES		YES
IMX351			YES		YES		YES
IMX353			YES		YES		YES
IMX355			YES		YES		YES
IMX356			YES		YES		YES
IMX357			YES		YES		YES
IMX37			YES		YES		YES
MB86H60			YES				YES
MV78130V6			YES				YES
MV78160V6			YES				YES
MV78230V6			YES				YES
MV78260V6			YES				YES
MV78460V6			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MXC91131			YES		YES		
MXC91231			YES		YES		
MXC91321			YES		YES		
MXC91323			YES		YES		
MXC91331			YES		YES		
OMAP2420			YES		YES		YES
OMAP2430			YES		YES		YES
OMAP2431			YES		YES		YES
OMAPV2230			YES		YES		YES
S3C6400			YES				YES
S3C6410			YES				YES
SP2603			YES				YES
SP2606			YES				YES
SP2612			YES				YES
SP2704			YES		YES		YES
SP2716			YES		YES		YES
STA2064			YES		YES		YES
STA2065			YES		YES		YES
STA2164			YES		YES		YES
STA2165			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
66AK2E02			YES		YES		YES
66AK2E05			YES		YES		YES
66AK2H06			YES		YES		YES
66AK2H12			YES		YES		YES
66AK2H14			YES		YES		YES
88F6707			YES				YES
88F6710			YES				YES
88F6810			YES				YES
88F6820			YES				YES
88F6828			YES				YES
88F6W11			YES				YES
A20			YES				YES
A9500			YES		YES		YES
A9540			YES		YES		YES
ADSP-SC570			YES		YES		YES
ADSP-SC571			YES		YES		YES
ADSP-SC572			YES		YES		YES
ADSP-SC573			YES		YES		YES
ADSP-SC582			YES		YES		YES
ADSP-SC583			YES		YES		YES
ADSP-SC583W			YES		YES		YES
ADSP-SC584			YES		YES		YES
ADSP-SC584W			YES		YES		YES
ADSP-SC587			YES		YES		YES
ADSP-SC589			YES		YES		YES
AM3352			YES				YES
AM3354			YES				YES
AM3356			YES				YES
AM3357			YES				YES
AM3358			YES				YES
AM3359			YES				YES
AM3505			YES		YES		YES
AM3517			YES		YES		YES
AM3703			YES		YES		YES
AM3715			YES		YES		YES
AM3872			YES				YES
AM3874			YES				YES
AM3892			YES				YES
AM3894			YES				YES



CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
AM4372			YES		YES		YES
AM4376			YES		YES		YES
AM4377			YES		YES		YES
AM4378			YES		YES		YES
AM4379			YES		YES		YES
AM571X			YES		YES		YES
AM572X			YES		YES		YES
AM574X			YES		YES		YES
AM5K2E02			YES		YES		YES
AM5K2E04			YES		YES		YES
ARRIA10SOC			YES		YES		YES
ARRIAVSOC			YES		YES		YES
ATSAMA5D21			YES				YES
ATSAMA5D22			YES				YES
ATSAMA5D23			YES				YES
ATSAMA5D24			YES				YES
ATSAMA5D26			YES				YES
ATSAMA5D27			YES				YES
ATSAMA5D28			YES				YES
ATSAMA5D31			YES				YES
ATSAMA5D33			YES				YES
ATSAMA5D34			YES				YES
ATSAMA5D35			YES				YES
ATSAMA5D36			YES				YES
ATSAMA5D41			YES				YES
ATSAMA5D42			YES				YES
ATSAMA5D43			YES				YES
ATSAMA5D44			YES				YES
AXM5516			YES				YES
BCM2836			YES				YES
BCM4708			YES		YES		YES
BCM47081			YES		YES		YES
CORTEX-A15			YES		YES		YES
CORTEX-A17			YES		YES		YES
CORTEX-A5			YES		YES		YES
CORTEX-A7			YES		YES		YES
CORTEX-A8			YES		YES		YES
CORTEX-A9			YES		YES		YES
CORTEX-R4			YES		YES		YES
CORTEX-R4F			YES		YES		YES
CORTEX-R5			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
CORTEX-R5F			YES		YES		YES
CORTEX-R7			YES		YES		YES
CORTEX-R7F			YES		YES		YES
CORTEX-R8			YES		YES		YES
CORTEX-R8F			YES		YES		YES
CS7522			YES		YES		YES
CS7542			YES		YES		YES
CYCLONEVSOC			YES		YES		YES
DB5500			YES		YES		YES
DB8500			YES		YES		YES
DB8540			YES		YES		YES
DRA6XX			YES		YES		YES
DRA71X			YES		YES		
DRA72X			YES		YES		
DRA74X			YES		YES		
DRA74XP			YES		YES		
DRA75X			YES		YES		
DRA75XP			YES		YES		
DRA76XP			YES		YES		
DRA77XP			YES		YES		
DRA79X			YES		YES		
EXYNOS4212			YES		YES		YES
EXYNOS4412			YES		YES		YES
EXYNOS5250			YES		YES		YES
IMX502			YES		YES		YES
IMX503			YES		YES		YES
IMX507			YES		YES		YES
IMX508			YES		YES		YES
IMX512			YES		YES		YES
IMX513			YES		YES		YES
IMX514			YES		YES		YES
IMX515			YES		YES		YES
IMX516			YES		YES		YES
IMX534			YES		YES		YES
IMX535			YES		YES		YES
IMX536			YES		YES		YES
IMX537			YES		YES		YES
IMX538			YES		YES		YES
IMX6DUAL			YES		YES		YES
IMX6DUALITE			YES		YES		YES
IMX6DUALPLUS			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
IMX6QUAD			YES		YES		YES
IMX6QUADPLUS			YES		YES		YES
IMX6SLL			YES		YES		YES
IMX6SOLO			YES		YES		YES
IMX6SOLOX			YES		YES		YES
IMX6ULL			YES		YES		YES
IMX6ULTRALITE			YES		YES		YES
IMX7DUAL			YES		YES		YES
IMX7SOLO			YES		YES		YES
IMX7ULP			YES		YES		YES
KRAIT			YES		YES		YES
LS1020A			YES		YES		YES
LS1021A			YES		YES		YES
LS1022A			YES		YES		YES
LS2080A			YES				YES
M7400			YES		YES		YES
MB86R11			YES		YES		YES
MB86R11F			YES		YES		YES
MB86R12			YES		YES		YES
MB86R24			YES		YES		YES
MB9DF125			YES		YES		YES
MB9DF126			YES		YES		YES
MB9DF564LAE			YES		YES		YES
MB9DF564LGE			YES		YES		YES
MB9DF564LLE			YES		YES		YES
MB9DF564LQE			YES		YES		YES
MB9DF564MAE			YES		YES		YES
MB9DF564MGE			YES		YES		YES
MB9DF564MLE			YES		YES		YES
MB9DF564MQE			YES		YES		YES
MB9DF565LAE			YES		YES		YES
MB9DF565LGE			YES		YES		YES
MB9DF565LLE			YES		YES		YES
MB9DF565LQE			YES		YES		YES
MB9DF565MAE			YES		YES		YES
MB9DF565MGE			YES		YES		YES
MB9DF565MLE			YES		YES		YES
MB9DF565MQE			YES		YES		YES
MB9DF566LAE			YES		YES		YES
MB9DF566LGE			YES		YES		YES
MB9DF566LLE			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MB9DF566LQE			YES		YES		YES
MB9DF566MAE			YES		YES		YES
MB9DF566MGE			YES		YES		YES
MB9DF566MLE			YES		YES		YES
MB9DF566MQE			YES		YES		YES
MB9EF126			YES		YES		YES
MB9EF226			YES		YES		YES
MV78130V7			YES				YES
MV78160V7			YES				YES
MV78230V7			YES				YES
MV78260V7			YES				YES
MV78460V7			YES				YES
NETX4000			YES				YES
NETX4000RLXD			YES				YES
NETX4100			YES				YES
OLEA-T222			YES		YES		YES
OLEA-T464			YES				YES
OMAP3410			YES		YES		YES
OMAP3420			YES		YES		YES
OMAP3430			YES		YES		YES
OMAP3440			YES		YES		YES
OMAP3503			YES		YES		YES
OMAP3515			YES		YES		YES
OMAP3525			YES		YES		YES
OMAP3530			YES		YES		YES
OMAP3610			YES		YES		YES
OMAP3620			YES		YES		YES
OMAP3630			YES		YES		YES
OMAP3640			YES		YES		YES
OMAP4430			YES		YES		YES
OMAP4460			YES		YES		YES
OMAP4470			YES		YES		YES
OMAP5430			YES		YES		YES
OMAP5432			YES		YES		YES
QSD8250			YES		YES		YES
QSD8650			YES		YES		YES
R7S721000			YES		YES		YES
R7S721001			YES		YES		YES
R7S721010			YES		YES		YES
R7S721011			YES		YES		YES
R7S721020			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
R7S721021			YES		YES		YES
R7S721030			YES		YES		YES
R7S721031			YES		YES		YES
R7S721034			YES		YES		YES
R7S721042			YES		YES		YES
R7S721043			YES		YES		YES
R7S721046			YES		YES		YES
R7S721047			YES		YES		YES
R7S721062			YES		YES		YES
R7S721063			YES		YES		YES
R7S721066			YES		YES		YES
R7S721067			YES		YES		YES
R7S910001			YES		YES		
R7S910002			YES		YES		
R7S910006			YES		YES		
R7S910007			YES		YES		
R7S910011			YES		YES		
R7S910013			YES		YES		
R7S910015			YES		YES		
R7S910016			YES		YES		
R7S910017			YES		YES		
R7S910018			YES		YES		
R7S910025			YES		YES		
R7S910026			YES		YES		
R7S910027			YES		YES		
R7S910028			YES		YES		
R7S910035			YES		YES		
R7S910036			YES		YES		
R7S910101			YES		YES		
R7S910102			YES		YES		
R7S910106			YES		YES		
R7S910107			YES		YES		
R7S910111			YES		YES		
R7S910113			YES		YES		
R7S910115			YES		YES		
R7S910116			YES		YES		
R7S910117			YES		YES		
R7S910118			YES		YES		
R7S910125			YES		YES		
R7S910126			YES		YES		
R7S910127			YES		YES		

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
R7S910128			YES		YES		
R7S910135			YES		YES		
R7S910136			YES		YES		
R8A77420			YES		YES		YES
R8A77430			YES		YES		YES
R8A77440			YES		YES		YES
R8A77450			YES		YES		YES
R8A77470			YES		YES		YES
R8A77790			YES		YES		YES
R8A7790X			YES		YES		YES
R8A7791X			YES		YES		YES
R8A7792X			YES		YES		YES
R8A7793X			YES		YES		YES
R8A77940			YES		YES		YES
R8A77951			YES		YES		YES
R8A77960			YES		YES		YES
R8A77965			YES		YES		YES
R8A77970			YES		YES		YES
R8A77980			YES		YES		YES
R8A77990			YES		YES		YES
R8A77995			YES		YES		YES
R9A06G032			YES				YES
R9A06G033			YES				YES
R9A06G034			YES				YES
RM42L432			YES				YES
RM46L430-PGE			YES				YES
RM46L430-ZWT			YES				YES
RM46L440-PGE			YES				YES
RM46L440-ZWT			YES				YES
RM46L450-PGE			YES				YES
RM46L450-ZWT			YES				YES
RM46L830-PGE			YES				YES
RM46L830-ZWT			YES				YES
RM46L840-ZWT			YES				YES
RM46L850-PGE			YES				YES
RM46L850-ZWT			YES				YES
RM46L852-PGE			YES				YES
RM46L852-ZWT			YES				YES
RM48L530-PGE			YES				YES
RM48L530-ZWT			YES		YES		YES
RM48L540-PGE			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
RM48L540-ZWT			YES		YES		YES
RM48L550-PGE			YES				YES
RM48L550-ZWT			YES		YES		YES
RM48L730-PGE			YES				YES
RM48L730-ZWT			YES		YES		YES
RM48L740-PGE			YES				YES
RM48L740-ZWT			YES		YES		YES
RM48L750-PGE			YES				YES
RM48L750-ZWT			YES		YES		YES
RM48L930-PGE			YES				YES
RM48L930-ZWT			YES		YES		YES
RM48L940-PGE			YES				YES
RM48L940-ZWT			YES		YES		YES
RM48L950-PGE			YES				YES
RM48L950-ZWT			YES		YES		YES
RM48L952-PGE			YES				YES
RM48L952-ZWT			YES		YES		YES
RM57L843-ZWT			YES		YES		YES
S5PV210			YES				YES
S5PV310			YES		YES		YES
S6J3118HAA			YES		YES		YES
S6J3119HAA			YES		YES		YES
S6J311AHAA			YES		YES		YES
S6J311BJAA			YES		YES		YES
S6J311CJAA			YES		YES		YES
S6J311DJAA			YES		YES		YES
S6J311EJAA			YES		YES		YES
S6J3128HAA			YES		YES		YES
S6J3129HAA			YES		YES		YES
S6J312AHAA			YES		YES		YES
S6J323CKU			YES		YES		YES
S6J323CLS			YES		YES		YES
S6J323CLU			YES		YES		YES
S6J324CKS			YES		YES		YES
S6J324CKU			YES		YES		YES
S6J324CLS			YES		YES		YES
S6J324CLU			YES		YES		YES
S6J325CKS			YES		YES		YES
S6J325CKU			YES		YES		YES
S6J325CLS			YES		YES		YES
S6J325CLU			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J326CKS			YES		YES		YES
S6J326CKU			YES		YES		YES
S6J326CLS			YES		YES		YES
S6J326CLU			YES		YES		YES
S6J327CKS			YES		YES		YES
S6J327CKU			YES		YES		YES
S6J327CLS			YES		YES		YES
S6J327CLU			YES		YES		YES
S6J328CKS			YES		YES		YES
S6J328CKU			YES		YES		YES
S6J328CLS			YES		YES		YES
S6J328CLU			YES		YES		YES
S6J32AAKS			YES		YES		YES
S6J32AAKU			YES		YES		YES
S6J32AALS			YES		YES		YES
S6J32AALU			YES		YES		YES
S6J32BAKS			YES		YES		YES
S6J32BAKU			YES		YES		YES
S6J32BALS			YES		YES		YES
S6J32BALU			YES		YES		YES
S6J32CAKS			YES		YES		YES
S6J32CAKU			YES		YES		YES
S6J32CALS			YES		YES		YES
S6J32CALU			YES		YES		YES
S6J32DAKS			YES		YES		YES
S6J32DAKU			YES		YES		YES
S6J32DALS			YES		YES		YES
S6J32DALU			YES		YES		YES
S6J32EEKS			YES		YES		YES
S6J32EELS			YES		YES		YES
S6J32FEKS			YES		YES		YES
S6J32FELS			YES		YES		YES
S6J32GEKS			YES		YES		YES
S6J32GELS			YES		YES		YES
S6J331BHA			YES		YES		YES
S6J331BHB			YES		YES		YES
S6J331BHC			YES		YES		YES
S6J331BHD			YES		YES		YES
S6J331BHS			YES		YES		YES
S6J331BHU			YES		YES		YES
S6J331BJA			YES		YES		YES



CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J331BJB			YES		YES		YES
S6J331BJC			YES		YES		YES
S6J331BJD			YES		YES		YES
S6J331BJS			YES		YES		YES
S6J331BJU			YES		YES		YES
S6J331BKA			YES		YES		YES
S6J331BKB			YES		YES		YES
S6J331BKC			YES		YES		YES
S6J331BKD			YES		YES		YES
S6J331BKS			YES		YES		YES
S6J331BKU			YES		YES		YES
S6J331CHA			YES		YES		YES
S6J331CHB			YES		YES		YES
S6J331CHC			YES		YES		YES
S6J331CHD			YES		YES		YES
S6J331CHS			YES		YES		YES
S6J331CHU			YES		YES		YES
S6J331CJA			YES		YES		YES
S6J331CJB			YES		YES		YES
S6J331CJC			YES		YES		YES
S6J331CJD			YES		YES		YES
S6J331CJS			YES		YES		YES
S6J331CJU			YES		YES		YES
S6J331CKA			YES		YES		YES
S6J331CKB			YES		YES		YES
S6J331CKC			YES		YES		YES
S6J331CKD			YES		YES		YES
S6J331CKS			YES		YES		YES
S6J331CKU			YES		YES		YES
S6J331DHA			YES		YES		YES
S6J331DHB			YES		YES		YES
S6J331DHC			YES		YES		YES
S6J331DHD			YES		YES		YES
S6J331DHS			YES		YES		YES
S6J331DHU			YES		YES		YES
S6J331DJA			YES		YES		YES
S6J331DJB			YES		YES		YES
S6J331DJC			YES		YES		YES
S6J331DJD			YES		YES		YES
S6J331DJS			YES		YES		YES
S6J331DJU			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J331DKA			YES		YES		YES
S6J331DKB			YES		YES		YES
S6J331DKC			YES		YES		YES
S6J331DKD			YES		YES		YES
S6J331DKS			YES		YES		YES
S6J331DKU			YES		YES		YES
S6J331EHA			YES		YES		YES
S6J331EHB			YES		YES		YES
S6J331EHC			YES		YES		YES
S6J331EHD			YES		YES		YES
S6J331EHS			YES		YES		YES
S6J331EHU			YES		YES		YES
S6J331EJA			YES		YES		YES
S6J331EJB			YES		YES		YES
S6J331EJC			YES		YES		YES
S6J331EJD			YES		YES		YES
S6J331EJS			YES		YES		YES
S6J331EJU			YES		YES		YES
S6J331EKA			YES		YES		YES
S6J331EKB			YES		YES		YES
S6J331EKC			YES		YES		YES
S6J331EKD			YES		YES		YES
S6J331EKS			YES		YES		YES
S6J331EKU			YES		YES		YES
S6J332BHA			YES		YES		YES
S6J332BHB			YES		YES		YES
S6J332BHC			YES		YES		YES
S6J332BHD			YES		YES		YES
S6J332BHS			YES		YES		YES
S6J332BHU			YES		YES		YES
S6J332BJA			YES		YES		YES
S6J332BJB			YES		YES		YES
S6J332BJC			YES		YES		YES
S6J332BJD			YES		YES		YES
S6J332BJS			YES		YES		YES
S6J332BJU			YES		YES		YES
S6J332BKA			YES		YES		YES
S6J332BKB			YES		YES		YES
S6J332BKC			YES		YES		YES
S6J332BKD			YES		YES		YES
S6J332BKS			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J332BKU			YES		YES		YES
S6J332CHA			YES		YES		YES
S6J332CHB			YES		YES		YES
S6J332CHC			YES		YES		YES
S6J332CHD			YES		YES		YES
S6J332CHS			YES		YES		YES
S6J332CHU			YES		YES		YES
S6J332CJA			YES		YES		YES
S6J332CJB			YES		YES		YES
S6J332CJC			YES		YES		YES
S6J332CJD			YES		YES		YES
S6J332CJS			YES		YES		YES
S6J332CJU			YES		YES		YES
S6J332CKA			YES		YES		YES
S6J332CKB			YES		YES		YES
S6J332CKC			YES		YES		YES
S6J332CKD			YES		YES		YES
S6J332CKS			YES		YES		YES
S6J332CKU			YES		YES		YES
S6J332DHA			YES		YES		YES
S6J332DHB			YES		YES		YES
S6J332DHC			YES		YES		YES
S6J332DHD			YES		YES		YES
S6J332DHS			YES		YES		YES
S6J332DHU			YES		YES		YES
S6J332DJA			YES		YES		YES
S6J332DJB			YES		YES		YES
S6J332DJC			YES		YES		YES
S6J332DJD			YES		YES		YES
S6J332DJS			YES		YES		YES
S6J332DJU			YES		YES		YES
S6J332DKA			YES		YES		YES
S6J332DKB			YES		YES		YES
S6J332DKC			YES		YES		YES
S6J332DKD			YES		YES		YES
S6J332DKS			YES		YES		YES
S6J332DKU			YES		YES		YES
S6J332EHA			YES		YES		YES
S6J332EHB			YES		YES		YES
S6J332EHC			YES		YES		YES
S6J332EHD			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J332EHS			YES		YES		YES
S6J332EHU			YES		YES		YES
S6J332EJA			YES		YES		YES
S6J332EJB			YES		YES		YES
S6J332EJC			YES		YES		YES
S6J332EJD			YES		YES		YES
S6J332EJS			YES		YES		YES
S6J332EJU			YES		YES		YES
S6J332EKA			YES		YES		YES
S6J332EKB			YES		YES		YES
S6J332EKC			YES		YES		YES
S6J332EKD			YES		YES		YES
S6J332EKS			YES		YES		YES
S6J332EKU			YES		YES		YES
S6J333BHA			YES		YES		YES
S6J333BHB			YES		YES		YES
S6J333BHC			YES		YES		YES
S6J333BHD			YES		YES		YES
S6J333BHS			YES		YES		YES
S6J333BHU			YES		YES		YES
S6J333BJA			YES		YES		YES
S6J333BJB			YES		YES		YES
S6J333BJC			YES		YES		YES
S6J333BJD			YES		YES		YES
S6J333BJS			YES		YES		YES
S6J333BJU			YES		YES		YES
S6J333BKA			YES		YES		YES
S6J333BKB			YES		YES		YES
S6J333BKC			YES		YES		YES
S6J333BKD			YES		YES		YES
S6J333BKS			YES		YES		YES
S6J333BKU			YES		YES		YES
S6J333CHA			YES		YES		YES
S6J333CHB			YES		YES		YES
S6J333CHC			YES		YES		YES
S6J333CHD			YES		YES		YES
S6J333CHS			YES		YES		YES
S6J333CHU			YES		YES		YES
S6J333CJA			YES		YES		YES
S6J333CJB			YES		YES		YES
S6J333CJC			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J333CJD			YES		YES		YES
S6J333CJS			YES		YES		YES
S6J333CJU			YES		YES		YES
S6J333CKA			YES		YES		YES
S6J333CKB			YES		YES		YES
S6J333CKC			YES		YES		YES
S6J333CKD			YES		YES		YES
S6J333CKS			YES		YES		YES
S6J333CKU			YES		YES		YES
S6J333DHA			YES		YES		YES
S6J333DHB			YES		YES		YES
S6J333DHC			YES		YES		YES
S6J333DHD			YES		YES		YES
S6J333DHS			YES		YES		YES
S6J333DHU			YES		YES		YES
S6J333DJA			YES		YES		YES
S6J333DJB			YES		YES		YES
S6J333DJC			YES		YES		YES
S6J333DJD			YES		YES		YES
S6J333DJS			YES		YES		YES
S6J333DJU			YES		YES		YES
S6J333DKA			YES		YES		YES
S6J333DKB			YES		YES		YES
S6J333DKC			YES		YES		YES
S6J333DKD			YES		YES		YES
S6J333DKS			YES		YES		YES
S6J333DKU			YES		YES		YES
S6J333EHA			YES		YES		YES
S6J333EHB			YES		YES		YES
S6J333EHC			YES		YES		YES
S6J333EHD			YES		YES		YES
S6J333EHS			YES		YES		YES
S6J333EHU			YES		YES		YES
S6J333EJA			YES		YES		YES
S6J333EJB			YES		YES		YES
S6J333EJC			YES		YES		YES
S6J333EJD			YES		YES		YES
S6J333EJS			YES		YES		YES
S6J333EJU			YES		YES		YES
S6J333EKA			YES		YES		YES
S6J333EKB			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J333EKC			YES		YES		YES
S6J333EKD			YES		YES		YES
S6J333EKS			YES		YES		YES
S6J333EKU			YES		YES		YES
S6J334BHA			YES		YES		YES
S6J334BHB			YES		YES		YES
S6J334BHC			YES		YES		YES
S6J334BHD			YES		YES		YES
S6J334BHS			YES		YES		YES
S6J334BHU			YES		YES		YES
S6J334BJA			YES		YES		YES
S6J334BJB			YES		YES		YES
S6J334BJC			YES		YES		YES
S6J334BJD			YES		YES		YES
S6J334BJS			YES		YES		YES
S6J334BJU			YES		YES		YES
S6J334BKA			YES		YES		YES
S6J334BKB			YES		YES		YES
S6J334BKC			YES		YES		YES
S6J334BKD			YES		YES		YES
S6J334BKS			YES		YES		YES
S6J334BKU			YES		YES		YES
S6J334CHA			YES		YES		YES
S6J334CHB			YES		YES		YES
S6J334CHC			YES		YES		YES
S6J334CHD			YES		YES		YES
S6J334CHS			YES		YES		YES
S6J334CHU			YES		YES		YES
S6J334CJA			YES		YES		YES
S6J334CJB			YES		YES		YES
S6J334CJC			YES		YES		YES
S6J334CJD			YES		YES		YES
S6J334CJS			YES		YES		YES
S6J334CJU			YES		YES		YES
S6J334CKA			YES		YES		YES
S6J334CKB			YES		YES		YES
S6J334CKC			YES		YES		YES
S6J334CKD			YES		YES		YES
S6J334CKS			YES		YES		YES
S6J334CKU			YES		YES		YES
S6J334DHA			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J334DHB			YES		YES		YES
S6J334DHC			YES		YES		YES
S6J334DHD			YES		YES		YES
S6J334DHS			YES		YES		YES
S6J334DHU			YES		YES		YES
S6J334DJA			YES		YES		YES
S6J334DJB			YES		YES		YES
S6J334DJC			YES		YES		YES
S6J334DJD			YES		YES		YES
S6J334DJS			YES		YES		YES
S6J334DJU			YES		YES		YES
S6J334DKA			YES		YES		YES
S6J334DKB			YES		YES		YES
S6J334DKC			YES		YES		YES
S6J334DKD			YES		YES		YES
S6J334DKS			YES		YES		YES
S6J334DKU			YES		YES		YES
S6J334EHA			YES		YES		YES
S6J334EHB			YES		YES		YES
S6J334EHC			YES		YES		YES
S6J334EHD			YES		YES		YES
S6J334EHS			YES		YES		YES
S6J334EHU			YES		YES		YES
S6J334EJA			YES		YES		YES
S6J334EJB			YES		YES		YES
S6J334EJC			YES		YES		YES
S6J334EJD			YES		YES		YES
S6J334EJS			YES		YES		YES
S6J334EJU			YES		YES		YES
S6J334EKA			YES		YES		YES
S6J334EKB			YES		YES		YES
S6J334EKC			YES		YES		YES
S6J334EKD			YES		YES		YES
S6J334EKS			YES		YES		YES
S6J334EKU			YES		YES		YES
S6J335DHA			YES		YES		YES
S6J335DHB			YES		YES		YES
S6J335DHC			YES		YES		YES
S6J335DHD			YES		YES		YES
S6J335DHS			YES		YES		YES
S6J335DHU			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
S6J335DJA			YES		YES		YES
S6J335DJB			YES		YES		YES
S6J335DJC			YES		YES		YES
S6J335DJD			YES		YES		YES
S6J335DJS			YES		YES		YES
S6J335DJU			YES		YES		YES
S6J335DKA			YES		YES		YES
S6J335DKB			YES		YES		YES
S6J335DKC			YES		YES		YES
S6J335DKD			YES		YES		YES
S6J335DKS			YES		YES		YES
S6J335DKU			YES		YES		YES
S6J335EHA			YES		YES		YES
S6J335EHB			YES		YES		YES
S6J335EHC			YES		YES		YES
S6J335EHD			YES		YES		YES
S6J335EHS			YES		YES		YES
S6J335EHU			YES		YES		YES
S6J335EJA			YES		YES		YES
S6J335EJB			YES		YES		YES
S6J335EJC			YES		YES		YES
S6J335EJD			YES		YES		YES
S6J335EJS			YES		YES		YES
S6J335EJU			YES		YES		YES
S6J335EKA			YES		YES		YES
S6J335EKB			YES		YES		YES
S6J335EKC			YES		YES		YES
S6J335EKD			YES		YES		YES
S6J335EKS			YES		YES		YES
S6J335EKU			YES		YES		YES
SCORPION			YES		YES		YES
SPEAR1300			YES		YES		YES
SPEAR1310			YES		YES		YES
SPEAR1340			YES		YES		YES
STA1074			YES		YES		YES
STA1078			YES		YES		YES
STA1079			YES		YES		YES
STA1080			YES		YES		YES
STA1085			YES		YES		YES
STA1088			YES		YES		YES
STA1090			YES		YES		YES



CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
STA1095			YES		YES		YES
TCI6630K2L			YES		YES		YES
TCI6636K2H			YES		YES		YES
TCI6638K2K			YES		YES		YES
TDA2EX			YES		YES		
TDA2PX			YES		YES		
TDA2X			YES		YES		
TEGRAK1			YES				YES
TMS320C6A8143			YES				YES
TMS320C6A8147			YES				YES
TMS320C6A8148			YES				YES
TMS320C6A8167			YES				YES
TMS320C6A8168			YES				YES
TMS320DM3725			YES		YES		YES
TMS320DM3730			YES		YES		YES
TMS320DM8147			YES				YES
TMS320DM8148			YES				YES
TMS320DM8165			YES				YES
TMS320DM8166			YES				YES
TMS320DM8167			YES				YES
TMS320DM8168			YES				YES
TMS570LC4357			YES		YES		YES
TMS570LS0232			YES				YES
TMS570LS0332			YES				YES
TMS570LS0432			YES				YES
TMS570LS0714-PGE			YES				YES
TMS570LS0714-PZ			YES				YES
TMS570LS0914-PGE			YES				YES
TMS570LS0914-PZ			YES				YES
TMS570LS10106-PGE			YES				YES
TMS570LS10106-ZWT			YES		YES		YES
TMS570LS10116-PGE			YES				YES
TMS570LS10116-ZWT			YES		YES		YES
TMS570LS10206-PGE			YES				YES
TMS570LS10206-ZWT			YES		YES		YES
TMS570LS10216-PGE			YES				YES
TMS570LS10216-ZWT			YES		YES		YES
TMS570LS1113-PGE			YES				YES
TMS570LS1113-ZWT			YES				YES
TMS570LS1114-PGE			YES				YES
TMS570LS1114-ZWT			YES				YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
TMS570LS1115-PGE			YES				YES
TMS570LS1115-ZWT			YES				YES
TMS570LS1224-PGE			YES				YES
TMS570LS1224-ZWT			YES				YES
TMS570LS1225-PGE			YES				YES
TMS570LS1225-ZWT			YES				YES
TMS570LS1227-PGE			YES				YES
TMS570LS1227-ZWT			YES				YES
TMS570LS20206-PGE			YES				YES
TMS570LS20206-ZWT			YES		YES		YES
TMS570LS20216-PGE			YES				YES
TMS570LS20216-ZWT			YES		YES		YES
TMS570LS2124-PGE			YES				YES
TMS570LS2124-ZWT			YES		YES		YES
TMS570LS2125-PGE			YES				YES
TMS570LS2125-ZWT			YES		YES		YES
TMS570LS2134-PGE			YES				YES
TMS570LS2134-ZWT			YES		YES		YES
TMS570LS2135-PGE			YES				YES
TMS570LS2135-ZWT			YES		YES		YES
TMS570LS3134-PGE			YES				YES
TMS570LS3134-ZWT			YES		YES		YES
TMS570LS3135-PGE			YES				YES
TMS570LS3135-ZWT			YES		YES		YES
TMS570LS3137-EP			YES		YES		YES
TMS570LS3137-PGE			YES				YES
TMS570LS3137-ZWT			YES		YES		YES
VF11xR			YES		YES		YES
VF12xR			YES		YES		YES
VF31xR			YES		YES		YES
VF32xR			YES		YES		YES
VF3xx			YES		YES		YES
VF4xx			YES		YES		YES
VF51xR			YES		YES		YES
VF52xR			YES		YES		YES
VF5xx			YES		YES		YES
VF6xx			YES		YES		YES
VF7xx			YES		YES		YES
ZYNQ-7000			YES		YES		YES
ZYNQ-ULTRASCALE+			YES		YES		YES



Language	Compiler	Company	Option	Comment
ADA	GNAT PRO	AdaCore	ELF/DWARF	not all ADA constructs/DWARF
C	ARMCC	ARM Ltd.	AIF	
C	ARMCC	ARM Ltd.	ELF/DWARF	
C	REALVIEW-MDK	ARM Ltd.	ELF/DWARF	
C	GCCARM	Free Software Foundation, Inc.	COFF/STABS	
C	GCCARM	Free Software Foundation, Inc.	ELF/DWARF	
C	GREENHILLS-C	Greenhills Software Inc.	ELF/DWARF	
C	ICCARM	IAR Systems AB	ELF/DWARF	
C	ICCV7-ARM	Imagecraft Creations Inc.	ELF/DWARF	ARM7
C	CARM	ARM Germany GmbH	ELF/DWARF	
C	HIGH-C	Synopsys, Inc	ELF/DWARF	
C	TI-C	Texas Instruments	COFF	
C	GNU-C	Wind River Systems	COFF	
C	D-CC	Wind River Systems	ELF	
C++	ARM-SDT-2.50	ARM Ltd.	ELF/DWARF	
C++	REALVIEW-MDK	ARM Ltd.	ELF/DWARF	
C++	GCCARM	Free Software Foundation, Inc.	COFF/STABS	
C++	GNU	Free Software Foundation, Inc.	EXE/STABS	
C++	GCCARM	Free Software Foundation, Inc.	ELF/DWARF	
C++	GREENHILLS-C++	Greenhills Software Inc.	ELF/DWARF	
C++	MSVC	Microsoft Corporation	EXE/CV5	WindowsCE
C++	HIGH-C++	Synopsys, Inc	ELF/DWARF	
C/C++	GNAT PRO	AdaCore	ELF/DWARF	
C/C++	XCODE	Apple Inc.	Mach-O	
C/C++	GCC	HighTec EDV-Systeme GmbH	ELF/DWARF	
C/C++	VX-ARM	TASKING	ELF/DWARF	

Company	Product	Comment
KadakProducts Ltd.	AMX	
-	Android	Android based on Dalvik VM/Android RunTime
Oracle Corporation	ChorusOS	
CMX Systems Inc.	CMX-RTX	
Elektrobit Automotive GmbH	EB tresos AutoCore OS	via ORTI
Elektrobit Automotive GmbH	EB tresos Safety OS	via ORTI
eCosCentric Limited	ECOS	1.3, 2.0 and 3.0
Segger	embOS	4.24
Evidence	Erika	via ORTI
Cypress Semiconductor Corporation	FAMOS	
freeRTOS	FreeRTOS	up to v9
HIPPEROS S.A.	HIPPEROS	implemented by HIPPEROS
-	Linux	Kernel version 2.4, 2.6, 3.x, 4.x
MontaVista Software, LLC	Linux	3.0, 3.1, 4.0, 5.0
Timesys Corporation	Linux	
LiteOS Project	LiteOS	
ARM Ltd.	mbed OS	via RTX-ARM
NXP Semiconductors	MQX	3.x and 4.x
Synopsys, Inc	MQX	2.40 and 2.50
-	NetBSD	
Mentor Graphics Corporation	Nucleus	
NuttX Community	NuttX	
Radisys Inc.	OS-9	
ST Microelectronics N.V.	OS21	
Enea OSE Systems	OSE Basic	(OSARM)
Enea OSE Systems	OSE Delta	4.x and 5.x
Enea OSE Systems	OSE Epsilon	(OSARM), 3.x
-	OSEK	via ORTI
Sysgo AG	PikeOS	up to 4.2.1
eSOL Co., Ltd.	prKERNEL	
Elektrobit Automotive GmbH	ProOSEK	via ORTI
Wind River Systems	pSOS+	2.1 to 2.5, 3.0
QNX Software Systems	QNX	6.0 to 7.0
Quantum Leaps	QXK	
Hilscher GmbH	rcX	implemented by Hilscher
	RealTime Craft	(XECARM)
RTEMS	RTEMS	up to v5

Company	Product	Comment
ARM Germany GmbH	RTX-ARM	v3, v4 and v5
Quadros Systems Inc.	RTXC 3.2	
Quadros Systems Inc.	RTXC Quadros	
Sciopta	Sciopta	
Coressent Technology Inc.	SMX	3.4 to 4.3
Micro Digital Inc.	SMX	3.4 to 4.3
Symbian	Symbian OS	6.x, 7.0s, 8.0a 8.1a
Symbian	Symbian OS	8.0b, 8.1b, 9.x, S^3
Texas Instruments	SYS/BIOS	
eSOL Co., Ltd.	T-Kernel	
Express Logic Inc.	ThreadX	3.0, 4.0, 5.0
Micrium Inc.	uC/OS-II	2.0 to 2.92
Micrium Inc.	uC/OS-III	3.0
E-Force Corporation eForce Co., Ltd.	uC3/Compact	v2
E-Force Corporation eForce Co., Ltd.	uC3/Standard	
-	uClinux	Kernel Version 2.4, 2.6, 3.x, 4.x
-	ulTRON	HI7000, RX4000, NORTi, PrKernel
Wind River Systems	VxWorks	5.x to 7.x
Microsoft Corporation	Windows CE	4.0 to 6.0
Microsoft Corporation	Windows Embedded Compact 2013	
Microsoft Corporation	Windows Embedded Compact 7	
Microsoft Corporation	Windows Mobile	4.0 to 6.0
Microsoft Corporation	Windows Phone 7	
Microsoft Corporation	Windows Standard	
Zephyr Project	Zephyr	1.0 to 1.9

## Boot Loaders

Company	Product	Comment
American Megatrends Inc.	UEFI Aptio V	
Intel Corporation	UEFI TianoCore	

## 3rd-Party Tool Integrations

CPU	Tool	Company	Host
	WINDOWS CE PLATF. BUILDER	-	Windows
	CODE::BLOCKS	-	-
	C++TEST	-	Windows
	ADENEO	-	
	X-TOOLS / X32	blue river software GmbH	Windows
	CODEWRIGHT	Borland Software Corporation	Windows
	CODE CONFIDENCE TOOLS	Code Confidence Ltd	Windows
	CODE CONFIDENCE TOOLS	Code Confidence Ltd	Linux
	EASYCODE	EASYCODE GmbH	Windows
	ECLIPSE	Eclipse Foundation, Inc	Windows
	CHRONVIEW	Inchron GmbH	Windows
	LDRA TOOL SUITE	LDRA Technology, Inc.	Windows
	UML DEBUGGER	LieberLieber Software GmbH	Windows
	SIMULINK	The MathWorks Inc.	Windows
	ATTOL TOOLS	MicroMax Inc.	Windows
	VISUAL BASIC INTERFACE	Microsoft Corporation	Windows
	LABVIEW	NATIONAL INSTRUMENTS Corporation	Windows
	TPT	PikeTec GmbH	Windows
	CANTATA	QA Systems Ltd	Windows
	RAPITIME	Rapita Systems Ltd.	Windows
	RHAPSODY IN MICROC	IBM Corp.	Windows
	RHAPSODY IN C++	IBM Corp.	Windows
	DA-C	RistanCASE	Windows
	TRACEANALYZER	Symtavision GmbH	Windows
	ECU-TEST	TraceTronic GmbH	Windows
	UNDODB	Undo Software	Linux
	TA INSPECTOR	Vector	Windows
	VECTORCAST UNIT TESTING	Vector Software	Windows
	VECTORCAST CODE COVERAGE	Vector Software	Windows
ARM	GURUCE	-	Windows
ARM	VIVADO	XILINX	Windows

## Product Information

### ARM7

OrderNo Code	Text
<b>LA-7746</b> JTAG-ARM7-20	<b>JTAG Debugger for ARM7 20 Pin Connector (ICD)</b> supports ARM7 (0.4 V - 5 V) supports 5-pin standard JTAG, cJTAG and Serial Wire Debug Port includes software for Windows, Linux and MacOSX requires Power Debug Module cJTAG and SWD require Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace, Power Debug II or PowerDebug PRO
<b>LA-7746A</b> JTAG-ARM7-A	<b>JTAG Debugger License for ARM7 Add.</b> supports ARM7 please add the serial number of the base debug cable to your order
<b>LA-7746X</b> JTAG-ARM7-X	<b>JTAG Debugger Extension for ARM7</b> supports ARM7 requires a valid software guarantee or a valid software maintenance key please add the serial number of the base debug cable to your order
<b>LA-7748</b> JTAG-ARM-CON-20-TI14	<b>Converter ARM-20 to TI-14</b> Converter to connect a Debug Cable to a TI-14 connector which is used on many targets with processors from Texas Instruments
<b>LA-3780</b> JTAG-ARM-CON-20-TI20	<b>Converter ARM-20 to TI-14 or TI-20-Compact</b> Converter to connect a Debug Cable to a TI-14 or TI-20-Compact connector which is used on many targets with processors from Texas Instruments.



OrderNo Code	Text
<b>LA-3770</b> CONV-ARM20/MIPI34	<b>ARM Converter ARM-20 to MIPI-10/20/34</b> Converter to connect a Debug Cable to 10/20/34 pin connectors specified by MIPI. Converts to CombiProbe connector
<b>LA-7747</b> JTAG-ARM-CON-14-20	<b>ARM Converter ARM-20 to/from ARM-14</b> Converter to connect an ARM Debug Cable V1 (ARM-14) to ARM-20 or to connect a newer ARM Debug Cable (ARM-20) to ARM-14 target connector ARM-14 is an obsolete connector specification, do not use for new designs
<b>LA-3726</b> JTAG-ARM-CON-20-20	<b>ARM Converter 2x ARM-20 to ARM-20</b> Converter to connect two ARM Debug Cable to one connector on the target. Old method to handle multicore debugging by using two debugger hardware modules.
<b>LA-3717</b> MES-AD-JTAG20	<b>Measuring Adapter JTAG 20</b> Adapter to measure JTAG signals by a logic analyzer or to disconnect single JTAG lines from the target
<b>LA-3862</b> CON-ARM/MIPI34-MIC	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b> Converter to connect the ARM Debug Cable or the CombiProbe to a Mictor connector on the target. This is needed if you want to debug without a Preprocessor and if there is only a Mictor connector on the target. The trace signals of the CombiProbe are connected to the lowest four trace signals of the Mictor (ETMv3 pinout, continuous mode). But tracing is normally no use case due to the bandwidth limitations of the CombiProbe.

OrderNo Code	Text
<b>LA-7742</b> JTAG-ARM9	<b>JTAG Debugger for ARM9</b> supports ARM9 (0.4 V - 5 V) supports 5-pin standard JTAG, cJTAG and Serial Wire Debug Port includes software for Windows, Linux and MacOSX requires Power Debug Module cJTAG and SWD require Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace, Power Debug II or PowerDebug PRO
<b>LA-7742A</b> JTAG-ARM9-A	<b>JTAG Debugger License for ARM9 Add.</b> supports ARM9 please add the serial number of the base debug cable to your order
<b>LA-7742X</b> JTAG-ARM9-X	<b>JTAG Debugger Extension for ARM9</b> supports ARM9 requires a valid software guarantee or a valid software maintenance key please add the serial number of the base debug cable to your order
<b>LA-7970X</b> TRACE-LICENSE-ARM	<b>Trace License for ARM (Debug Cable)</b> Supports On-chip Trace for ARM Cores (ETB, ETF, ETR, TBR) please add the base serial number of your debug cable to your order
<b>LA-3722</b> CON-JTAG20-MICTOR	<b>ARM Converter ARM-20 to Mictor-38</b> Converter to connect the ARM Debug Cable to a Mictor connector on the target providing both debug and trace signals. This is needed if you want to connect the Debug Cable without a Preprocessor and if there is only a Mictor on the target. Suitable for MMDSP and ARC as well.
<b>LA-3717</b> MES-AD-JTAG20	<b>Measuring Adapter JTAG 20</b> Adapter to measure JTAG signals by a logic analyzer or to disconnect single JTAG lines from the target
<b>LA-3862</b> CON-ARM/MIPI34-MIC	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b> Converter to connect the ARM Debug Cable or the CombiProbe to a Mictor connector on the target. This is needed if you want to debug without a Preprocessor and if there is only a Mictor connector on the target. The trace signals of the CombiProbe are connected to the lowest four trace signals of the Mictor (ETMv3 pinout, continuous mode). But tracing is normally no use case due to the bandwidth limitations of the CombiProbe.

OrderNo Code	Text
<b>LA-7744</b> JTAG-ARM10	<b>JTAG Debugger for ARM10 (ICD)</b> supports ARM10 (0.4 V - 5 V) supports 5-pin standard JTAG, cJTAG and Serial Wire Debug Port includes software for Windows, Linux and MacOSX requires Power Debug Module cJTAG and SWD require Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace, Power Debug II or PowerDebug PRO
<b>LA-7744A</b> JTAG-ARM10-A	<b>JTAG Debugger License for ARM10 Add.</b> supports ARM10 please add the serial number of the base debug cable to your order
<b>LA-7744X</b> JTAG-ARM10-X	<b>JTAG Debugger Extension for ARM10</b> supports ARM10 requires a valid software guarantee or a valid software maintenance key please add the serial number of the base debug cable to your order
<b>LA-7970X</b> TRACE-LICENSE-ARM	<b>Trace License for ARM (Debug Cable)</b> Supports On-chip Trace for ARM Cores (ETB, ETF, ETR, TBR) please add the base serial number of your debug cable to your order
<b>LA-3717</b> MES-AD-JTAG20	<b>Measuring Adapter JTAG 20</b> Adapter to measure JTAG signals by a logic analyzer or to disconnect single JTAG lines from the target
<b>LA-3862</b> CON-ARM/MIPI34-MIC	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b> Converter to connect the ARM Debug Cable or the CombiProbe to a Mictor connector on the target. This is needed if you want to debug without a Preprocessor and if there is only a Mictor connector on the target. The trace signals of the CombiProbe are connected to the lowest four trace signals of the Mictor (ETMv3 pinout, continuous mode). But tracing is normally no use case due to the bandwidth limitations of the CombiProbe.

OrderNo Code	Text
<b>LA-7765</b> JTAG-ARM11	<b>JTAG Debugger for ARM11 (ICD)</b> supports ARM11 (0.4 V - 5 V) supports 5-pin standard JTAG, cJTAG and Serial Wire Debug Port includes software for Windows, Linux and MacOSX requires Power Debug Module cJTAG and SWD require Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace, Power Debug II or PowerDebug PRO
<b>LA-7765A</b> JTAG-ARM11-A	<b>JTAG Debugger License for ARM11 Add.</b> supports ARM11 please add the serial number of the base debug cable to your order
<b>LA-7765X</b> JTAG-ARM11-X	<b>JTAG Debugger Extension for ARM11</b> supports ARM11 requires a valid software guarantee or a valid software maintenance key please add the serial number of the base debug cable to your order
<b>LA-7970X</b> TRACE-LICENSE-ARM	<b>Trace License for ARM (Debug Cable)</b> Supports On-chip Trace for ARM Cores (ETB, ETF, ETR, TBR) please add the base serial number of your debug cable to your order
<b>LA-3717</b> MES-AD-JTAG20	<b>Measuring Adapter JTAG 20</b> Adapter to measure JTAG signals by a logic analyzer or to disconnect single JTAG lines from the target
<b>LA-3862</b> CON-ARM/MIPI34-MIC	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b> Converter to connect the ARM Debug Cable or the CombiProbe to a Mictor connector on the target. This is needed if you want to debug without a Preprocessor and if there is only a Mictor connector on the target. The trace signals of the CombiProbe are connected to the lowest four trace signals of the Mictor (ETMv3 pinout, continuous mode). But tracing is normally no use case due to the bandwidth limitations of the CombiProbe.

OrderNo Code	Text
<b>LA-7843</b> JTAG-ARMV7-A/R	<b>Debugger for Cortex-A/R (ARMv7 32-bit)</b> Supports ARMv7-A/R based Cortex-A and Cortex-R 32-bit cores supports 5-pin standard JTAG, cJTAG and Serial Wire Debug Port (0.4 V - 5 V) includes software for Windows, Linux and MacOSX requires Power Debug Module cJTAG and SWD require Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace, Power Debug II or PowerDebug PRO
<b>LA-7843A</b> JTAG-ARMV7-A/R-A	<b>Debug Cortex-A/R (ARMv7 32-bit) Add.</b> Supports ARMv7-A/R based Cortex-A and Cortex-R 32-bit cores please add the base serial number of your debug cable to your order
<b>LA-7843X</b> JTAG-ARMV7-A/R-X	<b>Debug Cortex-A/R (ARMv7 32-bit) Ext.</b> supports ARM Cortex-A and Cortex-R (ARMv7, 32-bit) requires a valid software guarantee or a valid software license key please add the base serial number of your debug cable to your order
<b>LA-7970X</b> TRACE-LICENSE-ARM	<b>Trace License for ARM (Debug Cable)</b> Supports On-chip Trace for ARM Cores (ETB, ETF, ETR, TBR) please add the base serial number of your debug cable to your order
<b>LA-3717</b> MES-AD-JTAG20	<b>Measuring Adapter JTAG 20</b> Adapter to measure JTAG signals by a logic analyzer or to disconnect single JTAG lines from the target
<b>LA-3881</b> CONV-ARM20/XILINX14	<b>ARM Converter ARM-20 to XILINX-14</b> Converter to connect an ARM Debug Cable to a 14-pin JTAG connector found on Xilinx target boards
<b>LA-3862</b> CON-ARM/MIPI34-MIC	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b> Converter to connect the ARM Debug Cable or the CombiProbe to a Mictor connector on the target. This is needed if you want to debug without a Preprocessor and if there is only a Mictor connector on the target. The trace signals of the CombiProbe are connected to the lowest four trace signals of the Mictor (ETMv3 pinout, continuous mode). But tracing is normally no use case due to the bandwidth limitations of the CombiProbe.
<b>LA-2746</b> CON-ARM20-ECU14	<b>Conv. ARM-20 MIPI-20 to 10-pin ECU14</b> ARM Converter ARM-20 or MIPI-20 to 10-pin ECU14

Order No.	Code	Text
LA-7746	JTAG-ARM7-20	JTAG Debugger for ARM7 20 Pin Connector (ICD)
LA-7746A	JTAG-ARM7-A	JTAG Debugger License for ARM7 Add.
LA-7746X	JTAG-ARM7-X	JTAG Debugger Extension for ARM7
LA-7748	JTAG-ARM-CON-20-TI14	Converter ARM-20 to TI-14
LA-3780	JTAG-ARM-CON-20-TI20	Converter ARM-20 to TI-14 or TI-20-Compact
LA-3770	CONV-ARM20/MIPI34	ARM Converter ARM-20 to MIPI-10/20/34
LA-7747	JTAG-ARM-CON-14-20	ARM Converter ARM-20 to/from ARM-14
LA-3726	JTAG-ARM-CON-20-20	ARM Converter 2x ARM-20 to ARM-20
LA-3717	MES-AD-JTAG20	Measuring Adapter JTAG 20
LA-3862	CON-ARM/MIPI34-MIC	ARM Conv. ARM-20, MIPI-34 to Mictor-38
<b>Additional Options</b>		
LA-2101	AD-HS-20	Adapter Half-Size 20 pin
LA-2720	CON-ARM20-MIPS14	Converter ARM-20 to MIPS-14
LA-3722	CON-JTAG20-MICTOR	ARM Converter ARM-20 to Mictor-38
LA-3788	DAISY-CHAINER-JTAG20	Daisy Chainer 4 JTAG 20
LA-7760A	EJTAG-MIPS32-A	Debugger License for MIPS32 Add.
LA-3501	GALVANIC-ISOLATION	Galvanic Isolation for Debug Cable
LA-3756A	JTAG-ANDES-A	JTAG Debugger License for AndeStar Add.
LA-3778A	JTAG-APS-A	JTAG Debugger License for APS Add.
LA-3750A	JTAG-ARC-A	JTAG Debugger License for ARC Add.
LA-7744X	JTAG-ARM10-X	JTAG Debugger Extension for ARM10
LA-7765X	JTAG-ARM11-X	JTAG Debugger Extension for ARM11
LA-7742X	JTAG-ARM9-X	JTAG Debugger Extension for ARM9
LA-7843X	JTAG-ARMV7-A/R-X	Debug Cortex-A/R (ARMv7 32-bit) Ext.
LA-3743X	JTAG-ARMV8-A/R-X	Debug Cortex-A/R (ARMv8 32/64-bit) Ext.
LA-7831A	JTAG-C54X-A	JTAG Debugger License for TMS320C54X Add.
LA-7830A	JTAG-C55X-A	JTAG Debugger License for TMS320C55x Add.
LA-7838A	JTAG-C6XXX-A	JTAG Debugger License for TMS320C6xxx Add.
LA-3711A	JTAG-CEVAX-A	JTAG Debugger License for CEVA-X Additional
LA-7844X	JTAG-CORTEX_M-X	Debug Cortex-M (ARMv6/7/8 32-bit) Ext.
LA-3741A	JTAG-HEXAGON-A	JTAG Debugger License for Hexagon Add.
LA-7836A	JTAG-MMDSP-A	JTAG Debugger License for MMDSP
LA-7789A	JTAG-OAK-SEIB-A	JTAG Debugger for TeakLite/OAK SEIB (ICD)
LA-7817A	JTAG-SH4-20-A	JTAG Debugger License for SH2/SH3/SH4 Add.
LA-7845A	JTAG-STARCORE-20-A	JTAG Debugger License for StarCore 20 Pin Add
LA-7774A	JTAG-TEAK-JAM-20-A	JTAG Debug. for Teak/TeakLite JAM 20 Add.

Order No.	Code	Text
LA-3844A	JTAG-TEAKLITE-4-A	JTAG Debugger for TeakLite-4 Add. (ICD)
LA-3774A	JTAG-TEAKLITE-III-A	JTAG Debugger for TeakLite III Add. (ICD)
LA-7847A	JTAG-TMS320C28X-A	JTAG Debugger License for TMS320C28X Add.
LA-3747A	JTAG-UBI32-A	JTAG/SPI Debugger License for UBI32 Add.
LA-7762X	JTAG-XSCALE-X	JTAG Debugger Extension for XSCALE (ICD)
LA-3760A	JTAG-XTENSA-A	JTAG Debugger License for Xtensa Add.
LA-7832A	JTAG-ZSP400-A	JTAG Debugger for ZSP400 DSP Core Additional
LA-3712A	JTAG-ZSP500-A	JTAG Debugger for ZSP500 DSP Core Additional
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging
LA-7970X	TRACE-LICENSE-ARM	Trace License for ARM (Debug Cable)

## ARM9

Order No.	Code	Text
<b>LA-7742</b>	<b>JTAG-ARM9</b>	<b>JTAG Debugger for ARM9</b>
<b>LA-7742A</b>	<b>JTAG-ARM9-A</b>	<b>JTAG Debugger License for ARM9 Add.</b>
<b>LA-7742X</b>	<b>JTAG-ARM9-X</b>	<b>JTAG Debugger Extension for ARM9</b>
<b>LA-7970X</b>	<b>TRACE-LICENSE-ARM</b>	<b>Trace License for ARM (Debug Cable)</b>
<b>LA-3722</b>	<b>CON-JTAG20-MICTOR</b>	<b>ARM Converter ARM-20 to Mictor-38</b>
<b>LA-3717</b>	<b>MES-AD-JTAG20</b>	<b>Measuring Adapter JTAG 20</b>
<b>LA-3862</b>	<b>CON-ARM/MIPI34-MIC</b>	<b>ARM Conv. ARM-20, MIPI-34 to Mictor-38</b>
<b>Additional Options</b>		
LA-2101	AD-HS-20	Adapter Half-Size 20 pin
LA-2720	CON-ARM20-MIPS14	Converter ARM-20 to MIPS-14
LA-3770	CONV-ARM20/MIPI34	ARM Converter ARM-20 to MIPI-10/20/34
LA-3788	DAISY-CHAINER-JTAG20	Daisy Chainer 4 JTAG 20
LA-7760A	EJTAG-MIPS32-A	Debugger License for MIPS32 Add.
LA-3501	GALVANIC-ISOLATION	Galvanic Isolation for Debug Cable
LA-3756A	JTAG-ANDES-A	JTAG Debugger License for AndeStar Add.
LA-3778A	JTAG-APS-A	JTAG Debugger License for APS Add.
LA-3750A	JTAG-ARC-A	JTAG Debugger License for ARC Add.
LA-7747	JTAG-ARM-CON-14-20	ARM Converter ARM-20 to/from ARM-14
LA-3726	JTAG-ARM-CON-20-20	ARM Converter 2x ARM-20 to ARM-20
LA-7748	JTAG-ARM-CON-20-TI14	Converter ARM-20 to TI-14
LA-3780	JTAG-ARM-CON-20-TI20	Converter ARM-20 to TI-14 or TI-20-Compact
LA-7744X	JTAG-ARM10-X	JTAG Debugger Extension for ARM10
LA-7765X	JTAG-ARM11-X	JTAG Debugger Extension for ARM11
LA-7746X	JTAG-ARM7-X	JTAG Debugger Extension for ARM7
LA-7843X	JTAG-ARMV7-A/R-X	Debug Cortex-A/-R (ARMv7 32-bit) Ext.

Order No.	Code	Text
LA-3743X	JTAG-ARMV8-A/R-X	Debug Cortex-A/R (ARMv8 32/64-bit) Ext.
LA-7831A	JTAG-C54X-A	JTAG Debugger License for TMS320C54X Add.
LA-7830A	JTAG-C55X-A	JTAG Debugger License for TMS320C55x Add.
LA-7838A	JTAG-C6XXX-A	JTAG Debugger License for TMS320C6xxx Add.
LA-3711A	JTAG-CEVAX-A	JTAG Debugger License for CEVA-X Additional
LA-7844X	JTAG-CORTEX_M-X	Debug Cortex-M (ARMv6/7/8 32-bit) Ext.
LA-3741A	JTAG-HEXAGON-A	JTAG Debugger License for Hexagon Add.
LA-7836A	JTAG-MMDSP-A	JTAG Debugger License for MMDSP
LA-7789A	JTAG-OAK-SEIB-A	JTAG Debugger for TeakLite/OAK SEIB (ICD)
LA-7850A	JTAG-R8051XC-A	JTAG Debugger for R8051XC Add.
LA-7817A	JTAG-SH4-20-A	JTAG Debugger License for SH2/SH3/SH4 Add.
LA-7845A	JTAG-STARCORE-20-A	JTAG Debugger License for StarCore 20 Pin Add
LA-7774A	JTAG-TEAK-JAM-20-A	JTAG Debug. for Teak/TeakLite JAM 20 Add.
LA-3844A	JTAG-TEAKLITE-4-A	JTAG Debugger for TeakLite-4 Add. (ICD)
LA-3774A	JTAG-TEAKLITE-III-A	JTAG Debugger for TeakLite III Add. (ICD)
LA-7847A	JTAG-TMS320C28X-A	JTAG Debugger License for TMS320C28X Add.
LA-3747A	JTAG-UBI32-A	JTAG/SPI Debugger License for UBI32 Add.
LA-7762X	JTAG-XSCALE-X	JTAG Debugger Extension for XSCALE (ICD)
LA-3760A	JTAG-XTENSA-A	JTAG Debugger License for Xtensa Add.
LA-7832A	JTAG-ZSP400-A	JTAG Debugger for ZSP400 DSP Core Additional
LA-3712A	JTAG-ZSP500-A	JTAG Debugger for ZSP500 DSP Core Additional
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging
LA-7759A	OCDS-C166S-V2-A	OCDS Debugger for XC2000/C166S V2 Additional



Order No.	Code	Text
LA-7744	JTAG-ARM10	JTAG Debugger for ARM10 (ICD)
LA-7744A	JTAG-ARM10-A	JTAG Debugger License for ARM10 Add.
LA-7744X	JTAG-ARM10-X	JTAG Debugger Extension for ARM10
LA-7970X	TRACE-LICENSE-ARM	Trace License for ARM (Debug Cable)
LA-3717	MES-AD-JTAG20	Measuring Adapter JTAG 20
LA-3862	CON-ARM/MIPI34-MIC	ARM Conv. ARM-20, MIPI-34 to Mictor-38
<b>Additional Options</b>		
LA-2101	AD-HS-20	Adapter Half-Size 20 pin
LA-3722	CON-JTAG20-MICTOR	ARM Converter ARM-20 to Mictor-38
LA-3770	CONV-ARM20/MIPI34	ARM Converter ARM-20 to MIPI-10/20/34
LA-3501	GALVANIC-ISOLATION	Galvanic Isolation for Debug Cable
LA-3756A	JTAG-ANDES-A	JTAG Debugger License for AndeStar Add.
LA-3778A	JTAG-APS-A	JTAG Debugger License for APS Add.
LA-3750A	JTAG-ARC-A	JTAG Debugger License for ARC Add.
LA-7747	JTAG-ARM-CON-14-20	ARM Converter ARM-20 to/from ARM-14
LA-3726	JTAG-ARM-CON-20-20	ARM Converter 2x ARM-20 to ARM-20
LA-7748	JTAG-ARM-CON-20-TI14	Converter ARM-20 to TI-14
LA-3780	JTAG-ARM-CON-20-TI20	Converter ARM-20 to TI-14 or TI-20-Compact
LA-7765X	JTAG-ARM11-X	JTAG Debugger Extension for ARM11
LA-7746X	JTAG-ARM7-X	JTAG Debugger Extension for ARM7
LA-7742X	JTAG-ARM9-X	JTAG Debugger Extension for ARM9
LA-7843X	JTAG-ARMV7-A/R-X	Debug Cortex-A/-R (ARMv7 32-bit) Ext.
LA-3743X	JTAG-ARMV8-A/R-X	Debug Cortex-A/R (ARMv8 32/64-bit) Ext.
LA-7831A	JTAG-C54X-A	JTAG Debugger License for TMS320C54X Add.
LA-7830A	JTAG-C55X-A	JTAG Debugger License for TMS320C55x Add.
LA-7838A	JTAG-C6XXX-A	JTAG Debugger License for TMS320C6xxx Add.
LA-3711A	JTAG-CEVAX-A	JTAG Debugger License for CEVA-X Additional
LA-7844X	JTAG-CORTEX_M-X	Debug Cortex-M (ARMv6/7/8 32-bit) Ext.
LA-3741A	JTAG-HEXAGON-A	JTAG Debugger License for Hexagon Add.
LA-7836A	JTAG-MMDSP-A	JTAG Debugger License for MMDSP
LA-7817A	JTAG-SH4-20-A	JTAG Debugger License for SH2/SH3/SH4 Add.
LA-7845A	JTAG-STARCORE-20-A	JTAG Debugger License for StarCore 20 Pin Add
LA-7774A	JTAG-TEAK-JAM-20-A	JTAG Debug. for Teak/TeakLite JAM 20 Add.
LA-3844A	JTAG-TEAKLITE-4-A	JTAG Debugger for TeakLite-4 Add. (ICD)
LA-3774A	JTAG-TEAKLITE-III-A	JTAG Debugger for TeakLite III Add. (ICD)
LA-7847A	JTAG-TMS320C28X-A	JTAG Debugger License for TMS320C28X Add.
LA-3747A	JTAG-UBI32-A	JTAG/SPI Debugger License for UBI32 Add.
LA-7762X	JTAG-XSCALE-X	JTAG Debugger Extension for XSCALE (ICD)
LA-3760A	JTAG-XTENSA-A	JTAG Debugger License for Xtensa Add.

Order No.	Code	Text
LA-7832A	JTAG-ZSP400-A	JTAG Debugger for ZSP400 DSP Core Additional
LA-3712A	JTAG-ZSP500-A	JTAG Debugger for ZSP500 DSP Core Additional
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging

Order No.	Code	Text
LA-7765	JTAG-ARM11	JTAG Debugger for ARM11 (ICD)
LA-7765A	JTAG-ARM11-A	JTAG Debugger License for ARM11 Add.
LA-7765X	JTAG-ARM11-X	JTAG Debugger Extension for ARM11
LA-7970X	TRACE-LICENSE-ARM	Trace License for ARM (Debug Cable)
LA-3717	MES-AD-JTAG20	Measuring Adapter JTAG 20
LA-3862	CON-ARM/MIPI34-MIC	ARM Conv. ARM-20, MIPI-34 to Mictor-38
<b>Additional Options</b>		
LA-2101	AD-HS-20	Adapter Half-Size 20 pin
LA-2720	CON-ARM20-MIPS14	Converter ARM-20 to MIPS-14
LA-3722	CON-JTAG20-MICTOR	ARM Converter ARM-20 to Mictor-38
LA-3770	CONV-ARM20/MIPI34	ARM Converter ARM-20 to MIPI-10/20/34
LA-3788	DAISY-CHAINER-JTAG20	Daisy Chainer 4 JTAG 20
LA-7760A	EJTAG-MIPS32-A	Debugger License for MIPS32 Add.
LA-3501	GALVANIC-ISOLATION	Galvanic Isolation for Debug Cable
LA-3756A	JTAG-ANDES-A	JTAG Debugger License for AndeStar Add.
LA-3778A	JTAG-APS-A	JTAG Debugger License for APS Add.
LA-3750A	JTAG-ARC-A	JTAG Debugger License for ARC Add.
LA-7747	JTAG-ARM-CON-14-20	ARM Converter ARM-20 to/from ARM-14
LA-3726	JTAG-ARM-CON-20-20	ARM Converter 2x ARM-20 to ARM-20
LA-7748	JTAG-ARM-CON-20-TI14	Converter ARM-20 to TI-14
LA-3780	JTAG-ARM-CON-20-TI20	Converter ARM-20 to TI-14 or TI-20-Compact
LA-7744X	JTAG-ARM10-X	JTAG Debugger Extension for ARM10
LA-7746X	JTAG-ARM7-X	JTAG Debugger Extension for ARM7
LA-7742X	JTAG-ARM9-X	JTAG Debugger Extension for ARM9
LA-7843X	JTAG-ARMV7-A/R-X	Debug Cortex-A/R (ARMv7 32-bit) Ext.
LA-3743X	JTAG-ARMV8-A/R-X	Debug Cortex-A/R (ARMv8 32/64-bit) Ext.
LA-7831A	JTAG-C54X-A	JTAG Debugger License for TMS320C54X Add.
LA-7830A	JTAG-C55X-A	JTAG Debugger License for TMS320C55x Add.
LA-7838A	JTAG-C6XXX-A	JTAG Debugger License for TMS320C6xxx Add.
LA-3711A	JTAG-CEVAX-A	JTAG Debugger License for CEVA-X Additional
LA-7844X	JTAG-CORTEX_M-X	Debug Cortex-M (ARMv6/7/8 32-bit) Ext.
LA-3741A	JTAG-HEXAGON-A	JTAG Debugger License for Hexagon Add.
LA-7836A	JTAG-MMDSP-A	JTAG Debugger License for MMDSP
LA-7789A	JTAG-OAK-SEIB-A	JTAG Debugger for TeakLite/OAK SEIB (ICD)
LA-7817A	JTAG-SH4-20-A	JTAG Debugger License for SH2/SH3/SH4 Add.
LA-7845A	JTAG-STARCORE-20-A	JTAG Debugger License for StarCore 20 Pin Add
LA-7774A	JTAG-TEAK-JAM-20-A	JTAG Debug. for Teak/TeakLite JAM 20 Add.
LA-3844A	JTAG-TEAKLITE-4-A	JTAG Debugger for TeakLite-4 Add. (ICD)
LA-3774A	JTAG-TEAKLITE-III-A	JTAG Debugger for TeakLite III Add. (ICD)
LA-7847A	JTAG-TMS320C28X-A	JTAG Debugger License for TMS320C28X Add.

Order No.	Code	Text
LA-3747A	JTAG-UBI32-A	JTAG/SPI Debugger License for UBI32 Add.
LA-7762X	JTAG-XSCALE-X	JTAG Debugger Extension for XSCALE (ICD)
LA-3760A	JTAG-XTENSA-A	JTAG Debugger License for Xtensa Add.
LA-7832A	JTAG-ZSP400-A	JTAG Debugger for ZSP400 DSP Core Additional
LA-3712A	JTAG-ZSP500-A	JTAG Debugger for ZSP500 DSP Core Additional
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging

Order No.	Code	Text
LA-7843	JTAG-ARMV7-A/R	Debugger for Cortex-A/R (ARMv7 32-bit)
LA-7843A	JTAG-ARMV7-A/R-A	Debug Cortex-A/R (ARMv7 32-bit) Add.
LA-7843X	JTAG-ARMV7-A/R-X	Debug Cortex-A/R (ARMv7 32-bit) Ext.
LA-7970X	TRACE-LICENSE-ARM	Trace License for ARM (Debug Cable)
LA-3717	MES-AD-JTAG20	Measuring Adapter JTAG 20
LA-3881	CONV-ARM20/XILINX14	ARM Converter ARM-20 to XILINX-14
LA-3862	CON-ARM/MIPI34-MIC	ARM Conv. ARM-20, MIPI-34 to Mictor-38
LA-2746	CON-ARM20-ECU14	Conv. ARM-20 MIPI-20 to 10-pin ECU14
<b>Additional Options</b>		
LA-2101	AD-HS-20	Adapter Half-Size 20 pin
LA-2720	CON-ARM20-MIPS14	Converter ARM-20 to MIPS-14
LA-3722	CON-JTAG20-MICTOR	ARM Converter ARM-20 to Mictor-38
LA-3770	CONV-ARM20/MIPI34	ARM Converter ARM-20 to MIPI-10/20/34
LA-3788	DAISY-CHAINER-JTAG20	Daisy Chainer 4 JTAG 20
LA-7760A	EJTAG-MIPS32-A	Debugger License for MIPS32 Add.
LA-3501	GALVANIC-ISOLATION	Galvanic Isolation for Debug Cable
LA-3756A	JTAG-ANDES-A	JTAG Debugger License for AndeStar Add.
LA-3744A	JTAG-APEX-A	JTAG Debugger Lic. for APEX ADD.
LA-3778A	JTAG-APS-A	JTAG Debugger License for APS Add.
LA-3750A	JTAG-ARC-A	JTAG Debugger License for ARC Add.
LA-7747	JTAG-ARM-CON-14-20	ARM Converter ARM-20 to/from ARM-14
LA-3726	JTAG-ARM-CON-20-20	ARM Converter 2x ARM-20 to ARM-20
LA-7748	JTAG-ARM-CON-20-TI14	Converter ARM-20 to TI-14
LA-3780	JTAG-ARM-CON-20-TI20	Converter ARM-20 to TI-14 or TI-20-Compact
LA-7744X	JTAG-ARM10-X	JTAG Debugger Extension for ARM10
LA-7765X	JTAG-ARM11-X	JTAG Debugger Extension for ARM11
LA-7746X	JTAG-ARM7-X	JTAG Debugger Extension for ARM7
LA-7742X	JTAG-ARM9-X	JTAG Debugger Extension for ARM9
LA-3743X	JTAG-ARMV8-A/R-X	Debug Cortex-A/R (ARMv8 32/64-bit) Ext.
LA-2705A	JTAG-ARP32-A	JTAG Debugger for ARP32 Add.
LA-7831A	JTAG-C54X-A	JTAG Debugger License for TMS320C54X Add.
LA-7830A	JTAG-C55X-A	JTAG Debugger License for TMS320C55x Add.
LA-7838A	JTAG-C6XXX-A	JTAG Debugger License for TMS320C6xxx Add.
LA-3711A	JTAG-CEVAX-A	JTAG Debugger License for CEVA-X Additional
LA-7844X	JTAG-CORTEX_M-X	Debug Cortex-M (ARMv6/7/8 32-bit) Ext.
LA-3741A	JTAG-HEXAGON-A	JTAG Debugger License for Hexagon Add.
LA-3730A	JTAG-MICROBLAZE-A	JTAG Debug. License for MicroBlaze Additional
LA-7836A	JTAG-MMDSP-A	JTAG Debugger License for MMDSP
LA-7837A	JTAG-NIOS-II-A	JTAG Debugger License for NIOS-II Add.
LA-2756A	JTAG-PRU-A	JTAG Debugger for PRU Add.

Order No.	Code	Text
LA-2717A	JTAG-RISC-V-A	JTAG Debugger for RISC-V Add.
LA-7817A	JTAG-SH4-20-A	JTAG Debugger License for SH2/SH3/SH4 Add.
LA-7845A	JTAG-STARCORE-20-A	JTAG Debugger License for StarCore 20 Pin Add
LA-7774A	JTAG-TEAK-JAM-20-A	JTAG Debug. for Teak/TeakLite JAM 20 Add.
LA-3844A	JTAG-TEAKLITE-4-A	JTAG Debugger for TeakLite-4 Add. (ICD)
LA-3774A	JTAG-TEAKLITE-III-A	JTAG Debugger for TeakLite III Add. (ICD)
LA-7847A	JTAG-TMS320C28X-A	JTAG Debugger License for TMS320C28X Add.
LA-3747A	JTAG-UBI32-A	JTAG/SPI Debugger License for UBI32 Add.
LA-7762X	JTAG-XSCALE-X	JTAG Debugger Extension for XSCALE (ICD)
LA-3760A	JTAG-XTENSA-A	JTAG Debugger License for Xtensa Add.
LA-7832A	JTAG-ZSP400-A	JTAG Debugger for ZSP400 DSP Core Additional
LA-3712A	JTAG-ZSP500-A	JTAG Debugger for ZSP500 DSP Core Additional
LA-7960X	MULTICORE-LICENSE	License for Multicore Debugging
LA-7756A	OCDS-TRICORE-A	Debugger for TriCore Standard Additional