

MIPS Debugger and Trace

Release 09.2023

MANUAL

MIPS Debugger and Trace

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

[TRACE32 Documents](#)



[ICD In-Circuit Debugger](#)



[Processor Architecture Manuals](#)



[MIPS](#)



MIPS Debugger and Trace

1

Introduction

5

Brief Overview of Documents for New Users

5

Demo and Start-up Scripts

5

WARNING

6

Quick Start of the EJTAG Debugger

7

Troubleshooting

8

SYStem.Up Errors

8

FAQ

9

CPU specific Implementations

10

Breakpoints

10

Instruction Breakpoints (Software Breakpoints)

10

Instruction Breakpoints in ROM (On-chip Breakpoints)

10

Breakpoints on Read/Write Access to Data(On-chip Breakpoints)

10

Example for Standard Breakpoints

11

Trigger

12

Runtime Measurement

12

Register

12

Memory Classes

13

SPR Memory Overlay

14

MIPS specific SYStem Commands

16

SYStem.CONFIG Configure debugger according to target topology

16

SYStem.CPU Select the used CPU

28

SYStem.DETECT.CORENUMBER Detect core number

29

SYStem.JtagClock Define JTAG clock

30

SYStem.LOCK Tristate the JTAG port

31

SYStem.MemAccess Run-time memory access

31

SYStem.Mode Establish the communication with the target

33

SYStem.Option.Address32	Define address format display	34
SYStem.Option.DCFREEZE	Freeze data cache	34
SYStem.Option.DCREAD	Use DCACHE for data read	35
SYStem.Option.DisMode	Define disassembler mode	35
SYStem.Option.Endianness	Define endianness of target memory	37
SYStem.Option.EnReset	Control target system reset	37
SYStem.Option.EnTRST	Control TAP reset	37
SYStem.Option.HoldReset	Set system reset hold time	38
SYStem.Option.FlowTrace	Define operating mode of RISC TRACE	38
SYStem.Option.FREEZE	Freeze system timer in stop mode	38
SYStem.Option.ICFLUSH	Flush of instruction cache during step and go	39
SYStem.Option.ICREAD	Use ICACHE for program read	39
SYStem.Option.IMASKASM	Disable interrupts while ASM single stepping	39
SYStem.Option.IMASKHLL	Disable interrupts while HLL single stepping	40
SYStem.Option.KEYCODE	Define key code to unsecure processor	40
SYStem.Option.MCBreaksynch	Select break synchronization method	40
SYStem.Option.MMUPhysLogMemaccess	Memory access preferences	41
SYStem.Option.MMUSPACES	Separate address spaces by space IDs	41
SYStem.Option.MonBase	Base address for monitor download routine	43
SYStem.Option.OVERLAY	Enable overlay support	43
SYStem.Option.PROTECTION	Sends an unsecure sequence to the core	44
SYStem.Option.ResBreak	Halt the core after reset	44
SYStem.Option.STEPONCHIP	Use onchip breakpoints for ASM stepping	44
SYStem.Option.STEPSOFT	Use software breakpoints for ASM stepping	45
SYStem.Option.TURBO	Enable fast download	45
SYStem.Option.UnProtect	Unprotect memory addresses	45
SYStem.Option.WaitReset	Set system reset wait time	46
SYStem.Option.WATCHDOG	Disable hardware watchdogs	46
SYStem.RESetOut	Assert nRESET/nSRST on JTAG connector	47
On-chip Breakpoints		48
TrOnchip.AddressMask	Define an address mask	48
TrOnchip.CORERESET	Halt at reset vector after core reset	48
TrOnchip.RESet	Set on-chip trigger to default state	48
TrOnchip.StepVector	Halt on exception vector during step	48
TrOnchip.UseWatch	Use watchpoints	49
TrOnchip.state	Display on-chip trigger window	49
CPU specific MMU Commands		50
MMU.DUMP	Page wise display of MMU translation table	50
MMU.FORMAT	Define MMU table structure	51
MMU.List	Compact display of MMU translation table	55
MMU.SCAN	Load MMU table from CPU	56
MMU.Set	Set MMU registers	57
MMU.TLB.Set	Set MMU registers	58

MMU.TLBSET	Set MMU registers	58
TCB		59
TCB Control		59
Configuring your FPGA		61
Using JTAG for FPGA configuration		61
EJTAG Connector		62
Mechanical Description of the 14-pin EJTAG Connector		62
Electrical Description of the 14-pin EJTAG Connector		63
Mechanical Description of the 24-pin EJTAG Connector		64
Electrical Description of the 24-pin EJTAG Connector		65
Recommended JTAG Circuit on Target		66
Technical Data Debugger		67
Operation Voltage		67
Mechanical Dimensions		67
Technical Data Trace		69
Operation Voltage		69
Mechanical Dimensions		70

Introduction

Please keep in mind that only the **Processor Architecture Manual** (the document you are reading at the moment) is CPU specific, while all other parts of the online help are generic for all CPUs supported by Lauterbach. So if there are questions related to the CPU, the Processor Architecture Manual should be your first choice.

Brief Overview of Documents for New Users

Architecture-independent information:

- “**Training Basic Debugging**” (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- “**T32Start**” (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- “**General Commands**” (general_ref_<x>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- “**Processor Architecture Manuals**”: These manuals describe commands that are specific for the processor architecture supported by your Debug Cable. To access the manual for your processor architecture, proceed as follows:
 - Choose **Help** menu > **Processor Architecture Manual**.
- “**OS Awareness Manuals**” (rtos_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Demo and Start-up Scripts

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

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

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

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

You can also manually navigate in the `~/demo/mips/` and `~/demo/mips64/` subfolders of the system directory of TRACE32.

WARNING

WARNING:	<p>To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF.</p> <p>Recommendation for the software start:</p> <ol style="list-style-type: none">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. <p>Power down:</p> <ol style="list-style-type: none">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.
-----------------	--

Quick Start of the EJTAG Debugger

All default settings should be fine. Therefore the only required command should be **SYStem.Up**. This command resets the processor, establish connection via EJTAG, and requests the processor to enter debug mode. After this command is executed, it is possible to access memory and registers.

A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO <file>**.

```
Reset ; Only required if you do not start
       ; immediately after booting

WinCLEAR ; Clear all windows

MAP.BOnchip 0x100000++0xffff ; Specify where ROM/Flash is, on-chip
                             ; breakpoints will be automatically
                             ; used there

SYSTEM.Up ; Reset the target and enter debug mode

Data.LOAD.Ieee demo.abs ; Load the application program

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

List.Mix ; Open source code window *) */

Register.view /SpotLight ; Open register window *) */

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

Var.Watch flags ast ; 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)
```

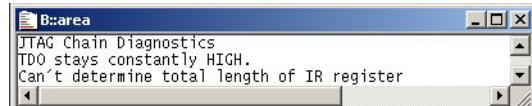
*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

SYStem.Up Errors

The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this can have many reasons.

A first test, the JTAG Chain Diagnostics, determines if there is a basic electrical problem with the JTAG interface. For this test, a area window has to be opened and the **SYStem.Mode** must be down. The following command sequence starts the diagnostics:

```
diag 10000 1  
diag 16001
```



In this example, no reasonable values for JTAG chain properties could be detected. There seems to be a **general electrical problem** with the JTAG port.



The diagnostics has detected some reasonable values for the JTAG chain. There seems to be a more **advanced problem**.

General electrical problems with the JTAG interface:

- The target has no power.
- The target is in reset.
- The processor has no clock.
- The EJTAG connection is not done properly (see [EJTAG connector](#)).
- On the board can be switched between JTAG and EJTAG and JTAG is active. E.g. a jumper is wrongly set or a resistor must be removed.
- Selected JTAG frequency is too high.
- The target's JTAG circuit is incompatible with LAUTERBACH JTAG adapter. See [recommended JTAG schematics](#) for more information.

Advanced problems:

- The wrong processor type is selected in the **SYStem.CPU** list.
- The target is a multicore device. See **SYStem.CONFIG** for more information.
- The JTAG frequency is too high or no RTCK is available.
- The target is in an unrecoverable state. Re-power the target and try again.

FAQ

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

Breakpoints

Onchip instruction and data breakpoints and software breakpoints are supported.

NOTE: For all MIPS cores with VPEs it is only possible to set onchip breakpoints on active VPEs. Setting an onchip breakpoint during SMP debugging with inactive VPEs a warning will be displayed. To guarantee that all TCs on all VPEs will halt at the onchip breakpoint the user should set an instruction breakpoint after creation of all TCs on all VPEs.

Instruction Breakpoints (Software Breakpoints)

The program code will be patched to force the processor entering debug mode when reaching this instruction. Therefore unlimited number of software breakpoints are available. But there is the need to modify the program memory (RAM).

It is not allowed to place a software breakpoint on an instruction in a delay slot of a branch or jump instruction.

Instruction Breakpoints in ROM (On-chip Breakpoints)

With the command **MAP.BOnchip <range>** it is possible to tell the debugger where you have ROM (FLASH, EPROM) on the target. If a breakpoint is set into a location mapped as BOnchip on-chip breakpoints will be used. Depending on the used processor type 0 to 15 on-chip breakpoints are available.

Breakpoints on Read/Write Access to Data(On-chip Breakpoints)

Breakpoints on data can be set with the options /Write or /Read of the Break.Set command. Depending on the used processor type 0 to 15 data breakpoints are available.

Example for Standard Breakpoints

Assume you have a target with FLASH from 0 to fffff and RAM from 100000 to 11ffff. The command to configure TRACE32 correctly for this configuration is:

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

The following standard breakpoint combinations are possible.

1. Instruction breakpoints in RAM and one breakpoint in ROM

```
Break.Set 0x100000 /Program ; software breakpoint 1  
Break.Set 0x101000 /Program ; software breakpoint 2  
Break.Set addr /Program ; software breakpoint 3 to x  
Break.Set 0x100 /Program ; on-chip breakpoint
```

2. Instruction breakpoints in RAM and one data breakpoint

```
Break.Set 0x100000 /Program ; software breakpoint 1  
Break.Set 0x101000 /Program ; software breakpoint 2  
Break.Set addr /Program ; software breakpoint 3 to x  
Break.Set 0x108000 /Write ; write data breakpoint
```

3. Two instruction breakpoints in ROM

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

4. Two data breakpoints

```
Break.Set 0x108000 /Write ; write data breakpoint  
Break.Set 0x108010 /Read ; read data breakpoint
```

5. One breakpoint in ROM and one data breakpoint

```
Break.Set 0x100 /Program ; Hardware Breakpoint  
Break.Set 0x108010 /Read ; Read Watchpoint
```

Trigger

A bidirectional trigger system allows the following two events:

- trigger an external system (e.g. logic analyzer) if the MIPS breaks (**TrBus.Out**)
- break emulation if an external trigger is asserted (**TrBus.Set**)

The location of the bidirectional trigger connector which is on the host interface (PODPC, PODPAR, PODETH) is shown in the ICD Debugger Users Guide.

The trigger system has the following specific restriction:

- If a terminal window is open the response time of the trigger system is undefined. It is recommended not to use the trigger system and terminal window at the same time.

Runtime Measurement

The function RunTime allows run time measurement. The measurement is done by software control. Therefore the result is not an exact value.

Register

In the register window the 32 general-purpose registers of the core are named R0 - R31. You can change the default names to "ZERO", "AT", "V0", "V1", ... with the command **SETUP.DIS** ,,,,,,, SPECIAL (9 commas to skip *don't care* parameters).

If implemented, GPR shadow register sets can be displayed with the command **Register.view /REGSET**.

Register.view /REGSET Current	; shows the GPR registers R0-R31 of ; the current context. ; it is equivalent to the command ; Register.view
Register.view /REGSET Previous	; in case of a register set change ; e.g caused by an exception, it ; shows the register set of the ; previous context.
Register.view /REGSET 15.	; shows GPR register set 15.

Memory Classes

The following MIPS specific memory classes are available.

Memory Class	Description
AP	Program Memory physically addressed
EAP	Run-time Program Memory (access also during running CPU), physically addressed
EP	Run-time Program Memory (access also during running CPU), virtually addressed
P	Program Memory virtually addressed
AD	Data Cache / Memory physically addressed.
D	Data Memory virtually addressed
EAD	Data Memory via DMA (access also during running CPU), physically addressed
ED	Data Memory via DMA (access also during running CPU), virtually addressed
CBU	CBUS Register (only for MDED)
CC0	Coprocessor 0 Control Register (only for Lexra cores)
CP0	Coprocessor 0 Register
CP1	Coprocessor 1 Register (if implemented)
CP2	Coprocessor 2 Register (if implemented)
CP3	Coprocessor 3 Register (if implemented)
DBG	Debug Memory Class (gives additional information)
E	Emulation Memory, Pseudo Dualport Access to Memory (see SYSystem.MemAccess and SYSystem.CpuAccess)
ECBU	CBUS Register (only for MDED) (access also during running CPU)
VM	Virtual Memory (memory on the debug system)
IC	Virtually addressed Instruction Cache
AIC	Physically addressed Instruction Cache
DC	Virtually addressed Data Cache
ADC	Physically addressed Data Cache
NC	Uncached memory access.
ANC	Physically addressed Data Memory without Cache

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

Examples:

“Data.dump CP0:0--3” displays the register 0 (Index), 1 (Random), 2 (EntryLo0), 3 (EntryLo1) of the System Control Coprocessor (=CP0).

The register number can have values between 0 and 31. The value of “select” must be multiplied by 32 and added to the register number. “Data.dump CP0:0x30--0x30” displays the Config1 register (register number: 0x10; select: 0x01). Select is 0 for the registers mentioned above.

Virtual Memory could be helpful, if the memory of the target should not be used e.g. to load and examine a program.

ICD-MIPS64: For the memory classes CPx and DBG are only 64-bit (QUAD) write accesses possible.

SPR Memory Overlay

In case Target Scratch Pad RAM is available and enabled all TRACE32 accesses for the defined SPR address range are automatically redirected to the referring physical memory.

TRACE32 PowerView does not support a SPR memory class which forces SPR memory access!

If SPR is implemented, the current SPR settings could be seen and changed within Cache Control peripheral window. It could be found within MIPS drop down list in the tool bar.

B::CACHE					
Config	81840483	M	Config1 on	K23	Cached write-through
		KU	Cached write-through	ISP	Present
		DSP	Present	SB	Disabled
		MDU	Fast Multiplier Array	MM	Full
		BM	Sequential	BE	Little
		AT	MIPS32	AR	Release 2
		MT	TLB	K0	Cache/no coh./wr-thr/no wr-alloc
Config1	BEE3719E	M	Config2 on	MMUSize	20
		IS	512	IL	32 bytes
		DS	512	DL	32 bytes
		PC	Implemented	WR	Implemented
		CA	Implemented	EP	Present
		FP	Not implemented		
Config2	80000000	M	Config3 on	TU	Not implemented
		TS	64	TL	Disabled
		TA	Direct mapped	SU	0
		SS	64	SL	Disabled
		SA	Direct mapped		
TagLo	00008000	PA	000020	Valid[3]	Invalid
		Valid[1]	Invalid	Valid[0]	Invalid
		LRF	Low	L	Unlocked
DataLo	3B3A3938				
ErrCtl	00010000	WST	Disabled	SPR	Disabled
ErrorEPC	00000000				
I-SPR	00008080	Base	00008000	En	enabled
	00008000	Size	00008000		
D-SPR	00008080	Base	00008000	En	enabled
	00008000	Size	00008000		

Instruction SPR accesses are handled only for virtual addresses within KSEG0 and KSEG1 and for physical addresses!

Following examples refer to the ISPR and DSPR settings in the CACHE window above and a 1:1 virtual to physical address mapping for KUSEG.

```
Data.Set D:0x80008000 0x11 ; Write 0x11 to the first address  
; of the data SPR via KSEG0 access.  
  
Data.Set P:0xA0008000 0x22 ; Write 0x22 to the first address  
; of the instruction SPR via KSEG1  
; access.  
  
Data.In AM:0x8000 ; Read first address of instruction  
; SPR via physical access.  
  
Data.In M:0x8000 ; Read from virtual SDRAM address  
; 0x8000 via USEG access, because  
; ISPR accesses are only handled  
; for KSEG0 and KSEG1.  
  
Data.In D:0x8000 ; Read from first data SPR address.
```

SYStem.CONFIG

Configure debugger according to target topology

Format:

SYStem.CONFIG <sub_cmd> <parameter> ... <parameter>

SYStem.MultiCore (deprecated)

<sub_cmd>:

CORE <core> <chip>

CORENUMBER <number>

BaseCoreNumber <number>

BaseCoreOrder DESCENDING | ASCENDING

CMTap ON | OFF

IRPRE <bits> ... <bits>

IRPOST <bits> ... <bits>

DRPRE <bits> ... <bits>

DRPOST <bits> ... <bits>

DAPDRPOST <bits>

DAPDRPRE <bits>

DAPIRPOST <bits>

DAPIRPRE <bits>

CHIPIRPRE <bits>

CHIPIRPOST <bits>

CHIPDRPRE <bits>

CHIPDRPOST <bits>

DMAIRPRE <bits> ... <bits>

DMAIRPOST <bits> ... <bits>

DMADRPRE <bits> ... <bits>

DMADRPOST <bits> ... <bits>

BYPASS <pattern>

CJTAGFLAGS <flags>

CJTAGTCA <value>

DEBUGPORTTYPE [JTAG | SWD | CJTAG]

SWDPIDLEHIGH [ON | OFF]

SWDPTargetSel <value>

TAPState <state>

TCKLevel 0 | 1

TriState ON | OFF

Slave ON | OFF

state

BaseCore.Base <address>

BaseCore.RESet

BaseCore.view

GcrBaseAddress <address>

The four parameter IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the MIPS TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. MIPS + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**.

Debugging an SMP system, there are more than one core for which the Jtag chain must be defined within one TRACE32 PowerView instance. So the pre- and post bits will be defined for all cores within one command e.g. an SMP system with 3 cores must be configured as follows:

SYStem.CONFIG IRPRE 0. 5. 10. means core 0 has 0, core 1 has 5 and core 2 has 10 IRPRE bits.

SYStem.CONFIG IRPOST 10. 5. 0. means core 0 has 10, core 1 has 5 and core 2 has 0 IRPRE bits.

SYStem.CONFIGDRPRE 0. 1. 2. means core 0 has 0, core 1 has 1 and core 2 has 2 IRPRE bits.

SYStem.CONFIGDROST 2. 1. 0. means core 0 has 2, core 1 has 1 and core 2 has 0 IRPRE bits

If the CPU is defined in the CPU selection list, the configuration of the pre- and post-coordinates is predefined in the TRACE32 software, so there's nothing to be done by the user.

Some chip vendors implement an extra Chip TAP for controlling, among other things, the JTAG chain establishing. The position of the Chip TAP is determined by CHIPIRPRE, CHIPIRPOST, CHIPDRPRE and CHIPDRPOST. The Chip TAP position must be defined for the fully established JTAG chain which is not necessarily the case after reset!

To keep the JTAG chain with all TAPS alive a special bypass command has to be shifted in the IR register of the chip TAP with each JTAG transaction. This bypass command is defined with the BYPASS parameter.

The position of an optional EJTAG DMA TAP could be defined with the parameters DMAIRPRE, DMAIRPOST, DMADRPRE and DMADRPOST.

TriState has to be used if more than one debugger are connected to the common JTAG port at the same time. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode.

NOTE: nTRST must have a pull-up resistor on the target, EDBGREQ must have a pull-down resistor.

CORE For multicore debugging one TRACE32 PowerView GUI has to be started per core. To bundle several cores in one processor as required by the system this command has to be used to define core and processor coordinates within the system topology.

CoreNumber Set number of cores per SMP system.

BaseCoreNumber Set number of base cores. For cores, consisting of a cluster of base cores, the base core number has to be set for correct hardware resource assignment within the debug driver.

BaseCoreOrder	Set ordering rule for base cores. Ascending order means that base core 0 is next to TDI, descending order means core 0 is next to TDO. Ascending : TDI --> BaseCore 0 --> ... --> BaseCore n --> TDO Descending : TDI --> BaseCore n --> ... --> BaseCore 0 --> TDO Currently not used.
CoherenceManagerTap	Set if this core has an additional coherence manager tap. If necessary this option is set implicitly by the CPU selection. So that command is only needed for bringing up new MIPS cores and therefore is not mentioned in following configuration examples!
...IRPRE	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.
...IRPOST	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.
...DRPRE	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.
...DRPOST	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.
TAPState	This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.(default: 7 = Select-DR-Scan)
TCKLevel	Level of TCK signal when all debuggers are tristated. (default: 0)
TriState	The debugger switches to tristate mode after each JTAG access. Then other debuggers can access the port. This option is required if more than one debugger hardware is used share the same JTAG port. (default: OFF)

Slave	Only one debugger (master) is allowed to control the signals nTRST and nRST. If more than one debugger hardware is used to share the same JTAG port, all except the master must have this option active. (default: OFF)
CHIPIRPRE	Definition of a TAP in a scan chain that needs a different IR and DR pattern than the default BYPASS (1...1) pattern.
CHIPIRPOST	
CHIPDRPRE	
CHIPDRPOST	
CHIPIRLENGTH	
CHIPIRPATTERN	
CHIPDRLENGTH	
CHIPDRPATTERN	
 BYPASS	Special Chip TAP bypass pattern. (default: 0)
 DMAIRPRE	Definition of a DMA TAP in a scan chain.
DMAIRPOST	
DMADRPRE	
DMADRPOST	
 state	Show state.
 GcrBaseAddress	Set non default global control register base address. This command is only available if the core has a coherence manager block. The default Gcr Base Address is 0xBFBF8000.
 CJTAGFLAGS <i><flags></i>	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.
 DEBUGPORTTYPE [JTAG SWD CJTAG]	It specifies the used debug port type “JTAG”, “SWD”, “CJTAG”. It assumes the selected type is supported by the target.
	Default: JTAG.

SWDPIidleHigh [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 SWDPIidleHigh ON
	Default: OFF.
SWDPTargetSel <i><value></i>	Device address in case of a multidrop serial wire debug port. Default: none set (any address accepted).
... .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.
... .RESet	Undo the configuration for this component. This does not cause a physical reset for the component on the chip.
... .view	Opens a window showing the current configuration of the component.

Example for configuration of a chip with 3 cores

```
SYStem.CONFIG.CORE 1. 1.
SYStem.CONFIG.CORE 2. 1.
SYStem.CONFIG.CORE 3. 1.
```

Example for configuration of a Jtag daisy chain

TDI ---> ChipTAP ---> Mips1 ---> Mips2 ---> DMATAP ---> TDO

Instruction register length of

- ChipTap: 3 bit
- Mips1: 5 bit
- Mips2: 5 bit
- DMATap: 6 bit

The example below shows the commands necessary for setting up the Mips 1 Core:

SYStem.CONFIG.IRPRE	11.	IR Mips2 Core + DMA TAP
SYStem.CONFIG.IRPOST	3.	IR Chip TAP
SYStem.CONFIG.DRPRE	2.	DR Mips2 Core + DMA TAP
SYStem.CONFIG.DRPOST	1.	DR Chip TAP
SYStem.Up		

The example below shows the commands necessary for setting up the Mips 2 Core in a second TRACE32 PowerView instance:

SYStem.CONFIG.IRPRE	6.	IR DMA TAP
SYStem.CONFIG.IRPOST	8.	IR Chip TAP + Mips1 Core
SYStem.CONFIG.DRPRE	1.	DR DMA TAP
SYStem.CONFIG.DRPOST	2.	DR Chip TAP + Mips1 Core
SYStem.CONFIG.CORE 2. 1.		Assign Mips2 core to chip1 core2
SYStem.Up		

Note:

While defining the Mips2 core in a second TRACE32 PowerView instance (AMP System) it will get the core and chip coordinates 1, 2. But if the target is one chip with two cores inside we have to reassign the coordinates of the Mips2 core to core2 chip1 which is done by SYStem.CONFIG.Core 2. 1.

If the chip has an additional Chip Tap and the device is not yet supported by our debugger following settings have to be done before SYStem.Up.

SYStem.CONFIG.CHIPIRPRE 16.	IR Mips1 Core + Mips2 Core + DMA TAP
SYStem.CONFIG.CHIPIRPOST 0.	
SYStem.CONFIG.BYPASS 3.	Set special Chip TAP Bypass pattern.

If a chip provides EJTAG DMA access on an extra TAP these TAP could be defined with following commands.

SYStem.CONFIG.DMAIRPRE	0 .	
SYStem.CONFIG.DMAIRPOST	13 .	IR Chip TAP + Mips1 Core + Mips2 Core
SYStem.CONFIG.DMADRPRE	0 .	
SYStem.CONFIG.DMADRPOST	3 .	DR Chip TAP + Mips1 Core + Mips2 Core
SYStem.MEMACCESS DMA	Enable DMA Access in Debugger.	

Configuration of Mips34K

Mips34k may be used as a single or a dual core. Each core/vpe has its own TAP. The Jtag scan chain for a single MIPS34K core with two VPEs is

TDI ---> VPE0 ---> VPE1 ---> TDO

The Mips34k VPE0 and VPE1 TAP access is completely controlled within the T32 Mips debug driver. Therefore a single core/vpe Mips34k is debugged as all other single core chips and no multi core settings have to be set at all.

Depending on the number of opened PowerView instances and their Core-Chip assignment AMP debugging is automatically determined and supported by the debugger.

Setup of an SMP system

- Start one TRACE32 instance
- Select MIPS34K in CPU selection list
- Set TAP coordinates to VPE0 of referring Mips34K core.
- Set total number of implemented cores (threads)
- Define number of cores (threads) which participate the SMP system.

See below the configuration for a Mips34K single SMP system.

SYStem.CPU MIPS34K		; select Mips34k core
SYStem.CONFIG.IRPRE	0 .	; set TAP coordinates to VPE0 of
SYStem.CONFIG.IRPOST	0 .	; Mips34K core (default values) .
SYStem.CONFIG.DRPRE	0 .	
SYStem.CONFIG.DRPOST	0 .	

```

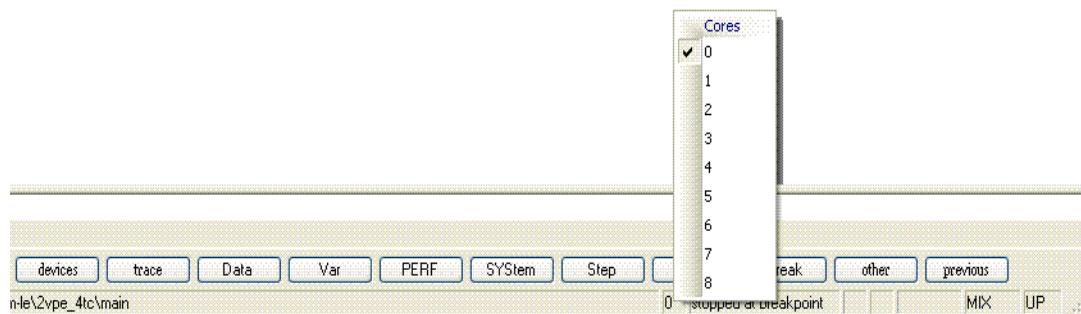
SYStem.CONFIG.CoreNumber 9. ; set total number of cores
                             ;(threads).

Core.Number 9. : assign all available cores
                  ;(threads) to one SMP system

SYStem.Up ; bring up debugger

```

PowerView shows always the context of the current core. A manual switching between the TCs (Thread Context) could be done with the **CORE** command or with help of the core drop down list.



Setup of a Multi TAP system:

The Mips34k may be used together with additional TAP's in the Jtag chain.

TDI ---> Chip TAP --> VPE0 ---> VPE1 ---> DMA Tap --> TDO

See below the configuration for a Mips34K system with additional Chip- (IR width=7 bit) and DMA TAP(IR width=6 bit) in the Jtag chain.

```

SYStem.CPU MIPS34K ; select Mips34k core

SYStem.CONFIG.IRPRE 6. ; set TAP coordinates to VPE0 of
SYStem.CONFIG.IRPOST 7. ; Mips34K core (default values).
SYStem.CONFIG.DRPRE 1.
SYStem.CONFIG.DRPOST 1.

SYStem.Up ; bring up debugger

```

Configuration of Mips1004K / 1004KMT / InterAptiv

The Mips1004k core is a cluster of up to 4 Mips 1004K base cores which are derived from the MIPS34K core. Therefore the configuration is mainly the same and only the differences will be described here. The Jtag scan chain for a MIPS1004K core with 4 base cores and two VPEs each is

Base Core 3 (BC3) Base Core 0 (BC0)
 TDI ---> VPE0 ---> VPE1 ---> ---> VPE0 ---> VPE1 ---> TDO

Since PRID Revision 0x2f the Mips 1004K core has additional multithreading capability an extra Coherence Manager TAP and an opposite numbering of the Base Cores. In that Case CPU selection has to be MIPS1004KMT instead. Below the Jtag Scan Chain for Mips1004KMT with same properties as above could be seen.

Base Core 3 (BC0) Base Core 0 (BC3) Coherence Manager
TDI ---> VPE0 ---> VPE1 --->---> VPE0 ---> VPE1 ---> CM ---> TDO

From debug configuration point of view the MIPSInterAptiv is equivalent to Mips1004KMT. So the following description is also valid for this core.

Setup of a single core system:

- Start one TRACE32 instance
- Select MIPS1004K /MIPS1004KMT in CPU selection list
- Set TAP coordinates to VPE0 of referring Mips1004K core.
- Set number of Base cores.
- Set total number of implemented cores
- Set number of cores which participate the system to 1.

See below the configuration for a Mips1004K single SMP system.

```
SYStem.CPU MIPS1004K / MIPS1004KMT ; select Mips1004k core
SYStem.CONFIG.IRPRE    0.             ; set TAP base coordinates of
SYStem.CONFIG.IRPOST   0.             ; Mips1004K core (default
SYStem.CONFIG.DRPRE    0.             ; values).
SYStem.CONFIG.DRPOST   0.

SYStem.CONFIG.BCN 4.                ; set number of Base Cores within
                                    ; MIPS 1004k core (default)

SYStem.CONFIG.CoreNumber 8.         ; set total number of cores

Core.Number 1.                    ; assign one core to system
                                    ; (default)

SYStem.Up                      ; bring up debugger
```

Setup of an SMP system:

- Start one TRACE32 instance
- Select MIPS1004K / 1004KMT in CPU selection list
- Set TAP coordinates to VPE0 of referring Mips1004K core.
- Set number of Base cores.
- Set total number of implemented cores
- Define number of cores which participate the SMP system.

See below the configuration for a Mips1004K single SMP system.

```
SYStem.CPU MIPS1004K / MIPS1004KMT ; select Mips1004k core
SYStem.CONFIG.IRPRE 0.                ; set TAP base coordinates of
SYStem.CONFIG.IRPOST 0.               ; Mips1004K core (default
SYStem.CONFIG.DRPRE 0.                ; values).
SYStem.CONFIG.DRPOST 0.

SYStem.CONFIG.BCN 4.                 ; set number of Base Cores within
                                      ; MIPS 1004k core (default)

SYStem.CONFIG.CoreNumber 8.          ; set total number of cores

Core.Number 8.                      ; assign all available cores to
                                      ; SMP system

SYStem.Up                           ; bring up debugger
```

Setup of an AMP system:

Start two or more PowerView instances.

- Select CPU MIPS1004K / MIPS1004KMT in all PowerView instances.
- Set TAP coordinates to referring Mips1004K core in all PowerView instances.
- Assign all other cores to the first Mips1004K core.
- Set number of Base Cores.
- Bring up the TRACE32 PowerView instances.

See below command sequence to bring up MIPS1004K with 4 Base Cores and 2 VPEs each as AMP system:

PV1 for BC0 VPE0

```
SYStem.CPU MIPS1004K
```

PV2 for BC0 VPE1

```
SYStem.CONFIG.CORE 1. 1.
```

PV8 for BC3 VPE1

```
SYStem.CPU MIPS1004K
SYStem.CONFIG.CORE 2. 1.
SYStem.CONFIG.Slave On
SYStem.MODE NODEBUG
...
...
SYStem.CPU MIPS1004K
SYStem.CONFIG.CORE 8. 1.
SYStem.CONFIG.Slave On
SYStem.MODE NODEBUG

SYStem.Up

GO (start booting of
all VPE's within
Base Cores BC0 to
BC3)

SYStem.MODE ATTACH
...
...
SYStem.MODE ATTACH
```

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

■ SYStem.LOCK

Displays the Instruction TLB or Data TLB MMU entries.

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

<cpu>: **ICD-MIPS32:**

**MIPS4K | MIPS4KC | MIPS4KEC |
MIPSM14K | MIPSM14KC |
MIPS24K | MIPS24KE |
MIPS34K |
MIPS74K |
MIPS1004K |
ADM5120 | ADM8686 |
AU1000 | AU1100 | AU1200 | AU1500 | AU1550
BCM1101 | BCM1103 | BCM1113 | BCM3349 | BCM3380 | BCM35230 |
BCM3549 | BCM3556 | BCM4704 | BCM471x | BCM4748 | BCM5331x |
BCM5350 | BCM5354 | BCM5358 | BCM5365 | BCM56xxx | BCM5836 |
BCM63268 | BCM6328 | BCM6338 | BCM6345 | BCM6348 | BCM6358 |
BCM6362 | BCM6368 | BCM6369 | BCM6550 | BCM6816 | BCM7111 |
BCM7312 | BCM7317 | BCM7318 | BCM7325 | BCM7335 | BCM7400 |
BCM7401 | BCM7402 | BCM7405 | BCM7407 | BCM7413 | BCM7420 |
C7108 |
COACH12 |
F731940 |
FALCON |
HIDTV_PRO_QX |
IKF6833 | IKF6834 | IKF6836 | IKF6850 | IKF6860 |
LX4X80 | LX4189 | LX5180 | LX5280 |
MDEB | MDED |
MP32,
MSP20xx | MSP71xx |
PIC32MX |
PNX8330 | PNX8331 | PNX8332 | PNX8335 | PNX8541 | PNX8542 | PNX8543
| PNX8932 | PNX8935 | PNX85500_MIPS4K | PNX85500_MIPS24K |
PSB21553 | PSB21653 |
RT3052 | RT3662,
RC32334 | RC32355 |
VGCA | VGCB | VCTH | VCTV |
WP3**

(For ICD-MIPS64, see next page.)

<cpu>:

ICD-MIPS64:

MIPS5K

**BCM1125 | BCM1250 | BCM1255 | BCM1280 | BCM1455 | BCM1480 |
BCM7038 |**

**CN30XX | CN31XX | CN38XX | CN50XX | CN54XX | CN55XX | CN56XX |
CN57XX | CN58XX | CN63XX |**

MSP8510 |

PXB4261 |

RM9000 |

TX4938 |

WIN1XX | WIN7XX

Selects the processor type.

Default selection:

- MIPS4K if the JTAG Debugger for MIPS4K is used.
- MIPS5K if the JTAG Debugger for MIPS5K is used.

SYStem.DETECT.CORENUMBER

Detect core number

Format:

SYStem.DETECT.CORENUMBER

MIPS64 only.

Detects the core number of the target and set up TRACE32 accordingly.

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

<frequency>: **5 kHz ... 25 MHz.**

Default frequency: 2 MHz.

Selects the EJTAG port frequency (TCK). The frequency affects e.g. the download speed and scrolling speed in dump windows.

It could be required to reduce the EJTAG frequency if there are buffers, additional loads or high capacities on the EJTAG lines or if the target voltage (VIO) is very low. A very high frequency will not work on all systems and will result in an erroneous data transfer. Therefore we recommend to use the default setting if possible.

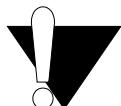
<frequency> The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the **SYStem.state** window.

If you want to enter a decimal value, please do not forget the dot “.” at the end of the number. Otherwise it is taken hexadecimal. Besides a decimal number like “100000.” short forms like “10kHz” or “15MHz” can also be used. The short forms imply a decimal value, although no “.” is used.

RTCK The JTAG clock is controlled by the RTCK signal (**Returned TCK**). This signal isn't a standard pin of the Mips Jtag connector.

Example:

```
SYStem.JtagClock RTCK
```



The clock mode RTCK cannot be used if a common debug cable with 14-pin flat cable (LA-7760) is used. A special dongle must be ordered. And it is required that the target provides an RTCK signal.

Format:	SYStem.LOCK [ON OFF]
---------	-------------------------------

Default: OFF.

If the system is locked, no access to the EJTAG port will be performed by the debugger. While locked the EJTAG connector of the debugger is tristated. The intention of the **SYStem.LOCK** command is to give EJTAG access to a debugger for another core if the EJTAG port of both cores are multiplexed.

It must be ensured that the state of the MIPS core EJTAG state machine remains unchanged while the system is locked. To ensure correct hand-over between two debuggers, a pull-up or pull-down resistor on TCK and a pull-up resistor on /TRST is required. In case you use a pull-up at TCK, you have to inform the debugger about that -> "SYStem.CONFIG TCKLevel 1". VIO and GND should be kept connected or be re-connected first.

There is an additional plug on the debug cable on the debugger side. This signal can be used to detect if the EJTAG connector is tristated. If tristated also this signal is tristated, it is pulled low otherwise.

SYStem.MemAccess

Run-time memory access

Format:	SYStem.MemAccess <mode>
<mode>:	Enable Denied StopAndGo DMA DAP

This option declares how memory access can take place while the CPU is executing code (run-time memory access). The run-time memory access has to be activated for each window by using the memory class E: (e.g. Data.Dump ED:0x800000) or by using the format option %E (e.g. Var.View %E var1).

Enable not possible.
CPU (deprecated)

Denied Dualport access is blocked.

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

DMA

Direct memory access/dual port access allowed.

DAP

A run-time memory access is done via the CoreSight v2 Debug Access Port (DAP). This is only possible if a DAP is available on the chip and if the memory bus is connected to it.

NOTE: The debugger accesses the memory bus and cannot see caches.

```
Data.dump ED:0x80000100
```

```
Data.dump EAD:0x100
```

```
Var.View %E flags
```

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

SYStem.Attach (alias for SYStem.Mode Attach)
SYStem.Down (alias for SYStem.Mode Down)
SYStem.Up (alias for SYStem.Mode Up)

<mode>:
Down
NoDebug
Go
Attach
Up
StandBy

Down (default)	(Disables the debugger and keeps the CPU in reset.)
NoDebug	Disables the debugger. The state of the CPU remains unchanged. The EJTAG port is tristated.
Go	Resets the target and enables the debugger. The CPU is running. Program execution can be stopped by the break command or external trigger. This command is only allowed if SYStem.Option.FlowTrace is OFF.
Attach	No reset is performed. The CPU keeps running. Program execution can be stopped by the break command or external trigger. This command is only allowed when CPU is in NoDebug mode and when SYStem.Option.FlowTrace is OFF.
Up	Resets the target and sets the CPU to debug mode. After the execution of this command the CPU is stopped and all registers are set to the default level.
StandBy	Not implemented.

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

Default: AUTO.

Selects the number of displayed address digits in various windows, e.g. [List.auto](#) or [Data.dump](#).

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

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

Default: OFF.

This option has no function for the MIPS architecture.

If DCFREEZE is set on, the debugger leaves the data cache as far as possible unchanged. I.e. if data is written by the debugger, it will be written into the data cache if the corresponding line is loaded and valid in the data cache. If no cache line contains the address or the line isn't valid, the data will be written into main memory. This option has only effect for virtual addressing. If physical addresses are used, they will always be handled as if dcfreeze is set.

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

- | | |
|---------------------|---|
| ON (default) | If data memory is displayed (memory class AD:) the memory contents from the D-cache is read via dedicated cache opcodes. If D-cache is not valid the physical memory is read. |
| OFF | If data memory is displayed (memory class AD:) the memory contents from the D-cache is read via mapping to KSEG0 for addresses < 0x2000000 respectively via cached TLB entry for larger addresses. If D-cache is not valid the physical memory is read. |

SYStem.Option.DisMode

Define disassembler mode

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

<mode>:
AUTO
ACCESS
MIPS32
MIPS16
MICROMIPS
NANOMIPS
MIPSR6

Default: AUTO.

This command specifies the selected disassembler.

- | | |
|------------------|---|
| AUTO | Automatic selection of disassembler mode. The information provided by the compiler output format is used for the disassembler selection. If no information is available it has the same behavior as ACCESS. (default) |
| ACCESS | Disassembler mode will be selected by entered access class. |
| MIPS32 | The MIPS32 disassembler is used. |
| MIPS16 | The MIPS16 disassembler is used. |
| MICROMIPS | The microMIPS disassembler is used. |

NANOMIPS The nanoMIPS disassembler is used.

MIPSR6 The MIPS R6 disassembler is used.

Format:	SYStem.Option.Endianness [AUTO Little Big]
---------	---

Default: AUTO.

This option selects the byte ordering mechanism. If it is set to AUTO, the kernel mode endianness will be detected and selected.

SYStem.Option.EnReset

Control target system reset

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

Default: ON.

During **SYStem.Up** the target is reset by the debugger. If the target reset is to be inhibited for some reason in general, this can be done with the command **SYStem.Option.EnReset OFF**. Note that it is recommended to leave the option ON because it ensures a more robust startup of the debug session. Consider using **SYStem.Mode.Attach** instead of **SYStem.Up** if you don't want to issue a target reset during the startup of the debug session.

Note that for multicore debug sessions only the master session issues a system reset.

SYStem.Option.EnTRST

Control TAP reset

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

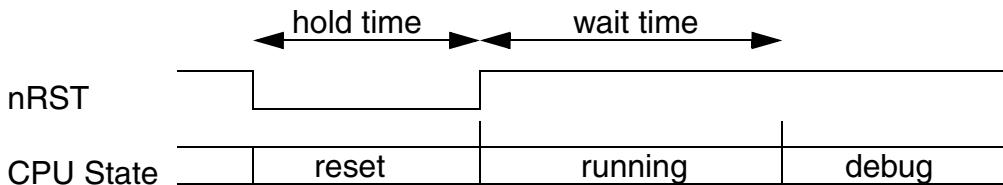
Default: ON.

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

Format: **SYStem.Option.HoldReset [<time>]**

Default: 300ms

With this option the default reset hold time could be set to a user-defined value.



SYStem.Option.FlowTrace

Define operating mode of RISC TRACE

Format: **SYStem.Option.FlowTrace [ON | Real-Time | OFF]**

Default: OFF.

Flow Trace must be switched to ON or Real-Time, if a Trace module is used. Using no trace FlowTrace must be switched off, otherwise a correct working of the debugger can't be guaranteed.

On Real-Time the processor is not stalled if the trace port can not output all data in real time, trace data get lost. On ON the processor will be stalled until all trace data have been transferred.

SYStem.Option.FREEZE

Freeze system timer in stop mode

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

Enabling this option will lead the debugger to stop the target CPU system timer since entering stop mode.

SYStem.Option.ICFLUSH

Flush of instruction cache during step and go

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

Default: OFF.

If this option is ON the instruction cache will be invalidated automatically before debug mode will be left (in case of a [Step](#) or a [Go](#)).

SYStem.Option.ICREAD

Use ICACHE for program read

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

ON

If program memory is displayed (memory class AP:) the memory contents from the I-cache is shown if the I-cache is valid. If I-cache is not valid the physical memory will be read. Typical command for program memory display are: [Data.List](#), [Data.dump](#).

OFF (default)

If program memory is displayed (memory class AP:) the memory contents from the physical memory is displayed.

SYStem.Option.IMASKASM

Disable interrupts while ASM single stepping

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

Default: OFF.

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

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

Default: OFF.

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

SYStem.Option.KEYCODE

Define key code to unsecure processor

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.

SYStem.Option.MCBreaksynch

Select break synchronization method

Format: **SYStem.Option.MCBreaksynch [MCBU | SOFT]**

Default: MCBU for CPUs with hardware MultiCore Breakpoint Unit support, SOFT otherwise.

In SMP mode all cores in an SMP system are required to stop synchronously when a breakpoint is hit. In CPUs with a *MultiCore Breakpoint Unit* (MCBU) the other cores can be stopped through a dedicated hardware interrupt once a core hits a breakpoint. In CPUs without MCBU a TRACE32 software loop is used to stop all SMP cores upon entry of debug mode. Since the hardware synchronization is much faster than the software solution it is used by default on CPUs that support it. However, if more than one SMP system is running on one CPU but the MCBU features only one synchronization channel, it might be necessary to set the MultiCore Break Synchronization of all but the first SMP system to *SOFT*. Thus, the breaking behavior of the SMP systems can be decoupled.

This option is not available for all CPUs.

MIPS 32 only

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

Default: ON.

Controls whether TRACE32 prefers a cached logical memory access over a (potentially uncached) physical memory access to keep caches updated and coherent.

NOTE: This option should usually not be changed.

ON	A cached logical memory access is used.
OFF	A (potentially uncached) physical memory access is used.

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

Default: OFF.

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

For an explanation of the TRACE32 concept of [address spaces](#) ([zone spaces](#), [MMU spaces](#), and [machine spaces](#)), see “[TRACE32 Concepts](#)” (trace32_concepts.pdf).

NOTE:

SYStem.Option.MMUSPACES should not be set to **ON** if only one translation table is used on the target.

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

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

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

Examples:

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

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

Default: 0.

This option selects an available memory area, where the debugger can load and execute a small program (48 bytes) to realize a fast download. See [SYStem.Option.TURBO](#).

SYStem.Option.OVERLAY

Enable overlay support

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

Default: OFF.

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

Example:

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

SYStem.Option.PROTECTION

Sends an unsecure sequence to the core

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

This option was made for unsecure protected debug interfaces. 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.

SYStem.Option.ResBreak

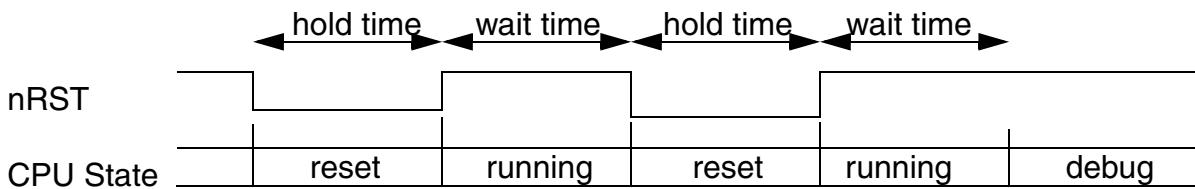
Halt the core after reset

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

Default: ON.

The common system-up procedure is that the debugger resets the target and forces the core into debug mode before any program will run. A prerequisite is that the TAP controller may be enabled during an asserted nRST line. Some cores have unwanted correlations between nRST and nTRST, so it isn't possible for the debugger to communicate with the core during reset. For those cores/boards (BCMxxxx and LX4x80/MDEB) nRST must be deasserted before the TAP may be reset. Thus will be done by the debugger, if ResBreak is switched off. For resetting all register values and allow debugging from the ResetVector an additional Reset pulse is asserted.

System.Option.ResBreak OFF:



SYStem.Option.STEPONCHIP

Use onchip breakpoints for ASM stepping

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

Default: OFF.

If this option is ON, onchip breakpoints are used for single stepping on assembler level instead of using the hardware single step feature of the CPU.

Use of STEPONCHIP ON:

On some CPUs the MIPS hardware single step feature does not function correctly in certain address ranges, e.g. due to hardware issues. The STEPONCHIP ON option allows to workaround such problems. Please note that STEPONCHIP ON has no effect if option STEPSOFT ON is used.

SYStem.Option.STEPSOFT

Use software breakpoints for ASM stepping

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

Default: OFF.

If this option is ON, software breakpoints are used for single stepping on assembler level.

Use of STEPSOFT ON for HLL debugging:

In several cases, the debugger executes an assembler single step by itself (e.g. continue on a breakpoint). If this single step results in a jump to an exception, the exception release come back to the breakpoint and the core stops at there again. STEPSOFT ON avoids this.

SYStem.Option.TURBO

Enable fast download

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

Default: OFF.

If TURBO is on, a fast download is possible. It will be assumed that the memory is uncached and can be accessed without errors. A program running on the target will be used to realize this fast download. A small program will be loaded at the location specified by [SYStem.Option.MonBase](#). This mode should be switched off after the download command is used, since it includes no error checks.

See [SYStem.Option.MonBase](#).

SYStem.Option.UnProtect

Unprotect memory addresses

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

Default: OFF.

If UnProtect is on, access to all addresses with entries in the TLB are possible. I.e. a write access is possible, although the access is set to "read only" in the target TLB. This option is often necessary for application debugging on Linux. If Linux marks pages as "read-only", setting a SW-breakpoint on those addresses will fail. To enable SW-breakpoint UnProtect must be switched on.

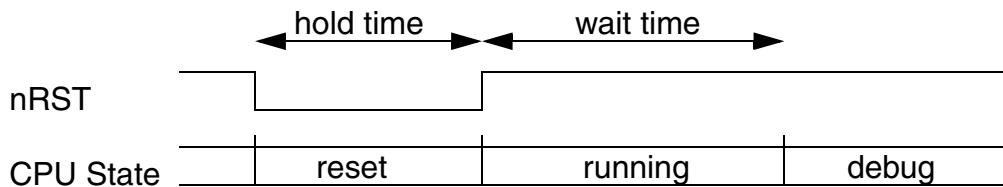
SYStem.Option.WaitReset

Set system reset wait time

Format: **SYStem.Option.WaitReset [<time>]**

Default: 300ms

With this option the default reset wait time could be set to a user-defined value. That could become necessary if the nRST hold time becomes extended by an onboard reset controller.



SYStem.Option.WATCHDOG

Disable hardware watchdogs

Cavium OCTEON only

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

Default: ON.

If set to ON, the hardware watchdog of CAVIUM OCTEON cores will be disabled upon debug mode entry. The debugger will set the watchdog mode in the CIU WATCHDOG registers to OFF.

Format: **SYStem.RESetOut**

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.

On-chip Breakpoints

TrOnchip.AddressMask

Define an address mask

Format: **TrOnchip.AddressMask <value> | <bitmask>**

TrOnchip.CORERESET

Halt at reset vector after core reset

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

OFF (default)	Don't stop the program execution at reset vector after any core reset.
ON	Stop the program execution at reset vector after any core reset

TrOnchip.RESet

Set on-chip trigger to default state

Format: **TrOnchip.RESet**

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

TrOnchip.StepVector

Halt on exception vector during step

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

Default: OFF.

Stepvector ON/OFF determines the behavior of a single step, when an exception or an interrupt occurs. If StepVector is ON, the core halts on the exception/interrupt routine, otherwise the core halts on the next instruction (after the instruction where the single step is performed).

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

Watchpoints instead of onchip breakpoints are used.

Default: OFF.

TrOnchip.stateDisplay on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

CPU specific MMU Commands

MMU.DUMP

Page wise display of MMU translation table

Format: **MMU.DUMP <table> [<range> | <address> | <range> <root> | <address> <root>]**
MMU.<table>.dump (deprecated)

<*table*>: **PageTable**
KernelPageTable
TaskPageTable <task_magic> | <task_id> | <task_name> | <space_id>:0x0
<cpu_specific_tables>

Displays the contents of the CPU specific MMU translation table.

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

< <i>root</i> >	The < <i>root</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.
< <i>range</i> > < <i>address</i> >	Limit the address range displayed to either an address range or to addresses larger or equal to < <i>address</i> >. For most table types, the arguments < <i>range</i> > or < <i>address</i> > can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. <ul style="list-style-type: none">• if <<i>range</i>> or <<i>address</i>> have a space ID: displays the translation table of the specified process• else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the MMU FORMAT command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable <i><task_magic></i> <i><task_id></i> <i><task_name></i> <i><space_id>:0x0</i>	Displays the MMU translation table entries of the given process. Specify one of the TaskPageTable arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. <ul style="list-style-type: none"> • For information about the first three parameters, see "What to know about the Task Parameters" (general_ref_t.pdf). • See also the appropriate OS Awareness Manuals.

CPU specific tables in MMU.DUMP <table>

TLB	Displays the contents of the Translation Lookaside Buffer. Displays the actual target TLB. Lines which are invalid will be displayed as empty lines. On the right side of table the contents of the belonging CP0 registers (pagemask, entryhi, entrylo0 and entrylo1) are displayed.
------------	--

MMU.FORMAT

Define MMU table structure

[[Examples](#)]

Format 1:	MMU.FORMAT <format> [<base_address> [<logical_kernel_address_range> <physical_kernel_address>]]
Format 2: MIPS64 only	MMU.FORMAT <format> [<base_address> [<base_address_highrange> [<logical_kernel_address_range> <physical_kernel_address>]]]

Default <format>: STD.

Defines the information needed for the page table walks, which are performed by TRACE32 for debugger address translation, page table dumps, or page table scans.

Format 1 is the normal, CPU-architecture independent command syntax. This format does *not* require the additional input parameter <base_address_highrange> of format 2.

Format 2: For MIPS64, there are four **MMU.FORMAT** <format> keywords which require the additional input parameter <base_address_highrange>. These keywords are **LINUX64**, **LINUX64RIXI**, **LINUX64HTLB**, and **LINUX64HTLBP16**.

<format>

<format> is to be replaced with a CPU architecture specific keyword which defines the structure of the MMU page tables used by the kernel. By default, TRACE32 assumes that the MMU format is **STD**, unless you specify the **MMU.FORMAT <format>** explicitly.

The table below indicates if a <format> requires the additional parameter <base_address_highrange>.

<format>	Description
STD	Standard format defined by the CPU
LINUX32	Linux 32-bit, page size 4kB
LINUX32RIXI	Linux 32-bit with RI/XI bits
LINUX32R4K	Linux 32-bit, page size 4kB, like LINUX32 but different page flags
LINUX32P16	Linux 32-bit, page size 16kB
LINUX32P16R2	Linux 32-bit, page size 16kB, used on MIPS32 R2 or R6 (internally identical to format LINUX32P16R41)
LINUX32P16R2	Deprecated: internally identical to format LINUX32P16R41
LINUX64	Linux 64-bit with 64-bit PTEs, page size 4kB. Separate page table for high address range can be specified with optional extra parameter <base_address_highrange>.
LINUX64P16	Linux 64-bit with 64-bit PTEs, page size 16kB. Depth 3 levels.
LINUX64P64	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 3 levels.
LINUX64P64LT	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 2 levels with large level 1 table (used for BROADCOM(R) XLP SDK 3.7.10 and alike)
LINUX64RIXI	Linux 64-bit with 64-bit PTEs with RI/XI bits, page size 4kB. Separate page table for high address range can be specified with optional extra parameter <base_address_highrange>.
LINUX64HTLB	Linux 64-bit with 64-bit PTEs, page size 4kB for huge TLB. Uses separate sub table for addresses > 0xFFFFFFFFFC0000000.
LINUX64HTLBP16	Linux 64-bit like LINUX64HTLB but pagesize 16kB.
LINUXBIG	Linux 32-bit with 64-bit PTEs on MIPS32
LINUXBIG64	Linux 32-bit with 64-bit PTEs on MIPS64
WINCE6	Format used by Windows CE6
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.

<base_address>

<base_address> defines the start address of the default page table which is usually the kernel page table. The kernel page table contains translations for mapped address ranges owned by the kernel.

The debugger address translation uses the default page table if no process specific page table (task page table) is available to translate an address.

<base_address> can be left empty by typing a comma or set to zero if there is no default page table available in the system.

<base_address_highrange>

Using <base_address_highrange>, you can specify a second page table responsible for the translation of addresses >= 0xFFFFFFFF00000000. Then, two page tables are in use:

- Addresses in range 0x0--0xFFFFFFFFFFFFFFFFFF will be translated with the page table defined by the argument <base_address>.
- Addresses in range 0xFFFFFFFF00000000--0xFFFFFFFFFFFFFFFFFF will be translated with the page table defined by the argument <base_address_highrange>.

<logical_kernel_address_range> and <physical_kernel_address> for the Default Translation

The arguments <logical_kernel_address_range> and <physical_kernel_address> define a linear logical-to-physical address translation for the kernel addresses, called *kernel translation* or *default translation*. This translation should cover all statically mapped logical address ranges of kernel code or kernel data.

For the <physical_kernel_address> you just need to specify the start address.

NOTE:	If no kernel translation is specified for a given memory access, TRACE32 tries to use static address translations defined by the command TRANSlation.Create . The kernel translation is shown in the TRANSlation.List window.
--------------	---

Examples

NOTE:	A backslash \ is used as a line continuation character in PRACTICE script files (*.cmm). No white space permitted after the backslash.
--------------	--

Examples of Format 1:

;	<format>	<base_address>	<logical_range>	<phys_range>
MMU.FORMAT LINUX		swapper_pg_dir		
MMU.FORMAT LINUX		swapper_pg_dir \	0xC000000000000000--0xc0000007FFFFFFFFFF	0x20000000

Examples of Format 2 with <base_address_highrange>:

```
;           <format>   <base_address>   <base_address_highrange>
MMU.FORMAT LINUX64    swapper_pg_dir    module_pg_dir

MMU.FORMAT LINUX64    swapper_pg_dir    module_pg_dir \
                      0xC000000000000000--0xC00000007FFFFFFF 0x20000000
;

;                           <logical_range>           <phys_range>
```

Examples of Format 2 without <base_address_highrange>:

In this example, not only the <base_address_highrange> is omitted but also all remaining parameters.

```
;           <format>   <base_address>   <base_address_highrange>
MMU.FORMAT LINUX64    swapper_pg_dir
```

If you need all parameters of Format 2 except for <base_address_highrange>, then use two commas to specify an empty input parameter.

```
;           <format>   <base_address>   <base_address_highrange>
MMU.FORMAT LINUX64    swapper_pg_dir

MMU.FORMAT LINUX64    swapper_pg_dir , , \
                      0xC000000000000000--0xC00000007FFFFFFF 0x20000000
;

;                           <logical_range>           <phys_range>
```

Format: **MMU.List <table> [<range> | <address> | <range> <root> | <address> <root>]**
MMU.<table>.List (deprecated)

<*table*>: **PageTable**
KernelPageTable
TaskPageTable <task_magic> | <task_id> | <task_name> | <space_id>:0x0

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

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

< <i>root</i> >	The < <i>root</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.
PageTable	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.
KernelPageTable	Lists the MMU translation table of the kernel. If specified with the MMU.FORMAT command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable <i><task_magic> </i> <i><task_id> </i> <i><task_name> </i> <i><space_id>:0x0</i>	<p>Lists the MMU translation of the given process. Specify one of the TaskPageTable arguments to choose the process you want.</p> <p>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.</p> <ul style="list-style-type: none"> • For information about the first three parameters, see “What to know about the Task Parameters” (general_ref_t.pdf). • See also the appropriate OS Awareness Manuals.

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

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

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

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

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. <ul style="list-style-type: none">• if <range> or <address> have a space ID: loads the translation table of the specified process• else, this command loads the table the CPU currently uses for MMU translation.
KernelPageTable	Loads the MMU translation table of the kernel. If specified with the MMU FORMAT command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.

TaskPageTable <i><task_magic></i> <i><task_id></i> <i><task_name></i> <i><space_id>:0x0</i>	<p>Loads the MMU address translation of the given process. Specify one of the TaskPageTable arguments to choose the process you want.</p> <p>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.</p> <ul style="list-style-type: none"> For information about the first three parameters, see “What to know about the Task Parameters” (general_ref_t.pdf). See also the appropriate OS Awareness Manual.
ALL	<p>Loads all known MMU address translations.</p> <p>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.</p> <p>See also the appropriate OS Awareness Manual.</p>

CPU specific Tables in MMU.SCAN <table>

TLB	Loads the translation table from the CPU to the debugger-internal translation table.
------------	--

MMU.Set

Set MMU registers

Format:	MMU.Set TLB < <i>index</i> > < <i>pagemask</i> > < <i>entryhi</i> > < <i>entrylo0</i> > < <i>entrylo1</i> >
---------	--

Sets the specified MMU TLB table entry in the CPU.

TLB	Writes data to the processor’s TLB. (Translation Lookaside Buffer)
< <i>index</i> >	Index of entry in target TLB.
< <i>pagemask</i> >	Content of pagemask register.
< <i>entryhi</i> >	Content of entryhi register.
< <i>entrylo0</i> >	Content of entrylo0 register.
< <i>entrylo1</i> >	Content of entrylo1 register.

Same command with same parameters as MMU.Set TLB. See command description above.

Command obsolete. Use MMU.Set TLB instead. Sets the specified MMU TLB table entry in the CPU.

The abbreviation TCB stands for Trace Control Block, and is the HW control interface to the MIPS hardware trace block. For details please refer to the MIPS Trace specifications. In the following TCB specific controlling and the referring commands are described.

TCB Control

The TCB triggering and filtering can be done in two ways:

- GUI based by the settings in the **TCB.state** combined with the breakpoint windows.
- Command line based by the TCB and **Break.Set** commands.

The triggering of the trace is controlled by the TraceOn and TraceOFF option of the break.set command. The trace trigger is non intrusive and therefore each break action use one onchip breakpoint resource. The number of available onchip breakpoints is implementation dependent and could be found in the instruction and data breakpoint status register.

```
Break.Set 0x4dd84 /Program /TraceON           ; start broadcasting the
                                                ; instruction flow after
                                                ; the instruction at the
                                                ; address 0x4dd84 was executed
                                                ; by the hardware thread 3

Break.Set 0x4ffa8 /Program /TraceOFF          ; stop broadcasting the
                                                ; instruction flow after
                                                ; the instruction at the
                                                ; address 0x4dd84 was executed
                                                ; by the hardware thread 3
```

Onchip trace filtering by data, CPU operation mode and, in case of multi thread or core devices, by CPU and tc number could be done with the TCB commands. In the example below, the TCB broadcasts only trace information for hardware thread 1 in user mode.

```
TCB.TRACETC TC1                           ; broadcast trace information
                                            ; only if TC1 execute
                                            ; instructions.

TCB TRACEKE OFF                          ; switch off broadcasting in
                                            ; kernel mode.

TCB TRACESV OFF                          ; switch off broadcasting in
                                            ; supervisor mode.

TCB TRACEEX OFF                          ; switch off broadcasting in
                                            ; exception mode.
```

In case of combined trace trigger and CPU operation mode filtering, the operation mode filtering has no effect!

A description of all **TCB** commands can be found in “[General Commands Reference Guide T](#)” (general_ref_t.pdf).

Configuring your FPGA

Before you can start debugging, your FPGA has to contain a valid design. The design has to include a Mips core, for which JTAG debugging is enabled.

You can use the debugger to configure your FPGA, if you provide a suitable BIT file.



Be sure to have correct multicore settings before configuring the FPGA (The settings are identical when connecting to the Mips core), otherwise the configuration will fail.

Also ensure that the debugger is in SYStem.down mode, before configuring your FPGA. Configuring the FPGA will break the communication link between the debugger and the Mips core, if your debugger is in **SYStem.Up** mode.

It is recommended to configure the target with the config option “JTAG dedicated” i.e. not using a mode where JTAG overrides other configurations like MSI, SPI etc. In the latter case configuration via TRACE32 may fail silently (no error message), though configuration via Xilinx Impact works.

Using JTAG for FPGA configuration

Dependant on the above mentioned conditions FPGA configuration is possible with a TRACE32 command.

Mechanical Description of the 14-pin EJTAG Connector

This connector is defined by MIPS in the EJTAG specification revision 2.5 and we recommend this connector for all future designs. Our debugger is supplied with this connector:

This connector does not provide trace signals, since the new EJTAG specification revision 2.5 does not yet include a trace definition. The trace feature will currently be redefined by MIPS. We expect that an additional connector will be specified for the trace signals.

Signal	Pin	Pin	Signal
TRST-	1	2	GND
TDI	3	4	GND
TDO	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RST-	11	-	Key
DINT	13	14	VIO (Reference Voltage)

This is a standard 14 pin double row connector (pin-to-pin spacing: 0.100 in.).

On target side a common pin strip with or without housing, for example SAMTEC: TSW-107-23-L-D can be used. Pin12 should be removed to provide mechanical keying.

Electrical Description of the 14-pin EJTAG Connector

- The input and output signals are connected to a supply translating transceiver (74ALVC164245). Therefore the ICD can work in an voltage range of (1.5 V) 1.8 ... 3.3 V (3.6 V). Please note that a 5 V supply environment is not supported! This would cause damage on the ICD. Please contact us for alternate solutions if you need to work with 5V.
- VIO is used as a sense line for the target voltage. It is also used as supply voltage for the supply translating transceiver of the ICD interface to make an adaptation to the target voltage (1.5 V) 1.8 ... 3.3 V (3.6 V).
- nTRST, TDI, TMS, TCK are driven by the supply translating transceiver. In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the EJTAG port. In environments where multiple tools can access the EJTAG port, it is absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a hand-over between different tools.
- TDO is an ICD input. It is connected to the supply translating transceiver.
- nRST is used by the debugger to reset the target CPU or to detect a reset on the target. It is driven by an open collector buffer. A 47 k pull-up resistor is included in the ICD connector. The debugger will only assert a pulse on nRST when the SYStem.UP, the SYStem.Mode Go or the SYStem.RESetOUT command is executed. If it is ensured that the MIPS is able to enter debug mode every time (no hang-up condition), the nRST line is optional.
- DINT is driven by the supply translating transceiver. This line is optional. It allows to halt the program execution by an external trigger signal.
- key pin is blocked to avoid wrong connection

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

Mechanical Description of the 24-pin EJTAG Connector

This connector is used on IDT boards. It provides the debugger signals plus the signals required for the trace. This debug interface is based on an older MIPS EJTAG specification revision 1.5.3.. This interface is not available for ICD-MIPS64.

Signal	Pin	Pin	Signal
TRST-	1	2	GND
TDI/DINT	3	4	GND
TDO/TPC	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RST-	11	12	GND
PCST[0]	13	14	GND
PCST[1]	15	16	GND
PCST[2]	17	18	GND
DCLK	19	20	GND
DEBUGBOOT	21	22	GND
VIO (Reference Voltage)	23	24	GND

An adapter is available if only the debugger should be connected. If debugger and trace is used, the debugger can be plugged on the trace probe. The trace probe uses this connector type.

The connector on the tool side is the 1,27 mm pitch sockets from
SAMTEC: SFMC-112-T1-S-D

As an appropriate connector on the target side can for example be used
SAMTEC: FTS-112-... (LIF) or FW-112-... (LIF) or DIS5-112-...

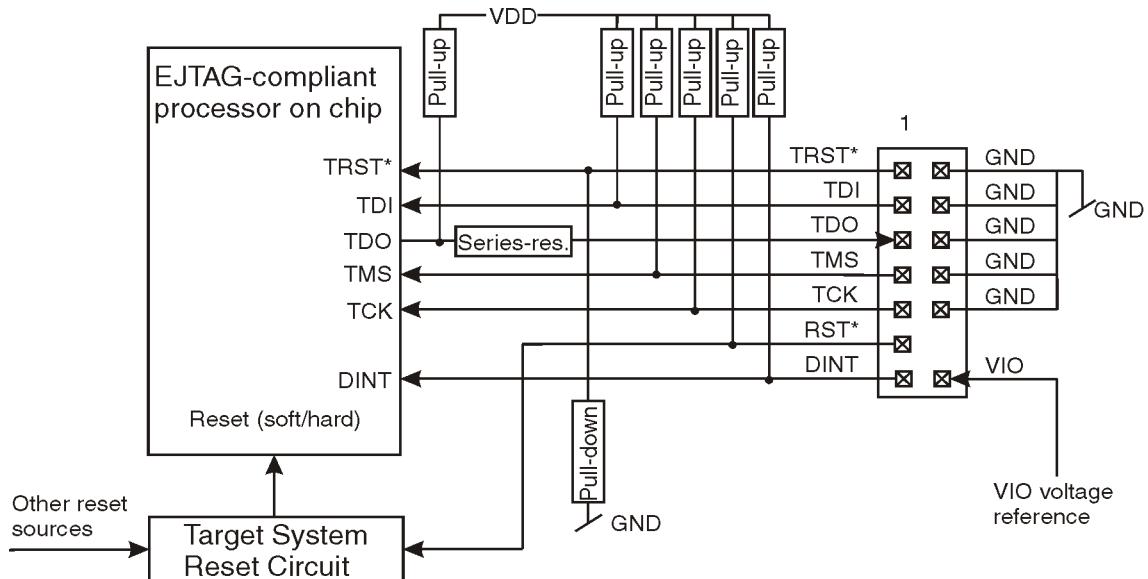
Electrical Description of the 24-pin EJTAG Connector

- The input and output signals are connected to a supply translating transceiver (74ALVC164245). Therefore the ICD can work in an voltage range of (1.5 V) 1.8 ... 3.3 V (3.6 V). Please note that a 5 V supply environment is not supported! This would cause damage on the ICD. Please contact us for alternate solutions if you need to work with 5 V.
- VIO is used as a sense line for the target voltage. It is also used as supply voltage for the supply translating transceiver of the ICD interface to make an adaptation to the target voltage (1.5 V) 1.8 ... 3.3 V (3.6 V).
- nTRST, TDI/DINT, TMS, TCK are driven by the supply translating transceiver. In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the EJTAG port. In environments where multiple tools can access the EJTAG port, it is absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a hand-over between different tools.
- TDO/TPC is an ICD input. It is connected to the supply translating transceiver.
- nRST is used by the debugger to reset the target CPU or to detect a reset on the target. It is driven by an open collector buffer. A 47 kΩ pull-up resistor is included in the ICD connector. The debugger will only assert a pulse on nRST when the SYStem.UP, the SYStem.Mode Go or the SYStem.RESetOUT command is executed. If it is ensured that the MIPS is able to enter debug mode every time (no hang-up condition), the nRST line is optional.
- Debugboot is driven by the supply translating transceiver. This line is optional. This line is currently not used, but will probably be used in the future for additional features.
- The signals DCLK, PCST0, PCST1, PCST2 are only connected to the trace tool if a trace tool is used. Otherwise they are not required. TDO/TPC is used by the trace and the debugger (see above).

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

Recommended JTAG Circuit on Target

MIPS recommends to configure the electrical JTAG connection as shown in the schematic below. LAUTERBACH's JTAG adapters are conform to this proposal.



Referring to MIPS specification, the recommended pull-up/pull-down resistor is $1\text{ k}\Omega$, the recommended serial resistor is $33\text{ }\Omega$.

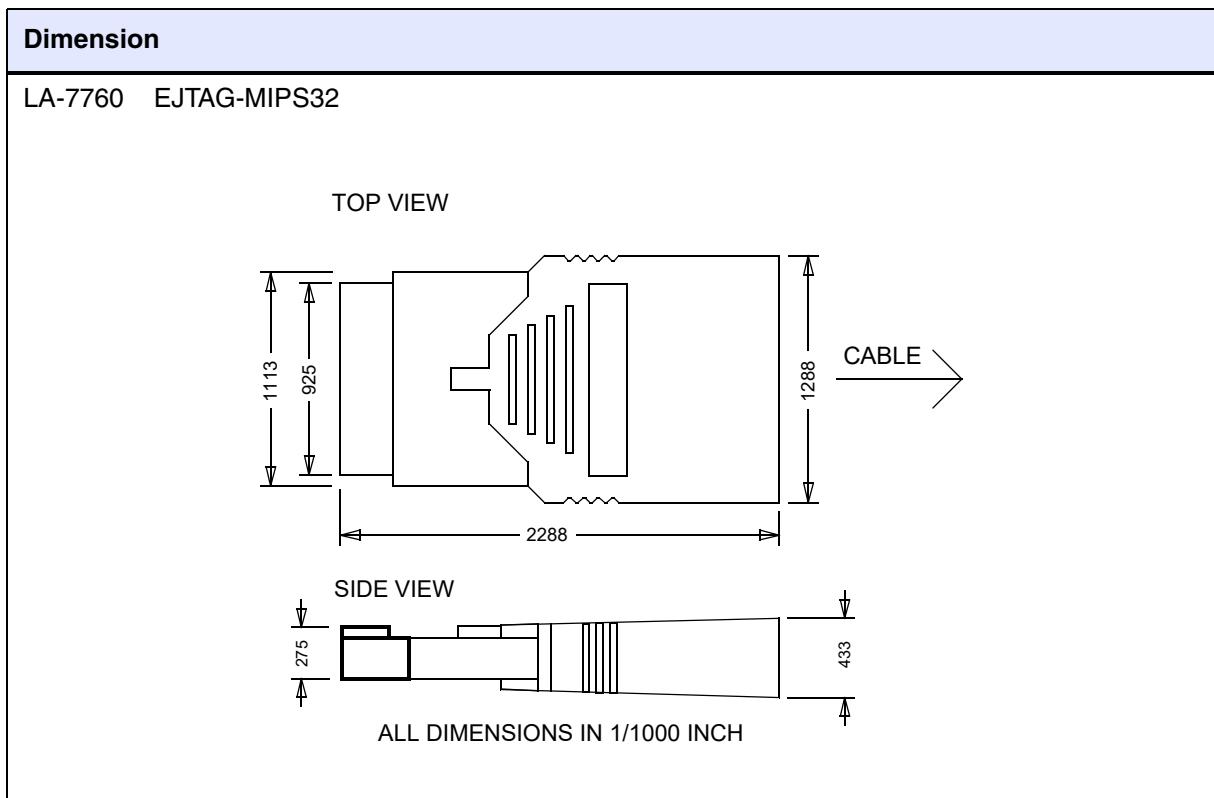


On some evaluation boards, there is a pull-up resistor on VIO. Since the LAUTERBACH JTAG adapter is supplied by target's VIO, a pull-up resistor is not allowed. In such a case, this resistor has to be bridged.

Operation Voltage

Adapter	OrderNo	Voltage Range
JTAG Debugger for MIPS32 (ICD)	LA-7760	1.8 .. 3.6 V
JTAG Debugger for MIPS64 (ICD)	LA-7761	1.8 .. 3.6 V

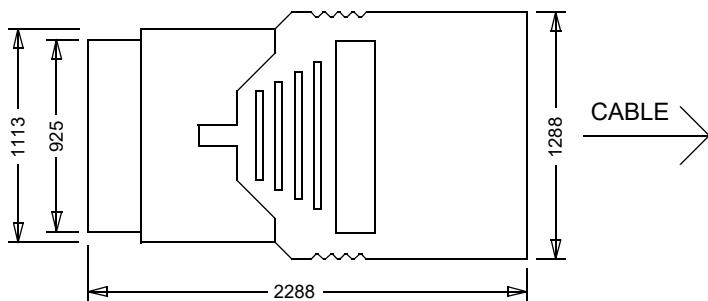
Mechanical Dimensions



Dimension

LA-7761 EJTAG-MIPS64

TOP VIEW



ALL DIMENSIONS IN 1/1000 INCH

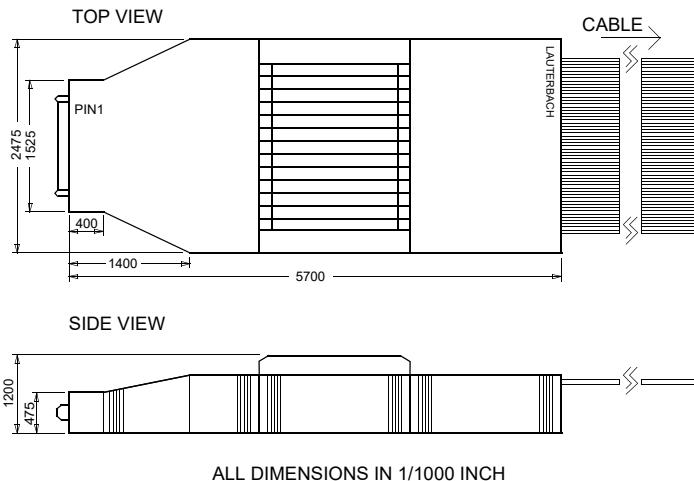
Operation Voltage

Adapter	OrderNo	Voltage Range
Preproc. for MIPS32 AUTOFOCUS 600 MIPI	LA-3906	1.8 .. 3.3 V
Preprocessor for MIPS flex cable	LA-7894	0.9 .. 3.3 V

Mechanical Dimensions

Dimension

LA-3906 PP-MIPS32-AF-2



LA-7894 PP-MIPS

