LAUTERBACH
DEVELOPMENT TOOLS

PPC600 Family Debugger

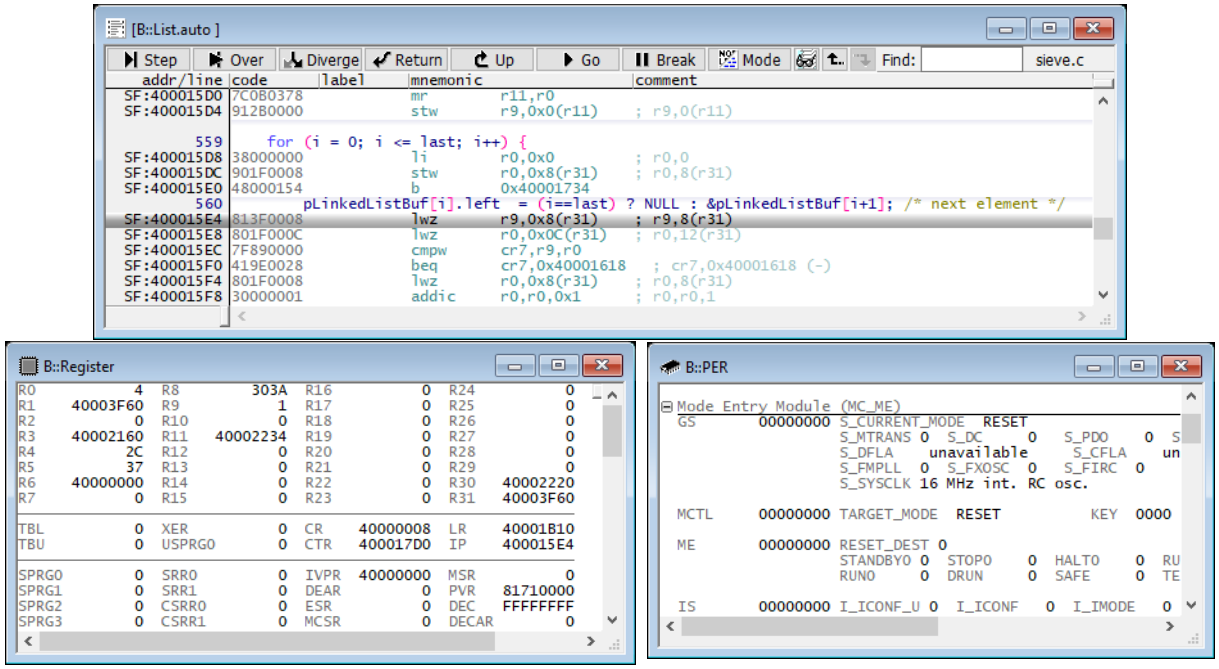
MANUAL

PPC600 Family Debugger

Version 06-Jun-2024

History

20-Jul-22 For the MMU.SCAN ALL command, CLEAR is now possible as an optional second parameter.



Introduction

This document describes the processor specific settings and features of TRACE32-ICD for the following CPU families:

- MPC603x
- MPC51xx, MPC5200, MPC5200B
- MPC7xx, MPC74xx
- MPC82xx, MPC83xx
- MPC86xx

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.

If some of the described functions, options, signals or connections in this Processor Architecture Manual are only valid for a single CPU or for specific families, the name(s) of the family(ies) is added in brackets.

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 PPC600 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/powerpc/ subfolder of the system directory of TRACE32.

Warning

Signal Level

All	The debugger drives the output pins of the BDM/JTAG/COP connector with the same level as detected on the VCCS pin. If the I/O pins of the processor are 3.3 V compatible then the VCCS should be connected to 3.3 V. See also **System.up errors**. Supported debug voltage: Debug cable with blue ribbon cable 2.5 … 5.0 V. Debug cable with gray ribbon cable 1.8 … 5.0 V (Available since 03/2004).

ESD Protection

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.

General

- Locate the JTAG / COP connector as close as possible to the processor to minimize the capacitive influence of the trace length and cross coupling of noise onto the JTAG signals. Don't put any capacitors (or RC combinations) on the JTAG lines.
- Connect TDI, TDO, TMS and TCK directly to the CPU. Buffers on the JTAG lines will add delays and will reduce the maximum possible JTAG frequency. If you need to use buffers, select ones with little delay. Most CPUs will support JTAG above 50 MHz, and you might want to use high frequencies for optimized download and upload performance.
- Ensure that JTAG $\overline{\text{HRESET}}$ is connected directly to the $\overline{\text{HRESET}}$ of the processor. This will provide the ability for the debugger to drive and sense the status of $\overline{\text{HRESET}}$. The target design should only drive $\overline{\text{HRESET}}$ with open collector/open drain.
- For optimal operation, the debugger should be able to reset the target board completely (processor external peripherals, e.g. memory controllers) with $\overline{\text{HRESET}}$.
- In order to start debugging right from reset, the debugger must be able to control CPU $\overline{\text{HRESET}}$ and CPU $\overline{\text{TRST}}$ independently. There are board design recommendations to tie CPU $\overline{\text{TRST}}$ to CPU $\overline{\text{HRESET}}$, but this recommendation is not suitable for JTAG debuggers.
- If the processor does not have $\overline{\text{QACK}}$/$\overline{\text{QREQ}}$ pins, leave the corresponding pins on the debug connector N/C.
- If the processor has $\overline{\text{QACK}}$/$\overline{\text{QREQ}}$ pins, $\overline{\text{QACK}}$ must be LOW in order to halt the core for debugging. If $\overline{\text{QACK}}$ is connected to the debug connector, the debugger can drive it LOW by command. If $\overline{\text{QACK}}$ is not connected to any system controller, it is recommended to tie it to GND.
- The debug cable uses VCCS on the JTAG-VREF pin to generate the power supply for the JTAG output buffers. The load on the JTAG-VREF pin caused by the debug cable depends on the debug cable version:

Gray ribbon cable	The VCCS pin is used as reference voltage for the internal power supply in the debug cable. This causes a load of about 50 kΩ. It is recommended to use a resistor with max. 5 kΩ to VCC, and max 1 kΩ for systems with VCCS = 1. 8V
Blue ribbon cable	The VCCS pin should be connected to VCC through a resistor with max. 10 Ω, as the output buffers are directly supplied by the VCCS pin.

Starting up the Debugger is done as follows:

1. Select the device prompt B: for the ICD Debugger, if the device prompt is not active after the TRACE32 software was started.

```
B:
```

2. Select the CPU type to load the CPU specific settings.

```
SYStem.CPU MPC8323
```

3. Tell the debugger where's FLASH/ROM on the target.

```
MAP.BOnchip 0xFF000000++0xFFFFFFFF
```

This command is necessary for the use of on-chip breakpoints.

4. Enter debug mode.

```
SYStem.Up
```

This command resets the CPU (HRESET) and enters debug mode. After this command is executed, it is possible to access the registers.

5. Show registers of on-chip peripherals.

```
PER.view
```

6. Set the chip selects to get access to the target memory.

```
Data.Set ANC:(IOBASE()|0x00010100) %Long 0xFF801801
Data.Set ANC:(IOBASE()|0x00010104) %Long 0xFF800EF4
;.....
```

7. Load the program and debug symbols.

```
Data.LOAD.Elf diabc.x
```

8. If the program was compiled on a different computer / environment, the source file path might have to be adopted.

```
Data.LOAD.Elf diabc.x /StripPART 5. /SOURCEPATH "L:\prj\src"
```

The option of the Data.LOAD command depends on the file format generated by the compiler. A detailed description of the **Data.LOAD** command is given in the **"General Commands Reference"**.

9. Set a breakpoint to the function to be debugged.

```
Break.Set main
```

10. Start application. The core will halt when the breakpoint is reached.

```
Go
```

11. Open windows to show source code, core registers and local variables. The window position can be specified with the **WinPOS** command.

```
Data.List
Register.view /SpotLight
Frame.view /Locals /Caller
```

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 may have the following reasons.

Error Message	Reason
target power fail	Target has no power or debug cable is not connected. Check if the JTAG VCC pin is driven by the target.
debugger configuration error	The debugger was not able to determine the connected processor. There are two possible reasons for this error: The CPU you are using is not supported by the used software, or a communication error prevented a correct determination. Check the AREA window for more information.
target processor in reset	The reset line is/was asserted by the **target** while the debugger performed a power-on reset. Try **SYStem.Option.SLOWRESET**, and check signal level of the JTAG HRESET pin.
emulation debug port fail	The debugger was unable to perform a power-on reset with the processor. Check all JTAG port signals.
emulation debug port fail target reset fail emulator debug port reset error	If the target reset is asserted for >500ms, or the target reset state is not reflected on the JTAG_HReset pin, **SYStem.Option.SLOWRESET** might be necessary.

Problems with Memory Access

For processors of some device families (esp. MPC82XX/MPC83XX), it is important that **no unimplemented memory addresses** are accessed by the debugger. Unimplemented memory means address ranges which would cause a **data access exception** when accessed by the target application in the current target state. Memory that is only available after target initialization, like SDRAM is unimplemented memory until initialized (e.g. a **Data.dump** window (or the stack view in the Register window) to SDRAM directly after reset). Also, virtual addresses are unimplemented if the memory management unit is currently disabled or the address unmapped (e.g. a Data.List window to Linux code at 0xC0000000 directly after reset).

The effects of accessing unimplemented memory are temporarily flickering memory windows up to permanently hanging memory buses, which can only be recovered by a reset. The debugger can rarely detect if a memory bus is hanging or not. Typical values displayed in dump/list windows are 0x00000000, 0xDEADBEE0, 0xDEADBEE1 or “????????” (bus error).

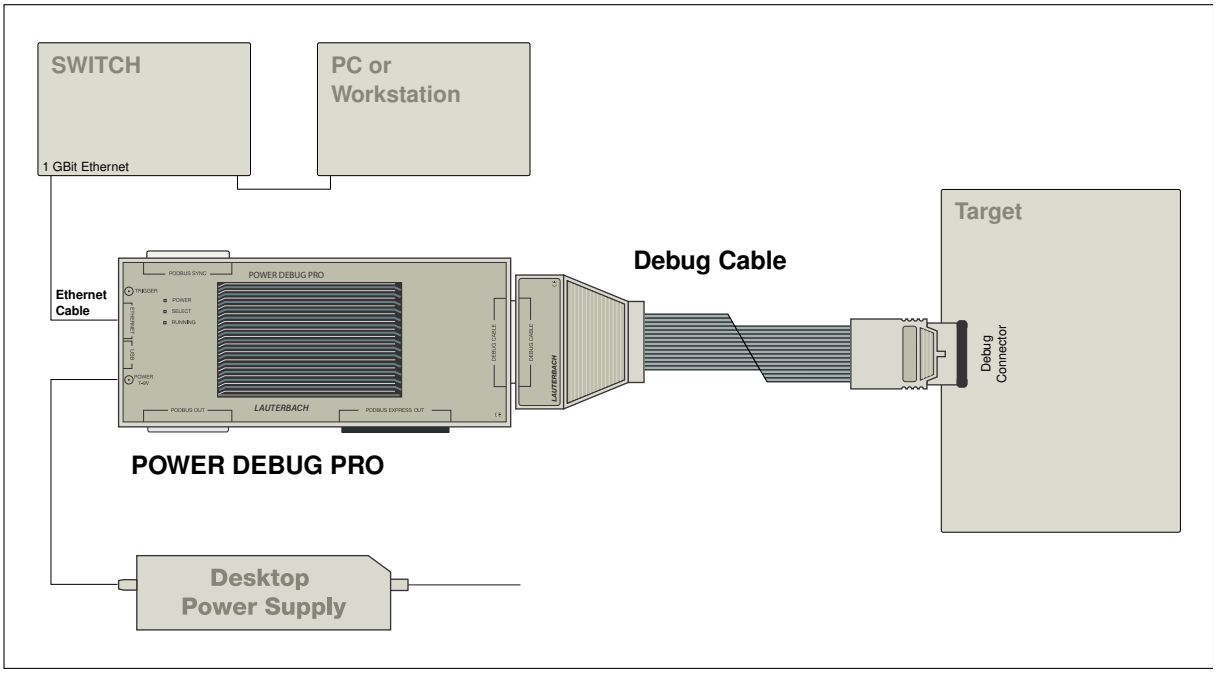
Hints for safe memory accesses:

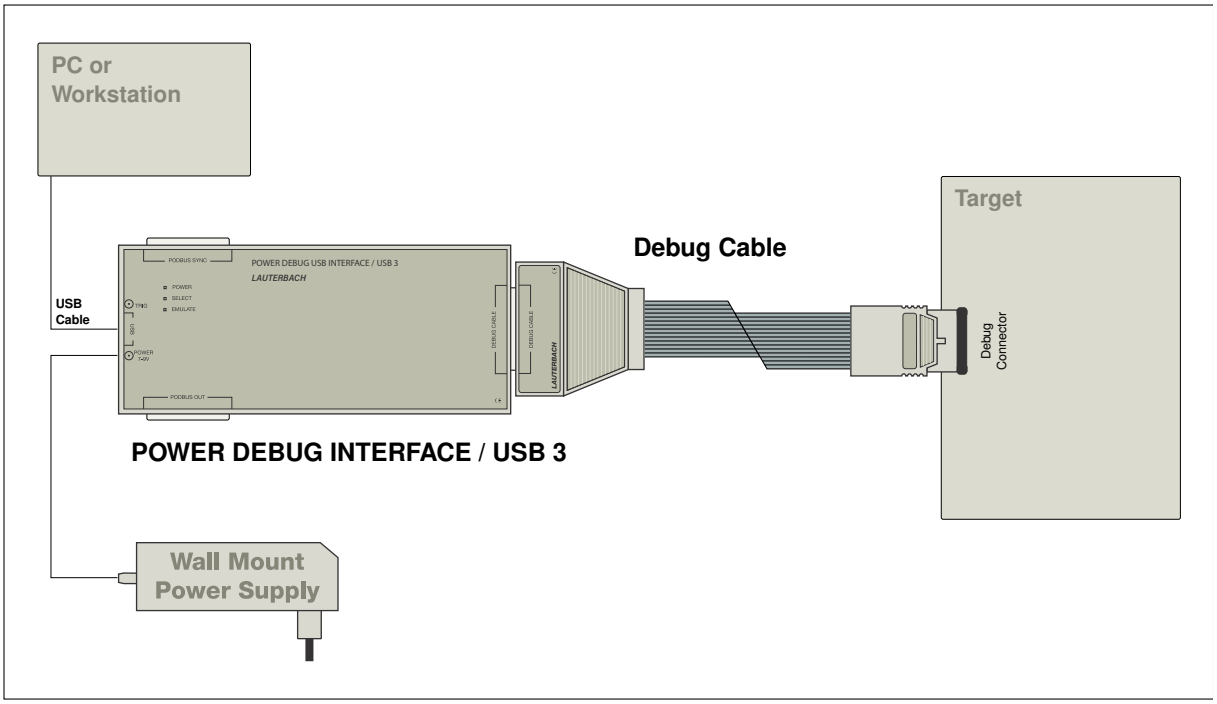
- directly after reset, set R1 to zero before opening the register window (which includes the stack view)
- directly after reset, close all windows that display data from SDRAM etc. which is not accessible directly after reset
- MPC82XX: close the peripheral view window before SYStem.UP. Usually the IMMR base address is different after reset and after target initialization. Always set the right base address with **SYStem.Option.BASE** before opening the peripheral view.
- Protect the debugger from accessing unimplemented memory using **MAP.DENYACCESS**.

FAQ

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

System Overview





POWER DEBUG INTERFACE / USB 3

Breakpoints

There are two types of breakpoints available: Software breakpoints (SW-BP) and on-chip breakpoints.

Software Breakpoints

Software breakpoints are the default breakpoints. They can only be used in RAM areas. There is no restriction in the number of software breakpoints. Please consider that setting a large number of software breakpoints will reduce the debug speed.

All	Since this CPU can only be stopped by an on-chip breakpoint, TRACE32-ICD sets an on-chip breakpoint to the Trap exception handler, whenever a software breakpoint is used. Because of that, software breakpoints can not be used if all on-chip breakpoints are directly used.
MPC60X, MPC7XX, MPC824X/6X, MPC74XX, MPC5100, MPC86XX	The current exception position must be known by the debugger at that time the SW-BP take place. See also **SYStem.Option.IP**.
MPC512X, MPC5200,MPC8280, MPC827X,MPC8247, MPC8248, MPX83XX	CPUs with at least two on-chip breakpoints can use **SYStem.Option.IP** BOTH. The debugger will set on-chip breakpoints to both interrupt addresses.

For software breakpoint functionality, the debugger must set an on-chip breakpoint to the program interrupt address. In some applications, especially during the target initialization stage, some applications have interrupts disabled and use the interrupt address range for non-interrupt code. In this situation, there are two possible workarounds:

- Configure CPU and debugger to use the interrupt addresses that are not used at this stage. This can be done by setting MSR_IP. Please note that the target application can modify this value any time.
- Force on-chip breakpoints to a different address until target initialization is finished. E.g. set the on-chip breakpoint to the address where the code at the interrupt addresses is not executed anymore. If this point is reached, clear the on-chip breakpoint and continue debugging. If the used CPU has more than one on-chip breakpoint, set the second breakpoint to an unused address

For software breakpoint functionality, the debugger must set an on-chip breakpoint to the program interrupt address. As PPC603-based cores have two possible interrupt addresses based on MSR[IP].

In situations where there are less than two on-chip breakpoints available there is a resource conflict. The unavailability can be caused by CPU design, or if the user makes direct use of on-chip breakpoints.

If the source code modifies MSR[IP], then a manual correction is necessary to use the correct exception handler.

Following some logic structure examples to explain this special situations.

Source code structure for all modes:

```
0x00        code
0x04        code
0x08        1. SW-BP
0x0C        code
0x10        code
0x14        code change MSR[IP] bit to 0
0x18        code
0x1C        2. SW-BP
0x20        code
…           …
```

AUTO-Mode:

Command Sequence / CPU Status	MSR[IP]	Exception Pos	Comment
CPU is stopped, PC at 0x00	1	1	
go	1	1	
CPU stop at 0x08	1	1	Break OK.
go	1/0	1	
CPU is still running	0	1	Break error!

Manual-Mode 0/1

Command Sequence / CPU Status	MSR[IP]	Exception Pos	Comment
CPU is stopped, PC at 0x00	1	1	
go	1	1	
CPU stop at 0x08	1	1	Break OK.
set sys.option.ip 0	1	0	
go	1/0	0	
CPU stop at 0x1C	0	0	Break OK.

Conclusion:

This means, if you know, that your source code will change the MSR[IP] bit and your first SW-BP will take affect after this alteration, so use the **SYStem.Option.IP** to select the right exception handler.

NOTE: If the target application uses page tables, software breakpoints can only be set to page tables which are already available. If it is necessary to set breakpoints in pages not yet mapped, only on-chip breakpoints can be used.

Software breakpoints can be overwritten by the target application, e.g. if a breakpoint is set in an area which will be loaded by a boot loader. Use on-chip breakpoints in this case.

The following list gives an overview of the usage of the on-chip breakpoints by TRACE32-ICD:

- **CPU family**
- **Instruction breakpoints**: Number of on-chip breakpoints that can be used for program and spot breakpoints
- **Read/Write breakpoints**: Number of on-chip breakpoints that can be used as read or write breakpoints. Can be only set on 8 byte boundaries
- **Data breakpoints**: 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.

CPU Family	Instruction Breakpoints	Read/Write Breakpoints	Data Value Breakpoints	Notes
PPC603 **RHPPC** **MPC8240** **MPC8245** **MPC8255** **MPC8260** **MPC8265** **MPC8266**	1 single address	—	—	Instruction breakpoint is not available if software breakpoints are used
MGT5100 **PPC7XX** **MPC74XX** **MPC86XX**	1 single address	1 single address	—	Instruction breakpoint is not available if software breakpoints are used
MPC512X **MPC5200** **MPC8247** **MPC8248** **MPC8270** **MPC8275** **MPC8280** **MPC83XX**	2 single addresses or 1 range	2 single addresses or 1 range		If software breakpoints are used, instruction breakpoints are reduced to one single address

NOTE: On-chip breakpoints can be cleared by the target application or by a target reset. If an on-chip breakpoint is not hit, first check (with the peripheral view), if the on-chip breakpoint is set or not.

Software Breakpoints in Interrupt Handlers

If software breakpoints are used in interrupt handlers, the registers SRR0 and SRR1 will be overwritten, because software breakpoints also use SRR0/1. There are several ways to debug interrupt handlers without corrupting SRR0/1:

- If MPC82XX, MPC5200 or RHPPC (G2_LE cores) is under debug, set **SYStem.Option.NOTRAPILL**.
- Use on-chip breakpoints. On-chip breakpoints will not corrupt SRR0/1. Please note that if only a single on-chip instruction address breakpoint is available, using the on-chip breakpoint will prevent using any further software breakpoints.
- Patch the interrupt handler, so that SRR0/1 are saved upon interrupt entry and restored before interrupt exit. If the interrupt handler it patched that way, it is safe to use software breakpoints after SRR0/1 have been saved.

Breakpoints in FLASH/ROM

If an instruction breakpoint is set, per default, the debugger tried to set a software breakpoint. If writing to the breakpoint address failed, the debugger will set an on-chip breakpoint.

With the command **MAP.BOnchip** *<range>* it is possible to inform the debugger where you have ROM (FLASH, EPROM) on the target. If a breakpoint is set within the specified address range, the debugger uses automatically the available on-chip breakpoints. Use this command, if write accesses to a read-only memory space are forbidden, e.g. because it could cause a reset etc.

Example:

```
MAP.BOnchip 0xFF800000--0xFFFFFFFF
```

Breakpoints on Physical or Virtual Addresses

On-chip breakpoints of almost all PPC603 based processors have a TE bit to configure if the breakpoint matches, if the access was performed on physical addresses (MSR_IR / MSR_DR off) of on virtual addresses (MSR_IR / MSR_DR on). In order to match, the processor compares IABR_TE / DABR_TE with MSR_IR for instruction and with MSR_DR for data accesses.

Per default, the debugger configures the breakpoints to match on physical addresses. In order to set the on-chip breakpoints to virtual addresses, use the command **TRANSlation.ON (Activate MMU translation)**. This command will enable MMU support, including breakpoint configuration.

Software breakpoints hit on virtual addresses if MSR_IR is set, and on physical addresses if MSR_IR is not set, regardless of any other configuration.

Examples for Breakpoints

Software Breakpoints

```
Break.Set 0x101000                      ; single address

Break.Set FooBar                        ; function name
```

Software breakpoints on ranges not possible.

On-chip Program Address Breakpoints

```
Break.Set 0xFFF00244 /onchip /program   ; single address

Break.Set 0xFFF00244 /onchip            ; single address

Break.Set MyFlashFunction /onchip       ; function name

Break.Set 0x2000--0x2fff /onchip        ; address range
```

NOTE: Address ranges are only possible with CPUs that have at least two on-chip program address breakpoints. The option `/program` is optional.

On-chip Data Address Breakpoints

```
Break.Set 0xFFF00244 /read              ; single address read

Break.Set 0xFFF00244 /write             ; single address write

Break.Set 0xFFF00244 /readwrite         ; single address any

Break.Set nMyValue /write               ; variable name

Break.Set 0x2000--0x2fff /readwrite     ; address range
```

Data address breakpoints of all PPC603e based cores will operate on 8 byte boundaries.

Access Classes

Access classes are used to specify how TRACE32 PowerView accesses memory, registers of peripheral modules, addressable core resources, coprocessor registers and the **TRACE32 Virtual Memory**.

Addresses in TRACE32 PowerView consist of:

- An access class, which consists of one or more letters/numbers followed by a colon (:)
- A number that determines the actual address

Here are some examples:

Command:	Effect:
Data.List P:0x1000	Opens a List window displaying **program** memory
Data.dump D:0xFF800000 /LONG	Opens a DUMP window at **data** address 0xFF800000
Data.Set SPR:415. %Long 0x00003300	Write value 0x00003300 to the **SPR** IVOR15
PRINT Data.Long(**ANC:**0xFFF00100)	Print data value at physical address 0xFFF00100

Access Classes to Memory and Memory Mapped Resources

The following memory access classes are available:

Access Class	Description
P	Program (memory as seen by core's instruction fetch)
D	Data (memory as seen by core's data access)
IC	L1 Instruction Cache (or L1 Unified cache)
DC	L1 Data Cache
L2	L2 Cache
NC	No Cache (access with caching inhibited)

In addition to the access classes, there are access class attributes: Examples:

Command:	Effect:
Data.List SP:0x1000	Opens a List window displaying **supervisor** program memory
Data.Set ED:0x3330 0x4F	Write 0x4F to address 0x3330 using **real-time memory access**

The following access class attributes are available:

Access Class Attributes	Description
E	Use real-time memory access
A	Given address is physical (bypass MMU)
U	TS (translation space) == 1 (user memory)
S	TS (translation space) == 0 (supervisor memory)

If an Access class attributes is specified without an access class, TRACE32 PowerView will automatically add the default access class of the used command. For example, **Data.List** U:0x100 will be changed to **Data.List** UP:0x100.

Access Classes to Other Addressable Core and Peripheral Resources

The following access classes are used to access registers which are not mapped into the processor's memory address space.

Access Class	Description
SPR	Special Purpose Register (SPR) access
PMR	Performance Monitor Register (PMR) access

SPRs and PMRs are addressed by specifying the register number after the access class.

Cache

Memory Coherency

The following table describes which memory will be updated depending on the selected access class

Access Class	D-Cache	I-Cache	L2 Cache	Memory (uncached)
DC:	updated	not updated	not updated	not updated
IC:	not updated	updated	not updated	not updated
L2:	not updated	not updated	updated	not updated
NC:	not updated	not updated	not updated	updated
D:	updated	not updated	updated	updated
P:	not updated	updated (*)	updated	updated

(*) Depending on the debugger configuration, the coherency of the instruction cache will not be achieved by updating the instruction cache, but by invalidating the instruction cache. See **"SYStem.Option ICFLUSH Invalidate instruction cache before go/step"** (debugger_ppc600.pdf) for details.

MESI States

The cache logic of e300, e600, e700 and PPC603e based cores is described as MESI states. This MESI state are represented in the CPU as the flags Valid and Dirty. The debugger will display both MESI state and the status flag representation.

State translation table:

MESI state	Flag
M (modified)	Valid && Dirty
E (exclusive)	Valid && NOT Dirty
S (shared)	Shared
I (invalid)	NOT Valid

Little Endian Operation

TRACE32 supports debugging processors which are operated in big-endian mode, true little-endian mode and modified (PowerPC) little-endian mode. The debugger switched to little-endian presentation, if MSR[LE] and one of the following system options is set:

- For true little-endian mode, enable **SYStem.Option.LittleEnd**.
- For modified (PowerPC) little-endian mode, enable **SYStem.Option.PPCLE**.

The following table lists processors which support one or both little endian modes:

Processor	true little endian	modified little endian	Notes
MPC603 **MPC745** **MPC750 (FSL)** **MPC755** **PPC750xx (IBM)**	—	Yes	
MPC5121 **MPC5123** **MPC5125**	Yes	—	e300 core only supports true little endian
MGT5100 **MPC5200**	Yes	Yes	HID2[TLE] == 1 for true little endian
MPC74XX	—	Yes	
MPC8247 **MPC8248** **MPC8271** **MPC8272**	—	Yes	
MPC83XX	Yes	—	e300 core only supports true little endian
MPC86XX	—	Yes	

SYStem.BdmClock — Set JTAG frequency

Format: **SYStem.BdmClock** <*frequency*>

<*frequency*>: **0.1** … **50.0 MHz.**

Default: 10.0 MHz

Selects the frequency for the debug interface. Usually, the maximum allowed JTAG frequency for PowerPC is 1/10th of the core frequency.

SYStem.CPU — Select the CPU type

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

<*cpu*>: **603EV2** | **750** | **8240** | **8260** | **7448** | **5200** …

Selects the CPU type. If the needed CPU type is not available in the CPU selection of the SYStem window, or if the command results in an error,

- check if the licence of the debug cable includes the desired CPU type. You will find the information in the VERSION window.
- check if the debugger software is up-to-date. Please check the VERSION window to see which version is installed. CPUs that appeared later than the software release are usually not supported. Please check **www.lauterbach.com** for updates. If the needed CPU appeared after the release date of the debugger software, please contact technical support and request a software update.
- if the CPU name matches one the names in the CPU selection. Search for the CPU name in the SYStem window, or type SYStem.CPU to the command line and click through the hot keys.

Format: **SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool.

SYStem.MemAccess Select run-time memory access method

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

Enable
CPU (deprecated)
Memory access during program execution to target is enabled.

Denied (default)
Memory access during program execution to target is disabled.

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

The run-time memory access has to be activated for each window by using the access class E: (e.g. **Data.dump** E:0x100) or by using the format option %E (e.g. Var.View %E var1). It is also possible to activate this non-intrusive memory access for all memory ranges displayed on the TRACE32 screen by setting **SYStem.Option.DUALPORT ON**.

NOTE: SYStem.MemAccess Enable is only available for the MPC86XX.

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** | **StandBy** | **Up**

Select target reset mode.

Down	Disables the Debugger. The debugger does not influence the target or the running application. The output signals of the debug cable are tristated.
NoDebug	Resets the target with debug mode disabled. In this mode no debugging is possible. The CPU state keeps in the state of NoDebug and the output signals of the debug cable are tristated.
Go	Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is in the system.up mode and running. Now, the processor can be stopped with the break command or until any break condition occurs.
Attach	Connect to the processor without asserting reset. The state of the target/application does not change. After this command the CPU is in the system.up mode and running.
StandBy	If this mode is used to start debugging from power-on. The debugger will wait until power-on is detected and then stop the CPU at the first instruction at the reset address. Not available for all PowerPC families covered by this manual.
Up	Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging. All register are set to the default value.

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

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

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

<tab>	Opens the **SYStem.CONFIG.state** window on the specified tab. For tab descriptions, see below.
DebugPort	Lets you configure the electrical properties of the debug connection, such as the communication protocol or the used pinout.
Jtag	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.

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

<parameter>: **CORE** *<core>*

<parameter>: **DRPRE** *<bits>*
(JTAG): **DRPOST** *<bits>*
IRPRE *<bits>*
IRPOST *<bits>*

CHIPDRLENGTH *<bits>*
CHIPDRPATTERN [**Standard** | **Alternate** *<pattern>*]
CHIPDRPOST *<bits>*
CHIPDRPRE *<bits>*
CHIPIRLENGTH *<bits>*
CHIPIRPATTERN [**Standard** | **Alternate** *<pattern>*]
CHIPIRPOST*<bits>*
CHIPIRPRE *<bits>*

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

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. ARM + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See **Daisy-chain Example**.
For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

TriState has to be used if several debuggers (“via separate cables”) are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. 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.

Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).

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.
Further information can be found in **SYStem.CONFIG.CORE**.

… **DRPOST** *<bits>*
(default: 0) *<number>* of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.

… **DRPRE** *<bits>*
(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.

… **IRPOST** *<bits>*
(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.

… **IRPRE** *<bits>*
(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.

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

CHIPDRPATTERN [**Standard** | **Alternate** *<pattern>*]
Data Register (DR) pattern which shall be used for BYPASS instead of the standard (1...1) pattern.

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

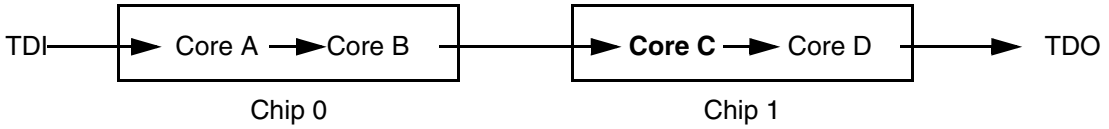
CHIPIRPATTERN [**Standard** | **Alternate** *<pattern>*]
Instruction Register (IR) pattern which shall be used for BYPASS instead of the standard pattern.

TAPState
(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.

TCKLevel
(default: 0) Level of TCK signal when all debuggers are tristated.

TriState (default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.

Slave (default: OFF) If more than one debugger 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).



Below, configuration for core C.

Instruction register length of

- Core A: 3 bit
- Core B: 5 bit
- Core D: 6 bit

```
SYStem.CONFIG.IRPRE  6.              ; IR Core D
SYStem.CONFIG.IRPOST 8.              ; IR Core A + B
SYStem.CONFIG.DRPRE  1.              ; DR Core D
SYStem.CONFIG.DRPOST 2.              ; DR Core A + B
SYStem.CONFIG.CORE 0. 1.             ; Target Core C is Core 0 in Chip 1
```

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.CONFIG.CHKSTPIN Control pin 8 of debug connector

Format: **SYStem.CONFIG.CHKSTPIN LOW | HIIGH**

Default: HIGH.

Controls the level of pin 8 (/CHKSTP_IN or /PRESENT) of the debug connector.

Format:	**SYStem.CONFIG.CORE** *<core_index> <chip_index>* **SYStem.MultiCore.CORE** *<core_index> <chip_index>* (deprecated)
<chip_index>:	**1** … **i**
<core_index>:	**1** … **k**

Default *core_index*: depends on the CPU, usually 1. for generic chips

Default *chip_index*: derived from CORE= parameter of the configuration file (config.t32). The CORE parameter is defined according to the start order of the GUI in T32Start with ascending values.

To provide proper interaction between different parts of the debugger, the systems topology must be mapped to the debugger's topology model. The debugger model abstracts chips and sub cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected, a generic chip or non-generic chip is created at the default *chip_index.*

Non-generic Chips

Non-generic chips have a fixed number of sub cores, each with a fixed CPU type.

Initially, all GUIs are configured with different *chip_index* values. Therefore, you have to assign the *core_index* and the *chip_index* for every core. Usually, the debugger does not need further information to access cores in non-generic chips, once the setup is correct.

Generic Chips

Generic chips can accommodate an arbitrary amount of sub-cores. The debugger still needs information how to connect to the individual cores e.g. by setting the JTAG chain coordinates.

Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a non-generic chip, two GUIs are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is valid since every new GUI uses a new *chip_index* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

Format: **SYStem.CONFIG DriverStrength** *<signal>* <**LOW** | **MID** | **HIGH**>

<signal>: **TCK**

Default: HIGH.

Configures the driver strength of the TCK pin.

Available for debug cables with serial number C15040204231 and higher.

SYStem.CONFIG.QACK

Control QACK pin

Format: **SYStem.CONFIG QACK TRISTATE** | **QREQ** | **LOW** | **HIGH**

Controls the level and function of pin 2 (/QACK) of the debug connector. Default: TRISTATE.

TRISTATE	Pin is disabled (tristate).
QREQ	Pin is driven to level of QREQ (pin 5).
LOW	Pin is driven to GND permanently.
HIGH	Pin is driven to JTAG_VREF permanently.

SYStem.Option.BASE Set base address for on-chip peripherals

Format: **SYStem.Option.Base** [**AUTO** | *<value>*]

MPC8260, MPC8280 and compatible

Set **SYStem.Option.BASE** to the base address of the internal memory map.The debugger uses this address to disable the watchdog and to show the memory mapped registers of the on-chip peripherals (see **PER**).

AUTO The debugger reads the RCW from FLASH to detect the base address of the internal memory map address. Only works during **SYStem.Up**. AUTO does not work, if the default reset configuration is used or if the RCW is only visible during reset (e.g. when provided by an EPLD).

<value> Use if AUTO does not work, if using **SYStem.Attach**, or if the application changes IMMR.

Before SYStem.Up:
If the default reset configuration is used, set value 0x00000000.
If the RCW is only visible during reset (e.g. when provided by an EPLD), set the appropriate value.

SYStem.Attach, or when application changes IMMR:
Set the value that the internal memory map address set by the application. Must be set correctly before core is halted.

MPC8240

SYStem.Option.BASE is not required and can be set to AUTO or 0x00000000.

MPC83XX, MPC512X, MPC5200, MPC86XX

The debugger will determine the current base address via JTAG access. This option has no effect.

PPC603x, PPC750xx, MPC755, PPC74XX

SYStem.Option.BASE is usually not required. It can be used to set the base address of the memory mapped registers of an external memory/peripheral controller (MPC10X, TSI1xx, MV6xxxx, etc.)

Format: **SYStem.Option.BUS32** [**ON** | **OFF**]

Default: OFF. Enable this option if the CPU is operated in the reduced 32-bit data bus width mode. This mode is often used in designs with PPC603e processors.

SYStem.Option.CONFIG Select RCW configuration

Format: **SYStem.Option.CONFIG** [**Master** | **Slave1..7**]

For MPC82XX only. When **SYStem.Option.BASE** is set to AUTO, this setting defines if the RCW is read from the location designated to the configuration master, or from one of the seven locations designated to the configuration slaves. By default setting, the debugger will read from the configuration master location.

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

Data.dump windows for access class D: displays the memory value from the data caches if valid. If no valid data is found in the caches, the physical memory will be read. If supported by the CPU unified L2/L3 caches will also be used if this system option is enabled

If caching is disabled via the appropriate hardware registers (HID0 for PPC603 Series) or cache is invalid, read and writes from/to memory will directly reflect to contents of physical memory even if a cache access class is selected.

The following table describes how DCREAD and ICREAD influence the behavior of the debugger commands that are used to display memory.

	DC:	**IC:**	**NC:**	**D:**	**P:**
ICREAD off DCREAD off	D-Cache	I-Cache	phys. mem.	phys. mem.	phys. mem.
ICREAD on DCREAD off	D-Cache	I-Cache	phys. mem.	phys. mem.	I-Cache
ICREAD off DCREAD on	D-Cache	I-Cache	phys. mem.	D-Cache	phys. mem.
ICREAD on DCREAD on	D-Cache	I-Cache	phys. mem.	D-Cache	I-Cache

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

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

Forces all list, dump and view windows to use the access class **E:** (e.g. **Data.dump E:**0x100) or to use the format option **%E** (e.g. **Var.View %E** var1) without being specified. Use this option if you want all windows to be updated while the processor is executing code. This setting has no effect if **SYStem.Option.MemAccess** is disabled or real-time memory access not available for used CPU.

Please note that while the CPU is running, MMU address translation can not be accesses by the debugger. Only physical addresses accesses are possible. Use the access class modifier “A:” to declare the access physical addressed, or declare the address translation in the debugger-based MMU manually using **TRANSlation.Create**.

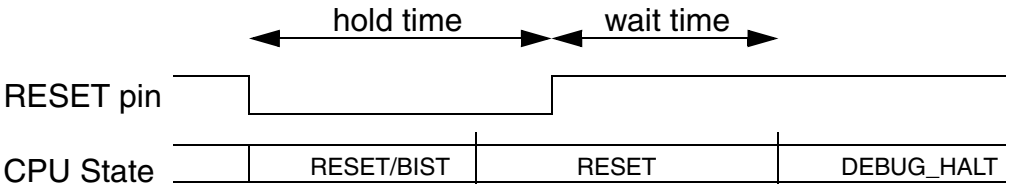
Format: **SYStem.Option.FREEZE** [**ON** | **OFF**]

When enabled, the core's timebase is stopped when the core is halted in debug mode. It is recommended to set this option ON.

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

<time>: **1us** … **10s**

Set the time that the debugger will drive the reset pin LOW, e.g. at **SYStem.Up**. The time must be longer than the BIST takes to complete. If called without parameter, the default reset hold time is used. The default reset hold time is 100ms for processors that require a BIST delay, else 100us.



See also **SYStem.Option.WaitReset** and **SYStem.Option.SLOWRESET**.

SYStem.Option.HOOK Compare PC to hook address

Format: **SYStem.Option.HOOK** *<address>* | *<address_range>*

The command defines the hook address. After program break the hook address is compared against the program counter value.

If the values are equal, it is supposed that a hook function was executed. This information is used to determine the right break address by the debugger.

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

MPC83XX and MPC512X only. Override the HRCW on SYStem.Up via JTAG. To disable this system option, call without parameter. This command is usually required to connect to a processor with erased/empty flash (HRCW not set).

<value> Hard reset configuration word (64 bit) in the order 0xHHHHHHHHLLLLLLLL

NOTE:

- The CPU will remember and use the overridden HRCW until the next power cycle or power-on reset.
- If JTAG_HRESET is connected to CPU_PORESET, **SYStem.Option.HRCWOVerRide** will only work in conjunction with **SYStem.Option.ResetMode JTAG_HRST**.

Usage:

```
SYStem.CPU MPC8360                                  ; select CPU
SYStem.Option.HRCWOVerRide 0x8060000004040006       ; desired HRCW
SYStem.Up                                           ; reset processor
SYStem.Option.HRCWOVerRide                          ; disable HRCW
                                                    ; override
```

SYStem.Option.ICFLUSH Invalidate instruction cache before go/step

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

ON Invalidates the instruction cache before starting the target program (Step or Go).

OFF Write accesses by the debugger to the memory of the class P: are performed in the instruction cache and the memory.

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

Data.List window and **Data.dump** window for access class P: displays the memory value from the instruction cache if valid. If I-cache is not valid the physical memory will be read. If supported by the CPU, L2 caches will also be used if this system option is enabled.

SYStem.Option.IMASKASM Disable interrupts while single stepping

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

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

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

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

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

NOTE: Don't enable this option for code that disables MSR_EE. The debugger will disable MSR_EE while the CPU is running and restore it after the CPU stopped. If a part of the application is executed that disables MSE_EE, the debugger can not detect this change and will restore MSE_EE.

Format: **SYStem.Option.IP** [**0** | **1** | **AUTO** | **BOTH**]

This option is used by the debugger to use the correct exception handler for the software breakpoints. See also **Software Breakpoints**.

AUTO	Depend on the current/last state of the MSR[IP] bit the debugger uses the lower or the higher exception handler.
0	Independent of the MSR[IP] the debugger uses the lower exception handler at 0x00000700.
1	Independent of the MSR[IP] the debugger uses the higher exception handler at 0xFFFF0700.
BOTH	Use both exception handler addresses. Only available for CPUs with two or more instruction address on-chip breakpoints (MPC8280, MPC83XX and compatible).

SYStem.Option.LittleEnd True little endian mode

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

Enable this system option if the PowerPC core is operated in **true little endian** mode. If the CPU is operated in modified (PowerPC) little endian mode, use command **SYStem.Option.PPCLittleEnd**.

To find out which mode is supported by the target processor, see **Little Endian Operation**.

SYStem.Option.MemProtect Enable memory access safeguard

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

PowerQuicc II (MPC824X, MPC826X, MPC827X, MPC8280) only.

This option can help to prevent a hanging memory bus caused by debugger accesses to unimplemented memory. USe together with **SYStem.Option.BASE AUTO**.

Usage:

- Set **SYStem.Option.BASE** to AUTO if RSTCONF is read from FLASH, or set the IMMR base address manually for any other options.
- **SYStem.Option.MemProtect** ON; CS7 or 11 will be enable on system.up for safe memory accesses
- SYStem.Up
- **SYStem.Option.BASE** AUTO; enable automatic IMMR change detection
- Start execution until the instruction that changes IMMR is reached, e.g. GO 0xFFF09038 /ONCHIP
- **Step.ASM**; assembler single step
- Now the debugger will use the new IMMR address for peripheral view and servicing the watchdog.

SYStem.Option.MemSpeed Configure memory access timing

Format:	**SYStem.Option.MemSpeed** <*value*>
<*value*>:	**1** (fastest) … **255** (slowest) **0** (default speed)

This option can be used to configure the access speed for memory accesses by the debugger. Only use this option when advised by Lauterbach.

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 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
```

SYStem.Option.NoDebugStop — Disable JTAG stop on debug events

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

Default: OFF.

On-chip debug events Breakpoint (instruction/data address), single step and branch trace can be configured to cause one of two actions. If a JTAG debugger is used, the CPU is configured to stop for JTAG upon these debug events.

If this option is set to ON, the CPU will be configured to not stop for JTAG, but to enter the breakpoint/trace interrupt, like it does when no JTAG debugger is used.

Enable this option if the CPU should not stop for JTAG on debug events, in order to allow a target application to use debug events. Typical usages for this option are run-mode debugging (e.g. with gdbserver) or setting up the system for a branch trace via LOGGER (trace data in target RAM) or INTEGRATOR.

Format:	**SYStem.Option.NOTRAP** *<type>*
<type>:	**OFF** | **FPU** | **ILL** **ON** (deprecated, same as **FPU**)

Defines which instruction is used as software breakpoint instruction.

OFF	Use TRAP instructions as software breakpoint (default setting). Software breakpoint will overwrite SRR0/1 registers.
FPU	Use an FPU instruction as software breakpoint. Gives the ability to use the program interrupt in the application without halting for JTAG. This setting only works if the application does not use floating point instructions (neither hardware nor software emulated). MSR[FP] must be set to 0 at all times. Software breakpoint will overwrite SRR0/1 registers.
ILL	Use an illegal instruction as software breakpoint. This setting is recommended for MPC82XX, MPC5200, RHPPC (G2/G2_LE cores) and MPC830X, MPC831X, MPC832X and MPC512X (e300c2/3/4). Gives the ability to use the program interrupt in the application without halting for JTAG. Illegal instructions as software breakpoints will preserve SRR0/1 registers.

If the program interrupt is required by the application, and both FPU and ILL are not usable, use **SYStem.Option.PINTDebug** as workaround.

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

Default: OFF.

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

OFF Disables support for code overlays.

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

Example:

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

SYStem.Option.PARITY Generate parity on memory access

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

Compute the parity bit for the Data.Set command to support memory with parity.

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

Software breakpoints for e300/e600 (former PPC603e based) cores are implemented using the TRAP instruction. However, the CPU will not stop for JTAG directly on the TRAP instruction. Instead, the TRAP instruction causes a program interrupt. To let the CPU stop for JTAG, the debugger sets an on-chip breakpoint to the program interrupt address (’0700).

The on-chip breakpoint at the program interrupt address will stop on all program interrupts, not just for TRAP instructions. If the cause of the program interrupt is other than TRAP, the debugger will print a message, the instruction pointer will be set to the instruction that caused the interrupt.

Enable this option, if it is necessary to execute program interrupts not caused by TRAP. The debugger will restart the CPU automatically. This event will be displayed in the status line. Please note that this feature has an impact on the real-time behavior, because the CPU will stop for a short time every time a program interrupt occurs.

NOTE:

- On some PowerPC core derivatives, **SYStem.Option.PINTDebug** can not support debugging of illegal instruction type program interrupts. In this case, illegal instructions halt the core similar to software breakpoints, but without affecting SRR registers).

  Affected processors are:
  - MPC82XX, MPC5200 and RHPPC (G2/G2_LE cores)
  - MPC830x (e200c4), MPC831x (e300c3) and MPC832x (e200c2)
  - MPC512x (e200c4)

- If **SYStem.Option.PINTDebug** is enabled, on-chip breakpoints at the first instruction of the program interrupt handler (’0700) are not possible. Set the on-chip breakpoint to ’0704 or other.

SYStem.Option.PPCLittleEnd PPC little endian mode

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

Enable this system option if the PowerPC core is operated in **modified (PowerPC) little endian** mode. If the CPU is configured for true little endian mode, use the command **SYStem.Option.LittleEnd**.

To find out which mode is supported by the target processor, see **Little Endian Operation**.

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

When OFF, the debugger will only evaluate BAT and ITLB/DTLB for address translation. When set to ON, the debugger will also evaluate the PTE table in memory for address translation.

Important: At the time this option is enabled, PTE table and SDR1 register have to be fully set up. If this option is enabled without PTE ready (or when memory is not yet initialized), wrong address translation or even general memory access fail can result. Related to this, make sure to disable this option before SYStem.Up or target reset.

SYStem.Option.RESetBehavior Set behavior when target reset detected

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

<mode>: **Disabled**
AsyncHalt

Defines the debugger's action when a reset is detected. Default setting is **Disabled**. The reset can only be detected and actions taken if it is visible to the debugger's reset pin.

Disabled No actions to the processor take place when a reset is detected. Information about the reset will be printed to the message **AREA**.

AsyncHalt Halt core as soon as possible after reset was detected. The core will halt shortly after the reset event.

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

<mode> **PIN** | **JTAG_PORST** | **JTAG_HRST** | **JTAG_SRST**

MPC83XX and MPC512x only. Default: PIN. Selects the method the debugger uses to reset the processor at SYStem.Up.

<mode>	Effect at **SYStem.Up**.
PIN	The reset pin (debug connector: pin 13) is asserted to reset the processor.
JTAG_HRST	A hard reset issue is issued via JTAG. The debug connector's reset pin is not asserted. This option requires that the HRCW is set via JTAG (see **SYStem.Option.HRCWOVerRide**).
JTAG_PORST	A power-on reset is issued via JTAG. The debug connector's reset pin is not asserted. This option requires that the HRCW is set via JTAG (see **SYStem.Option.HRCWOVerRide**).
JTAG_SRST	A soft reset is issued via JTAG. The debug connector's reset pin is not asserted.

SYStem.Option.SLOWRESET Relaxed reset timing

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

This system option defines, how the debugger will test JTAG_HRESET. For some system mode changes, the debugger will assert JTAG_HRESET. PEr default (OFF), the debugger will release RESET and then read the HRESET signal until the HRESET pin is released. Reset circuits of some target boards prevent that the current level of HRESET can be determined via JTAG_HRESET. If this system option is enabled, the debugger will not read JTAG_HRESET, but instead waits four seconds and then assumes that the boards HRESET is released.

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

The alternative method circumvents a processor problem when a store type instruction is executed at the time a debug event occurs. This option is a workaround for the following errata:

- MPC7448 errata #24
- MPC8610 errata JTAG #2
- MPC8640/41 errata JTAG #4

Only enable this option, if one of the above processors is used and if the effect of this errata has been observed.

If the code to be debugged is located in RAM, **SYStem.Option.STEPSOFT** can be used without further configuration.

If the code to be debugged is located in read-only memory, the alternative method can be used if RAM is available and free for debugger use. In this case, declare the read-only memory using MAP.BOnchip, and the RAM used by the debugger using FLASH.TARGET.

NOTE: The alternative workaround can only fix issues caused by single steps. Manual breaks and on-chip breakpoints can still be affected by the problem.

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

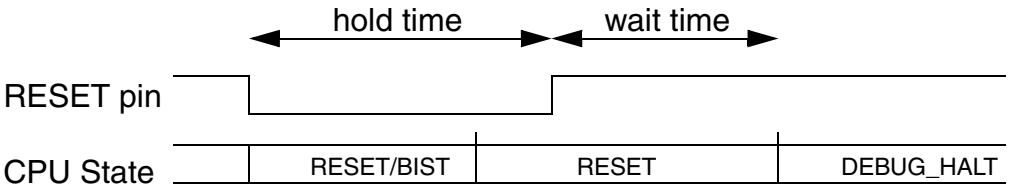
<time>: **1us**…**10s**

<reference>: **default**
RESET
RSTOUT

Set the time that the debugger will wait after releasing the reset pin, e.g. at **SYStem.Up**. If called without parameter, the default reset wait time is used (10us).

If the reference is set to **default**, the wait time starts when the debugger releases reset. If the reference is set to **RESET** or **RSTOUT**, the wait time starts when the debugger detects that reset is released on the corresponding pin.

Use this command when **SYStem.Up** fails, and the message **AREA** shows the message “Target reset detected during system.up sequence”. A wait time of several ms should be sufficient. If a wait time > 10ms is required, the target might require a stronger RESET pull-up resistor.



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

MPC8260, MPC8280, MPC83XX and compatible CPUs only.

ON While the CPU is stopped, the debugger will service the watchdog. When the application is running, the application is expected to service the watchdog.

OFF The debugger permanently disables the watchdog at SYStem.Up.

Software Watchdog Timer (SWT) — The SWT asserts a reset or non-maskable interrupt (as selected by the system protection control register) if the software fails to service the SWT for a designated period of time (e.g., because the software is trapped in a loop or lost). After a system reset, this function is enabled with a maximum time-out period and asserts a system reset if the time-out is reached. The SWT can be disabled or its time-out period can be changed in the SYPCR. Once the SYPCR is written, it cannot be written again until a system reset.

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.

<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.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* 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. • if *<range>* or *<address>* 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 *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	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. • 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**.

ITLB	Displays the contents of the Instruction Translation Lookaside Buffer.
DTLB	Displays the contents of the Data Translation Lookaside Buffer.
BAT	Displays the contents of the BAT table.
PTE	Displays the contents of the PTE table.

MMU.List — Compact display of MMU translation table

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.

<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.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: list the translation table of the specified process • else, this command lists the table the CPU currently uses for MMU 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 *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation 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 lists its address translation. • 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** [**Clear**] *<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. • 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 *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Loads the MMU address translation 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 copies its address translation into the debugger-internal static translation table. • 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 [**Clear**]	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 **OS Awareness Manual**. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.

CPU specific Tables in MMU.SCAN <table>

ITLB	Loads the instruction translation table from the CPU to the debugger-internal translation table.
DTLB	Loads the data translation table from the CPU to the debugger-internal translation table.
BAT	Loads the Block Address Translation table from the CPU to the debugger-internal translation table.
PTE	Loads the PTE table from the CPU to the debugger-internal translation table.

Format:	**MMU.Set** <*table*> <*index*> <*way*> <*tlbhi*> <*tlblo*> [<*tlbext*>]
<*table*>:	**ITLB** **DTLB**

Loads the specified MMU translation table from the CPU to the debugger-internal MMU table. Writing ITLB and DTLB is not supported for all processors.

<*index*>	Index (entry/set number) in TLB table
<*tlbhi*>, <*tlblo*>, <*tlbext*>	Data of the TLB entry.
<*way*>	Way number within the set
DTLB	Translation lookaside buffer for data load and store accesses
ITLB	Translation lookaside buffer for instruction fetches

CPU specific BenchMarkCounter Commands

The BenchMarkCounter features are based on the core's performance monitor, accessed through the performance monitor registers (PMR). Only processors with **e300c3 and e300c4 cores** have performance monitor registers:

- **MPC830X**
- **MPC831X**
- **MPC512X**

PMR access is only possible while the core is halted. For other processors, the BenchMarkCounter features are not available.

NOTE:
- 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.
- Events can be assigned to the **BMC** commands **BMC.<counter>.EVENT <event>** and **BMC.<counter>.RATIO X/<counter n>**. For descriptions of available events, see Freescale's e300 core reference manual (Table 11-9, Performance Monitor Event Selection).

BMC.<counter>.FREEZE — Freeze counter in certain core states

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

<state>: **USER** | **SUPERVISOR** | **MASKSET** | **MASKCLEAR**

Halts the selected performance counter if one or more of the enabled states (i.e. states set to ON) match the current state of the core. If contradicting states are enabled (e.g. SUPERVISOR and USER), the counter will be permanently frozen. The table below explains the meaning of the individual states.

<state>	Dependency in core
USER	Counter frozen if MSR[PR]==1
SUPERVISOR	Counter frozen if MSR[PR]==0
MASKSET	Counter frozen if MSR[PMM]==1
MASKCLEAR	Counter frozen if MSR[PMM]==0

Format: **BMC.FREEZE** [**ON** | **OFF**]

Enable this setting to prevent that actions of the debugger have influence on the performance counter. As this feature software controlled (no on-chip feature), some events (especially clock cycle measurements) may be counted inaccurate even if this setting is set ON.

The features supported by the TrOnchip command for TRACE32-ICD vary for the different PowerPC families.

TrOnchip.DISable

Disable debug register control

Format: **TrOnchip.DISable**

Not supported. The debugger always controls the debug registers.

TrOnchip.ENable

Enable debug register control

Format: **TrOnchip.ENable**

Not supported. The debugger always controls the debug registers.

TrOnchip.CONVert

Adjust range breakpoint in on-chip resource

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

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

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

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

TrOnchip.VarCONVert Adjust complex breakpoint in on-chip resource

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

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

TrOnchip.RESet Reset on-chip trigger settings

Format: **TrOnchip.RESet**

Resets the trigger system to the default state.

TrOnchip.state Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.TEnable Set filter for the trace

Format: **TrOnchip.TEnable** <*par*> (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TOFF — Switch the sampling to the trace to OFF

Format: **TrOnchip.TOFF** (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TON — Switch the sampling to the trace to “ON”

Format: **TrOnchip.TON EXT** | **Break** (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TTrigger — Set a trigger for the trace

Format: **TrOnchip.TTrigger** <*par*> (deprecated)

Refer to the **Break.Set** command to set a trigger for the trace.

JTAG/COP Connector PPC603e/700/MPC8200

Signal	Pin	Pin	Signal
TDO	1	2	(QACK-)
TDI	3	4	TRST-
(QREQ-)	5	6	JTAG-VREF
TCK	7	8	(PRESENT-)
TMS	9	10	N/C
(SRESET-)	11	12	GND
HRESET-	13	-	N/C (KEY PIN)
(CKSTOP-)	15	16	GND

NOTE:

- This is a standard 16 pin double row (two rows of eight pins) connector (pin-to-pin spacing: 0.100 in).
- Pin 6 (connected to VCCS) should have a resistance less than 5kOhm for 3.0~5.0V, less than 2kOhm for 1.8~3.0V.
- Pin 8 is permanently driven high (level of VCCS) by the debug cable.
- Signal in brackets are not needed by the debugger and can be left unconnected.
- If CPUs have an QACK input and this input is unused, QACK should be connected to GND. If the processor does not have QACK/QREQ pins, leave pin 2 and 15 N/C