LAUTERBACH
DEVELOPMENT TOOLS

PPC400/PPC440 Debugger and Trace

MANUAL

PPC400/PPC440 Debugger and Trace

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

ICD In-Circuit Debugger ..... 🗁

Processor Architecture Manuals ..... 🗁

PPC400/PPC440 ..... 🗁

PPC400/PPC440 Debugger and Trace 1

History 5

Introduction 5

Brief Overview of Documents for New Users 6

Demo and Start-up Scripts 6

Warning 7

Target Design Requirement/Recommendations 8

General 8

Quick Start JTAG 9

Troubleshooting 11

SYStem.Up Errors 11

FAQ 11

Configuration 12

System Overview 12

ICD Trace Extension for PPC400 (ICT) 13

General Fact for PPC403 RiscTrace Use 13

Debugging and Trace Mode 13

What does the PPC403 Trace Mode provide? 13

Used Options for RiscTrace 14

CPU specific Implementations 15

General Restrictions 15

Breakpoints 15

Software Breakpoints 15

On-chip Breakpoints 16

Breakpoint Restrictions 16

Breakpoint in ROM 16

Example for Breakpoints 17

Memory Classes 18
Memory Coherency 18

CPU specific SYStem Commands 19
SYStem.BdmClock Set JTAG clock frequency 19
SYStem.CPU Select the used CPU 19
SYStem.LOCK Lock and tristate the debug port 19
SYStem.MemAccess Select run-time memory access method 20
SYStem.Mode Select operation mode 20
SYStem.CONFIG.state Display target configuration 21
SYStem.CONFIG Configure debugger according to target topology 22
Daisy-Chain Example 24
TapStates 25
SYStem.CONFIG.CORE Assign core to TRACE32 instance 26

CPU specific SYStem Commands 27
SYStem.Option.CLOCKX2 Selects the clock for the real-time trace 27
SYStem.Option.DCFREEZE Freeze contents of cache while debugging 27
SYStem.Option.DCREAD Read from data cache 28
SYStem.Option.DMALOW Switch DMA to low priority 28
SYStem.Option.DataTrace Enable data trace via branch table method 28
SYStem.Option.FREEZERUN Stop timer in user mode 28
SYStem.Option.FREEZE Stop timer in debug mode 29
SYStem.Option.FlowTrace Prepare CPU for real-time trace 29
SYStem.Option.FOLDING Execute more instructions per cycle 29
SYStem.Option.HOOK Compare PC to hook address 29
SYStem.Option.ICFLUSH Invalidate instruction cache 31
SYStem.Option.ICREAD Read from instruction cache 31
SYStem.Option.IMASKASM Disable interrupts while single stepping 31
SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping 31
SYStem.Option.ISOCM Configure first address of ISOCM 32
SYStem.Option.MMUSPACES Separate address spaces by space IDs 32
SYStem.Option.NoDebugStop Disable JTAG stop on debug events 33
SYStem.Option.NoJtagHalt Disable HALT line 33
SYStem.Option.NOTRAP Use alternative instruction to enter debug mode 34
SYStem.Option.OVERLAY Enable overlay support 34
SYStem.Option.ResetMode Selects the reset mode 35
SYStem.Option.SLOWRESET Activate SLOWRESET 35
SYStem.Option.STEPSOFT Use alternative method for ASM single step 35
SYStem.Option.TURBO Skip additional checks/waits 36

CPU specific TrOnchip Commands 37
TrOnchip.state Setup window 37
TrOnchip.CONVert Adjust range breakpoint in on-chip resource 38
TrOnchip.DISable Disable NEXUS trace register control 38

TrOnchip.ENable Use CPU internal trigger logic 38
TrOnchip.RESet Set on-chip trigger to default state 39
TrOnchip.Set Trigger sources 39
TrOnchip.TEnable Set filter for the trace 39
TrOnchip.TOFF Switch the sampling to the trace to OFF 39
TrOnchip.TON Switch the sampling to the trace to 'ON' 40
TrOnchip.TTrigger Set a trigger for the trace 40
TrOnchip.VarCONVert Adjust complex breakpoint in on-chip resource 40
TrOnchip.SYNCHRONOUS Switches mode for data breakpoints 40

CPU specific MMU Commands 42
MMU.DUMP Page wise display of MMU translation table 42
MMU.List Compact display of MMU translation table 44
MMU.SCAN Load MMU table from CPU 46
MMU.FORMAT Define MMU table structure 48
MMU.Set.TLB Create a TLB entry on the TARGET 48
MMU.TLBINIT Reset TLB 51
MMU.TLBRESET Reset TLB 51

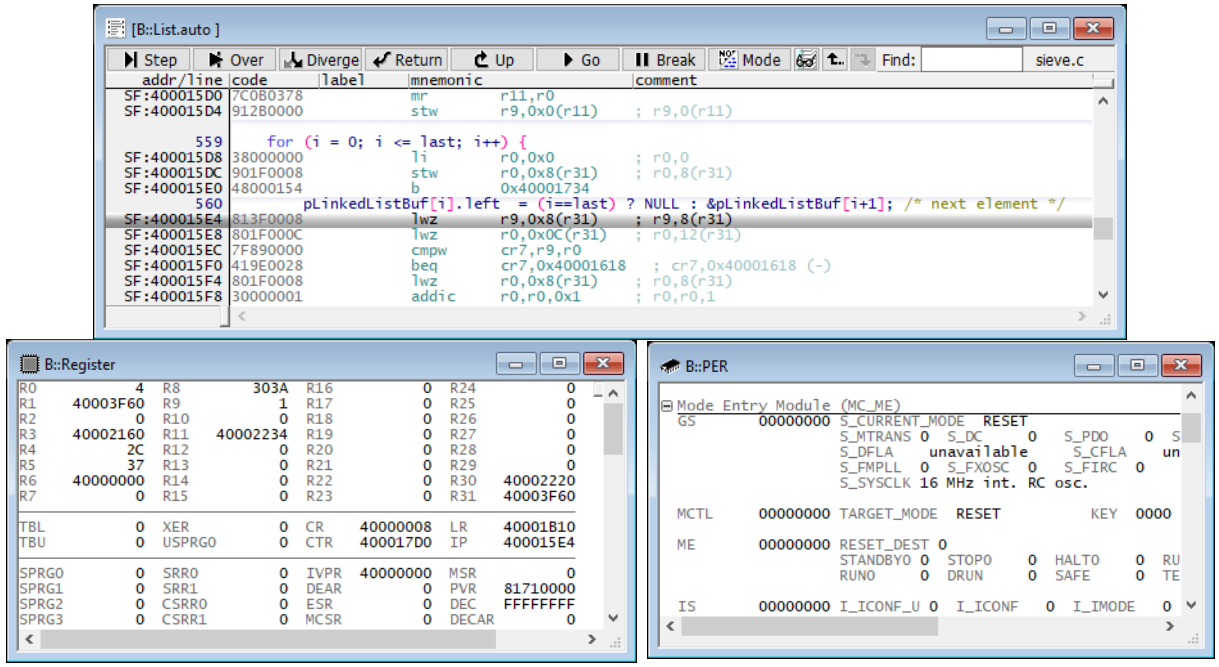
Debug Connector 52
Mechanical Description 52
JTAG Connector PPC401/403/405 and IOP480 52
Mictor Connector PPC440 52

Trace Connectors 53
Mictor Connector 38 pin (Version B) for PPC440 53
Mictor Connector 38 pin (Version B) for PPC405 54
Connector 20 pin (Version A) for PPC405 (obsolete) 54
Mictor Connector 38 pin (Version B) for PPC403 55
Connector 20 pin (Version A) for PPC403 55

Version 06-Jun-2024

History

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



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 PowerPC400/PowerPC440 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

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.

Target Design Requirement/Recommendations

General

Locate the **JTAG connector** as close as possible to the processor to minimize the capacitive influence of the trace length and cross coupling of noise onto the BDM signals.

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 starting the TRACE32 software.

```
B:
```

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

```
SYStem.CPU PPC403gcx
```

3. Map the EPROM simulator (optional).

```
MAP.ROM 0x0--0x1FFFF
```

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

```
MAP.BOnchip 0x100000++0x0fffff
```

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

5. Enter debug mode

```
SYStem.Up
```

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

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

```
Data.Set
```

7. Load the program.

```
Data.LOAD.Elf GNU603          ; ELF specifies the format, GNU603 is
                              ; the file name)
```

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"**.

The start-up can be automated using the programming language PRACTICE. 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>*.

```
B::                                ; Select the ICD device prompt

WinCLEAR                           ; Delete all windows

MAP.BOnchip 0x100000++0x0fffff     ; Specify where's FLASH/ROM

SYStem.CPU 403gcx                  ; Select the processor type

SYStem.Up                          ; Reset the target and enter debug
                                   ; mode

Data.LOAD.Elf GNU403               ; Load the application

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

List.Mix                           ; Open disassembly window         *)

Register.view /SpotLight           ; Open register window            *)

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

Var.Watch %Spotlight flags ast     ; Open watch window for variables *)

PER.view                           ; Open window with peripheral
                                   ; register                        *)

Break.Set sieve                    ; Set breakpoint to function sieve

Break.Set 0x1000 /Program          ; Set software breakpoint to address
                                   ; 1000 (address 1000 is in RAM)

Break.Set 0x101000 /Program        ; Set on-chip breakpoint to address
                                   ; 101000 (address 101000 is in ROM)
                                   ; For the PPC603e refer to the
                                   ; restrictions in On-chip Breakpoints.
```

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

SYStem.Up Errors

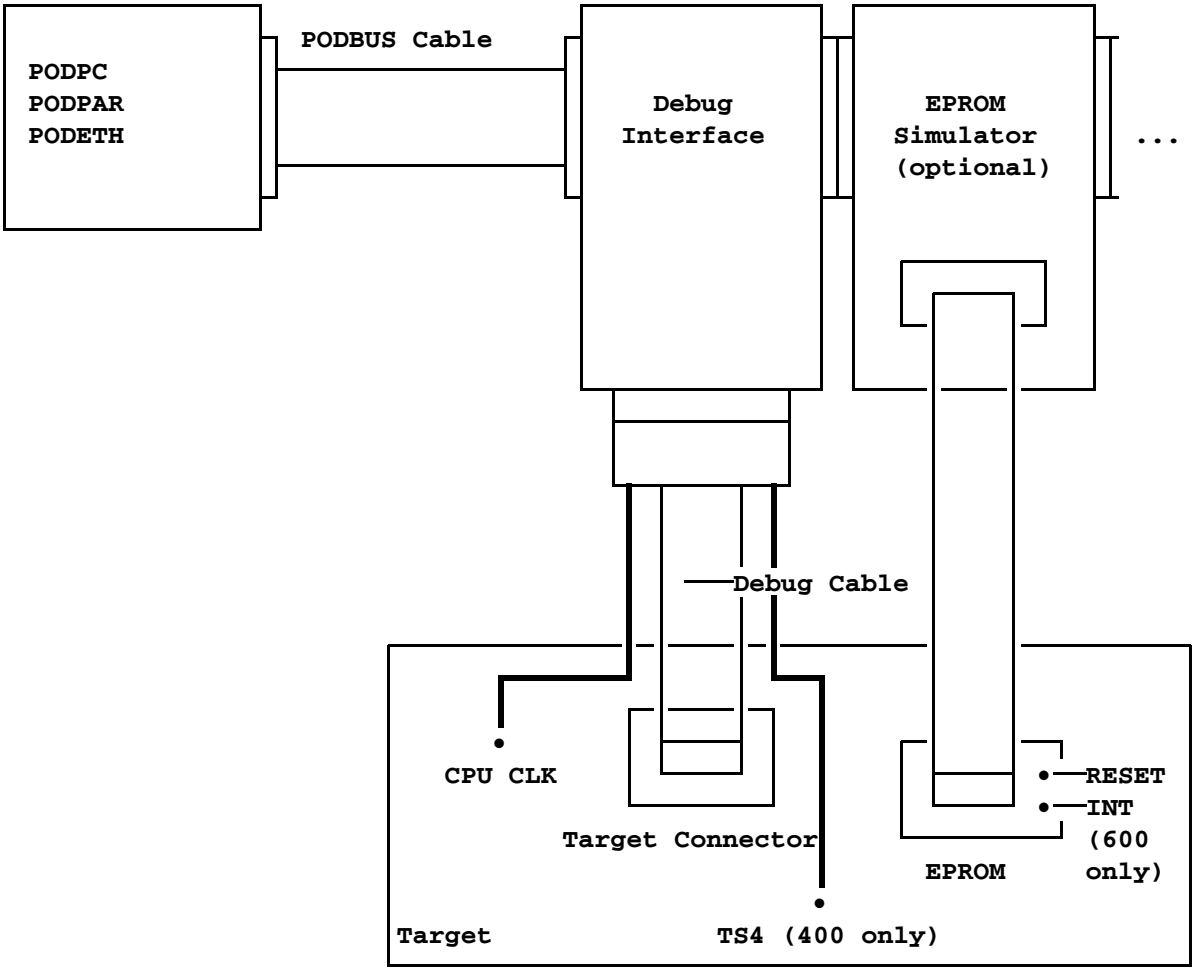
The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

All	**The target has no power.**
All	**The pull-up resistor between the JTAG/COP[VCCS] pin and the target VCC is too large.**
All	**The target is in reset:** **The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up.**
All	**There is logic added to the JTAG/COP state machine:** **The debugger supports only one processor on one JTAG chain. Only the debugged processor has to be between TDI and TDO in the scan chain. No further devices or processors are allowed.**
All	**There are additional loads or capacities on the JTAG lines.**

FAQ

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

System Overview



Basic configuration for the BDM Interface

NOTE:	Together with the debug interface you get a small black wire to connect the CPU clock to the plug on the debug module. This way you can use the divided CPU clock as clock for the debug interface.

NOTE:	If you use the PPC400 family you get a second wire to connect the TS4 signal. This is only necessary if you want to use the **TrBus.Out** command.

General Fact for PPC403 RiscTrace Use

The PPC403 supports BDM debug features and FlowTrace features. Both uses the 403 debug logic on the chip. During normal BDM debugging any debug event will stop the processor. When using the Trace mode any debug event will start tracing. Therefore there are some restrictions to use both at the same time.

Debugging and Trace Mode

In the FlowTrace mode the SW-breakpoints normally cannot be used. This means also HLL steps, step over or functions like “go *<addr/label>*” are not working. The user is responsible that all breakpoints are cleared (see **Break.List**). If the trap exception handler can be used and modified be the Trace Extension, then SW-Breakpoint will be available and also Debugging and Trace Mode at the same time will be possible with some restrictions (See Trace Extension for the PPC403).

What does the PPC403 Trace Mode provide?

The Trace Extension supports all features that are offered from the PPC403 Real-Time Trace functionality. With the Trace Extension you are able to trace from the current instruction on, till the analyzer stack is full (Stack Mode) or the break button is pushed (Fifo Mode). The trace feature allows to follow the source code. There are five cases where the Trace Extension needs additional information from the trace signals to follow the source code:

1. Exceptions
2. Branch to Link Instruction
3. Branch to Count Instruction
4. Return from Interrupt Instruction
5. Return from Critical Interrupt Instruction

To allow tracing the processor will broadcast the following:

- Count Register contents after Move to Count has occurred
- Instruction address after an exception has occurred
- Instruction address after a return from interrupt or return from critical interrupt(RFI,RFCI)
- Link Register contents after Move to Link has occurred

If a Trace Start events has occurred (depend on DBCR Register configuration) then the first RiscTrace program flow synchronization take place after one of the five cases explained before are executed. This means that the program should be traced, must consist one of the five special cases. Source code parts which do not consist of one of the five cases, which start the FlowTrace broadcast, cannot be traced in the fifo mode.

Used Options for RiscTrace

- **SYStem.OPTION.FlowTrace** ON /OFF
- **SYStem.OPTION.CLOCKX2** ON /OFF
- **SYStem.OPTION.HOOK** *<address>*
- IOCR Register [RMD,2CX] (Peripheral Window)
- DBCR Register (Peripheral Window)

General Restrictions

PPC400	Make sure, that you don't increase the debug clock without decreasing the internal waitstates, when the TURBO option is enabled. If external waitstates are used it is recommended to switch TURBO mode off. The BDM driver may not work with older PPC403GA-JA25 samples.

Breakpoints

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

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 increasing the number of software breakpoints will reduce the debug speed.

On-chip Breakpoints

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

- **CPU family**
- **On-chip breakpoints**: Total amount of available on-chip breakpoints.
- **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.
- **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	On-chip Breakpoints	Instruction Breakpoints	Read/Write Breakpoints	Data Breakpoints
PPC401/403	2 Instruction 2 Read/Write	2	2	—
PPC405	4 Instruction 2 Read/Write	4	2	2

Breakpoint Restrictions

You can check your currently set breakpoints with the command **Break.List**

Breakpoint in ROM

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.

Example for Breakpoints

Assume you have a target with FLASH from `0` to `0xFFFFF` and RAM from `0x100000` to `0x11FFFF`. The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x0--0x0FFFFF
```

The following breakpoint combinations are possible.

1. Software breakpoints:

```
Break.Set 0x100000 /Program     ; Software Breakpoint 1

Break.Set 0x101000 /Program     ; Software Breakpoint 2

Break.Set 0xx /Program          ; Software Breakpoint 3
```

2. On-chip breakpoints:

```
Break.Set 0x100 /Program        ; On-chip Breakpoint 1

Break.Set 0x0ff00 /Program      ; On-chip Breakpoint 2
```

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
D	Data
SPR	Special Purpose Register
DCR	Device Control Register (PPC40x only)
IC	Instruction Cache
DC	Data Cache
NC	No Cache (only physically memory)

If caching is disabled via the appropriate hardware registers (DCCR/ICCR for PPC400 series, HID0 for PPC603 series), memory accesses to the memory classes IC or DC are realized by TRACE32-ICD as reads and writes to physical memory.

Memory Coherency

Memory coherency on access to the following memory classes. If data will be set to DC, IC, NC, D or P the D-Cache, I-Cache or physical memory will be updated.

	D-Cache	I-Cache	Physical Memory
DC:	Yes	No	Yes
IC:	No	Yes	Yes
NC:	No	No	Yes
D:	Yes	Yes	Yes
P:	Yes	Yes	Yes

See also **SYStem.Option**

SYStem.BdmClock — Set JTAG clock frequency

Format:	**SYStem.BdmClock** <*rate*>
<*rate*>:	**EXT/2** | **EXT/4** | <*fixed*>
<*fixed*>:	**1000.** … **5000000.** | **10 000 000.** (Default 1 MHz)

Selects the frequency for the debug interface. A fixed frequency or a divided external clock can be used.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** <*cpu*>
<*cpu*>:	**403GA** | **403GB** | **403GC** | **403GCX** | **405CR** | **405GP** | **440GP**

SYStem.LOCK — Lock and tristate the debug port

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.

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

Denied 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.

SYStem.Mode Select operation mode

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

Select target reset mode.

Down Disables the Debugger. The state of the CPU remains unchanged.

NoDebug Resets the target with debug mode disabled (for the PPC400 family the same as Go). In this mode no debugging is possible. The CPU state keeps in the state of NoDebug.

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.

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.
Attach	This command works similar to Up command. The difference is that the target CPU is not reset. The BDM/JTAG/COP interface will be synchronized and the CPU state will be read out. After this command the CPU is in the SYStem.Up mode and can be stopped for debugging.
StandBy	Not available PPC400/PPC440.

SYStem.CONFIG.state — Display target configuration

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	Informs the debugger about the debug connector type and the communication protocol it shall use.
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>: (JTAG):	**DRPRE** *<bits>* **DRPOST** *<bits>* **IRPRE** *<bits>* **IRPOST** *<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**.

DRPRE (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.

DRPOST (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.

IRPRE (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.

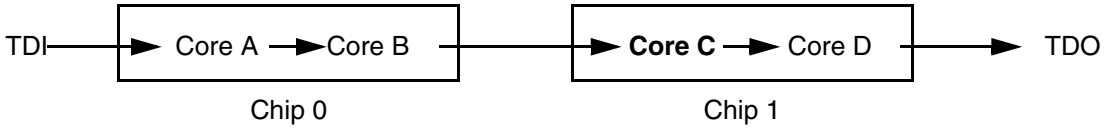
IRPOST (default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.

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

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**.

SYStem.Option.CLOCKX2 Selects the clock for the real-time trace

Available on: MPC403

Format: **SYStem.Option.CLOCKX2** [**ON** | **OFF**]

This option select the clock for the Real-Time Trace. (Required for the TRACE32-ICD Risc Trace Modul).If the 403GCX works with internal double clock (IOCR [2XC]), this option must be on before starting to record with the trace.

If the source code being traced change the IOCR[2XC] register by its own during the trace, the RiscTrace doesn't works properly.

SYStem.Option.DCFREEZE Freeze contents of cache while debugging

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

If this feature is enabled the status of the data caches is preserved while debugging. This feature should be used in combination with **SYStem.Option.DCREAD** in order to read data as seen by the core. Otherwise all memory accesses are as for access class NC.
If disabled, the debugger might modify the caches contents with each data access e.g. a Data.dump window.

For caches that use hardware coherency (e.g. MESI protocol), the DCFREEZE feature is not supported. This respects multicore architectures that use non-shared caches.

SYStem.Option.DCREAD Read from data cache

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

Data.dump windows for memory class D: displays the memory value from the d-cache if valid. If d-cache is not valid the physical memory will be read.

If caching is disabled via the appropriate hardware registers (DCCR/ICCR for PPC400 Series) or cache is invalid, read and writes from/to memory will directly reflect to contents of physical memory even if a cache memory class is selected.

SYStem.Option.DMALOW Switch DMA to low priority

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

All DMA transfers continue in debug mode. If DMALOW is enabled all DMA activities are switched to low priority.

SYStem.Option.DataTrace Enable data trace via branch table method

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

Enables data trace support via branch table method. See examples in `demo\powerpc\etc\trace4xx`

SYStem.Option.FREEZERUN Stop timer in user mode

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

Controls the internal CPU timer. If FREEZERUN is enabled, the timer will be stopped whenever the CPU enters the user mode.

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

Controls the internal CPU timer. If FREEZE is enabled, the timer will be stopped whenever the CPU enters the debug mode.

SYStem.Option.FlowTrace

Prepare CPU for real-time trace

Available on: MPC403

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

Prepare the CPU for real-time trace. (Required for the TRACE32-ICD RISC Trace Module). If switched on, on every step or go the DBCR[EDM,IDM] bits are switched off and the IOCR[RDM] bits are switched to Trace Mode automatically.

SYStem.Option.FOLDING

Execute more instructions per cycle

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

The PPC400 CPUs can execute more than one instruction per cycle. If FOLDING is disabled, exactly one instruction is executed per cycle.

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.

This option make it possible to use breakpoints in the Real-Time Trace Mode. (Required for the TRACE32-ICD RISC Trace Module) This assume that the trap exception handler can be modified for the RiscTrace. After any synchronize break (using breakpoints) the IP will be compared with the Hook value. If true than the last exception of will be canceled.

For example. Do use breakpoints, the trap exception handler must be prepared with some instructions.

P:FFF00700 lis r3,0

If the CPU runs the instruction P:FFF0070C mtxxx, it stops. The IP will be compared to the Hook value. If the Hook value is also 0xFFF0070C the exception will be canceled and the CPU register reconstructed to the last breakpoint.

If you start again after a break with a breakpoint the IP is on the breakpoint. This means that in the HLL mode a step are executed before the processor put into run mode.

SYStem.Option.ICFLUSH Invalidate instruction cache

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

Invalidates the instruction cache and flush the data cache before starting the target program (Step or Go). This is required when the ICACHEs are enabled and software breakpoints are set to a cached location.

SYStem.Option.ICREAD Read from instruction cache

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

Data.List window and Data.dump window for memory class P: displays the memory value from the I-cache if valid. If I-cache is not valid the physical memory will be read.

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.

Format: **SYStem.Option.ISOCM** *<base_address>*

On **Virtex4FX, Virtex5FX** the ISOCM memory can be read using a special access mechanism. Configure the first address of the ISOCM using the command **SYStem.Option.ISOCM**. The default value is 0xFFFF.FFFF, indicating that the system under debug does not have ISOCM memory.

For a design with ISOCM memory from 0xFFFF.8000--0xFFFF.FFFF you should therefore use:

```
SYStem.Option.ISOCM 0xFFFF8000
```

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.

SYStem.Option.NoJtagHalt Disable HALT line

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

Default: OFF.

The JTAG connection for an PowerPC 4xx type CPU features an HALT- signal which will stop the CPU. The HALT- line enables to stop an core independently from the BDM/JTAG clock. As the HALT- line is a shared signal for all cores & chips in a JTAG chain all cores & chips stop if the HALT- line is asserted.

By disabling the HALT- line it is possible to debug only specific cores in the chain without interference with other PowerPC 4xx cores/chips (AMP). A side effect of this option is that the SYStem.Up behavior will change as the core will then not be prevented from executing code after an reset. Thus if this option is enabled the CPU will have executed some code after **SYStem.Up** and the system might be no longer in reset status.

SYStem.Option.NOTRAP Use alternative instruction to enter debug mode

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

If the user software uses the TRAP command, the CPU performs a BREAK in debug mode and does not jump to the interrupt handler of the TRAP command.

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 *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.ResetMode — Selects the reset mode

Format: **SYStem.Option.ResetMode** [**SYSTEM** | **CHIP** | **CORE**]

Use this option to select the type of reset at SYStem.Up. There are three types of resets:

- SYSTEM will reset the peripherals and the core.
- CHIP
- CORE will only reset the core.

Note that a reset of the core does not reset the register.

SYStem.Option.SLOWRESET — Activate SLOWRESET

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

After the debugger resets the CPU (e.g. via SYStem.Up), the debugger senses $\overline{\texttt{HRESET}}$ for 2 … 3 s before an error message is displayed.

SYStem.Option.STEPSOFT — Use alternative method for ASM single step

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

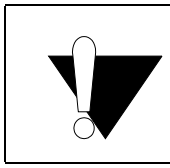
This method uses software breakpoints to perform an assembler single step instead of the processor's built-in single step feature. Works only for software in RAM. Do not turn ON unless advised by Lauterbach.

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

If there are buffers, additional loads or high capacities on the JTAG/COP lines, reduce the debug speed.

Enables Turbo debugging. If Turbo is disabled, the CPU checks after each memory access in debug mode if the CPU is ready. This check will decrease debug speed (30-40%).

If Turbo is enabled, the CPU will make no checks. The internal waitstates for a memory access must be decremented before increasing the debug frequency. With the default debug frequency of 1 MHz Turbo can always be enabled.



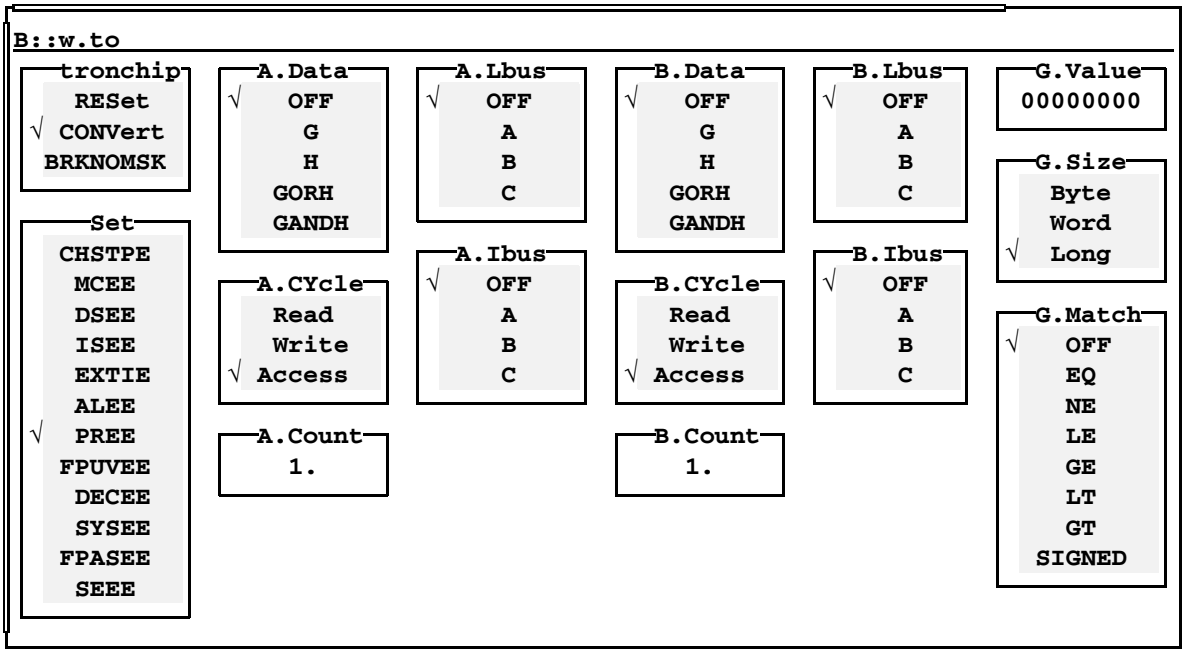
The features supported by the TrOnchip command for TRACE32-ICD vary for the different PowerPC families.

TrOnchip.state

Setup window

Format: **TrOnchip.state**

Control panel to configure the on-chip breakpoint registers (here **MPC860**).



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

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.DISable Disable NEXUS trace register control

Format: **TrOnchip.DISable**

Disables NEXUS register control by the debugger. By executing this command, the debugger will not write or modify any registers of the NEXUS block. This option can be used to manually set up the NEXUS trace registers. The NEXUS memory access is not affected by this command. To re-enable NEXUS register control, use command **TrOnchip.ENable**. Per default, NEXUS register control is enabled.

TrOnchip.ENable Use CPU internal trigger logic

Format: **TrOnchip.ENable** <*item*> [**ON** | **OFF**]

If TrOnchip.Enable is ON (by default) the CPU internal trigger/trace/debug feature like IACx (Instruction Address Compare Register) and DACx (Data Address Compare Register) will be used by the debugger. If TrOnchip.Enable is OFF, the registers can be manually programmed by user or application.

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

Format: **TrOnchip.RESet**

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

TrOnchip.Set — Trigger sources

Format: **TrOnchip.Set** <*item*> [**ON** | **OFF**]

<*item*>: **BRANCH**
eXception

Enables various trigger events. Detailed description of the trigger events can be found in the processor manuals.

eXception Debug mode is entered if an exception occurs.

BRANCH Debug mode is entered if a branch is taken.

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.

TrOnchip.VarCONVert — Adjust complex breakpoint in on-chip resource

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

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 on** the breakpoint will automatically be converted into a single address breakpoint. This is the default setting. Otherwise an error message is generated.

TrOnchip.SYNCHRONOUS — Switches mode for data breakpoints

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

Default: OFF.

Switches the mode of the DAC (Data Address Compare Register) for debug events on PPC44x/PPC46x cores. This mode setting is only effective if read/write breakpoints are used.

If the DAC works in synchronous mode the processor enters the stop state when reaching load/store instructions and ceases the processing of instructions. This means the CPU will stop on a load/store instruction without executing the read/write cycle. The disadvantage is that the core performance for load/store instructions will be reduced in synchronous mode. Switch the synchronous mode OFF in order to maintain normal processor performance .

In asynchronous DAC mode the processor enters stop state on load/store instructions either before or after the completion of the instruction. This means the CPU will execute the read/write cycle and stop some instructions later for the most cases.

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**.

TLB	Displays the contents of the Translation Lookaside Buffer.

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**.

CPU specific Tables in MMU.List <table>

TLB	Displays the contents of the Translation Lookaside Buffer.

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>

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

Format:	**MMU.FORMAT** <*format*> [<*effective_range*> <*real_base*>]
<*format*>:	**LINUX** **LINUX26** **LINUXEXT** **LINUXE5** **LYNXOS** **LYNXOSPHYS** **QNX** **QNXBIG** **DEOS** **DEOS64**

Defines the structure of the MMU table and optionally the base for the kernel space table.

LINUX	Standard Page-Table format for PPC405 running Linux
LINUXEXT	Standard Page-Table format for PPC44x/46x running Linux

If you require support for a particular operating system, please contact **support@lauterbach.com**.

See also **MMU.FORMAT** in general_ref_m.pdf.

MMU.Set.TLB Create a TLB entry on the TARGET

Format 1: PPC 40x	**MMU.Set.TLB** <*index*> <*hi*> <*lo*> **MMU.TLBSET** (deprecated)
Format 2: PPC 44x/46x	**MMU.Set.TLB** <*index*> <*ws0*> <*ws1*> <*ws2*> **MMU.TLBSET** (deprecated)
Format 3: PPC 47x	**MMU.Set.TLB** <*index*> <*way*> <*ws0*> <*ws1*> <*ws2*> [**/Bolted** <*index*>] **MMU.TLBSET** (deprecated)

Creates/modifies an TLB entry addressed by index (optional: way) in the target CPU. The provided settings match the format of the **tlbwe** instruction of the target CPU and is thus CPU/Architecture specific. For the exact meaning of the Bits provided below please refer to the CPUs/Architecture User-Guide.

Common arguments:

<index>	The line/index of the corresponding TLB entry
<way>	The way of the corresponding TLB entry

PowerPC 405 specific arguments:

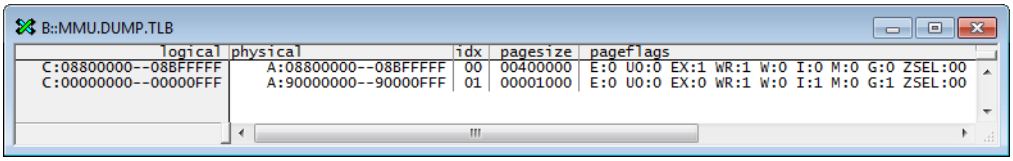
<lo>	EPN	SIZ	V	E	U0	TID
Bit	[0:21]	[22:24]	[25]	[26]	[27]	[28:35]

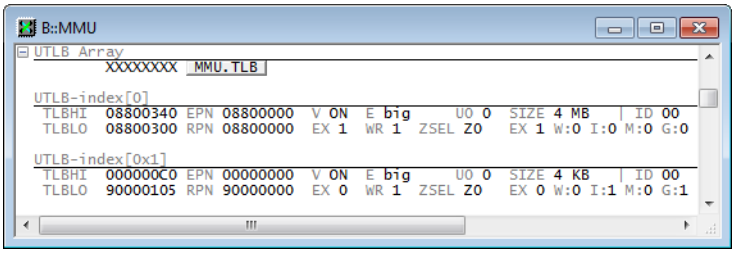
<hi>	RPN	SXW	ZSEL	WIMG
Bit	[0:21]	[22:23]	[24:27]	[28:31]

Examples:

```
; Create TLB Entry, A:0x08800000++0x3FFFFF <-> SD:0x08800000++0x3FFFFF
; TID: 0x0, Permissions: eXecute, Write
MMU.TLBSET 0. 0x0880034000 0x08800300

; Create TLB Entry, A:0x90000000++0xFFF <-> SD:0x0++0xFFF
; TID: 0x0, Permissions: Write,          Cache: Inhibit, Guarded
MMU.TLBSET 1. 0x000000C000 0x90000105
```





PowerPC 44x/46x specific arguments:

<ws0>	EPN	V	TS	SIZE	-	TID
Bit	[0:21]	[22]	[23]	[24:27]	[28:31]	[32:39]

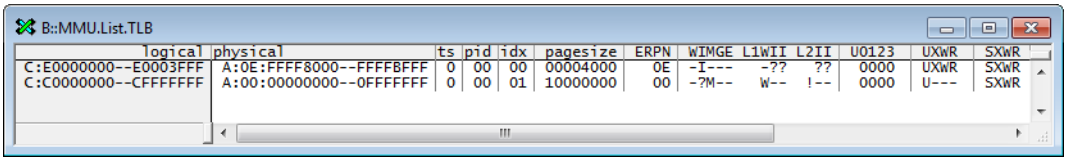
<ws1>	RPN	-	ERPN
Bit	[0:21]	[20:21]	[22:31]

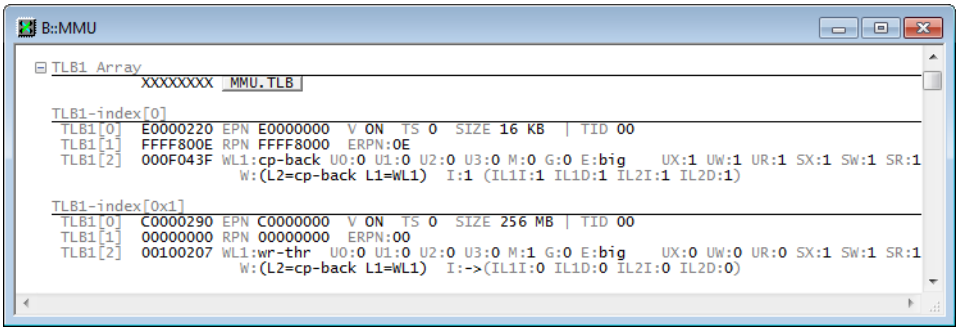
<ws2>	-	FAR,WL1	IL1ID2ID	U	WIMGE	UXWRSXWR
Bit	[0:9]	[10:11]	[12:15]	[16:19]	[20:24]	[26:31]

Examples:

```
; Create TLB Entry, A:0xE:0x0xFFFF8000++0x3FFF <-> SD:0xE0000000++0x3FFF
; TID: 0x0, Permissions: User XWR, Supervisor XWR, Cache: Inhibit (L1&L2)
MMU.TLBSET 0. 0xE000022000 0xFFFF800E 0x000F043F

; Create TLB Entry, A:0x0:0x0++0x0FFFFFFF <-> SD:0xC0000000++0x0FFFFFFF
; TID: 0x0, Permissions: User ---, Supervisor XWR, Bolted: 0x1,
; Coherency: Enabled => WL1=1
MMU.TLBSET 1. 0xC000029000 0x00000000 0x00100207
```





PowerPC 47x specific arguments:

<ws0>	EPN	V	TS	DSIZ	-	TID
Bit	[0:19]	[20]	[21]	[22:27]	[28:31]	[32:39]

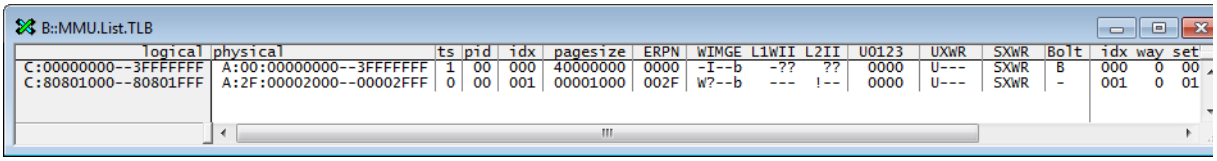
<ws1>	RPN	-	ERPN
Bit	[0:19]	[20:21]	[22:31]

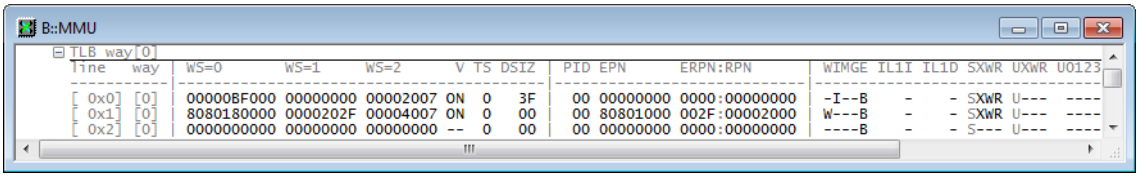
<ws2>	-	IL1ID	U	WIMGE	-	UXWRSXWR
Bit	[0:12]	[14:15]	[16:19]	[20:24]	[25]	[26:31]

Examples:

```
; Create TLB Entry, A:0x2F:0x2000++0xFFF <-> SD:0x80801000++0xFFF
; TID: 0x0, Permissions: User ---, Supervisor XWR, Cache: Inhibit
; EA=0x80801000, DSIZ=0x0 => Index=0x1
MMU.TLBSET 0x1 0x0 0x8080180000 0x0000202F 0x00004007

; Create TLB Entry, A:0x0:0x0++0x3FFFFFFF <-> SD:0x0++0x3FFFFFFF
; TID: 0x0, Permissions: User ---, Supervisor XWR, Bolted: 0x1,
; Coherency: Enabled
; EA=0x00000000, DSIZ=0x3F => Index=0xC0, Bolted => Way=0x0
MMU.TLBSET 0x0 0x0 0xC0000BF000 0x00000000 0x00002007 /Bolted 1.
```





MMU.TLBINIT

Reset TLB

Format: **MMU.TLBINIT**

Resets TLB. This command is an alias for **MMU.TLBRESET**.

MMU.TLBRESET

Reset TLB

Format: **MMU.TLBRESET**

Resets TLB. This command is an alias for **MMU.TLBINIT**.

Mechanical Description

JTAG Connector PPC401/403/405 and IOP480

It is recommended to connect all N/C Pins to GND (if you work with LAUTERBACH tools only).

Signal	Pin	Pin	Signal
TDO	1	2	N/C
TDI	3	4	TRST- (*)
N/C	5	6	VCCS
TCK	7	8	N/C
TMS	9	10	N/C
HALT-	11	12	N/C
N/C	13	-	KEY
N/C	15	16	GND

This is a standard 16 pin double row (two rows of eight pins) connector (pin-to-pin spacing: 0.100 in.).

Mictor Connector PPC440

The mictor connector can also be used for debugging. For a description of the pinout please refer to **“Trace Connectors”**.

Mictor Connector 38 pin (Version B) for PPC440

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
HALT-	7	8	N/C
N/C	9	10	N/C
TDO	11	12	VTREF
N/C	13	14	N/C
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	N/C
TRST-	21	22	N/C
N/C	23	24	ES4
BS0	25	26	TS0
BS1	27	28	TS1
BS2	29	30	TS2
ES0	31	32	TS3
ES1	33	34	TS4
ES2	35	36	TS5
ES3	37	38	TS6

Connect Pin 39,40,41,42 and 43 to GND.

Mictor Connector 38 pin (Version B) for PPC405

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
HALT	7	8	N/C
N/C	9	10	N/C
TDO	11	12	VTREF
N/C	13	14	N/C
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	N/C
!TRST	21	22	N/C
N/C	23	24	TS1O
GND	25	26	TS2O
GND	27	28	TS1E
GND	29	30	TS2E
GND	31	32	TS3
GND	33	34	TS4
GND	35	36	TS5
GND	37	38	TS6

Connect Pin 39,40,41,42 and 43 to GND.

Connector 20 pin (Version A) for PPC405 (obsolete)

Signal	Pin	Pin	Signal
N/C	1	2	N/C
CLK	3	4	N/C
N/C	5	6	N/C
N/C	7	8	N/C
N/C	9	10	N/C
N/C	11	12	TS1O
TS2O	13	14	TS1E
TS2E	15	16	TS3
TS4	17	18	TS5
TS6	19	20	GND

This is a standard 20 pin double row (two rows of eight pins) connector (pin-to-pin spacing: 0.100 in.).

Mictor Connector 38 pin (Version B) for PPC403

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
HALT	7	8	N/C
N/C	9	10	N/C
TDO	11	12	VTREF
N/C	13	14	N/C
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	N/C
N/C	21	22	N/C
N/C	23	24	GND
GND	25	26	TS0
GND	27	28	TS1
GND	29	30	TS2
GND	31	32	TS3
GND	33	34	TS4
GND	35	36	TS5
GND	37	38	TS6

Connect Pin 39,40,41,42 and 43 to GND.

Connector 20 pin (Version A) for PPC403

Signal	Pin	Pin	Signal
N/C	1	2	N/C
CLK	3	4	N/C
N/C	5	6	N/C
N/C	7	8	N/C
N/C	9	10	N/C
N/C	11	12	N/C
TS0	13	14	TS1
TS2	15	16	TS3
TS4	17	18	TS5
TS6	19	20	GND

This is a standard 20 pin double row (two rows of eight pins) connector (pin-to-pin spacing: 0.100 in.).