QorIQ Debugger and NEXUS Trace

Version 18-Nov-2020

History

20-Aug-18	Added descriptions for NEXUS.LaneMapping and NEXUS.USEPORT commands.
20-Aug-18	Updated Break.List screenshot in the chapter Breakpoints.
15-Aug-18	Updated the chapter Trace Information with NEXUS PCIe trace description.
15-Aug-18	Updated NEXUS.state screenshots and their descriptions in the chapter Trace Information.
15-Aug-18	Updated the chapter "Tool Configuration".
11-Jul-18	Added description for SYStem.Option TranslationSPACE.
17-Apr-18	Updated the descriptions of MMU.DUMP, MMU.List, and MMU.SCAN.
16-Apr-18	Added descriptions for SYStem.Option ZoneSPACES, SYStem.Option MACHINESPACES, and MMU.FORMAT.

Introduction

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

- QorIQ Series with e500mc cores (P204X, P30XX, P40XX)
- QorIQ Series with e5500 cores (P50XX, T10XX)
- QorIQ Series with e6500 cores (T2XXX, T4XXX, B4XXX)

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:

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

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:

  - Choose **Help** menu > **Processor Architecture Manual**.

- **"OS Awareness Manuals"** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Warning

Signal Level

P204X **P30XX** **P40XX** **P50XX** **T10XX** **T2XXX** **T4XXX** **B4XXX**	The debugger drives the output pins of the BDM/JTAG/COP connector with the same level as detected on the VCCS pin. If the IO pins of the processor are 3.3 V compatible then the VCCS should be connected to 3.3 V. See also **System.up Errors**.

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 **JTAG/COP and Aurora NEXUS connectors** 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. Do not put any termination (e.g. R/C/RC) 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 20 MHz, and you might want to use high frequencies for optimized download performance.
- For optimal operation, the debugger should be able to reset the target board completely (processor external peripherals, e.g. memory controllers) with the COP connector signal HRESET (respectively the CPU pin PORESET). For further details please see the QorIQ documents “Integrated Processor Hardware Specifications”, part “Hardware design considerations”.
- In order to start debugging right from reset, the debugger must be able to control the COP connector signals TRST and HRESET independent of each other.

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. If your CPU is not listed, you should request a software update that handles this CPU.

```
SYStem.CPU P4080
```

3. Specify that on-chip breakpoints should be used by the debugger if a program breakpoint is set to the boot page (read-only memory):

```
MAP.BOnchip 0xFFFFF000--0xFFFFFFFF
```

4. Enter active debug mode.

```
SYStem.Up
```

This command resets the CPU (HRESET), enters debug mode and stops all cores of the CPU at the reset vector. See also **SYStem.Up Errors** if problems occur.

5. After **SYStem.Up**, only the boot page is visible for the CPU. Specify Local Access Windows (LAWs) and initialize MMU TLBs to configure which memory is visible to the CPU at which address. In the example, we map the P4080 internal SRAM (CoreNet platform cache) to logical address 0x00000000. See **MMU.TLB.Set** and **Data.Set** for details.

```
Data.Set ANC:iobase.address()+0x0C00 %LONG %BE 0x00000000     ;Set
Data.Set ANC:iobase.address()+0x0C04 %LONG %BE 0x00000000     ;LAW 0
Data.Set ANC:iobase.address()+0x0C08 %LONG %BE 0x81000013     ;and
MMU.TLB1.Set 1. 0x80000500 0x00000002 0x00000015 0x00000000   ;TLB 1
```

6. Load the program.

```
Data.LOAD.ELf demo.elf              ;(ELF specifies the format,
                                    ;demo.elf 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"**.

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, there can be several reasons. The following chapters list possible errors and explain how to fix them.

Target Power Fail

The Target has no power, the debug cable is not connected or not connected properly. Check if the JTAG VCC pin is driven by the target. The voltage of the pin must be identical to the debug voltage of the JTAG signals. It is recommended to connect VCC directly to the pin, or via a resistor < 5 kOhm.

Emulation Pod Configuration Error

The debugger was not able to determine the connected processor. There are three possible reasons for this error. In all cases, please check the **AREA** window for more information:

- The connected processor is not supported by the used software. Please check if the processor is supported by the debugger. Processors that appeared later than the debugger software version are usually not supported. Please download and install the latest software from our website, or contact technical support to get a newer software. Please also check if the processor or the software update is covered by your current licence.
- A JTAG communication error prevented correct determination of the connected processor. Please check if the debugger is properly connected to the target.

Target Reset Fail

On **SYStem.Up**, the debugger will assert $\overline{\text{HRESET}}$ in order to stop the CPU at the reset address. A target reset fail means, that an unexpected reset behavior caused an error:

- The reset is asserted longer than 500ms and is not visible on the JTAG connector. Try **SYStem.Option.SLOWRESET**, and check signal level of the JTAG HRESET pin.
- The target reset is permanently asserted. Check target reset circuitry and reset pull-up.
- A chip external watchdog caused a reset after the debugger asserted reset. Disable the watchdog and try again.

Emulation Debug Port Fail

An emulation debug port fail can have a variety of reasons. Please check the **AREA** window for a detailed error message. Here is a collection of frequent issues:

- JTAG communication error. Please check the signals on the debug connector.
- Problems related with Reset can not always be detected as those. Please check **Target Reset Fail**.

CPU Setting Error

- The detected quantity of cores does not fit to the CPUs default. Most QorIQ CPUs offer the possibility to completely disable cores (typically via the dedicated TEST_SEL pin). If any of the cores are disabled you have to configure the debugger to restrict the access to the active cores using the **CORE.ASSIGN** command.

```
; e.g. P2041, TEST_SEL wired to high level.
; -> Just the first two cores are active.
SYStem.CPU P2041
SYStem.Up             ;Expected 4 active cores -> Error message

CORE.ASSIGN 1,2       ;Configure the active usable cores
SYStem.Up             ;Debugging possible
```

FAQ

Please refer to our Frequently Asked Questions page on the Lauterbach website.

Tool Configuration

QorIQ development boards typically offer one of the following connector options:

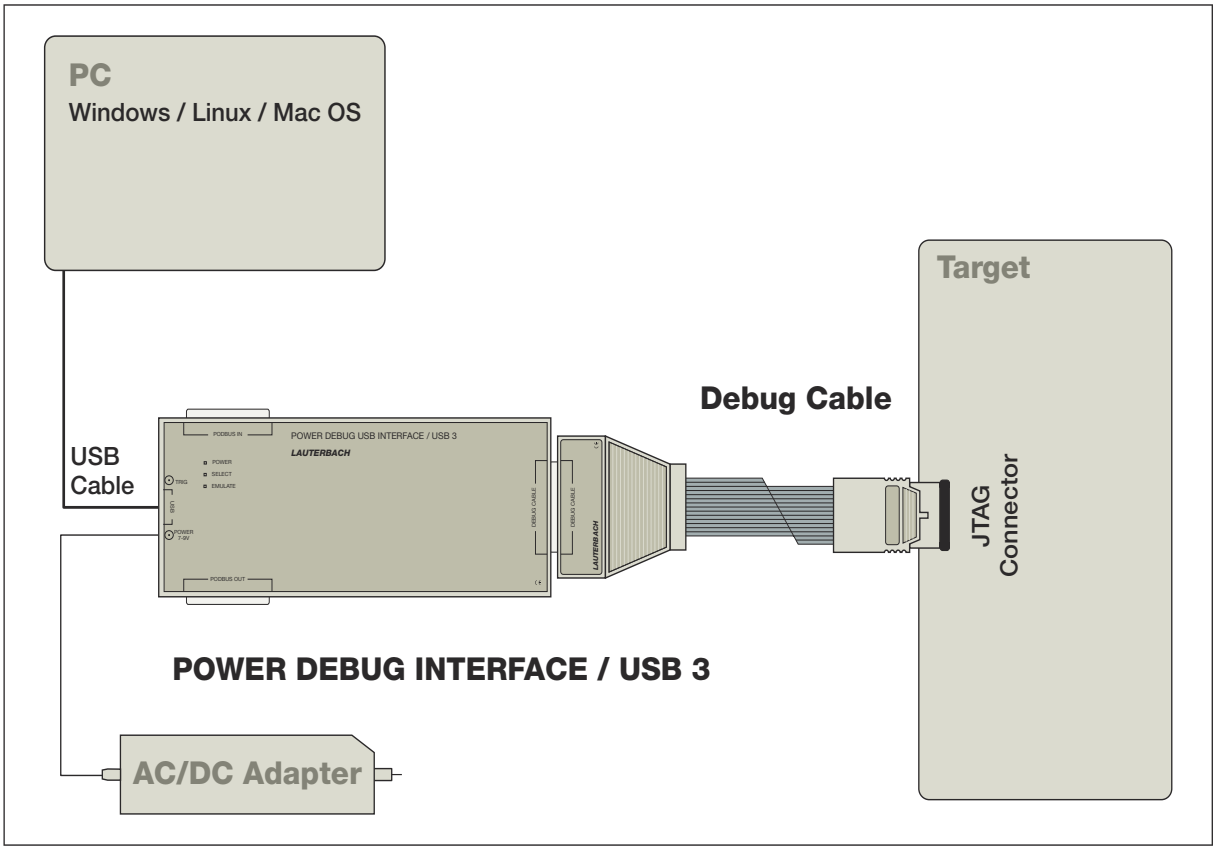
- JTAG connector only
- JTAG connector and Aurora connector (Power.org 22-, 46- or 70-pin connector)
- Aurora connector only (Power.org 22-, 46- or 70-pin connector)

Depending on your board, you might need to adjust some board specific settings to define which connector you want to use. Please refer to the configuration sheet of your board for further details.

If you want to start debugging right away, then simply check the two configuration options and use the working one. For the working configuration option, TRACE32 accepts the **SYStem.Up** command without displaying an error message.

TRACE32 Debugger

A QorIQ development board that allows only debugging, typically comes with a JTAG connector.

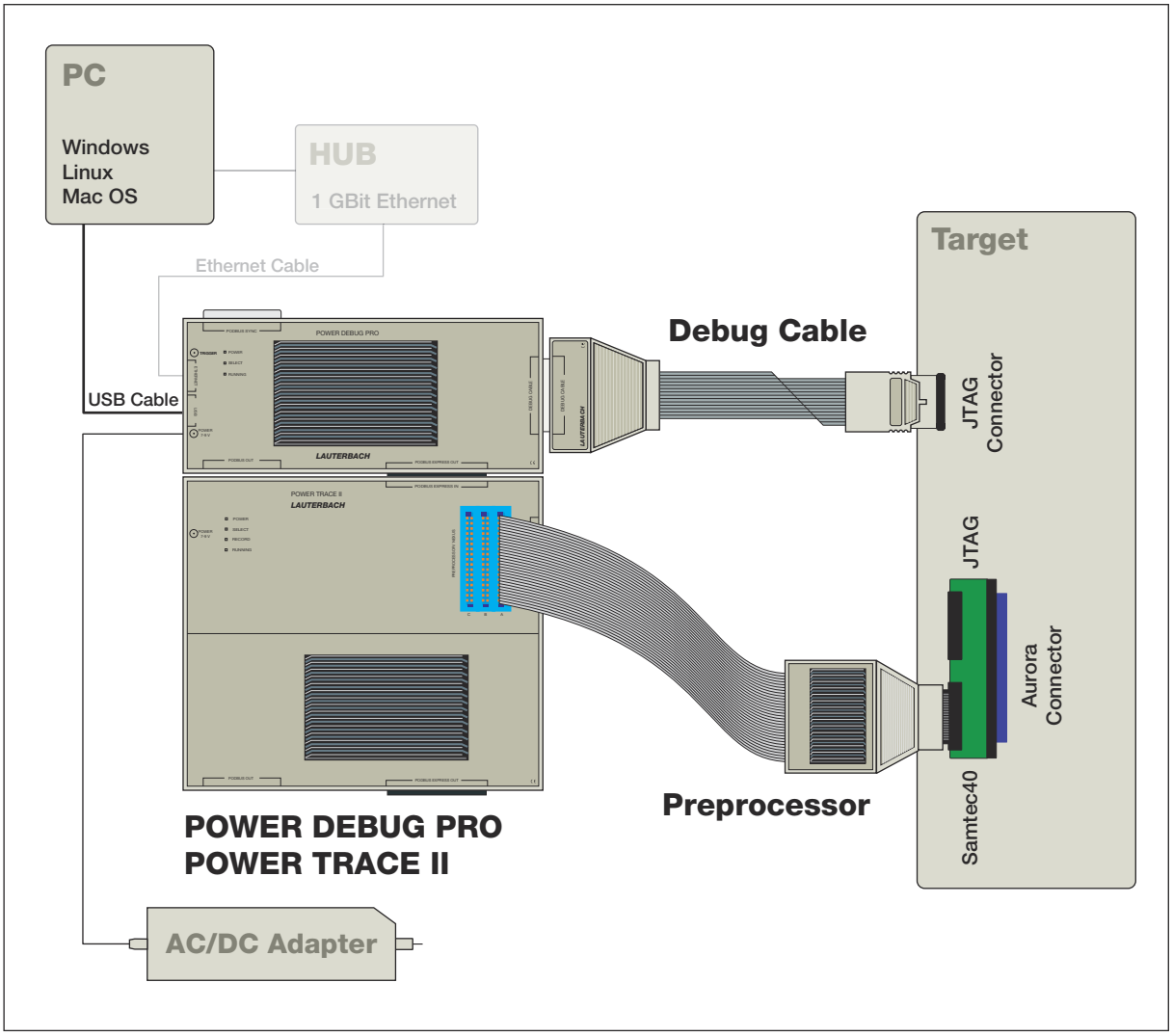


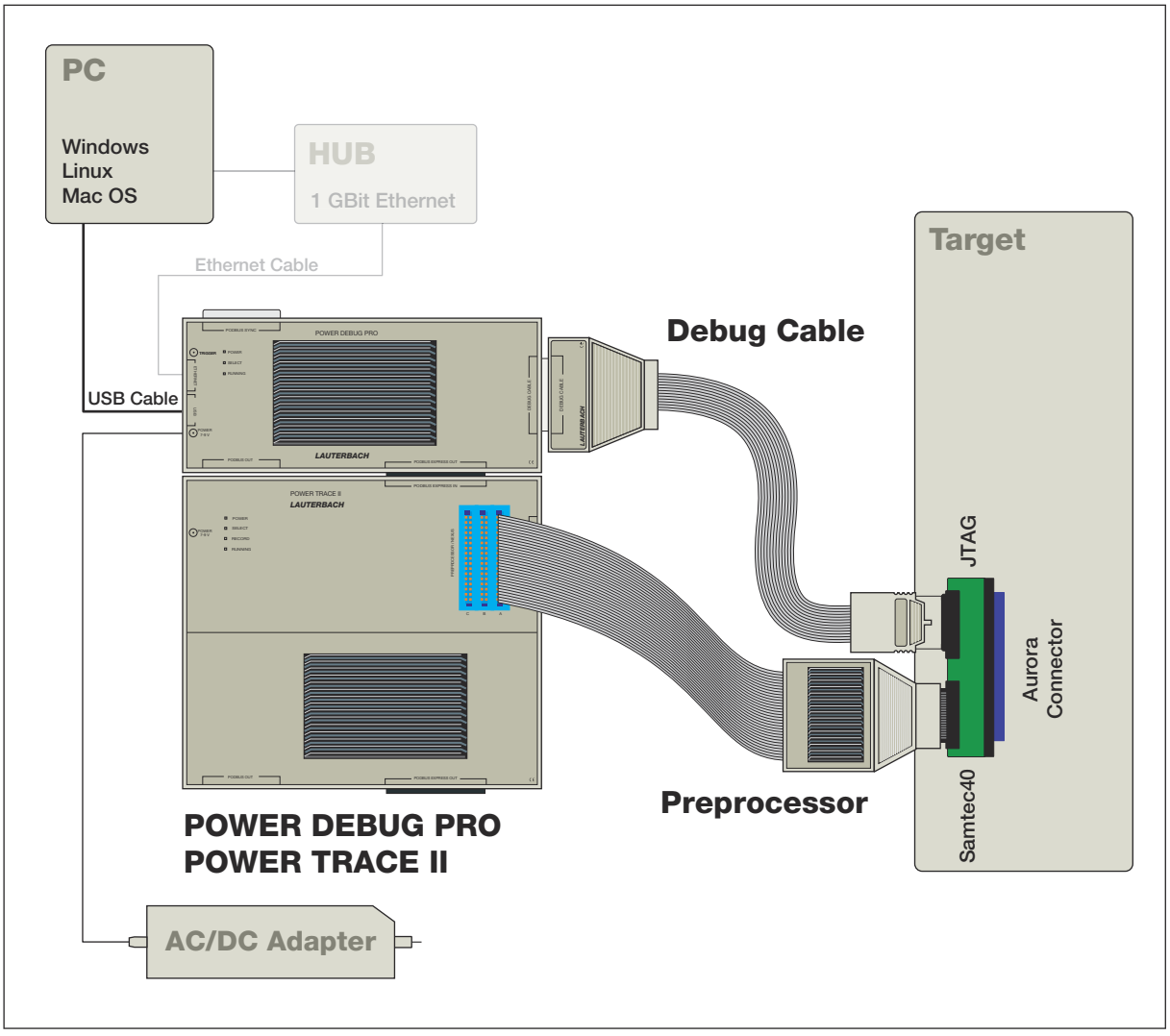
The TRACE32 PREPROCESSOR SERIAL for the QorIQ has a 40-pin connector. If you are using a Power.org defined connector (22-, 46- or 70-pin) on your target you will need a fitting Aurora converter to connect TRACE32 PREPROCESSOR SERIAL.

All Lauterbach Aurora converters provide a JTAG connector and a Samtec40 connector for the tool side and a Power.org connector for the target side. Depending on your board design you have to use either:

- The board JTAG connector to connect the TRACE32 Debug Cable and the Lauterbach Aurora converter to connect the TRACE32 Serial Preprocessor.
- Or you use the Aurora converter to connect both, the TRACE32 Debug Cable and the TRACE32 Serial Preprocessor.

JTAG and Aurora Connector





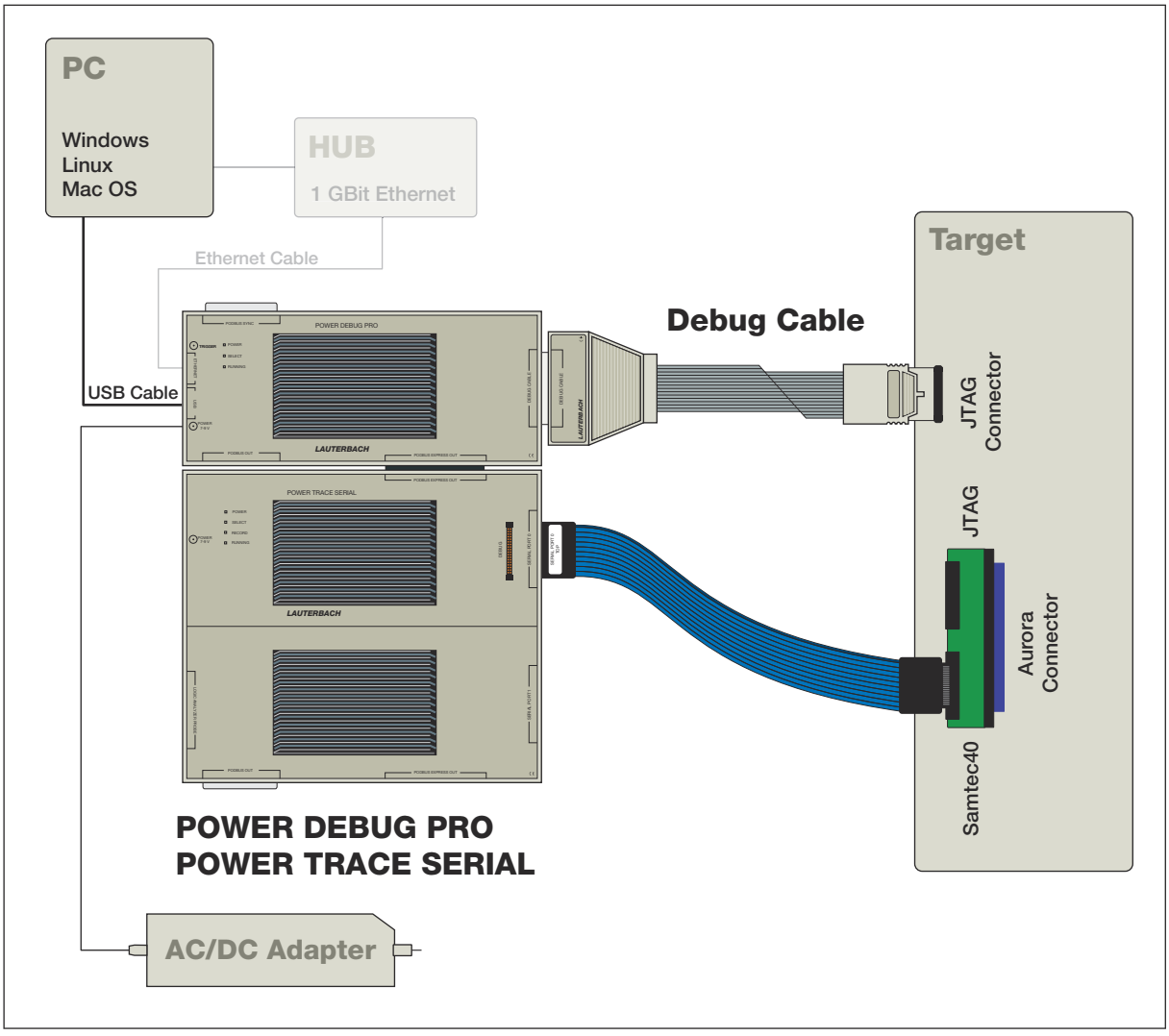
If you are interested in general information on PowerTrace Serial, please refer to **"PowerTrace Serial User´s Guide"** (serialtrace_user.pdf).

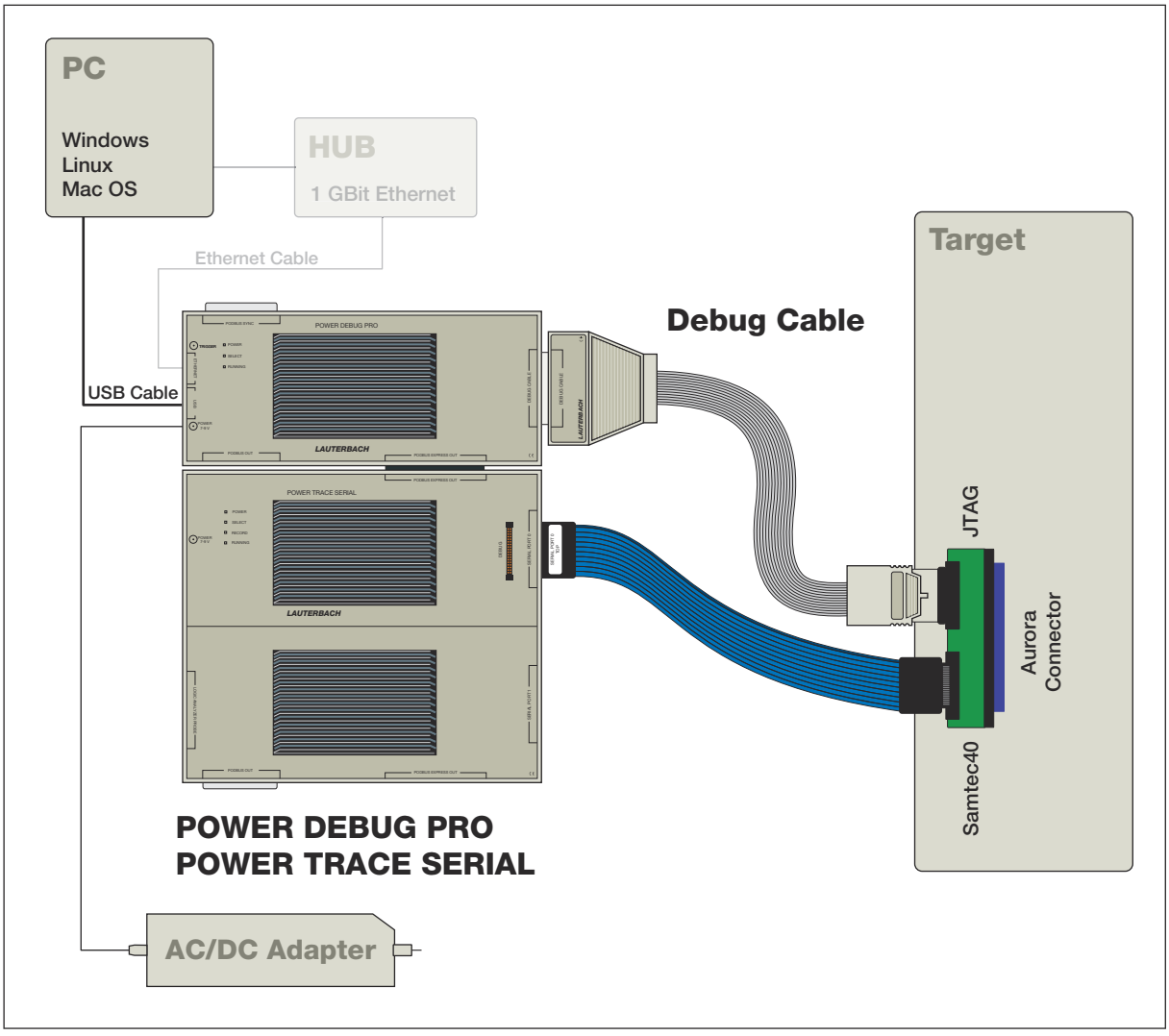
Aurora Traceport

The TRACE32 POWER TRACE SERIAL for the QorIQ has a 40-pin connector. If you are using a Power.org defined connector (22-, 46- or 70-pin) on your target you will need a fitting Aurora converter to connect TRACE32 POWER TRACE SERIAL.

All Lauterbach Aurora converters provide a JTAG connector and a Samtec40 connector for the tool side and a Power.org connector for the target side. Depending on your board design you have to use either:

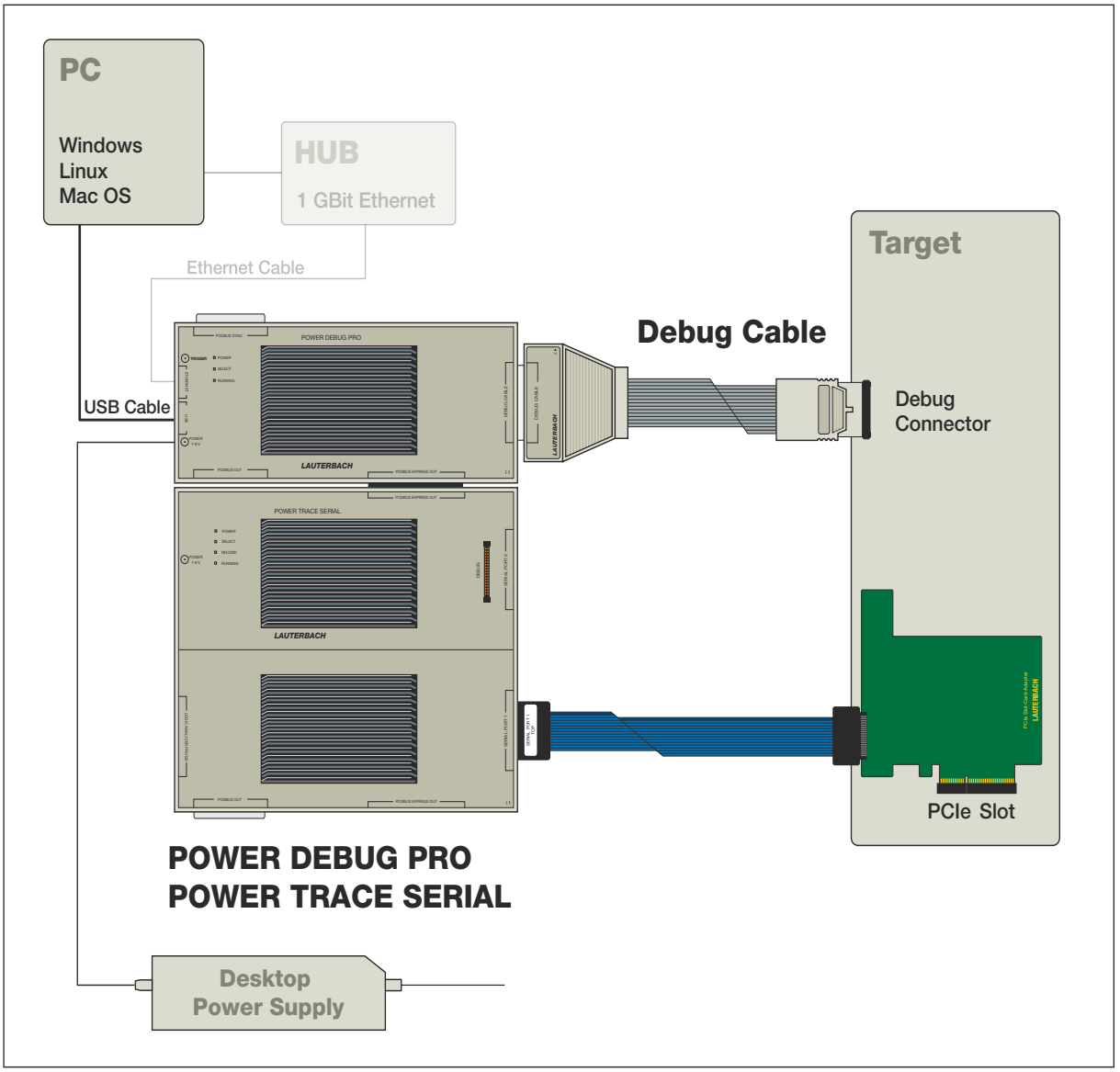
- The board JTAG connector to connect the TRACE32 Debug Cable and the Lauterbach Aurora converter to connect TRACE32 POWER TRACE SERIAL.
- Or you use the Aurora converter to connect both, the TRACE32 Debug Cable and TRACE32 POWER TRACE SERIAL.





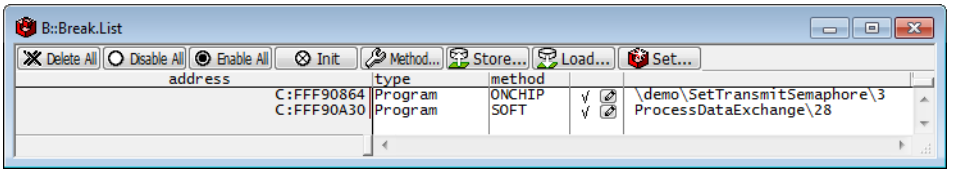
If your board does not provide an Aurora traceport, it is also possible to convey the NEXUS core trace information off-chip via the PCIe interface.

The TRACE32 POWER TRACE SERIAL for the QorIQ needs additionally a *TRACE32 License for PCI Express* in this case, and probably a *Lauterbach Slot-Card-Converter.*



Breakpoints

There are two types of breakpoints available: ONCHIP breakpoints and SOFTware breakpoints.



Software Breakpoints

To set a software breakpoint, before resuming the CPU, the debugger replaces the instruction at the breakpoint address with a **DNH** instruction.

On-chip Breakpoints

To set breakpoints on code in read-only memory, only the on-chip instruction address breakpoints are available. With the command **MAP.BOnchip** *<range>* it is possible to declare memory address ranges for use with on-chip breakpoints to the debugger. The number of breakpoints is then limited by the number of available on-chip instruction address breakpoints.

- **On-chip breakpoints:** Total amount of available on-chip breakpoints.
- **Instruction address breakpoints:** Number of on-chip breakpoints that can be used to set program breakpoints into ROM/FLASH/EEPROM.
- **Data address breakpoints:** Number of on-chip breakpoints that can be used as Read or Write breakpoints.

Core type (CPU types):	**On-chip Breakpoints**	**Instruction Address Breakpoints**	**Data Address Breakpoints**
e500mc (P204X, P30XX, P40XX), e5500 (P5XXX, T10XX)	2 instruction 2 read/write no counters	2 single breakpoints -- or -- 1 exact breakpoint range	2 single breakpoints -- or -- 1 exact breakpoint range -- or -- 2 ranges up to 4kB each **MAP.BOnchip** (exact or extended range) -- or -- 1 range up to 4kB (exact or extended range) and 1 single breakpoint

Core type (CPU types):	On-chip Breakpoints	Instruction Address Breakpoints	Data Address Breakpoints
e6500 (T2XXX, T4XXX, B4XXX)	8 instruction 2 read/write no counters	8 single breakpoints -- or -- 4 exact breakpoint ranges	2 single breakpoints -- or -- 1 exact breakpoint range -- or -- 2 ranges up to 4kB each (exact or extended range) -- or -- 1 range up to 4kB (exact or extended range) and 1 single breakpoint

NOTE:

- "exact or extended range": To use the increased number of data address breakpoint ranges with up to 4kB each, either **TrOnchip.CONVert** must be enabled or exact 4kB ranges must be used.
- Setting on-chip breakpoints with physical (real) address (Access Class Attribute "A:") is possible to simplify the usage for 1:1 translations and the peripheral handling. In any case the resulting hardware address comparison is based on effective addresses, TRACE32 will not convert physical to logical (effective) addresses!

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

If no more on-chip breakpoints are available, you will get an error message when trying to set a new on-chip breakpoint.

Breakpoints on Program Addresses

The debugger sets software and on-chip breakpoints to the effective address. If a breakpoint is set on a program address, the debugger will first try to set a software breakpoint. If writing the software breakpoint fails (translation error or bus error), then an on-chip breakpoint will be set instead. If a memory range must not be written by the debugger, it can be declared for on-chip breakpoint usage using **MAP.BOnchip**. Alternatively, it is also possible to force a single breakpoint to on-chip using the command **Break.Set** with option **/Onchip**:

```
Map.BOnchip 0xF8000000--0xFFFFFFFF ;use on-chip breakpoints in FLASH
Break.Set 0xFFFFF064               ;debugger sets on-chip breakpoint

Break.Set my_func1                 ;debugger sets on-chip or sw breakp.
Break.Set my_func1 /Onchip         ;debugger sets on-chip breakpoint
```

Breakpoints can be configured to stop if the break event occurred a given number of times. For all QorIQ CPUs no on-chip counter will be used.

```
;stop on the 20th call of function foo
Break.Set foo /Onchip /COUNT 20.
```

Breakpoints on Data Addresses

Data address breakpoints cause a debug event when a certain address or address range is read or written by the core. A data address breakpoint to a single address has a granularity of 1 byte.

```
Break.Set 0xC3F80004 /Read       ;break when core reads from 0xC3F80004
Break.Set 0xC3F80004 /Write      ;break when core writes to 0xC3F80004
Break.Set 0xC3F80004 /ReadWrite  ;break on read or write access

Break.Set 0xC3F80000--0xC3F80023 /Write    ;break address range
```

Similar to program address breakpoints, data address breakpoints can be configured to stop if the break event occurred a given number of times:

```
;stop on the 8th write to arrayindex
Break.Set arrayindex /Write /COUNT 20.
```

Data address breakpoint limitations:

1. The source of the data access (read and/or write) must be the core, as the data address breakpoints are part of the core. Any other accesses from on-chip or off-chip peripherals (DMA etc.) will not be recognized by the data address breakpoints.
2. The data being targeted must be qualified by an address in memory. It is not possible to set a data address breakpoint to GPR, SPR etc.

Breakpoints on Data Access at Program Address

A normal data access breakpoint as described above hits on all data accesses to the memory address or address range, independent of the program address which caused the access. It is also possible to set a data address breakpoint which only hits if the access is performed from a specified program address. The specified program address must be a load or store instruction.

```
;Break if the instruction at address 0x40001148 reads from variable count
  Break.Set 0x40001148 /MemoryRead count

;Break if the instruction at address 0x40001148 writes to range
  Break.Set 0x40001148 /MemoryWrite 0xFFFFF000--0xFFFFFFFF
```

The program address can also be an address range or a range of debug symbols:

```
;Break on all accesses to count from code of the address range
  Break.Set 0x40000100--0x400001ff /MemoryReadWrite count

;Break if variable nMyIntVar is written by an interrupt handler
;(debug symbols IVORxx_Handler loaded from debug symbols)
  Break.Set IVOR0_Handler--IVOR15_Handler /MemoryWrite nMyIntVar

;Break if variable nTestValue is written within function test_func
  Break.Set sYmbol.RANGE(test_func) /MemoryWrite nTestValue

;Break if variable nTestValue is written outside of test_func
  Break.Set sYmbol.RANGE(test_func) /EXclude /MemoryWrite nTestValue
```

Breakpoints on Data Value

The e500mc and e5500 cores do not support on-chip breakpoints on data values, but TRACE32 supports them by software emulation. When a data value breakpoint is set, the debugger will use one of the data address breakpoints. When the core hits that breakpoint, the target application will stop and the debugger will evaluate if the data value matches. If the value matches, the debugger will stop execution, if it does not match, the debugger will restart the application. Using software emulated data value breakpoints will cause the target application to slow down.

Examples for setting data value breakpoints:

```
;Break when the value 0x1233 is written to the 16-bit word at 0x40000200
  Break.Set 0x40000200 /Write /Data.Word 0x1233

;Break when a value not equal 0x98 is written to the 8-bit variable xval
  Break.Set xval /Write /Data.Byte !0x98

;Break when decimal 32-bit value 4000 is written
;to variable count within function foo
  Break.Set sYmbol.RANGE(foo) /MemoryWrite count /Data.Long 4000.
```

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 Attribute	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)
H	Hypervisor privilege level based access. The H access class is a generic placeholder for either the HS or the HU access class or a combination of both.
HS	Hypervisor-supervisor access. Access to supervisor memory with hypervisor privilege level.
HU	Hypervisor-user access. Access to user memory with hypervisor privilege level.
G	Guest privilege level based access. The G access class is a generic placeholder for either the GS or the GU access class or a combination of both.
GS	Guest-supervisor access. Access to supervisor memory with guest privilege level.
GU	Guest-user access. Access to user memory with guest privilege level.

If an access class attribute 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 expanded to **Data.List** UP:0x100.

The guest and hypervisor privilege level access classes H, HS, HU, G, GS and GU are important if **SYStem.Option.ZoneSPACES** is set to **ON**.

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
DBG	Special debug register access, e.g. Reset Configuration Word (RCW) register access

SPR and PMR registers are addressed by specifying the register number after the access class.

The access class DBG, which covers a wide variety of accesses, has a special encoding. The encoding as listed below is valid only for the QorIQ debugger.

DBG access mask	Description
`DBG:0x0100000R`	Access to the 16 RCW registers to set another RCW before the (next) **SYStem.Up**. `R`: Nexus register ID (0x0-0xF) For further details, please refer to **Programming the Reset Configuration Word**.

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** for details.				

The data cache logic of Power Architecture cores is described as states of the MESI protocol. The combinations for the MESI states are just available for DC and the unified L2 cache and thus the MESI state is just displayed for these cache windows in the column “#”.

State translation table:

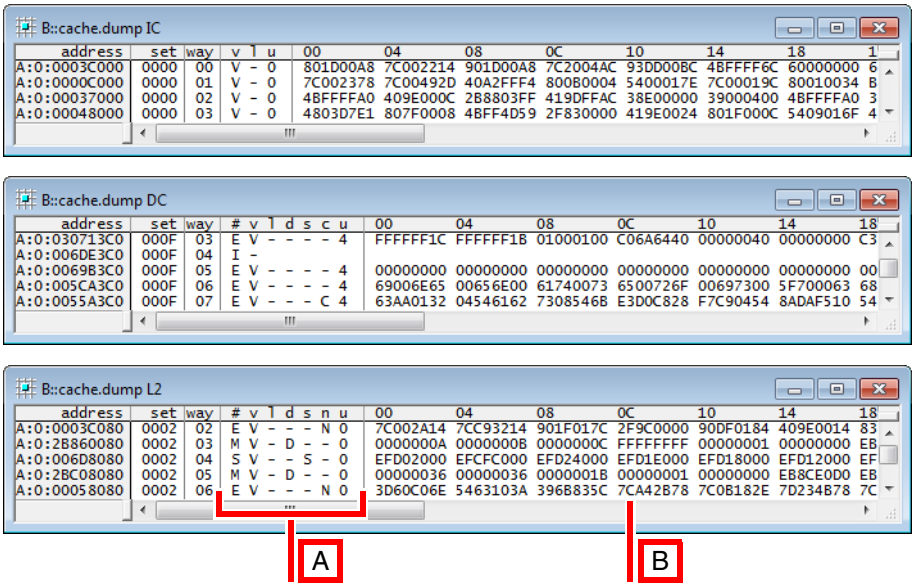
MESI state (#)	Flag
M (modified)	V(alid) && D(irty)
E (exclusive)	V(alid) && NOT D(irty)
S (shared)	V(alid) && S(hared)
I (invalid)	NOT V(alid)

The debugger also displays the cache state using the following cache line status flags:

- **v**alid (IC, DC, L2)
- **l**ocked (IC, DC, L2)
- **di**rty (DC, L2)
- **s**hared (DC, L2)
- **n**oncoherent (L2)
- **c**ast-out (DC)
- plr**u** (IC, DC, L2)

The cache contents can be viewed using the **CACHE.DUMP** command.

```
;Command            ; Cache
CACHE.DUMP IC       ; L1 instruction cache
CACHE.DUMP DC       ; L1 data dache
CACHE.DUMP L2       ; L2 (unified cache)
```



A MESI state, cache line status bits

B Cache line data

The meaning of the data fields in the **CACHE.DUMP** windows is explained in the following table:

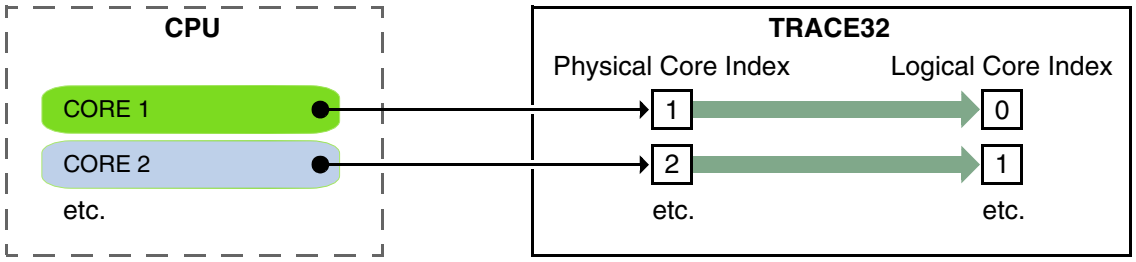
Data field	Meaning
address	Physical address of the cache line. The address is composed of cache tag and set index.
set way	Set and way index of the cache
#, v, l, d, s, n, c, u	Status bits of the cache line: # (MESI state), v(alid), l(ocked), d(irty), s(hared), n(oncoherent), c(ast-out), (pseudo least recently) u(sed)
00 04 08 ...	Address offsets within cache line corresponding to the cached data
address (right field)	Debug symbol assigned to address

Multicore Debugging

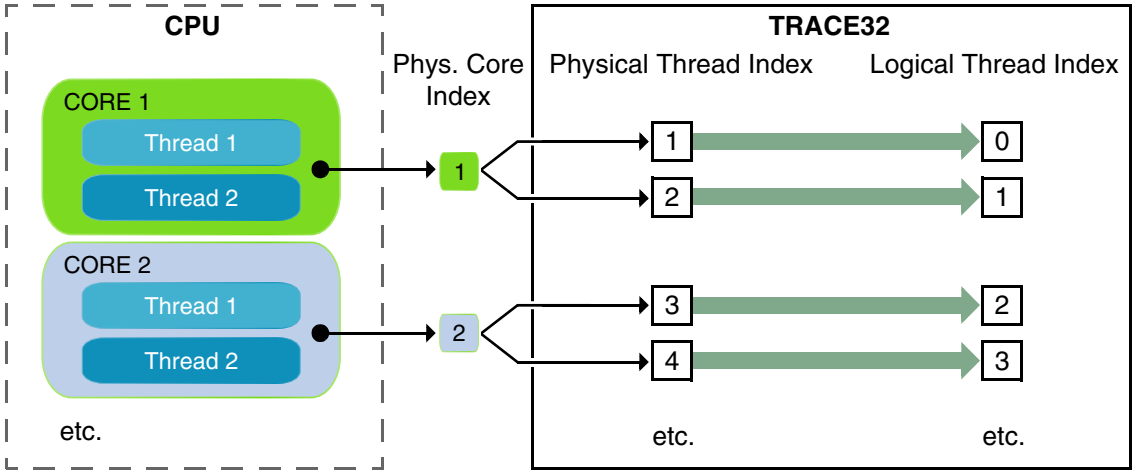
General Information

All QorIQ processors contain multiple cores that can be debugged as an **SMP** or an **AMP** system configuration. After the CPU has been selected, all physical cores / physical threads are assigned to this TRACE32 instance per default. The user can then choose which **logical core is displayed by TRACE32**. The resulting relationship between physical and logical cores is shown below:

- Processors with physical e500mc and e5500 cores, e.g. P4080:



- Processors with physical e6500 cores and physical threads, e.g. T2080:

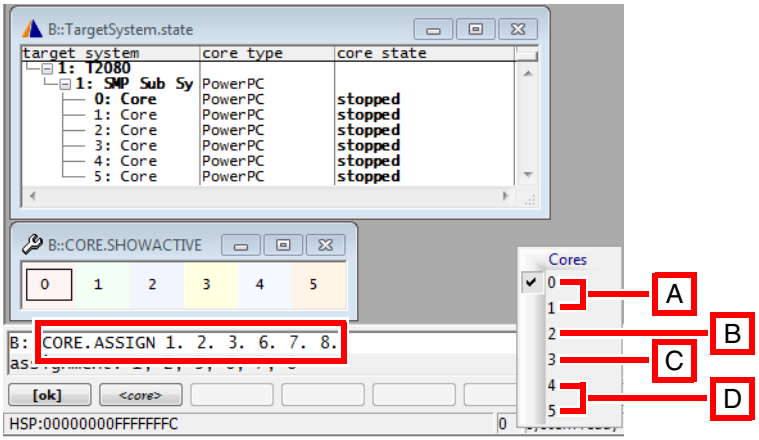


To choose a physical core or physical thread, you have the following options:

- Open the **TargetSystem.state** window, and double-click the logical core.
- Open the **CORE.SHOWACTIVE** window, and click the logical core.
- Right-click the status line core number box to display the list of logical cores, and click the logical core you want.
- Use the **CORE.select** *<logical_core_index>* command.

TRACE32 handles cores and threads with a unique TRACE32 instance related logical number.

Example for the T2080 processor with e6500 cores including two physical threads for each physical core:



A Physical core index 1, both threads (see **CORE.ASSIGN 1. 2.**)

B Physical core index 2, only thread 1 (see **CORE.ASSIGN 3.**)

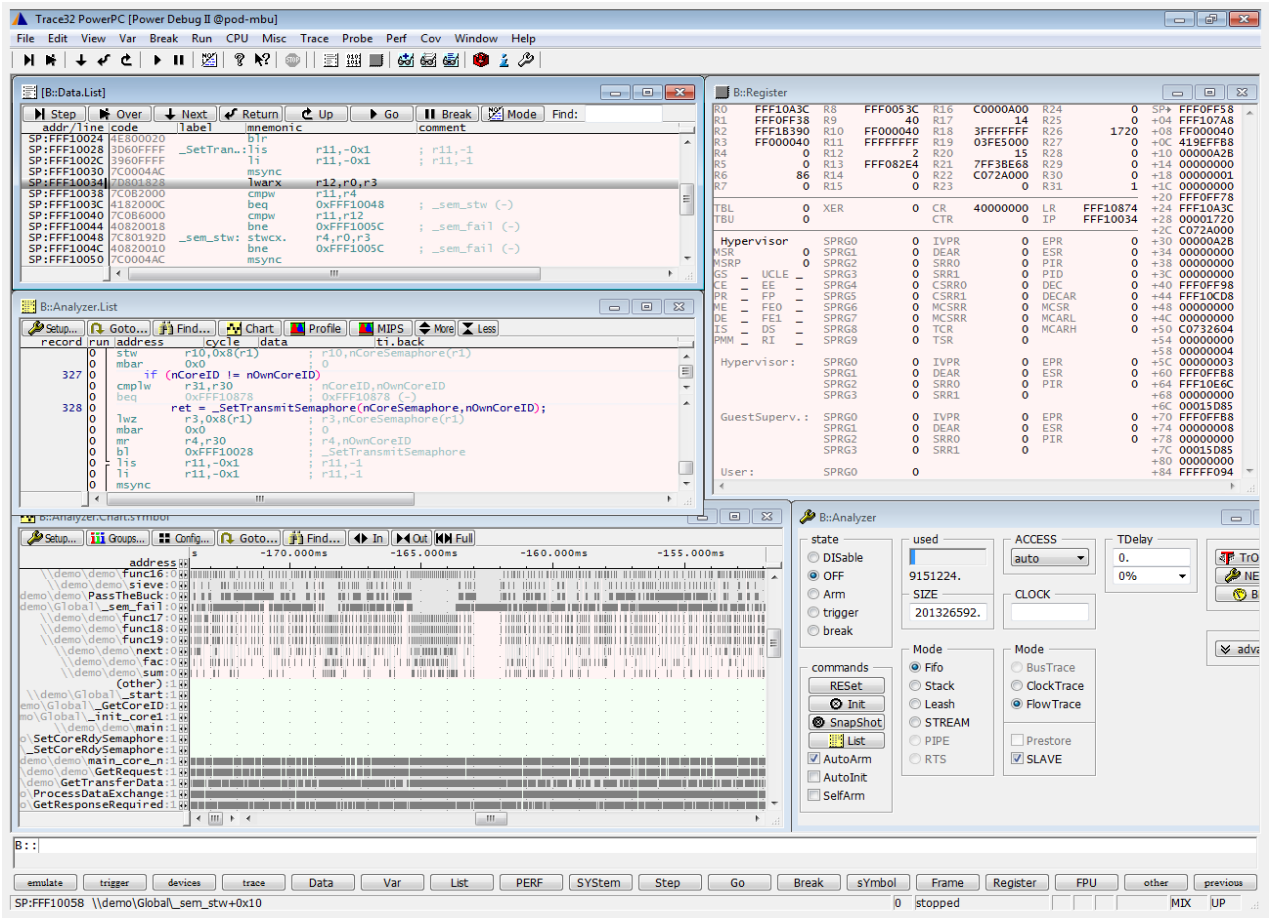
C Physical core index 3, only thread 2 (see **CORE.ASSIGN 6.**)

D Physical core index 4, both threads (see **CORE.ASSIGN 7. 8.**)

SMP Debugging

For all QorIQ processors **SMP (symmetric multiprocessing)** debugging is selected by default. No further configuration is needed if you want to debug all of the cores. If you want to specify which physical cores and threads you want to debug, use the commands **CORE.NUMber** or **CORE.ASSIGN.**

As soon as the debugger is connected (**SYStem.Up**, **SYStem.Attach** etc.), it is possible to switch to any assigned core using the **CORE.select** *<logical_core_index>* command. The currently selected core is displayed in the status line. If the cores are running and one of the cores hits a breakpoint, the debugger's view will automatically switch to this core. Further, all other assigned cores will be stopped nearly simultaneously. When resuming program execution (Go, HLL Step), all assigned cores will start simultaneously. If you step in assembler mode, just the selected core will execute the code.



AMP Debugging

There is a complete demo for debugging QorIQ processors on AMP mode in ~~/demo/powerpc/hardware/qoriq_p204x/all_boards/demo_amp_4cores_sram.
For **AMP (asymmetric multiprocessing)** debugging, a separate instance of PowerView has to be started for each core or each core compound. It is recommended to use **T32Start** to start the PowerView instances. Optionally, all other instances can also be started by PRACTICE script (see the demo mentioned above).

Each PowerView instance has to be configured to address at least one of the cores or rather the right core compound. This is done using the commands **SYStem.CONFIG.CORE** and either **CORE.ASSIGN** or **CORE.NUMber**.

SYStem Option DCFREEZE has to be turned OFF to maintain cache coherency for the times when one of the cores is running and the others stopped.

The following commands show the basic setup commands for two PowerView instances using all cores of the P4080. Using the command **CORE.ASSIGN** any of the cores can be assigned to a specific PowerView instance:

```
; CORE 0,1,5 and 6 setup script:

SYStem.CPU P4080

SYStem.CONFIG.CORE 1. 1.

CORE.ASSIGN 1,2,7,6

SYStem.Up
```

```
; CORE 2,3,4 and 7 setup script:

SYStem.CPU P4080

SYStem.CONFIG.CORE 2. 1.

CORE.ASSIGN 3,4,5,8


SYStem.Mode.ATTACH
```

If you just want to assign sequential cores to PowerView, you can also use the **CORE.NUMber** command, as in the example below.

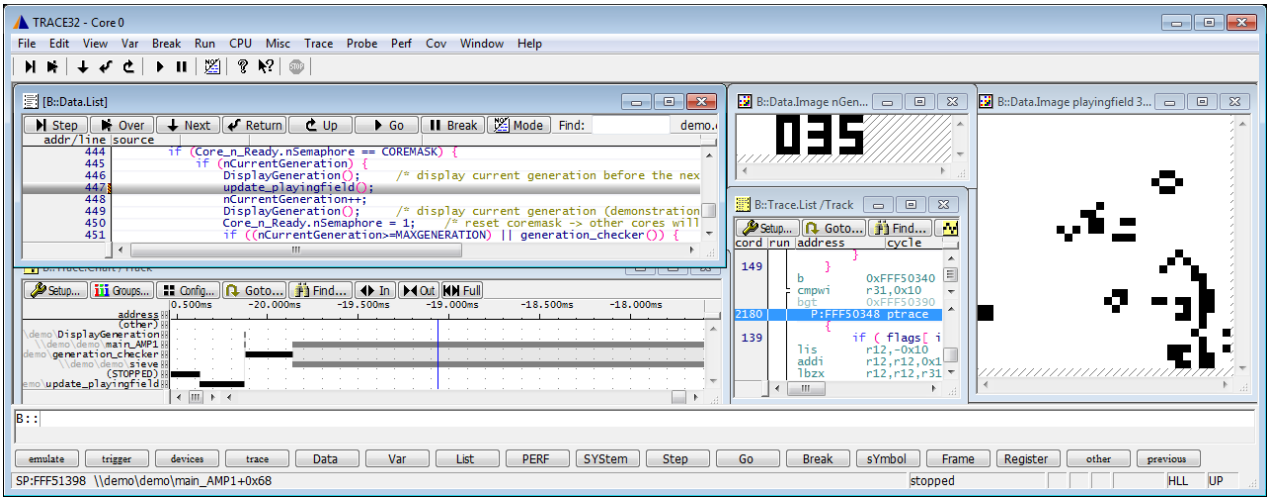
- In this case, the **SYStem.CONFIG.CORE** command specifies the number of the start core, e.g. start at core 5.
- The **CORE.NUMber** command then specifies the number of cores in the sequence, e.g. 4 cores starting at core 5 inclusive.

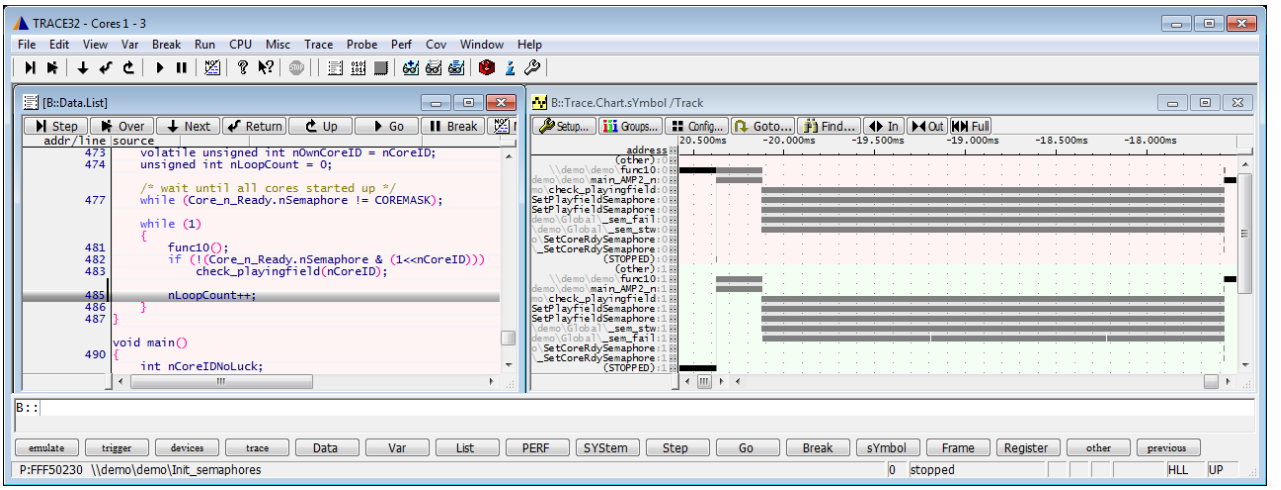
```
; CORE 0-3 setup script:

SYStem.CPU P4080

SYStem.CONFIG.CORE 1. 1.

CORE.NUMBER 4.

SYStem.Up
```

```
; CORE 4-7 setup script:

SYStem.CPU P4080

SYStem.CONFIG.CORE 5. 1.

CORE.NUMBER 4.


SYStem.Mode.ATTACH
```

In order to synchronously run and halt the cores of the two PowerView instances, use the **SYnch** commands.

The cores of one core compound (or rather one PowerView instance with multiple cores) behave like described in section “**SMP Debugging**”.





Synchronous Go of the Cores

In SMP mode all cores assigned to the PowerView instance will be started simultaneously.

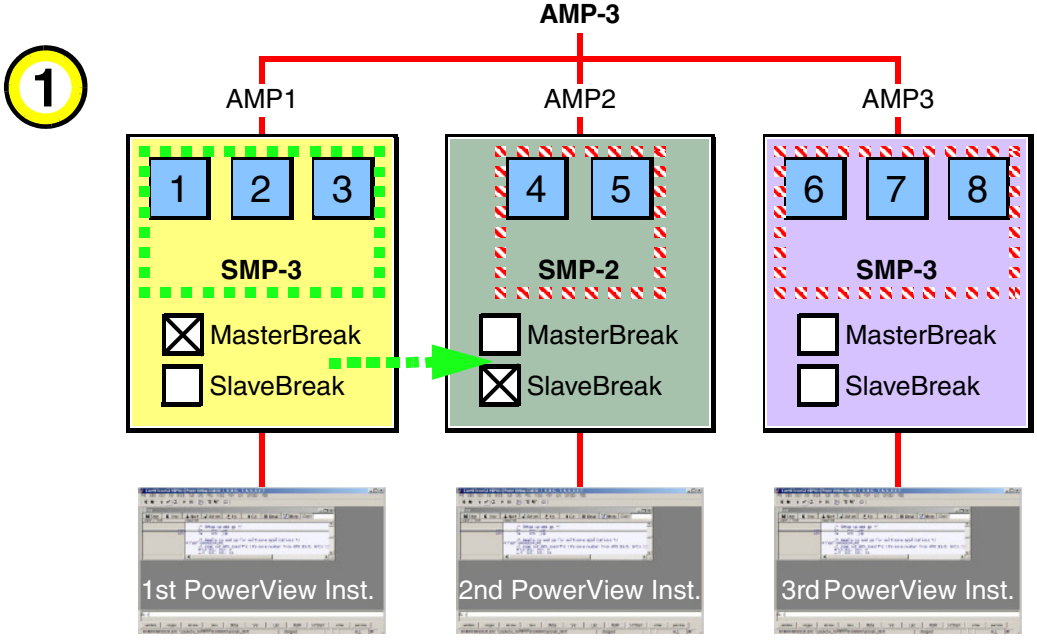
Also in AMP mode all cores can be started simultaneously, depending on the **SYnch** settings.

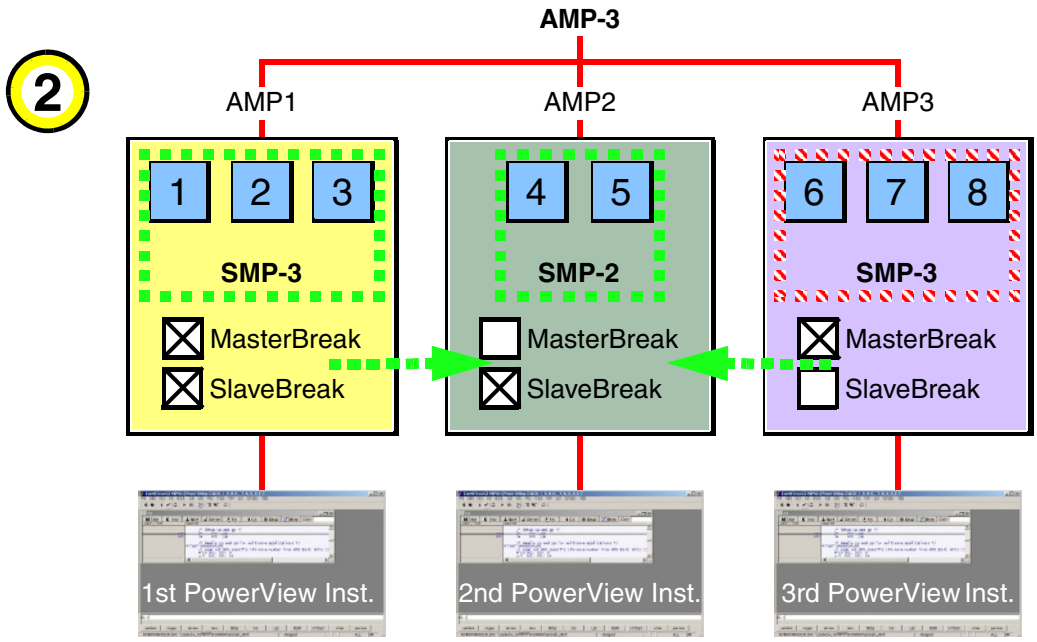
Synchronous Stop of the Cores

All QorIQ processors implement a break switch on silicon. If **SYnch** is configured to synchronous break in AMP mode (or always if SMP mode is selected), the core(s) that did not hit a breakpoint will be stopped by the processor hardware. This implementation causes a delay between all cores typically in the range of 5-50 instruction cycles.

Nevertheless, the hardware based synchronous break mechanism **may be limited if more than one instance of PowerView handles multiple cores**. Depending on the **SYNCH.MasterBreak** and **SYNCH.SlaveBreak** settings not all cores can be stopped synchronously by the hardware in all cases. The AMP synchronous break across instances of PowerView will always be handled by the hardware. The synchronous break of an SMP core compound inside of an AMP system may be handled by TRACE32, which typically leads to an increased break delay of the cores up to several milliseconds.

Please see the following two examples 1 and 2 in addition to the table below to get further information.





- Hardware based SMP synchronous halt. If one of these cores stops, all of them will stop simultaneously.
- Hardware based AMP synchronous halt. If any of the MasterBreak related cores stops, all of the SlaveBreak related cores will stop simultaneously.
- TRACE32 based SMP synchronous halt. If one of these cores stops, all others will be stopped by TRACE32 with increased delay.

	PowerView instance 1			*PowerView instance 2*			*PowerView instance 3*		
	Core count	**SYnch settings**	**HW synch. stop**	**Core count**	**SYnch settings**	**HW synch. stop**	**Core count**	**SYnch settings**	**HW synch. stop**
	1	any	YES	1	any	YES	1	any	YES
	n	any	YES	1	any	YES	1	any	YES
	n	Off MB SB	YES	n	Off MB SB	NO SMP	1	any	YES
	n	Off MB SB	YES	n	MB && SB	YES	1	any	YES
	n	MB && SB	YES	n	any	YES	1	any	YES
1	n	Off MB SB	YES	n	Off MB SB	NO SMP	n	Off MB SB	NO SMP
	n	Off MB SB	YES	n	Off MB SB	NO SMP	n	MB && SB	YES
	n	Off MB SB	YES	n	MB && SB	YES	n	Off MB SB	NO SMP
	n	Off MB SB	YES	n	MB && SB	YES	n	MB && SB	YES
2	n	MB && SB	YES	n	Off MB SB	YES	n	Off MB SB	NO SMP
	n	MB && SB	YES	n	Off MB SB	YES	n	MB && SB	YES
	n	MB && SB	YES	n	MB && SB	YES	n	Off MB SB	YES
	n	MB && SB	YES	n	MB && SB	YES	n	MB && SB	YES

- n >= 2
- MB = **SYNCH.MasterBreak** activated
- SB = **SYNCH.SlaveBreak** activated

The table above is also valid for more than three PowerView instances.

It explains that in any case the first PowerView instance with

- multiple cores assigned
- and turned off **SYNCH.MasterBreak** and / or **SYNCH.SlaveBreak**

will use the Hardware break switch for its SMP core compound.

All other PowerView instances (with the same requirements) will break its SMP related cores using a debugger based break mechanism.

Demo scripts for NOR FLASH and NAND FLASH are available in the following folders:

- ~~/demo/powerpc/hardware/qoriq_p204x/
- ~~/demo/powerpc/hardware/qoriq_p3/
- ~~/demo/powerpc/hardware/qoriq_p4/
- ~~/demo/powerpc64bit/hardware/qoriq_p5/
- ~~/demo/powerpc64bit/hardware/qoriq_t1/
- ~~/demo/powerpc64bit/hardware/qoriq_t2/
- ~~/demo/powerpc64bit/hardware/qoriq_t4/
- ~~/demo/powerpc64bit/hardware/qoriq_b4/

For NOR FLASH on eLBC or IFC, there are ready-to-use flash scripts, i.e. you do not need to modify them. These scripts can be found in the all_boards subfolders.

Some boards with faulty NOR FLASH FPGAs require a special handling with a slower flash algorithm. These scripts can be found in the subfolders of the respective board (e.g. t4240qds).

Scripts for NAND flash programming have to be modified with regard to the target board's characteristics and used FLASH devices. Therefore reference scripts usable on evaluation boards are included in the corresponding subfolder.

The RCW data is 512 bits long and is used by the pre-boot loader (PBL) to check consistency of the RCW data and load it into the RCW status registers. These 16 registers are for example responsible for the initial settings of the PLL configurations, SerDes lane assignments and settings, DDR configuration, and the boot location. If the RCW is unprogrammed or inconsistent, no program code will be executed. The RCW is typically part of the flash image, but it can also be generated using the debugger, e.g. if the SerDes lane settings have to be changed to enable Aurora HSTP trace (e.g. P2041):

```
;Enter prepare mode for restricted target access to read the current RCW
;for further adaptions
SYStem.Mode.Prepare
;Enable manipulation of the RCW
SYStem.Option.HRCWOVerRide ON
;Set user-defined RCW: SRDS_PRTCL for tracing purposes
Data.Set DBG:0x01000004 0x509f40C0
;Reset CPU with the user-defined RCW
SYStem.Up
;Disable manipulation of the RCW again
SYStem.Option.HRCWOVerRide OFF
...
;Reset CPU with the RCW from the target
SYStem.Up
```

Scripts for programming the RCW are available in the demo folder, e.g.
~~/demo/powerpc/hardware/qoriq_p204x/p2041rdb/demo_set_rcw.cmm

NOTE:

- The RCW is adapted only when **SYStem.Option HRCWOVerRide** is enabled before a **SYStem.Up.** When this system option is disabled again, all user-defined values will be lost and the original RCW will be used again for the following **SYStem.Up**.
- The PBL data structure consists of the RCW data, a preamble, pre-boot initialization commands and an end command including a CRC. If the user wants to **flash a new RCW**, this complete structure needs to be flashed including an appropriate CRC. Default values for Freescale evaluation boards are given in the above mentioned RCW demo scripts.
- Comparing the board initialization from a flash based RCW with the board initialization from a debugger set RCW might show PLL related differences. This issue is a typical behavior for the QorIQ devices and necessary to get the board into a working state again in every case. Therefore, setting the RCW using a debugger should be considered to be more like a help in need than a frequently-used method.

The QorIQ processors offer two trace destination possibilities. You can instruct TRACE32 to prepare the processor to send trace data to (1) the external Aurora HSTP port or (2) any other on-chip memory. The following list compares the two trace sinks:

1. External Aurora HSTP port

   - The Lauterbach Power Trace II module offers up to 4GByte, the POWER TRACE SERIAL offers 4GByte of trace memory.
   - The maximum QorIQ lane speed is supported (6.25 GBaud/s). Some of the QorIQ processors are restricted to a maximum of 5 or 6 GBaud/s.
   - Timestamps are added automatically by the Lauterbach Power Trace module. If an accuracy higher than ~4ns is required, you need to manually enable the QorIQ target timestamps in the **NEXUS.state** window. This leads to a higher bandwidth consumption of about 30 percent.
   - ETH GBit or USB3 connection for fast trace data transfer.

2. External NEXUS PCIe port

   - Available only for the POWER TRACE SERIAL, which is connected to the *Lauterbach PCIe Slot-Card-Converter* (see **PCIe Traceport**).
   - Same advantages as using the external Aurora HSTP port, but uses a standard PCIe slot that is available on most QorIQ target boards.
   - Depending on the RCW setting and lane routing even more bandwidth than with the external Aurora HSTP port.

3. On-chip memory

   - Dedicated trace memory on the target is needed. Typically a part of the DDR-SDRAM is used for tracing; max. 512Mb can be used due to QorIQ e500mc and e5500 processor restrictions (e6500 processors offers more, dependent on the available memory).
   - After halting and re-starting the core by the debugger, the onchip trace buffer will be reset.
   - If timestamps are required, you need to manually enable the target timestamps in the **NEXUS.state** window. This leads to a higher bandwidth consumption of about 30 percent.
   - Slow readout of the On-chip trace memory through JTAG.

Most of the target-specific demo scripts include examples of how to use the two trace sinks. The demo scripts reside in the following folders:

- ~~/demo/powerpc/hardware/qoriq_pxxxx
- ~~/demo/powerpc64bit/hardware/qoriq_xxxxx

Per default, the external Aurora HSTP will be set if a PowerTrace module is detected, otherwise a small onchip trace memory will be used in these demonstration scripts.

NOTE:

- The **Trace.state** window gives access to specific options of the trace method.
- The **NEXUS.state** window gives access to advanced options for both trace modes.
- The **Trace.List** window gives access to the recorded program trace data.

For more information about general trace commands see:

- 'Trace' in 'General Commands Reference Guide T'
- 'Analyzer' in 'General Commands Reference Guide A'
- 'Onchip Trace Commands' in 'General Commands Reference Guide O'
- **"Nexus Training"** (training_nexus.pdf)

Supported Trace Features

Core Feature	e500mc	e5500	e6500
Program Trace	Branch History	Branch History	Branch History
Data Trace	Address and Value (up to two 4kB ranges), write only	Address and Value (up to two 4kB ranges), write only	Address and Value (up to two 4kB ranges), write only
Data acquisition Trace	8bit Tag and 32bit Value (DEVENT/DDAM registers)	8bit Tag and 32bit Value (DEVENT/DDAM registers)	8bit Tag and 32bit Value (DEVENT/DDAM registers)
Watchpoint Message	Yes	Yes	Yes
Ownership Message	8bit PID / 32bit NPIDR	8bit PID / 32bit NPIDR	14bit PID / 32bit NPIDR
Filters	POTD, PTMARK	POTD, PTMARK	POTD, PTFPMM, PTFPR, PTFGS

All options are available in the **NEXUS.state** window and described in the **Trace Source settings section**.

The various PowerPC based QorIQ platforms also support In-Circuit Trace Messages, which are independent of the cores. These messages are used by TRACE32 to analyze the following trace sources:

- **DDR Trace: Includes memory controller ID, Read/Write address, ...****
- **OCeaN Trace: PCIe/sRapidIO. includes address, port, transmitted data, ...****

Both In-Circuit traces include much more information which is very dependent on the configured Verbose/Terse modes in the **NEXUS.state window and the used SoC. TRACE32 will analyze and display all available DDR and OCeaN trace message data. Please check your SoC specific manuals to get more information about the included trace data and the partly specific meaning.

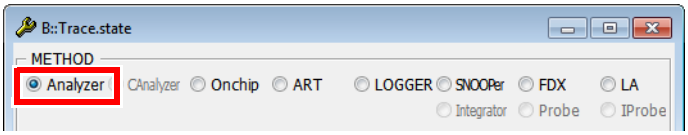
NOTE:	The data trace is configured using the triggers of breakpoints. The breakpoint has to be set up with “TraceData” as action and “Write” as type. More details are explained in **Filters and Triggers for the Nexus Trace**.

Aurora HSTP Trace

Processors of the QorIQ series offer the possibility to select the external Aurora HSTP port as a trace message destination. If a Power Trace module is connected, the processor will be automatically initialized and configured to record the program trace via this port, using our highest recommended lane speed for the set CPU.

Some QorIQ processors do not offer dedicated debug lanes (e.g. P2041). These lanes are configured by the Reset Configuration Word of the processor. The user needs to adapt the **SRDS_PRTCL** field of the **RCW** if no lane is configured for debug purposes. Please see **Programming the Reset Configuration Word** for further details.

The Aurora HSTP trace can be configured and accessed via the **Analyzer.state** window; alternatively, via the **Trace.state** window if the trace method is set to **Analyzer**. Then, click the **List** button in the window you have opened:



Nexus PCIe Trace

Not all boards offer the previously described Aurora port to give users the possibility to take advantage of external PowerTrace modules. But most boards offer a standard PCIe slot that can also be used for the connection of external tracing tools (see **PCIe Traceport**). The software configuration for this scenario is more complex but can be done by scripts or even the OS that is running on the target board. These steps should be followed in any case:

- Check the board schematics to know which SerDes lanes are routed to which PCIe slot.
- Check the processor reference manual, table “SerDes Lanes Assignments and Multiplexing” if your current **Reset Configuration Word** already supports PCIe on lanes that are routed to a PCIe slot. If this is not the case you can **temporarily override the current RCW** to set up the right lane assignment to the PCIe slot of your choice.
- Especially on evaluation boards from Freescale / NXP also check the qixis CPLD settings to ensure the lanes are really multiplexed the right way and the board is running in the right mode (if there is e.g. a “standalone mode” available).

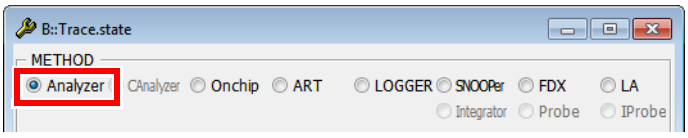
You now should know which PCIe controller (PCIe#) is connected to the PCIe slot of your choice to use it in the following configuration of TRACE32:

```
;Ensure the Lauterbach PCIe Slot-Card-Converter is inserted in the right
PCIe slot and the board powered afterwards. The values in this example
are valid for the T2080RDB board from Freescale / NXP with usage of PCIe
slot J20.
SYStem.CPU T2080
;Uses the same trace method as Aurora HSTP
Trace.METHOD Analyzer
;Configure the traceport to use PCIe as target connection.
SYStem.CONFIG.TRACEPORT.Type PCIE
;Set the used PCIe controller on the target (PCIe1 ... PCIe4)
NEXUS.USEPORT PCIE1
;Now connect to the target. If you need to override the RCW this is the
line to insert the commands as described in the chapter
Programming the Reset Configuration Word
SYStem.Mode UP
;Ensure the T2080RDB is running in standalone mode (qixis)
DO
~~/demo/powerpc64bit/hardware/qoriq_t2/t2080rdb/qixis_config_pciestandal
onemode.cmm

Trace.Arm
;Check the AREA window for error messages and warnings. At this point the
PCIe configuration on the target will be missing and TRACE32 will inform
you with a warning. It might not be missing if you attached to a running
target with an OS that already initialized the PCIe controller before.
In case of software configuration during bootup let the target run until
the task is done and afterwards re-arm the trace.
;In case of manual configuration some example scripts for evaluation
boards are provided (see below)
Go
Break
;list the recorded trace
Trace.List
```

Examples for manual PCIe configuration (please also see the comments inside the scripts) of some evaluation boards are provided in the board specific subdirectories of the demo folder, e.g.

- ~~/demo/powerpc/hardware/qoriq_p204x/p2041rdb/demo_pcie_trace.cmm
- ~~/demo/powerpc64bit/hardware/qoriq_t2/t2080rdb/demo_pcie_trace.cmm

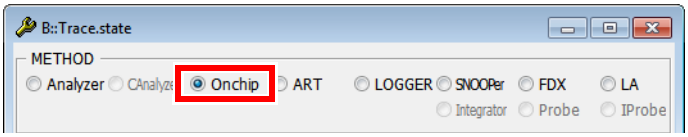
The Nexus PCIe trace can be configured and accessed via the **Analyzer.state** window; alternatively, via the **Trace.state** window if the trace method is set to **Analyzer**. Then, click the **List** button in the window you have opened:



Processors of the QorIQ series offer the possibility to select the memory bus as an on-chip trace message destination. Therefore, the size of the trace buffer is not fixed but limited, dependent on the SoC, the available memory size and **Onchip.TBARange** settings. Typically a part of the DDR-SDRAM is used for tracing:

```
SYStem.CPU P2041

;Initialize memory controller, LAWs, MMU
DO ~~/demo/powerpc/hardware/qoriq_p204x/p2041rdb/init.cmm

;load application, e.g.
Data.LOAD ~~/demo/powerpc/compiler/diab/diabcc.x

;Set onchip trace base address range to the initialized DDR-SDRAM
Onchip.TBARange 0x100000--0x4100000  ;e.g. 64MB

;let the CPU run to function sieve, automatically record trace
Go sieve

;list the recorded trace
Onchip.List
```

The on-chip trace can be configured and accessed via the **Onchip.state** window; alternatively via the **Trace.state** window if the trace method is set to **Onchip**. Then, click the **List** button in the window you have opened:



Trace initialization is a two-step process:

1. Initialization of the receiver / message destination, called the **Trace Sink**

   This step covers all target and debugger modules which are involved after a NEXUS message has been produced. E.g. Aurora specific or Onchip specific settings, ...

2. Initialization of the transmitter(s), called the **Trace Source**(s)

   This step covers all target modules which are necessary to produce a NEXUS Trace message. E.g. Core Trace (Program Trace, ...), DDR Trace, ...

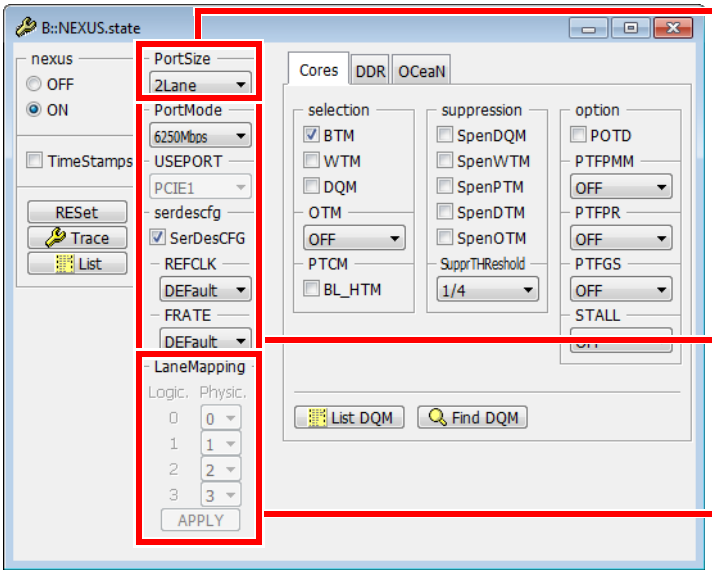
Trace Sink settings and processes - depending on the system state

1. After starting TRACE32 and selecting a QorIQ CPU: Debugger is in SYStem.Down state

- If a Lauterbach PowerTrace module is connected to the debugger, TRACE32 automatically selects the trace method **Analyzer**.
- If no external trace module is connected, the trace method **Onchip** is selected as soon as a QorIQ CPU is set in the **SYStem.state** window or after the CPU is set by the command:

```
SYStem.CPU P2041     ;Set the QorIQ CPU P2041

;Set a QorIQ CPU always leads to an automatic selection of on-chip or
;Aurora HSTP trace, depending on the debug hardware configuration.
```

- Trace method **Onchip**: While the debugger is in SYStem.Down, there is no target access, and consequently the onchip trace is disabled (**Onchip.DISable**).
- Trace method **Analyzer**: In the **Trace.state** window, you have access to the sink settings of the Lauterbach PowerTrace module, regardless of the system state. In the **NEXUS.state** window, you can set the port size, but only while the debugger is in **SYStem.Down** state.



Set **NEXUS.PortSize** to the lane number used on the target side.

NEXUS.PortMode and **NEXUS.SerDesCFG** can be changed in any system state and will influence the target settings directly in the state **SYStem.Up** or during the next **SYStem.Up** process. **NEXUS.USEPORT** (available for POWER TRACE SERIAL only) is available in case of PCIe-Trace to define the used PCIe module#.

NEXUS.LaneMapping (available for POWER TRACE SERIAL only) changes the logical to physical lane mapping, needed rarely.

2. **During a SYStem.Mode.Attach or SYStem.Up process**

   - If the trace method **Analyzer** is selected, the target CPU will be configured to bring the Aurora HSTP channel up and to send trace data as configured in the **NEXUS.state** window (also described in **Trace Source settings**).

   - The process will fail if the **NEXUS.state** settings described before don't match the target settings (e.g. wrong **SRDS_PRTCL** field settings of the **RCW**). Please observe the **AREA.view** window in case of problems and follow the instructions or contact our support.

3. **Active system: Debugger is in SYStem.Up state**

   - If the trace method **Analyzer** is selected, any change in the **NEXUS.state** window will lead to the same initializing procedure again **as during the SYStem.Up process**.

   - The trace method **Onchip** will be disabled until an appropriate **Onchip.TBARange** is set. TRACE32 will access and check if this range is really available and configure the target to use this memory area as trace message destination. These range addresses are physical addresses in any case. **Onchip.DISable** automatically switches to **Onchip.OFF** if the range is accepted.

Trace Source settings and trace access - regardless of the system state

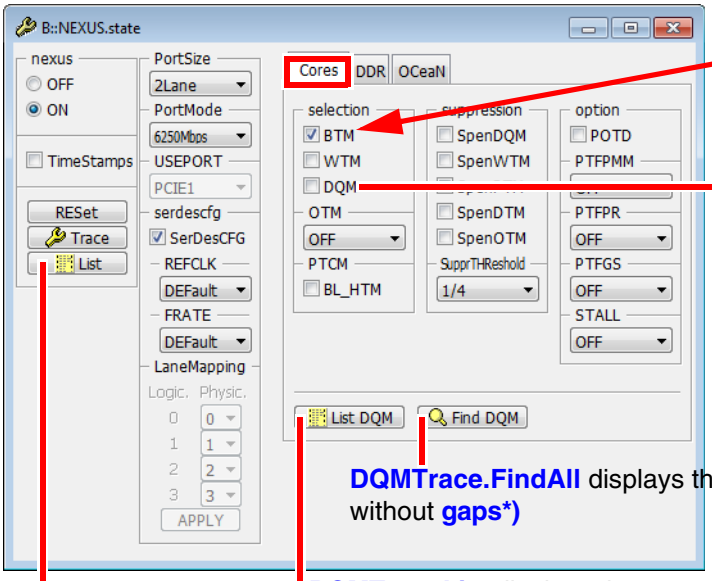
Unlike the **trace sink settings**, the trace source settings can be adapted regardless of the system state. The configuration will be modified directly in the **SYStem.Up** state or during the next **SYStem.Mode.Attach** or **SYStem.Up** process.

The **main trace sources** are available on the tabs in the **NEXUS.state** window:

1. **Cores**

   - NEXUS.BTM (branch history trace messages)

   - NEXUS.DQM (data acquisition trace messages)



NEXUS.BTM controls the program trace

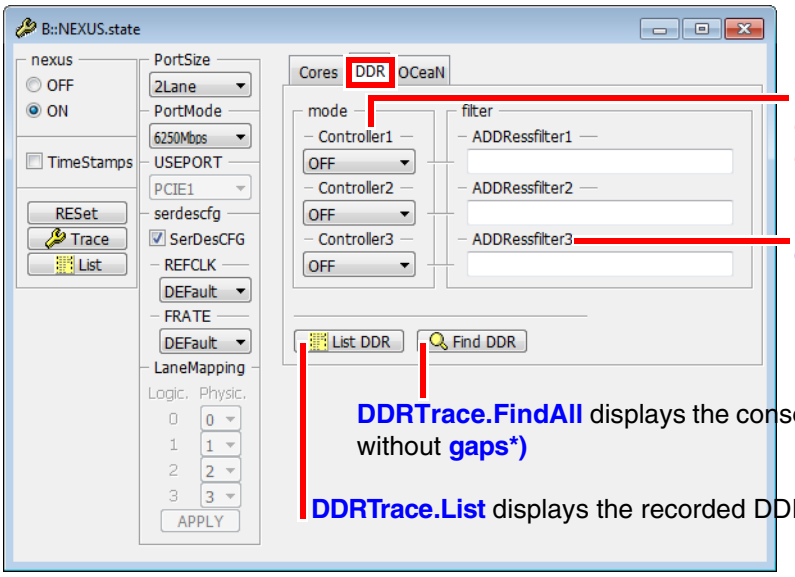
NEXUS.DQM controls the data acquisition trace

DQMTrace.FindAll displays the consolidated DQM trace messages without **gaps*)**

DQMTrace.List displays the recorded DQM trace.

Trace.List, **Analyzer.List**, **Onchip.List** display the recorded program trace.

2. DDR (DDR controller debug trace)



NEXUS.DDRConfig.Controller1 controls the DDR trace for the DDR controller 1.

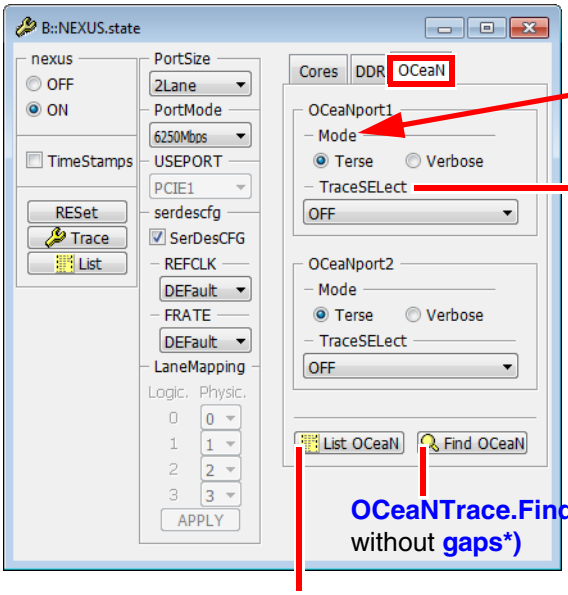
NEXUS.DDRConfig.ADDRessfilter3 restricts the addresses / ranges which produce DDR trace messages for DDR controller 3.

DDRTrace.FindAll displays the consolidated DDR trace messages without **gaps*)**

DDRTrace.List displays the recorded DDR trace.

NOTE:	The number of DDR memory controllers varies depending on the QorIQ CPU. The look of this window and the available commands regarding the memory controller number vary consequently.

3. OCeaN (On Chip Network debug trace)



NEXUS.OCeaNport1.Mode controls the OCeaN trace message format for port 1

NEXUS.OCeaNport1.TraceSELect configures an OCeaN trace source of port 1 to produce OCeaN trace messages.

OCeaNTrace.FindAll displays the consolidated OCeaN trace messages without **gaps*)**

OCeaNTrace.List displays the recorded OCeaN trace.

***)** A consolidated trace listing can be useful if different trace sources are combined. Finding all relevant trace messages matching the search criteria is more time consuming than displaying the default listing of the whole trace contents.

SYStem.BdmClock — Set debug clock frequency

Format:	**SYStem.BdmClock** *<rate>*
<rate>:	**100000.** … **50000000.** **100kHz** … **50MHz**

Selects the frequency for the debug interface. For multicore debugging, it is recommended to set the same JTAG frequency for all cores.

NOTE:	The recommended maximum JTAG frequency is 1/10th of the core frequency. The maximum JTAG frequency for multicore debugging of QorIQ processors is typically about 20 to 25MHz. The maximum JTAG frequency for single core debugging of QorIQ processors is typically about 50MHz.

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> (JTAG):

DRPRE
DRPOST
IRPRE
IRPOST

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

TAPState
TCKLevel
TriState
Slave

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 processor in the JTAG chain. The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See example below.

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

NOTE: When using the TriState mode, nTRST must have a pull-up resistor on the target. In TriState mode, a pull-down is recommended for TCK, but targets with pull-up are also supported.

… **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

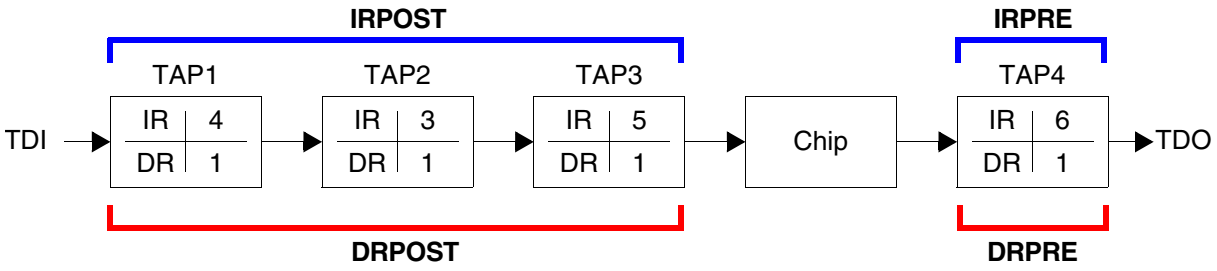
(default: 0) Level of TCK signal when all debuggers are tristated.

TriState

(default: OFF) If two or more debuggers share the same JTAG port, this option is required. The debugger switches to tristate mode after each JTAG access. Then other debuggers can access the port.

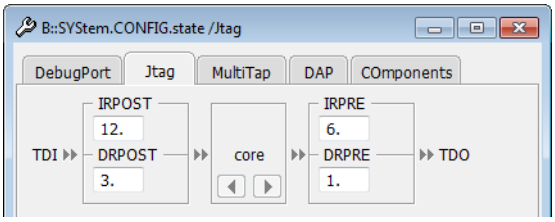
Slave

(default: OFF) If two or more debuggers share the same JTAG port, all except one must have this option active. Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).



IR: Instruction register length **DR**: Data register length **Chip**: The chip you want to debug

Daisy chains can be configured using a PRACTICE script (*.cmm) or the **SYStem.CONFIG.state** window.



Example: This script explains how to obtain the individual IR and DR values for the above daisy chain.

```
SYStem.CONFIG.state /Jtag     ; optional: open the window

SYStem.CONFIG IRPRE   6.      ; IRPRE: There is only one TAP.
                              ; So type just the IR bits of TAP4, i.e. 6.

SYStem.CONFIG IRPOST 12.      ; IRPOST: Add up the IR bits of TAP1, TAP2
                              ; and TAP3, i.e. 4. + 3. + 5. = 12.

SYStem.CONFIG DRPRE   1.      ; DRPRE: There is only one TAP which is
                              ; in BYPASS mode.
                              ; So type just the DR of TAP4, i.e. 1.

SYStem.CONFIG DRPOST  3.      ; DRPOST: Add up one DR bit per TAP which
                              ; is in BYPASS mode, i.e. 1. + 1. + 1. = 3.
                              ; This completes the configuration.
```

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 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.

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

<cpu>: **P2040** | **P2041** | **P3041** | **P4040** | …

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

- Check if the licence of the debug cable includes the desired CPU type. You will find the information in the **VERSION.view** window.
- Check the **VERSION.view** window to see which version is installed. CPUs that appeared later than the software release are usually not supported. Please check **www.lauterbach.com/update.html** for updates. If the needed CPU appeared after the release date of the debugger software, please contact technical support and request a software update.

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.

SYStem.MemAccess Run-time memory access (non-intrusive)

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

<mode>: **Denied** | **SAP** | **StopAndGo**

This option declares if and how a non-intrusive memory access can take place while the CPU is executing code. Although the CPU is not halted, run-time memory access creates an additional load on the processor's internal data bus.

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 using the setting **SYStem.Option DUALPORT ON**.

Denied	Memory access is disabled while the CPU is executing code.
SAP	The debugger performs memory accesses via the dedicated System Access Port. This memory access will snoop data cache and L2 cache if a access class for data (“D:”) is used.
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.

Format:	**SYStem.Mode** *<mode>*
<mode>:	**Down** | **NoDebug** | **Prepare** | **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. In this mode no debugging is possible. The CPU state keeps in the state of NoDebug.

Prepare Nearly disabled debugger. The state of the CPU remains unchanged, but dedicated access to the debug logic is possible. This state is needed to set a temporary new RCW when the current configuration is invalid. For further details please refer to **Programming the Reset Configuration Word**.

Go Resets the target with debug mode enabled and prepares the CPU for debug mode entry. Now, the processor can be stopped with the **Break** command or any break condition.

Attach Connect to the processor without resetting target/processor. Use this command to connect to the processor without changing its current state.

Up Resets the target/processor and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging.

SYStem.Option Address32 Define address format display

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

Default: AUTO.

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

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

SYStem.Option DCFREEZE Data cache state frozen while core halted

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

Default: OFF.

If **OFF**, the debugger will maintain D/L2 cache coherency by performing cache snoops for memory accesses. During the cache snoop, the processor will flush (clean and invalidate) dirty lines from data caches before the debugger's memory access takes place. This setting allows better data throughput and is recommended for normal application level debugging. In order to see changes to the cache state caused by debugging in the **CACHE.DUMP** window, use the command **CACHE.RELOAD**.

If **ON**, the debugger will maintain cache coherency by reading or writing directly to the cache. This method guarantees that the D/L2 cache tags and status bits (valid, dirty) remain unaffected by the memory accesses of the debugger. This setting is recommended for low-level and cache debugging.

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

Default: ON.

If enabled, **Data.dump** windows for access class D: (data) and variable windows display the memory values from the d-cache or L2 cache, if valid. If data is not available in cache, physical memory will be read.

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

Default: 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.

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

SYStem.Option FREEZE Freeze system timers on debug events

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

Default: OFF.

Enabling this option will instruct the debugger to set the FT bit in the DBCR0 register. This bit will cause the CPU to stop the system timers (TBU/TBL and DEC) upon all debug events that can be defined in DBCR0. The system timers will not be frozen on events like EVTI or the breakpoint instruction. Die timers/clocks or watchdogs of the on-chip peripherals are not affected by this option.

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** [**ON** | **OFF**]

Default: OFF.

Override the hard Reset Configuration Word on **SYStem.Up** via JTAG.

ON Every time this option is enabled, the current RCW configuration is read and set to the corresponding RCWSR registers. You can now change the entire RCW or just any of the 16 words. For further details please refer to **Programming the Reset Configuration Word**.
The user-set values will be set every time a **SYStem.Up** is performed.

OFF No more overriding of the RCW when a **SYStem.Up** is performed.

SYStem.Option ICFLUSH Invalidate instruction cache before go and step

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

Default: ON.

Invalidates the instruction cache before starting the target program (Step or Go). If this option is disabled, the debugger will update memory and instruction cache for program memory downloads, modifications and breakpoints. Disabling this option might cause performance decrease on memory accesses.

SYStem.Option ICREAD Read from instruction cache

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

Default: OFF:

If enabled, **Data.List** window and **Data.dump** window for access class P: (program memory) display the memory values from the instruction/unified cache or L2 cache if valid. If the data is not available in cache, the physical memory will be displayed.

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 MACHINESPACES** [**ON** | **OFF**]

Default: OFF

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

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

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

- Addresses are extended with an identifier called machine ID. The machine ID clearly specifies to which host or guest machine the address belongs.

  The host machine always uses machine ID 0. Guests have a machine ID larger than 0. TRACE32 currently supports machine IDs up to 30.
- The debugger address translation (**MMU** and **TRANSlation** command groups) can be individually configured for each virtual machine.
- Individual symbol sets can be loaded for each virtual machine.

SYStem.Option MMUSPACES Separate address spaces by space IDs

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

Default: OFF.

Enables the use of space IDs for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Glossary"** (glossary.pdf).

NOTE:	**SYStem.Option MMUSPACES** should not be 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
```

NOTE:	The option can only be enabled when there are no symbols loaded. Address dependent windows (e.g. **Data.List**) need to be closed and opened again after this setting is changed.

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

Default: OFF.

This setting affects the handling of on-chip debug events.

ON	The CPU will be configured to not stop for JTAG, but to enter the debug interrupt, like it does when no JTAG debugger is used.
OFF	If a JTAG debugger is used, the CPU is configured to stop for JTAG upon debug events.

Enable this option if the CPU should not stop for JTAG on debug events, in order to allow a target application to use the debug interrupt. Typical usages for this option are run-mode debugging (e.g. with t32server/gdbserver) or setting up the system for a branch trace via **LOGGER** (trace data in target RAM).

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>
```

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.

SYStem.Option SLOWRESET Relaxed reset timing

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

Default: OFF.

This system option defines how the debugger will test JTAG_HRESET. For some system mode changes, the debugger will assert JTAG_HRESET.

ON If this system option is enabled, the debugger will not read JTAG_HRESET, but instead waits 4 s and then assumes that the boards $\overline{\text{HRESET}}$ is released.

OFF Per default (OFF), the debugger will release RESET and then read the $\overline{\text{HRESET}}$ signal until the $\overline{\text{HRESET}}$ pin is released. Reset circuits of some target boards prevent that the current level of $\overline{\text{HRESET}}$ can be determined via JTAG_HRESET.

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

Default: 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.

SYStem.Option TranslationSPACE Identify user and hypervisor modes

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

Default: ON.

This system option configures the debugger how to distinguish between user and supervisor modes.

In bare-metal applications or uncomplex operating systems typically the MSR[IS] bit is used to isolate user from supervisor address space. There is no way to get the information about this bit within the trace information, the program trace will be decoded using the current context of the cores.

In complex or hypervisor systems, typically the MSR[PR] bit is used to handle user and supervisor modes. This bit will also be included in ownership trace messages. TRACE32 will therefore be able to decode the program trace depending on this privilege information, which doesn't have to be compliant to the current context of cores.

a) **SYStem.Option.TranslationSPACE ON** (default)

Mode	MSR.GS bit	MSR.IS bit
Hypervisor-supervisor mode	0	0
Hypervisor-user	0	1
Guest-supervisor	1	0
Guest-user	1	1

b) **SYStem.Option.TranslationSPACE OFF**

Mode	MSR.GS bit	MSR.PR bit
Hypervisor-supervisor mode	0	0
Hypervisor-user	0	1
Guest-supervisor	1	0

Mode	MSR.GS bit	MSR.PR bit
Guest-user	1	1

SYStem.Option ZoneSPACES Enable symbol management for zones

[Example]

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

Default: OFF.

The **SYStem.Option ZoneSPACES** command must be set to **ON** if separate symbol sets and MMU translation tables are used for the CPU operation modes:

- Hypervisor-supervisor mode
- Hypervisor-user mode
- Guest-supervisor mode
- Guest-user mode

Within TRACE32, these CPU operation modes are referred to as zones. For information about the status bits controlling these modes, see **SYStem.Option TranslationSPACE**.

NOTE: For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Glossary"** (glossary.pdf).

In each CPU operation mode (zone), the CPU's TLB may contain separate translations, and a kernel or hypervisor may uses separate MMU translation tables for memory accesses and separate register sets. Consequently, in each zone, different code and data can be visible on the same logical address.

OFF	TRACE32 does not separate symbols by access class. Loading two or more symbol sets with overlapping address ranges will result in unpredictable behavior. Loaded symbols are independent of the CPU mode.
ON	Separate symbol sets can be loaded for each zone, even with overlapping address ranges. Loaded symbols are specific to one of the CPU zones.

SYStem.Option ZoneSPACES ON in Detail

SYStem.Option ZoneSPACES is usually set to **ON** if you need to debug virtualized systems with guest and hypervisor. For both guest and hypervisor, TRACE32 also separates between supervisor mode and user mode. Typical scenarios use separate symbol sets for the hypervisor-supervisor mode, the guest-supervisor and the guest-user mode. The hypervisor-user mode is rarely used. The symbol sets are loaded to the access classes HS: (hypervisor-supervisor mode, GS: (guest-supervisor mode) and GU: (guest-user mode).

If **SYStem.Option ZoneSPACES** is **ON**, TRACE32 enforces any memory address specified in a TRACE32 command to have an access class which clearly indicates to which of the four zones the memory address belongs.

If an address specified in a command uses an anonymous access class such as D:, P: or C:, the access class of the current PC context is used to complete the access class of the addresses. Also, if an incomplete access class where either the guest/hypervisor information is missing (such as SP: or UP:) or the supervisor/user information is missing (such as GP: or HP:), the missing information will automatically be expanded from the access class of the current PC context.

Example: If the CPU is currently in user mode, a memory access with the access class GP: will be expanded by TRACE32 to become GUP:

If a symbol is referenced by name, the associated access class of its zone will be used automatically, so that the memory access is done within the correct CPU mode context. As a result, the symbol's effective address will be translated to the physical address with the correct MMU translation table.

Example 1

In this script, **SYStem.Option ZoneSPACES** is used for a simple host and guest debugging.

```
SYStem.Option ZoneSPACES ON

; 1. Load the Xen hypervisor symbols to the hypervisor-supervisor
; access class HS:
Data.LOAD.ELF xen-syms HS:0x0 /NoCODE

; 2. Load the vmlinux kernel symbols to the guest-supervisor access class
; GS:
Data.LOAD.ELF vmlinux GS:0x0 /NoCODE

; 3. Load the guest application symbols (the 'sieve' application in this
; example) to the guest-user access class GU:
Data.LOAD.ELF sieve GU:0x0 /NoCODE
```

Effect on the TRANSlation command group: **SYStem.Option.ZoneSPACES ON** enforces separate address spaces for the four zones HS:, HU:, GS: and GU:. Commands affecting the address translation, such as **TRANSlation.Create**, **TRANSlation.COMMON**, **TRANSlation.Protect** or **MMU.FORMAT**, must be executed individually for each of the four zones.
It is, however, possible to use the generic access classes G: and H: as "joker". This simplifies the scripts if identical translations for GS: and GU: are needed or identical translations for HS: and HU: are needed.

```
SYStem.Option ZoneSPACES ON
; show the list of static translations created by the commands
; TRANSlation.Create and TRANSlation.COMMON
TRANSlation.List

;1. the command
TRANSlation.Create G:0x80000000--0x8FFFFFFF 0x0
; is equivalent to the commands
TRANSlation.Create GS:0x80000000--0x8FFFFFFF 0x0
TRANSlation.Create GU:0x80000000--0x8FFFFFFF 0x0

;2. the command
TRANSlation.Create H:0xA0000000--0xAFFFFFFF 0x0
; is equivalent to the commands
TRANSlation.Create HS:0xA0000000--0xAFFFFFFF 0x0
TRANSlation.Create HU:0xA0000000--0xAFFFFFFF 0x0

;3. the command
TRANSlation.COMMON G:0xC00000000--0xFFFFFFFF
; is equivalent to the commands
TRANSlation.COMMON GS:0xC00000000--0xFFFFFFFF
TRANSlation.COMMON GU:0xC00000000--0xFFFFFFFF
```

MMU.DUMP — Page wise display of MMU translation table

Format:	**MMU.DUMP** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] [**/***<option>*]
	MMU.*<table>***.dump** (deprecated)
<table>:	**PageTable**
	KernelPageTable
	TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

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 or a specific machine if a space ID and/or a machine ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID and/or machine ID: displays the translation table of the specified process and/or machine • 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**.
MACHINE *<machine_magic>* | *<machine_id>* | *<machine_name>*	This option is only available if **SYStem.Option MACHINESPACES** is set to **ON**. Dumps a page table of a virtual machine. The **MACHINE** option applies to **PageTable** and **KernelPageTable** and some *<cpu_specific_tables>*. The parameters *<machine_magic>*, *<machine_id>* and *<machine_name>* are displayed in the **TASK.List.MACHINES** window.

CPU specific Tables in MMU.DUMP <table>

TLB0	Displays the contents of TLB0.
TLB1	Displays the contents of TLB1.
TLB1PT *<tlb1_index>*	Displays the indirect page table which is associated with the TLB1 entry *<tlb1_index>*

Additionally, if both **SYStem.Option ZoneSPACES** and **SYStem.Option MACHINESPACES** are set to **ON**, then these CPU specific tables are available:

SupervisorPT	Displays the supervisor mode page table of the machine specified with option **MACHINE**.
UserPT	Displays the user mode page table of the machine specified with the option **MACHINE**.

Additionally, if only **SYStem.Option ZoneSPACES** is set to **ON**, then these CPU specific tables are available:

HSPageTable	Displays the page table which is defined for the hypervisor supervisor mode.
HUPageTable	Displays the page table which is defined for the hypervisor user mode.
GSPageTable	Displays the page table which is defined for the guest supervisor mode.
GUPageTable	Displays the page table which is defined for the guest user mode.

[Examples]

Format:	**MMU.FORMAT** *<format>* [*<base_address>* [*<logical_kernel_address_range>* *<physical_kernel_address>*]] [**/***<option>*]
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **Hypervisormode** | **Guestmode** | **Supervisormode** | **Usermode**

Default *<format>*: STD.

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

<format>

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

<format>	Description
VXWORKS.E500	VxWorks specific format for PowerPC e500 core
VXWORKS.E500_64	VxWorks specific format for PowerPC e500 core (PPC64 only)
VXWORKS.E6500	VxWorks specific format for PowerPC e6500 core
PIKEOS.OEA	PIKEOS specific format for PowerPC core (formerly named PIKEOS) */
PIKEOS.E500	PIKEOS specific format for PowerPC e500 core (formerly named PIKEOSE5).Works for PikeOS 4.1 and older. For e500 cores with PikeOS 4.2 and newer use E500MC format.*/
PIKEOS.E500MC	PIKEOS specific format for PowerPC e500mc core (PPC64 only).Can also be used with PikeOS 4.2 and newer on PPC32 e500 cores.*/
PIKEOS.E500MC4G	PIKEOS specific format for PowerPC e500mc core addressing 4GB of memory.Has no common address range.*/
PIKEOS.E5500	PIKEOS specific format for PowerPC e5500 core
STD	Standard format defined by the CPU
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
LINUXEXT	Linux with 64-bit PTEs, no e500 core
LINUXE5	Linux with 64-bit PTEs, e500 core

<format>	Description
LINUX64_E6	Use LINUX64_E6 for e6500 core devices
LYNXOS	LynxOS format, virtual table pointers
LYNXOSPHYS	LynxOS format, physical table pointers
QNX	QNX standard format
QNXBIG	QNX format with 64-bit table entries
DEOS	DEOS OS (32 bit) specific MMU format
DEOS64	DEOS OS (64 bit) specific MMU format
OSE	OSE format for load modules
VX653	MMU format for VXWORKS 653
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.

<base_address>

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

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

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

<logical_kernel_address_range> and <physical_kernel_address> for the Default Translation

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

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

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

Supervisormode	If **SYStem.Option.MACHINESPACES** is set to **OFF**: Specifies the format, default page table, and default translation for one or both supervisor zones (access class HS: or GS:). Can be combined with the **Hypervisormode** or **Guestmode** option. If **SYStem.Option.MACHINESPACES** is set to **ON**: Specifies the format, default page table, and default translation for the supervisor mode zone of the machine selected with the **MACHINE** option. For an example, see below.
Usermode	If **SYStem.Option.MACHINESPACES** is set to **OFF**: Specifies the format, default page table, and default translation for one or both user mode zones (access class HU: or GU:). Can be combined with the **Hypervisormode** or **Guestmode** option. If **SYStem.Option.MACHINESPACES** is set to **ON**: Specifies the format, default page table, and default translation for the user mode zone of the machine which is selected with the **MACHINE** option.
Hypervisormode	Specifies the format, default page table, and default translation for one or both hypervisor zones (access class HS: or HU:). Can be combined with the **Supervisormode** or **Usermode** option. For an example, see below.
Guestmode	Specifies the format, default page table, and default translation for one or both guest zones (access class GS: or GU:). Can be combined with the **Supervisormode** or **Usermode** option.
MACHINE	For a description of the **MACHINE** option, see **MMU.DUMP**.

If both **SYStem.Option ZoneSPACES** and **SYStem.Option MACHINESPACES** are set to **ON**, then these options are available:

- **MACHINE**
- **Supervisormode**
- **Usermode**

If only **SYStem.Option ZoneSPACES** is set to **ON**, then these options are available:

- **Hypervisormode**
- **Guestmode**
- **Supervisormode**
- **Usermode**

If only **SYStem.Option MACHINESPACES** is set to **ON**, then these option is available:

- **MACHINE**

Examples for Page Tables in Virtualized Systems

[Back to MMU.FORMAT]

NOTE:
- The MMU format and default page table base address of each zone can be viewed with the command **TRANSlation.state**.
- The default translation of each zone can be viewed with the command **TRANSlation.List**

Example 1: This script shows how to define separate default page tables and separate default translations for various zones (*without* Hypervisor Awareness and *without* machine IDs).
The backslash \ is used as a line continuation character. No white space permitted after the backslash.

```
; enable symbol management for zones
SYStem.Option.ZoneSPACES ON

; define the format for the hypervisor-supervisor zone (access class HS:)
MMU.FORMAT STD HS:0xC8000000 HS:0x80000000++0x0FFFFFFF                  \
                           A:0x00000000 /Hypervisormode /Supervisormode

; define the format for the hypervisor-user zone (access class HU:)
MMU.FORMAT STD HU:0x34000000 HU:0x30000000++0x0FFFFFFF                  \
                           A:0x00800000 /Hypervisormode /Usermode

; define the format for guest-supervisor zone (access class GS:)
MMU.FORMAT VX653 GS:0xA4000000 GS:0xA0000000++0x0FFFFFFF                \
                           A:0x10000000 /Guestmode  /Supervisormode

; define the format for guest-user zone (access class GU:)
MMU.FORMAT VX653 GU:0x22000000 GU:0x20000000++0x1FFFFFFF                \
                           A:0x18000000 /Guestmode  /Usermode

; show the result of the format definition
TRANSlation.state
TRANSlation.List
```

Example 2: This script shows how to define separate default page tables and separate default translations for various zones (**with** Hypervisor Awareness and **with** machine IDs).
The backslash \ is used as a line continuation character. No white space permitted after the backslash.

```
; enable symbol management for zones
SYStem.Option.ZoneSPACES ON

; enable address extension for guest OSes
SYStem.Option.MACHINESPACES ON

; define the format for the supervisor zone of machine 0
; (access class HS:)
MMU.FORMAT STD  HS:0:::0xC8000000 HS:0:::0x80000000++0x0FFFFFFF        \
                                  A:0x00000000 /MACHINE 0 /Supervisormode

; define the format for the supervisor zone of machine 1
; (access class GS:)
MMU.FORMAT STD  GS:1:::0xA4000000 GS:1:::0xA0000000++0x0FFFFFFF        \
                                  A:0x10000000 /MACHINE 1 /Supervisormode

; define the format for the guest-user zone of machine 1
; (access class GU:)
MMU.FORMAT VX653 GU:1:::0x22000000 GU:1:::0x20000000++0x1FFFFFFF       \
                                  A:0x18000000 /MACHINE 1 /Usermode

; define the format for both the guest-supervisor zone and the guest-user
; zone of machine 2 concurrently (access class G:)
MMU.FORMAT VX653 G:0xB8000000 G:0xB0000000++0x1FFFFFFF                 \
                                  A:0x40000000 /MACHINE 2

; show the result of the format definition
TRANSlation.state
TRANSlation.List
```

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] [**/***<option>*] **MMU.***<table>***.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

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 or a specific machine if a space ID and/or a machine ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID and/or machine ID: list the translation table of the specified process and/or machine • 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**.
MACHINE	For a description of the **MACHINE** option, see **MMU.DUMP**.

CPU specific Tables in MMU.List <table>

TLB1PT *<tlb1_index>*	Displays the indirect page table which is associated with the TLB1 entry *<tlb1_index>*

Additionally, if both **SYStem.Option ZoneSPACES** and **SYStem.Option MACHINESPACES** are set to **ON**, then these CPU specific tables are available:

SupervisorPT	Displays the supervisor mode page table of the machine specified with option **MACHINE**.
UserPT	Displays the user mode page table of the machine specified with the option **MACHINE**.

Additionally, if only **SYStem.Option ZoneSPACES** is set to **ON**, then these CPU specific tables are available:

HSPageTable	Displays the page table which is defined for the hypervisor supervisor mode.
HUPageTable	Displays the page table which is defined for the hypervisor user mode.
GSPageTable	Displays the page table which is defined for the guest supervisor mode.
GUPageTable	Displays the page table which is defined for the guest user mode.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] [*/<option>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** *<cpu_specific_tables>*
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

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 and/or machine ID: loads the translation table of the specified process and/or machine • 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	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**.
MACHINE	For a description of the **MACHINE** option, see **MMU.DUMP**.

CPU specific Tables in MMU.SCAN <table>

TLB0	Loads the TLB0 from the CPU to the debugger-internal translation table.
TLB1	Loads the TLB1 from the CPU to the debugger-internal translation table.

Additionally, if both **SYStem.Option ZoneSPACES** and **SYStem.Option MACHINESPACES** are set to **ON**, then these CPU specific tables are available:

SupervisorPT	Displays the supervisor mode page table of the machine specified with option **MACHINE**.
UserPT	Displays the user mode page table of the machine specified with the option **MACHINE**.

Additonally, if only **SYStem.Option ZoneSPACES** is set to **ON**, then these CPU specific tables are available:

HSPageTable	Displays the page table which is defined for the hypervisor supervisor mode.
HUPageTable	Displays the page table which is defined for the hypervisor user mode.
GSPageTable	Displays the page table which is defined for the guest supervisor mode.
GUPageTable	Displays the page table which is defined for the guest user mode.

Formats:	**MMU.Set TLB0** <*index*> <*mas1*> <*mas2*> <*mas3*> <*mas7*> <*mas8*> **MMU.Set TLB1** <*index*> <*mas1*> <*mas2*> <*mas3*> <*mas7*> <*mas8*> **MMU.**<*table*>**.SET** (deprecated)

Sets the specified MMU TLB table entry in the CPU. The parameter <*tlb*> is not available for CPUs with only one TLB table.

<*index*>	TLB entry index. From 0 to (number of TLB entries)-1 of the specified TLB table
<*mas1*> <*mas2*> <*mas3*> <*mas7*> <*mas8*>	Values corresponding to the values that would be written to the MAS registers in order to set a TLB entry. See the processor's reference manual for details on MAS registers.

CPU specific BenchMarkCounter Commands

The BenchMarkCounter features are based on the core's performance monitor, accessed through the performance monitor registers (PMR).

NOTE:
- Chips with e6500 cores provide PMR access while the core is running. Otherwise, PMR access is only possible while the core is halted.
- 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.
- For a description of events that can be assigned to **BMC.<counter>.EVENT <event>**, please check Freescale's core reference manual.

BMC.FREEZE — Freeze counters while core halted

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

Default: ON.

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.

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

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

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

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

GUEST Freeze counters in guest state
0 The PMC is incremented (if permitted by other PM control bits).
1 The PMC is not incremented if MSR[GS] = 1.

HYPERVISOR Freeze counters in hypervisor state
0 The PMC is incremented (if permitted by other PM control bits).
1 The PMC is not incremented if MSR[GS] = 0.

TrOnchip.CONVert — Adjust range breakpoint in on-chip resource

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

Default: ON.

This command influences the behavior when there are no more on-chip resources for exact **data address range breakpoints** available. The QorIQ processors offer the possibility to set one exact data address range breakpoint or up to two data address range breakpoints with a maximum of up to 4kB each. These ranges are not exact in all cases, depending on the maskable start and end address. Using this command the user can allow or prohibit a conversion to ranges which exceed the exact one in cases it is necessary. In all other cases the exact range setting will be preferred.

ON	Data address range breakpoints which do not exceed 4kB can be converted to ranges which exceed the exact range. This offers the possibility to use up to two data address range breakpoints with maximum 4kB each instead of just one exact data address range breakpoint. It is also possible to use one data address breakpoint in combination with one (non-exact) max. 4kB large data address range breakpoint. Please be aware, that the range breakpoint is still listed as the original, exact range breakpoint in the **Break.List** window. Use the **Data.View** command to verify the extended data address range breakpoints.
OFF	When there is already a data address breakpoint set, an error message is displayed when the user wants to set a new data address range breakpoint. When there is already a data address range breakpoint set, an error message is displayed when the user wants to set a new data address breakpoint or data address range breakpoint.

Example:

```
TrOnchip.CONVert ON                    ;Enable conversion of data ranges
Break.Set 0x6020++0x1f /Write          ;First range, <4kB
Break.Set 0x7024++0x1f /Write          ;Second range, <4kB
Data.View 0x6020                       ;First range, exact conversion
Data.View 0x7000                       ;Second range extended to
                                       ;0x7000--0x707F because of address
                                       masking.

Break.RESet                            ;Start example without CONVert
TrOnchip.CONVert OFF                   ;No conversion allowed
Break.Set 0x6020++0x1f /Write          ;First range, exact range set
Break.Set 0x7024++0x1f /Write          ;Second rang won't be set, an error
                                       message is displayed.
```

TrOnchip.RESet — Reset on-chip trigger settings

Format: **TrOnchip.RESet**

Resets the on-chip trigger system to the default state.

TrOnchip.Set — Enable special on-chip breakpoints

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

<event>:
BRT
IRPT
RET
CIRPT
CRET

Default: All events OFF.

Enables the specified on-chip trigger facility to stop the CPU on the following break events:

BRT	Break on branch taken event.
IRPT	Break on interrupt entry.

RET	Break on return from interrupt.
CIRPT	Break on critical interrupt entry.
CRET	Break on return from critical interrupt.

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

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

Default: ON.

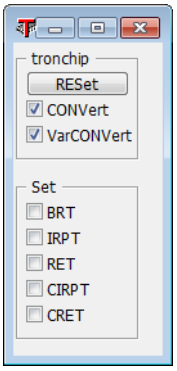
ON	After a data address breakpoint is set to an HLL variable all on-chip breakpoints are spent to cover all the bytes of the variable as a range. As soon as a new data address breakpoint is set the data address breakpoint to the HLL variable is converted to a single data address breakpoint.
OFF	An error message is displayed when the user wants to set a new data address breakpoint after all on-chip breakpoints are spent by a data address breakpoint to an HLL variable.

Example:

```
TrOnchip.VarCONVert ON
Var.Break.Set static_int1
TrOnchip.VarCONVert OFF
Var.Break.Set static_int2
Break.List                              ;byte address for static_int1
                                        ;address range for static_int2
```

Format:	**TrOnchip.state**

Display the trigger setup dialog window.



DDRTrace.List — List DDR trace contents

Format:	**DDRTrace.List** [*<items>*…]
<items>:	[**DEFault** | **DDR** | **DDRSYNC** | **DDRMID** | **DDRSIZE** | …]

Opens a window showing the recorded DDR trace data.

<items>	The *<items>* define the columns to be displayed in the DDR trace listing. You can combine any of the available *<items>*. The availability of *<items>* is dependent on the CPU and even its revision.
DEFault	By default, just the basic information is displayed, which includes the address and read / write information.
DDR	Displays all details included in the DDR trace message. All other *<items>* display only a subset of the DDR trace message.

For information about how to access other trace listings, see chapter **Trace Sources**.

DQMTrace.List — List DQM trace contents

Format:	**DQMTrace.List** [*<items>*…]
<items>:	[**DEFault** | **Address** | **CYcle** | **Data** | …]

Opens a window showing the recorded DQM (data acquisition message) trace data.

<items>	The *<items>* define the columns to be displayed in the DQM trace listing. You can combine any of the available *<items>*.
DEFault	By default, all DQM trace message included information is displayed, which includes the **DEVENT** register value in the Address column and the **DDAM** register value in the Data column.

For information about how to access other trace listings, see chapter **Trace Sources**.

Format:	**NEXUS.BTM** [**ON** | **OFF**]

Default: ON.

Global control for Nexus program trace messaging.

ON	Program trace messaging enabled.
OFF	Program trace messaging disabled.

For core specific trace control, please see the **NEXUS.CoreENable** command.

NEXUS.CoreENable Core specific trace configuration

Format:	**NEXUS.CoreENable** [<*core_numbers*>]

Access to core specific trace configuration.

Default: All cores of the CPU are enabled and the program trace is just managed by the global setting of **NEXUS.BTM**. For e.g. a CPU with eight cores the default <**core numbers**> setting is <**0,1,2,3,4,5,6,7**>.

To disable the generation of trace messages for specific cores exclude them from the <**core numbers**> list.

Format: **NEXUS.DDRConfig.ADDRessfilter***<index>* [*<address>* | *<range>* | *<bitmask>*]

Default: inactive.

Activates filters to restrict the generation of DDR trace messages.

<index>	1, 2 There are two filters available, valid for all DDR memory controllers.
<address> *<range>* *<bitmask>*	Restrict the DDR trace message generation to the specified physical address, range or bitmask.
without the optional arguments: *<address>*, *<range>*, *<bitmask>*	Apply this command without an address, range or bitmask to all filters to enable Nexus DDR messages for every address (if a mode is selected by **NEXUS.DDRConfig.Controller**).

NEXUS.DDRConfig.Controller Configure Nexus DDR message type

Format: **NEXUS.DDRConfig.Controller***<index>* [**Terse** | **Verbose** | **OFF**]

Default: OFF.

Enables Nexus DDR trace and configures the type of Nexus DDR trace messages.

<index>	1, 2, 3 The index specifies the DDR memory controller. The number of DDR memory controllers varies depending on the QorIQ CPU.
OFF	DDR trace deactivated for this memory controller.
Terse **Verbose**	Terse trace messages need less bandwidth but do not offer as detailed information as the verbose trace messages. Please refer to the appropriate reference manual of the CPU for further details.

Format: **NEXUS.DQM** [**ON** | **OFF**]

Default: OFF.

Set to ON to enable data acquisition messaging.

NOTE: When instrumented software uses the **DEVENT** and **DDAM** registers, the corresponding **IDTAG** and **DQDATA** values are transmitted within a data acquisition message.
This message is produced at the time a core writes to the **DDAM** register using a **mtspr** instruction.

POWER TRACE SERIAL only

Using the **NEXUS.LaneMapping** command group, you can configure the mapping of logical to physical Aurora trace lanes.

NOTE:	This configuration is typically needed if the routing of the target lanes to the trace port is disordered (which is rarely the case).

NEXUS.LaneMapping APPLY Apply logical to physical lane mapping

POWER TRACE SERIAL only

Format: **NEXUS.LaneMapping APPLY**

Apply the current mapping of logical to physical Aurora trace lanes, defined by **NEXUS.LaneMapping.SetLane**.

NEXUS.LaneMapping.SetLane Configure logical to physical lane mapping

POWER TRACE SERIAL only

Format: **NEXUS.LaneMapping.SetLane** *<logical_id> <physical_id>*

Default: 1:1 configuration.

Maps the logical Aurora trace lanes to the physical ones. This command takes effect only if **NEXUS.LaneMapping APPLY** is used afterwards.

Format:	**NEXUS.OCeaNport**<*index*>**.Mode** [**Terse** | **Verbose**]

Default: Terse.

Configure the type of Nexus OCeaN messages.

<*index*>	1, 2 The index specifies the OCeaN port.
Terse **Verbose**	Terse trace messages offer less detailed address information than the verbose trace messages, but information about the source. Please refer to the appropriate reference manual of the CPU for further details.

Format: **NEXUS.OCeaNport1.TraceSELect** <*source*>

<*source*>:
OFF
P0OUT-CHB
P1OUT-PCIE1
P1OUT
P2OUT-PCIE2SRIO1
P2OUT
P3OUT-PCIE3
P3OUT
P4OUT-SRIO2
P4OUT

Default: OFF.

Format: **NEXUS.OCeaNport2.TraceSELect** <*source*>

<*source*>:
OFF
P0IN-CHB
P1IN-PCIE1
P1IN
P2IN-PCIE2SRIO1
P2IN
P3IN-PCIE3
P3IN
P4IN-SRIO2
P4IN

Default: OFF.

Select the sources which produce Nexus OCeaN messages.

NEXUS.OFF Switch the Nexus trace port off

Format: **NEXUS.OFF**

Default: ON.

Turn off if you neither want to use the Onchip nor the HSTP trace.

Format:	**NEXUS.ON**

The Nexus trace port is switched on. All trace registers are configured by the debugger.

NEXUS.OTM Enable ownership trace messaging

Format:	**NEXUS.OTM** [**ON** | **PID0** | **NPIDR** | **OFF**]

Default: OFF.

Controls ownership trace messaging.

OFF	Ownership trace messaging is disabled.
ON **PID0**	Enable ownership trace messaging. An OTM is generated if the application writes to the PID0 register.
NPIDR	Enable ownership trace messaging. An OTM is generated if the application writes to the NPIDR register.

NOTE:	Enable ownership trace messaging in order to get trace information about task switches. Some operating systems use a set of OTMs to transfer task switch information to the trace tool. In this case periodic ownership trace must be disabled using **NEXUS.POTD ON**.

Format:	**NEXUS.PortMode** *<mode>*
<mode>:	**625MBPS** | **750MBPS** | **850MBPS** | **1000MBPS** | **1250MBPS** | **1500MBPS** | **1700MBPS** | **2000MBPS** | **2500MBPS** | **3000MBPS** | **3125MBPS** | **3400MBPS** | **4000MBPS** | **4250MBPS** | **5000MBPS** | **6000MBPS** | **6250MBPS**

Sets the Nexus trace port frequency. For Aurora Nexus, the setting is a fixed bit rate which is independent of the system frequency.

NOTES: Depending on the processor, bit rates may be unsupported. Set the bit rate according to the processor's data sheet. You will get a warning if a set bit rate is not supported by your processor.

QorIQ processors usually do not need an external reference clock for Aurora operation. Nevertheless if needed, the Aurora preprocessor can provide that clock signal. It is enabled using **NEXUS.RefClock ON**.

NEXUS.PortSize Set trace port width

Format:	**NEXUS.PortSize** *<port_size>*
<port_size>:	**1Lane** | **2Lane**

Default: Varied, depending on the processor.

Sets the Nexus port width to the number of used Aurora lanes. The setting can only be changed if no debug session is active (**SYStem.Down**).

NOTE: Depending on the processor there are no dedicated debug lanes available. In this case you need to set an appropriate RCW (SRDS_PRTCL field) to configure one or more lanes for debugging purposes. Please refer to **Programming the Reset Configuration Word** for further details.

Format: **NEXUS.POTD** [**ON** | **OFF**]

Default: OFF.

When enabled, the core is configured to suppress periodic ownership trace messages. A periodic ownership trace message is an OTM, which is generated without a write access to the PID register once every 256 messages. Enable this option, when the OTM is used to generate trace information about task switches.

NEXUS.PTCM Enable program trace correlation messages

Format: **NEXUS.PTCM.**<*event*> [**ON** | **OFF**]

<*event*>: **BL_HTM**

Default: OFF.

Enables a program trace correlation message (PTCM) for the specified event. These program trace correlation messages are not needed to reconstruct the program flow, but give additional information which can increase the precision of statistic measurements.

BL_HTM	Core generates PTCM on Branch and Link occurrence (EVCODE 0xA). Enable this PTCM to improve function profiling.

NEXUS.PTFGS Program trace mark

Format: **NEXUS.PTFGS** [**OFF** | **GS0** | **GS1**]

Default: OFF.

Controls the influence of MSR[GS] in program trace messaging. Only available for e6500.

OFF	Ignore MSR[GS] for masking program trace messages.
GS0	Generate program trace messages only when MSR[GS] = 0
GS1	Generate program trace messages only when MSR[GS] = 1

Format: **NEXUS.PTFPMM** [**OFF** | **PMM0** | **PMM1**]

Default: OFF.

Controls the influence of MSR[PMM] in program trace messaging. Only available for e6500.

OFF	Ignore MSR[PMM] for masking program trace messages.
PMM0	Generate program trace messages only when MSR[PMM] = 0
PMM1	Generate program trace messages only when MSR[PMM] = 1

NEXUS.PTFPR Program trace mark

Format: **NEXUS.PTFPR** [**OFF** | **PR0** | **PR1**]

Default: OFF.

Controls the influence of MSR[PR] in program trace messaging. Only available for e6500.

OFF	Ignore MSR[PR] for masking program trace messages.
PR0	Generate program trace messages only when MSR[PR] = 0
PR1	Generate program trace messages only when MSR[PR] = 1

Format: **NEXUS.PTMARK** [**ON** | **OFF**]

Default: OFF.

Controls the influence of MSR[PMM] in program trace messaging.

OFF	Ignore MSR[PMM] for masking program trace messages.
ON	Mask (disable) program trace messages when MSR[PMM] = 0, unmask (enable) program trace messages when MSR[PMM] = 1

NEXUS.RefClock Enable Aurora reference clock

Format: **NEXUS.RefClock** [**ON** | **OFF**]

Default: OFF.

Typically this settings should not be changed for QorIQ processors.

When set to ON, the preprocessor provides the reference clock for the Aurora Nexus block on the processor. Only enable when the processor requires this reference clock and when no module provides the Aurora clock source for the processor.

NEXUS.Register Display NEXUS trace control registers

Format: **NEXUS.Register**

This command opens a window which shows the NEXUS configuration and status registers.

Format:	**NEXUS.RESet**

Resets Nexus trace port settings to default settings.

NEXUS.SerDesCFG Enable SerDes PLL control register manipulation

Format:	**NEXUS.SerDesCFG** [**ON** | **OFF**]

Default: ON.

Enables the SerDes PLL control register manipulation.

ON	The SerDes PLL control register will be modified according to the **NEXUS.SerDesCFG.REFCLK** and **NEXUS.SerDesCFG.FRATE** settings.
OFF	The SerDes PLL control register is not touched.

NEXUS.SerDesCFG.FRATE Select frequency of SerDes PLL VCO

Format:	**NEXUS.SerDesCFG.FRATE** *<mode>*
<mode>:	**DEFault** | **3GHz** | **3.125GHz** | **4GHz** | **5GHz** | **6GHz** | **6.25GHz**

Default: DEFault.

Sets the FRATE field of the SerDes PLL control register. This value sets the frequency of PLL VCO as described in the reference manual of the CPU.

<mode>	Sets the appropriate FRATE field value of the SerDes PLL control register according to the CPU reference manual. The available *<mode>* values are dependent on the CPU.
DEFault	This setting uses values compatible with the corresponding Freescale evaluation boards.

NOTE:	The value of the **NEXUS.SerDesCFG.FRATE** (or the corresponding bit field in the SerDes PLL control register) must be divisible by the **NEXUS.PortMode** setting, otherwise the Aurora channel won't come up because of different lane frequencies of the transmitter and the receiver. In this case a warning will be displayed in the **message line** and the **AREA.view** window.

NEXUS.SerDesCFG.REFCLK Select frequency of SerDes reference clock

Format:	**NEXUS.SerDesCFG.REFCLK** *<mode>*
<mode>:	**DEFault** | **100MHz** | **125MHz** | **150MHz** | **156.25MHz** | **161.13MHz**

Sets the RFCLK field of the SerDes PLL control register. This value selects the SerDes reference clock frequency as described in the CPU reference manual.

DEFault	This setting uses values compatible with the corresponding Freescale evaluation boards.
<mode>	Sets the appropriate RFCLK field value of the SerDes PLL control register according to the CPU reference manual. The available *<mode>* values are dependent on the CPU.

NOTE:	The value of the **NEXUS.SerDesCFG.REFCLK** (or the corresponding bit field in the SerDes PLL control register) must be adjusted to your board settings, otherwise the Aurora channel won't come up because the PLL won't lock. In this case a warning will be displayed in the **message line** and the **AREA.view** window.

NEXUS.Spen<messagetype> Enable message suppression

Format:	**NEXUS.SpenDQM** [**ON** | **OFF**] **NEXUS.SpenWTM** [**ON** | **OFF**] **NEXUS.SpenPTM** [**ON** | **OFF**] **NEXUS.SpenDTM** [**ON** | **OFF**] **NEXUS.SpenOTM** [**ON** | **OFF**]

Default: OFF.

Configures the core to suppress one or more message types (DQM, WTM, PTM, DTM and OTM) when the on-chip Nexus message FIFO reaches a certain fill level. Enabling one of these options will in most cases cause problems in trace analysis, because the trace message stream contains no information about if and when messages have been suppressed. The fill level at which message suppression occurs can be configured via the command **NEXUS.SupprTHReshold**.

NEXUS.STALL Stall the program execution when FIFO level is reached

Format: **NEXUS.STALL** [**OFF** | **1/4** | **1/2** | **3/4**]

Default: OFF.

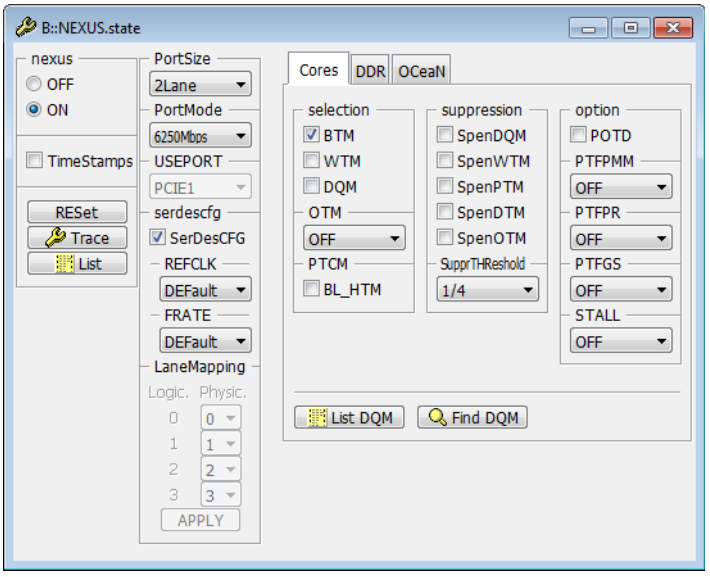
Stall the program execution whenever the configured on-chip Nexus-FIFO (internal buffer) fill level is reached. If this option is enabled, the Nexus port controller (NPC) will stop the core's execution pipeline if the set fill level of the buffer is reached, e.g. fill level 1/2.
In the meantime, the NPC sends the messages of the buffer to the defined trace sink. The NPC will start the core's execution pipeline again if the next lower fill level of the buffer is reached, e.g. fill level 1/4.

Enabling this command will affect (delay) the instruction execution timing of the CPU. This system option, which is a representation of a feature of the processor, will remarkably reduce the amount FIFO OVERFLOW errors, but can not avoid them completely.

Format: **NEXUS.state** [*/<tab>*]

<tab>: **Cores** | **DDR** | **OCeaN**

Displays the Nexus trace port configuration window.



<tab> Opens the **NEXUS.state** window on the specified tab.

NEXUS.SupprTHReshold Set fill level for message suppression

Format: **NEXUS.SupprTHReshold** [**1/4** | **1/2** | **3/4**]

Sets the Nexus message FIFO fill level, at which messages will be suppressed by the core. The message types which will be suppressed are configured via the **NEXUS.Suppr<message>** command.

Format:	**NEXUS.TimeStamps** [**ON** | **OFF**]

Default: OFF.

ON	All Nexus messages will be extended with a target counter timestamp. Use this option to increase the timestamp accuracy.
OFF	No timestamp will be appended to the Nexus messages. Use this option to save bandwidth.

NOTE:	If you uses the external Aurora HSTP, all messages will get a timestamp from the serial preprocessor, independent of this option.

NEXUS.USEPORT Define used PCIe controller for PCIe trace

Format:	**NEXUS.USEPORT** [**PCIE1** | **PCIE2** | **PCIE3** | **PCIE4**]

Default: PCIE1.

Defines the PCIe controller that will be used for PCIe tracing (see **Nexus PCIe Trace**).

NEXUS.WTM Enable watchpoint messaging

Format:	**NEXUS.WTM** [**ON** | **OFF**]

Default: OFF.

ON	Nexus outputs watchpoint messages.
OFF	No watchpoint messages are output by Nexus.

NOTE:	When a watchpoint is set via a **Break.Set** command, the **NEXUS.WTM** setting will be internally overridden to ON.

Format:	**OCeaNTrace.List** [*<items>*...]
<items>:	[**DEFault** | **OCeaN** | **OCeaNSYNC** | **OCeaNMID** | ...]

Opens a window showing the recorded OCeaN trace data.

<items>	The *<items>* define the columns to be displayed in the OCeaN trace listing. You can combine any of the available *<items>*.
DEFault	By default, just the basic information is displayed, which includes the address, the transmitted data and the message type including the source port.
OCeaN	Displays all details included in the OCeaN trace message. All other *<items>* display only a subset of the OCeaN trace message.

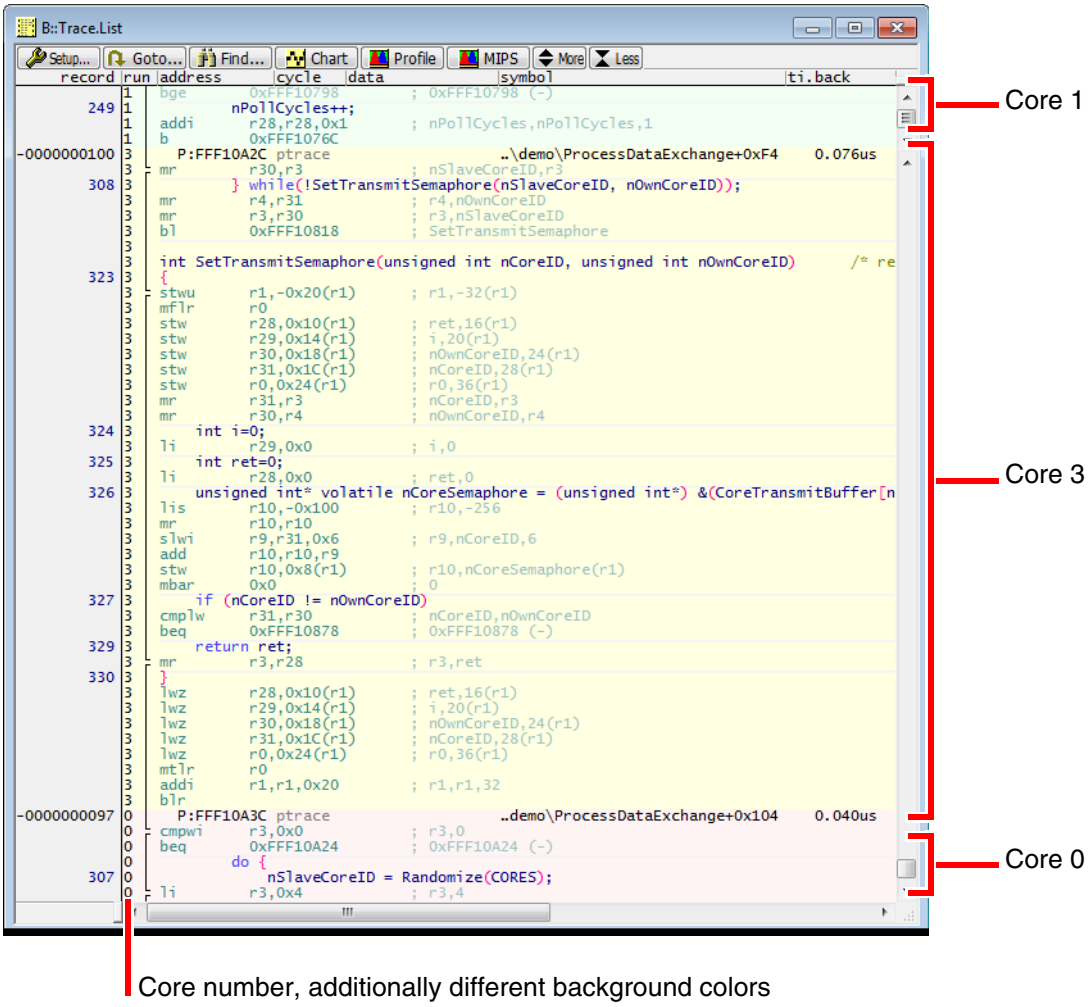
For information about how to access other trace listings, see chapter **Trace Sources**.

Format:	**Trace.List** [*<items>*…]
<items>:	[**DEFault** | **Address** | **CYcle** | **sYmbol** | …]

Opens a window showing the recorded program and data trace contents.

<items>	The *<items>* define the columns to be displayed in the program and data trace listing. You can combine any of the available *<items>*.
DEFault	By default, the decoded program and data trace information is displayed.

For information about how to access other trace listings, see chapter **Trace Sources**.



Onchip.TBARange — Configure on-chip trace base address range

Format: **Onchip.TBARange** *<address_range>*
Onchip.TBAddress *<address_range>* (deprecated)

Define the address range for the onchip trace buffer. The address range is always based on physical addresses and thus is not dependent on the MMU, but on the LAW settings. The user-defined address range will always be adapted to correct aligned 64byte block size.

NOTE: Setting a TBARange includes a fast read-write validation of this memory. A warning will be displayed and the TBARange is reset if this memory access fails.
The maximum onchip trace size for QorIQ processors is restricted to 512MB (exact (2^29)-1 byte).

Filters and Triggers for the Nexus Trace

This section describes filters and triggers provided by the processor.

The internal watchpoints of the QorIQ processors can be used to control the output of the trace data. The following actions for the Nexus trace are provided through the **Break.Set** command:

Actions for the Trace (provided by the CPU)

TraceEnable	Configure the trace source to only generate a trace message if the specified event occurs. Complete program flow or data trace is disabled. If more than one **TraceEnable** action is set, all **TraceEnable** actions will generate a trace message.
TraceData	Use this action to configure data access trace messaging to an address range or single address. The QorIQ processors offer just the possibility to trace write accesses. In order to use the **TraceData** action it has to be combined with **/Write**. Please also note the data trace restrictions in **Supported Trace Features**. There is no need to enable **NEXUS.BTM** to use the data trace.
TraceON **TraceOFF**	If the specified event occurs, program and data trace messaging is started (**TraceON**) or ends (**TraceOFF**). In order to perform event based trace start/end to program trace and data trace separately, use Alpha-Echo actions.
WATCH	Set a watchpoint on the event. The CPU will trigger the EVTO pin if the event occurs and generate a watchpoint hit message if the trace port is enabled.

Examples for exclusive selective tracing. **TraceEnable** enables tracing exclusively for the selected events. All other program and data trace messaging is disabled.

```
;Only generate a trace message when the instruction
;at address 0x00008230 is executed.
  Break.Set 0x00008230 /Program /TraceEnable

;Only generate a trace message when the core writes to variable flags[3].
  Var.Break.Set flags[3] /Write /TraceEnable
```

Examples for data trace messaging (**TraceData**):

```
;Enable data trace for write accesses for one specific address
  Break.Set 0x10000000 /Write /TraceData
;Enable data trace for the maximum address range (8kB, consisting of
2x4kB ranges. Possible only if no other onchip DAC resources are used.)
  Break.Set 0x10000000--0x10001FFF /Write /TraceData
;Enable data trace for write accesses to the array flags
  Var.Break.Set flags /Write /TraceData
```

Examples to turn on/off trace recording based on debug/trace events. **TraceON/TraceOFF** control program and data trace depending on NEXUS.BTM/DTM setting:

```
;Enable program/data trace when func2 is entered
;Disable program/data trace when last instruction of func2 is executed.
  Break.Set sYmbol.BEGIN(func2)          /Program /TraceON
  Break.Set sYmbol.END(func2)&0xFFFFFFFC /Program /TraceOFF

;Enable program/data trace when variable flags[3] is written
  Var.Break.Set flags[3] /Write /TraceON

;Disable program/data trace data when 16-bit value 0x1122 is
;written to address 0x40000230
  Break.Set 0x40000230 /Write /Data.Word 0x1122

;Enable program/data trace only when a specific task is active
;NOTE: RTOS support must be set up correctly
  &magic=0x40001280     ;set &magic to the task of interest
  Break.Set task.config(magic) /Write /Data  &magic /TraceON
  Break.Set task.config(magic) /Write /Data !&magic /TraceOFF
```

Mechanical Description

JTAG Connector QorIQ (COP)

Signal	Pin	Pin	Signal
TDO	1	2	N/C
TDI	3	4	TRST-
(RUNSTOP-)	5	6	JTAG-VREF
TCK	7	8	(CHKSTPIN-)
TMS	9	10	N/C
(SRESET-)	11	12	GND
PORESET-	13	14	N/C (KEY PIN)
(CKSTOPOUT-)	15	16	GND

This is a standard 16 pin double row (two rows of eight pins) connector (pin-to-pin spacing: 0.100 in.). (Signals in brackets are not necessary for basic debugging, but it is recommended to take them into consideration for future designs.)

Samtec22 (Power.org)

Signal	Pin	Pin	Signal
TXP0	1	2	JTAG-VREF
TXN0	3	4	TCK
GND	5	6	TMS
TXP1	7	8	TDI
TXN1	9	10	TDO
GND	11	12	TRST-
TXP2	13	14	(VENDOR-IO0)
TXN2	15	16	(VENDOR-IO1)
GND	17	18	(VENDOR-IO2)
TXP3	19	20	(VENDOR-IO3)
TXN3	21	22	PORESET-

Samtec46 (Power.org)

Signal	Pin	Pin	Signal
TXP0	1	2	JTAG-VREF
TXN0	3	4	TCK
GND	5	6	TMS
TXP1	7	8	TDI
TXN1	9	10	TDO
GND	11	12	TRST-
TXP2	13	14	(VENDOR-IO0)
TXN2	15	16	(VENDOR-IO1)
GND	17	18	(VENDOR-IO2)
TXP3	19	20	(VENDOR-IO3)
TXN3	21	22	PORESET-
GND	23	24	GND
(TXP4)	25	26	(CLKP)
(TXN4)	27	28	(CLKN)
GND	29	30	GND
(TXP5)	31	32	(VENDOR-IO4)
(TXN5)	33	34	(VENDOR-IO5)
GND	35	36	GND
(TXP6)	37	38	N/C
(TXN6)	39	40	N/C
GND	41	42	GND
(TXP7)	43	44	N/C
(TXN7)	45	46	N/C

Signal	Pin	Pin	Signal
TXP0	1	2	JTAG-VREF
TXN0	3	4	TCK
GND	5	6	TMS
TXP1	7	8	TDI
TXN1	9	10	TDO
GND	11	12	TRST-
(RXP0)	13	14	(VENDOR-IO0)
(RXN0)	15	16	(VENDOR-IO1)
GND	17	18	(VENDOR-IO2)
(RXP1)	19	20	(VENDOR-IO3)
(RXN1)	21	22	PORESET-
GND	23	24	GND
TXP2	25	26	(CLKP)
TXN2	27	28	(CLKN)
GND	29	30	GND
TXP3	31	32	(VENDOR-IO4)
TXN3	33	34	(VENDOR-IO5)
GND	35	36	GND
(RXP2)	37	38	N/C
(RXN2)	39	40	N/C
GND	41	42	GND
(RXP3)	43	44	N/C
(RXN3)	45	46	N/C
GND	47	48	GND
(TXP4)	49	50	N/C
(TXN4)	51	52	N/C
GND	53	54	GND
(TXP5)	55	56	N/C
(TXN5)	57	58	N/C
GND	59	60	GND
(TXP6)	61	62	N/C
(TXN6)	63	64	N/C
GND	65	66	GND
(TXP7)	67	68	N/C
(TXN7)	69	70	N/C