eTPU Debugger and Trace

eTPU Debugger and Trace

Version 18-Nov-2020

Introduction

This document describes the processor specific settings and features of TRACE32-ICD for the eTPU core.

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

MPC55XX	The debugger drives the output pins of the JTAG/OnCE 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.

ESD Protection

WARNING:	To prevent debugger and target from damage it is recommended to connect or disconnect the debug cable only while the target power is OFF. Recommendation for the software start: 1. Disconnect the debug cable from the target while the target power is off. 2. Connect the host system, the TRACE32 hardware and the debug cable. 3. Power ON the TRACE32 hardware. 4. Start the TRACE32 software to load the debugger firmware. 5. Connect the debug cable to the target. 6. Switch the target power ON. 7. Configure your debugger e.g. via a start-up script. Power down: 1. Switch off the target power. 2. Disconnect the debug cable from the target. 3. Close the TRACE32 software. 4. Power OFF the TRACE32 hardware.

General

- Locate the **JTAG/OnCE/Nexus connector** as close as possible to the processor to minimize the capacitive influence of the trace length and cross coupling of noise onto the BDM signals.
- Ensure that the debugger signal ($\overline{\text{HRESET}}$) is connected directly to the $\overline{\text{HRESET}}$ of the processor. This will provide the ability for the debugger to drive and sense the status of $\overline{\text{HRESET}}$. The target design should only drive the HRESET with open collector, open drain. $\overline{\text{HRESET}}$ should not be tied to $\overline{\text{PORESET}}$, because the debugger drives the HRESET and DSCK to enable BDM operation.

Quick Start eTPU Debugger

Starting up the debugger is done as follows.

NOTE: Debugger for e200 (or ColdFire) has to be started and configured first.

1. Set the CPU type to load the CPU specific settings.:

```
SYStem.CPU MPC5676R
```

2. Configure select target core. See **SYStem.CONFIG.CORE for details**.

```
SYStem.CONFIG.CORE <core_index>. 1.
```

3. Start debug session by attaching to the eTPU:

```
SYStem.Mode.Attach
```

4. Break eTPU and initialize program and data memory (optional).

```
Break
Data.Set P:0x0000--0x0BFF %Long 0xFFFFFFFF
Data.Set D:0x0000--0x02FF %Long 0xDEADDEAD
Go
```

5. Load the debug symbols. The program code will be usually loaded by the master core (Coldfire/PowerPC)

```
Data.LOAD.Elf app.elf /NoCODE /NoRegister
```

6. Set up breakpoint(s) and run master CPU afterwards:

```
Break.Set func_increment /Onchip
```

7. Or set a debug event on a service request:

```
TrOnchip.Set HSR ON
```

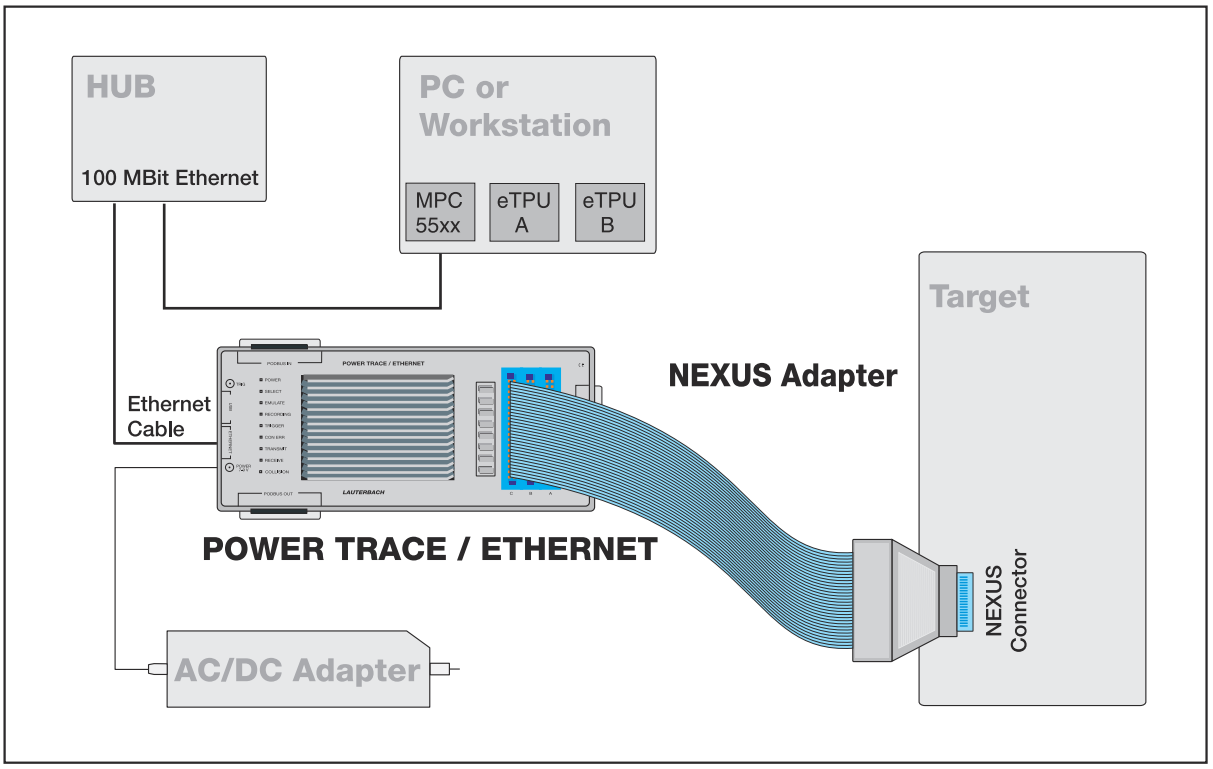
Troubleshooting

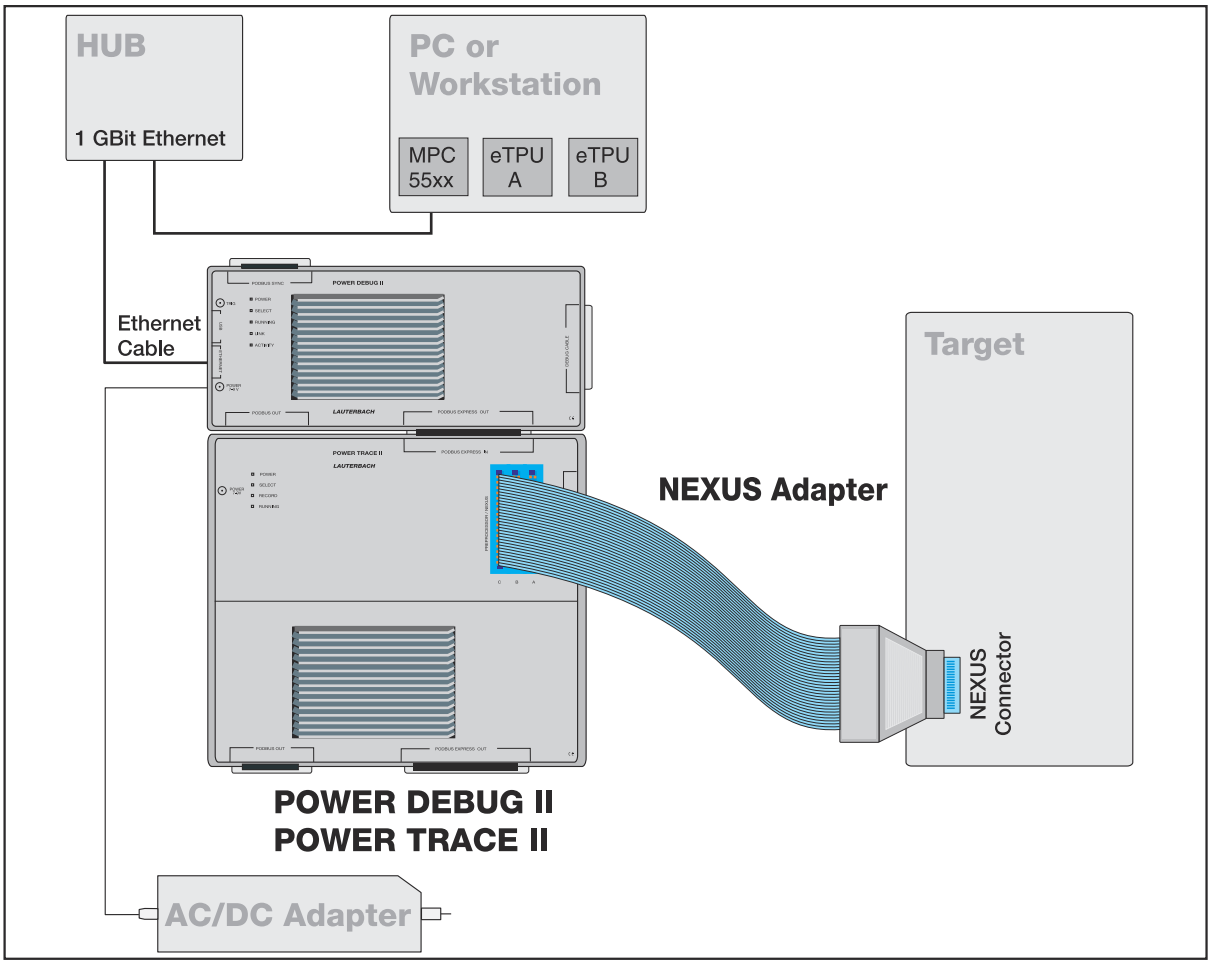
No information available.

FAQ

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

System Overview





eTPU operating modes

The eTPU is event driven. When no service request is pending, the eTPU is in IDLE mode. When the eTPU is halted by the debugger (command Break) while it is not processing a service request, the debugger will display IDLE in the status line.

When IDLE, the register set is invalid and will not be displayed in the **Register.view** window. There is also no valid MPC (microprogram counter, i.e. instruction pointer) in this state. Therefore **Data.List** will fail, but **Data.List** *<address>* or **Data.List** *<function/label>* can be always used.

Debugging the eTPU

Before the eTPU debug session can be started, the main core (e200 or ColdFire) has to initialize the eTPUs i.e. load the eTPU program and initiate a host service request. Make sure that the eTPUs are not halted for debugging, because this would prevent the main core from accessing the eTPUs.

This is the recommended method to start an eTPU debug session:

1. Program main application to FLASH using main core (if not already programmed)
2. Reset processor and begin debug session on main core (SYStem.Up)
3. Begin eTPU debug session using SYStem.Mode.Attach
4. Load debug symbols for all main and eTPU cores (Data.LOAD.Elf ....). Note: The eTPUs have their own debug symbols. The source file of the main core does *not* include debug symbols of the eTPUs.
5. To debug a service request or function, set a Breakpoint or enable a debug event on a service request. When another eTPU uses the same SCM, i is recommended to enable the “halt on twin engine” debug event on the other eTPU. See **TrOnchip.HTWIN**.
6. Make sure that all eTPU debuggers are in state running (green filed displaying “running” in the status bar)
7. Run application on main core (Go).
8. The main core will initialize the eTPUs. The eTPU should halt for the debugger when the set breakpoint or service request’s debug event occurs.

While an eTPU is halted for the debugger, it will not process any pending service requests. Some target applications wait for a response from an eTPU and in some cases it was seen that a halted eTPU caused a processor reset (e.g. by watchdog) in this case. Make sure that the main core’s application can handle a halted eTPU e.g. by deactivating the watchdog.

On processors with eTPU2, the eTPU watchdog must be disabled for debugging.

Breakpoints and Watchpoints

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

Software Breakpoints

The debugger will use software breakpoints as default. The debugger supports an unlimited number of software breakpoints.When using software breakpoints, MISC has to be disabled.

On a chip with more than one eTPU, the SCM (shared code memory) is only visible if both eTPUs connected to the same SCM (A and B) are stopped. Therefore, using software breakpoints is not supported in all cases, esp. if special break conditions (CBI, HTWIN) are enabled.

In systems with two eTPUs connected to the same SCM, software breakpoints will be visible for both cores. If both eTPUs run the same code, on-chip breakpoints should be used.

On-chip Breakpoints/Watchpoints

An eTPU has two on-chip break-/watchpoints. They can be used to

- generate a debug event (core halts for debugger)
- generate a watchpoint hit trace message
- enable/disable trace message generation when the event occurs.

The on-chip break-/watchpoints can be configured for

- instruction address comparison (instruction break/watchpoint)
- data address comparison (optional with data value comparison)

In addition, the break/watchpoints can be enabled for one channel, all channels or a certain set of channels. See **TrOnchip.SCM** for details.

Breakpoints/Watchpoints on Service Request or channel register write

The eTPU supports also debug events on service request starts and on channel register writes. See **TrOnchip.Set** for details. Like the on-chip break/watchpoints, they can be can be enabled for one channel, all channels or a certain set of channels.

The debugger uses watchpoints on service request starts and on channel register writes to generate a trigger signal (**TrOnchip.BusTrigger**) or to stop the trace recording (**TrOnchip.TraceTrigger**).

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program Memory (SCM)
D	Data Memory (SPRAM)
H	Memory space of the main core (HOST)

Address Spaces and Addressing Modes

The eTPU cores have an address space which is independent of the main core (e200 or ColdFire). Also program and data address space is separated (Harvard architecture).

In contrast to the main core, which uses byte addressing, the eTPU uses addresses its memory in 32-bit words. The following table shows some examples:

eTPU address	main core address
eTPU_**A/B** SCM **P:0x0000**	MPC5XXX: A:0xC3FD0000 ColdFire: IPSBAR + 0x1E0000
eTPU_**A/B** SCM **P:0x0001**	MPC5XXX: A:0xC3FD0004 ColdFire: IPSBAR + 0x1E00004
eTPU_**C** SCM **P:0x0000**	MPC5XXX: A:0xC3E30000
ETPU_**A/B** SPRAM **D:0x0000**	MPC5XXX: A:0xC3FC8000 ColdFire: IPSBAR + 0x1D8000
ETPU_**A/B** SPRAM **D:0x0001**	MPC5XXX: A:0xC3FC8004 ColdFire: IPSBAR + 0x1D8004
ETPU_**C** SPRAM **D:0x0000**	MPC5XXX: A:0xC3E28000

If two eTPUs share one SCM (e.g. eTPU_a and eTPU_B), SCM is only accessible if both eTPUs are stopped.

SYStem.CONFIG — Configure debugger according to target topology

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

For the description of **SYStem.CONFIG** commands, refer to the debugger manual for the main core in **SYStem.CONFIG** in **"Qorivva MPC5xxx/SPC5xx Debugger and NEXUS Trace"** (debugger_mpc5500.pdf). This setting is only available for CPUs with JTAG as debug port (not available for BDM).

SYStem.CONFIG.CORE — Assign core to TRACE32 instance

Format: **SYStem.CONFIG CORE** *<core_index>* [*<chip_index>*]
SYStem.MutiCore.Core *<core_index>* (deprecated)

This command is used to assign a specific core to a TRACE32 instance. Please make sure that the host debugger's CPU selection is appropriate before this command is called. If this command is called while a CPU without eTPU is selected, the command will fail. The valid parameters for <core-id> are given by debugger implementation:

Architecture / eTPU	Core-ID
MPC5XXX/SPC56XX with one e200 core	2 (eTPU_A), 3 (eTPU_B)
MPC5XXX with two e200 cores	3 (eTPU_A), 4 (eTPU_B), 5 (eTPU_C)
ColdFire	2

SYStem.CPU — Select the CPU type

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

<cpu>: **MPC5554** | **MCF5232** | …

Selects the CPU type.

Format: **SYStem.JtagClock** *<frequency>*
SYStem.BdmClock *<frequency>* (deprecated)

<frequency>: **1 000 000.** … **50 000 000.** (Default 4 MHz)

NOTE:
- If possible, use the same JTAG clock frequency for all cores debugged with the same debug interface.
- MPC55XX: the max. allowed JTAG clock frequency is 1/4th of the core frequency.

SYStem.LOCK Lock and tristate the debug port

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

Default: OFF.

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

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

<mode>: **Denied** | **Enable** | **NEXUS** | **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 memory class E: (e.g. Data.dump E:0x100) or by using the format option **%E** (e.g. **Var.View %E var1**). It is also possible to activate this non-intrusive memory access for all memory ranges displayed on the TRACE32 screen by setting **SYStem.Option DUALPORT ON**.

Denied Memory access is disabled while the CPU is executing code.

Enable The debugger performs memory accesses via a dedicated CPU interface.
CPU (deprecated)

NEXUS Memory access is done via the NEXUS interface. Available for MPC55XX/MPC56XX family, for both the NEXUS and JTAG-only debugger.

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

SYStem.Mode Select operation mode

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

<mode>: **Down** | **Attach**

Select target reset mode.

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

Attach	Establishes connection to the eTPU.
NoDebug **Go** **StandBy** **Up**	Not applicable for eTPU.

SYStem.Option ByteWise — Use byte addressing for eTPU memory space

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

The eTPU addresses data and code memory in 32-bit (words). In the default setting (OFF), the debugger addresses the eTPU memories also in 32-bit words.

There are however instructions, which can modify partitions of a 32-bit word (byte and 24-bit operations). In this case, it might be more convenient to address the eTPU memories in byte units.

Set this option to ON to configure the debugger to use byte addressing. This setting should only be changed before the debug session begins.

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

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

Forces all list, dump and view windows to use the memory 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.

SYStem.Option FreezeCLKS — Freeze eTPU clocks if eTPU halted

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

Stop TCR clocks. Controls whether the TCR clocks from the eTPU stop running when the eTPU is halted for the debugger.

SYStem.Option FreezePINS — Freeze pins if eTPU is halted

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

Stop pins in debug mode. Controls whether the eTPU pins are sampled when the eTPU is halted for the debugger. When set to ON, the pins are not sampled during debug mode. The pins are sampled during normal single steps.

Note: The following processors do **not** include a NEXUS trace module (tracing not possible):

- MPC563xM, SPC563M (Monaco)
- MPC564xA, SPC564A (Andorra)
- eTPU in ColdFire processors

NEXUS.BTM — Control for branch trace messages

Format: **NEXUS.BTM** [**ON** | **OFF**]
SYStem.Option BTM [**ON** | **OFF**] (deprecated)

Control for the NEXUS branch trace messages.

NEXUS.CHAN — Enable CHAN register write trace messages

Format: **NEXUS.CHAN** [**ON** | **OFF**]
SYStem.Option CHAN [**ON** | **OFF**] (deprecated)

Control for the NEXUS channel register write trace messages. CHAN register write tracing requires the channel being serviced to have program trace enabled.

NEXUS.CLIENT<x>.MODE — Set data trace mode of nexus client

Format: **NEXUS.CLIENT1.MODE** [**OFF** | **Read** | **Write** | **ReadWrite**]

Sets the data trace mode of the selected trace client. Select the trace client using **NEXUS.CLIENT<x>.SELECT** before setting the trace mode.

Format: **NEXUS.CLIENT1.SELECT** <*client*>

<*client*>: **OFF** | **CDC** | **CDC2**

Select the eTPU Coherent Dual-Parameter Controller's trace client for data tracing. CDC belongs to eTPU_A and aTPU_B, CDC2 belongs to eTPU_C and eTPU_D.

NEXUS.DTM Control for data trace messages

Format: **NEXUS.DTM** [**OFF** | **Read** | **Write** | **ReadWrite**]
SYStem.Option DTM [**OFF** | **Read** | **Write** | **ReadWrite**] (deprecated)

OFF (default)	No data trace messages are output by NEXUS.
Read	NEXUS outputs data trace messages for read accesses.
Write	NEXUS outputs data trace messages for write accesses.
ReadWrite	NEXUS outputs data trace messages for read and write accesses.

NEXUS.OFF Switch the NEXUS trace port off

Format: **NEXUS.OFF**

If the debugger is used stand-alone, the trace port is disabled by the debugger.

NEXUS.ON Switch the NEXUS trace port on

Format: **NEXUS.ON**

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

Format: **NEXUS.OTM** [**ON** | **OFF**]
SYStem.Option OTM [**ON** | **OFF**] (deprecated)

Enables ownership trace messaging. On the eTPU, an OTM is generated each time a channel starts or ends and contains (amongst others) channel number and HSR ID. The information of OTMs is displayed in the flow trace and also in trace chart views (e.g. **Trace.CHART.TASKSRV**)

NEXUS.PortMode Define MCKO frequency

Format: **NEXUS.PortMode** *<divider>*
SYStem.Option MCKO *<divider>* (deprecated)

<divider>: **1/1** | **1/2** | **1/3** | **1/4** | **1/8**

Set the frequency of MCKO relative to the core frequency. The port mode setting must be the same for all cores (e200 and eTPU).

NEXUS.PortSize Define the width of MDO

Format: **NEXUS.PortSize** *<port_size>*
SYStem.Option NEXUS *<port_size>* (deprecated)

<port_size>: **MDO2** | **MDO4** | **MDO8** | **MDO12** | **MDO16**

The width of MDO can only be set if the SYStem mode is DOWN. The port size setting must be the same for all cores (e200 and eTPU).

NEXUS.PTCE — Program trace enable per channel

Format: **NEXUS.PTCE** <*value*>
SYStem.Option PTCE <*value*> (deprecated)

<*value*>: **bit mask** [**ch31, ch30, ch29** … **ch1, ch0**]

Enables program trace for channels, which have the regarding bit of the value set to one. e.g. 0x00000009: enable program trace for channel 0 and 4.

NEXUS.Register — Display NEXUS trace control registers

Format: **NEXUS.Register**

This command opens a window which shows the NEXUS configuration and status registers of NPC, core and other trace clients.

NEXUS.RESet — Reset NEXUS trace port settings

Format: **NEXUS.RESet**

Resets NEXUS trace port settings to default settings.

NEXUS.STALL — Stall the program execution

Format: **NEXUS.STALL** [**ON** | **OFF**]
SYStem.Option STALL [**ON** | **OFF**] (deprecated)

Stall the program execution whenever the on-chip NEXUS-FIFO threatens to overflow. If this option is enabled, the NEXUS port controller will stop the core's execution pipeline until all messaged in the on-chip NEXUS FIFO are sent. 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 cannot avoid them completely.

Format: **NEXUS.state**

Display NEXUS trace configuration window.

TrOnchip.BusTrigger — Trigger bus on debug event

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

<event>:
CRW
HSR
LINK
MRL
TDL

Enables or disables events on which a bus trigger signal will be generated.

Enables or disables events on which the trace will be triggered.

CRW	Channel register write.
HSR	Host service request.
LINK	Link service request.
MRL	Match recognition request.
TDL	Transition detect request.

NOTE: The eTPU implementation for the special events CRW, HSR, LINK, MRL and TDL allows configuring each event independently to generate a watchpoint **or** a breakpoint. It is however not possible to generate a watchpoint and a breakpoint at the same time for the same event. See **TrOnchip.Set** for details.

TrOnchip.CBI — Halt on client breakpoint input

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

Enables or disables the “Halt on Client breakpoint” break condition. If enabled, the eTPU will halt and run synchronized to the master core (e.g. PowerPC).

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

With this setting you can select how the eTPU should react on a client breakpoint input / twin engine breakpoint. If this setting is **OFF** (default), the eTPU will stop on completion of the current micro cycle. If **ON**, it will stop on completion of the current instruction thread, i.e. the eTPU only stop when it is in IDLE mode.

TrOnchip.CONVert Adjust range breakpoint in on-chip resource

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

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

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

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

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

Default: OFF. If enabled, the debugger will use the EVTI signal to break program execution instead of sending a JTAG command. This will speed up reaction time. If the complex trigger unit is used to stop program execution, it is recommended to enable this option to achieve a shorter delay. If this option is disabled, the debugger will drive EVTI permanently high.

NOTE:
- Only enable this option if the EVTI pin of the processor is connected to the NEXUS connector.
- This option has no effect if TrOnchip.EVTEN is disabled in the PowerPC debugger.

TrOnchip.EXTernal External signals

Format: **TrOnchip.EXTernal** *<input>*

<input>:
OFF
IN0
IN1

Enables / selects a external input to trigger the trace. The inputs are locates at the TRACE32 Nexus Adapter.

TrOnchip.HTWIN Halt on twin engine breakpoint

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

Enables or disables the “Halt on Twin Engine” breakpoint. If enabled, the eTPU will halt and run synchronized to the eTPU connected to the same SCM.

Format: **TrOnchip.RESet**

Resets the trigger system to the default state.

TrOnchip.SCM Select channels for that breakpoints are effective

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

On-chip instruction and data address breakpoints/watchpoints by default match for any service channel. If the SCM value is different from “0xxx”, these breakpoints/watchpoints will only be effective for those channels that match to the used value/bit mask.

<value> | *<bitmask>* A value or bit mask to specify service channels.

TrOnchip.Set Break on debug event

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

<event>:
CRW
HSR
LINK
MRL
TDL

Enables or disables events on which the eTPU core will be halted. If the SCM value is different from “0xxx”, then the events will occur only if the current service channel number matches the SCM setting. Please refer to the eTPU user’s manual for more information.

NOTE: The eTPU implementation for the special events CRW, HSR, LINK, MRL and TDL allows configuring each event independently to generate a watchpoint **or** a breakpoint. It is however not possible to generate a watchpoint and a breakpoint at the same time for the same event.

SCM	Service channel number mask, value or bitmask are allowed.
CRW	Channel register write.
HSR	Host service request.
LINK	Link service request.
MRL	Match recognition request.
TDL	Transition detect request.
<value> | *<bitmask>*	A value or bit mask to specify service channels.

For example **Tronchip.Set HSR ON** (breakpoint on HSR) cannot be used together with **Tronchip.TraceTrigger HSR ON** (trace trigger on HSR) or **TrOnchip.BusTrigger HSR ON** (bus trigger on HSR). TraceTrigger and BusTrigger events can be enabled at the same time, because both configure for watchpoints.

If both breakpoint and watchpoint on a special event are enabled, the resulting action is undefined.

TrOnchip.TEnable — Set filter for the trace

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

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

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

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

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

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

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

TrOnchip.TraceTrigger Trigger trace on debug event

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

<event>: **CRW**
HSR
LINK
MRL
TDL

Enables or disables events on which the trace will be triggered.

NOTE: The eTPU implementation for the special events CRW, HSR, LINK, MRL and TDL allows configuring each event independently to generate a watchpoint **or** a breakpoint. It is however not possible to generate a watchpoint and a breakpoint at the same time for the same event. See **TrOnchip.Set** for details.

CRW Channel register write.

HSR Host service request.

LINK Link service request.

MRL Match recognition request.

TDL Transition detect request.

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

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

TrOnchip.state Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

Usage

The Complex Trigger Unit for eTPU is only available for the NEXUS class2/3+ debugger. It is only supported for NEXUS port sizes MDO8, MDO12 and MDO16

The Complex Trigger Unit for eTPU can not be programmed through the Analyzer.Program dialog in the eTPU debugger. Use the dialog of the PowerPC debugger instead. In order to declare an event for a eTPU NEXUS message, add the option `/<source>` to the event.

Example:

```
OTME task_count1 0x0900 /ETPU1
OTME task_count2 0x0100 /ETPU2
```

Complex Trigger programs can handle PowerPC and eTPU events at the same time. It is possible to e.g. start tracing on a PowerPC action and stop at an eTPU action.

NOTE:	For all events based on NEXUS trace messages, please make sure that the corresponding message type in enabled in the eTPU SYStem window, e.g. OTMEs need owner trace messages enabled. See **"NEXUS specific SYStem Settings"** for details.

Complex Trigger Examples for eTPU

Here are some examples on eTPU specific complex trigger programs. Please see **"Trace Filtering and Triggering with Debug Events"** (debugger_mpc5500.pdf) for a detailed description and more examples on general complex trigger features.

Example 1: Break if eTPU executed a task a given number of times

```
; Example                    ; open a trace programming window to enter
Trace.Program time_watch     ; the trigger program for the CTU

; trigger program

OTME task_start 0x0900 /ETPU1          ; event on eTPU1-OTM (task start)
OTME task_end   0x0000 /ETPU1          ; event on eTPU1-OTM (task end)

EVENTCOUNTER taskcount 1000.           ; task counter for 1000 events

start:
     Counter.Increment taskcount, GOTO intask IF task_start

intask:
     GOTO start IF task_end
     BREAK.PROGRAM IF taskcount

Go
```

Example 2: Break if eTPU task execution time exceeds a maximum time

```
; Example                        ; open a trace programming window to
                                 ; enter
Trace.Program time_watch         ; the trigger program for the CTU

; trigger program

OTME task_start 0x0900 /ETPU1    ; event on eTPU1-OTM (task start)
OTME task_end   0x0000 /ETPU1    ; event on eTPU1-OTM (task end)
TIMECOUNTER   tasktime 2.400ms   ; timer 2.4 ms
start:
     GOTO intask if task_start

intask:
     Counter.Increment tasktime
     Counter.Restart   tasktime  IF task_end
     GOTO start                  IF task_end
     BREAK.PROGRAM               IF tasktime&&!task_end
Go
```

Input Event	Meaning
IN	external input event IN0 or IN1 occurred
CRWM, TCODE_3C, TCODE_CRWM	channel register write message
CSSM, TCODE_3A, TCODE_CSSM	channel start service message
CTEM, TCODE_3B, TCODE_CTEM	channel trace enable message
DRM, TCODE_6, TCODE_DRM	data read message
DRSM, TCODE_E, TCODE_DRSM	data read sync message
DSM, TCODE_0, TCODE_DSM	debug status message
DWM, TCODE_5, TCODE_DWM	data write message
DWSM, TCODE_D, TCODE_DWSM	data write sync message
EM, TCODE_8, TCODE_EM	error message
EM_0, TCODE_8_0	error message 0 - OTM loss
EM_1, TCODE_8_1	error message 1 - BTM loss
EM_2, TCODE_8_2	error message 2 - DTM loss
EM_6, TCODE_8_6	error message 6 - WHM loss
EM_7, TCODE_8_7	error message 7 - BTM/DTM/OTM loss
EM_8, TCODE_8_8	error message 8 - BTM/DTM/OTM/WHM loss
EM_18, TCODE_8_18	error message 18 - DSM loss
EM_19, TCODE_8_19	error message 19 - BTM/DSM/DTM/OTM loss
EM_1A, TCODE_8_1A	error message 1A - BTM/DSM/DTM/OTM/WHM loss
IHM, TCODE_1C, TCODE_IHM	hardware event message
IHSM, TCODE_1D, TCODE_IHSM	hardware event sync message
OTM, TCODE_2, TCODE_OTM	ownership trace message
PTCM, TCODE_21, TCODE_PTCM	repeat branch message
RFM, TCODE_1B, TCODE_RFM	resource full message
WHM, TCODE_F, TCODE_WHM	watchpoint hit message

For not CPU-specific keywords, see **non-declarable input variables** in **“ICE/FIRE Analyzer Trigger Unit Programming Guide”** (analyzer_prog.pdf).

JTAG Connector

Mechanical Description

JTAG Connector MPC55XX (OnCE)

Signal	Pin	Pin	Signal
TDI	1	2	GND
TDO	3	4	GND
TCK	5	6	GND
(EVTI-)	7	8	N/C
RESET-	9	10	TMS
JTAG-VTREF	11	12	GND
(RDY-)	13	14	JCOMP

This is a standard 14 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.). (Signals in brackets are not strong necessary for basic debugging, but its recommended to take in consideration for future designs.)

Connector for COLDFIRE

Signal	Pin	Pin	Signal
N/C	1	2	BKPT-
GND	3	4	DSCLK
GND	5	6	N/C
RESET-/RSTI-	7	8	DSI
1.8-5.0V	9	10	DSO
GND	11	12	PST3
PST2	13	14	PST1
PST0	15	16	DDATA3
DDATA2	17	18	DDATA1
DDATA0	19	20	GND
N/C	21	22	N/C
GND	23	24	PSTCLK/CPUCLK
1.8-5.0V	25	26	TEA-/TA-/DTACK-