LAUTERBACH
DEVELOPMENT TOOLS

RX Debugger

MANUAL

RX Debugger

RX Debugger

Version 06-Jun-2024

Introduction

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

- RX61x
- RX62x
- RX63x

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

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

Brief Overview of Documents for New Users

Architecture-independent information:

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

Architecture-specific information:

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

Demo and Start-up Scripts

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

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

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

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

You can also manually navigate in the ~~/demo/rx/ subfolder of the system directory of TRACE32.

Warning

Signal Level

Debug signals are driven with the same voltage level as the target voltage.

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.

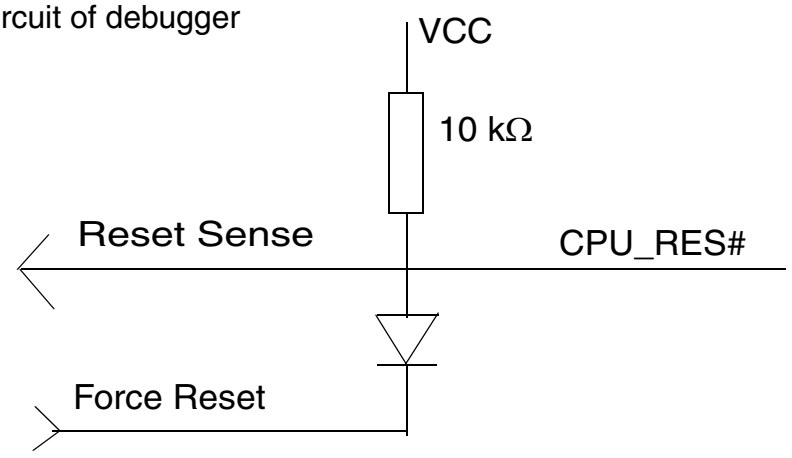
Location of Debug Connector

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

Reset Line

Ensure that the debugger signal $\overline{\text{RESET}}$ is connected directly to the $\overline{\text{RESET}}$ of the processor. This will provide the ability for the debugger to drive and sense the status of $\overline{\text{RESET}}$.

Reset circuit of debugger



Enable Debug Mode

The debugger hardware forces the signal EMLE (emulation enable) to VCC. Same can be done on the target board by a switch or jumper. In that case there is no need to connect the EMLE signal to the debug connector.

Enable AUD Trace lines

For some CPUs the AUD trace lines are shared with port lines. The AUD signals have to be enabled by the appropriate port function registers.

Use command: **Data.Set** *<register_address>* **%***<register_width>* *<register_value>*

Starting up the Debugger is done as follows:

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

```
b:
```

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

```
SYStem.CPU RX6108
```

3. If the TRACE32-ICD hardware is installed properly, the following CPU is the default setting:

   RX6108

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

```
MAP.BOnchip 0xFFE00000++0xFFFFFFFF
```

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

5. Enter debug mode

```
SYStem.Up
```

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

```
Data.Set …
```

6. Load the program.

```
Data.LOAD.ELF diabc.elf      ; elf specifies the format, diabc.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"**.

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

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

WinCLEAR                            ; Delete all windows

MAP.BOnchip                         ; Specify where's FLASH/ROM
0xFFE00000++0x0fffffffff

SYStem.CPU RX6108                   ; Select the processor type

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

Data.LOAD example.elf               ; Load the application

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

List.Mix                            ; Open disassembly window         *)

Register.view /SpotLight            ; Open register window            *)

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

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

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

Break.Set sieve                     ; Set breakpoint to function sieve

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

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

SYStem.Up Errors

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

All	The target has no power.
All	EMLE pin not at VCC level.
All	The target is in reset: The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up.
All	There is logic added to the JTAG state machine: By default the debugger supports only one processor on one JTAG chain. If the processor is member of a JTAG chain the debugger has to be informed about the target JTAG chain configuration. See Multicore Debugging.
All	There are additional loads or capacities on the JTAG lines.

Trace Errors

There are several reasons for Trace Errors.

1. AUD pins not enabled:

   For some CPUs the AUD trace lines are shared with port lines. The AUD signals have to be enabled by the appropriate port function registers.

2. Hardware problems with AUD trace interface:

   The TRACE32 AUD trace is designed for up to 200 MHz AUDCLK. Take care about the layout of your target especially the routing of AUDCLK. In case of Trace Errors try lower AUDCLK speeds with command **SYStem.Option.AUDCLK** 1/1, 1/2, 1/4 1/8.

3. AUD protocol errors

   In case of RealTimeTrace mode (**SYStem.Option.AUDRTT** ON) it might happen the CPU executes program quicker than the AUD interface can transfer its information. In this case the current AUD transfer is skipped, trace information gets lost and as a result it is not possible to calculate the correct program flow. To prevents this kind of error the AUD clock should be as high as possible. If this does not solve the problem you have to switch OFF the RealTimeTrace mode (**SYStem.Option.AUDRTT** OFF)

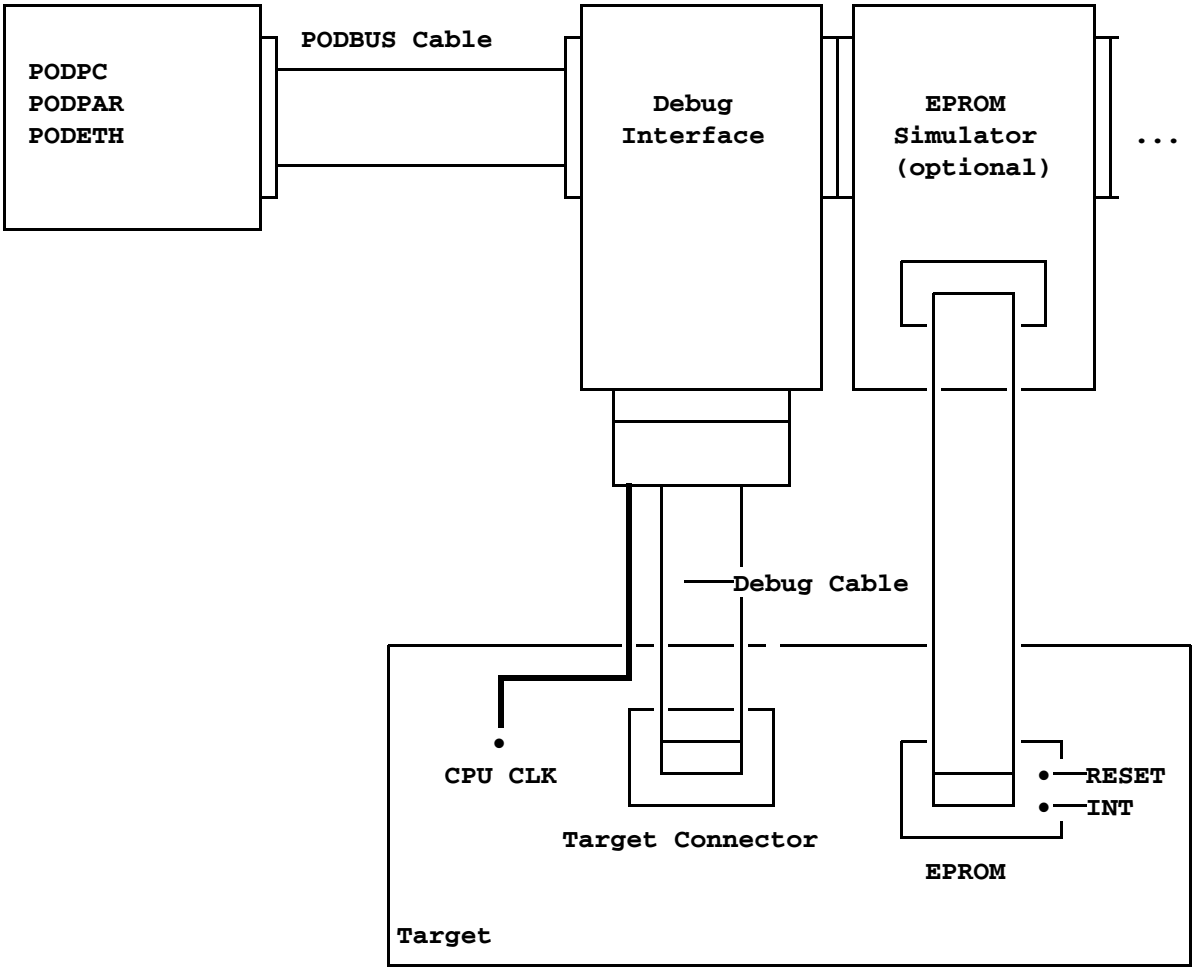
4. Calculation Error

   The trace listing is calculated in conjunction of the trace records plus the memory contents. If the memory content has changed (self modified code, different chipselect setting, MMU …) in between run time and calculation time there will be mismatches of the trace records compared to the current program in memory.

FAQ

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

System Overview



Basic configuration for the BDM Interface

SYStem.CONFIG Configure debugger according to target topology

Format:	**SYStem.CONFIG** *<parameter> <number_or_address>* **SYStem.MultiCore** *<parameter> <number_or_address>* (deprecated)
<parameter>:	**CORE** *<core>*
<parameter>: (JTAG):	**DRPRE** *<bits>* **DRPOST** *<bits>* **IRPRE** *<bits>* **IRPOST** *<bits>* **TAPState** *<state>* **TCKLevel** *<level>* **TriState** [**ON** | **OFF**] **Slave** [**ON** | **OFF**]

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

TriState has to be used if several debuggers ("via separate cables") are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode. Please note: nTRST must have a pull-up resistor on the target, TCK can have a pull-up or pull-down resistor, other trigger inputs need to be kept in inactive state.

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

CORE

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

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

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

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

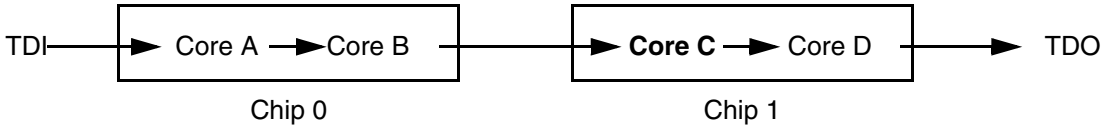
IRPOST (default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.

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

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

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

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



Below, configuration for core C.

Instruction register length of

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

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

0	Exit2-DR
1	Exit1-DR
2	Shift-DR
3	Pause-DR
4	Select-IR-Scan
5	Update-DR
6	Capture-DR
7	Select-DR-Scan
8	Exit2-IR
9	Exit1-IR
10	Shift-IR
11	Pause-IR
12	Run-Test/Idle
13	Update-IR
14	Capture-IR
15	Test-Logic-Reset

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

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

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

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

Non-generic Chips

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

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

Generic Chips

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

Start-up Process

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

Format: **SYStem.CONFIG.state**

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.

SYStem.CPU CPU type selection

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

<cpu>: **AUTO** | **RX6108** | **RX62N7** …

Default selection: RX6108.

Selects the CPU type. **AUTO**: Automatic CPU detection during SYStem.UP. The JTAG clock has to be less/equal 5 MHz. The detected CPU type can be checked with the function **CPU()**.

SYStem.JtagClock JTAG clock selection

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

Default frequency: 10 MHz.

Selects the JTAG port frequency (TCK). The RX-Core is designed for a maximum TCK clockspeed of 20 MHz!

Any frequency can be entered, it will be generated by the debuggers internal PLL.

There is an additional plug on the debug cable on the debugger side. This plug can be used as an external clock input. With setting **EXT/x** the external clock input (divided by **x**) is used as JTAG port frequency.

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

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

Default: OFF.

If the system is locked (ON) no access to the JTAG port will be performed by the debugger. All JTAG connector signals of the debugger are tristated.

This command is useful if there are additional CPUs (Cores) on the target which have to use the same JTAG lines for debugging. By locking the T32 debugger lines, a different debugger can own mastership of the JTAG interface.

It must be ensured that the state of the RX-core JTAG state machine remains unchanged while the system is locked. To ensure correct hand-over between two debuggers, a pull-down resistor on TCK and a pull-up resistor on /TRST is required.

SYStem.MemAccess Select run-time memory access method

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

Enable **CPU** (deprecated)	Memory access during program execution to target is enabled.
Denied (default)	Memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

If MemAccess is set to **Enable**, setting breakpoints and memory accesses (access class “E”) is possible even if the core is running.

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

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

<*mode*>: **Down**
Go
Up

Down Disables the Debugger.

Go Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is in the system.up mode and running. Now, the processor can be stopped with the break command or until any break condition occurs.

Up Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging. All register are set to the default value.

Attach Not supported.

NoDebug Not supported.

StandBy Not supported.

SYStem.Option.BigEndian Define byte order (endianness)

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

Default: OFF.

This option selects the byte ordering mechanism.

SYStem.Option.IMASKASM Interrupt disable

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

Mask interrupts during assembler single steps. Useful to prevent interrupt disturbance during assembler single stepping.

SYStem.Option.IMASKHLL Interrupt disable

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

Mask interrupts during HLL single steps. Useful to prevent interrupt disturbance during HLL single stepping.

SYStem.Option.KEYCODE Keycode

Format: **SYStem.Option.KEYCODE** [*<16x_8bit_values>*]

Has to be the same value as present in CPU Flash at address 0xFFFFFFA0--0xFFFFFFAF

The KEYCODE is sent to the CPU during system up to unlock the ID-Code-Protection unit. A matching KEYCODE is a must to get debug control. More details on ID-Code-Protection can be found in the CPU-Users-Manual.

Breakpoints

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

Software Breakpoints

Software breakpoints are the default breakpoints. A special breakcode is patched to memory so it only can be used in RAM or FLASH areas.There is no restriction in the number of software breakpoints.

On-chip Breakpoints

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

CPU Family	Number of Address Breakpoints	Number of Data Breakpoints	Sequential Breakpoints
RX	8	4	---

Breakpoint in ROM

With the command **MAP.BOnchip** *<range>* it is possible to inform the debugger about ROM (FLASH,EPROM) address ranges in target. If a breakpoint is set within the specified address range the debugger uses automatically the available on-chip breakpoints.

Example for Breakpoints

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

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

The following breakpoint combinations are possible.

Software breakpoints:

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

Break.Set 0x101000 /Program          ; Software Breakpoint 2

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

On-chip breakpoints:

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

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

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

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

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

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

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

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

Format: **TrOnchip.RESet**

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

Format:	**TrOnchip.SEQ** *<mode>*
<mode>:	**OFF** **CD** **BCD** **ABCD**

This trigger-on-chip command selects sequential breakpoints.

OFF	Sequential break off.
BA, CD	Sequential break, first condition, then second condition.
BCD, CBA	Sequential break, first condition, then second condition, then third condition.
ABCD, DCBA	Sequential break, first condition, then second condition, then third condition and the fourth condition.

```
Break.Set sieve /Charly /Program

Var.Break.Set flags[3] /Delta /Write

TrOnchip.SEQ CD
```

TrOnchip.state — Display on-chip trigger window

Format:	**TrOnchip.state**

Opens the **TrOnchip.state** window.

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
D	Data

Trace

Analysis of the program history is supported as AUD-Trace and Onchip-Trace.

AUD-Trace

The AUD trace interface supports the branch trace function and the window data trace function.

Each change in program flow caused by execution or interruption of branch instructions are detected and branch destination and branch source address are output.

The data trace function is for outputting memory access information. Two data-addresses (ranges) are supported.

Selection of Branch and Data Trace Recording

Trace recording is defined by four debugger settings.

- **SYStem.Option.AUDBT** (Branch Trace enable)
- **SYStem.Option.AUDDT** (Data Trace enable)
- Break Action setting “TRaceEnable”
- Break Action setting “TRaceData”

TRaceEna	TRaceData	AUDBT	AUDDT	ProgTrace	DataTrace
0	0	0	0		
0	0	0	1		all data
0	0	1	0	all program	
0	0	1	1	all program	all data
0	1	0	X		selective
0	1	1	X	all program	selective
1	X	X	X		selective

The BreakAction “TRaceEnable” has highes priority to get selective DataTrace recording only.

The BreakAction “TTraceData” comes next to enable selective DataTrace. Depending on **SYStem.Option.AUDBT** also the program flow will be traced.

SYStem.Option.AUDBT AUD branch trace enable

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

If ON all changes in program flow are output on the AUD trace port. By default this option is enabled.

SYStem.Option.AUDDT AUD data trace enable

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

If ON all accesses to data range A and/or range B are output on the AUD trace port. By default this option is OFF.

SYStem.Option.AUDRTT AUD real time trace enable

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

AUD full-trace / real-time-trace selection.

If OFF all trace information is output on the AUD trace port. In case of overrun of the AUD interface the CPU is stopped till overrun condition is no more present. This way all trace records contain valid data.

If ON application runtime is not influenced by the AUD interface. In case of overrun of the AUD interface there might be missing or not valid trace cycles which cause a buggy trace listing.

Default setting is OFF.

SYStem.Option.AUDClock AUD clock select

Format: **SYStem.Option.AUDClock** [**1/1** | **1/2** | **1/4** | **1/8**]

Selects the clockspeed of the AUD interface. CPU system clock divided by 1,2,4 or 8.

The AUD clock should be as fast as possible to prevent AUD overrun condition.

On-chip Trace

The RX core devices are equipped with an onchip trace buffer. Depending on the device in use it can cover up to 256 branch and/or data records.

The trace functionality is equal to an AUD trace. It requires no extra pins and has no influence on the performance of program execution.

See also: **AUD-Trace**.

Onchip.Mode.ProgramTrace

Program flow trace enable

Format: **Onchip.Mode.ProgramTrace** [**ON** | **OFF**]

Default: ON

Enables tracing of program flow activity.

Onchip.Mode.DataTrace

Data trace enable

Format: **Onchip.Mode.DataTrace** [**ON** | **OFF**]

Default: OFF

Enables tracing of the data-cylces. This setting is ignored if selective trace (TraceEnable) is active.

On-chip Performance Analysis

The RX-Core supports two performance counters. This counters can be configured to count a wide range of different events.

Runtime Measurement

The RX debug interface includes one signal which gives information about the program-run-status (application code running). This status line is sensed by the ICD debugger with a resolution of **100ns**.

The debuggers RUNTIME window gives detailed information about the complete run-time of the application code and the run-time since the last GO/STEP/STEP-OVER command.

JTAG Connector

Signal	Pin	Pin	Signal
TCK	1	2	GND
TRST-	3	4	(EMLE)
TDO	5	6	(MDE)
(MD1)	7	8	VCC
TMS	9	10	(MD0)
TDI	11	12	GND
RESET-	13	14	GND

JTAG Connector	Signal Description	CPU Signal
TMS	Jtag-TMS, output of debugger	TMS
TDI	Jtag-TDI, output of debugger	TDI
TCK	Jtag-TCK, output of debugger	TCK
/TRST	Jtag-TRST, output of debugger	TRST#
TDO	Jtag-TDO, input for debugger	TDO
/ASEBRK	Break Acknowledge, input/output for debugger	**ASEBRK,BRKACK**
/RESET	RESET input/output for debugger	/RESET
EMLE	Emulation mode enable input/output of debugger Default setting: output HIGH	EMLE
MD0..2	CPU mode pins input/output of debugger Default setting: Input	MD0..2

AUD Trace Connector

Signal	Pin	Pin	Signal
(MDE)	1	2	(MD0)
(EMLE)	3	4	N/C
N/C	5	6	AUDCK
N/C	7	8	(MD1)
RESET-	9	10	N/C
TDO	11	12	N/C
N/C	13	14	VCC
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	N/C
TRST-	21	22	N/C
N/C	23	24	AUDATA3
N/C	25	26	AUDATA2
N/C	27	28	AUDATA1
N/C	29	30	AUDATA0
N/C	31	32	AUDSYNC-
N/C	33	34	N/C
N/C	35	36	N/C
N/C	37	38	N/C

Mictor Connector	Signal Description	CPU Signal
AUDCK	AUD clock, output of cpu	AUDCK
AUDSYNC-	AUD sync, output of cpu	AUDSYNC-
AUDATA0..3	AUD data, output of cpu	AUDATA0..3

All other signals are described in chapter **JTAG Connector**.