LAUTERBACH
DEVELOPMENT TOOLS

H8S/23x9 Debugger

MANUAL

H8S/23x9 Debugger

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

ICD In-Circuit Debugger ..... 🗁

Processor Architecture Manuals ..... 🗁

H8S ..... 🗁

H8S/23x9 Debugger **1**

Introduction **4**

Brief Overview of Documents for New Users 4

Demo and Start-up Scripts 5

Warning **6**

Application Note **7**

Location of Debug Connector 7

Reset Line 7

Enable JTAG Mode 7

Quick Start JTAG **8**

Troubleshooting **10**

SYStem.Up Errors 10

FAQ **10**

Configuration **11**

System Overview 11

System Commands **12**

SYStem.CPU CPU type selection 12

SYStem.JtagClock JTAG clock selection 12

SYStem.Option.Advanced Advanced addressing mode 13

SYStem.Option.BrkVector Breakpoint trap 13

SYStem.Option.IMASKASM Interrupt disable on ASM 13

SYStem.Option.IMASKHLL Interrupt disable on HLL 13

SYStem.Option.KEYCODE Keycode 14

SYStem.Option.SLOWRESET Slow reset 14

SYStem.MemAccess Select run-time memory access method 15

SYStem.Mode System mode selection 15

Multicore Debugging **17**

SYStem.LOCK JTAG lock 17
SYStem.CONFIG Configure debugger according to target topology 18
Daisy-Chain Example 20
TapStates 21
SYStem.CONFIG.CORE Assign core to TRACE32 instance 22
SYStem.CONFIG.state Display target configuration 23

Breakpoints ... **24**
Software Breakpoints 24
On-chip Breakpoints 24
Breakpoint in ROM 24
Example for Breakpoints 24

TrOnchip Commands ... **26**
TrOnchip.state Display on-chip trigger window 26
TrOnchip.CONVert Adjust range breakpoint in on-chip resource 26
TrOnchip.DMA Trigger on DMA cycle 26
TrOnchip.DTC Trigger on DTC cycle 27
TrOnchip.SIZE Trigger on byte, word, long memory accesses 27
TrOnchip.RESet Set on-chip trigger to default state 27
TrOnchip.SEQ Sequential breakpoints 28

Memory Classes ... **29**

Trace ... **30**
FIFO Trace 30

Runtime Measurement ... **31**

JTAG Connector ... **32**

H8S/23x9 Debugger

Version 06-Jun-2024

Introduction

This document describes the processor specific settings and features of the TRACE32 debugger for the following CPUs:

- H8S/2329, H8S/2339
- H8S/2367, H8S/2377

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

Brief Overview of Documents for New Users

Architecture-independent information:

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

Architecture-specific information:

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

Demo and Start-up Scripts

Lauterbach provides ready-to-run start-up scripts for known H8S/23x9 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/h8s/ subfolder of the system directory of TRACE32.

Warning

Signal Level

The debugger drives the output pins of the JTAG connector with 3.3 V always.

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.

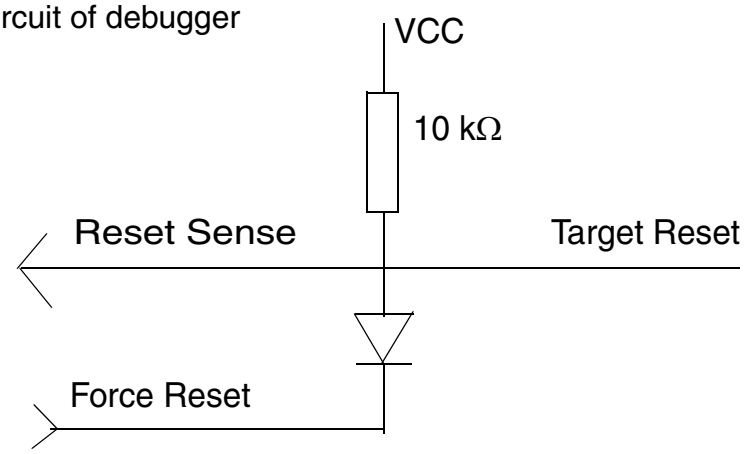
Location of Debug Connector

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 JTAG Mode

Connect signal EMLE to VCC (enable debug mode)

Connect signal FWE to VCC (enable FLASH programming)

Connect signals MD[2..0] to VCC (enable Mode-7)

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 H8S/2339
```

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

   H8S

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

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

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

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

SYStem.CPU H8S/2339                 ; Select the processor type

SYStem.Up                           ; Reset the target and enter debug
                                    mode

Data.LOAD.COFF GNUSH7.X             ; Load the application

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

Data.List                           ; Open disassembly window *)

Register                            ; Open register window *)

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

PER                                 ; Open window with peripheral register

Break.Set sieve                     ; Set breakpoint to function sieve

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

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

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

Troubleshooting

SYStem.Up Errors

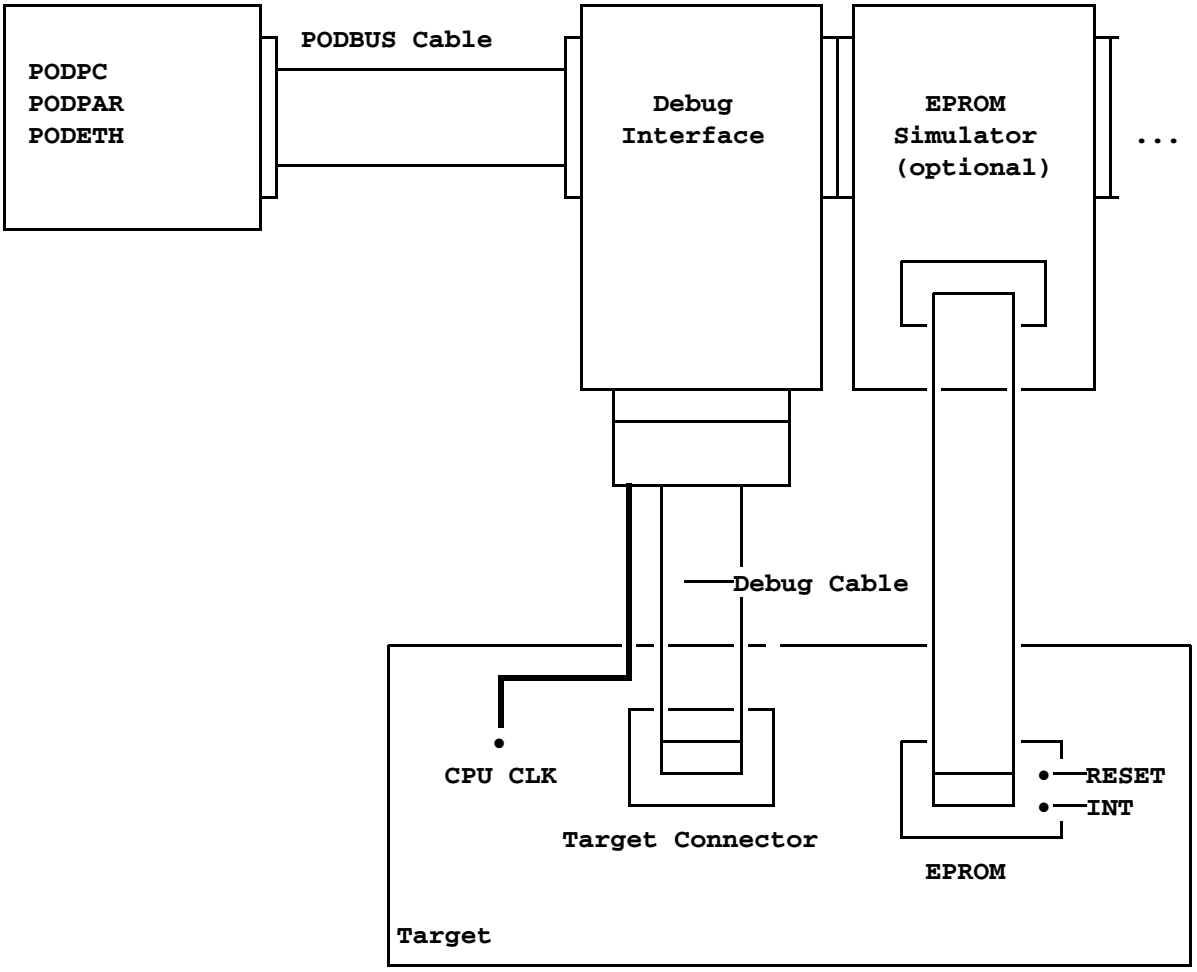
The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

All	The target has no power.
All	The 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.
All	KEYCODE if does not match FLASH content and FLASH programming disabled (FWE pin low level)

FAQ

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

System Overview



Basic configuration for the BDM Interface

SYStem.CPU — CPU type selection

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**H8S/2339**

Default selection: H8S/2339.

Selects the CPU type.

SYStem.JtagClock — JTAG clock selection

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

Default frequency: 20 MHz.

Selects the JTAG port frequency (TCK). The maximum frequency is the same as the operation frequency of the chip.

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.Option.Advanced** [**ON** | **OFF**]

Defines the address mode of the CPU.

OFF Normal address mode (64K).

ON Advanced address mode (16M).

SYStem.Option.BrkVector Breakpoint trap

Format: **SYStem.Option.BrkVector** *<0..3>*

Defines the number of the TRAPA-Instruction used for breakpoints and single stepping.

SYStem.Option.IMASKASM Interrupt disable on ASM

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 on HLL

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

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

Format: **SYStem.Option.KEYCODE** [*<32bit_value>*]

During **SYStem.Up** the KEYCODE is sent to the CPU and compared against certain Flash contents. If the KEYCODE does not fit, the CPU automatically erases its FLASH before the debug monitor can be downloaded. This is a special security feature of the CPU.

H8S/2319	Keycode has to be the same value as present in CPU Flash at address 0x78--0x7B
H8S/2367 **H8S/2377** **H8SX**	Keycode has to be the same value as present in CPU Flash at address 0x4--0x7

SYStem.Option.SLOWRESET Slow reset

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

Default: OFF.

When this command is set to ON, debug communication starts five seconds after rising edge of reset.

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.

SYStem.Mode

System mode selection

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

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.

If your H8S device is the only one connected to the JTAG connector then the following system setting should be left in their default position.

If your H8S CPU is lined up in a target JTAG chain then the debugger has to be informed about the “position” of the H8S device inside the JTAG chain. Following system settings have to be done according to your target configuration.

SYStem.LOCK JTAG lock

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 H8S debugger lines a different debugger can own mastership of the JTAG interface.

It must be ensured that the state of the H8S 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.

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.

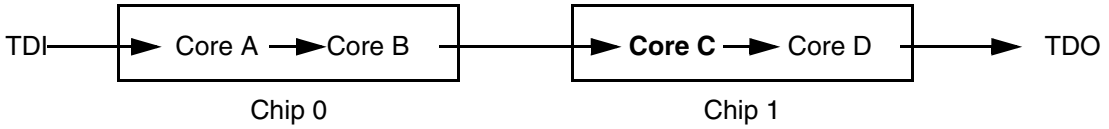
IRPOST (default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.

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 configurations settings.

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 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
H8S	2	2	---
H8SX	4	1	B>A C>B>A D>C>B>A

Breakpoint in ROM

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

Example for Breakpoints

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

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

The following breakpoint combinations are possible.

1. Software breakpoints:

```
Break.Set 0x100000 /Program    ; Software Breakpoint 1
Break.Set 0x101000 /Program    ; Software Breakpoint 2
Break.Set 0xx /Program         ; Software Breakpoint 3
```

2. On-chip breakpoints:

```
Break.Set 0x100 /Program       ; On-chip Breakpoint 1
Break.Set 0x0ff00 /Program     ; On-chip Breakpoint 2
```

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

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.DMA — Trigger on DMA cycle

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

Enable trigger on DMA cycle.

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

Enable trigger on DTC cycle.

TrOnchip.SIZE Trigger on byte, word, long memory accesses

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

If ON, breakpoints on single-byte, two-byte or four-byte addressranges only hit if the CPU accesses this ranges with a byte, word or long buscycle.

Default: OFF

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** **BA** **CBA** **DCBA**

This trigger-on-chip command selects sequential breakpoints.

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

```
Break.Set sieve /Delta /Program

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

TrOnchip.SEQ DC
```

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
D	Data

Trace

To support analysis of the program history the ICD supports following algorithm

FIFO Trace

This CPU includes a 4-stage branch trace. This trace holds the source address of the last four program flow changes.

The ICD command “FIFO” opens a window which displays the content of the branch trace.

This trace method does not slow down program execution.

Runtime Measurement

Runtime measurement is done with about **5us** resolution.

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	GND
TDO	5	6	GND
RSTOUT-	7	8	VCC
TMS	9	10	GND
TDI	11	12	GND
RESET-	13	14	GND

Signal description:

TMS	Jtag-TMS, output of debugger
TDI	Jtag-TDI, output of debugger
TCK	Jtag-TCK, output of debugger
/TRST	Jtag-TRST, output of debugger
TDO	Jtag-TDO, input for debugger
/RESET	RESET, input/output for debugger Connect this pin direct to the CPUs /RESET line
/RSTOUT	/RESET-Out, output of debugger This signal can be use as additional reset source for the target reset logic.
VCC	MCU Vcc pin, input for debugger (10 mA)