SH2, SH3 and SH4 Debugger

MANUAL

SH2, SH3 and SH4 Debugger

Version 06-Jun-2024

History

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

Introduction

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

- SH4A
- SH4 (7750, 7751)
- SH3 (7709A, 7729)
- SH2A
- SH2 (7047F, 7058FCC, 7144/45)
- ST40 (ST40STB1, ST40RA166, ST40GX1, ST40NGX1, SH4-202)

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 SuperH 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/sh/` 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.

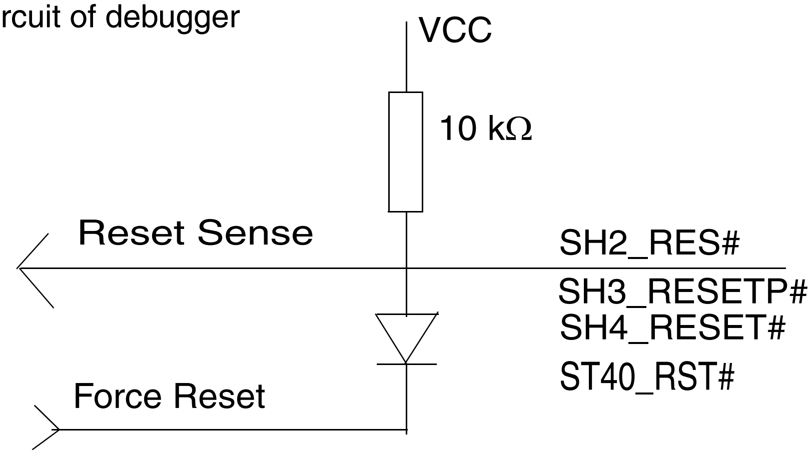
Location of Debug Connector

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 SH2

SH7047:

- Signal /DBGMD has to be forced to GND (debug mode enable)

SH7144/45:

- Signal DBGMD has to be forced to VCC (debug mode enable)
- Signal FWE has to be forced to GND (FLASH write enable)

Enable JTAG Mode SH3

Signal ASEMD0 has to be forced to GND

SH7710/12 Solution Engine

The debug connector of the SH7710 Solution Engine requires a modification to support AUD trace. Please connect pin 1 (NC) with pin 35 (AUDCK).

Enable AUD Trace lines of SH7760

The CPUs AUD trace lines are shared with port lines. Trace functionality has to be enabled in CPU register IPSELR (set bit 12 and 13).

Use command: DATA.SET 0xFE400034 %Word 3003

Memory Mapping of SH7615/ SH7616 BusControlRegisters

As long as emulation is stopped the peripheral registers of addressrange

0xFFFFFFC0--0xFFFFFFFF are mapped to address range **0xFFFFFDC0--0xFFFFFDFF**.

This address range covers the BusControlRegisters. During program execution they can be accessed at their original address. When emulation is stopped they have to be accessed in the range **0xFFFFFDC0--0xFFFFFDFF**.

Enable 8-bit AUD Trace Interface of SH4-202

The CPUs AUD trace lines AUD[7..4] are shared with other CPU peripherals. For 8-bit AUD trace usage, these trace lines have to be enabled by setting bit-4 of CPU register SYS_CONF_REG (0xb9ee0004).

Attention: The access to SYS_CONF_REG only works if clocking of PLL2 is already initialized!

Find here a setup example:.

```
; inform TRACE32 software about 8-bit AUD trace usage
System.Option.AUD8 ON

; PLL2 init
Data.Set 0xb8800038 %Long 0x3000560e
; PLL2 enable (read-modify-write action)
Data.Set 0xb8800004 %Long DATA.LONG(d:0xb8800004)|0x1

; AUD8 bit enable (SYS_CONF_REG) bit-4
Data.Set 0xb9ee0004 %Long DATA.LONG(d:0xb9ee0004)|0x10
```

Add this lines to your TRACE32 setup file.

For 4-bit AUD trace mode no setup is required (default setting).

Quick Start JTAG

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 SH7750
```

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

   SH7750

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:

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

WinCLEAR                          ; Delete all windows

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

SYStem.CPU SH7750                 ; 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.view /SpotLight          ; Open register window *)

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

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

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

Break.Set sieve                   ; Set breakpoint to function sieve

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

Break.Set 0x101000 /Program       ; Set on-chip breakpoint to address
                                  ; 101000 (address 101000 is in ROM)
                                  ; (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.

Monitor Download Error

At System.Up the debugger loads a monitor program into the target CPU and checks if communication with the monitor works well.

Each CPU type has it’s own monitor program, so it is a must to inform the debugger about the **CPU in use** and the **endianness**. Use commands:

- **System.CPU**
- **System.Option.LittleEnd**

Trace Errors

There are several reasons for Trace Errors.

1. 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.

2. 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)

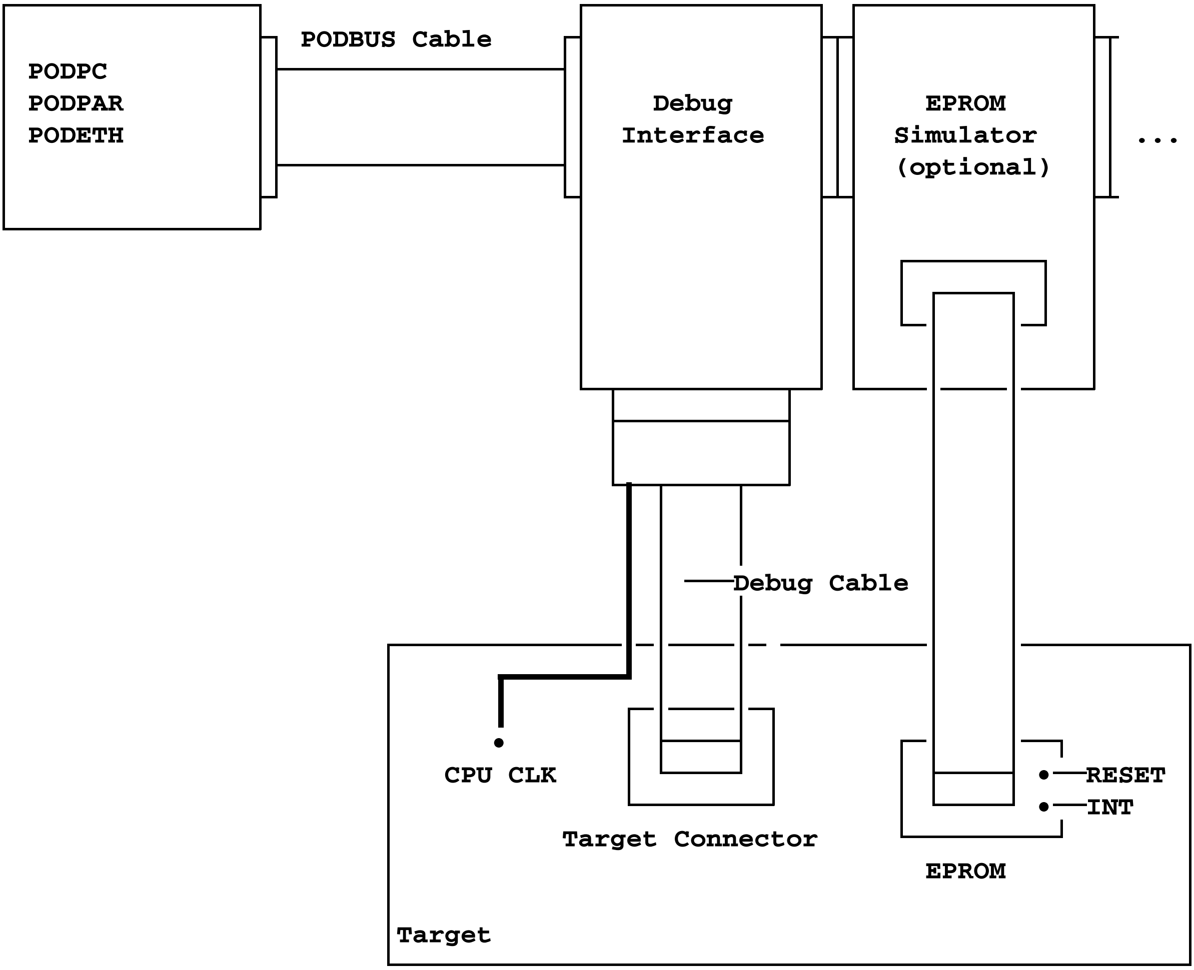
3. Calculation Error

   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.state Display target configuration

Format:	**SYStem.CONFIG.state** [**/**<*tab*>]
<*tab*>:	**DebugPort** | **Jtag**

Opens the **SYStem.CONFIG.state** window, where you can view and modify most of the target configuration settings. The configuration settings tell the debugger how to communicate with the chip on the target board and how to access the on-chip debug and trace facilities in order to accomplish the debugger's operations.

Alternatively, you can modify the target configuration settings via the TRACE32 command line with the **SYStem.CONFIG** commands. Note that the command line provides *additional* **SYStem.CONFIG** commands for settings that are *not* included in the **SYStem.CONFIG.state** window.

<*tab*>	Opens the **SYStem.CONFIG.state** window on the specified tab. For tab descriptions, see below.
DebugPort	Informs the debugger about the debug connector type and the communication protocol it shall use.
Jtag	Informs the debugger about the position of the Test Access Ports (TAP) in the JTAG chain which the debugger needs to talk to in order to access the debug and trace facilities on the chip.

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

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

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

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

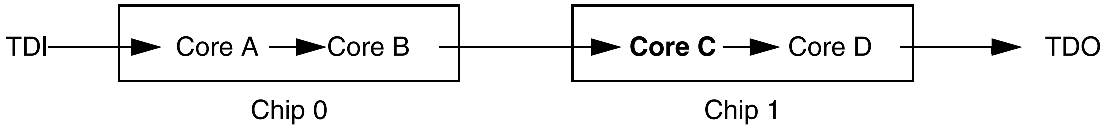
CORE

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

DRPRE

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

DRPOST	(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.
IRPRE	(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.
IRPOST	(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.
TAPState	(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.
TCKLevel	(default: 0) Level of TCK signal when all debuggers are tristated.
TriState	(default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.
Slave	(default: OFF) If more than one debugger share the same debug port, all except one must have this option active. JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).



Below, configuration for core C.

Instruction register length of

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

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

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

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

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

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

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

Non-generic Chips

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

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

Generic Chips

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

Start-up Process

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

SYStem.CPU — CPU type selection

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**AUTO** | **SH7750** | **SH7751** …

Default selection: SH7750.

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: 20 MHz.

Selects the JTAG port frequency (TCK). The SH3/4-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 SHx/ST40 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.

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

NOTE:

- Memory Access while core is running is only supported by SH2A and SH4A cores.
- Memory Access does not support the access to cache contents! To follow up variable changes in cached memory areas, the cache has to be switched OFF or set to WriteTrough mode. Write accesses only modify system- or target-memory **no** cache content!

SYStem.Mode — System mode selection

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** **Attach**

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	Attach to cpu without entering debug mode. There is no debug control but memory contents can be accessed. Only supported for SH4A cores.
NoDebug	Not supported.
StandBy	Not supported.

SYStem.Option.EnReset — Allow the debugger to drive nRESET

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

Default: ON.

If this option is disabled the debugger will never drive the nRESET line of the JTAG connector. This is necessary if nRESET is no open collector or tristate signal.

From the view of the SH core it is not necessary that nRESET becomes active at the start of a debug session (**SYStem.Up**), but there may be other logic on the target which requires a reset.

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.

Command is valid for SH2 only. Hook address for on-chip breakpoints. See also **Onchip Break SH7047**.

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.JtagWait JTAG wait enable

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

Has to be switched “ON” for SH7705, SH7709A till revision “S” and SH7729 till revision “R”.

This option enables a special bugfix for the CPUs Jtag interface. Jtag communication becomes slower!

SYStem.Option.KEYCODE Keycode SH7144/45

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

Has to be the same value as present in CPU Flash at address 0x20--0x23

The KEYCODE is sent to the CPU during system up. If the KEYCODE does not fit then the CPU automatically erases its FLASH before the debug monitor can be downloaded. This is a special security feature of the SH7144/45.

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

Default: OFF.

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

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Concepts"** (trace32_concepts.pdf).

NOTE:	**SYStem.Option.MMUSPACES** should not be set to **ON** if only one translation table is used on the target. If a debug session requires space IDs, you must observe the following sequence of steps: 1. Activate **SYStem.Option.MMUSPACES**. 2. Load the symbols with **Data.LOAD**. Otherwise, the internal symbol database of TRACE32 may become inconsistent.

Examples:

```
;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x012A:
Data.dump D:0x012A:0xC00208A

;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x0203:
Data.dump D:0x0203:0xC00208A
```

SYStem.Option.NoRunCheck

No check of the running state

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

Default: OFF.

This option advises the debugger not to do any running check. In this case the debugger does not even recognize that there will be no response from the processor. Therefore there is always the message “running” independent if the core is in power down or not. This can be used to overcome power saving modes in case the user knows when this happens and that he can manually de-activate and re-activate the running check.

SYStem.Option.SLOWRESET

Slow reset enable

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

Has to be switched “ON” if the reset line of the debug connector is not(!) connected direct to the CPU reset pin.

Problem: At system-up the debugger has to enable the CPUs debug mode first. This is done by a certain sequence of the debug signals. This sequence becomes faulty if the target includes a reset-circuit which hold the reset line for a unknown period.

If SlowReset is switched “ON” the debugger accepts a reset-hold period of up to 1 s. A system up needs about 3 s then!

SYStem.Option.SOFTLONG

Use LONG access for softbreak patching

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

Default: OFF.

A software breakpoint is a certain 16bit CPU instruction which is patched to the code. For applications which support 32bit write cycles only this option has to be switched ON. This way the break patching will not corrupt the instruction before/after the break address.

SYStem.Option.SOFTSLOT Prevent softbreak in slot-instruction

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

Default: OFF.

If set to ON, TRACE32 gives an error message if a software breakpoint should be set to a slot-instruction. It is a CPU restriction which does not allow to set software breakpoints to slot-instructions.

SYStem.Option.STEPSOFT Use software breakpoints for ASM stepping

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

Default: OFF.

If this option is ON software breakpoints are used for single stepping on assembler level (advanced users only).

SYStem.Option.LittleEnd Selection of little endian mode

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

With this option data is displayed little endian style.

SYStem.RESetOut Reset target without reset of debug port

Format:	**SYStem.RESetOut**

If possible (nRESET is open collector), this command asserts the nRESET line on the debug connector. This will reset the target including the CPU but not the debug port. The function only works when the system is in **SYStem.Mode.Up**.

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

Enter Vector-Base-Address here.

This value is used to detect and display exception table accesses in the trace listing. In case the application dynamically changes the VBR register settings the trace.list algorithm can use this value instead of the VBR register content.

Multicore Debugging

If your SHx/ST40 device is the only one connected to the JTAG connector then the following system setting should be left in their default position.

If your SHx/ST40 CPU is lined up in a target JTAG chain then the debugger has to be informed about the “position” of the device inside the JTAG chain. Following system settings have to be done according to your target configuration.

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
SH2A ST4A	10	2	C->D B->C->D A->B->C->D
SH4 ST40	6	2	C->D B->C->D A->B->C->D
SH3	2	1	---
SH7047 SH7144/45	1	---	---
SH7058	12	12	A->B->C->D

On-chip Breakpoints SH7047, SH7144, SH7145

The SH2 debugger uses the CPU internal UserBreakControl unit. This break unit generates an user exception, so some special settings and software changes are needed.

1. Define the UBC exception vector-12 (address 0x30++3)

2. The first instruction of the UBC exception handler must be a BRK (0x0000)

3. UBC exceptions are only accepted if the interrupt mask of SR register is less than 15. This means the application should not set the interrupt mask to 15!

4. The debugger has to be informed about the start address of the UBC exception. Use command **SYStem.Option.HOOK** *<ubc_exception_address>*

Example: Patch a 0x00000030 to address 0x30. This way the exception vector points to UBC-exception handler at address 0x30. There the first instruction is a BRK (0x0000).

```
SYSTEM.Option.HOOK 0x30
Register.Set SR 0xE0
```

On-chip Breakpoints SH72513

For SH2A production devices the debugger uses the CPU internal UserBreakControl unit. This break unit generates an user exception, so some special settings and software changes are needed.

1. Define the UBC exception vector-12 (address 0x30++3)

2. The first instruction of the UBC exception handler must be a BRK (0x003B)

3. UBC exceptions are only accepted if the interrupt mask of SR register is less than 15. This means the application should not set the interrupt mask to 15!

4. The debugger has to be informed about the start address of the UBC exception. Use command **SYStem.Option.HOOK** *<ubc_exception_address>*

Example: Patch a 0x00000008 value to address 0x30. This way the UBC-exception vector points to the exception handler at address 0x08.

There the first instruction is a BRK instruction (0x003B).

```
SYSTEM.Option.HOOK 0x08
Register.Set SR 0xE0
```

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
```

CPU specific BenchMarkCounter Commands

The benchmark counters can be read at run-time. Events can be assigned to **BMC.<counter>.EVENT <event>**. For a list of supported events, refer to **TrOnchip.PMCTRx**.

For information about *architecture-independent* **BMC** commands, refer to **"BMC"** (general_ref_b.pdf).

For information about *architecture-specific* **BMC** command(s), see command description(s) below.

BMC.<counter>.ATOB — Advise counter to count within AB-range

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

Advise the counter to count the specified event only in AB-range. Alpha and Beta markers are used to specify the AB-range.

Example to measure the time used by the function sieve:

```
BMC.<counter> ClockCylces                ; <counter> counts clock cycles

BMC.CLOCK 450.Mhz                        ; core is running at 450.MHz

Break.Set sieve /Alpha                   ; set a marker Alpha to the entry
                                         ; of the function sieve

Break.Set V.END(sieve)-1 /Beta           ; set a marker Beta to the exit
                                         ; of the function sieve

BMC.<counter>.ATOB ON                    ; advise <counter> to count only
                                         ; in AB-range
```

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
```

TrOnchip.IOB

I/O breakpoints (SH4, ST40)

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

Enable break on I/O access.

TrOnchip.LDTLB

LDTLB breakpoints

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

Enable break on LDTLB instruction.

TrOnchip.A.IBUS — I-bus breakpoints (SH2A)

Format:	**TrOnchip.ABCD.IBUS** *<action>*

Defines a trigger or trace action for I-Bus activity.

Selects onchip break action for /Alpha, /Beta, /Charly and /Delta breaks. The selected action becomes active for breakpoints which are set with option /Alpha, /Beta, /Charly or /Delta.

Actions can be defined for any I-Bus master (CPU, DMA, ADMA):

- **Break**: Stop program execution
- **TraceEnable**: Do selective trace
- **TraceOff**: Stop trace recording

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

Format:	**TrOnchip.RESet**

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

TrOnchip.RPE — Reset sequential trigger on reset point

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

If ON: If the break reset point register (BRPR) setting matches the instruction fetch address, the sequential state and execution count break register value are initialized. Default: OFF

TrOnchip.SEQ — Sequential breakpoints (SH4, ST40)

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

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

If ON, breakpoints on single-byte, two-byte or four-byte address ranges only hit if the CPU accesses this ranges with a byte, word or long bus cycle. Default: OFF

TrOnchip.state — Display on-chip trigger window

Format:	**TrOnchip.state**

Opens the **TrOnchip.state** window.

MMU.DUMP

Page wise display of MMU translation table

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

Displays the contents of the CPU specific MMU translation table.

- If called without parameters, the complete table will be displayed.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: displays the translation table of the specified process • else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Displays the MMU translation table entries of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

ITLB	Displays the contents of the ITLB translation table. Deprecated command syntax: MMU.ITLB.
UTLB	Displays the contents of the UTLB translation table. Deprecated command syntax: MMU.UTLB.

MMU.List — Compact display of MMU translation table

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.**<*table*>**.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**

Lists the address translation of the CPU-specific MMU table.

- If called without address or range parameters, the complete table will be displayed.
- If called without a table specifier, this command shows the debugger-internal translation table. See **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: list the translation table of the specified process • else, this command lists the table the CPU currently uses for MMU translation.

KernelPageTable	Lists the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

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

Loads the CPU-specific MMU translation table from the CPU to the debugger-internal static translation table.

- If called without parameters, the complete page table will be loaded. The list of static address translations can be viewed with **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, page table entries will only be loaded if their **logical** address matches with the given parameter.

Use this command to make the translation information available for the debugger even when the program execution is running and the debugger has no access to the page tables and TLBs. This is required for the real-time memory access. Use the command **TRANSlation.ON** to enable the debugger-internal MMU table.

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. • if *<range>* or *<address>* have a space ID: loads the translation table of the specified process • else, this command loads the table the CPU currently uses for MMU translation.

KernelPageTable	Loads the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Loads the MMU address translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manual**.
ALL [**Clear**]	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate **OS Awareness Manual**. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.

CPU specific tables in MMU.SCAN <table>

ITLB	Loads the ITLB translation table from the CPU to the debugger-internal translation table.
UTLB	Loads the UTLB translation table from the CPU to the debugger-internal translation table.

Memory Classes and Cache Handling

Memory Classes (SH2)

The following memory classes are available:

Memory Class	Description
P	Program
D	Data

Memory Classes (SH3, SH4, ST40)

The following memory classes are available:

Memory Class	Description
P	Program
D	Data
IC	Instruction Cache
DC	Data Cache
NC	No Cache (only physically memory)

If caching is disabled via the appropriate hardware registers, memory accesses to the memory classes IC or DC are realized by TRACE32-ICD as reads and writes to physical memory.

Memory Coherency

If data will be set to DC, IC, NC, D or P memory class, the Data-Cache, Instruction-Cache or physical memory will be updated.

	Data Cache	Instruction Cache	Physical Memory
write to **DC:**	updated	--	updated if write through mode
write to **IC:**	--	--	updated
write to **NC:**	--	--	updated
write to **D:**	updated	--	updated if write through mode
write to **P:**	--	--	updated

SYStem.Option.ICFLUSH Cache invalidation option

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

Default: ON. Invalidates the instruction cache before starting the target program (Step or Go). This is required if the CACHEs are enabled and software breakpoints are set to a cached location.

SYStem.Option.DCFREEZE Freeze data cache contents

not supported

SYStem.Option.DCCOPYBACK Cache copy back

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

forces a Cache Copy Back action in case of physical memory access (memory class **A:**).

This option should be switched ON if the data cache is configured for copyback mode. Before accessing physical memory the cache contents are copied back to target memory.

SYStem.Option.ICREAD Cache read option

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

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

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

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

The following table describes how DCREAD and ICREAD influence the behavior of the debugger commands that are used to display memory.

	DC:	**IC:**	**NC:**	**D:**	**P:**
ICREAD off DCREAD off	D-Cache	I-Cache	phys. mem.	phys. mem.	phys. mem.
ICREAD on DCREAD off	D-Cache	I-Cache	phys. mem.	phys. mem.	I-Cache
ICREAD off DCREAD on	D-Cache	I-Cache	phys. mem.	D-Cache	phys. mem.
ICREAD on DCREAD on	D-Cache	I-Cache	phys. mem.	D-Cache	I-Cache

Trace

Analysis of the program history is supported in different ways.

FIFO Trace (SH2A, SH3, SH4, ST40)

This CPUs includes a 8-stage branch trace. This trace holds the source and destination address of the last eight 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!

Analysis of the program history is supported in different ways.

SYStem.Option.FIFO — FIFO trace configuration

SH4, ST40, SH7705, SH7294

Format:	**SYStem.Option.FIFO** *<mode>*
<mode>:	**OFF** **eXception** **Subroutine** **ALL**

Selects the kind of program-flow-change which should be traced in FIFO trace mode.

OFF	FIFO disabled
eXception	trace on exceptions, interrupts and RTE instructions
Subroutine	trace on exceptions, interrupts and on RTE, BSR, BSRF, JSR, RTS instructions
ALL	trace any change in program flow

LOGGER Trace (SH4, ST40, SH7705)

This method offers a much deeper trace than the FIFO method with the disadvantage of being time and target memory intrusive.

The SH4 branch trace is configured to generate a TRACE-exception after one/six valid branch trace entries. Program is stopped then, the branch trace contents are copied to a predefined area in user memory and finally the program is restarted.

The following script should be used to initialize the LOGGER-Trace. For further details please refer to the LOGGER online help or training manuals.

Run this script after(!) initialization of target memory.

```
logger.mode create on          ; enable automatic Logger-Structure
                               ; generation

logger.mode flowtrace all      ; define the kind of program-flow-changes
                               ; to be traced

logger.address 0ac020000       ; define startaddress of trace in user
                               ; memory

logger.size 512.               ; define trace depth (number of records)

logger.timestamp.up            ; define count direction of timestamp

logger.timestamp.rate          ; define frequency of timestamp counter
100000000.

logger.init                    ; enable Logger
```

The influence on runtime depends on the target program. With fewer changes in program flow the runtime relation between target-program to logger-trace-program becomes better. With estimated program-flow-changes every five instructions the complete runtime will increase about x5.

NOTE: CPU internal WatchDogTimer are stopped during logger-trace-program execution!

The required target memory size can be calculated this way:

Logger-Memory-Size = 32 Byte + (Logger.Size x 16 Byte)

AUD-Trace (SH2A, SH4, ST40)

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 selectiv 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.

Format:	**SYStem.Option.AUD8** [**ON**	**OFF**]

This option informs the TRACE32 software to use the AUD 8bit algorithm to reconstruct the program flow.

Default setting is OFF (4-bit mode).

See also application note: **Enable 8-bit AUD Trace Interface of SH4-202**

AUD-Trace (SH3)

The AUD trace interface of the SH3 family supports the branch 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.

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. Frequency of clock generator divided by 1,2,4 or 8.

The preprocessor of the SH-AUD trace contains a clock generator circuit which easily can be changed to fit for your application.

The maximum frequency of AUDCK is that of the CPU clock or less. Furthermore it must be less then 100 MHz!

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

On-chip Trace SH2A

Some of the SH2A core devices are equipped with an onchip trace buffer. Depending on the device in use it can cover up to 1024 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 (SH2A, SH4, ST40)**

The onchip trace supports tracing of the M-Bus and/or I-Bus activity. The I-Bus-Master flags can be displayed in the Trace.List window with command:

Onchip.List IADMA IDMA ICPU def

Trigger and trace control on **I-Bus** activity is enabled by setting a breakpoint with option /Alpha, /Beta, /Charly or /Delta. The /Alpha, /Beta, /Charly or /Delty activity has to be defined in the **Trigger Onchip** window (**TrOnchip.A.IBUS**). Two onchip breakpoints can be used for I-Bus trigger and trace control. There is only one I-Bus breakpoint available if I-Bus **and** M-Bus tracing is enabled.

Onchip.Mode.MBusTrace — Mbus trace enable

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

Default: ON

Enables tracing of the MBus activity (ProgramTrace, DataReadTrace and DataWriteTrace).

Onchip.Mode.IBusTrace

Ibus trace enable

Format:	**Onchip.Mode.IBusCpuTrace** [**ON** | **OFF**]
Format:	**Onchip.Mode.IBusDmaTrace** [**ON** | **OFF**]
Format:	**Onchip.Mode.IBusAdmaTrace** [**ON** | **OFF**]

Default: OFF

Enables tracing of the I-Bus activity (CPU-, DMA-, ADMA-busmaster).

NOTE:	**If tracing of M-Bus and I-Bus activity is enabled, the onchip trace buffer is split. Each bus can be traced with a maximum of TraceBufferSize/2 records.**

Onchip.Mode.ProgramTrace

Program flow trace enable

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

Default: ON

Enables tracing of program flow activity of the M-Bus.

Onchip.Mode.DataReadTrace

Data read trace enable

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

Default: OFF

Enables read-cycle tracing of the enabled busses (M-Bus and/or I-Bus). This setting is ignored if selective trace (TraceEnable) is active.

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

Default: OFF

Enables write-cycle tracing of the enabled busses (M-Bus and/or I-Bus). This setting is ignored if selective trace (TraceEnable) is active.

On-chip Performance Analysis (SH4, ST40)

The SH4/ST40-Core supports two performance counters. This counters can be configured to count a wide range of different events.

TrOnchip.PMCTRx

Performance counter configuration

Format:	**TrOnchip.PMCTRx** *<mode>*

<mode>	**function**	**count/time measurement**
Init	Clear performance counter	
OARC	Operand Access Read with Cache	count
OAWC	Operand Access Write with Cache	count
UTLBM	UTLB Miss	count
OCRM	Operand Cache Read Miss	count
OCWM	Operand Cache Write Miss	count
IFC	Instruction Fetch with Cache (*2)	count
ITLBM	Instruction TLB Miss	count
ICM	Instruction Cache Miss	count
AOA	All Operand Access	count
AIF	All Instruction Fetch (*2)	count
OROA	On-chip RAM Operand Access	count
OIOA	On-chip I/O Access	count
OA	Operand Access with Cache	count
OCM	Operand Cache Miss	count
BI	Branch Instruction Issued	count
BT	Branch Instruction Taken	count

SRI	Subroutine Instruction Issued	count
II	Instruction Issued	count
2II	Two Instructions Issued	count
FPUI	FPU Instruction Issued	count
INT	Interrupt Normal	count
NMI	Interrupt NMI	count
TRAPA	TRAPA Instruction	count
UBCA	UBC A Match	count
UBCB	UBC B Match	count
ICF	Instruction Cache Fill	time
OCF	Operand Cache Fill	time
TIME	Elapsed Time	time
PFCMI	Pipeline Freeze by Cache Miss Instruction	time
PFCMD	Pipeline Freeze by Cache Miss Data	time
PFBI	Pipeline Freeze by Branch Instruction	time
PFCPU	Pipeline Freeze by CPU Register	time
PFFPU	Pipeline Freeze by FPU	time

Runtime Measurement

The SH 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	GND
TDO	5	6	GND
ASEBRK-	7	8	N/C
TMS	9	10	GND
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	SHx: **TCK** ST40: **DCLK**
/TRST	Jtag-TRST, output of debugger	**TRST**#
TDO	Jtag-TDO, input for debugger	TDO
/ASEBRK	Break Acknowledge, input/output for debugger	SH4: **ASEBRK,BRKACK** SH3: **/ASEBRKAK** SH2: **/ASEBRKAK** ST40: **/ASEBRK,BRKACK**
/RESET	RESET input/output for debugger	SH4: **/RESET** SH3: **/RESETP** SH2: **/RES** ST40: **/RST**
/DebugMode	CPU debug mode enable GND-output of debugger	SH4: GND (not used) SH3: **/ASEMD0** SH7047: **/DBGMD** ST40: GND (not used)