

# V850 Debugger and Trace

Release 02.2025



# V850 Debugger and Trace

### **TRACE32 Online Help**

**TRACE32 Directory** 

**TRACE32 Index** 

TRACE32 Documents		
ICD In-Circuit Debugger	Ø	
Processor Architecture Manuals		<u></u>
V850		
V850 Debugger and Trace		1
		6
Brief Overview of Documents for New U	Jsers	6
Demo and Start-up Scripts		6
Warning		8
Application Note		9
Location of Debug Connector		9
Reset Line		9
FLMD0 Line		10
Mask-Options of V850/Fx3, Cargate	•	11
Quick Start JTAG		12
Troubleshooting		<b>15</b> 15
FAQ		15
Configuration System Overview		<b>16</b> 16
•		
SYStem.CONFIG.state		<b>17</b> 17
SYStem.CONFIG.State SYStem.CONFIG	. , , ,	17 18
Daisy-Chain Example		20
TapStates		20 21
SYStem.CONFIG.CORE		21 22
SYStem.CONFIG.EXTWDTDIS	-	 23
SYStem.CONFIG.DEBUGPORTTYPE	_	- 24
SYStem.CONFIG.PortSHaRing		24
SYStem.CPU	3	25
SYStem.JtagClock		25
SYStem.LOCK		25

SYStem.MemAccess	Select run-time memory access method
SYStem.Mode	System mode selection
SYStem.Option.IMASKASM	Interrupt disable
SYStem.Option.IMASKHLL	Interrupt disable
SYStem.Option.PERSTOP	Disable CPU peripherals if stopped
SYStem.RESetOut	Reset target without reset of debug port
Exception Lines Enable	
SYStem.Option.RESET	Reset line enable
SYStem.Option.STOP	Stop line enable
SYStem.Option.WAIT	Wait line enable
SYStem.Option.REQest	Request line enable
SYStem.Option.NMI0	NMI0 line enable
SYStem.Option.NMI1	NMI1 line enable
SYStem.Option.NMI2	NMI2 line enable
SYStem.Option.CPINT	CPINT line enable
•	
SYStem.Option.BTM	Branch trace message
SYStem.Option.DTM	Data trace message
SYStem.Option.KEYCODE	Keycode
SYStem.Option.OPWIDTH	Trace interface width
SYStem.Option.STALL	Trace STALL mode
SYStem.Option.TCMODE	Trace clock mode
•	
Software Breakpoints	
On-chip Breakpoints	
Breakpoint in ROM	
Example for Breakpoints	
•	
TrOnchip.CONVert	Adjust range breakpoint in on-chip resource
TrOnchip.RCU	ROM-Correction breakpoints
TrOnchip.RESet	Set on-chip trigger to default state
TrOnchip.Set.Alignment	Alignment error breakpoints
TrOnchip.Set.MissAlign	Alignment error breakpoints
TrOnchip.SIZE	Trigger on byte, word, long memory accesses
TrOnchip.state	Display on-chip trigger window
TrOnchip.VarCONVert	Adjust complex breakpoint in on-chip resource
•	051/4
CPU.BASEFAMILY()	CPU family
CPU.SUBFAMILY()	CPU subfamily
_	
DataFlash: Memory Class	

V850 Debugger and Trace | 3

NBD Interface	43
Runtime Measurement	44
JTAG Connector	45
Connector 20 pin 100mil /NWire	45
Trace Connector	46
Connector MICTOR/N-Wire and Trace	46
Connector KEL/N-Wire and Trace	47
NBD Connector	48

# V850 Debugger and Trace

Version 13-Feb-2025

### Introduction

This document describes the processor specific settings and features for NEC V850E(S). TRACE32-ICD supports all V850 devices which are equipped with the N-wire debug interface.

Please note that only the **Processor Architecture Manual** (the document you are currently reading) is specific to the core architecture. All other parts of the online help are general and independent of any core architecture. Therefore, if you have questions related to the core architecture, the **Processor Architecture Manual** should be your primary reference.

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 family lines, the name(s) of the family/families is/are added in brackets.

#### **Brief Overview of Documents for New Users**

#### **Architecture-independent information:**

- "Debugger Tutorial" (debugger\_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- "General Commands" (general\_ref\_<x>.pdf): Alphabetic list of debug commands.
- "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.

#### **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**.

### **Demo and Start-up Scripts**

Lauterbach provides ready-to-run start-up scripts for known V850 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.



# Warning

### Signal Level

The debugger output voltage follows the target voltage level. It supports a voltage range of 0.4 ... 5.2 V.

#### **ESD Protection**

#### WARNING:

To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF.

Recommendation for the software start:

- 1. Disconnect the Debug Cable from the target while the target power is off.
- Connect the host system, the TRACE32 hardware and the Debug Cable.
- 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:

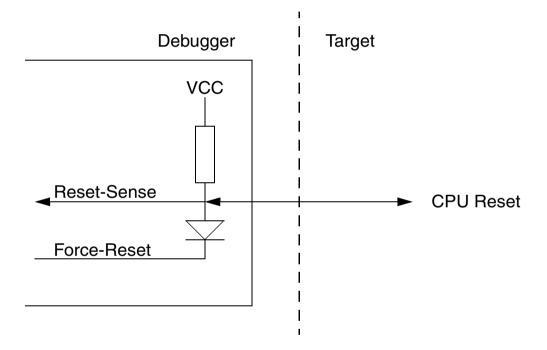
- 1. Switch off the target power.
- 2. Disconnect the Debug Cable from the target.
- Close the TRACE32 software.
- 4. Power OFF the TRACE32 hardware.

# **Location of Debug Connector**

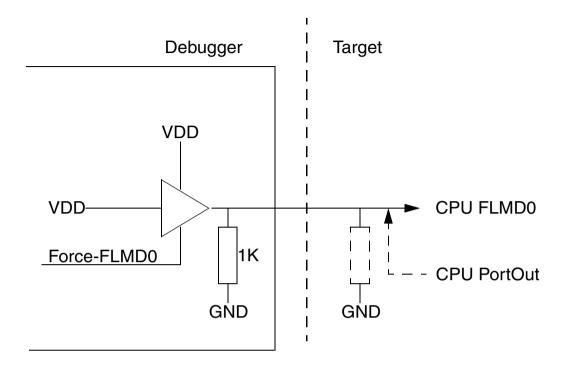
Locate the debug connector as close as possible to the processor to minimize the capacitive influence of the trace length and cross coupling of noise onto the JTAG signals.

### **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  $\overline{\text{RESET}}$ .



The debugger forces this line to VDD to enable flash programming.



### Mask-Options of V850/Fx3, Cargate

The mask options require a special handling. In normal operation mode the mask option values are located in flash memory at address 0x7A, 0x7B. In emulation mode these values have to be copied to a certain debug register EMUMO at address 0xFFFFF9FA.

- the value of address 0x7A has to be copied to the low byte of EMUMO
- the value of address 0x7B has to be copied to the high byte of EMUMO

The new options become active at the next SYStem.UP.

Add following startup sequence to your script:

```
SYStem.Up
                         ; initial startup
disable RomSecurityUnit ; see demo scripts
; set MaskOptions to EMUMO register
Data.Set 0xFFFFF9FA %Word 0x0800
SYStem.Up
                        ; now the MaskOption settings are active
disable RomSecurityUnit ; see demo scripts
. . .
```

### **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 TBACE32 software was started.

В:

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

SYStem.CPU 70F3281

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

JTAG Debugger for V850

V850SA

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

```
MAP.BOnchip 0x00000000++0x7FFFF
```

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...

Following command sequence is required for CPU types which are equipped with a ROM Security Unit (RSU). As long as the ROM Security is active the debugger gets no access to CPU memory.

This example estimates 0xff as memory content at address 0x70 ... 0x79.

```
; BROM switching
Data.Set 0xfffff8d0 %Byte 0xa5
Data.Set 0xfffff8d4 %Byte 0x08
Data.Set 0xfffff8d4 %Byte 0xf7
Data.Set 0xfffff8d4 %Byte 0x08
; KeyCode setting
; data at 0x70 ... x79 is estimated as 0xff
Data.Set 0xfffff9c0 %Word 0xffff 0xffff
Print DATA.LONG(D:0x70)
Data.Set 0xfffff9c0 %Word 0xffff 0xffff
Print DATA.LONG(D:0x74)
Data.Set 0xfffff9c0 %Word 0x0000 0xffff
Print DATA.LONG(D:0x78)
Print DATA.LONG(D:0xfffff9c4)
```

#### 6. Load the program.

```
Data.LOAD.ubrof sieve.d85
                                 ; (ubrof specifies the format,
                                 ; sieve.d85 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 0x000000++0x07ffff
                                  ; Specify where's FLASH/ROM
SYStem.CPU 70F3281
                                  ; Select the processor type
SYStem.Up
                                  ; Reset the target and enter debug
                                  ; mode
Data.Load.ubrof sieve.d85
                                  ; 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 on-chip breakpoint to address
                                  ; 1000 (address 1000 is in FLASH)
                                  : (Refer to the restrictions in
                                  ; On-chip Breakpoints.)
Break.Set 0x3FFB100 / Program
                                  ; Set software breakpoint to address
                                  ; 3FFB100 (address 3FFB100 is in RAM)
```

<sup>\*)</sup> These commands open windows on the screen. The window position can be specified with the WinPOS command.

# **Troubleshooting**

# **SYStem.Up Errors**

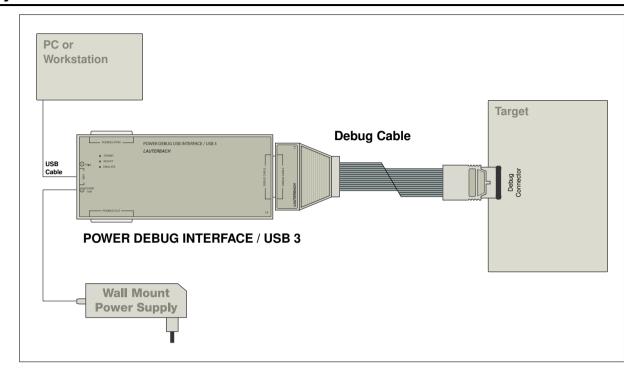
The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

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 <b>SYStem.Up</b> .
All	There are additional loads or capacities on the JTAG lines.
All	The JTAG clock is too fast.

# **FAQ**

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

# **System Overview**



### 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></tab>	Opens the <b>SYStem.CONFIG.state</b> 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** cparameter> <number\_or\_address>

**SYStem.MultiCore** cated)

<parameter>: CORE <core>

DRPRE <br/>
<br/>
hits> <parameter>: (JTAG): DRPOST <bits>

> IRPRE <br/>
> <br/> IRPOST <hits> **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

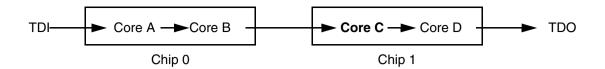
connected by a JTAG joiner adapter to access a single JTAG chain.

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

# **TapStates**

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 nongeneric 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.EXTWDTDIS < option>

<option>: OFF

High Low

HighwhenStopped LowwhenStopped

Default for Automotive/Automotive PRO Debug Cable: High.

Default for XCP: OFF.

Controls the WDTDIS pin of the debug port. This configuration is only available for tools with an Automotive Connector (e.g., Automotive Debug Cable, Automotive PRO Debug Cable) and XCP.

**OFF** The WDTDIS pin is not driven. (XCP only)

**High** The WDTDIS pin is permanently driven high.

**Low** The WDTDIS pin is permanently driven low.

**HighwhenStopped** The WDTDIS pin is driven high when program is stopped (not XCP).

**LowwhenStopped** The WDTDIS pin is driven low when program is stopped (not XCP).

Format: SYStem.CONFIG.DEBUGPORTTYPE JTAG

<port type>: JTAG

It specifies the used debug port type. It assumes the selected type is supported by the target.

# SYStem.CONFIG.PortSHaRing Control sharing of debug port with other tool

Format: SYStem.CONFIG.PortSHaRing [ON | OFF | DownState <downmode> |

**CPUAccEvt** < number>

<downmode>: RESET | TRISTATE

<number>: **0..8** 

Configures if the debug port is shared with another tool, e.g., an ETAS ETK or ETKX. This option is only available if an motive Debug Cable is connected to the PowerDebug module.

**ON** Request for access to the debug port and wait until the access is granted

before communicating with the target.

**OFF** Communicate with the target without sending requests.

**DownMode** Select the mode of the reset signal when TRACE32 is in **SYStem.Down** 

mode.

**CPUAccEvt** Defines the maximum number of TriggerEventBreakpoints reserved for

TRACE32 usage.

Default = 8. Only relevant for data-access breakpoints.

Reduce the number if the chip internal TriggerEventUnit has to be shared

with other tools.

The current setting can be obtained by the **PORTSHARING()** function, immediate detection can be performed using **SYStem.DETECT.PortSHaRing**.

Format: SYStem.CPU <cpu>

*<cpu>*: **70F3143** | **70F3186** ...

Default selection: V850SA. Selects the CPU type.

# SYStem.JtagClock

JTAG clock selection

Format: SYStem.JtagClock [<frequency>]

SYStem.BdmClock [<frequency>] (deprecated)

Default frequency: 1 MHz.

Selects the JTAG port frequency (TCK). Any frequency up to 25 MHz can be entered, it will be generated by the debuggers internal PLL.

For CPUs which come up with very low clock speeds it might be necessary to slow down the JTAG frequency. After initialization of the CPUs PLL the JTAG clock can be increased.



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

### SYStem.LOCK

Lock and tristate the debug port

Format: SYStem.LOCK [ON | OFF]

Default: OFF.

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

Format: SYStem.MemAccess < mode>

<mode>: QUICK

NBD Denied StopAndGo

Selects the method for memory access while the core is running.

All debugger windows which are opened with the option **/E** will use the selected Non-intrusive memory access.

**QUICK** Does a pseudo real-time access. For each single memory access the

application is interrupted for about 50 CPU clocks (10 MHz --> 5 us interruption). This method can only be used if  $\bf NO$  breakpoints are set. The JTAG clock speed should be as fast as possible to get good performance.

NBD Requires extra debugger hardware to handle the CPUs NBD-interface.

This interface allows a Non-intrusive memory access while the core is

running.

**Denied** Disables any memory access while core is running.

**StopAndGo** Temporarily halts the core(s) to perform the memory access. Each stop

takes some time depending on the speed of the JTAG port, the number of

the assigned cores, and the operations that should be performed.

Format: SYStem.Mode < mode>

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

<mode>: Down

NoDebug

Go Up

**Down** Disables the Debugger.

**NoDebug** Disables the Debugger. The debug interface is forced to high impedance

mode.

**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.

StandBy Not supported.

# 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.

Format:

SYStem.Option.IMASKHLL [ON | OFF]

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

# SYStem.Option.PERSTOP

Disable CPU peripherals if stopped

Format:

SYStem.Option.PERSTOP [ON | OFF]

Stop CPU peripherals if program is stopped. Useful to prevent timer exceptions. Only supported for V850/E2 cores.

### 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**.

# **Exception Lines Enable**

The V850 supports disabling of several CPU pins. This can be very useful to prevent watchdog resets or external NMI sources.

# SYStem.Option.RESET

Reset line enable

Format:

SYStem.Option.RESET [ON | OFF]

Enable/Disable Reset line.

Default: ON

# SYStem.Option.STOP

Stop line enable

Format:

SYStem.Option.STOP [ON | OFF]

Enable/Disable Stop line.

Default: ON

# SYStem.Option.WAIT

Wait line enable

Format:

SYStem.Option.WAIT [ON | OFF]

Enable/Disable Wait line.

Default: ON

Format:

SYStem.Option.REQ [ON | OFF]

Enable/Disable Request line.

Default: ON

# SYStem.Option.NMI0

NMI0 line enable

Format:

SYStem.Option.NMI0 [ON | OFF]

Enable/Disable NMI0 line.

Default: ON

# SYStem.Option.NMI1

NMI1 line enable

Format:

SYStem.Option.NMI1 [ON | OFF]

Enable/Disable NMI1 line.

Default: ON

# SYStem.Option.NMI2

NMI2 line enable

Format:

SYStem.Option.NMI2 [ON | OFF]

Enable/Disable NMI2 line.

Default: ON

SYStem.Option.CPINT [ON | OFF] Format:

Enable/Disable CPINT line.

Default: ON

# SYStem.Option.BTM

### Branch trace message

Format: SYStem.Option.BDM < mode>

<mode>: ON

OFF MIN MAX

Select type of recorded branch trace messages:

**OFF** Program flow trace is disabled.

**MAX** Trace any branch-type, except "non-taken-conditional-branches".

ON (Default) like MAX but for "taken-direct-branches" only the branch-

source-address is recorded.

MIN Like ON but "unconditional-branches" are not recorded.

Format: SYStem.Option.DTM < mode>

<mode>: OFF

Read Write ReadWrite

Select type of recorded data trace messages:

**OFF** Data trace is disabled.

**Read** Read-cycles are recorded'.

Write Write-cycles are recorded'.

**readWrite** Read- and Write-cycles are recorded'.

### SYStem.Option.KEYCODE

Keycode

Format: **SYStem.Option.KEYCODE** [<12x 8bit values>]

Has to be the same value as present in CPUs ID-code input registers ID\_IN[0..2].

This command is only relevant for devices with V850E2 CPU core.

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. If bit-95 of the target KEYCODE is programmed to "0" then debug control can not be enabled even if the KEYCODE values match. More details on ID-Code-Protection can be found in the CPU-Users-Manual.

Attention: TRACE32 uses a different byte-order of the KEYCODE values than used by the Renesas Flash Programmer (RFP).

RFP order: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C

TRACE32 order: 0x04 0x03 0x02 0x01 0x08 0x07 0x06 0x05 0x0C 0x0B 0x0A 0x09

Format:	SYStem.Option.OPWIDTH < mode>
<mode>:</mode>	4 8 16

Selects the number of data channels of the trace interface.

Format: SYStem.Option.TCMODE < mode>

<mode>: ON | OFF

Selects Trace STALL mode.

**ON** Program execution might be stalled to prevent overrun of trace interface.

**OFF** Program execution is done in real-time. The trace interface might loose

trace messages.

# SYStem.Option.TCMODE

Trace clock mode

Format: SYStem.Option.TCMODE < mode>

<mode>: 1/1

1/2

**1/2DDR** 

Selects Trace clockspeed.

1/1 Trace clock is equal to CPU system clock.

1/2 Trace clock is equal to CPU system clock / 2.

1/2DDR Not supported.

# **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	Address Breakpoints	Data Breakpoints	Sequential Breakpoints
V850E(S) all devices	2 ranges - include or exclude Qualifier for: - Instruction-Fetch - Data-Read - Data-Write - Size ANY/8/16/32	2 ranges - include or exclude	A->B
V850E(S) devices with ROM Correction Unit (RCU)	4 or 8 additional breakpoints on - Instruction-Fetch		
Only in Flash area - requires onchip break mapping MAP.BOnchip <range> - can be disabled with command TO.RCU OFF</range>			

#### **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:

```
; On-chip Breakpoint 1
Break.Set 0x100 /Program
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 on-chip breakpoints can only cover specific ranges. If a range cannot be programmed into the breakpoint, it will automatically be converted into a single address breakpoint when this option is active. This is the default. Otherwise an error message is generated.

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

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

Format: TrOnchip.RCU [ON | OFF]

When enabled (default) the CPU's Rom-Correction-Unit is used to extend the number of Onchip Breakpoints. RCU breakpoints can only be used for program breaks in the FLASH area.

NOTE:

A DBTRAP instruction code is visible at the break address. It is visible for program and data accesses, which causes trouble if the application does memory checking like CRC.

#### **TrOnchip.RESet**

Set on-chip trigger to default state

Format: TrOnchip.RESet

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

#### **TrOnchip.Set.Alignment**

Alignment error breakpoints

Format: TrOnchip.Set.Alignment [ON | OFF]

When enabled (default) the CPU stops program execution on any miss-aligned memory access.

NOTE: Miss-aligned memory accesses are supported by the V850-ES core. The

TrOnchip.Set.Alignment should be set to OFF.

Format: TrOnchip.Set.MissAlign [ON | OFF]

When enabled (default) the CPU stops program execution on miss-align stack operations and on miss-align accesses in "miss-align access disable mode".

NOTE: Miss-aligned memory accesses are supported by the V850-ES core. The

TrOnchip.Set MissAlign should be set to OFF.

#### **TrOnchip.SIZE**

Trigger on byte, word, long memory accesses

TrOnchip.SIZE [ON | OFF] Format:

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.state**

Display on-chip trigger window

Format: TrOnchip.state

Opens the **TrOnchip.state** window.

#### TrOnchip.VarCONVert

Adjust complex breakpoint in on-chip resource

TrOnchip.VarCONVert [ON | OFF] (deprecated) Format:

Use Break.CONFIG.VarConvert instead

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

## **CPU specific Functions**

#### CPU.BASEFAMILY()

**CPU** family

[build 70426 - DVD 02/2016]

Syntax:

CPU.BASEFAMILY()

Returns the CPU family name "V850".

**Return Value Type: String.** 

#### CPU.SUBFAMILY()

**CPU** subfamily

[build 68566- DVD 02/2016]

Syntax:

CPU.SUBFAMILY()

Returns the CPU subfamily name.

Return Value Type: String.

### **Memory Classes**

The following memory classes are available:

Memory Class	Description
Р	Program
D	Data (also DataFlash without ID tag)
DF	DataFlash with ID-Tag: Memory contents are presented as 64bit value Data: bit [310] ID tag: bit [32]

### **DataFlash: Memory Class**

By default the DataFlash is handled like a normal 32bit flash memory, the ID-Tag is ignored. The contents are presented as 32bit values with addresses counting up 0x0, 0x4, 0x8, 0xC ... (use the command: **Data.dump D:**<address>/Long).

The presentation of the additional ID tag bit require slight changes in the display.

By using the **DF**: memory class the ID tag is handled like an additional databit, so the **Data.dump** window shows 64bit values, whereas the address counting is still 0x0, 0x4, 0x8, 0xC ... (use the command **Data.dump DF**:<address>/Quad).

Because of the 64bit presentation, a **Data.Save** *<file>* **DF**: *<addressrange>* command will save double of data than defined by the address range. Also the download of data flash contents with ID tag requires double of data than defined by the address range (**Data.Load** *<file>* **DF**: *<address>*).

### **NBD** Interface

The usage of NBD (Non Break Debug Interface) requires extra debug hardware to get access to the CPUs NBD interface. This extra hardware is plugged in between the debug box and the debug dongle. Connection to the CPUs NBD interface is done by a 16pin flat cable.

The interface allows real-time access to target memory while the application program is running. Furthermore it allows the access to certain debug configuration registers to:

- Replace CPU internal FLASH by RAM in blocks of 4 KByte
- Activate the NBD\_TRIGGER signal on access to certain memory locations
- Readout the CPUs ID-code

The NBD configuration registers are accessible in the CPUs peripheral window.

### **Runtime Measurement**

Runtime measurement is done with about 5 µs 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.

# Connector 20 pin 100mil /NWire

Signal	Pin	Pin	Signal
GND	1	2	DCK
GND	3	4	DMS
GND	5	6	DDI
GND	7	8	DRST-
GND	9	10	PORT0IN
GND	11	12	RESET-
GND	13	14	FLMD0
GND	15	16	PORT1IN/RDYZ
GND	17	18	DDO
GND	19	20	VDD

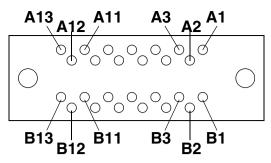
JTAG Connector	Signal Description	CPU Signal
DMS	JTAG-TMS, output of debugger	TMS
DDI	JTAG-TDI, output of debugger	TDI
DCK	JTAG-TCK, output of debugger	TCK
TRST	JTAG-TRST, output of debugger	TRST
DDO	JTAG-TDO, input for debugger	TDO
RESET	RESET input/output of debugger - Force target Reset - Sense target Reset (see application note)	RESET
FLMD0	FLASH Mode0 signal - enable flash programming (see application note)	FLMD0
PortIn0	Input Port for Debugger, currently unused	not connected
PortIn1/RDYZ	READY- input of debugger, only used for E2 core CPUs like Px4	RDYZ

## **Trace Connector**

The default connection for trace support is **MICTOR**. With additional adaptors also **KEL** and **GlenAir** is supported.

### **Connector MICTOR/N-Wire and Trace**

Signal	Pin	Pin	Signal
GND	1	2	GND
DCK	3	4	VDD
DMS	5	6	DRST-
DDI	7	8	RESET-
DDO	9	10	FLMD0
N/C	11	12	RESERVED
N/C	13	14	RESERVED
N/C	15	16	PORT1IN
TRCCLK	17	18	PORT2IN
TRCEND	19	20	TRCCE
TRCDATA0	21	22	TRCDATA8
TRCDATA1	23	24	TRCDATA9
TRCDATA2	25	26	TRCDATA10
TRCDATA3	27	28	TRCDATA11
TRCDATA4	29	30	TRCDATA12
TRCDATA5	31	32	TRCDATA13
TRCDATA6	33	34	TRCDATA14
TRCDATA7	35	36	TRCDATA15
GND	37	38	GND



## **Top View**

Pin Number	Signal Name	Input/Output (User Side) (User Side)	
A1	CLKOUT	Output 22 33 Ω series resistor (recommende	
A2	TRCDATA0	Output	22 33 $\Omega$ series resistor (recommended)
A3	TRCDATA1	Output	22 33 $\Omega$ series resistor (recommended)
A4	TRCDATA2	Output	22 33 $\Omega$ series resistor (recommended)
A5	TRCDATA3	Output	22 33 $\Omega$ series resistor (recommended)
A6	TRCEND	Output	$22 \dots 33 \Omega$ series resistor (recommended)
A7	DDI	Input	10 kΩ pull-up
A8	DCK	Input	10 kΩ pull-up
A9	DMS	Input	10 kΩ pull-up
A10	DDO	Output	22 33 $\Omega$ series resistor (recommended)
A11	DRST	Input	10 kΩ pull-up
A12	RESET	Input	10 kΩ pull-up
A13	FLMD0	Input	open
B1 B10	GND	-	Connection to the power GND
B11	Port0_In	-	Open
B12	Port1_IN	-	Open
B13	+ 3.3 V	-	Connection to the power

Signal	Pin	Pin	Signal
TRIG-	1	2	VCC
OUT-	3	4	GND
CLK	5	6	GND
SYNC	7	8	GND
DATA0	9	10	GND
DATA1	11	12	GND
DATA2	13	14	DATA3
MODE	15	16	RESETO-

NBD Connector	Signal Description	CPU Signal
TRIG	NBD_Trigger signal, debugger input	TRIG_DBG
OUT	NBD_DataDirection signal, debugger output	usually not used
	A LOW indicates direction Interface> CPU	
CLK	NBD_Clock, debugger output	CLK_DBG
SYNC	NBD_SYNC signal, debugger output	SYNC_DBG#
DATA[3 0]	NBD_DATA[3 0], debugger input/output	AD[3 0]_DBG
MODE	NBD_Mode enable, debugger output	MODE_NBD
RESETO	NBD_ResetOut signal, debugger input	RESETO_DBG
	Indicates any kind of Reset forced to the CPU	
VCC	Reference Voltage for NBD Interface (2 5 V) debugger input	PowerSupply of user system