

# StarCore Debugger and Trace





Release 02.2025

MANUAL

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .....	
ICD In-Circuit Debugger .....	
Processor Architecture Manuals .....	
StarCore .....	
StarCore Debugger and Trace .....	1
Introduction .....	6
Brief Overview of Documents for New Users	6
Demo and Start-up Scripts	6
Warning .....	8
Quick Start .....	9
Troubleshooting .....	12
SYStem.Up Errors	12
Memory Access Errors	13
NEXUS Flow Errors and FIFO Overflow Messages	14
FAQ .....	15
Configuration .....	16
CPU specific SYStem Settings and Restrictions .....	17
SYStem.CLOCK	Setup core clock 17
SYStem.CONFIG.state	Display target configuration 17
SYStem.CONFIG	Configure debugger according to target topology 18
Daisy-Chain Example	20
TapStates	21
SYStem.CONFIG.CORE	Assign core to TRACE32 instance 22
SYStem.CPU	Select the used CPU 23
SYStem.LOCK	Lock and tristate the debug port 23
SYStem.MemAccess	Select run-time memory access method 23
SYStem.Mode	Establish the communication with the target 25
SYStem.Option.BASE	Sets the SUI base address 25
SYStem.Option.DCFLUSH	Data cache flush before step/run 26
SYStem.Option.DTM	Enables data trace messages 26
SYStem.Option.EnReset	Allow the debugger to drive nRESET/nSRST 26
SYStem.Option.EnTrst	Allow debugger to drive TRST 27

SYStem.Option.HalfRate	Enable Nexus DDR mode	27
SYStem.Option.ICFLUSH	Instruction cache flush before step/run	28
SYStem.Option.IMASKASM	Disable interrupts while single stepping	28
SYStem.Option.IMASKHLL	Disable interrupts while HLL single stepping	28
SYStem.Option.IPLDI	Sets interrupt mask strategy	29
SYStem.Option.LittleEnd	Switches between endian modes	29
SYStem.Option.MCKO	Nexus output clock ratio	29
SYStem.Option.MPU	MPU disabled	30
SYStem.Option.Nexus	Nexus port width	30
SYStem.Option.OCEBASE	Base address for OnCE registers	30
SYStem.Option.OCECORE	OnCE selection	31
SYStem.Option.OVC	Trace message overrun control	31
SYStem.Option.PTM	Enables program trace messages	31
SYStem.Option.SAMPLE	Adjust NEXUS sample point	32
SYStem.Option.SLOWPOLL	Change timing of JTAG during runtime	32
SYStem.Option.SLOWRESET	Expand reset time for additional reset module	32
SYStem.Option.VBA	Set up VBA value for analysis	33
SYStem.Option.WaitReset	Halt the core after reset	34
SYStem.Option.WATCHDOG	Enable WATCHDOG	35
SYStem.JtagClock	Define JTAG clock	35
<b>CPU specific MMU Commands</b>		<b>39</b>
MMU.DUMP	Page wise display of MMU translation table	39
MMU.List	Compact display of MMU translation table	39
MMU.SCAN	Load MMU table from CPU	40
<b>BenchMarkCounter</b>		<b>41</b>
<b>TrOnchip</b>		<b>42</b>
TrOnchip	Control of on-chip resources	45
TrOnchip.CONVert	Automatically convert range to single address	46
TrOnchip.REGister	Shows custom on-chip trigger registers	46
TrOnchip.RESet	Set on-chip trigger to default state	46
TrOnchip.VarCONVert	Automatically convert range to single address	46
TrOnchip.state	Opens configure panel	47
<b>On-chip Trace</b>		<b>48</b>
Onchip.Mode	Select mode to control trace buffer and contents	48
Onchip.VTBA	Set the destination address of the onchip trace	49
<b>General Restrictions</b>		<b>50</b>
<b>Floating Point Formats</b>		<b>51</b>
<b>Integer Access Keywords</b>		<b>51</b>
<b>File I/O Support</b>		<b>52</b>
Metrowerks MSLIO Support		52

<b>JTAG Connection .....</b>	<b>53</b>
Mechanical Description of the 20-pin Debug Cable	53
Electrical Description of the 20-pin Debug Cable	54
JTAG Connector 14-pin	55
<b>Memory Classes .....</b>	<b>57</b>

The screenshot displays the StarCore Debugger and Trace interface, which is divided into three main sections:

- Registers (Top Left):** A window titled "Bsr" showing the state of 16 registers (R0-R15). Each register entry includes its address, value, name, and associated control signals. For example, R0 has value 00000000, R1 has 000005AC, and R15 has 00000000. Below the registers, a status bar shows various system flags like "OVE", "EXP", "E T", "C", "BEM", "DOVF", "1", "ILST", "O", "ILIN", "O", "I", "NMI", "SM", "O", "S", "S10", "O", "RM", "R", "DI", "O", "NMID", "O", "SLF", "1", "LF", "1000", "VF", "0000", "GP", "0010000".
- System Settings (Top Right):** A window titled "Bsys" containing configuration options for the debugger. It includes sections for "Mode" (Down, NoDebug, Go, Attach, StandBy, Up (StandBy), Up), "MemAccess" (CPU, Denied, Enable, Denied, Nonstop), "Option" (IMASKASM, IMASKHLL, LittleEnd, WATCHDOG, SLOWRESET), "JtagClock" (15.0MHz), and "CPU" (MSC8101).
- Assembly Code (Bottom):** A window titled "[B:d.I]" showing the assembly code for a program named "demo.c". The code is displayed in a table with columns for address/line, code, label, mnemonic, and comment. The code includes a function "sieve()" that initializes a sieve of Eratosthenes, using instructions like "move.l", "doensh3", "sub", "for", "nop", "if", "tst", and "bt".

# Introduction

---

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

- MSC8XXX, MSC711X, MXC91XXX from FreeScale
- TC1XXX, TC2XXX from StarCoreLLC
- SC1000, SC2000, SC3000 (Custom Chips with unknown peripherals)

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 families, the name(s) of the family(ies) is 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 StarCore 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/starcore/` subfolder of the system directory of TRACE32.

**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.



# Quick Start

---

Starting up the debugger is done as follows:

1. Select the device prompt for the ICD Debugger and reset the system.

```
b::  
  
RESet
```

The device prompt `B::` is normally already selected in the [TRACE32 command line](#). If this is not the case, enter `B::` to set the correct device prompt. The **RESet** command is only necessary if you do not start directly after booting the TRACE32 development tool.

2. Specify the CPU specific settings.

```
SYStem.CPU <cpu_type>
```

The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Set up data for electrical interface

```
SYStem.JtagClock <frequency>
```

Normally the default value is 1.0 MHz, but the it can be increased up to 80.0 MHz.

```
SYStem.Option.EnReset  
  
SYStem.Option.EnTrst  
  
SYStem.Option.WaitReset
```

4. Inform the debugger about read only and none-readable address ranges (ROM, FLASH).

```
MAP.DenyAccess  
  
MAP.NoDenyAccess <range>  
  
MAP.BOnchip <range>
```

The B(reak)on-chip information is necessary to decide where on-chip breakpoints must be used. On-chip breakpoints are necessary to set program breakpoints to FLASH/ROM. The sections of FLASH and ROM depend on the specific CPU and its chip selects. Accesses to invalid addresses can cause unrecoverable bus errors. To avoid bus errors from the debugger side use the subcommands of MAP to define inaccessible memory areas. Bus errors can be removed by executing SYStem.Up. Make sure that there isn't any TRACE32 window open which accesses to a inaccessible memory that is not masked out, otherwise the bus error can occur again.

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 memory and registers.

6. Configure chip according application.

Before loading binary data into the processor memory, the memory should be made writable for the debugger. Therefore processor configuration registers have to be set e.g. chip select register.

7. Load the program.

```
Data.LOAD.Elf program.elf /Verify      ; ELF specifies the format,  
                                         ; program.elf is the file name
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler. It is recommended to use the option /Verify that verifies all written data. This test discovers a problem with the electrical connection, wrong chip configurations or linker command file settings.

A detailed description of the **Data.LOAD** command and all available options is given in the “**General Commands Reference**”.

A typical start sequence for the MSC8101 is shown below. This sequence can be written to a PRACTICE script file (\*.cmm, ASCII format) and executed with the command **DO** <file>. Other sequences can be found in the ~/demo/ directory.

```

b::                                ; Select the ICD device prompt

WinClear                           ; Clear all windows

SYS.CPU MSC8101                     ; Select CPU

SYS.JC 15000000.                    ; Choose JTAG frequency

SYStem.Up                           ; Reset the target and enter debug mode

MAP.DENYACCESS                      ; Forbid any access to the memory in
                                   ; general

MAP.NODENYACCESS 0x00000000--0x0007FFFF

MAP.NODENYACCESS 0x00000000--      ; SRAM 515KB
0x0007FFFF ;SRAM 515KB            ; Allows access to a memory area

MAP.NODENYACCESS 0x00EFFF00--      ; Allows access to a memory area
0x00EFFFFF ;EOnce                 ;

MAP.NODENYACCESS 0x00F80000--      ; Allows access to a memory area
0x00F807FF ;ROM                   ;

MAP.NODENYACCESS                   ; Allows access to a memory area.
IOBASE()++13FFF; 60x BUS           ; IOBASE() is the base address of the
                                   ; internal memory space. It is derived
                                   ; from the EMR register bits 21-19.

MAP.BONCHIP 0x00F80000--           ; Specifies the program memory where
0x00F807FF ;ROM                   ; on-chip breakpoints must be used.

Data.LOAD.ELF demo_be.elf          ; Load the application, verify the
/VERIFY                            ; process

Go main                             ; Run and break at main()

List.Mix                           ; Open source window

Register.view /SpotLight           ; Open register window

Var.Local                          ; Open window with local variables

```

## 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 open the AREA window by the command “AREA”. This will show some further information. There are different type problem that may occur:

### Basic electrical problems with the JTAG interface

- The target has no power.
- The JTAG lines are not connected correctly.
- The pull-up resistor between the JTAG[VCCS] pin and the target VCC is too large.
- There are additional loads or capacities on the JTAG lines.
- There is logic added to the JTAG state machine:  
By default the debugger supports only one processor in one JTAG chain. If the processor is only one member of a JTAG chain the debugger has to be informed about the target JTAG chain configuration. Use the **SYS.CONFIG** to specify the position of the device in the JTAG-chain.
- The target is in reset:  
The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up. Therefore no external R-C combination or external reset controller is allowed.
- /TRST is connected with /RESET. If this is the case try **System.Option.WaitReset <time>**
- The TDO and/or RTCK line is not active while /RESET. **System.Option.WaitReset <time>** provides a workaround for this, but there can be unpredictable problems after using the option.

### Advanced problems

- The wrong CPU is selected or the multicore settings are wrong.
- The clock of the onchip-peripheral is not running or the core has no clock.
- The JTAG frequency is too high or no RTCK is available.
- The external bus is not released by the bus master. In this case it is possible to execute a **SYStem.Mode.Attach** command to attach to the core and wait until the bus becomes ready.

## Multi-Core configuration problems

- The most important option is **SYStem.CONFIG.Slave**. One debugger is the Master which must set SYStem.CONFIG.Slave to OFF. All Slaves must set this option to ON. To bring the system up, the Master needs to execute the command **SYStem.Mode.Up** first then the all slaves. Only the Master will assert the /RESET line and initialize the JTAG interface which is necessary for all further communication.
- If multiple debug boxes are used with the help of a JTAG-Join adaptor, consider the option **SYStem.CONFIG.TriState** (must be ON) and **SYStem.CONFIG.TCKLEVEL** (must be set to level of TCK line).
- The JTAG chain settings **IRPRE**, IRPOST, DRPRE and DRPOST must be set correctly according to the core position within the JTAG chain.

## AREA Diagnostics

- Open the AREA window and type “DIAG 0x16001” to see which JTAG devices are in the JTAG chain. If this JTAG chain analysis works fine, it can be assumed that the electrical conditions are correct and the JTAG tap controllers are working.
- By “DIAG 0x10008” the multicore configuration is show in the AREA window

## Memory Access Errors

---

After system up is completed successfully, data can be written to or read from memory. Trying to access memory not belonging to the memory map of the processor will be refused with the error message

```
no memory mapped at address          D:XXXXXXXX
```

and

```
bus error generated by CPU
```

When a unrecoverable bus error occurs the target processor has to be reset.



Advanced program flow analysis will not show valid results with Flow Errors or FIFO Overflow Messages. The concerning dialogs will show an indicator if they have processed with Flow Errors or FIFO Overflow Messages.

## Flow Errors

- Flow Errors can be internal processing errors of the TRACE32 software. The analysis of the NEXUS message is not clear in some rare situations. If Flow Errors occur the LAUTERBACH support has to be contacted in order to fine tune to software to detect this situations.
- Flow Errors can occur if the recorded data is damaged. Various reasons can lead to that problem. Most probably there are electrical problems with the probe or target. The probe can be fine tuned by the commands **SYSystem.Option.MCKO**, **SYSystem.Option.HalfRate**, **SYSystem.CLOCK** and **SYSystem.Option.Sample** to adjust the timing and bandwidth of the NEXUS signals as well as **Trace.Threshold** to adjust the signal level detection. The target can be often also tuned to produce better signals e.g. by set up the slew rate and driver strength of the pins.
- Due to the analysis uses also the disassembler and internal simulator, Flow Errors will also occur, if the code is damaged or has changed. Using a memory class VM: based analysis prevents from this problem as well as a post analysis in the simulator mode. The code and VBA register must exactly match with the conditions when the trace was recorded.

```
; VM: Based analysis

; Load the ELF-File into virtual RAM at Host and do all code patches
Data.Load.Elf TheElfFile.eld /VM
; Set the VM: based analysis
Trace.ACCESS VM:0x0
; Open the flow trace window
Trace.List
enddo

; Simulator based analysis
; Setup Simulator
System.CPU MyCPU
System.Up
; Load the ELF-File and do all code patches
Data.Load.Elf TheElfFile.eld
; Restore the VBA address (used for interrupt detection)
Register.Set.VBA 0x0
; Load Stored Trace Data
Trace.Load TheTraceData.ad
; Open the flow trace window
Trace.List
enddo
```

## FIFO Overflows

- FIFO Overflow NEXUS messages are released if the chip internal trace FIFO is full and program flow messages get lost. This issue can be solved by increasing the amount of data that is transferred via the NEXUS lines by increasing the MCKO clock with the command **SYStem.Option.MCKO**. Unfortunately the clock is limited to the bandwidth of the NEXUS data and clock lines. Another way to prevent the FIFO from becoming full is stalling the core or suppressing less important trace data trace messages. The option **SYStem.Option.OVC** can be used to specify this behavior.

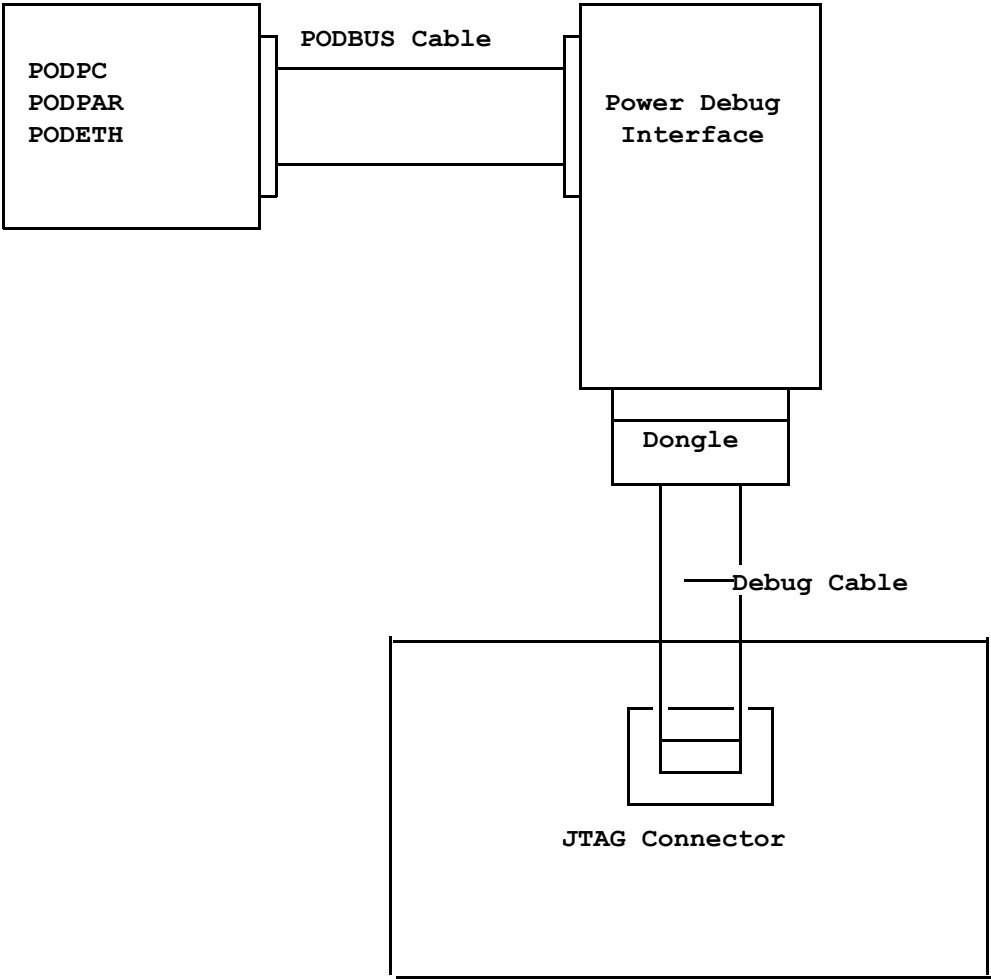
## FAQ

---

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

# Configuration

---



Basic configuration for the BDM Interface

The processor type must be selected by the **SYStem.CPU** command before issuing any other target related commands.



SYStem.CLOCK

Setup core clock

Format:	<b>SYStem.CLOCK</b> [ <i>&lt;frequency&gt;</i> ]
---------	--

Default: 0.

The option is used to calculate the timestamp in simulator mode and to set up the PLL of the Nexus preprocessor.

Nexus Preprocessor PLL ranges

The selection of the PLL range is based to the System Clock multiplied by the **SYStem.Option.MCKO** ratio.

- 1. 24.0 MHz < (SYStem.Clock \* SYStem.Option.MCKO) < 50.0 MHz
- 2. 50.0 MHz <= (SYStem.Clock \* SYStem.Option.MCKO) < 100.0 MHz
- 3. 100.0 MHz <= (SYStem.Clock \* SYStem.Option.MCKO) < 200.0 MHz

SYStem.CONFIG.state

Display target configuration

Format:	<b>SYStem.CONFIG.state</b> [ <i>/&lt;tab&gt;</i> ]
<i>&lt;tab&gt;</i> :	<b>DebugPort   Jtag</b>

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.

<i>&lt;tab&gt;</i>	Opens the <b>SYStem.CONFIG.state</b> window on the specified tab. For tab descriptions, see below.
--------------------	--

<b>DebugPort</b>	Lets you configure the electrical properties of the debug connection, such as the communication protocol or the used pinout.
<b>Jtag</b>	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.

SYStem.CONFIG


Configure debugger according to target topology

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

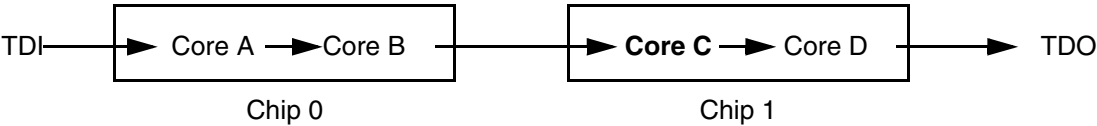
For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

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).
---	---

<b>CORE</b>	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 <a href="#">SYstem.CONFIG.CORE</a> .
<b>DRPRE</b>	(default: 0) <i>&lt;number&gt;</i> 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.
<b>DRPOST</b>	(default: 0) <i>&lt;number&gt;</i> 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.
<b>IRPRE</b>	(default: 0) <i>&lt;number&gt;</i> 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.
<b>IRPOST</b>	(default: 0) <i>&lt;number&gt;</i> 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.
<b>TAPState</b>	(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.
<b>TCKLevel</b>	(default: 0) Level of TCK signal when all debuggers are tristated.
<b>TriState</b>	(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.
<b>Slave</b>	(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).

# Daisy-Chain Example



Below, configuration for core C.

Instruction register length of

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

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

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

```
Format:          SYStem.CONFIG.CORE <core_index> <chip_index>
                SYStem.MultiCore.CORE <core_index> <chip_index> (deprecated)

<chip_index>:    1 ... i

<core_index>:    1 ... k
```

Default *core\_index*: depends on the CPU, usually 1. for generic chips

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

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

### Non-generic Chips

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

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

### Generic Chips

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

### Start-up Process

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

Format:	<b>SYStem.CPU</b> <cpu>
<cpu>:	<b>MSC8101</b>   <b>MSC8102</b>   <b>MSC8103</b>   <b>MSC8122</b>   <b>MSC8126</b> (FREESCALE) <b>MSC7110</b>   <b>MSC7112</b>   <b>MSC7113</b>   <b>MSC7115</b>   <b>MSC7116</b> (FREESCALE) <b>MXC91XXX</b> (FREESCALE) <b>SC100</b>   <b>SC1000</b> (single SC1000 core with 5bit IR-JTAG register width) <b>TC1202</b>   <b>TC1401</b>   <b>SC140</b> (StarCore LLCs TC1XXX chips) <b>SC2000</b> (Single SC2000 core with 5bit IR-JTAG register width) <b>TC2400</b> (StarCore LLCs TC2XXX chips)

Selects the processor type.

- SC100, SC1000, SC2000, SC3000 provide a basic access a customer SOC SC1000, SC2000, SC3000 device with no support for Watch Dog and Caches.
- OCECORE and SYStem.Option.OCEADDRESS should be set according the integration of the core. The SC100 CPU selection assumes a JTAG TAP controller compatible to the TCXXXX of StarCore LLCs design.

**SYStem.LOCK**

Lock and tristate the debug port

Format:	<b>SYStem.LOCK</b> [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.

**SYStem.MemAccess**

Select run-time memory access method

Format:	<b>SYStem.MemAccess</b> Enable   NEXUS   Cerberus   Denied   StopAndGo
---------	--

<b>Enable</b> <b>CPU</b> (deprecated)	Memory access of access class E: is done by the on-chip debug peripheral. The option is selectable if the simulator mode is active. OCE1 to OCE3 don't support the memory access while runtime..
<b>NEXUS</b>	Memory access of access class E: is done by the on-chip Nexus Trace block. The displayed memory is at level two behind the L1 data cache and before the L2 cache.
<b>Cerberus</b>	Memory access if access class E: is done by the on-chip Cerberus block.
<b>Denied</b> (default)	Memory access during program execution to target is disabled.
<b>StopAndGo</b>	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:	<b>SYStem.Mode</b> <mode>  <b>SYStem.Attach</b> (alias for SYStem.Mode Attach) <b>SYStem.Down</b> (alias for SYStem.Mode Down) <b>SYStem.Up</b> (alias for SYStem.Mode Up)
<mode>:	<b>Down</b> <b>NoDebug</b> <b>Go</b> <b>Attach</b> <b>Up</b>

<b>Down</b>	Disables the debugger (default). The state of the CPU remains unchanged. The JTAG port is tristated.
<b>NoDebug</b>	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
<b>Go</b>	Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or external trigger.
<b>Attach</b>	User program remains running (no reset) and the debug mode is activated. After this command the user program can be stopped with the break command or if any break condition occurs. The endian mode is not detected in that case, but can be set by SYStem.Option.LITTLEEND ON.
<b>StandBy</b>	This mode is used to start debugging from power-on. The debugger will wait until power-on is detected, then bring the CPU into debug mode, set all debug and trace registers and start the CPU. In order to halt the CPU at the first instruction, place an on-chip breakpoint to the reset address.
<b>Up</b>	Resets the target, sets the CPU to debug mode and stops the CPU. After the execution of this command the CPU is stopped and all register are set to the default level.

SYStem.Option.BASE

Sets the SUI base address

Format:	<b>SYStem.Option.BASE</b> [AUTO   VALUE]
---------	--

Default: AUTO.

System Integration Unit (SUI) base address. The base address is detected during the SYStem.Up command automatically if set to AUTO before. The IOBASE() function returns the value in scripts or peripheral description files. The debugger cannot detect a change of the base address after SYStem.Up. Please use SYStem.Option BASE to update the internal used value. When the assumed BASE value differs from the real value, bus errors can occur in scripts and peripheral browser.

SYStem.Option.DCFLUSH

Data cache flush before step/run

Format:	SYStem.Option.DCFLUSH [ON   OFF]
---------	----------------------------------

Default: OFF.

If enabled, the data cache is updated before the processor executes the next program instructions. This option must be turned on, if accesses to dual-port memory are taken by the debugger.

SYStem.Option.DTM

Enables data trace messages

Format:	SYStem.Option.DTM [ON   OFF   Read   Write   ReadWrite]
---------	---

Default: OFF.

The option can be switched when the chip has nexus trace support. When the option is set to not OFF, the cpu generates data trace messages. If set to ON or ReadWrite the messages are generated for write/read accesses. Option Read or Write enables only read or write messages.

SYStem.Option.EnReset

Allow the debugger to drive nRESET/nSRST

Format:	SYStem.Option.EnReset [ON   OFF]
---------	----------------------------------

Default: ON.

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

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

Format:	<b>SYStem.Option.EnTrst</b> [ON   OFF]
---------	--

Default: ON.

If this option is disabled the nTRST line is never driven by the debugger. Instead five consecutive TCK pulses with TMS high are asserted to reset the TAP controller which have the same effect.

Format:	<b>SYStem.Option.HalfRate</b> [ON   OFF]
---------	--

Default: OFF.

If the NEXUS probe has an enabled PLL, it will be possible to double the clock of the MCKO line in order to reconstruct the original clock when the DDR NEXUS mode is active. Use System.Option.HalfRate ON if the bandwidth of the clock line is too small and to activate the DDR mode. If the option is set to ON the data and clock lines will need the same bandwidth.

Format:	<b>SYStem.Option.ICFLUSH [ON   OFF]</b>
---------	---

Default: ON.

If enabled, the instruction cache is updated before the processor executes the next program instructions. This option must be turned on, if software breakpoints are set into cached code sections.

Format:	<b>SYStem.Option.IMASKASM [ON   OFF]</b>
---------	--

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during assembler single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

Format:	<b>SYStem.Option.IMASKHLL [ON   OFF]</b>
---------	--

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

Format:	<b>SYStem.Option.IPLDI</b> [IPL   DI]
---------	---------------------------------------

Default: DI.

SYStem.Option.IMASKHLL and IMASKASM use the SR status register to mask all interrupts. The debugger can use either the IPL bits or the DI bits to perform this task. Do to technical limitations it is not possible to detect a disable interrupt code section. After the execution of such a section the debugger cannot determine whether the interrupts are disabled by the target code or by the debugger depending on the IPLDI strategy. Use the IPL strategy, when the target code uses the SR.DI bit to disable the interrupts. The DI strategy should be applied, when the target code modifies the IPL bits in order to disable the interrupts.

**SYStem.Option.LittleEnd**

Switches between endian modes

Format:	<b>SYStem.Option.LittleEnd</b> [ON   OFF]
---------	---

Default: read after System.Up.

The StarCore cores can run in big and little endian mode. Usually the mode can be switched with the help of the state of system pins during the reset process. After System.Up the debugger reads the current mode from the EMR register. Normally it is not necessary to modify this mode during the debug process, but sometimes it is use full e.g. for verifying the endian mode of loaded code with help of help of the disassembler. In simulators the endian mode can be changed by editing the EMR:BEM bit in the Register Window.

**SYStem.Option.MCKO**

Nexus output clock ratio

Format:	<b>SYStem.Option.MCKO</b> [1/8   1/4   1/2   1/1]
---------	---

Default: 1/8.

The option can be set when the chip has nexus trace support and defines the frequency of the nexus output clock based on the processor frequency. High frequencies can cause electrical connection problems during the record of trace messages.

Format:	<b>SYStem.Option.MPU [ON   OFF]</b>
---------	-------------------------------------

Default: OFF.

If the option is set to On the MPU will be disabled while the core is stopped. In case of MMU address translation is enabled on the target there can be still bus errors displayed in case no MMU entry is found for a memory section. The memory behind the MMU can be seen by the memory class “A” e.g.

Format:	<b>SYStem.Option.Nexus [MDO16   MDO8]</b>
---------	---

Default: MDO16.

The option can be set when the chip has nexus trace support and defines port width of the MDO vector. The maximum is defined by the derivatives maximum MDO pin count.

Format:	<b>SYStem.Option.OCEBASE [AUTO   ADDRESS]</b>
---------	---

Default: AUTO.

Defines the base address for the memory mapped OnCE Register. Customer SOCs with addresses differing from 0x80000000 must set this option.

Format:	<b>SYStem.Option.OCECORE</b> [AUTO   NR]
---------	--

Default: AUTO.

In customer SOCs a standard JTAG StarCore TAP Controller implements a register, where the StarCore can be addressed when there are multiple cores present. The OCECORE sets the address for the current TRACE32 instance. Chips with a single StarCore inside don't need this option.

**SYStem.Option.OVC**

Trace message overrun control

Format:	<b>SYStem.Option.OVC</b> [NoStall   Stall1/4   Stall1/2   Stall3/4   Suppr1/4   Suppr1/2   Suppr3/4]
---------	--

Default: NoStall.

The option can be set when the chip has nexus trace support and defines the behavior that becomes active when the chip intern trace message FIFO buffer gets full. NoStall will cause losing of messages when the buffer overruns. All other StallX option will stall the processor until all trace buffers will reach a state with X elements. The SupprX option disables data trace messages until all trace buffers reach X elements. To do advanced flow analysis a trace record with no FIFO Overflow Messages is required.

**SYStem.Option.PTM**

Enables program trace messages

Format:	<b>SYStem.Option.PTM</b> [ON   OFF]
---------	-------------------------------------

Default: OFF.

The option can be switched when the chip has nexus trace support. When PTM is ON, the chip will produce program trace messages.

Format:

**SYStem.Option.SAMPLE** [-4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4]

Default: 0.

If the NEXUS probe has an enabled PLL the phase shift of the MCKO line can be adjusted. This setting can help you to find a better sample point and reach higher *MCKO ratios*.

**SYStem.Option.SLOWPOLL**

Change timing of JTAG during runtime

Format:

**SYStem.Option.SLOWPOLL** [ON | OFF]

Default: ON.

While the starcore is in running mode internal stall cycles can affect the communication. The result can be the TAP ERROR state or data corruption together with the DCC protocol (used for semi hosting). The disturbance will increase with longer stall cycles that can occur while the cache is fetching memory or accesses to external slow busses (e.g. SDRAM with wait states set to high values). SLOWPOLL avoids the communication problems by decreasing the communication speed.

**SYStem.Option.SLOWRESET**

Expand reset time for additional reset module

Format:

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

Default: OFF.

Defines that the debugger waits additional time for asserting the reset line through an additional reset controller.



Format: **SYStem.Option.VBA** [**AUTO** | *<address>*]

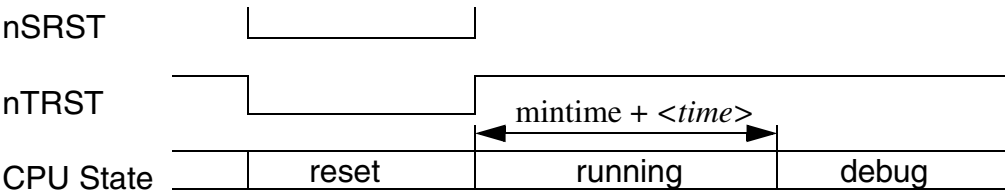
Default: AUTO.

Analysis functions like Nexus trace flow analysis depend to a correct VBA value. When the core is running or down the last known VBA address may be not up to date, therefore it is possible to use a fixed VBA address specified by this command. The option **AUTO** will force the debugger to use the last known value. If any address is set by this command then this address will be used.

Format:	<b>SYSystem.Option.WaitReset</b> [OFF   <time>]
---------	---

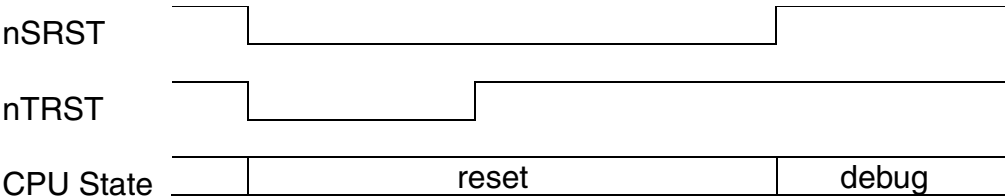
Default: OFF.

This option has to be enabled if the nTRST line is connected to the nRESET/nSRST line on the target. In this case the CPU executes some cycles while the **SYSystem.Up** command is executed. The reason for this behavior is the fact that it is necessary to halt the core (enter debug mode) by a JTAG sequence. This sequence is only possible while nTRST is inactive. In the following figure the time between the de-assertion of reset and the entry to debug mode is the time for this JTAG sequence.



Often a reset delay unit is used or the JTAG is still unavailable some time after nRESET is released. In this cases a time >0µs should entered to define the timing when the debugger is allowed to begin to access to the JTAG after nRESET is released. It is suggested to set the time as small as necessary because the StarCore can reach an unrecoverable state due to the code execution within the time between reset and debug mode. The executed code can touch the peripherals which would become active when the core starts to execute code again. The total between the end of reset and debug mode has a minimum *mintime* because the sequence to enter debug time consumes time. The minimum time depends to the JTAG frequency and core.

If nTRST is available and not connected to nRESET/nSRST it is possible to force the CPU directly after reset (without cycles) into debug mode. This is also possible by pulling nTRST fixed to VCC (inactive), but then there is the problem that it is normally not ensured that the JTAG port is reset in normal operation. If the WaitReset option is disabled the debugger first de-asserts nTRST, then it executes a JTAG sequence to enter the debug mode and then it de-asserts nRESET/nSRST.



Format:	<b>SYStem.Option.WATCHDOG</b> [ON   OFF]
---------	--

Default: OFF.

The watchdog remains active when this option is set to ON [MSC8101]. In this case the watchdog is served periodically by the debugger, when the chip is in debug mode. Deactivating the watchdog through option OFF implicates a write access to SYPCR which is writable only one-time after the reset. When a special configuration word SYPCR is required, disable the watchdog in the startup script by writing the correct value to SYPCR, or disable it directly in your application.

Example for disabling the watchdog in a script

```
; Watchdog remains active
SYS.Option WATCHDOG ON
SYStem.Up
; the following command read and writes the SYPCR and
; de-asserts the watchdog bit
Data.Set IOBASE()+0x10004 %Long
(Data.Long (D: (IOBASE()+0x10004)) &0xFFFFFFFFB)
```

Format:	<b>SYStem.JtagClock</b> [<frequency>   <b>RTCK</b>   <b>ARTCK</b> <frequency>   <b>CTCK</b> <frequency>   <b>CRTCK</b> <frequency>]
<frequency>:	<b>6kHz ... 80MHz</b> <b>1250000.   2500000.   5000000.   10000000.</b>

Default frequency: 10 MHz.


Selects the JTAG port frequency (TCK) used by the debugger to communicate with the processor. The frequency affects e.g. the download speed. It could be required to reduce the JTAG frequency if there are buffers, additional loads or high capacities on the JTAG lines or if VTREF is very low. A very high frequency will not work on all systems and will result in an erroneous data transfer. Therefore we recommend to use the default setting if possible.


<frequency>:

The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the **SYStem.state** window.

Besides a decimal number like “100000.” short forms like “10kHz” or “15MHz” can also be used. The short forms imply a decimal value, although no “.” is used.

- When the debugger is not working correctly (e.g. memory is flickering) decrease the JtagClock.

	The JTAG clock must be limited to 1/4 of the StarCore core clock
---	--

	Buffers, additional loads or high capacities on the JTAG/COP lines reduce the debug speed.
---	--

The following sections about **RTCK**, **ARTCK**, **CTCK**, **CRTCK** are only relevant if a Board with a 20-pin Debug Cable is used e.g. LA-7845. Usually this is the case if an ARM core is mixed with a StarCore core in a device.


**RTCK:** The JTAG clock is controlled by the RTCK signal (**R**eturned **T**CK).

On some processor derivatives (e.g. chips including an ARMxxxE-S) there is the need to synchronize the processor clock and the JTAG clock. In this case RTCK shall be selected. Synchronization is maintained, because the debugger does not progress to the next TCK edge until after an RTCK edge is received.

In case you have a processor derivative requiring a synchronization of the processor clock and the JTAG clock, but your target does not provide an RTCK signal, you need to select a fix JTAG clock below 1/6 of the processor clock (chips including ARM7, ARM9), below 1/8 of the processor clock (chips including ARM11), respectively.

When RTCK is selected, the frequency depends on the processor clock and on the propagation delays. The maximum reachable frequency is about 16 MHz.

Example: `SYStem.JtagClock RTCK`

	The clock mode RTCK can not be used if a debug cable with 14-pin flat cable (LA-7834) is used. And it is required that the target provides an RTCK signal.
---	--

**ARTCK:** Accelerated method to control the JTAG clock by the RTCK signal (**A**ccelerated **R**eturned **T**CK).

RTCK mode allows theoretical frequencies up to 1/6 (chips including ARM7, ARM9) or 1/8 (chips including ARM11) of the ARM core processor clock. For designs using a very low processor clock we offer a different mode (ARTCK) which does not work as recommended by ARM and might not work on all target systems. In

ARTCK mode the debugger uses a fixed JTAG frequency for TCK, independent of the RTCK signal. This frequency must be specified by the user and has to be below 1/3 of the ARM processor clock speed. TDI and TMS will be delayed by 1/2 TCK clock cycle. TDO will be sampled with RTCK



The mode ARTCK can not be used if a debug cable with 14-pin flat cable (LA-7834) is used. And it is required that the target provides an RTCK signal.

**CTCK:** With this option higher JTAG speeds can be reached. The TDO signal will be sampled by a signal which derives from TCK, but which is timely compensated regarding the debugger-internal driver propagation delays (**Compensation by TCK**). This feature can be used with a debug cable versions 3b or newer. If it is selected, although the debug cable is not suitable, a fix JTAG clock will be selected instead (minimum of 10 MHz and selected clock).

**CRTCK:** With this option higher JTAG speeds can be reached. The TDO signal will be sampled by the RTCK signal. This compensates the debugger-internal driver propagation delays, the delays on the cable and on the target (**C**ompensation by **RTCK**). This feature requires that the target provides an RTCK signal. In contrast to the **RTCK** option, the TCK is always output with the selected, fixed frequency.



The mode CRTCK can not be used if a debug cable with 14-pin flat cable (LA-7834) is used. And it is required that the target provides an RTCK signal.

MMU.DUMP

Page wise display of MMU translation table

Format:

MMU.DUMP <table> [<range> | <address> | <range> <root> | <address> <root>]

MMU.<table>.dump (deprecated)

<table>:

ITLB

DTLB

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 <a href="#">space ID</a> is given.
ITLB	Displays the contents of the Instruction Translation Lookaside Buffer.
DTLB	Displays the contents of the Data Translation Lookaside Buffer.

MMU.List

Compact display of MMU translation table

Format:

MMU.List

This command shows the debugger-internal translation table. See [TRANslation.List](#).

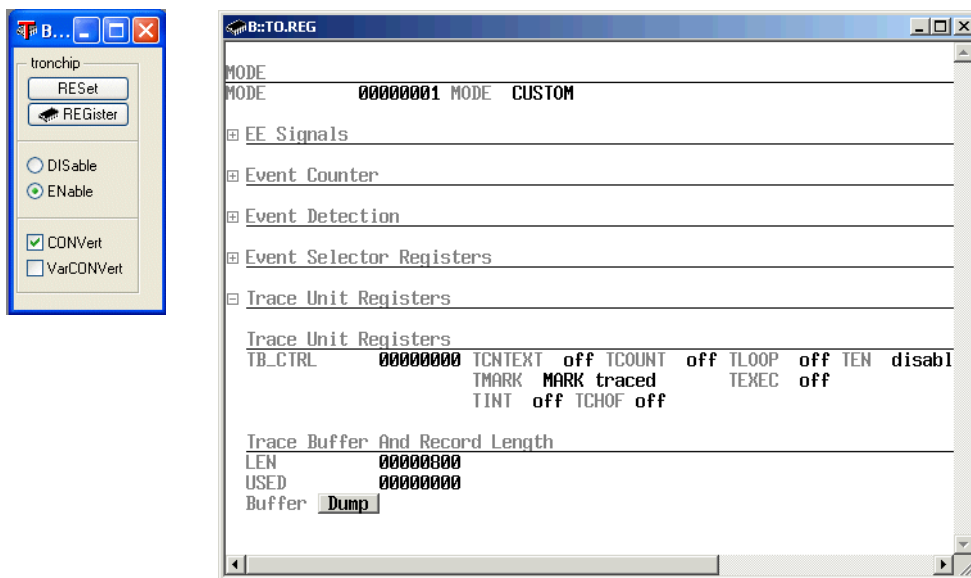
Format:	<b>MMU.SCAN</b>
---------	-----------------

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

This command is not supported.



For information about *architecture-independent* **BMC** commands, refer to “**BMC**” (general\_ref\_b.pdf).



The OCE/EOnCE unit of the starcore allows to set on-chip breakpoints and to use the on-chip trace unit. The registers can be controlled either by TRACE32 or by setting it directly through the DBG memory class and the target program. TRACE32 uses the on-chip trigger EOnCE registers to perform on-chip breakpoints, which can be set in the [List.auto](#) window or in the [Break.Set](#) dialog. The current user interface of TRACE32 offers many possible configurations of the EOnCE unit:

- Up to 12 program address breakpoints
- Or 6 program address range breakpoints
- Or 6 data address breakpoints
- Or 3 data address range breakpoint
- Or 10 program address breakpoints and 1 program address count breakpoint
- Or 1 data address range with value range

The amount of range breakpoints is limited that's why it is sometimes useful to set the [TrOnchip.CONVrt](#) option. When enabled, this option let transform range breakpoints into normal, if necessary. The EOnCE can perform more operations than TRACE32 offers with it's user interface e.g. build a chain of breakpoints. If additional features are required, a special mode can be used, where TRACE32 takes no influence on the on-chip trigger resources. The dialog TrOnchip includes the switch for this mode. If the special user mode is activated by selecting "disable", the registers can be accessed by clicking on "view Register".

The capability and amount of breakpoints depends to the OCE/EOnce trigger unit structure. An EDCA unit can hold 2 program breakpoints or 1 program range breakpoint. Data address breakpoints need one single EDCA. Data address range breakpoints need two EDCA since data bus A and B needs to be checked independently. The EDCA, EDCA and the ECNT can be connected all by conjunction or disjunction. A data address range breakpoint with additional data value range is not possible since this contradicts in the connection, but single data address with value range are possible. The options TrOnchip.CONVrt and TrOnchip.VarCONVrt help to convert ranges to single addresses. Also the ECNT unit has only one input line so that data range address breakpoints cannot be counted.

The on-chip counter can be used by several components. If the counter is not used by the on-chip trace leash mode or on-chip trace stack mode then it can be used as breakpoint counter limited to it's one input channel. If no component uses the counter then it will be used as cycle counter and is accessible by the CYCLO and CYCHI register of the register window.

## Nexus related actions

The on-chip trigger unit events can be also used to control the on-chip trace and nexus trace. The possible actions can be defined in the Breakpoint.Set dialog. When the nexus trace is active, the ordinary on-chip trace will be not active. To control the trace unit an appropriate action has to be selected for the **Break.Set** command. The event detection units cannot be associated by conjunction or disjunction that's why data ranges or event data value triggered actions are not possible.

```
b.s main /TRACEON           ; Enabled program Trace at main
b.s sieve /TRACEOFF         ; Disabled program Trace at sieve
b.s flags /TRACEDATA        ; Set up a filter for Data Trace (only
                             ; Nexus with DTM option set to on)
b.s main /TRACETRIGGER      ; Set Watchpoint message or/and EVTO pin
                             ; to generate Trigger
b.s main /BUSTRIGGER         ; Set Watchpoint message or/and EVTO pin
                             ; to generate a trigger pulse on the
                             ; PodBus
b.s main /BUSCOUNT         ; Set Watchpoint message or/and EVTO pin
                             ; to allow frequency counter feature
```

## On-chip Trace related actions

The on-chip trace can be enabled or disabled by on-chip breakpoints.

```
b.s main /TRACEON           ; Enabled program Trace at main
b.s sieve /TRACEOFF         ; Disabled program Trace at sieve
```

## Determining the break source

The command **TrOnchip.REGister** opens a window displaying the OCE/EOnce status registers ESR and EMCR. The ESR register contains bits to show why the core entered debug mode. If there was no break indication bit true, the reason was probably a manual break coming from JTAG.

Format:	<b>TrOnchip.ENABLE</b> <b>TrOnchip.DISABLE</b>
---------	---

Default: Enable

**ENABLE**

Enable on-chip breakpoints from the GUI.

**DISABLE**

Disable influence of GUI. In the CUSTOM mode it is allowed to modify the EOnCE register of the trigger on-chip unit to implement extended behavior e.g. breakpoint chains or EE-Signal control.

**Example for using T32 mode**

```
; activate mode
TrOnchip.ENABLE
; set on-chip breakpoint on main
Break.Set main /Onchip
; go and break
go
enddo
```

**Example for using CUSTOM mode**

```
; activate custom mode
TrOnchip.DISABLE
; configure trigger onchip unit and set on-chip breakpoint on main with
EDCA2
    ; open register file to determine addresses in DBG: memory space
    TrOnchip.REGISTER
    ; EDCA2_CTRL: ENA, A only, PC, read, equal
    D.S DBG:0x390 %LONG %LE 0x3C03
    ; EDCA2_REFA: REFA = main
    D.S DBG:0x394 %LONG %LE main
    ; ESEL_CTRL: SELDM = OR
    D.S DBG:0x3E8 %LONG %LE 0x0
    ; ESEL_DM: EDCA2 = ON
    D.S DBG:0x3EC %LONG %LE 0x4
; go and break
go
enddo
```

Format:

TrOnchip.CONVert [ON | OFF]

When enabled (default) the address on-chip breakpoints are automatically converted from a range to a single address if required. If the switch is off, the system will only accept breakpoints which exactly fit to the on-chip breakpoint hardware.

Format:

TrOnchip.REGister

The command opens a dialog, where the custom on-chip trigger and trace registers can be set. This is only useful, if TrOnchip.Mode equaled CUSTOM.

Format:

TrOnchip.RESet

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

Format:

TrOnchip.VarCONVert [ON | OFF]

When enabled (default) the data address on-chip breakpoints are automatically converted from a range to a single address if required. If the switch is off, the system will only accept breakpoints which exactly fit to the on-chip breakpoint hardware.

Format:	TrOnchip.state
---------	----------------

Control panel to configure the on-chip breakpoint and trace registers. The details are described in section [TrOnchip](#).

## Onchip.Mode

Select mode to control trace buffer and contents

Format: **Onchip.Mode** [<mode>]

<mode> **Fifo | Stack | Leash  
ExecutionTrace  
FlowTrace  
LoopTrace  
CompressTrace  
InterruptTrace  
TimeStamp**

Trace Buffer Modes that control the behavior of the trace buffer if it is full:

- **Fifo.** In the Fifo mode the trace buffer is implemented as a Ring Buffer. When the trace is full the write pointer just returns to the begin of the trace buffer and overwrites previous data.
- **Stack.** If the Stack mode is active, the record is stopped when the trace buffer is full.
- **Leash.** The Leash mode is a special Stack mode where the target also stops when the trace is full. The Leash mode is cycle accurate. When the target stops due to that event all previous messages can be seen. The Leash mode may conflict with the TimeStamp setting or counted breakpoints, because the same counter is used in another way.

Trace modes that control the contents of the trace buffer:

- **ExecutionTrace.** The Address of every executed VLES is written to the trace buffer. The advantage of this mode is that all analysis based on this buffer can be done very fast and easy.
- **FlowTrace.** Only change of flow instructions are recorded to save trace buffer memory.
- **LoopTrace.** Hardware Loops are not displayed in the trace to save trace buffer memory. Usually this does not matter unless the Hardware Loops don't contain other jumps. Use FlowTrace or CompressTrace if the program flow cannot be reconstructed.
- **CompressTrace.** Some derivatives support an additional compression for loops (repeated branch messages) and Task\_ID information. The CompressTrace is based on the FlowTrace with the advantage of the LoopTrace to save trace buffer memory in case of loops.
- **InterruptTrace.** If enabled interrupts will be traced.
- **TimeStamp.** Change of flow oriented trace modes offer the possibility to trace the OCE/EOnce counter in order to reconstruct a timestamp. If TimeStamp is set to ON, the counter cannot be used anymore for the Leash mode or counting other events like breakpoints. To derive a timestamp from the clock cycles the options **SYStem.Clock** or **OnChip.CLOCK** need to be set to a correct value.



Format:	<b>Onchip.VTBA</b> [<addressrange>]
---------	-------------------------------------

The command VTBA sets the Virtual Trace Buffer Address range in order to program the onchip trace unit with this information. After the command is execute the onchip trace has a defined size and can be armed.

Some derivatives do not have a separate onchip memory to store the trace records, but can write the entries to an arbitrary memory location, which can be set by the VTBA command. The real-time capability is limited as more the write process stalls the core. The onchip trace size only depends to the memory layout of the target application. The speed of post analysis is limited to the download speed of the target via JTAG. Higher JTAG frequencies will speed up the process as well as smaller trace buffer ranges.

## **ASM debugging in hardware loops - set PC**

When the PC register is set in a hardware loop, the loop flags and loop count register may be change.

## **ASM debugging in hardware loops - stepping**

The debugger tries to step over delay slots. If the debugger is not successful, set a software breakpoint after the hardware loop and use go to step over the hardware loop.

## **HLL debugging in optimized code**

HLL debugging in optimized code is restricted. Source lines may be assigned wrong, local variables may not be displayed.

## **Software breakpoints in delay slots**

Breakpoints must not be placed in delay slots. Program flow and data integrity may be wrong after break.

## **Debugging with interrupts**

When IMASKHLL or IMASKASM is enabled the debugger won't update correctly the interrupt disable bit in the SR register in case the code executed the DI instruction. Use SYStem.Option.IPLDI to switch the behavior.

## **MultiCore synchronous start between StarCore and ARM cores**

The StarCore starts 1 JTAG clock cycle before the ARM core starts

## **On-chip trace with mode "LoopTrace"**

When a jump instruction is within a loop body the debugger warns about flow errors. As a workaround the trace mode "FlowTrace" should be used.

## **MSC8122 does not halt with SYStem.UP**

The core(s) cannot be stopped by JTAG when the reset is active. Make sure that the EE0 signal is high while reset in order to halt the core just after reset. "System.Option.WaitReset 0us" will also work, but the core will execute some instructions before it halts.

# Floating Point Formats

---

<b>F24</b>	Fractional fixed point 24 bit
<b>F48</b>	Fractional fixed point 48 bit
<b>F16</b>	Fractional fixed point 16 bit
<b>F32</b>	Fractional fixed point 32 bit

<b>NOTE:</b>	Fractional floating point numbers are always displayed with a fixed precision, i.e. a fixed number of digits. Small fractional numbers can have many non relevant digits displayed.
--------------	---

# Integer Access Keywords

---

<b>Word</b>	Word (16 bit)
<b>TByte</b>	Triple byte (24 bit)
<b>Long</b>	Double Word (32 bit), upper and lower word swapped
<b>HByte</b>	Hexabyte (48 bit)
<b>Quad</b>	Tertiary Word (64 bit), upper and lower word swapped

## Metrowerks MSLIO Support

---

The C-Library delivered with Metrowerks Compiler contains the Standard C I/O functions like *printf()* or *fopen()*. It is possible to make this functions working together with the host in nearly all configurations or targets. The key is to set a breakpoint to the `__syscall` address in the target program. TRACE32 for StarCore has implemented the protocol stack for the `__syscall` function. When the target stops at the breakpoint the data is processed by TRACE32 host software and the target is set to running mode again. The approach of course will have influence to the targets real time capability indicated by a red S in the TRACE32 [state line](#). To activate the MSLIO support the following line should be executed:

```
WinClear TERMW           ; Closes opened terminal window

Break.Set __syscall      ; Sets breakpoint to __syscall
                          ; function

Term.Method MSLIO __syscall ; Set MSLIO protocol activated by
                          ; break point __syscall

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

WinPOS 0 0 79 10 100 100 TERMW ; Setup position and name for Terminal
                          ; window

Term                     ; Opens terminal window

Data.List                ; Open source window

Register.view /SpotLight ; Open register window

Var.Local                ; Open window with local variables
```

In ROM, FLASH or shared memory sections it is not possible or not recommended to set software breakpoints. In this case it has to be considered that the StarCore on-chip breakpoint is not exact. It stops just after the VLES is executed. To trigger the Terminal Protocol to the right Program Counter, just add 0x2 to the `Term.Method MSLIO` address:

```
Break.Set __syscall /ONCHIP ; Sets on-chip breakpoint to __syscall
                          ; function

Term.Method MSLIO __syscall+0x2 ; Set MSLIO protocol activated by
                          ; break point __syscall
```

## Mechanical Description of the 20-pin Debug Cable

---

This connector is defined by ARM and we recommend this connector for all future designs. Our debugger “JTAG Debugger for StarCore” (LA-7845) is supplied with this connector:

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	VSUPPLY (not used)
TRST-	3	4	GND
TDI	5	6	GND
TMSITMSCISWDIO	7	8	GND
TCKITCKCISWCLK	9	10	GND
RTCK	11	12	GND
TDOI-ISWO	13	14	GND
RESET-	15	16	GND
DBGRRQ	17	18	GND
DBGACK	19	20	GND

This is a standard 20 pin double row connector (pin-to-pin spacing: 0.100 in.).

We strongly recommend to use a connector on your target with housing and having a center polarization (e.g. AMP: 2-827745-0). A connection the other way around indeed causes damage to the output driver of the debugger.

# Electrical Description of the 20-pin Debug Cable

---

- The input and output signals are connected to a supply translating transceiver (74ALVC164245). Therefore the ICD can work in a voltage range of (1.5 V) 1. ... 3.3 V (3.6 V). Please note that a 5V supply environment is not supported! This would cause damage on the ICD.

The newer debug cables (since about March 2004; having a different connector case design) can work in a voltage range of 0.4 ... 5.0 V (5.25 V).

- VTREF is used as a sense line for the target voltage. It is also used as supply voltage for the supply translating transceiver of the ICD interface to make an adaptation to the target voltage (1.5 V) 1.8 ... 3.3 V (3.6 V). On the newer debug cables (September 2003 and newer) it is used as sense line, only.
- nTRST, TDI, TMS, TCK are driven by the supply translating transceiver. In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the JTAG port. In environments where multiple tools can access the JTAG port, it is required that there is a pull-up or pull-down resistor at TCK. This is to ensure that TCK maintains its level during a hand-over between different tools. Depending to that level the option [SYStem.CONFIG.TCKLevel](#) has to be set. The nTRST signal must have a pull-up resistor to have the correct level while endeavor between different tools.
- RTCK is the return test clock signal from the target JTAG port. This signal can be used to synchronize JTAG clock with the processor clock (see [SYStem.JtagClock](#)).
- TDO is an ICD input. It is connected to the supply translating transceiver.
- nSRST (=nRESET) is used by the debugger to reset the target CPU or to detect a reset on the target. It is driven by an open collector buffer. A pull-up resistor is included in the ICD connector. The debugger will only assert a pulse on nSRST when the SYStem.UP, the SYStem.Mode Go or the SYStem.RESetOUT command is executed. If it is ensured that the ARM is able to enter debug mode every time (no hang-up condition), the nSRST line is optional.
- EDBGQRQ is driven by the supply translating transceiver. This line is optional. It allows to halt the program execution by an external trigger signal.
- DBGACK is an ICD input. It is connected to the supply translating transceiver. A pull-down resistor is included in the ICD connector. This line is optional. It allows exact runtime measurement and exact triggering of other devices on a program execution halt.
- N/C (= Vsupply) is not connected in the ICD. This pin is used by debuggers of other manufacturers for supply voltage input. The ICD is self-powered.

There is an additional plug in the connector on the debug cable to the debug interface. This signal is tristated if the JTAG connector is tristated by the debugger and it is pulled low otherwise. This signal is normally not required, but can be used to detect the tristate state if more than one debug tools are connected to the same JTAG port.

# JTAG Connector 14-pin

This connector is the standard for single starcore systems like on StarCoreLLCs evaluation boards or Freescale public evaluation board for MSC8xxx or MSC7xxx. Our debugger “JTAG Debugger for StarCore” (LA-7834) is supplied with the connector below. An “Adapter for StarCore Boards with Ejector” is offered by LA-3724:

Signal	Pin	Pin	Signal
TDI	1	2	GND
TDO	3	4	GND
TCK	5	6	GND
N/C	7	8	KEY
RESET-	9	10	TMS
VCCS	11	12	N/C
N/C	13	14	TRST-

Pins	Connection	Description	Recommendations
1	TDI	Test Data In	If there are multiple devices on the JTAG chain, connect TDI to the TDO signal of the previous device in the chain.
2,4,6	GND	System Ground Plan	Connect to digital ground.
3	TDO	Test Data Out	If there are multiple devices on the JTAG chain, connect TDO to the TDI signal of the next device in the chain.
5	TCK	Test Clock	Add 10 kΩ pull-up resistor to VCC.
7, 13, 12	NC	No Connect	Leave unconnected.
8	KEY	Mechanical Keying	Pin should be removed.
9	/RESET	Reset	May be tied to HRESET.
10	TMS	Test Mode Select	None.

Pins	Connection	Description	Recommendations
11	VCCS	VCC Sense	Connect to Chip I/O voltage VDDH through a 10 k $\Omega$ current limiting resistor.
14	/TRST	Test Reset	TRST has an internal pull-up resistor, so no external pull-up or pull-down resistor is required. However, a 10 k $\Omega$ pull-down resistor should be added to GND on this signal to keep the JTAG in reset mode while the device is operating regularly. When using more than one debug dongle driving this signal it is not recommended to pull down the signal in debug mode, because during the dongle source switch the signal output is set to tristate.



# Memory Classes

Memory Class to select real-time access	Description
E	While the CPU is running all memory accesses containing the E character will access the memory defined by <b>SYStem.MemAccess</b> . When the CPU is stopped the E character is not considered.

Memory Class	Description
D,C	Data memory. Memory seen from the cores point of view.
P	Program memory. Access is done with disabled data cache. The instruction cache state is not touched here. Set <b>SYStem.Option.ICFLUSH</b> to update the instruction cache before the core executes instructions.

Memory Class attributes to select Supervisor or User mode view	Description
S	Supervisor mode. The memory is seen from the cores point of view when it is in Supervisor/Exception mode.
U	User mode. The memory is seen from the cores point of view when it is in User mode.
not S or U	Automatically selects U or S derived from EXP and PE bit of the PSR register.

Memory Class attributes to select MMU address translation (ATE) and MMU protection (MPU)	Description
not A	Address translation enabled if it is enabled by the MMU (logical address).
A	Absolute address. Any access is done with ATE/MPU disabled (physical address). The A attribute should be used in peripheral files in order to be independent of the current MMU settings.

Example Address	Description
UD:0x0	References to logical user data memory at 0x0.
AD:0x1000	References to absolute physical address data memory at 0x1000.
SP:0x2000	References to logical supervisor program memory at 0x2000.

•