LAUTERBACH
DEVELOPMENT TOOLS

MAC71xx/72xx NEXUS Debugger and Trace

MANUAL

MAC71xx/72xx NEXUS Debugger and Trace

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

ICD In-Circuit Debugger ..... 🗁

Processor Architecture Manuals ..... 🗁

MAC71xx/72xx ..... 🗁

MAC71xx/72xx NEXUS Debugger and Trace 1

Brief Overview of Documents for New Users 5

Warning 6

Quick Start of the JTAG Debugger 7

Troubleshooting 9

Communication Between Debugger and Processor cannot be established 9

FAQ 9

Trace Extension 9

ARM specific Implementations 10

Breakpoints 10

Software Breakpoints 10

On-chip Breakpoints for Instructions 10

On-chip Breakpoints for Data 10

Hardware Breakpoints (Bus Trace only) 11

Example for Standard Breakpoints 11

Complex Breakpoints 13

Direct ICE Breaker Access 13

Trigger 14

Virtual Terminal 14

Semihosting 14

Runtime Measurement 15

Coprocessors 15

Memory Classes 16

Programming the On-chip FLASH of the MAC71/72xx 17

ARM specific SYStem Commands 18

SYStem.CONFIG.state Display target configuration 18

SYStem.CONFIG Configure debugger according to target topology 18

SYStem.CPU Select the used CPU 20
SYStem.JtagClock Define JTAG frequency 21
SYStem.LOCK Tristate the JTAG port 22
SYStem.MemAccess Select run-time memory access method 23
SYStem.Mode Establish the communication with the target 24
SYStem.Option.ABORTFIX Do not access 0x0-0x1f 25
SYStem.Option.BUGFIX Breakpoint bug fix for ARM7TDMI-S REV2 25
SYStem.Option.BigEndian Define byte order (endianness) 26
SYStem.Option.CFLUSH FLUSH the cache before step/go 26
SYStem.Option.DBGACK DBGACK active on debugger memory accesses 26
SYStem.Option.DisMode Define disassembler mode 27
SYStem.Option.EnReset Allow the debugger to drive nRESET/nSRST 27
SYStem.Option.IMASKASM Disable interrupts while single stepping 28
SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping 28
SYStem.Option.OVERLAY Enable overlay support 28
SYStem.Option.INTDIS Disable all interrupts 29
SYStem.Option.LOCKRES Go to 'Test-Logic Reset' when locked 29
SYStem.Option.NOIRCHECK No JTAG instruction register check 29
SYStem.Option.PC Define address for dummy fetches 30
SYStem.Option.ResBreak Halt the core after reset 31
SYStem.Option.RisingTDO Target outputs TDO on rising edge 31
SYStem.Option.ShowError Show data abort errors 32
SYStem.Option.SOFTLONG Use 32-bit access to set breakpoint 32
SYStem.Option.SOFTWORD Use 16-bit access to set breakpoint 32
SYStem.Option.SPLIT Access memory depending on CPSR 32
SYStem.Option.STEPSOFT Use software breakpoints for ASM stepping 33
SYStem.Option.TRST Allow debugger to drive TRST 33
SYStem.Option.TURBO Speed up memory access 33
SYStem.Option.WaitReset Wait with JTAG activities after deasserting reset 34
SYStem.RESetOut Assert nRESET/nSRST on JTAG connector 34

ARM specific NEXUS Commands 35
NEXUS.BTM Control for branch trace messages 35
NEXUS.ThumbBTM Control for branch trace messages 35
NEXUS.OTM Control for ownership trace messages 35
NEXUS.WTM Control for watchpoint messages 36
NEXUS.DTM Control for data trace messages 36
NEXUS.PortMode Set NEXUS trace port frequency 36
NEXUS.PortSize Set trace port width 36
NEXUS.UBA Specify user base address 37
NEXUS.STALL Stall the program execution 37

ARM specific TrOnchip Commands 38
TrOnchip.RESet Reset on-chip trigger settings 38
TrOnchip.CONVert Extend the breakpoint range 38

TrOnchip.Mode Configure unit A and B 39
TrOnchip.A Programming the ICE breaker module 39
TrOnchip.A.Value Define data selector 39
TrOnchip.A.Size Define access size for data selector 40
TrOnchip.A.CYcle Define access type 40
TrOnchip.A.Address Define address selector 41
TrOnchip.A.Trans Define access mode 41
TrOnchip.A.Extern Define the use of EXTERN lines 42
TrOnchip.state Display on-chip trigger window 42

Filter and Trigger for the NEXUS Trace 43
Filter and Trigger provided by the Processor 43

Nexus specific TrOnchip Commands 44
TrOnchip.EVTI Allow the EVTI signal to stop the program execution 44
TrOnchip.EXTernal Generate a trigger for the trace on high pulse on INx 45

JTAG Connection 46
Mechanical Description of the 20-pin Debug Cable 46
Electrical Description of the 20-pin Debug Cable 47
Mechanical Description of the 14-pin Debug Cable 48
Electrical Description of the 14-pin Debug Cable 48

Technical Data 49
Pinout MICTOR 49
Mechanical Dimension 50
Adapter 50
Operation Voltage 50

MAC71xx/72xx NEXUS Debugger and Trace

Version 06-Jun-2024

Brief Overview of Documents for New Users

Architecture-independent information:

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

Architecture-specific information:

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

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.

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 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>

SYStem.Option.BigEndian [ON | OFF]

SYStem.Option.EnReset [ON | OFF]
```

The 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. Map the EPROM simulator if available (optional).

```
MAP.ROM 0x0--0x1FFFF
```

This command maps a standard 8 bit wide 27x010 EPROM.

4. Inform the debugger about read-only address ranges (ROM, FLASH).

```
MAP.BOnchip 0x100000++0x0fffff
```

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.

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. Load the program.

```
Data.LOAD.Elf armf                ; (ELF specifies the format, armf
                                  ; is the file name)
```

The format of the Data.LOAD command depends on the file format generated by the compiler.

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

A typical start sequence without EPROM simulator is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII file format) and executed with the command **DO** *<filename>*.

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

WinCLEAR                          ; Clear all windows

MAP.BOnchip 0x100000++0xfffff     ; Specify where FLASH/ROM is

SYStem.Up                         ; Reset the target and enter debug mode

Data.LOAD.Elf armf.AXF            ; Load the application

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

Register.Set r13 0x8000           ; Set the stack pointer to address 8000

PER.view                          ; Show clearly arranged peripherals
                                  ; in window *)

Data.List                         ; Open source code window *)

Register /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 *)

Break.Set 0x1000 /Program         ; Set software breakpoint to address
                                  ; 1000 (address 1000 outside of BOnchip
                                  ; range)

Break.Set 0x101000 /Program       ; Set on-chip breakpoint (ice breaker)
                                  ; to address 101000 (address 101000 is
                                  ; within BOnchip range)
```

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

Troubleshooting

Communication Between Debugger and Processor cannot be established

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.

- The target has no power.
- The ARM core has no clock.
- The target is in reset.
- The ARM core is not enabled.
- There is logic added to the JTAG state machine.
- There are additional loads or capacities on the JTAG lines.
- There is a shortcut on at least one of the output lines of the core.

FAQ

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

Trace Extension

The MAC71/7200 family offers Nexus class 2+ or 3+ trace, depending on the derivative.

Breakpoints

Software Breakpoints

If a software breakpoint is used, the original code at the breakpoint location is patched by a breakpoint code.

While software breakpoints are used, one of both ICE breaker units is programmed with the breakpoint code. This means whenever a software breakpoint is set only one ICE unit breakpoint is remaining for other purposes. There is no restriction in the number of software breakpoints.

On-chip Breakpoints for Instructions

If on-chip breakpoints are used, the resources to set the breakpoints are provided by the CPU. For the ARM architecture the on-chip breakpoints are provided by the “ICEbreaker” unit. On-chip breakpoints are usually needed for instructions in FLASH/ROM.

With the command **MAP.BOnchip** *<range>* it is possible to tell the debugger where you have ROM / FLASH on the target. If a breakpoint is set into a location mapped as BOnchip one ICEbreaker unit is automatically programmed.

On-chip Breakpoints for Data

To stop the CPU after a read or write access to a memory location on-chip breakpoints are required. In the ARM notation these breakpoints are called watchpoints. A watchpoint may use one or two ICEbreaker units.

Overview

- **On-chip breakpoints:** Total amount of available on-chip breakpoints.
- **Instruction breakpoints:** Number of on-chip breakpoints that can be used to set program breakpoints into ROM/FLASH/EPROM.
- **Read/Write breakpoints:** Number of on-chip breakpoints that can be used as Read or Write breakpoints.
- **Data breakpoint:** Number of on-chip data breakpoints that can be used to stop the program when a specific data value is written to an address or when a specific data value is read from an address

	On-chip Breakpoints	Instruction Breakpoints	Read/Write Breakpoints	Data Breakpoint
MAC71/72 xx ARM7	2 (Reduced to 1 if software breakpoints are used)	2/1 Breakpoint ranges as bit masks	2/1 Breakpoint ranges as bit masks	2

Hardware Breakpoints (Bus Trace only)

When a Preprocessor for ARM7 family is used, hardware breakpoints are available to filter the trace information.

If a hardware breakpoint is used, the resources to set the breakpoint are provided by the TRACE32 development tool.

The Simple Trigger Unit (**STU**) as part of the Nexus trace probe, offers additional trigger and trace capabilities.

Example for Standard Breakpoints

Assume you have a target with

- FLASH from `0x0--0x1ffff`
- RAM from `0x100000--0x11ffff`

The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x0--0xfffff
```

The following standard breakpoint combinations are possible.

1. Example: Unlimited breakpoints in RAM and one breakpoint in ROM/FLASH

```
Break.Set 0x100000 /Program      ; Software breakpoint 1
Break.Set 0x101000 /Program      ; Software breakpoint 2
Break.Set addr /Program          ; Software breakpoint 3
Break.Set 0x100 /Program         ; On-chip breakpoint
```

2. Example: Unlimited breakpoints in RAM and one breakpoint on a read or write access

```
Break.Set 0x100000 /Program      ; Software breakpoint 1
Break.Set 0x101000 /Program      ; Software breakpoint 2
Break.Set addr /Program          ; Software breakpoint 3
Break.Set 0x108000 /Write        ; On-chip breakpoint
```

3. Example: Two breakpoints in ROM/FLASH

```
Break.Set 0x100 /Program         ; On-chip breakpoint 1
Break.Set 0x200 /Program         ; On-chip breakpoint 2
```

4. Example: Two breakpoints on a read or write access

```
Break.Set 0x108000 /Write        ; On-chip breakpoint 1
Break.Set 0x108010 /Read         ; On-chip breakpoint 2
```

5. Example: One breakpoint in ROM/FLASH and one breakpoint on a read or write access

```
Break.Set 0x100 /Program         ; On-chip breakpoint 1
Break.Set 0x108010 /Read         ; On-chip breakpoint 2
```

Complex Breakpoints

To use the advanced features of the ICE breaker unit the **TrOnchip** command group is possible. These commands provide full access to both ICE breaker units called A and B in the TRACE32 system. Most features can also be used by setting advanced breakpoints (e.g. task selective breakpoints, exclude breakpoints). Ranged breakpoints use multiple breakpoint resources to better fit the range when the resources are available.

Direct ICE Breaker Access

It is possible to program the complete ICE breaker unit directly, by using the access class ICE. E.g. the command `Data.Set ICE:10 %Long 12345678` writes the value 12345678 to the watchpoint 1 Address Value Register. The following table lists the addresses of the relevant registers.

Address	Register
ICE:8	Watchpoint 0 Address Value
ICE:9	Watchpoint 0 Address Mask
ICE:0A	Watchpoint 0 Data Value
ICE:0B	Watchpoint 0 Data Mask
ICE:0C	Watchpoint 0 Control Value
ICE:0D	Watchpoint 0 Control Mask
ICE:10	Watchpoint 1 Address Value
ICE:11	Watchpoint 1 Address Mask
ICE:12	Watchpoint 1 Data Value
ICE:13	Watchpoint 1 Data Mask
ICE:14	Watchpoint 1 Control Value
ICE:15	Watchpoint 1 Control Mask

For more details, please refer to the ARM data sheet. It is recommended to use the **Break.Set** or **TrOnchip** commands instead of direct programming, because then no special ICEbreaker knowledge is required.

Trigger

A bidirectional trigger system allows the following two events:

- trigger an external system (e.g. logic analyzer) if the program execution is stopped.
- stop the program execution if an external trigger is asserted.

For more information, refer to the **TrBus** command.

Virtual Terminal

The command **TERM** opens a terminal window which allows to communicate with the ARM core over the ICEbreaker Debug Communications Channel (DCC). All data received from the comms channel are displayed and all data inputs to this window are sent to the comms channel. Communication occurs byte wide or up to four bytes per transfer. The four bytes ASCII mode (**DCC4A**) does not allow to transfer the byte 00. Each non-zero byte of the 32 bit word is a character in this mode. The four byte binary mode (**DCC4B**) can be used to transfer non-ASCII 32bit data (e.g. to or from a file). The three bytes mode (**DCC3**) allows binary transfers of up to 3 bytes per DCC transfer. The upper byte defines how many bytes are transferred (0 = one byte, 1 = two bytes, 2 = three bytes). This is the preferred mode of operation, as it combines arbitrary length messages with high bandwidth. The **TERM.METHOD** command selects which mode is used (**DCC**, **DCC3**, **DCC4A** or **DCC4B**).

The communication mechanism is described e.g. in the ARM7TDMI data sheet in chapter 9.11. Only three move to/from coprocessor 14 instructions are necessary.

The TRACE32 demo/arm/etc/terminal directory contains the file `TERM.CMM` which demonstrates how the communication works.

Semihosting

The command **TERM.GATE** opens a terminal window which allows to support ARM compatible semihosting. The communication can either be done by stopping the target at the SWI or by using the DCC interface channel - which provides non-stop operation of the target.

The SWI emulation mode requires to stop the target at the SWI exception vector. On ARM7 this can be done only with an on-chip or software breakpoint at location 8. On other ARM cores it can be done by enabling the ICEbreaker breakpoint at the SWI vector (**TrOnchip.Set SWI ON**). The terminal must be set to the **ARMSWI** method (**TERM.METHOD ARMSWI**). The handling of the SWI is only active when the **TERM.GATE** window is existing.

The DCC communication mode requires an target agent for the SWI. The communication is done in the **DCC3** method of the **TERM** command.

Runtime Measurement

The command **RunTime** allows run time measurement based on polling the CPU run status by software. Therefore the result will be about few milliseconds higher than the real value.

If the signal DBGACK on the JTAG connector is available, the measurement will automatically be based on this hardware signal which delivers very exact results. Please do not disable the option **SYStem.Option.DBGACK**. The runtime of the debugger accesses while the CPU is halted would also be measured, otherwise.

Coprocessors

It is not possible to access coprocessors which are not included in an ARM macrocell from debug mode. This means all coprocessors which are added to ARM cores by customers can not be accessed from debug mode.

The following coprocessors can be accessed if available in the processor:

Coprocessor 14. Please refer to the chapter **Virtual Terminal** and to your ARM documentation for details.

Coprocessor 15, which allows the control of basic CPU functions. This coprocessor can be accessed with the access class C15. For the detailed definition of the CP15 registers please refer to the ARM data sheet. The CP15 registers can also be controlled in the **PER** window.

The TRACE32 address is composed of the CRn, CRm, op1, op2 fields of the corresponding coprocessor register command

```
<MCR|MRC> p15, <op1>, Rd, CRn, CRm, <op2>
BIT0-3:CRn, BIT4-7:CRm, BIT8-10:<op2>, BIT12-14:<op1>
```

is the corresponding TRACE32 address (one nibble for each field)

The following ARM specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
SP	Supervisor Program Memory (privileged access)
UP	User Program Memory (non-privileged access)
SR	Supervisor ARM Memory (privileged access)
ST	Supervisor Thumb Memory (privileged access)
UR	User ARM Memory (non-privileged access)
UT	User Thumb Memory (non-privileged access)
U	User Memory (non-privileged access)
S	Supervisor Memory (privileged access)
R	ARM Memory
T	Thumb Memory
ICE	ICE Breaker Register (debug register; ARM7)
C15	Coprocessor 15 Register (if implemented)
ETM	Embedded Trace Macrocell Registers (if implemented)
VM	Virtual Memory (memory on the debug system)
USR	Access to Special Memory via User-Defined Access Routines
E	Run-time memory access (see **SYStem.CpuAccess** and **SYStem.MemAccess**)

To access a memory class write the class in front of the address.

Example:

```
Data.dump ICE:0--3
```

Normally there is no need to use the following memory classes: P, D, SP, UP, SR, ST, UR, UT, U, S, R, or T. The memory class is set automatically depending on the setting of **SYStem.Option.DisMode**.

The memory class ICE should only be used from very advanced users. Wrong usage may cause unpredictable problems.

Programming the On-chip FLASH of the MAC71/72xx

Some example scripts for programming of the on-chip FLASH of the MAC71/72XX can be found in the TRACE32 demo folder ...\demo\arm\flash\<file>cmm (Example: M71x2.cmm.) Please be aware that these are just an example scripts. The scripts have to be adapted to your memory layout. The FLASH programming algorithm used is based on the FLASH library provided by Freescale.

SYStem.CONFIG.state — Display target configuration

Format: **SYStem.CONFIG.state**

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

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.

SYStem.CONFIG — Configure debugger according to target topology

Format: **SYStem.CONFIG** *<parameter>*
SYStem.MultiCore *<parameter>* (deprecated)

<parameter>:
DRPOST *<bits>*
DRPRE *<bits>*
IRPOST*<bits>*
IRPRE *<bits>*

Slave [**ON** | **OFF**]
TAPState *<state>*
TCKLevel *<level>*
TriState [**ON** | **OFF**]
FILLDRZERO [**ON** | **OFF**]

The **SYStem.CONFIG** commands inform the debugger about the available on-chip debug and trace components and how to access them.

...IRPRE Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between the TAP you are describing and the TDO signal. See possible TAP types and example below.

Default: 0.

...IRPOST Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between TDI signal and the TAP you are describing. See possible TAP types and example below.

Default: 0.

...DRPRE Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TAP you are describing and the TDO signal. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below.

Default: 0.

...DRPOST Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TDI signal and the TAP you are describing. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below.

Default: 0.

Slave [**ON** | **OFF**] If several debuggers 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). The other debuggers need to have the setting **Slave ON**.

Default: OFF.
Default: ON if `CORE=...` >1 in the configuration file (e.g. config.t32).

TriState [**ON** | **OFF**] TriState has to be used if several debug cables are connected to a common JTAG port. 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.

Default: OFF.

TAPState *<state>*

This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.

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

Default: 7 = Select-DR-Scan.

TCKLevel *<level>*

Level of TCK signal when all debuggers are tristated. Normally defined by a pull-up or pull-down resistor on the target.

Default: 0.

FILLDRZERO [**ON** | **OFF**]

This changes the bypass data pattern for other TAPs in a multi-TAP JTAG chain. It changes the pattern from all “1” to all “0”. This is a workaround for a certain chip problem. It is available on the ARM9 debugger, only.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**MAC71xx** | **MAC7111** | **MAC7202** | …

Selects the processor type. If your CPU is not listed, select the type of the integrated ARM core.

Default selection: There is an auto detect mechanism which automatically selects the right CPU depending on the debugger probe.

Format: **SYStem.JtagClock** [<*frequency*> | **RTCK** | **RTCK** <*frequency*>] | **ARTCK** <*frequency*>]

<*frequency*>: **10000.** … **40000000.**
10. kHz … **40. MHz**

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

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

On some processor derivatives (e.g. 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.

When RTCK is selected, the maximum reachable frequency is limited to 10 MHz. This limit can be changed by adding the frequency parameter. A limitation is required that the JTAG clock speed can not become higher than the physical interface can manage.

Example: `SYStem.JtagClock RTCK 20MHz`

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

RTCK mode allows theoretical frequencies up to 1/6 of the 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 1/2 of the processor clock speed. The signal RTCK clocks TDI and TMS and controls the sampling of TDO. ARTCK is available on ARM7 debuggers, only.

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

Default: OFF.

If the system is locked no access to the JTAG port will be performed by the debugger. While locked the JTAG connector of the debugger is tristated. The intention of the lock command is for example to give JTAG access to another tool. The process can also be automated, see **SYStem.CONFIG TriState**.

It must be ensured that the state of the ARM core JTAG state machine remains unchanged while the system is locked. To ensure correct hand over the options **SYStem.CONFIG TAPState** and **SYStem.CONFIG TCKLevel** must be set properly. They 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, EDBGRQ must have a pull-down resistor.

Format:	**SYStem.MemAccess** <*mode*>
<*mode*>:	**NEXUS** **Denied**

Default: Denied.

If **SYStem.MemAccess** is not Denied, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program.

NEXUS Memory access is done via the Nexus Interface.

TSMon3 A run-time memory access is done via a **Time Sharing Monitor.**
The application is responsible for calling the monitor polling code periodically. The call is typically included in a periodic interrupt or in the idle task of the kernel.
Handling of interrupts when the application is stopped is possible when the background monitor is activated. Manual break is not possible and can only be emulated by polling the DCC port.
See also the example in the `demo/arm/etc/tsmon` directory.

REALMon A run-time memory access is done via the Real Monitor from ARM.

TrkMon Select TRK for Run Mode Debugging of Symbian OS. DCC is used as communication interface

GbdMon Select T32server (extended gdbserver) for Run Mode Debugging of embedded Linux. DCC is used as communication interface. For more information refer to **"Run Mode Debugging Manual Linux"** (rtos_linux_run.pdf).

Denied No memory access is possible while the CPU is executing user code.

If specific windows that display memory or variables should be updated while the program is running, select the memory class **E:** or the format option **%E**.

```
Data.dump E:0x100

Var.View %E first
```

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

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

<*mode*>: **Down**
NoDebug
Go
Attach
StandBy
Up

Down Disables the debugger (default). The state of the CPU remains unchanged. The JTAG port is tristated.

NoDebug Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.

Go Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or external trigger.

Attach 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.

Standby Selects a special start-up procedure of the debugger.

- The debugger checks continuously if target power is ok.
- Then it pulls RESET and initializes the JTAG interface to enter debug mode and the Nexus cell as far as possible. Then it releases RESET
- The debugger checks then continuously if RESET is really deasserted.
- After entering debug mode, all the rest of the necessary initialization will be done. (Breakpoint settings, trace setup, etc.)
- If everything is done, the debugger starts the user program from Reset vector automatically.

Up 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.

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

Default: OFF.

Workaround for a special customer configuration. It suppresses all debugger accesses to memory area 0x0-0x1f.

SYStem.Option.BUGFIX Breakpoint bug fix for ARM7TDMI-S REV2

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

Default: OFF.

You need to activate this option when having an ARM7TDMI-S Rev2. The “consecutive breakpoint” bug is fixed in ARM7TDMI-S Rev3.

With this option activated and ARM7TDMS selected as CPU type, we enable the software breakpoint workaround as described in the ARM errata of ARM7TDMI-S Rev2. Software breakpoints are set as undefined opcodes that cause the core to enter the undefined opcode handler. The debugger tries to set a breakpoint at the undef vector (either software or on-chip). When a breakpoint is reached the core will take the undefined exception and stop at the vector. The debugger detects this state and displays the correct registers and cpu state. This workaround is only suitable where undefined instruction trap handling is not being used.

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

Default: OFF.

This option selects the byte ordering mechanism. For correct operation the following three settings must correspond:

- this option
- the compiler setting (-li or -bi compiler option)
- the level of the ARM BIGEND input pin (ARM7x0T the bit in the CP15 control register)

SYStem.Option.CFLUSH FLUSH the cache before step/go

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

Default: ON.

If this option is ON the cache is invalidated automatically before each step or go command. This is necessary to maintain software breakpoint consistency.

SYStem.Option.DBGACK DBGACK active on debugger memory accesses

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

Default: ON.

If this option is on the DBGACK signal remains active during memory accesses in debug mode. If the DBGACK signal is used to freeze timers or to disable other peripherals it is strictly recommended to enable this option.

Disabling of this option may be useful for triggering on memory accesses from debug mode (only useful for hardware developers).

Format: **SYStem.Option.DisMode** <*option*>

<*option*>: **AUTO**
ACCESS
ARM
THUMB
THUMBEE

This command specifies the selected disassembler. Default: AUTO.

AUTO The information provided by the compiler output file is used for the disassembler selection. If no information is available it has the same behavior as the option ACCESS.

ACCESS The selected disassembler depends on the T bit in the CPSR or on the selected access class. (e.g. `Data.List SR:0` for ARM mode or `Data.List ST:0` for THUMB mode).

ARM Only the ARM disassembler is used (highest priority).

THUMB Only the THUMB disassembler is used (highest priority).

THUMBEE Only the THUMB disassembler is used which supports the Thumb-2 Execution Environment extension (highest priority).

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 (ARM7) line on the JTAG connector. This is necessary if nRESET is no open collector or tristate signal.

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

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

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 a single step, the interrupt mask bits are restored to the value before the step.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

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

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 a single step, the interrupt mask bits are restored to the value before the step.

SYStem.Option.OVERLAY Enable overlay support

Format: **SYStem.Option.OVERLAY** [**ON** | **OFF** | **WithOVS**]

Default: OFF.

ON Activates the overlay extension and extends the address scheme of the debugger with a 16 bit virtual overlay ID. Addresses therefore have the format *<overlay_id>***:***<address>*. This enables the debugger to handle overlaid program memory.

OFF Disables support for code overlays.

WithOVS Like option **ON**, but also enables support for software breakpoints. This means that TRACE32 writes software breakpoint opcodes to both, the *execution area* (for active overlays) and the *storage area*. This way, it is possible to set breakpoints into inactive overlays. Upon activation of the overlay, the target's runtime mechanisms copies the breakpoint opcodes to the execution area. For using this option, the storage area must be readable and writable for the debugger.

Example:

```
SYStem.Option.OVERLAY ON
Data.List 0x2:0x11c4                    ; Data.List <overlay_id>:<address>
```

SYStem.Option.INTDIS Disable all interrupts

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

Default: OFF.

If this option is ON all interrupts to the ARM core are disabled.

SYStem.Option.LOCKRES Go to "Test-Logic Reset" when locked

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

This command is only available on obsolete ICD hardware. The state machine of the JTAG TAP controller is switched to Test-Logic Reset state (ON) or to Run-Test/Idle state (OFF) before a **SYStem.LOCK ON** is executed.

SYStem.Option.NOIRCHECK No JTAG instruction register check

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

Default: OFF.

Bug fix for derivatives which do not return the correct pattern on a JTAG instruction register (IR) scan. When activated the returned pattern will not be checked by the debugger. On ARM7 also the check of the return pattern on a scan chain selection is disabled.

The option is automatically activated when using **SYStem.Option.TURBO**.

Format: **SYStem.Option.PC** *<address>*

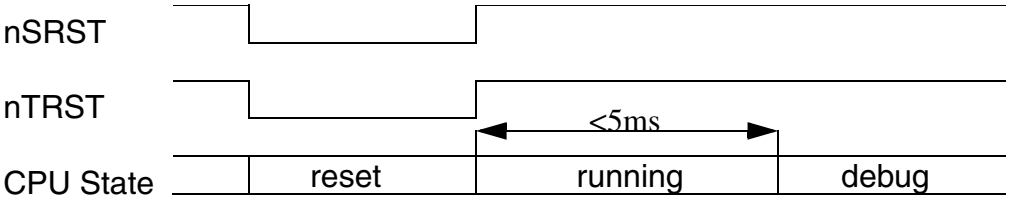
Default address: 0.

After each load or store operation from debug mode the ARM core makes some instruction fetches from memory. These fetches are not necessary for the debugger, but it is not possible to suppress them.

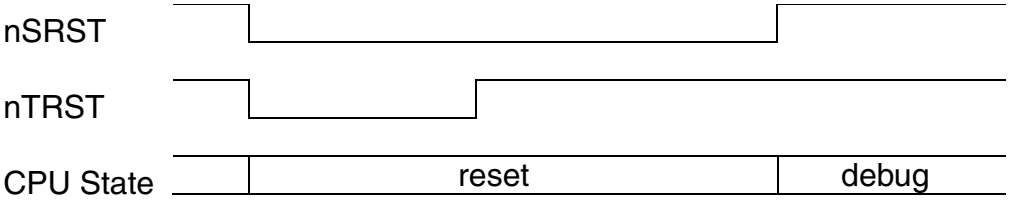
This option allows to specify the base address of these fetches. The fetch address is anywhere within a 64 KByte block that begins at the specified base address. It is necessary to modify this option if these fetches go to aborted memory locations.

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

Default: ON. This option has to be disabled if the nTRST line is connected to the nRESET (ARM7) line on the target. In this case the CPU executes some cycles while the **SYStem.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 deassertion of reset and the entry to debug mode is the time for this JTAG sequence.



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 ResBreak option is enabled the debugger first deasserts nTRST, then it executes a JTAG sequence to set the DBGRQ bit in the ICE breaker control register and then it deasserts nRESET.



SYStem.Option.RisingTDO Target outputs TDO on rising edge

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

Default: OFF.

Bug fix for chips which output the TDO on the rising edge instead of on the falling.

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

Default: ON. If the ABORT (if AMBA: BERROR) line becomes active during a system speed access the ARM core can change to ABORT mode. When this option is on this change of mode is indicated by the warning '**emulator berr error**'.

SYStem.Option.SOFTLONG Use 32-bit access to set breakpoint

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

Default: OFF. This option instructs the debugger to use 32-bit accesses to patch the software breakpoint code.

SYStem.Option.SOFTWORD Use 16-bit access to set breakpoint

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

Default: OFF. This option instructs the debugger to use 16-bit accesses to patch the software breakpoint code.

SYStem.Option.SPLIT Access memory depending on CPSR

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

Default: OFF. If this option is ON, the debugger does privileged or non-privileged memory access depending on the current CPU mode (CPSR register). If this option is OFF the debugger accesses the memory in privileged mode except another access mode is requested. This feature is only available if a DEBUG INTERFACE (LA-7701) is used for the ARM7.

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

Default: OFF.

If set to ON, software breakpoints are used for single stepping on assembler level (advanced users only).

SYStem.Option.TRST

Allow debugger to drive TRST

Format: **SYStem.Option**.**TRST** [**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.

SYStem.Option.TURBO

Speed up memory access

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

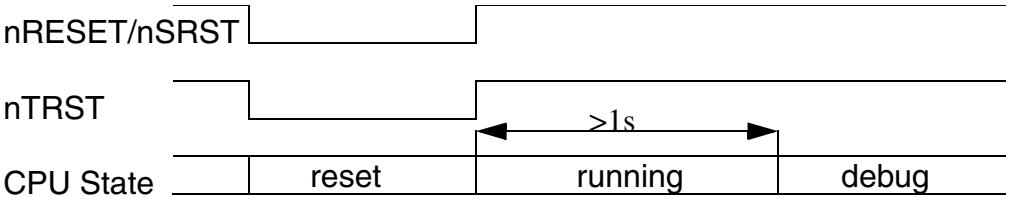
Default: OFF. If TURBO is disabled the CPU checks after each system speed memory access in debug mode if the CPU has finished the corresponding cycle. This check will significantly reduce the down- and upload speed (30-40%).

If TURBO is enabled the CPU will make no checks. This may result in unpredictable errors if the memory interface is slow. Therefore it is recommended to use this option only for a program download and in case you know that the memory interface is fast enough to take the data with the speed they are provided by the debugger.

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

Default: OFF.

If **SYStem.Option.WaitReset** is enabled and **SYStem.Option.ResBreak** is disabled the debugger waits after the deassertion of nRESET (ARM7) and nTRST before the first JTAG activity starts. It waits for at least 1 s, then it waits until nRESET/nSRST is released from target side, but at maximum 35 s. During this time the ARM core may execute some code, e.g. to enable the JTAG port. If **SYStem.Option.ResBreak** is enabled the **SYStem.Option.WaitReset** is ignored.



SYStem.RESetOut Assert nRESET/nSRST on JTAG connector

Format: **SYStem.RESetOut**

If possible (nRESET is open collector), this command asserts the nRESET line on the JTAG connector. While the CPU is in debug mode this function will be ignored. Use the **SYStem.Up** command if you want to reset the CPU in debug mode.

NEXUS.BTM Control for branch trace messages

Format: **NEXUS.BTM** [**ON** | **OFF**]

Control for the NEXUS branch trace messages.

ON (default) NEXUS outputs branch trace messages.

OFF No branch trace messages are output by NEXUS.

NEXUS.ThumbBTM Control for branch trace messages

Format: **NEXUS.ThumbBTM** [**ON** | **OFF**]

Control for the NEXUS branch trace messages in Thumb mode.

ON (default) NEXUS outputs branch trace messages.

OFF No branch trace messages are output by NEXUS.

NEXUS.OTM Control for ownership trace messages

Format: **NEXUS.OTM** [**ON** | **OFF**]

ON NEXUS outputs ownership trace messages.

OFF No ownership trace messages are output by NEXUS.

NEXUS.WTM Control for watchpoint messages

Format: **NEXUS.WTM** [**ON** | **OFF**]

ON NEXUS outputs watchpoint messages.

OFF No watchpoint messages are output by NEXUS.

NEXUS.DTM Control for data trace messages

Format: **NEXUS.DTM** [**Read** | **Write** | **ReadWrite** | **OFF**]

OFF No data trace message are output by the NEXUS cell

READ NEXUS output data trace message for read accesses.

Write NEXUS output data trace message for write accesses.

ReadWrite NEXUS output data trace message for read and write accesses

NEXUS.PortMode Set NEXUS trace port frequency

Format: **NEXUS.PortMode 1/1 | 1/2 | 1/4 | 1/8**

Sets the NEXUS trace port frequency. For parallel NEXUS, the setting is the system clock divider. For Aurora NEXUS, the setting is a fixed bit clock which is independent of the system frequency.

NEXUS.PortSize Set trace port width

Format: **NEXUS.PortMode MOD2 | MOD8**

Sets the NEXUS port width to the number of used MDO pins.

Format: **NEXUS.UBA** *<address>*

The UBA defines the memory mapped user base address for ownership trace register.

NEXUS.STALL Stall the program execution

Format: **NEXUS.STALL** [**ON** | **OFF**]

Stall the program execution whenever the NEXUS-FIFO threatens to overflow.

For older versions of the processor do not switching on the stall option. It can not prevent a trace data loss in all cases.

The **TrOnchip** command group provides full access to both ICE Breaker units called A and B. Most of the features can also utilized easier by setting regular breakpoints (**Break.Set** command).

TrOnchip.RESet — Reset on-chip trigger settings

Format: **TrOnchip.RESet**

Resets all TrOnchip settings.

TrOnchip.CONVert — Extend the breakpoint range

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

The ICE-breaker does not provide resources to set an on-chip breakpoint to an address range. Only bit masks can be used to mark a memory range with a breakpoint.

If **TrOnchip.Convert** is set to **ON** (default) and a breakpoint is set to a range, this range is extended to the next possible bit mask. The result is that in most cases a bigger address range is marked by the specified breakpoint. This can be easily controlled by the **Data.View** command.

If **TrOnchip.Convert** is set to **OFF**, the debugger will only accept breakpoints which exactly fit to the on-chip breakpoint hardware.

This setting affects all on-chip breakpoints.

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

<*mode*>: **AORB**
AANDB
BAFTERA

Defines the way in which unit A and B are used together.

AORB Stop the program execution if unit A or unit B match.

AANDB Stop the program execution if both units match.

BAFTERA Stop the program execution if first unit A and then unit B match.

TrOnchip.A Programming the ICE breaker module

TrOnchip.A.Value Define data selector

Format: **TrOnchip.A.Value** <*hexmask*> | <*bitmask*>
TrOnchip.B.Value <*hexmask*> | <*bitmask*>

Defines the two data selectors of ICE breaker as hex or binary mask (x means don't care). If you want to trigger on a certain byte or word access you must specify the mask according to the address of the access. E.g. you make a byte access on address 2 and you want to trigger on the value 33, then the necessary mask is 0xx33xxxx.

Format: **TrOnchip.A.Size** <*size*>
TrOnchip.B.Size <*size*>

<*size*>: **OFF**
Byte
Word
Long

Defines on which access size when ICE breaker stops the program execution.

TrOnchip.A.CYcle Define access type

Format: **TrOnchip.A.CYcle** <*cycle*>
TrOnchip.B.CYcle <*cycle*>

<*cycle*>: **OFF**
Read
Write
Access
Execute

Defines on which cycle the ICE breaker stops the program execution.

OFF	Cycle type doesn't matter.
Read	Stop the program execution on a read access.
Write	Stop the program execution on a write access.
Access	Stop the program execution on a read or write access.
Execute	Stop the program execution on an instruction is executed.

Format: **TrOnchip.A.Address** <*selector*>
TrOnchip.B.Address <*selector*>

<*selector*>: **OFF**
Alpha
Beta
Charly

The address/range for an address selector can not be defined directly. Set an breakpoint of the type Alpha, Beta or Charly to the address/range.

```
Break.Set 1000 /Alpha             ; set an Alpha breakpoint to 1000
TrOnchip.A.Address Alpha          ; use Alpha breakpoint as address
                                  ; selector for the unit A

Var.Break.Set flags[3] /Beta      ; set a Beta breakpoint to flags[3]
TrOnchip.B.Address Beta           ; use Beta breakpoint as address
                                  ; selector for the unit B
```

TrOnchip.A.Trans Define access mode

Format: **TrOnchip.A.Trans** <*mode*>
TrOnchip.B.Trans <*mode*>

<*mode*>: **OFF**
User
Svc

Defines in which mode ICE breaker should stop the program execution.

OFF	Mode does not matter.
User	Stop the program execution only in user mode.
Svc	Stop the program execution only in supervisor mode.

Format: **TrOnchip.A.Extern** <*mode*>
TrOnchip.B.Extern <*mode*>

<*mode*>: **OFF**
Low
High

Defines if the EXTERN lines are considered by unit A or unit B.

TrOnchip.state Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

Filter and Trigger provided by the Processor

The internal watchpoints of the MAC71/72xxcan be used to control the output of the trace data. The following actions for the NEXUS trace are provided by the **Break.Set** command:

Actions for the Trace (provided by the CPU)	
TraceON	Switch the sampling to the trace ON on the specified event.
TraceOFF	Switch the sampling to the trace OFF on the specified event.
TraceTrigger	Stop the sampling to the trace on the specified event. A trigger delay is possible.
BusTrigger	Generate a 100 ns high pulse on the trigger connector of the PowerTrace on the defined trigger event.
BusCount	Use the TRACE32 counter to analyze the trigger event.
WATCH	Set a watchpoint on the event. The CPU will trigger the EVTO pin if the event occurs.
Alpha - Echo	Used to configure DMA trace and trigger events. These conditions can be assigned to DMA trace actions in the TrOnchip window.
SPOT	Stops user program, updates all windows on the screen and continues user program execution

```
Var.Break.Set flags[3] /Write /TraceEnable

                                        ; NEXUS outputs only trace messages
                                        ; for write accesses to flags[3]

Var.Break.Set flags /Write /TraceData

                                        ; NEXUS outputs the complete
                                        ; program flow and all write
                                        ; accesses to the variable flags

Break.Set func2 /Program /TraceON       ; NEXUS switches the trace output
Break.Set v.end(func2)-3 /TraceOFF      ; to ON at the entry to func2 and
                                        ; switches the trace output to OFF
                                        ; at the exit of func2
```

TrOnchip.EVTI Allow the EVTI signal to stop the program execution

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

ON Allow the EVTI signal to stop the program execution (it's much faster).

OFF The program execution is stopped by sending a break sequence via JTAG.

Example: Stop the program execution on the falling edge of the external signal on the TRIGGER connector of POWERTRACE / ETHERNET.

```
TrOnchip.EVTI ON        ; Enable fast stop via an external signal

; Configure the internal trigger bus

TrBus.RESet             ; Reset trigger bus settings

TrBus.Connect In        ; Configure TRIGGER as input

TrBus.Mode Falling      ; The trigger is active on the falling edge of
                        ; the connected signal

TrBus.Set Break ON      ; Define Break as the trigger event

TrBus.Out Break OFF
```

Format: **TrOnchip.EXTernal** *<source>*

<source>: **OFF**
IN0
IN1

Generate a trigger for the trace on a high pulse (at least 20 ns) on the IN0 or the IN1 connector on the NEXUS Adapter. IN0 and IN1 are ORed for the trigger.

```
TrOnchip.EXTernal IN0

Go
```

Mechanical Description of the 20-pin Debug Cable

This connector is defined by ARM and we recommend this connector for all future designs. Our debuggers "JTAG Debugger for ARM7" (LA-7746) are supplied with this connector.

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	VSUPPLY (not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS|TMSC|SWDIO	7	8	GND
TCK|TCKC|SWCLK	9	10	GND
RTCK	11	12	GND
TDO|-|SWO	13	14	GND
RESET-	15	16	GND
DBGRQ	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.

- 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 5 V supply environment is not supported! This would cause damage on the ICD.

  The newer debug cables (October 2003 and newer) can work in a voltage range of 0.4 … 5.0 V (5.25 V). Check the serial number of the debug cable. The first four digits mean *<year> <month>*.

- 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 absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a hand over 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.

- EDBGRQ 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.

Mechanical Description of the 14-pin Debug Cable

This connector was defined by ARM. We do not recommend to use this connector in future designs. Our debugger “JTAG Debugger for ARM7” (LA-7740, obsolete) is supplied with this connector. An adapter to the 20-pin connector (see above) and vice versa is available (LA-7747: JTAG ARM Converter 14-20).

Signal	Pin	Pin	Signal
VCCS	1	2	GND
TRST-	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
TDO	11	12	SRST-
VTREF	13	14	GND

This is a standard 14 pin double row (two rows of seven pins) connector (pin to pin spacing: 0.100 in.).

Electrical Description of the 14-pin Debug Cable

- TCK, TMS, TDI and nTRST are driven by a VHC125 driver which is supplied with VCCS. Therefore the ICD can work in an voltage range of (2.25 V) 2.5… 5.0 V (5.5 V). In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the JTAG port. The TMS, TDI and nTRST lines have a 47k pull-up resistor in the ICD connector. In environments where multiple tools can access the JTAG port, it is absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a endeavor between different tools.
- TDO is ICD input only and needs standard TTL level.
- VCCS is used as a sense line for the target voltage. It is also used as supply voltage for the output driver of the ICD interface to make an adaptation to the target voltage (I(VCCS) approx. 3 mA).
- nRESET (= nSRST) 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 22 k pull-up resistor is included in the ICD connector. The debugger will only assert a pulse on nRESET when the SYStem.Up command is executed. If it is ensured that the ARM is able to enter debug mode every time (no hang-up condition), the nRESET line is optional.

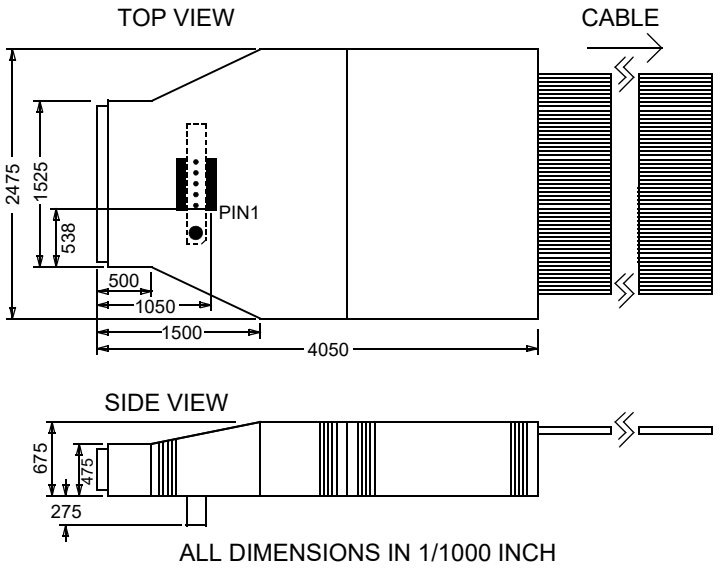
Pinout MICTOR

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
MDO09 (VENIO0)	5	6	CLKOUT
RSTOUT (VENIO2)	7	8	MDO08 (VENIO3)
RSTIN-	9	10	EVTI-
TDO	11	12	VTREF
MDO10 (VENIO4)	13	14	RDY-
TCK	15	16	MDO07
TMS	17	18	MDO06
TDI	19	20	MDO05
TRST-	21	22	MDO04
MDO11 (VENIO1)	23	24	MDO03
N/C	25	26	MDO02
TDET (TOOLIO2)	27	28	MDO01
ARBGRT(TOOLIO1)	29	30	MDO00
N/C	31	32	EVTO-
N/C	33	34	MCKO
ARBREQ(TOOLIO0)	35	36	MSEO1-
N/C	37	38	MSEO0-

- There are 5 additional GND pins in the center of the connector, which are not shown in the list.
- TCK, TMS, TDI and nTRST are driven by a driver which is supplied with a voltage derived by VTREF. It can be disabled to give another tool access to the JTAG port. The TMS, TDI and nTRST lines have no pull-up resistor in the probe. TDO is ICD input only and needs standard TTL level.
- VTREF is used as a sense line for the target voltage. It is also used as supply voltage for the output driver of the Debug interface as well as the supply of the AUX port input buffer.
- RSTI- is used by the debugger to reset the target CPU and to detect a reset on the target. It is driven by an open drain buffer. A 22 k pull-up resistor is included on the probe. The debugger will only assert a pulse on RSTI- when the SYStem.Up command is executed. If it is ensured that the ARM is able to enter debug mode every time (no hang-up condition), the nRESET line is optional.

Mechanical Dimension

Dimension
LA-7620 NEXUS-MAC71XX



Adapter

Not necessary.

Operation Voltage

Adapter	OrderNo	Voltage Range
NEXUS Debugger and Trace for MAC71xx/MAC72xx	LA-7620	1.6 .. 3.6 V