LAUTERBACH
DEVELOPMENT TOOLS

AVR32 Debugger and NEXUS Trace

AVR32 Debugger and NEXUS Trace

AVR32 Debugger and NEXUS Trace

Version 06-Jun-2024

Warning

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.

Introduction

This manual serves as a guideline for debugging AVR32 cores and describes all processor-specific TRACE32 settings and features.

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

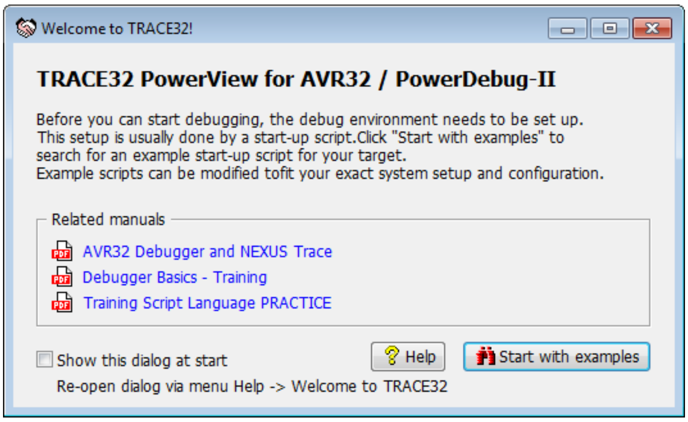
Brief Overview of Documents for New Users

Architecture-independent information:

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

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- To get started with the most important manuals, use the **Welcome to TRACE32!** dialog (**WELCOME.view**):



TRACE32 PowerView for AVR32 / PowerDebug-II

Before you can start debugging, the debug environment needs to be set up.
This setup is usually done by a start-up script.Click "Start with examples" to
search for an example start-up script for your target.
Example scripts can be modified tofit your exact system setup and configuration.

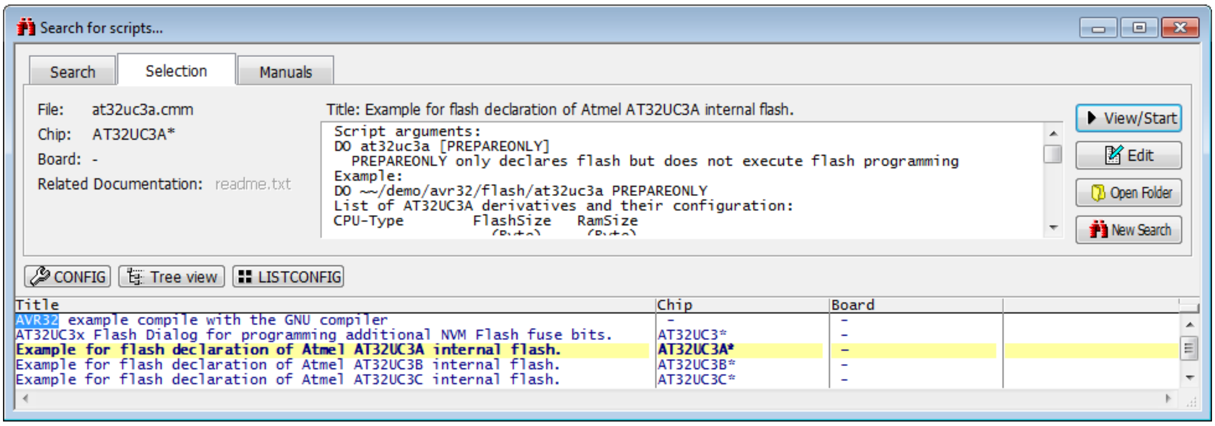
Related manuals
AVR32 Debugger and NEXUS Trace
Debugger Basics - Training
Training Script Language PRACTICE

Show this dialog at start
Re-open dialog via menu Help -> Welcome to TRACE32

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.



File: at32uc3a.cmm
Chip: AT32UC3A*
Board: -
Related Documentation: readme.txt

Title: Example for flash declaration of Atmel AT32UC3A internal flash.

```
Script arguments:
DO at32uc3a [PREPAREONLY]
  PREPAREONLY only declares flash but does not execute flash programming
Example:
DO ~~/demo/avr32/flash/at32uc3a PREPAREONLY
List of AT32UC3A derivatives and their configuration:
CPU-Type        FlashSize   RamSize
```

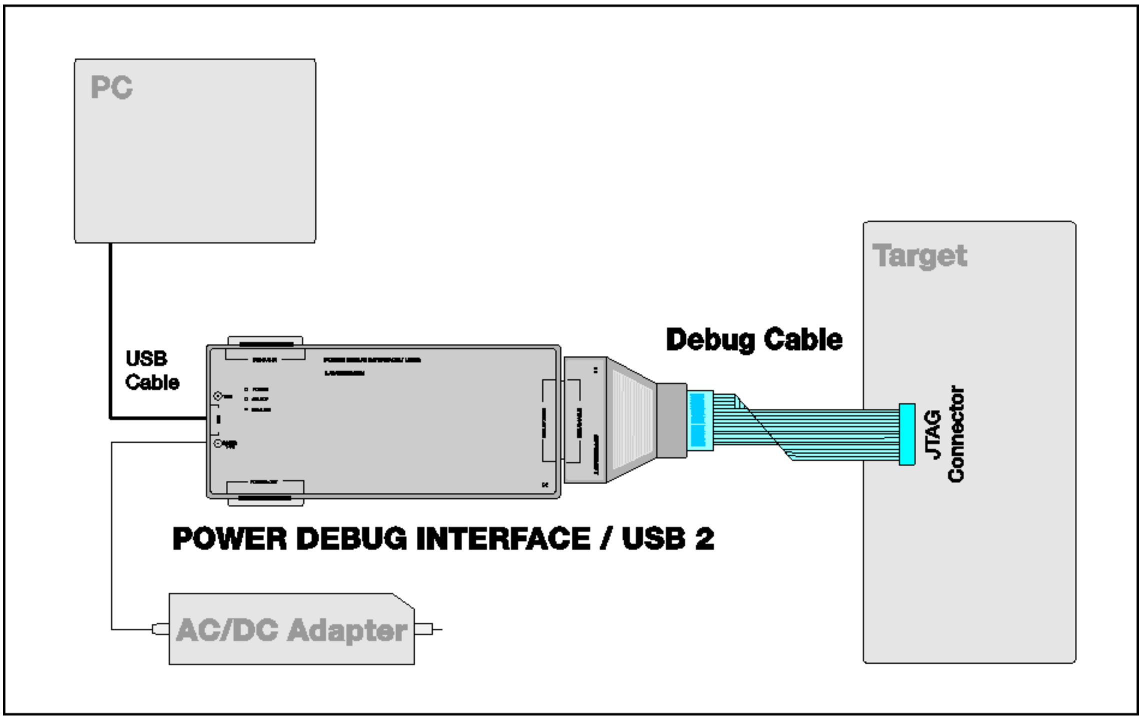
Title	Chip	Board
AVR32 example compile with the GNU compiler	-	-
AT32UC3x Flash Dialog for programming additional NVM Flash fuse bits.	AT32UC3*	-
Example for flash declaration of Atmel AT32UC3A internal flash.	AT32UC3A*	-
Example for flash declaration of Atmel AT32UC3B internal flash.	AT32UC3B*	-
Example for flash declaration of Atmel AT32UC3C internal flash.	AT32UC3C*	-

You can also manually navigate in the `~~/demo/avr32/` subfolder of the system directory of TRACE32.

Configuration

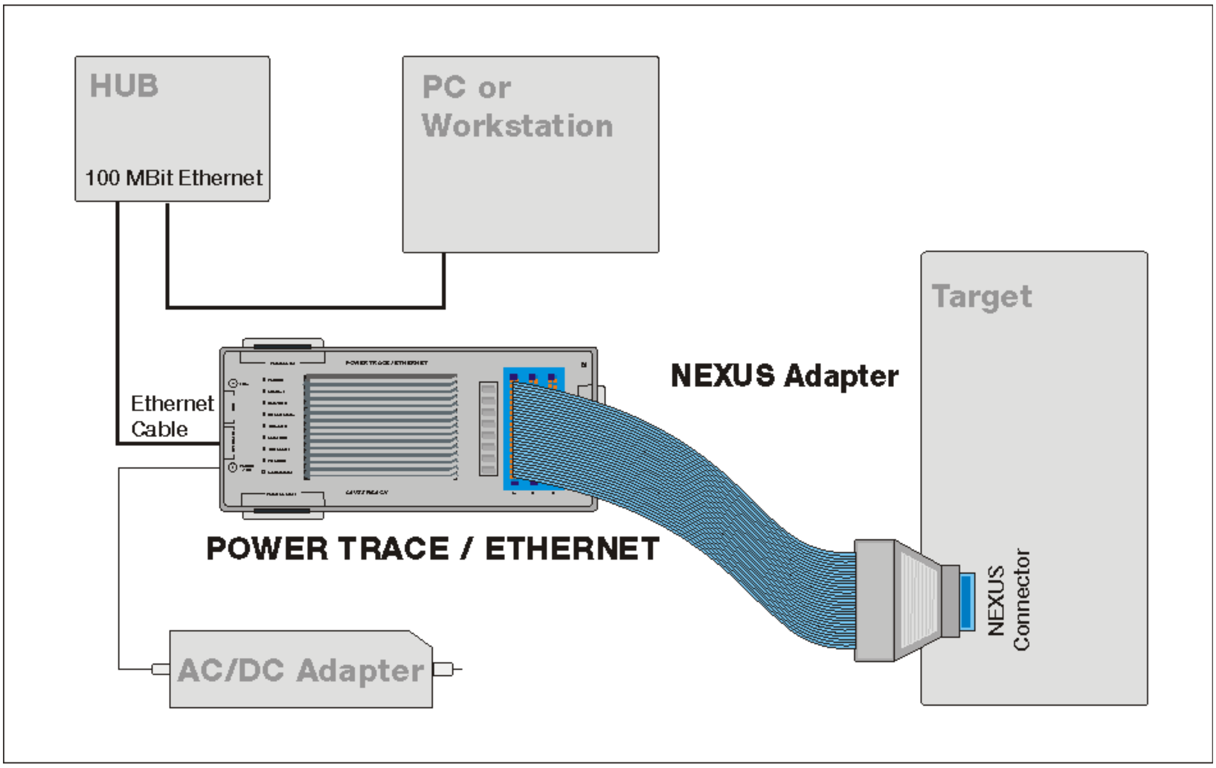
Debugger

Example configuration for an AVR32 debugger.



Debugger and NEXUS Trace

Example configuration for an AVR32 debugger and NEXUS trace.



Quick Start

Starting up the debugger is done as follows:

1. Select the device prompt B (BDM debugger) and reset TRACE32.

   ```
   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 UC3A0512
   ```

   This command selects the CPU type.

   The AP7 is not supported at the moment, but it will be in the future.

NOTE:	For a multicore target it is most likely necessary to configure the multicore settings using **SYStem.CONFIG** before continuing.

3. Inform the debugger about the cashable address range (FLASH/EEPROM).

   ```
   MAP.UpdateOnce p:0x8000--0xffff
   ```

   This is important to speed up the TRACE32 PowerView GUI responsiveness. The specified address range will be accessed only once after a break, thus avoiding unnecessary memory accesses.

4. Reset the target and enter debug mode.

   ```
   SYStem.Mode Up
   ```

   This command resets the CPU on the target, enables On-Chip-Debug Mode and issues a breakpoint right after the reset interrupt routine.The CPU stops executing any instruction, and the user is able to download and test the code. After this command is executed, it is possible to access memory and registers.

5. Load the program.

```
Data.LOAD.Elf userpgm           ; ELF specifies the format of the
                                ; symbol and debug information
```

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 Guide D"** (general_ref_d.pdf).

A typical start sequence of the AVR32 is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

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

RESet                            ; Reset the TRACE32 software

MAP.UpdateOnce p:0x8000--        ; Specify the address range for caching
0xffff

WinCLEAR                         ; Clear all windows

SYStem.Up                        ; Reset the target and enter debug mode

Data.LOAD.Elf sieve.elf          ; Load the target application

                                 ; Set the stack pointer to address 8000

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

List.Mix                         ; Open source code window          *)

Register.view /SpotLight         ; Open register window             *)

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

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

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

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

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

Error Message	Event	Reason
Target power fail	**SYStem.Mode.Up**	See below.
Target processor in reset	**SYStem.Down**	See below.
Target not connected or JTAG chain not configured correctly: Returned IR[1:0] != “01”	**SYStem.Mode.Up** **SYStem.Mode.Go**	The debugger expects to receive the bit sequence “01” for every command that is sent over JTAG. If this is not the case, an error message is displayed. Check the JTAG connections.
The number of *<number>* accessed bytes in memory is not a multiple of the access size *<size>* bytes.	No special event	Internal error, please consult your Lauterbach representative.
Memory address *<address>* is not aligned to access size *<size>*.	No special event	Internal error, please consult your Lauterbach representative.
Invalid memory access size: *<size>* bytes (@ address *<address>*)	No special event	Internal error, please consult your Lauterbach representative.
Memory access timeout: Reading from address *<address>*	No special event	Corrupted JTAG connection. Check JTAG hardware and settings.

Typically the **SYStem.Up** command is the first command of a debug session where communication with target is required. If you receive error messages like “debug port fail” or “debug port time out” while executing this command, this may have the reasons below. “target processor in reset” is just a follow-up error message.

- Open the **AREA.view** window to display all error messages.
- If the target has no power or the debug cable is not connected to the target, this results in the error message “target power fail”.
- Did you select the correct core type with **SYStem.CPU** *<cpu>*?
- There is an issue with the JTAG interface. Maybe there is the need to set jumpers on the target to connect the correct signals to the JTAG connector. The debugger will not work, for example, if nTRST signal is directly connected to ground on target side.
- The target is in an unrecoverable state. Re-power your target and try again.
- The default JTAG clock speed is too fast. In this case try **SYStem.JtagClock 50kHz** and optimize the speed when you got it working.
- The core is used in a multicore system and the appropriate multicore settings for the debugger are missing. See for example **SYStem.CONFIG IRPRE**. This is the case if you get a value.
- The core has no clock.
- The core is kept in reset.
- There is a watchdog which needs to be deactivated.

Special Nexus Trace Troubleshooting

For the case the debugger is working, but Nexus trace does not deliver reliable results, one can try the test instruction **DIAG 3016** and watch the result in the **AREA.view** window.
This test command does a hardware test of all relevant Nexus signals, independent on the Nexus trace mode. The **AREA** window delivers information about which signals possibly stack at logical High or LOW or are possibly connected to other Nexus signals. (walking H test).

FAQ

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

AVR Specific Implementations

Breakpoints

Software Breakpoints

If a software breakpoint is set, the corresponding program code is replaced by a break instruction. As soon as the program stops, the beak instruction is replaced with the original code. Thus software breakpoints can only be applied to program code residing in a RAM.

There is no restriction to the number of software breakpoints.

On-chip Breakpoints for Instructions

The AVR32 architecture provides six on-chip breakpoint registers for the program counter. The on-chip breakpoints are generally used to stop the program execution in ROM/Flash area. These on-chip breakpoint registers can also be used to define an address range where the program should stop. In order to define a range, two on-chip breakpoint registers are used; one for the address and another one to define the mask bits which means that only 3 breakpoint ranges can be defined using the on-chip breakpoint registers.

On-chip Breakpoints for Data

Data breakpoints are used to analyze the read and write accesses to global variables. The data breakpoints can be triggered with respect to the data address or access type, i.e. read, write or both, or the data value. There are a total of 2 on-chip data breakpoint registers available to the user.

In case of an on-chip data breakpoint, every load and store instruction is checked with respect to the breakpoint address, access type and the value. The data breakpoints are especially useful to find out when a global variable is written with a certain value. It is not possible to implement a similar breakpoint in software without affecting the real-time behavior of the system. Since the load and store instructions work on RAM, data breakpoints always point to addresses on RAM.

Example for Breakpoints

Some examples of software and hardware (i.e. on-chip) breakpoints are given below with the following assumption about the memory map.

- **FLASH address range from 0x1000--0x1fff**
- **RAM address range from 0x000--0x0fff**

In the first example, the breakpoint address lies inside the RAM area, a software breakpoint is automatically set. The option **/Program** is the default option and not required but displayed here for clarification.

```
Break.Set 0x0110  /Program ; Software breakpoint
```

If the breakpoint address lies within the Flash address range, it is automatically recognized and a hardware breakpoint is set.

```
Break.Set 0x1110 /Program ; Hardware breakpoint
```

This example specifies a data breakpoint which triggers when a read attempt is made at the address 0x0110.

```
Break.Set 0x0110 /Read; Data breakpoint for read access
```

Similarly, the break command below triggers when a write attempt is made at the address 0x0110.

```
Break.Set 0x0110 /Write; Data breakpoint for write access
```

This last example demonstrates a data breakpoint trigger mechanism for both read and write attempts.

```
Break.Set 0x0110 /ReadWrite; Data breakpoint for read and write accesses
```

Filter and Trigger for the NEXUS Trace

Filter and Trigger provided by the Processor (Simple Trigger Unit - STU)

The internal watchpoints of the AVR32 can be used to control trace message output. The following actions for the NEXUS trace are provided by the **Break.Set** command:

Actions for the Trace	
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.
SPOT	Stops user program, updates all windows on the screen and continues user program execution

Trigger

```
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
Break.Set Var.END(func2)-3 /TraceOFF         ; output to ON at the entry to
                                             ; func2 and switches the trace
                                             ; output to OFF at the exit of
                                             ; func2
```

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.

There is further document **STU-AVR32.PDF** which illustrates the Simple Trigger Unit (STU). The STU is just available for the Nexus probe, not for the JTAG dongle.

Runtime Measurement

The command **RunTime** allows run time measurement based on polling the CPU run status by software or by hardware. Therefore the result can be about a few milliseconds more than the real value.

As an idea one can expect about **6 ms** more in case of the JTAG Dongle (RT start stop just SW controlled) and about **1.2 us** in case of the Nexus probe.

Other Useful Trace Commands

Trace.TERMination ON | OFF	Use trace line termination of NEXUS adapter.
Trace.TestFocus	Test Trace port recording.

Memory Classes

The following memory access classes are available:

Access Class	Description
D	Data
P	Program
SR	System Registers
ED	Run-time data memory access (see **SYStem.MemAccess**)
EP	Run-time program memory access (see **SYStem.MemAccess**)

To access a memory class, write the class in front of the address. For example, use ED to access the data memory during run-time.

```
Data.dump ED:0x00
```

The memory class SR is used to denote the special system registers and available only during a CPU break.

```
Data.dump SR:0x00
```

Since the AVR32 architecture uses the same address range for both data and instructions, the memory access classes D and P in fact are same. So the following two examples return the same results.

```
Data.dump D:0x100
```

```
Data.dump P:0x100
```

Programming the On-chip FLASH of the AVR32

Some example PRACTICE scripts for programming the on-chip FLASH of the AVR32 can be found in the TRACE32 demo folder ~~/demo/avr32/flash/*.cmm, where * stands for the script file name, e.g. at32uc3a.cmm.

Please be aware that these are just 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 Atmel.

Special Hints, Restrictions, and Known Problems

Hints

- **JTAG/NEXUS:** After startup, OSCILLATOR0 will automatically be selected as clock source of the device. This is a difference to normal startup without the debugger.
  The command **DIAG 3018 0/1** allows to disable/enable OSCILL0 that feature permanently for a debug session.

Restrictions

- **JTAG:** Runtime counter causes about 6 ms mismatch.

Known Problems

- **JTAG/NEXUS:** Help system not available yet
- **NEXUS:** STU function DE-Pulse not implemented yet
- **NEXUS:** EVTI trigger working, but “Warning: CPU already in *Break* mode !”
  The warning can be ignored.
- **NEXUS:** Testfocus not implemented yet. (User DIAG 3016 meanwhile)
- **NEXUS:** SQA mode delivers Flowerror during trace list.
- **NEXUS:** If Data Trace is activated, data access messages can occasionally not yet properly be displayed, related to the corresponding code.
- **JTAG/NEXUS:** In-line assembler is not implemented yet.

NOTE:	All problems will be fixed in one of the next SW versions without notice!

Trace Extension

The AVR32 family offers NEXUS class 2+ or 3 trace features.

Depending on the debugger configuration (Debug Cable or Nexus Adapter), trace features are available or not. Device internal trace is not supported yet.

SYStem.CONFIG.state — Display target configuration

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

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

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

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

Format:	**SYStem.CONFIG** *<parameter>*
<parameter>:	**IRPRE** *<bits>* **IRPOST***<bits>* **DRPRE** *<bits>* **DRPOST** *<bits>* **IRLength** *<bits>* **MultiCoreLocal** [**ON** | **OFF**] **CoreNumber** *<number>* **TriState** [**ON** | **OFF**] **Slave** [**ON** | **OFF**] **TAPState** *<state>* **TCKLevel** *<level>*

If there is more than one TAP controller in the JTAG chain, the chain must be defined to be able to access the right TAP controller.

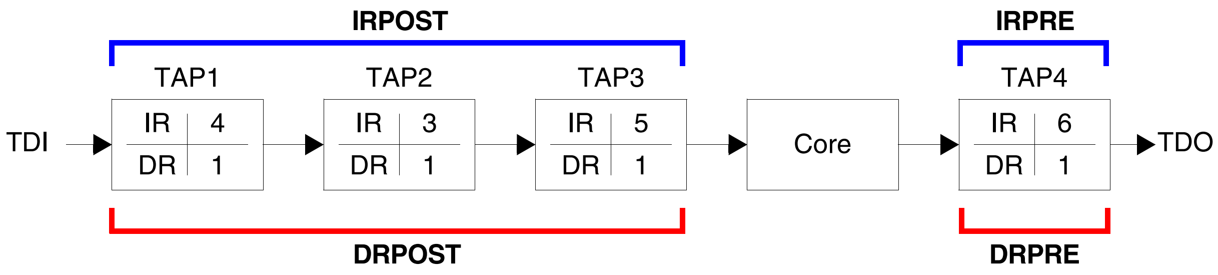
The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger of the TAP controller position in the JTAG chain if there is more than one core in the JTAG chain. The information is required before the debugger can be activated, e.g., by a **SYStem.Mode.Attach**.

TriState has to be used if several debuggers are connected to a common JTAG port at the same time. 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.

DRPRE	(default: 0) *<number>* of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.
DRPOST	(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.
IRPRE	(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.
IRPOST	(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest. See also **Daisy-Chain Example**.

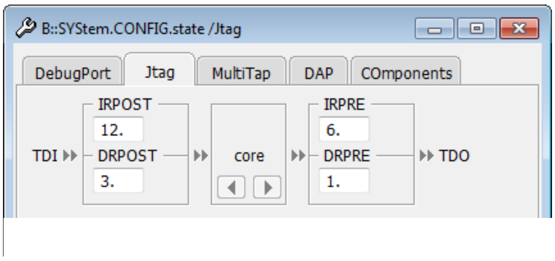
CoreNumber	*<number>* of cores in a shared-memory or local-memory multicore system. (default: 1)
TriState onoff	The debugger switches to tristate mode after each debug port access. If several debuggers share the same debug port, this option is required. Then other debuggers can access the port. (default: OFF)
Slave [**ON** | **OFF**]	Defines the master in a multicore chip. Only one core can be the master of the chip reset, the TAP reset and the chip initialization features. All other cores are slave cores. (default: OFF)
TAPState	This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable. (default: 7 = Select-DR-Scan)
TCKLevel [**0** | **1**]	Level of TCK signal when all debuggers are tristated. (default: 0)

Daisy-Chain Example



IR: Instruction register length **DR**: Data register length **Core**: The core you want to debug

Daisy chains can be configured using a PRACTICE script (*.cmm) or the **SYStem.CONFIG.state** window.



IRPOST 12.
DRPOST 3.
IRPRE 6.
DRPRE 1.

Example: This script explains how to obtain the individual IR and DR values for the above daisy chain.

```
SYStem.CONFIG.state /Jtag     ; optional: open the window

SYStem.CONFIG IRPRE   6.      ; IRPRE: There is only one TAP.
                              ; So type just the IR bits of TAP4, i.e. 6.

SYStem.CONFIG IRPOST 12.      ; IRPOST: Add up the IR bits of TAP1, TAP2
                              ; and TAP3, i.e. 4. + 3. + 5. = 12.

SYStem.CONFIG DRPRE   1.      ; DRPRE: There is only one TAP which is
                              ; in BYPASS mode.
                              ; So type just the DR of TAP4, i.e. 1.

SYStem.CONFIG DRPOST  3.      ; DRPOST: Add up one DR bit per TAP which
                              ; is in BYPASS mode, i.e. 1. + 1. + 1. = 3.
                              ; This completes the configuration.
```

NOTE:	In many cases, the number of TAPs equals the number of cores. But in many other cases, additional TAPs have to be taken into account; for example, the TAP of an FPGA or the TAP for boundary scan.

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

SYStem.CPU Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**UC3A0512** | **UC3A0256** | **UC3A0128** | **UC3A1512** | **UC3A1256** | **UC3A1128** | **UC3B0512** | **UC3B0256** | **UC3B0128** | **UC3B064** | **UC3B1512** | **UC3B1256** | **UC3B1128** | **UC3B164** | **UC3L064** | **UC3L032** | **UC3L016** | **UC3A3256S** | **UC3A3256** | **UC3A3128S** | **UC3A3128** | **UC3A364S** | **UC3A364** | **UC3A4256S** | **UC3A4256** | **UC3A4128S** | **UC3A4128** | **UC3A464S** | **UC3A464** | **UC3C064** | **UC3C0128** | **UC3C0256** | **UC3C0512** | **UC3C164** | **UC3C1128** | **UC3C1256** | **UC3C1512** | **UC3C264** | **UC3C2128** | **UC3C2256** | **UC3C2512** | **UC3D032** | **UC3D064** | **UC3D0128** | **UC3D132** | **UC3D164** | **UC3D1128** | **UC3D1256**

Default: UC3XXX.

Selects the processor type. All of the Atmel CPUs with AVR32 cores are supported.

SYStem.JtagClock Define JTAG clock

Format:	**SYStem.JtagClock** *<frequency>* **SYStem.BdmClock** *<frequency>* (deprecated)
<frequency>:	**4kHz …100 MHz** **1250000.** | **2500000.** | **5000000.** | **10000000.** (on obsolete ICD hardware)

Default frequency: 1 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 (i.e. 109 KHz will be converted to 125 KHz). 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.

SYStem.MemAccess — Select run-time memory access method

Format: **SYStem.MemAccess Enable | StopAndGo | Denied**

Default: Denied.

Nexus	Non-intrusive memory access during program execution is enabled. Only run-time **memory classes** can be accessed.
Enable CPU (deprecated)	This option is not available at the moment.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed.
Denied	Memory access during program execution to target is disabled.

SYStem.Mode — Establish the communication with the target

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
Attach
Go
Up

Default: Down.

Down	Disables the debugger. The state of the CPU remains unchanged.
NoDebug	The debug adapter gets tristated. The state of the CPU remains unchanged. Debug mode is not active. In this mode the target behaves as if the debugger is not connected.
Attach	Initializes the debug interface and connect to the core while program remains running. After this command, the user program can be stopped with the **Break** command or by any other break condition (e.g. breakpoints).
Go	Resets the target and starts execution.
Up	Resets the target and stops the CPU at the reset vector.
StandBy	Not available for AVR32.

SYStem.LOCK — Lock and tristate the debug port

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

Default: OFF.

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

SYStem.Option.IMASKASM — Disable interrupts while single stepping

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

Default: OFF.

If enabled, the interrupt enable flag of the EFLAGS register will be cleared during assembler single-step operations. After the single step, the interrupt enable flag is restored to the value it had before the step. It is turned on to make sure that no interrupt routine is serviced between **Break** and **Go** states.

SYStem.Option.IMASKHLL — Disable interrupts while HLL single stepping

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

Default: OFF.

If enabled, the interrupt enable flag of the EFLAGS register will be cleared during HLL single-step operations. After the single step, the interrupt enable flag is restored to the value it had before the step.

SYStem.Option.MPU — Disable MPU during memory access

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

Default: OFF.

The AVR32 architecture specifies an optional MPU unit which can restrict memory accesses. It's not possible to read memory when the MPU blocks it. If this option is turned on, the MPU is first turned off, then the memory is read, and the MPU is turned on again. This way, every memory address is accessible.

SYStem.Option.AUTO — Auto JTAG setting

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

Default: ON.

Calculates the maximum available JTAG frequency according to the PLL settings and sets up the JTAG frequency automatically. The calculation and setting of the JTAG frequency is done at every **Go** / **Break** command.

SYStem.EraseChip — Erases the Flash and the EEprom

Format:	**SYStem.EraseChip**

Erases the Flash memory. It is available to the user only in the **SYStem.Down** mode.

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.CONVert — Adjust range breakpoint in on-chip resource

Format: **TrOnchip.CONVert** [**ON** | **OFF**] (deprecated)
Use Break.CONFIG.InexactAddress instead

The on-chip breakpoints can only cover specific ranges. If a range cannot be programmed into the breakpoint, it will automatically be converted into a single address breakpoint when this option is active. This is the default. Otherwise an error message is generated.

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

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

TrOnchip.VarCONVert — Adjust complex breakpoint in on-chip resource

Format:	TrOnchip.VarCONVert [ON | OFF] (deprecated) **Use Break.CONFIG.VarConvert instead**

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

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

Format:	**TrOnchip.RESet**

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

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 (faster).
OFF	The program execution is stopped by sending a break sequence via NEXUS. (takes a bit longer)

Example: Sample all write accesses to the variable flags[3] to the trace and all cycles from func5.

```
; Set a program breakpoint to the entry of func5 and select the action
; TraceON
Break.Set func5 /Program /TraceON

; Set a program breakpoint to the exit of func5 and select the action
; TraceOFF
Break.Set Var.END(func5)-1 /Program /TraceOFF

; Set a write breakpoint to flags[3] and select the action TraceEnable
Var.Break.Set flags[3] /Write /TraceEnable
```

TrOnchip.EVTO — Output sync signals on EVT0

Format: **TrOnchip.EVTO** [**ON** | **BREAK** | **OFF**]

ON	Generates a signal if a Watch/Breakpoint has been passed.
BREAK	Generates a signal if the CPT enters Debug mode.
OFF	No sync signal is generated on EVTO pin. (Default)

The output signal EVTO is available at the Nexus probe connector pin **OX0**.

TrOnchip.EXTernal — Generate a trigger for the trace on high pulse on INx

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
```

NEXUS.BTM

Branch trace mode

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

Default: ON.

If turned on, nexus branch messages are generated each time a jump, return, branch, etc. command is executed. This option must always be turned on if the user wants to reconstruct the program flow.

NEXUS.DDR

Use the DDR transmission

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

Default: OFF.

The clock frequency of the nexus port is halved and the trace data are sent on both falling and rising edge of the trace clock.

NEXUS.DTM
Data trace mode

Format:	**NEXUS.DTM** *<mode>*
<mode>:	**ON** | **OFF** **Read** **Write** **ReadWrite**

Controls the NEXUS Data Trace Messages.

Default: Data Trace Messages are disabled (OFF).

OFF	No Data Trace Messages are generated.
Write	Data write accesses are output on the trace. Watchpoints and Trace Filters are used to control the data trace.
Read	Data read accesses are output on the trace. Watchpoints and Trace Filters are used to control the data trace.
ReadWrite	Data read and write accesses are output on the trace. Watchpoints and Trace Filters are used to control the data trace.

NEXUS.OFF
Switch the NEXUS trace port off

Format:	**NEXUS.OFF**

If the debugger is used stand-alone, the trace port is disabled by the debugger.

NEXUS.ON
Switch the NEXUS trace port on

Format:	**NEXUS.ON**

The NEXUS trace port is switched on. All trace registers are configured by debugger.

NEXUS.OTM Ownership trace messages

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

Default: OFF.

Ownership Trace Messages are created when the OTM is on.

NEXUS.PinConfig Override the nexus port pin mapping

Format: **NEXUS.PinConfig** [**0** | **1** | **2** | **3**]

Default: 0.

Overrides the default pin mapping of the nexus(aux) port. It allows the user to choose one of the 4 pin configurations as the trace data output.

NEXUS.PortMode Change the nexus port clock frequency

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

Default: 1/1.

Adjusts the nexus port clock frequency by dividing the CPU clock by 1, 2, 4 or 8.

NEXUS.Register Display NEXUS trace control registers

Format: **NEXUS.Register**

This command opens a window which shows the NEXUS configuration and status registers.

NEXUS.RESet — Reset NEXUS trace port settings

Format: **NEXUS.RESet**

Resets NEXUS trace port settings to default settings.

NEXUS.Spen<messagetype> — Avoid message overrun

Format: **NEXUS.SpenDTM** [**ON** | **OFF**]
NEXUS.SpenPTM [**ON** | **OFF**]

NEXUS.SpenDTM stalls the CPU to avoid the data trace message overrun if the FIFO is full.
Default: OFF.

NEXUS.SpenPTM stalls the CPU to avoid the program trace message overrun if the FIFO is full.
Default: ON.

NEXUS.SQA — Synchronize trace by using full address

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

Default: OFF.

Forces the CPU to generate trace messages with full address which allows the TRACE32 to quickly calculate the program counter. This option should be turned on if the user plans to connect/disconnect the Nexus Adapter during run-time. The side-effect of this option is that the trace FIFO on CPU is filled faster.

NEXUS.state — Display NEXUS port configuration window

Format: **NEXUS.state**

Displays the NEXUS trace configuration window.

NEXUS.WTM

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

Default: OFF.

Watch Trace Messages are output on the nexus port when the WTM is on.

Connectors

Debug Connector

Mechanical Description of the 10-pin Debug Cable

This connector is defined by Atmel, and we recommend this connector for all future designs.

Signal	Pin	Pin	Signal
TCK	1	2	GND
TDO	3	4	VCC
TMS	5	6	RST-
N/C	7	8	N/C
TDI	9	10	GND

NEXUS Connector

Mechanical Description of the MICTOR38 Debug Connector

This connector is defined by IEEE ISTO NEXUX5001, and we recommend this connector for all future designs.
We also recommend to leave the unused signals open.

Signal	Pin	Pin	Signal
MDO12	1	2	MDO13
MDO14	3	4	MDO15
MDO09	5	6	(CLKOUT)
N/C	7	8	MDO08
RSTIN-	9	10	EVTI-
TDO	11	12	VTREF
MDO10	13	14	RDY-
TCK	15	16	MDO07
TMS	17	18	MDO06
TDI	19	20	MDO05
JCOMP	21	22	MDO04
MDO11	23	24	MDO03
RESETOUT	25	26	MDO02
TDET/WDTDIS	27	28	MDO01
BGRNT	29	30	MDO00
N/C	31	32	EVTO-
N/C	33	34	MCKO
BREQ	35	36	MSEO1-
N/C	37	38	MSEO0-

Signal Load and Impedance

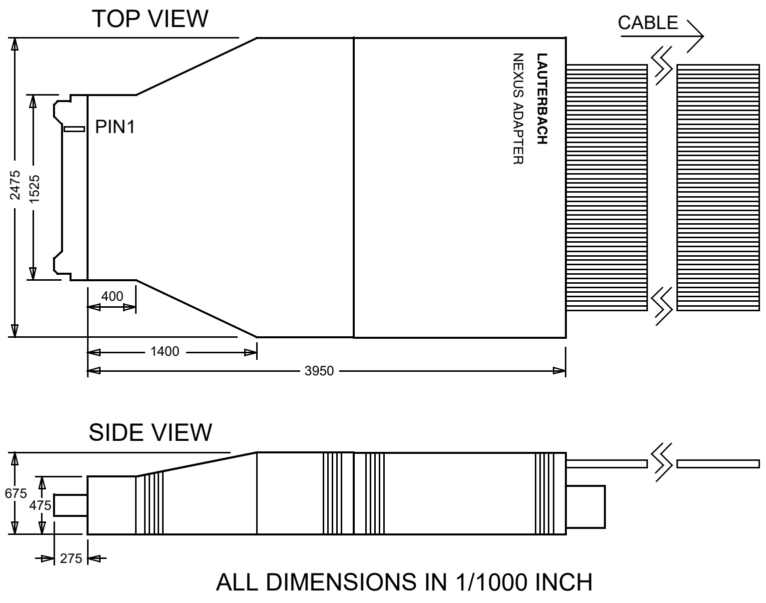
Important: The load values below are giving just an idea about the signals load and driver impedance (no commitment)

Signal	***Direction* (for the probe)**	***Resistance Capacity***	***Remark***
VREF (VTREF)	Input	50 KOhm PD-GND	
RSTIN	Input Output	10 pF 22 Ohm S, 10 KOhm VTAR	 OD
RSTOUT (RESETOUT)	Input	10 pF	*
CLKOUT	Input	100 Ohm VTT, 10 pF	*
TCK	Output	22 Ohm S	TS
RTCK	Input	100 Ohm VTT 10 pF	*
TRST (JCOMP)	Output	22 Ohm S	TS
DBACK (RDY)	Input	RC	*
DBREQ	Output	22 Ohm S	*,TS
TMS	Input Output	 22 Ohm S	cJTAG TS
TDI	Output	22 Ohm S	TS
TDO	Input	RC	cJTAG
MCKO	Input	100 Ohm VTT, 10 pF	
MSEO0	Input	100 Ohm VTT, 10 pF	
MSEO1	Input	100 Ohm VTT, 10 pF	
MDO00	Input	100 Ohm VTT, 10 pF	
MDO01	Input	100 Ohm VTT, 10 pF	
MDO02	Input	100 Ohm VTT, 10 pF	

MDO03	Input	100 Ohm VTT, 10 pF	
MDO04	Input	100 Ohm VTT, 10 pF	
MDO05	Input	100 Ohm VTT, 10 pF	
MDO06	Input	100 Ohm VTT, 10 pF	*
MDO07	Input	100 Ohm VTT, 10 pF	*
MDO08	Input	100 Ohm VTT, 10 pF	*
MDO09	Input	100 Ohm VTT, 10 pF	*
MDO10	Input	100 Ohm VTT, 10 pF	*
MDO11	Input	100 Ohm VTT, 10 pF	*
MDO12	Input	100 Ohm VTT, 10 pF	*
MDO13	Input	100 Ohm VTT, 10 pF	*
MDO14	Input	100 Ohm VTT, 10 pF	*
MDO15	Input	100 Ohm VTT, 10 pF	*
EVTI	Output	22 Ohm S	TS
EVTO	Input	100 Ohm VTT, 10 pF	
WDTE	Input Output	10 pF 22 Ohm S	 *,TS
ARBREQ	Output	22 Ohm S	*,TS
ARBGRANT	Input	10 pF	*

*	Not relevant vor AVR32, leave open.
VTT:	Connected to termination voltage. Can be 0.25 .. 3 V or disabled (tristate)
VTAR:	Connected to a voltage equivalent to the voltage at VTREF pin (target voltage).
S:	Serial resistor
OD:	Open drain output.
TS:	Can be tristated.
RC:	100 Ohm - 100 pF in series, one end to signal, other end to GND.

Mechanical Dimension


TOP VIEW
CABLE
LAUTERBACH
NEXUS ADAPTER
PIN1
2475
1525
400
1400
3950
SIDE VIEW
675
475
275
ALL DIMENSIONS IN 1/1000 INCH