LAUTERBACH
DEVELOPMENT TOOLS

AVR8 Debugger

Release 09.2023

MANUAL

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ...... 🗁

ICD In-Circuit Debugger ...... 🗁

Processor Architecture Manuals ...... 🗁

AVR8 ...... 🗁

AVR8 Debugger 1

History 4

Warning 5

Introduction 6

Brief Overview of Documents for New Users 6

Demo and Start-up Scripts 7

Configuration 8

System Overview 8

Quick Start 9

Troubleshooting 11

FAQ 12

AVR Specific Implementations 13

Breakpoints 13

Software Breakpoints 13

On-chip Breakpoints for Instructions 13

On-chip Breakpoints for Data 13

Overwriting Fuse and Lock Bits 13

Memory Classes 14

Programming the On-chip FLASH of the megaAVR 15

Special Hints, Restrictions, and Known Problems 15

Restrictions 15

Known Problems 15

CPU specific SYStem Settings 16

SYStem.CONFIG.state Display target configuration 16

SYStem.CONFIG Configure debugger according to target topology 17

SYStem.CONFIG.DEBUGPORTTYPE Select debug port type 20

SYStem.CPU Select the used CPU 21

SYStem.EraseChip Erases the Flash and the EEprom 21
SYStem.JtagClock Define JTAG clock 21
SYStem.LOCK Lock and tristate the debug port 22
SYStem.MemAccess Real-time memory access (non-intrusive) 22
SYStem.Mode Establish the communication with the target 23
SYStem.Option.IMASKASM Disable interrupts while single stepping 23
SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping 24

CPU specific TrOnchip Commands 25

TrOnchip.state Display on-chip trigger window 25
TrOnchip.CONVert Adjust range breakpoint in on-chip resource 25
TrOnchip.VarCONVert Adjust complex breakpoint in on-chip resource 26
TrOnchip.RESet Set on-chip trigger to default state 26

Connectors 27

Debug Connector 27
Mechanical Description of the 10-pin Debug Cable 27
Converter 10-pin JTAG to 6-pin SPI for AVR8 27
Converter 10-pin JTAG to 6-pin UPDI for AVR8 28
Converter 10-pin JTAG to 8-pin UPDI for AVR8 28

AVR8 Debugger

Version 10-Oct-2023

History

05-May-22	New command SYStem.EraseChip.
15-Jun-16	Initial version.

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 AVR8 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.

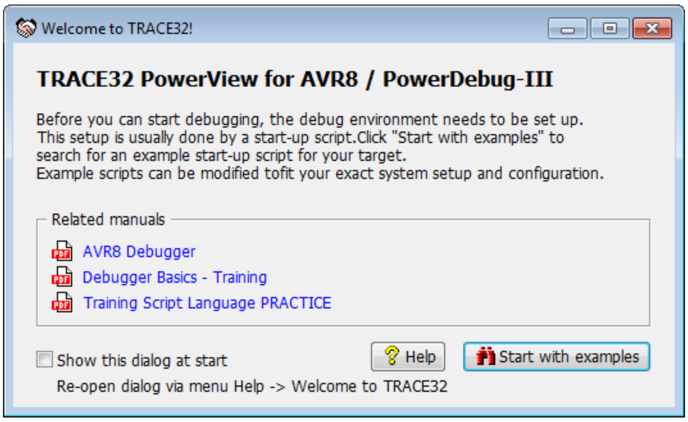
Brief Overview of Documents for New Users

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**):


Welcome to TRACE32!

TRACE32 PowerView for AVR8 / PowerDebug-III

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.

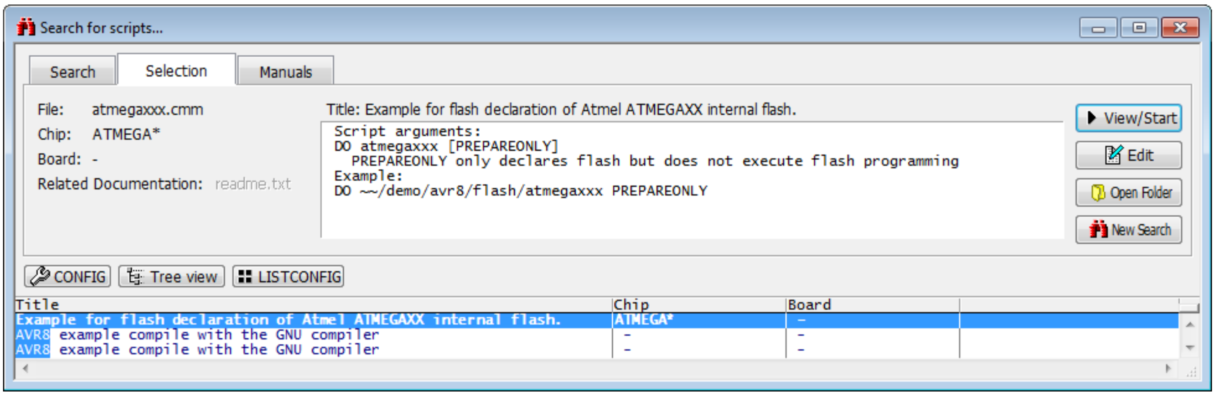
Related manuals
AVR8 Debugger
Debugger Basics - Training
Training Script Language PRACTICE
Show this dialog at start
Help
Start with examples

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: atmegaxxx.cmm
Chip: ATMEGA*
Board: -
Related Documentation: readme.txt

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

```
Script arguments:
DO atmegaxxx [PREPAREONLY]
  PREPAREONLY only declares flash but does not execute flash programming
Example:
DO ~~/demo/avr8/flash/atmegaxxx PREPAREONLY
```

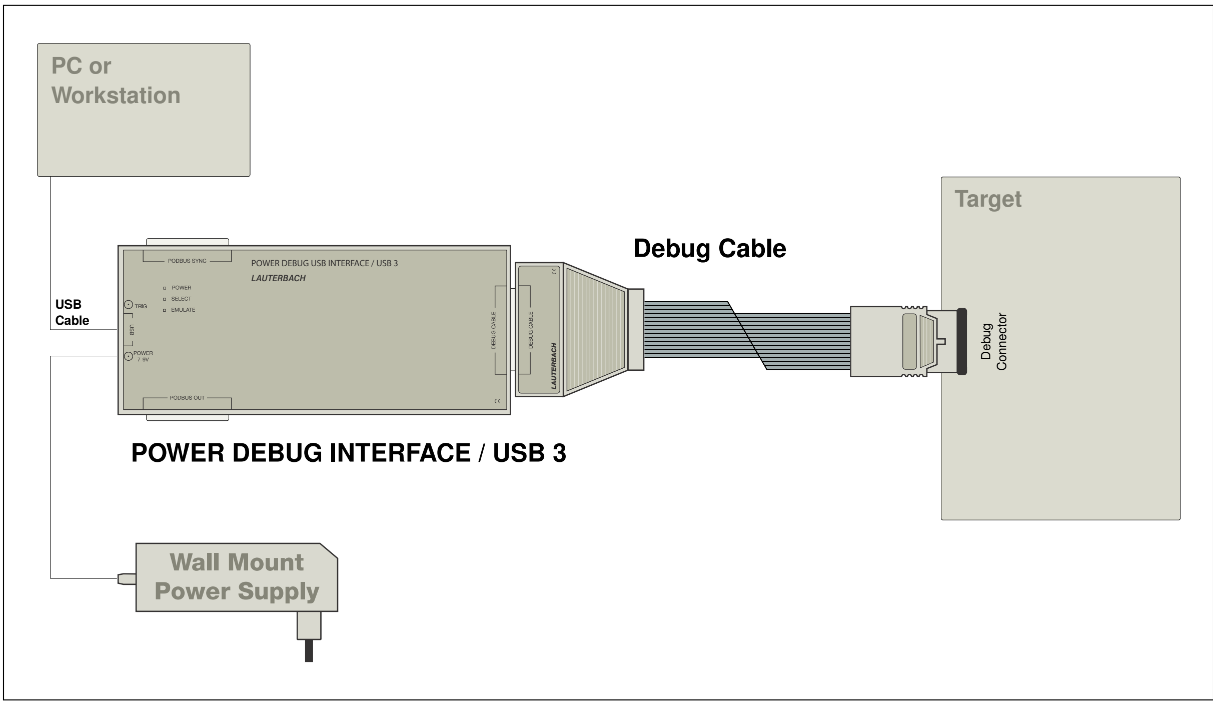
Title	Chip	Board
Example for flash declaration of Atmel ATMEGAXX internal flash.	ATMEGA*	-
AVR8 example compile with the GNU compiler	-	-
AVR8 example compile with the GNU compiler	-	-

You can also manually navigate in the `~~/demo/avr8/` subfolder of the system directory of TRACE32.

Configuration

System Overview

Example configuration for an AVR8 debugger.



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. Set your connection type.

```
SYStem.CONFIG.DEBUGPORTTYPE JTAG|SPI|UPDI
```

This command selects one of the two possible connections: JTAG or SPI.

3. Specify the CPU specific settings.

```
SYStem.CPU ATMEGA1280
```

This command selects the CPU type.

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

4. 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.

5. 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.

6. Load the program into the flash..

```
DO ~~/demo/avr8/flash/atmegaxxx.cmm
```

A typical start sequence of the AVR8 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

SYStem.CONFIG.DEBUGPORTTYPE        ; Select the connection type JTAG or SPI
JTAG                               or UPDI

MAP.UpdateOnce p:0x80000--         ; Specify the address range for caching
0xfffff

WinCLEAR                           ; Clear all windows

SYStem.Up                          ; Reset the target and enter debug mode

DO                                 ; Load the target application
~~/demo/avr8/flash/atmegaxxx.
cmm

                                   ; 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.

FAQ

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

AVR Specific Implementations

Breakpoints

Software Breakpoints

The Microchip megaAVR architecture does not support the software breakpoints.

On-chip Breakpoints for Instructions

The megaAVR MCUs support a total of four on-chip breakpoint registers which can be used as program breakpoints to stop and debug the program which executes always in the Flash.

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. The two instruction breakpoints of megaAVR MCUs can be used as data breakpoints

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.

Overwriting Fuse and Lock Bits

There are two options when overwriting the Fuse or Lock Bits. The preferred way is to use the periphery window (i.e. **PER** command), where you can manually check and overwrite single bits using the standard right-click and select method.

Optionally, you can modify the Fuse and Lock Bits with the **PER.Set.simple** command as given below:

```
SYStem.RESet
SYStem.Up

;Setting the LockBits
PER.Set.simple EE:0x200000 %Byte <8_bit_lock_bit_setting>

;Setting the FuseBits
PER.Set.simple EE:0x100000 %Long <24_bit_fuse_bit_setting>

SYStem.Down
```

Memory Classes

The following memory access classes are available:

Access Class	Description
D	Data
P	Program
EE	EEprom

To access a memory class, write the class in front of the address. For example, use D to access the data memory:

```
Data.dump D:0x00
```

The memory class EE is used to denote the EEprom memory.

```
Data.dump EE:0x00
```

The following examples return different results, since the megaAVR architecture uses the Harvard Architecture.

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

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

Programming the On-chip FLASH of the megaAVR

The PRACTICE script for programming of the on-chip FLASH of the megaAVR can be found in the TRACE32 demo folder ~~/demo/avr8/flash/atmegaxxx.cmm.

Please be aware that this is just an example script. The scripts has to be adapted to your memory layout, specifically the flash and EEprom sector size must be checked.

Special Hints, Restrictions, and Known Problems

Restrictions

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

Known Problems

- **JTAG**: Help system not available yet.

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

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.

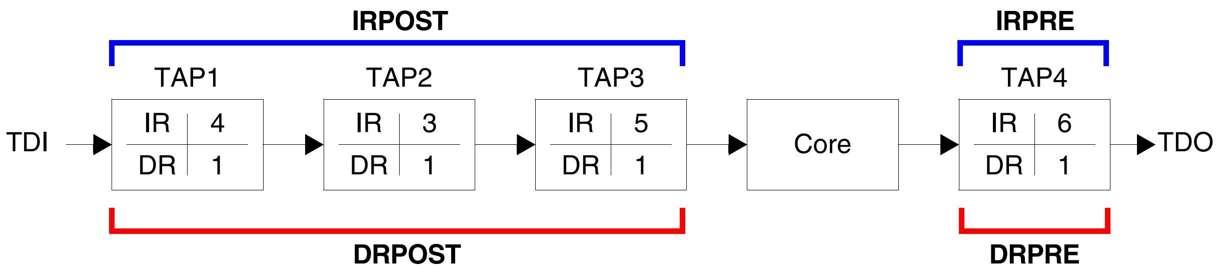
The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger of the TAP controller position in the JTAG chain if there is more than one core in the JTAG chain. The information is required before the debugger can be activated, e.g., by a **SYStem.Up**.

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**.

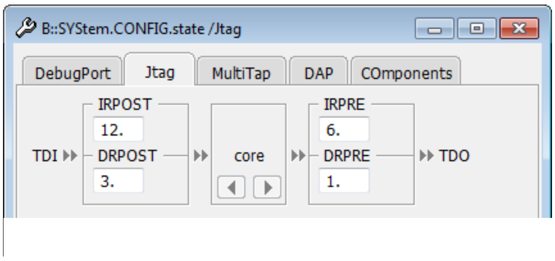
CoreNumber	*<number>* of cores in a shared-memory or local-memory multicore system. (default: 1)
TriState [**ON** | **OFF**]	The debugger switches to tristate mode after each debug port access. If several debuggers share the same debug port, this option is required. Then other debuggers can access the 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.CONFIG.DEBUGPORTTYPE   Select debug port type

Format:	**SYStem.CONFIG.DEBUGPORTTYPE** [<*type*>]
<*type*>:	**JTAG** | **SPI** | **UPDI**

Specifies the used debug port type “JTAG” or “SPI”. It assumes the selected type is supported by the target.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**ATMEGA1284** | **ATMEGA64** | **ATMEGA32** | …

Default: UC3XXX.

Selects the processor type. All of the ATMEGA MCU cores with JTAG port are supported.

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.

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.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.MemAccess Real-time memory access (non-intrusive)

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

Default: Denied.

Enable **CPU** (deprecated)	This option is not available at the moment.
Denied	Real-time memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

SYStem.Mode Establish the communication with the target

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

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

<mode>: **Down**
NoDebug
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.
Go	Resets the target and starts execution.
Up	Resets the target and stops the CPU at the reset vector.
Attach **StandBy**	Not available for AVR8.

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.

CPU specific TrOnchip Commands

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.

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

Converter 10-pin JTAG to 6-pin SPI for AVR8

The converter supports the Serial Peripheral Interface (SPI) as used by Microchip AVR128 devices.

Signal	Pin	Pin	Signal
TDO	1	2	VTREF
TCK	3	4	TDI
RESET-	5	6	GND

Order-Code	Description
LA-2732	Converter 10-pin JTAG to 6-pin SPI for AVR8

Converter 10-pin JTAG to 6-pin UPDI for AVR8

The converter supports the Unified Program and Debug Interface (UPDI) as used by Microchip AVR128 devices.

Pin	Signal
1	UPDI
2	VCC
3	N/C
4	N/C
5	N/C
6	GND

Order-Code	Description
LA-2733	Converter 10-pin JTAG to 6-pin UPDI for AVR8

Converter 10-pin JTAG to 8-pin UPDI for AVR8

The converter supports the Unified Program and Debug Interface (UPDI) as used by Microchip AVR128 devices.

Pin	Signal
1	N/C
2	VCC
3	GND
4	UPDI
5	N/C
6	N/C
7	N/C
8	N/C

Order-Code	Description
LA-2736	Converter 10-pin JTAG to 8-pin UPDI for AVR8