LAUTERBACH
DEVELOPMENT TOOLS

MSP430 Debugger

Release 02.2025

MANUAL

MSP430 Debugger

MSP430 Debugger

Version 13-Feb-2025

History

19-Dec-2023 SYStem.Mode Attach and SYStem.Mode NoDebug are no longer available.

Introduction

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

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Tutorial"** (debugger_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"General Commands"** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.
- **"OS Awareness Manuals"** (rtos_<*os*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your Debug Cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.

Demo and Start-up Scripts

Lauterbach provides ready-to-run start-up scripts for known MSP430 based hardware.

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

  You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.

You can also manually navigate in the ~~/demo/msp430/ subfolder of the system directory of TRACE32.

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.

General Notes/Target Design Requirements/Recommendations

Before starting please be sure to have up to date debugger software by getting an update from the LAUTERBACH website. Note that the downloads on the website are stable releases but not necessarily the latest versions. Therefore in case of problems please contact LAUTERBACH support at bdmmsp430-support@lauterbach.com

General

- The Lauterbach TRACE32 debugger for MSP430 is an on-chip debugging tool (OCD). It uses the debug function implemented in the target CPU.
- Available debug interfaces are the 4-wire JTAG interface or the Spy-Bi-Wire interface.
- The debugging support does also include CC430 devices.

Target Design Requirements

- Locate the debug connector as close as possible to the processor to minimize the capacitive influence and cross coupling of noise onto the signals.
- Reduce the cable length between CPU and Lauterbach connector to a minimum. Best results will be provided, if a adequate connector will be foreseen directly on the target board.
- The TEST pin of the MSP430 must be connected to the debugger if available. See **"Debug Connection"**, page 29 on **page 29** for connection information.

Limitations

- The debugger offers no target power supply.
- Locking the JTAG interface is currently not supported.
- Multicore debugging is currently not supported. Please contact technical support if you intend to debug multicore setups.

Contacting Support

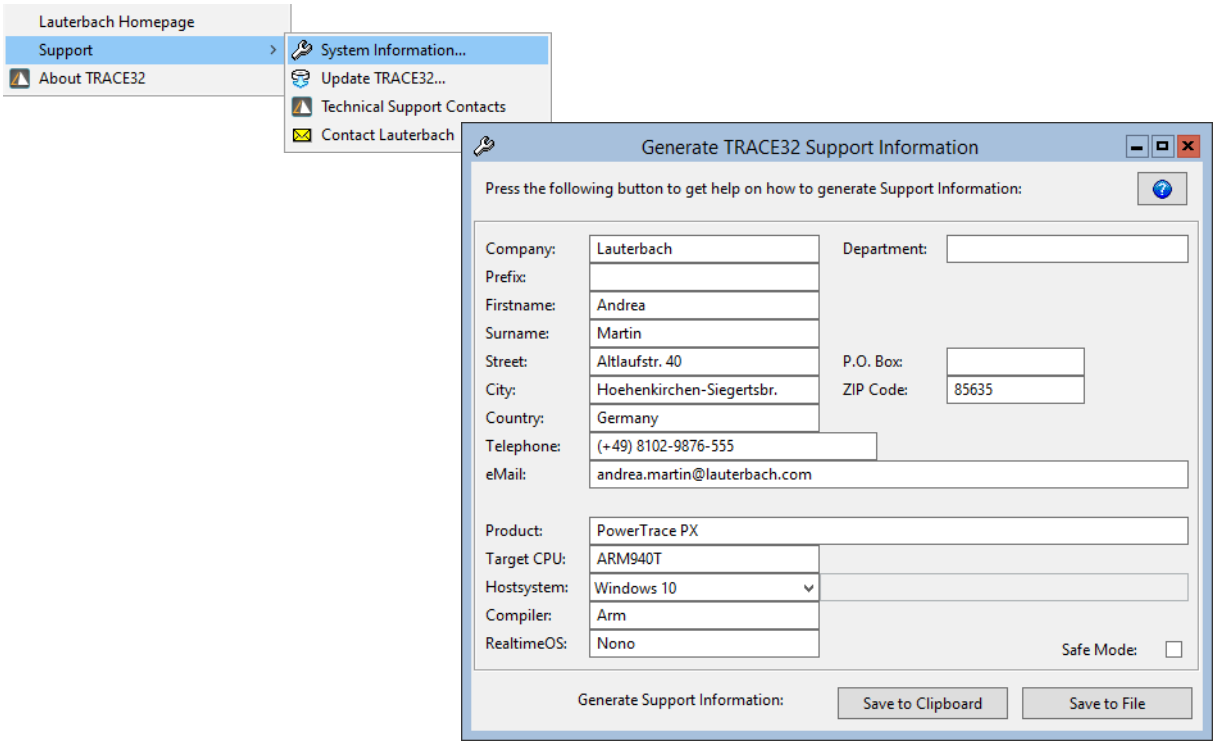
Use the Lauterbach Support Center: https://support.lauterbach.com

- To contact your local TRACE32 support team directly.
- To register and submit a support ticket to the TRACE32 global center.
- To log in and manage your support tickets.
- To benefit from the TRACE32 knowledgebase (FAQs, technical articles, tutorial videos) and our tips & tricks around debugging.

Or send an email in the traditional way to support@lauterbach.com.

Be sure to include detailed system information about your TRACE32 configuration.

1. To generate a system information report, choose **TRACE32** > **Help** > **Support** > **Systeminfo**.



NOTE:	Please help to speed up processing of your support request. By filling out the system information form completely and with correct data, you minimize the number of additional questions and clarification request e-mails we need to resolve your problem.

2. Preferred: click **Save to File**, and send the system information as an attachment to your e-mail.
3. Click **Save to Clipboard**, and then paste the system information into your e-mail.

Starting up the debugger is done by the following steps:

1. Select the device prompt B: for the TRACE32 ICD-Debugger, if the device prompt is not active after starting the TRACE32 software.

```
B::
```

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. A **RESet** command is useful if you do not start directly after booting the TRACE32 development tool.

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

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

The default value for SYStem.CPU is “MSP430”, which is a derivative that does not exist. You should always select an appropriate device. Otherwise the debug connection to the target might fail. The default values of all other SYStem 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. Enter debug mode.

```
SYStem.Mode Up
```

4. Declare size and type of **FLASH** memory is recommended doing via script.

```
DO ~~/demo/msp430/flash/msp430f*.cmm
```

Select the adequate PRACTICE script for the connected target. It will set up the flash memory to allow writing and setting of software-breakpoints. A number of demo *.cmm scripts for flash programming is included in your MSP430 installation demo directory.

You can load a program into flash (if this was not already done by the demo script) as follows:

5. Load the program.

```
Data.LOAD.auto ~~/demo/msp430/hardware/msp_exp430f5438/sieve.d43
```

This example loads a sieve demo for the MSP430F5438 evaluation board. Data.Load.AUTO detects automatically the correct format. The option auto is not mandatory an could be left aside. Loading an application to flash memory is only possible if the flash is declared correctly and unlocked. Please refer to the flash demo scripts if you need an example for this.

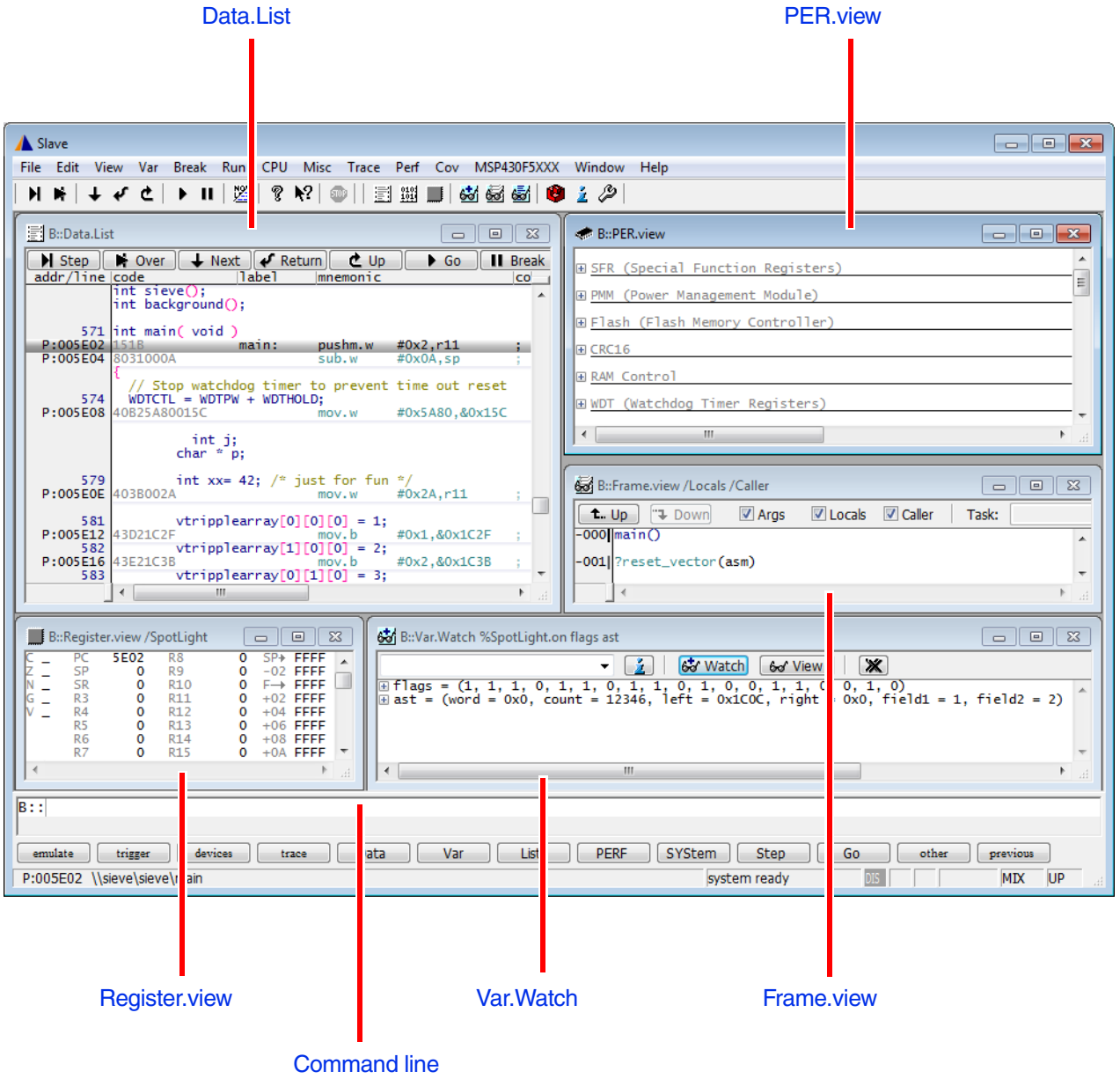
A detailed description of the **Data.LOAD** command and all available options is given in the **“General Reference Guide”**.

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

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

RESet                               ; Reset TRACE32 Software (not target!)

WinCLEAR                            ; Clear all windows

SYStem.CPU MSP430F5438              ; Select the CPU derivative type

SYStem.Up                           ; Reset the target and enter debug mode

DO                                  ; Start flash programming and load the
~~/demo/msp430/flash/msp430f5       ; application
xx.cmm

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

Register.view /SpotLight            ; Open register window                *)

List.Mix                            ; Open source code window             *)

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

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

PER.view                            ; Open a window for the special
                                    ; function registers and peripherals *)
                                    ; Appropriate cpu must be selected!
```

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

Having executed this script your debug session might look like this:



Communication between Debugger and Processor can not be established

Typically the **SYStem.Up** command is the first command of a debug session where communication with the 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** window to view all error messages.

- The target has no power or the debug cable is not connected to the target. This results in the error message “target power fail”.
- The target is in an unrecoverable state. Re-power your target and try again.
- The default debug clock speed is too fast, especially if the target is connected to the debugger by a long cable. Reduce the communication speed with **SYStem.JtagClock** command and optimize the speed when you got it working.
- The CPU has no clock.
- The CPU is kept in reset.
- Although the debugger takes care of the watchdog you should check if there is a watchdog which needs to be deactivated.
- The target is in low power mode and the JTAG interface is not available.
- The target is protected via JTAG fuse / password. In this case it is not possible to establish a JTAG connection.
- In case you’ve selected Spy-Bi-Wire (SBW) to connect to the target, check if there are capacities connected to the reset line. In SBW mode the MSP430 shares the data line with the reset line. Any capacities/loads might act as low-pass that reduces your possible clock speed.
- The core is in LPMx.5 (Low power mode with JTAG turned off). The core could not be recovered via JTAG. Try to connect to your core using the Spy-Bi-Wire (SBW) interface.

FAQ

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

Breakpoints

Two types of breakpoints are available for MSP430 architecture: Software breakpoints (SW-BP) and on-chip breakpoints (HW-BP).

Software Breakpoints

Software breakpoints are the default breakpoints. To set a software breakpoint, before resuming the CPU, the debugger replaces the instruction at the breakpoint address with a breakpoint code instruction. SW-BPs can be used in RAM areas and in FLASH areas if **FLASH.AUTO** is set properly.

There is no restriction in the number of software breakpoints. But it must be considered that by setting software breakpoints in flash the flash memory will be changed. Consequently the use of software breakpoints in flash will reduce the number of program/erase cycles that are left for the flash.

Please note that software breakpoints consume one on-chip breakpoint resource if one or more software breakpoints are set. This must be taken into account when combining on-chip and software breakpoints.

On-chip Breakpoints

If on-chip breakpoints are set the debugger will configure the integrated debug hardware of the MSP430 for this purpose. Current available MSP430 devices allow to set 2 to 8 on-chip breakpoints (device dependent). Breakpoints that are set on code in read only memory must be on-chip breakpoints. With the command **MAP.BOnchip** *<range>* it is possible to declare memory address ranges for use with on-chip breakpoints to the debugger. The number of on-chip breakpoints to be set depends on the complexity of the desired breakpoint. Complex breakpoints consume more hardware resources than simple ones. On-chip breakpoints take effect **before execution**.

The Lauterbach MSP430 debugger allows to use a powerful variety of breakpoints. Available breakpoint types are also device dependent. It is possible to use the following breakpoints:

- **Program/Instruction Breakpoints:** Break on instruction fetch, either on single address or address range
- **Data Breakpoints:** Break on data read and/or write on a single address or address range
- **Memory Breakpoints:** Break when a certain program part does a read/write access to a certain address range.
- **Register Breakpoint:** Break when a certain register is written (if available)
- All range breakpoints might be used with the exclude option

You can check your breakpoint setup with the command **Break.List.**

If no more on-chip breakpoints are available you will get an error message on trying to set a new on-chip breakpoint. Delete other breakpoints to regain debug resources.

Breakpoints on Data Addresses and Data Values

Breakpoints on data addresses are bound to several conditions:

1. The entity doing the data access (read and/or write) must be the CPU. Any other accesses from on-chip or off-chip peripherals (DMA etc.) will not be recognized by the data address breakpoints. If you would like to trigger on other sources, like the DMA, contact technical support.
2. The data being targeted must be qualified by an address in memory. It is not possible to target registers (GPRs), peripherals etc.
3. Per default the break will be done independently of the value (empty DATA field of **Break.Set** window).

Breakpoints on Registers

Only a break on a register write action is supported by the MSP430 hardware. Register breakpoints are not available for all MSP430. Limitations:

1. Only write actions on registers can be triggered. Other breakpoint options for this type like read or read/write would cause an error.
2. When the pc is changed after an instruction execution this will not trigger a break action.
3. Changes to SR (status register) might not always trigger a break action.
4. A register breakpoint is automatically combined with a range breakpoint since most variables mapped to registers are only valid in a certain address range.

Breakpoints on Interrupts

MSP430 devices do not offer a special mechanism to halt the device on an interrupt event. However, you can set a data read/write breakpoint on the interrupt vector table. Once an interrupt is triggered the cpu will fetch the address of the interrupt handler from the interrupt vector table. This operation will trigger the read/write breakpoint and therefore halt the cpu on an interrupt event.

Please refer to your MSP430 specific device documentation for more information on MSP430 interrupts.

Example for Standard Breakpoints

Assume you have a target (MSP430FG4618) with:

- Code flash memory from `0x3100--0x19fff`
- RAM from `0x1100--0x30ff`

The following standard breakpoint combinations are possible without activated auto flash mode:

1. Unlimited breakpoints in RAM and up to eight breakpoints in ROM/FLASH

```
Break.Set 0x11f0 /Program             ; Software breakpoint 1 (RAM)

Break.Set 0x1220 /Program             ; Software breakpoint 2 (RAM)

Break.Set ram_addr /Program           ; Software breakpoint 3 (RAM)

Break.Set 0x4100 /Program             ; On-chip breakpoint (flash)
```

2. Unlimited breakpoints in RAM and up to eight breakpoints (BP) on a read or write access and up to four breakpoints on a read and write access (On single address each). Up to two read/write range breakpoints.

```
Break.Set 0x11f0 /Program             ; Software breakpoint 1 (RAM)

Break.Set 0x1332 /Write               ; On-chip breakpoint  (RAM)

Break.Set 0x1332--0x135E /Write       ; On-chip range BP (RAM)

Break.Set 0x1334 /ReadWrite           ; On-chip breakpoint (RAM)

Break.Set 0x1334++0x1C /ReadWrite     ; On-chip range BP (RAM)
```

With activated auto flash mode even in code flash memory unlimited breakpoints (BP) are allowed. Like in RAM complex breakpoints will still need an on-chip breakpoint.

3. Unlimited breakpoints in ROM/FLASH

```
FLASH.AUTO 0x4000--0x43FF              Allow software breakpoints in
                                       specific Flash area

Break.Set 0x4100 /Program              ; Software BP 1 (flash)

Break.Set 0x5320 /Program              ; Software BP 2 (flash)

Break.Set flash_addr /Program          ; Software BP 3 (flash)

Break.Set 0x4200--0x423f /Program      ; On-chip BP (flash)
```

4. Breakpoints on registers: It is assumed that variable “i” is mapped to a register. The variable shall be part of the Lauterbach sieve demo. “sieve\7” is the address where the variable is accessed within the function.

```
Var.Break.Set sieve\7 /VarWrite        ; register breakpoint on
\main\sieve\i                          variable "i" in sieve demo
```

Cycle Counter

The Cycle Counter is used to evaluate the number of cycles certain actions take. This could be for example measuring the number of instruction fetches that occurred.

There is currently no support implemented. Request a software update.

Runtime Measurement

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

The measured value depends on the set **JtagClock** for the debugger polls the cpu. A higher clock means faster communication with the target an thus a more accurate measurement.

Memory Classes

Though the MSP430 has a linear memory space, the following specific memory classes are available:

Memory Class	Description
P	Program Memory
D	Data Memory
VM	Virtual Memory (memory on the debug system)
E	Emulation Memory, Pseudo Dual port Access to Memory.

To access a memory class, write the class in front of the address. Prepending an E as attribute to the memory class will make memory accesses possible, even when the target CPU is running. Such an access must be allowed: See **SYStem.MemAccess** for more information.

Examples:

```
Data.dump E:0x200       ; View data while CPU is running

Data.dump D:0x200       ; View data memory

Data.dump VM:0x200      ; Virtual memory, no target access
```

NOTE:	Since the address space of the MSP430 is linear and non-overlapping memory class D and P are not distinguished memory classes and can be left aside. The MSP430 does not allow to read/write memory during run time. This means that a memory access using the access class E: is intrusive, i.e. the CPU is continuously stopped and restarted.

State Storage

The State Storage of the MSP430 allows to records the last eight address bus, data bus and status register values, dependant on the trigger that triggers the state storage.
There is currently no support implemented. Request a software update.

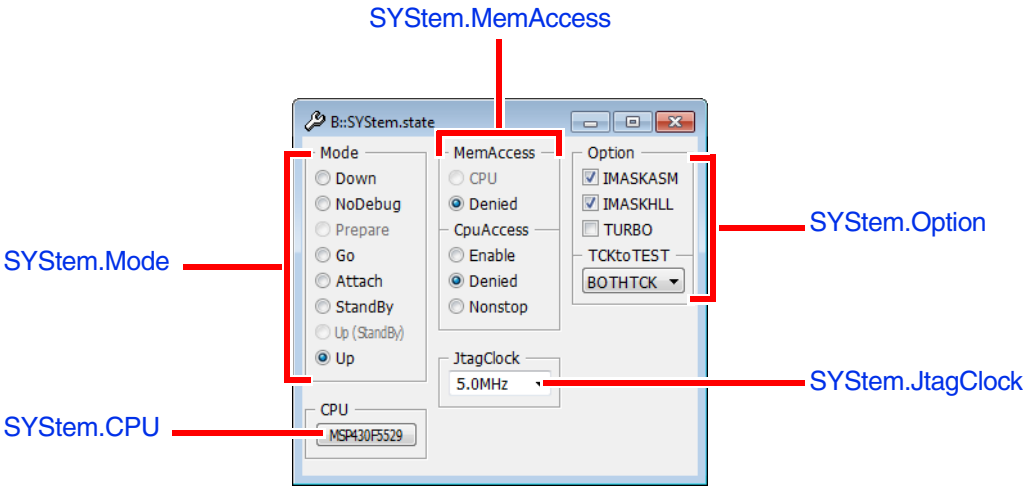
Trigger Sequencer

The trigger sequencer can be used to trigger an action under the condition that a certain programmed sequence happened.
There is currently no support implemented. Request a software update.

SYStem.state — Display SYStem.state window

Format:	**SYStem.state**

Opens the **SYStem.state** window, where you can configure the MSP430 debugger:



Format: **SYStem.CONFIG** *<parameter>*

<parameter>:
state
DEBUGPORTTYPE
IRPRE *<bits>*
IRPOST*<bits>*
DRPRE *<bits>*
DRPOST *<bits>*
TriState [**ON** | **OFF**]
Slave [**ON** | **OFF**]
TAPState *<state>*
TCKLevel *<level>*

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 debugger 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. TCK can have a pull-up or pull-down resistor, other trigger inputs need to be kept in inactive state.

DEBUGPORTTYPE [**JTAG** | **SPY-BI-WIRE**]
Select JTAG or Spy-Bi-Wire (SBW) interface. Only available for devices with SBW.

state
Not available yet. Contact technical support.

IRPRE
Not available yet. Contact technical support.

IRPOST
Not available yet. Contact technical support.

DRPRE
Not available yet. Contact technical support.

DRPOST
Not available yet. Contact technical support.

TriState [**ON | OFF**]
Not available yet. Contact technical support.

Slave [**ON | OFF**]
Not available yet. Contact technical support.

TAPState
Not available yet. Contact technical support.

TCKLevel [**0 | 1**]
Not available yet. Contact technical support.

Format: **SYStem.CPU** *<cpu>*

<cpu>: **MSP430xxx** | **CC430xxx**

Select the processor type. (Go to figure.)

SYStem.JtagClock Set jtag clock frequency

Format: **SYStem.JtagClock** *<frequency>*
SYStem.BdmClock (deprecated)

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 useful 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.

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

A decimal number like “100000.” short forms like “100kHz” or “15MHz” can also be used for *<frequency>*. 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.

Format: **SYStem.MemAccess** *<mode>*
SYStem.ACCESS (deprecated)

<mode>: **Enable**
Denied
StopAndGo

Default: Denied. (Go to figure.)

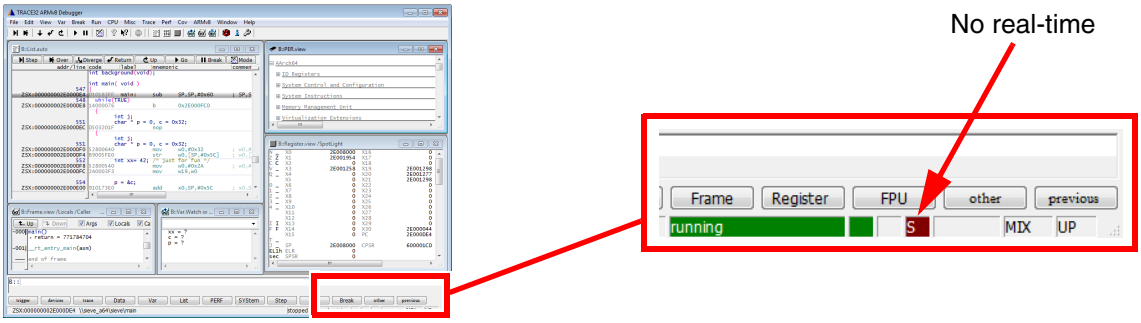
If SYStem.MemAccess is not Denied, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program. This is only possible for the instruction set simulator.

Enable
CPU (deprecated)
Used to activate the memory access while the CPU is running on the TRACE32 Instruction Set Simulator and on debuggers which do not have a fixed name for the memory access method.

Denied
No memory access is possible while the CPU is executing the program.

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.

If **SYStem.MemAccess StopAndGo** is set, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program. To make this possible, the program execution is shortly stopped by the debugger. Each stop takes some time depending on the speed of the JTAG port and the operations that should be performed. A white S against a red background in the TRACE32 state line warns you that the program is no longer running in real-time:



To update specific windows that display memory or variables 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** | **Go** | **Up**

Select target reset mode. (Go to figure.)

Down	Disables the debugger (default). The state of the CPU remains unchanged. The JTAG/SBW port is tristated.
NoDebug	Not available for MSP430.
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.
Up	Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging. All registers are set to the default level.
Attach	Not available for MSP430.
StandBy	Not available for MSP430.

System options allow to influence the behavior of the debugger.

SYStem.Option.IMASKASM Disable interrupts for assembler single steps

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

<option>: **ON** | **OFF**

Default: OFF

Disable interrupts while single stepping in assembler mode.

SYStem.Option.IMASKHLL Disable interrupts for HLL single steps

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

<option>: **ON** | **OFF**

Default: OFF

Disables interrupts while single stepping in HLL mode. An HLL step might execute several lines of code. Thus a target application might re-enable interrupts again during the step.

SYStem.Option.LPMX5 Enable LPMx5 support

Format: **SYStem.Option.LPMX5** *<option>*

<option>: **ON** | **OFF**

Default: OFF

Per default the support of the LPM5 mode is disabled. If enabled, the debugger will check the LPMx.5 state of a device. This takes additional time and will decrease the debug performance.

Only available for MSP430F5xxx, MSP430F6xxx, MSP30FR5xxx, CC4305xxx, CC430F5xxx devices that support the LPMx.5 mode.

SYStem.Option.TURBO — Speed up memory access

Format:	**SYStem.Option.TURBO** *<option>*
<option>:	**ON** | **OFF**

Default: OFF

If activated, additional error checks are avoided. This increases the read and write access to memory. Write or read errors might not be detected.

Only available for MSP430F5xxx, MSP430F6xxx, MSP30FR5xxx, CC4305xxx, CC430F5xxx devices.

SYStem.Option.TCKTOTEST — Configure clock output pins

Format:	**SYStem.Option.TCKTOTEST** *<option>*
<option>:	**NORMAL** | **BOTHTCK** | **BOTHTEST** | **SWAP**

Select output pin for debug clock. This option applies only for a Spy-Bi-Wire connection. It makes no sense to use this with 4-wire JTAG.

NORMAL	Clock is only output on TCK pin
BOTHTCK	Clock is output on TCK and TEST pin
BOTHTEST	Test is output on TCK and TEST pin
SWAP	Clock is only output on TEST pin

The **TrOnchip** command provides low-level access to the on-chip debug register.

TrOnchip.CONVert — Extend the breakpoint range

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

Default: ON.

The debug unit of some devices does not provide the resources to set an on-chip breakpoint to an address range. Instead only bit masks can be used to mark a memory range with a breakpoint. A mask has a reduced flexibility and cannot handle all ranges. It is therefore required to adapt the address range the user has entered so that it fits to the debug unit capabilities.

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.

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

Format: **TrOnchip.RESet**

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

Format:	**TrOnchip.state**

Opens the **TrOnchip.state** window.

Low Power Mode Debugging

MSP430 devices offer different low power modes. Low Power modes are designed to save energy. This is done by disabling certain clocks, the CPU itself and the JTAG connection. Which parts are deactivated depends on the Low Power Mode and the device. Details can be found in the device specific data sheet.

Avoid Loss of Device

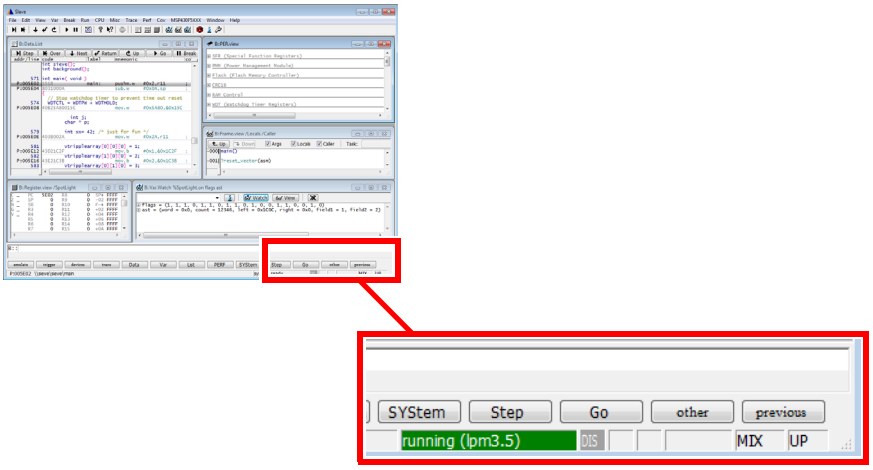
Devices that implement the LPMx.5 mode (i.e. LPM3.5 and/or LPM4.5) may disable JTAG in LPMx.5 mode. In this case the debugger has no access to the device until the next wake-up which cannot be done by the debugger. If the JTAG is kept alive during LPMx.5 the debugger may snoop the LPMx.5 state of the device and wake the device into debug mode. Hence the debugger might not be able to recover a device that goes to LPMx.5 shortly after the device went out from reset. This happens when JTAG is powered down to fast. Development recommendation:

- Add a wait time of 5 to 10 seconds in the LPMx.5 application before entering LPMx.5. The debugger has then a chance to bring the device to debug mode before LPMx.5 is entered.
- Check then if device can be recovered from LPMx.5. Remove wait time once the development of the LPMx.5 application has finished or if the device can be recovered by the debugger.

Supported Low Power Modes

LPM0 - LPM4	All MSP430 derivatives
LPMx.5	MSP430F5xxx, MSP430FR5xxx, MSP430F6xxx, CC430F5xxx, CC430F6xx. Some of these devices may not support LPMx.5. Refer to device data sheet.

The current low power mode of the device is indicated on the at the right bottom of TRACE32:



The following state information can be displayed

running	Core is active an running. Core power is up. All clocks are active.
running (lpm0)	Core is in lpm0. Core power is up. CPU and some clocks are disabled.
running (lpm1)	Core is in lpm1. Core power is up. CPU and some clocks are disabled.
running (lpm2)	Core is in lpm2. Core power is up. CPU and some clocks are disabled.
running (lpm3)	Core is in lpm3. Core power is up. CPU and some clocks are disabled.
running (lpm4)	Core is in lpm4. Core power is up. CPU and all clocks are disabled.
running (lpmx.5)	Core is in lpm3.5 or lpm4.5. Core power is down, JTAG is not accessible.
running (lpm3.5)	Core is in lpm3.5. Core power is down and JTAG is accessible
running (lpm4.5)	Core is in lpm4.5. Core power is down and JTAG is accessible

Please refer to your device specific data sheet for detailed information on supported low power modes.

Pinout of the 14-pin Debug Cable:

Signal	Pin	Pin	Signal
TDO|SBWTDIO RST	1	2	N/C
TDI TCLK	3	4	VTREF
TMS	5	6	N/C
TCK|TEST SBWTCK	7	8	TEST
GND	9	10	N/C
RST	11	12	N/C
N/C	13	14	NC

For details on logical functionality, physical connector, alternative connectors, electrical characteristics, timing behavior and printing circuit design hints, refer to the application note **"Arm Debug and Trace Interface Specification"** (app_arm_target_interface.pdf).

In case of problems contact support at **support@lauterbach.com**.

•