LAUTERBACH
DEVELOPMENT TOOLS

STM8 Debugger

Release 02.2026

MANUAL

STM8 Debugger

STM8 Debugger

Version 03-Apr-2026

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 document serves as a guideline for debugging STM8 MCUs and describes all MCU-specific TRACE32 settings and features.

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 Script

Lauterbach provides ready-to-run start-up scripts for known hardware that is based on STM8.

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/stm8/` subfolder of the system directory of TRACE32.

The on-chip FLASH and the EEProm memory can be programmed via the stm8.cmm script:

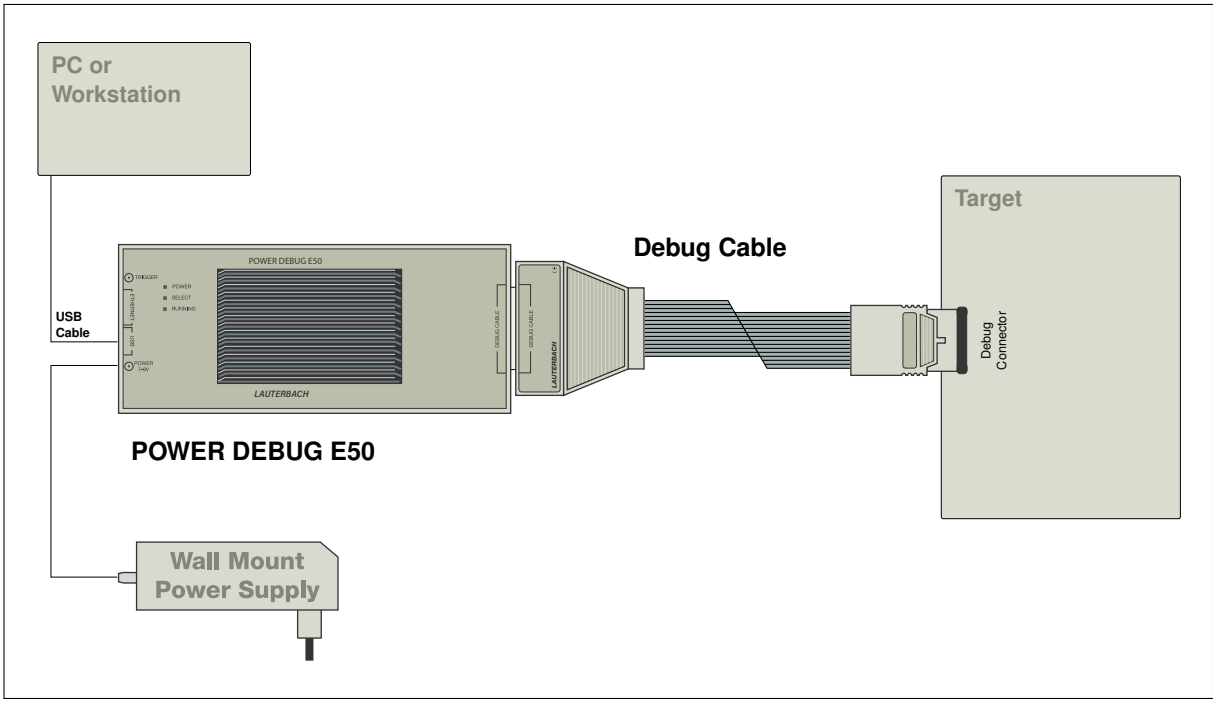
```
CD.DO ~~/demo/stm8/flash/stm8.cmm
```

Please be aware that you should check the Flash and EEProm size specified for your MCU in the stm8.cmm before executing this script.

Configuration

System Overview

Example configuration for an STM8 debugger.



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 STM8S005K6
```

This command selects the CPU type.

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 into the flash.

```
DO ~~/demo/stm8/flash/stm8.cmm
```

A typical start sequence of the STM8 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..0xffff         ; Specify the address range for
                                        ; caching

WinCLEAR                                ; Clear all windows

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

DO ~~/demo/stm8/flash/stm8.cmm          ; Load the target application into
                                        ; the Flash

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 is not connected or the SWD Interface is returning an error.	**SYStem.Mode.Up** **SYStem.Mode.Go**	The debugger expects to receive a confirmation for each command sent to the target. An error occurs in case the confirmation is not received.
The number of *<number>* accessed bytes in memory is not a multiple of the access *<size>* bytes.	No special event	Internal error, please consult your Lauterbach representative.
Memory *<address>* is not aligned to access *<size>*.	No special event	Internal error, please consult your Lauterbach representative.
Invalid memory access size: *<size>* bytes (@ *<address>*)	No special event	Internal error, please consult your Lauterbach representative.
Memory access timeout: Reading from *<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 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.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 SWD interface. Maybe there is the need to set jumpers on the target to connect the correct signals to the SWD connector.
- The target is in an unrecoverable state. Re-power your target and try again.
- The core is kept in reset.
- There is a watchdog which needs to be deactivated.

FAQ

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

SYStem.CPU — Select the used CPU

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

<cpu>: **STM8S005K6** | **STM8S003K3** | **STM8S001J3** | …

Default: STM8xxx.

Selects the processor type. All of the STM8 MCU cores with SWD Interface are supported.

SYStem.MemAccess — Select run-time memory access method

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

Default: Denied.

Enable
CPU (deprecated)
This option is not available at the moment.

Denied
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.

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

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

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

Default: Down.

Down Disables the debugger. The state of the CPU remains unchanged.

Go Resets the target and starts execution.

Up Resets the target and stops the CPU at the reset vector.

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
StandBy Not available.

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.

CPU specific TrOnchip Commands

The **TrOnchip** command group is not available for the STM8 debugger.

Breakpoints

Software breakpoints

The Microchip STM8 architecture does not support software breakpoints.

On-chip breakpoints for instructions

The STM8 MCUs support a total of two 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 STM8 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.

Memory Classes

The following memory access classes are available:

Access Class	Description
D	Data
P	Program

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 P is used to denote the Flash memory.

```
Data.dump P:0x00
```

Since the STM8 architecture uses a Unified Memory Architecture, the following two examples return the same results.

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

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

Connector Type and Pinout

Debug Cable

Pin	Signal
1	VDD
2	PD1
3	GND
4	RESET[PA0]

•