LAUTERBACH
DEVELOPMENT TOOLS

Blackfin Debugger

Release 09.2026

MANUAL

Blackfin Debugger

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

ICD In-Circuit Debugger ..... 🗁

Processor Architecture Manuals ..... 🗁

Blackfin ..... 🗁

Blackfin Debugger **1**

Introduction **4**

Brief Overview of Documents for New Users 4

Demo and Start-up Scripts 4

Location of Debug Connector 5

Warning **5**

Quick Start JTAG **6**

Troubleshooting **8**

SYStem.Up Errors 8

FAQ **8**

Configuration **9**

System Overview 9

Blackfin specific SYStem Commands **10**

SYStem.CONFIG Configure debugger according to target topology 10

Daisy-Chain Example 13

TapStates 14

SYStem.CONFIG.CORE Assign core to TRACE32 instance 15

SYStem.CPU CPU type selection 16

SYStem.JtagClock JTAG clock selection 17

SYStem.LOCK Lock and tristate the debug port 17

SYStem.MemAccess Select run-time memory access method 18

SYStem.Mode System mode selection 19

SYStem.Option.IMASKASM Interrupt disable 19

SYStem.Option.IMASKHLL Interrupt disable 20

Breakpoints **21**

Software Breakpoints 21

On-chip Breakpoints 21

Breakpoint in ROM 21

Example for Breakpoints 22
Memory Classes **23**
CPU specific TrOnchip Commands **24**
JTAG Connector **25**

Blackfin Debugger

Version 25-Sep-2026

Introduction

This document describes the processor specific settings and features for the Blackfin Embedded Media Processor. TRACE32-ICD supports all Blackfin devices which are equipped with the JTAG debug interface.

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.

If some of the described functions, options, signals or connections in this Processor Architecture Manual are only valid for a single CPU the name is added in brackets.

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 Blackfin 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/blackfin/ subfolder of the system directory of TRACE32.

Location of Debug Connector

Locate the debug connector on your target board as close as possible to the processor to minimize the capacitive influence of the trace length and cross coupling of noise onto the JTAG signals.

Warning

Signal Level

The debugger output voltage follows the target voltage level. It supports a voltage range of 0.4 … 5.2 V.

ESD Protection

NOTE:	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: • Disconnect the debug cable from the target while the target power is off. • Connect the host system, the TRACE32 hardware and the debug cable. • Start the TRACE32 software. • Connect the debug cable to the target. • Switch the target power ON. Power down: • Switch off the target power. • Disconnect the debug cable from the target.

Starting up the debugger is done as follows:

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

```
B:
```

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

```
SYStem.CPU BF537
```

3. Enter debug mode:

```
SYStem.Up
```

This command resets the CPU and enters debug mode. After the execution of this command access to the registers and to memory is possible. Before performing the first access to external SDRAM or FLASH the External Bus Interface Unit (EBIU) must be configured.

4. The following command sequence is for the BF537 processor and configures the SDRAM controller with default values that were derived for maximum flexibility. They work for a system clock frequency between 54 MHz and 133 MHz.

   In the example a ST M29W320DB flash device is used in 16-bit mode. All four memory banks and CLKOUT are enabled.

```
; configure SDRAM controller                    ; EBIU_SDGCTL
Data.Set 0xFFC00A1sLONG 0x0091998D              ; EBIU_SDBCTL
Data.Set 0xFFC00A14 %WORD 0x0025                ; EBIU_SDRRC
Data.Set 0xFFC00A1C %WORD 0x03A0


; enable all flash memory banks and clock
out                                             ; EBIU_AMGCTL
Data.Set 0xFFC00A00 %WORD 0x00FF


; ST M29W320DB flash device in 16-bit mode
FLASH.Create 1. 0x20000000--0x20003FFF 0x4000  AM29LV100 Word
FLASH.Create 1. 0x20004000--0x20007FFF 0x2000  AM29LV100 Word
FLASH.Create 1. 0x20008000--0x2000FFFF 0x8000  AM29LV100 Word
FLASH.Create 1. 0x20010000--0x203FFFFF 0x10000 AM29LV100 Word
```

5. Load the program.

```
Data.LOAD.Elf demo.dxe       ; The file demo.dxe is in ELF format
```

The option of the **Data.LOAD** command depends on the file format generated by the compiler. A detailed description of the **Data.LOAD** command is given in the **"General Commands Reference"**.

The start-up sequence can be automated using the programming language PRACTICE. 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 device prompt

WinClear                     ; Delete all windows

SYStem.CPU BF537             ; select the processor

SYStem.Up                    ; Reset the target and enter debug mode

Data.Load.Elf sieve.dxe      ; Load the application

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

List.Mix                     ; Open disassembly window              *)

Register.view                ; Open register window                 *)

PER.view                     ; Open window with peripheral register *)

Break.Set sieve              ; Set breakpoint to function sieve

Break.Set 0x1000 /p          ; Set on-chip breakpoint to address 1000
                             ; Refer to the restrictions in
                             ; On-chip Breakpoints.
```

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

Troubleshooting

SYStem.Up Errors

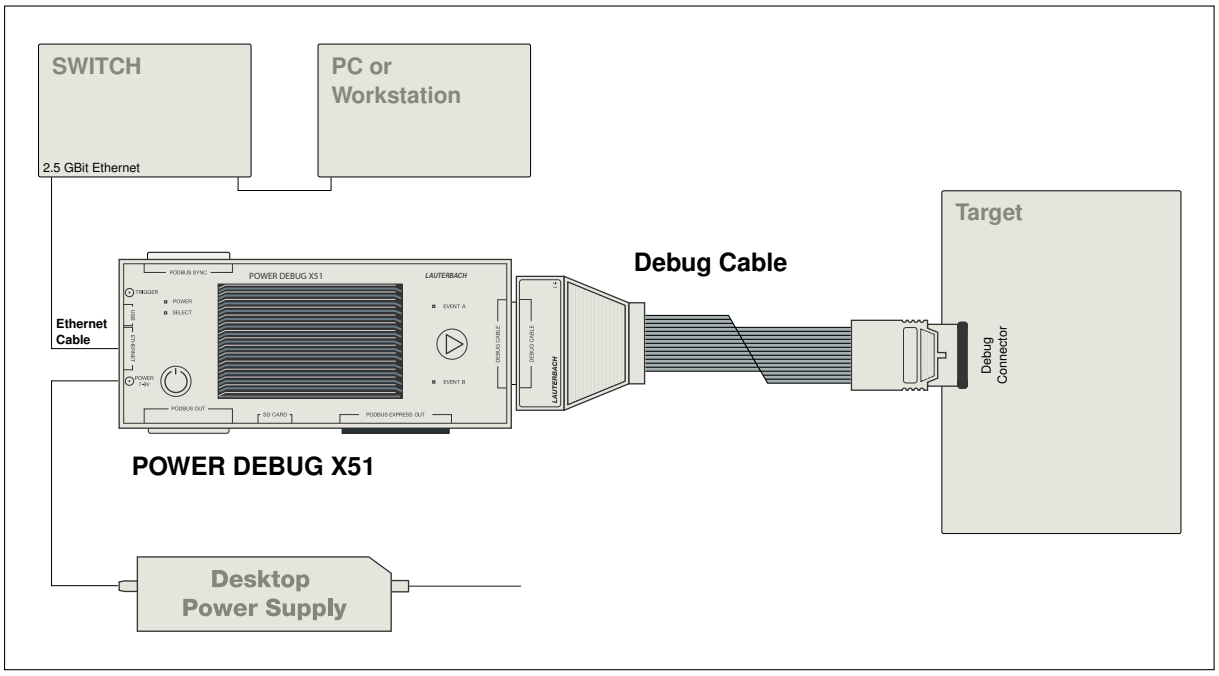
The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

All	The target has no power.
All	There are additional loads or capacities on the JTAG lines.
All	The JTAG clock is too fast.

FAQ

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

System Overview



SYStem.CONFIG Configure debugger according to target topology

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

<parameter>: **CORE** *<core>*

<parameter>: (JTAG):
DRPRE *<bits>*
DRPOST *<bits>*
IRPRE *<bits>*
IRPOST *<bits>*
DAPDRPOST *<bits>*
DAPDRPRE *<bits>*
DAPIRPOST *<bits>*
DAPIRPRE *<bits>*
TAPState *<state>*
TCKLevel *<level>*
TriState [**ON** | **OFF**]
Slave [**ON** | **OFF**]
DEBUGPORTTYPE [**JTAG** | **SWD**]
SWDPIDLEHIGH [**ON** | **OFF**]
SWDPTargetSel *<value>*

The **SYStem.CONFIG** commands have no effect in Simulator. These commands describe the physical configuration at the JTAG port and the trace port of a multi-core hardware target. Since the simulator normally just simulates the instruction set, these commands will be ignored. Refer to the relevant **Processor Architecture Manual** in case you want to know the effect of these commands on a debugger.

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. ARM + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See **Daisy-chain Example**.
For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

TriState has to be used if several debuggers (“via separate cables”) are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. 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.



Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).

CORE

For multicore debugging one TRACE32 PowerView GUI has to be started per core. To bundle several cores in one processor as required by the system this command has to be used to define core and processor coordinates within the system topology.
Further information can be found in **SYStem.CONFIG.CORE**.

… **DRPOST** *<bits>*

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

Default: 0.

… **DRPRE** *<bits>*

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

Default: 0.

… **IRPOST** *<bits>*

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

Default: 0.

… **IRPRE** *<bits>*

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

Default: 0.

TAPState

(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.

TCKLevel (default: 0) Level of TCK signal when all debuggers are tristated.

TriState (default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.

Slave (default: OFF) If more than one debugger share the same debug port, all except one must have this option active.
JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).

DEBUGPORTTYPE [**JTAG** | **SWD**] It specifies the used debug port type “JTAG”, “SWD”. It assumes the selected type is supported by the target.

Default: JTAG.

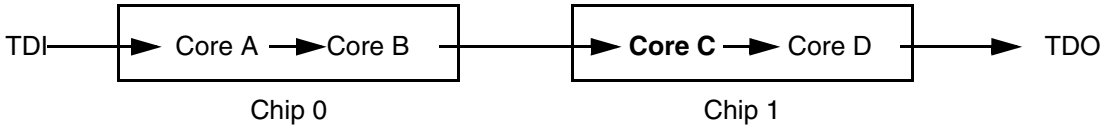
SWDPIdleHigh [**ON** | **OFF**] Keep SWDIO line high when idle. Only for Serialwire Debug mode. Usually the debugger will pull the SWDIO data line low, when no operation is in progress, so while the clock on the SWCLK line is stopped (kept low).

You can configure the debugger to pull the SWDIO data line high, when no operation is in progress by using **SYStem.CONFIG SWDPIdleHigh ON**

Default: OFF.

SWDPTargetSel *<value>* Device address in case of a multidrop serial wire debug port.

Default: none set (any address accepted).



Below, configuration for core C.

Instruction register length of

- Core A: 3 bit
- Core B: 5 bit
- Core D: 6 bit

```
SYStem.CONFIG.IRPRE  6.              ; IR Core D
SYStem.CONFIG.IRPOST 8.              ; IR Core A + B
SYStem.CONFIG.DRPRE  1.              ; DR Core D
SYStem.CONFIG.DRPOST 2.              ; DR Core A + B
SYStem.CONFIG.CORE 0. 1.             ; Target Core C is Core 0 in Chip 1
```

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

Format:	**SYStem.CONFIG.CORE** *<core_index> <chip_index>* **SYStem.MultiCore.CORE** *<core_index> <chip_index>* (deprecated)
<chip_index>:	**1** … **i**
<core_index>:	**1** … **k**

Default *core_index*: depends on the CPU, usually 1. for generic chips

Default *chip_index*: derived from CORE= parameter of the configuration file (config.t32). The CORE parameter is defined according to the start order of the GUI in T32Start with ascending values.

To provide proper interaction between different parts of the debugger, the systems topology must be mapped to the debugger's topology model. The debugger model abstracts chips and sub cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected, a generic chip or non-generic chip is created at the default *chip_index.*

Non-generic Chips

Non-generic chips have a fixed number of sub cores, each with a fixed CPU type.

Initially, all GUIs are configured with different *chip_index* values. Therefore, you have to assign the *core_index* and the *chip_index* for every core. Usually, the debugger does not need further information to access cores in non-generic chips, once the setup is correct.

Generic Chips

Generic chips can accommodate an arbitrary amount of sub-cores. The debugger still needs information how to connect to the individual cores e.g. by setting the JTAG chain coordinates.

Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a non-generic chip, two GUIs are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is valid since every new GUI uses a new *chip_index* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**BF531** | **BF532** | **BF533** | **BF534** …

Default selection: BF534.

Selects the CPU type.

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

Default frequency: 1 MHz.

Selects the JTAG port frequency (TCK). Any frequency up to 50 MHz can be entered, it will be generated by the debuggers internal PLL.
For CPUs which come up with very low clock speeds it might be necessary to slow down the JTAG frequency. After initialization of the CPUs PLL the JTAG clock can be increased.



If there are buffers, additional loads or high capacities on the JTAG/COP lines, reduce the debug speed.

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 Denied** | **StopAndGo** | **BTC**

BTC "BTC" allows a non-intrusive memory access while the core is running, if a Background Telemetry Channel (BTC) is defined in your application. Any information on how to create such a channel can be found in Analog Devices' VisualDSP++ user's manual. The JTAG clock speed should be as fast as possible to get good performance

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.

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
Attach
Up

Down	Disables the debugger.
Go	Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is in the system.up mode and running. Now, the processor can be stopped with the break command or if a break condition occurs.
Attach	User program remains running (no reset) and the debug interface is initialized.
Up	Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging.
StandBy	Not supported.
NoDebug	Not supported.

SYStem.Option.IMASKASM Interrupt disable

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

Mask interrupts during assembler single steps. Useful to prevent interrupt disturbance during assembler single stepping.

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

Mask interrupts during HLL single steps. Useful to prevent interrupt disturbance during HLL single stepping.

Breakpoints

There are two types of breakpoints available: software breakpoints and on-chip breakpoints.

Software Breakpoints

Software breakpoints are the default breakpoints. A special breakcode is patched to memory so it only can be used in RAM or FLASH areas.There is no restriction in the number of software breakpoints.

On-chip Breakpoints

The Blackfin processor has a total of six instruction and two data on-chip breakpoints.

A pair of two breakpoints may be further grouped together to form a range breakpoint. A range breakpoint can be including or excluding. In the first case the core is stopped if an address in the range is detected, in the second case the core is stopped when an address outside of the range is observed.

Breakpoint in ROM

With the command **MAP.BOnchip** *<range>* it is possible to inform the debugger about ROM (FLASH,EPROM) address ranges in target. If a breakpoint is set within the specified address range the debugger uses automatically the available on-chip breakpoints.

Example for Breakpoints

Assume you have a target with FLASH from `0x20000000` to `0x200FFFFF` and RAM from `0x0` to `0x1000000`. The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x20000000--0x200FFFFF
```

The following breakpoint combinations are possible.

Software breakpoints:

```
Break.Set 0x0 /Program                ; Software Breakpoint 1

Break.Set 0x1000 /Program             ; Software Breakpoint 2
```

On-chip breakpoints:

```
Break.Set 0x20000100 /Program         ; On-chip Breakpoint 1

Break.Set 0x2000ff00 /Program         ; On-chip Breakpoint 2
```

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
D	Data

CPU specific TrOnchip Commands

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

JTAG Connector

Signal	Pin	Pin	Signal
GND	1	2	EMU-
N/C	3	4	GND
VDDIO	5	6	TMS
N/C	7	8	TCK
N/C	9	10	TRST-
N/C	11	12	TDI
GND	13	14	TDO

JTAG Connector	Signal Description	CPU Signal
TMS	JTAG-TMS, output of debugger	**TMS**
TDI	JTAG-TDI, output of debugger	**TDI**
TCK	JTAG-TCK, output of debugger	**TCK**
/TRST	JTAG-TRST, output of debugger	**/TRST**
TDO	JTAG-TDO, input for debugger	**TDO**
/EMU	JTAG Emulation Flag	**/EMU**
VDDIO	This pin is used by the debugger to sense the target I/O voltage and to set the drive levels accordingly. If the sensed voltage level is too low (e.g. target has no power) the debugger powers down its drivers to prevent the target from damage.	**VDDIO**

•