

Debugger Tutorial

Release 02.2024



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Debugger Getting Started	Þ
Debugger Tutorial	1
About the Tutorial	4
Set up the TRACE32 Environment	6
Set Up the Debug Environment The Welcome Dialog A Typical Set Up Procedure How to Generate TRACE32 Support Information	7 7 8 10
Start-Up Scripts Write a Start-Up Script Run a Start-up Script Automated Start-up Scripts	11 11 12 13
User Interface - TRACE32 PowerView TRACE32 Command Line and Softkeys Window Captions - What Makes Them Special in TRACE32	14 16 17
Debugging the Program Basic Debug Commands Debug Modes Displaying the Stack Frame	18 18 19 21
Breakpoints Setting Breakpoints Listing all Breakpoints Setting Read/Write Breakpoints	22 22 23 24
Variables Displaying Variables Displaying Variables of the Current Program Context Using the Symbol Browser Formatting Variables Modifying Variables	25 26 26 27 28
Memory Displaying Memory Modifying Memory	29 29 30

Exit TRACE32	31
Getting Online Help	 32

Version 04-Mar-2024

About the Tutorial

What is it about?

This is a tutorial for all hardware-assisted TRACE32 Debuggers (TRACE32-ICD) that are implemented using an on-chip debug interface. The most common used on-chip debug interface is JTAG.

Preconditions:

The tutorial assumes that the TRACE32 debugger software is already installed. Please refer to **"TRACE32 Installation Guide"** (installation.pdf) for information about the installation process. Also a basic knowledge of software debugging and the C-programming language is helpful. This is required in order to be able to follow the example code found in this tutorial. Moreover, a basic knowledge of the target processor and used assembler/compiler is necessary to get your debug system running.

To work with a hardware-assisted TRACE32 Debugger (ICD) a working target system is required. Please also take care of the proper sequence on powering up/down:

- Power Up: debugger target
- Power down: target debugger

Purpose of this tutorial:

The purpose of this tutorial is to do the basic steps in setting up the debug environment and to make you familiar with the basic features of TRACE32. For simplicity, we use in this tutorial a single-core system example.

How to use this tutorial:

The tutorial contains a guided debug session. It uses a simple C-program example to show you the basic debug features. You should perform a number of exercises as you read this tutorial. We recommend to go completely through all chapters.

Where can I get more information:

A detailed overview of all debug features offers "Training Basic Debugging" (training_debugger.pdf).

The TRACE32 PowerView GUI contains a detailed online help that offers descriptions of all debug features. Refer to the "**Online Help**" chapter on how to start and use the online help system.

How long does it take?

60 minutes

After installing the TRACE32 on your host PC, a default environment is set up. This configuration can be adapted to your debugging environment.

By default the configuration file **config.t32** in the system directory (e.g. **C:\T32** or **/opt/t32**) is used. The option **-c** allows you to define your own location and name for the configuration file.



The TRACE32 executables are named *t32m*<*architecture>*[.*exe*] (e.g. t32marm.exe in our example) and are placed in the TRACE32 system directory (**SYS=**, see **[A]**) under **bin**<*os>* (e.g. **bin**/**windows64** or **bin**/**pc_linux64**).

Example for Windows:

```
C:\T32\bin\windows64\t32marm.exe -c C:\workspace\config_user.t32
```

Example for Linux:

```
/opt/t32/bin/pc_linux64/t32marm -c /home/user/config_user.t32
```

In order to set up your debugger, you need some knowledge about your CPU and your target configuration. A basic start-up procedure and the CPU specific setting for the debugger are described in the **Processor Architecture Manual** that can be opened by selecting the menu **Help > Processor Architecture Manual**.

The Welcome Dialog

Per default a **Welcome to TRACE32!** dialog is displayed when TRACE32 PowerView is started. This dialog shows the target architecture and debug module. Additionally, the dialog includes links to most important manuals.

Welcome to TRACE32!
TRACE32 PowerView for ARM / PowerDebug PRO
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 to fit your exact system setup and configuration.
Related manuals
ARM Debugger
👸 Debugger Basics - Training
ma Training Script Language PRACTICE
Show this dialog at start 🔗 Help 👬 Start with examples



Please make sure that you have not started, by mistake, a **TRACE32** Instruction Set Simulator instead of an **Debugger**. You would see in this case the message "**TRACE32 PowerView for ... / Simulator**" in the **Welcome to TRACE32!** dialog. Lauterbach provides ready-to-run PRACTICE start-up scripts for public known architecture hardware. You can search for PRACTICE scripts by pushing the **Start with examples** button from the **Welcome to TRACE32!** dialog.

P Search for scripts			- • ×
Search Selection Manuals			
Example search: OMAP44* Linux			
arm* flash v 💥 🎁 Search 699 dem	o files found.		
Filter			
None O Chip O Board			
Search for newest scripts at https://www.lauterbach.com/scripts.html			
CONFIG			
Title	Chip	Board	V
66AK2H12 (KeyStone2) Serial FLASH Programming Script	66AK2H12	-	^
Example for illash declaration of ST A2Fx internal illash	-	-	
Serial plass Programming script for AU25-SC384-E2LITE (Analog Device)	ADSP-5C58" ADuC702*		
Marvell 800500 OSPT ELAST Programming Template	8805050		
OSPI FLASH Program script for the AM654x	-	-	
Example for flash declaration of Ambig Apollo internal flash	APOLLO*	-	×
<			>

You can also inspect the demo directory manually from the TRACE32 system directory.

A Typical Set Up Procedure

This chapter describes a typical start-up procedure for the debugger. To demonstrate the steps needed, we will do a manual setup. Later on we will show you how to use PRACTICE scripts (*.cmm) for this purpose. For simplicity, we use here a single-core system example. The *SYStem* Window provides all CPU specific settings. You can open this window by selecting the menu **CPU** > **SYStem Settings...**

1. Inform TRACE32 about the core/chip on your target **[A, B]**, if an automatic detection is not possible.

B::SYStem				
Mode MemAccess Dap Dap Openation Dap Openat	Option Option Option	Option DisMode	B:SYS.CPU CortexM4 CortexM0 CortexM1 CortexM1 CortexM3 CortexM3 CortexM3 CortexM3 CortexM3 CortexM3 CortexM3 CortexM4 C	× +B
SYStem.DETECT CI	PU	Auto	o detection of CPU	
SYStem.CPU <cpu></cpu>		Sele	ect the CPU/chip	

- 2. Some cores require additional settings before the communication can be established. You can set these options from the **SYStem** windows **[C]**. For details, refer to the **Processor Architecture Manual**.
- 3. Establish the communication between the debugger and the core. The most common way is to select the mode *Up* [D].

If *Up* is selected, the following steps are performed:

- Reset of the core.
- Initialization of the communication between the debugger and the core.
- Stop of the core at the reset vector, if supported by the core in use.

SYStem.Up	Establish the communication between the debugger and the
	core

A second useful way to establish the communication between the debugger and the core is *Attach* [E]. *Attach* allows to connect the debugger to an already running core.

SYStem.Mode Attach	Establish the communication between the debugger and the
	target core (without reset)

If you get an error after selecting Up or Attach, refer to the Processor Architecture Manual.

- 4. The next step is to download your application into the target:
 - if the application should run out of RAM then you can directly use the **Data.LOAD** command. Just type **Data.LOAD.*** then select the file you want to download.
 - For on-chip and off-chip NOR as well as memory-mapped serial flash programming, refer to the FLASH command group and to the "Onchip/NOR FLASH Programming User's Guide" (norflash.pdf).

A video tutorial about programming the processor internal FLASH in TRACE32 is available here:

support.lauterbach.com/kb/articles/flash-programming

For non-memory-mapped flash programming (NAND, SPI, eMMC) refer to the FLASHFILE command group. Depending on your flash device, you can find more details in "NAND FLASH Programming User's Guide" (nandflash.pdf), "Serial FLASH Programming User's Guide" (serialflash.pdf) or "eMMC FLASH Programming User's Guide" (emmcflash.pdf).

The different debug scenarios are described in details in "Establish Your Debug Session" (tutor_setup.pdf).

If you need assistance in setting up the debugging environment, be sure to include detailed system information.

1. To generate a system information report, choose Help > Support > Systeminformation ...

🖉 Generate TRACE32 Support Information				
Press the following button to get help on how to generate Support Information:				
Company	Lautorbach	Dopartment	Training	
Prefix:		Department.	rraining	
Firstname:	Andrea			
Surname:	Martin			
Street:	Altlaufstr. 40	P.O. Box:		
City:	Hoehenkirchen-Siegertsbrunn	ZIP Code:	85635	
Country:	Germany			
Telephone:	++49-8104-9843-555			
eMail:	training@lauterbach.com			
Product.:	Power Debug Interface / USB 3	3		
Target CPU:	CortexA9			
Hostsystem:	PC Windows 7 🔹			
Compiler:	ARM			
RealtimeOS:	None		Sa	afe Mode: 🔲
G	enerate Support Information:	Save to Clip	oboard S	ave to File

- 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.

It is strongly recommended to summarize the commands, that are used to set up the debug environment, in a start-up script. The script language PRACTICE is provided for this purpose.

The standard extension for a script file is . cmm.

Write a Start-Up Script

The debugger provides an ASCII editor, that allows to write, to run and to debug a start-up script.

PEDIT <file></file>	Open < <i>file></i> with the script editor	
PEDIT my_startup		

The debugger provides two commands, that allow you to convert debugger configuration information to a script.

STOre <file> [<item>]</item></file>	Generate a script that allows to reproduce the current settings			
ClipSTOre [<item>]</item>	Generate a command list in the clip-text that allows to reproduce the current settings			
STOre system_settings	SYStem	; Generate a script that allows you ; to reproduce the settings of the ; SYStem window at any time		
PEDIT system_settings		; Open the file <pre>system_settings</pre>		

ClipSTOre SYStem	; Generate a command list that
	; allows you to reproduce the
	; settings of the SYStem window
	; at any time
	; The generated command list can be
	; pasted in any editor



ChDir.DO <file>

Change directory and run script

```
ChDir.DO my_startup.cmm
```

There are two ways to define a start-up script, that is automatically started, when the debugger is started.

1. Define start-up script in conjunction with the executable

The debugger-executable can be started with the start-up script as parameters.

c:\t32\t32arm.exe -s g:\and\arm**start.cmm**

2. Use T32Start to define an automated start-up script

If you use T32Start to start the debugger, an automated start-up script can be defined.

1 T32Start V2.2.22 *	- 0 X
🔺 🛅 Configuration Tree	
⊳ - 🛅 Settings	Start
▷ 😰 Example Configuration	Add
ARM	1.100.00111
🖌 🖛 🖬 1: Podbus Device Chain	Delete
⊿ · ബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബബ	
🕢 💽 ConnectionType: USB	Up
⊳ - 🛅 USB Settings	Down
🔺 🐴 1: Core	0.01111
Target: ARM/XScale/Janus	Instances
Advanced Settings	
⊳ - <mark>©</mark> Paths	Information
Display	Save and Exit
A 🛅 StartupScript	Cauc
Source: File	Jave
I File: C:\132_ARM\demo\arm\compiler\arm\arm.cmm	Help
Harameters:	
	1
File C:\T32_ARM\demo\arm\compiler\arm\arm.c Link to Edit	
ID: //Configuration2/Podbus Device Chain/Power Debug II/Core/CoreAdvancedOptions/	.11

The graphical user interface (GUI) of TRACE32 is called TRACE32 PowerView.

The following screen shot presents the main components of the user interface.



We'll briefly explain the GUI using the List command and List window as an example.

A video tutorial about the TRACE32 PowerView GUI is available here: support.lauterbach.com/kb/articles/introduction-to-trace32-gui

Do one of the following to open the **List** window:

- Choose View menu > List Source
- or, at the TRACE32 command line, type: List (or L)

The **List** window displays the code in assembler mnemonic and HLL (HLL stands for High-Level Language and means the programming language of your source code).



In the **List** window, the gray bar indicates the position of the program counter (PC). Right now, it is located on the symbolic address of the label **main**.

A video tutorial about the source code display in TRACE32 is available here: **support.lauterbach.com/kb/articles/displaying-the-source-code**

To summarize it, you can execute commands in TRACE32 PowerView via the usual suspects:

- 1. Menus on the menu bar
- 2. Buttons on the main toolbar and the buttons on the toolbars of TRACE32 windows
- 3. Context menus in TRACE32 windows

Additionally in TRACE32, you can execute commands via the TRACE32 command line.

TRACE32 commands are **not** case sensitive: **register.view** is the same as **Register.view**

UPPER CASE letters indicate the short forms of commands and must not be omitted. All lower case letters can be omitted. This makes short forms an efficient time saver when you are entering frequently-used commands in the command line. Examples:

- Instead of the long form **Register.view** type just the short form **r** or **R**
- Instead of the long form List type just the short form 1 or L

The softkeys are below the command line. The camel casing (i.e. upper and lower case letters) on any softkey tells you the long form of a command. The softkeys guide you through the command input, displaying all possible commands and parameters.

Example - To assemble the Data.dump command using the softkeys:

- 1. Click Data.
- 2. Click **dump**.

B:

- 3. Type the *<range>* or *<address>* you want to dump. For example, **0x1000--0x2000**
- 4. Click **[ok]** to execute the command. The **Data.dump** window opens.

Command line
late trigger devices trace Data Var List other previous Softkeys
B::DATA. [ok] dump View Print List Set Assemble other previous
SR:00001FF8 \\armle\arm\main system ready MIX UP
B::DATA.DUMP_0x10000x2000
Lokj <range> <address> options previous SR:00001FF8 \armle\arm\main system ready MIX UP</address></range>

The command with which you open a window will be shown as the window caption. The parameters and options are also included in the window caption.

<mark>▲</mark> TRACE32 ARM <u>File E</u> dit <u>V</u> iew Var <u>B</u> reak <u>R</u> un <u>C</u> PU <u>M</u> isc <u>T</u> race	e <u>P</u> erf C <u>ov W</u> indow <u>H</u> elp	×												
▶ ▶ ↓ ✔ ⊄ ▶ Ⅱ 🖄 🤋 № 🌚 🗄	🗐 斑 🔲 🖬 📾 🏟 🌘 🤰 📰													
B::List														
Step Noer Next	C Up													
SR:00001FF4 00005AF4	dcd 0x5AF4													
586 { SR:00001FF8 E92D4030 SR:00001FFC E24DD008 int j;	<pre>main: stmdb r13!,{r4-r5,r14} sub r13,r13,#0x8 ; r13,r13,#8</pre>													
590 SR:00002000 E3A <u>accress</u> <u>v</u> SR:00002004 E59 SD:00001000 ►59F0034 SR:00002004 E59	0 V 4 8 C 0123456789ABCDEF 4 E59F1034 E59F3034 E1500001 409E409E409E409E500F													
SR:0002008 ESC SD:00001010 0A00005 591 SD:00001020 3AFFFFB SR:0002002 E3A SD:00001030 34832004 SR:00002010 E59 SD:00001040 0000559C SR:00002014 E5C SD:00001050 E1A0F00E	S E1510003 54902004 54612004 X00FX0017-5647-34 E B E59F101C E3A02000 E1530001 FFF: 56850.658NS5 4 3AFFFFC EA000BD2 0000559C - 3475FF: 5704800 C 000057A0 00006F08 E3A00000 20008590 500000005 E E5901000 E2811001 E5801000 564500550500000													
592 SD:00001060 E1A0F00E SR:00002018 E3A SD:00001070 E58D2000 SR:0000201C E59 SD:00001080 E1A0000D SR:00002020 E5C SD:00001090 E3A02000 SP3 SD:00001040 E2822001	E E9204008 E59F0094 E59020A0 355530-341955305 0 E59D0000 E2800001 E58D0000 0 0.0550055400520055 D EBFFFF2 E59F0078 EBFFFFF0 6055765765765765 0 E3520005 BA00002 EA000009 0.055076577540474045 1 EAFFFFA E59D1000 E0000291 3.0557675400753006 ~													
SR:0002024 E3A SR:00002028 E59 SR:0000202C E5C01001	strb r1,[r0,#0x1]	-												
	•													
B: Data.dump 0x10000x2000														
[ok] options	previous													

You can **re-**insert a command from a window caption (a) into the command line (b) in order to modify the command. Let's do this with the **Register** window.

- 1. Choose **View** menu > **Register**.
- 2. Right-click the window caption (a).
- 3. Modify the command, e.g. by adding the /SpotLight option: It will highlight changed registers.

Register.view	(a)le:Reģister.view////////////////////////////////////
R4 1 R12 25 R5 564C R13 0FE4 R6 0 R14 00 R7 0 PC 2258 SPSR 10 CPSR 800000D3	R4 1 R12 5 R5 564C R13 0FE4 R6 0 R14 0 R7 0 PC 22A0 SPSR 10 CPSR 800000D3 ∢
	(b) B::B::Register.view /SpotLight [ok] options

- 4. Click **[ok]** to execute the modified command.
- 5. Click Single Step on the TRACE32 toolbar. Changed registers are highlighted immediately.

Basic Debug Commands

The basic debug commands are available via the **Run** menu, the toolbar of the **List** window, the main toolbar, and via the TRACE32 command line.

Single stepping **I** is one of the basic debug commands.



TRACE32 provides also more complex debug control commands. You can step until an expression changes or becomes true.

Example: **Var.Step.Till i>11.** single-steps the program until the variable **i** becomes greater than **11.** Please note that TRACE32 uses a dot to denote decimal numbers.

A TRACE32 ARM File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help 😫 🛓 🧾 - - -B::List Step Vor addr/line code SR:00001FF4 00005 🖄 Mode 🗸 Return Find: ↓ Next Ċ Up Go Go Break label mnemonic commen 0x5AF4 0005AF dcd main() 586 SR:00001FF8 E92D4030 stmdb r13!,{r4-r5,r14} r13,r13,#0x8 main. SR:00001FFC sub ; r13,r13,#8 int j; char * p; vtripplearray[0][0][0] = 1; 590 r0,#0x1 SR:00002000 E3A00001 ; r0,#1 mov SR:00002004 E59F1200 ldr r1,0x220C • B:: emulate trigger devices Data other previous trace Var List SR:00001FFC \\armle\arm\main+0x4 stopped MIX UP С В А

Take a look at the state line at the bottom of the TRACE32 main window:

The state line tells you:

- A The (symbolic) address of the current cursor position. The program counter (PC) is highlighted in gray.
- **B** The state of the debugger: **stopped** means program execution is stopped. You can now, for example, inspect or change memory.
- **C** The state line displays the currently selected debug mode: The code display will be **HLL** (High Level Language) or **ASM** (assembler) or a **MIX**ed mode with HLL and its corresponding assembler mnemonic.
- 6. On the toolbar of the List window, click Mode to toggle the debug mode to HLL.

ĺ	Debug mode HLL		Debug mode MIX
🗾 [B::List]	- • •		[B::List]
Step	🖌 Over 🛛 🕂 Next 🖌 🖋 Return 🖉 🙋		N Step N Over ↓ Next ✔ Return ♥ Up ► Go II
581	for $(x = 0.0; x < 62.8;$		SR:00001FF4 00005AF4 dcd 0x5AF4
582 583 }	sinewave[index++] =		main()
main()			SR:00001FF8 E92D4030 main: stmdb r13!,{r4-r5,r14}
586 {	1		SR:00001FFC E24DD008 sub r13,r13,#0x8
	int j; char * p;		char * p;
590	vtripplearrav[0][0][0] = 1:	L.	590 vtripplearray[0][0] = 1;
591	vtripplearray[1][0][0] = 2;	•	SR:00002000 E3A00001 mov r0,#0x1
592	vtripplearray[0][1][0] = 3;		SR:00002004 E59F1200 Idr r1,0x220C
595	Vtripplearray[0][0][1] = 4;		591 vtripplearray[1][0][0] = 2
595	func2():		SR:0000200C E3A01002 mov r1,#0x2
			SR:00002010 E59F01F4 ldr r0,0x220C
597	func2a();		SR:00002014 E5C 0C strb r1,[r0,#0x0C]
500	func2h():		592 Vtrippiearray[0][1][0] = 3; SR \cdot 00002018 E3400003 mov r0 \pm 0x3
355	runczo(),		SR:0000201C E59F11E8 1dr r1,0x220C
601	func2c();		SR:00002020 E5C10004 strb r0,[r1,#0x4]

.

- Click Step Step.
 The step you are taking is debug mode HLL goes to the next source code line.
- 8. Click Mode again to toggle the debug mode to **MIX**.
- 9. Click H Step Step.

This time, the step executes one assembler line.

10. Right-click a code line, and then select **Go Till**. The program execution starts. It stops when the program reaches the selected code line.

TRACE32 ARM	
<u>File Edit View Var Break Run CPU Misc Trace Perf Co</u>	<u>v W</u> indow <u>H</u> elp
N M ↓ √ Ć ▶ II 🕅 \$ № 🏐 II 📰 🏢 📗	🐱 🗟 🖾 🗳 🧎 📃
I (Buliet)	
N Step N Over ↓ Next ← Return C Up	Go Break Mode Find:
607 ast. count = 12345:	mnemonic
SR:00002050 E3A00039	mov r0,#0x39 ; r0,#57
SR:00002054 E2800A03	add r0,r0,#0x3000 ; r0,r0,#12288
SR:00002058 ESTFILISC SR:0000205C E5810004	str r0.[r1.#0x4]
608 ast.left = *	
SR:00002060 E51F1144	Idr r1,0x1F24
SR:00002004 E51F0148	str r1. r0 #001
609 ast.field1 = 1;	Program Address
SR:0000206C E51F0150	
SR:00002074 E3C00003	bic r0.r0.
SR:00002078 E3800001	onn n0,n0, 🕲 Breakpoints 🕨
SR:0000207C E51F1160	ldr r1,0x1 📷 Display Memory
610 ast.field2 = 2:	Bookmark
SR:00002084 E51F0168	ldr r0,0x11 📣 Toggle Bookmark
SR:00002088 E5900010	ldr r0, [r0
SR:0000208C ESCOULC SR:00002090 E3801008	orr r1,r0,- K Edit Source
SR:00002094 E51F0178	ldr r0,0x11 View Info
SR:00002098 E5801010	str r1,[r0] 4
SK:0000209C ETAUODOT	Go Till There
612 ast = func4(ast);	遥 List There
SR:000020A0 E59F116C	ldr r1,0x21 Assemble here
	Modify here
B::	Patch here
emulate trigger devices trace Data	Var List other previous
SR:00002068 \\armle\arm\main+0x70	system ready MIX UP

For the following example, let's assume we have the following call hierarchy: **main()** calls **func2()** and **func2()** calls **func1()**:

IB:: List Step ♥ Over ↓ Next ♥ R line source	eturn C Up
<pre>main() 586 { int j; char * p; 590 vtripplearray[0][0] vtripplearray[1][0] vtripplearray[0][1] 592 vtripplearray[0][1] 593 vtripplearray[0][0] 595 func2(); 597 func2a(); </pre>	<pre> I [B::List] I Step Nover ↓ Next ✔ Return Up Iine source I () I ()</pre>
	static int fstatic2; 166 autovar = regvar = f 169 func1(&autovar); 171 func1(&fstatic); 175 176 177 func1(&fstatic); 178 179 171 <

Choose Show Stack in the Var menu. The Frame.view window displays the call hierarchy.

- The /Locals option shows the local variables of each function.
- The /Caller option shows a few source code lines to indicate where the function was called.

This screenshot corresponds to the calling hierarchy shown above.

TRACE32 ARM SIMULATOR
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
N N + + +
Breakpoint
Image: Show Stack Image: Show Stack Image: Show Current Vars -001 Image: Show Current Vars <
-002 main() . j = 0 . P = 0x0
end of frame
B:: FRAME.VIEW /LOCALS /CALLER
[ok] NoVar Args Locals NoCaller Caller LIMIT TASK CORE REGSET previous SR:00001054 \\armle\arm\func1 stopped HLL UP

Video tutorials about breakpoints in TRACE32 are available here: support.lauterbach.com/kb/articles/using-breakpoints-in-trace32

Setting Breakpoints

Let's set a breakpoint to the instruction prime = i + i + 3 and the instruction k + prime

- 1. Double-click a code line to set a program breakpoint.
- 2. Make sure to click the white space in the code line, and not the code literal.

All code lines with a program breakpoint are marked with a red vertical bar.



To set a breakpoint to an instruction that is not in the focus of the current source listing

 Choose Var menu > Show Function. The sYmbol.Browse.Function window opens.

A TRACE32 ARM SIMULATOR		- • ×
File Edit View Var Break Run CPU	Misc Trace Perf Cov Window Help	
▶ ▶ ↓ ↓ ✔ 🐼 Watch	🌚 📰 🎟 😖 🐼 🚳 🕲 🚣 🔢	
God View		
Q Data View		
💕 Breakpoint		
Show Function	B::symbol.browse.Function */Click Data.List */Delete	
🐯 Show Watch	*** 1 Type: F	unctions 🔻
🚱 Show Locals	symbol type address	
Show Stack	func/ (double ()) R:000013C8000014 func8 (void ()) R:00001454000016	53 A
Show Current Vars	func9 (static int * ()) R:000016F0000017	83
sormat	sieve ((int ()) R:00002228000022	C7 -
		the state

 Select the function you are interested in e.g. sieve.
 The List window opens, displaying this function. This window is now fixed to the start address of the function sieve and does not move with the program counter cursor.

Listing all Breakpoints

 Choose Break menu > List to list all breakpoints. The Break.List window opens, providing an overview of the set breakpoints.

Break	Run	CPU	Misc	Trace	Probe	Perf	Cov	Window	He	lp						
😻 <u>S</u> et	t			1	010 101		* 6	8 🗟 🤨	1	B						
Co	t nfigura	tion														
💥 De	lete <u>A</u> ll				😕 Bi:Bi	eak.Lis	t /								- 0	x
- T - Irig	gger Bu	s			Sett add	p X	🗶 Del	ete All C	Disa	able All 🔘	Enable All	⊗ Init	😤 Store	😤 Load	😻 Set	
⊲ r <u>O</u> n	Chip Tr	igger				T:0 T:0	00016	50 Progr 6C Progr	am am	SOFT SOFT	V V	ø sieve ø sieve	\11 \20			^
Ing	gger Ke	set	-				-	<								>
				L												
							1	B		C						
						A		Б		C		U				

- A Address of the breakpoint.
- B Breakpoint type.
- C Breakpoint method: SOFTware, ONCHIP or DISABLED.
- **D** Symbolic address of the breakpoint. Example:
 - **sieve**\11 means source code line 11 in function **sieve**.
- 2. On the toolbar, click **•** Go to start the program execution.
- 3. When the program execution stops at a breakpoint, it is highlighted in the **Break.List** window.

📵 B::Break.l	List								×
Setup	💥 Delete Al	O Disable All	Enable	All	🛇 Init	😤 Store	😤 Load	😂 Set	
address	ty t	/pe m	ethod						
T:	00001650 Pi	ogram S	OFT	V Ø	sieve\	11			~
T:	0000166C Pr	rogram S	OFT N	10	sieve\	20			
									~
	<								>
	_								

You can set a breakpoint that stops the program execution at a read or write access to a memory location (e.g. global variable). To set a breakpoint on the array **flags** for instance, do a right mouse click on the array name in the **List** window then select **Breakpoints > Write**.



🕲 B::Break.List					- • •
X Delete All O Disable All Enable	All 🛇 Init	A Method	Store	🔀 Load	💕 Set
address	type	method			
C:2000550020005512	Write	ONCHIP	V 🖉	flags	*
					-
	•				ta ≉

Video tutorials about variable display in TRACE32 are available here: support.lauterbach.com/kb/articles/variable-logging-and-monitoring-in-trace32

Displaying Variables

Let's display the variables **flags**, **def**, and **ast**.

Choose Var menu > Watch...
 The Var.AddWatch window opens, displaying the variables known to the symbol database.



Double-click the variable flags.
 The Var.Watch window opens, displaying the selected variable.

Í	<u>ಟ</u>	j E	3::Va	ar.V	Vato	ch																E	-		2	٢.
l												- 6	4]	6	' Wa	tch		6~1	/iew			*			
	÷	f	lag	gs	=	(1,	1,	1,	0,	1,	1,	0,	1,	1,	0,	1,	1,	0,	1,	1,	0,	1,	1,	0)		*
İ																										
	•																								Þ	H

3. Alternative steps:

- In the Var.Watch window, click & Watch, and then double-click the variables def and ast to add them to the Var.Watch window.

🐼 B::Var.Watch	×
▼ 🔬 🖧 Watch 🎸 View 💥	
\blacksquare ast = (word = 0x0, count = 12346, iert = 0x03C, right = 0x0, field	L 🔺
	Ŧ

- From a List window, drag and drop any variable you want into the Var.Watch window.
- In a List window, right-click any variable, and then select Add to Watch window from the context menu.
- If you want to display a more complex structure or an array in a separate window, choose **Var** menu > **View**.

Displaying Variables of the Current Program Context

1. Set the program counter (PC) to **sieve()** by typing at the TRACE32 command line:

Register.Set PC sieve ; The command short form is: r.s pc sieve

2. Choose Var menu > Show Current Vars.

The Var.REF window opens, displaying all variables accessed by the current program context.



3. Click Step on the TRACE32 toolbar to execute a few single steps. The **Var.REF** window is updated automatically.

Using the Symbol Browser

The symbol browser provides an overview of the variables, functions, and modules currently stored in the symbol database.

- Choose Var menu > Watch... The Var.AddWatch window lets you browse through the contents of the symbol database. Global variables are displayed in black and functions in gray. By double-clicking a function, its local variables are displayed.
- 2. In the **Var.AddWatch** window, double-click **func2**.

B::Var.AddWatch *			• ×	
//*/*/*	1 Type:	Variables 🗸 🗌 Source		
symbol	type	address		
def	(struct_abc)	D:000068180	000681F 🔨	
defaultstring	(char [5])	D:000067200	0006724	
encode	(enum enumtyn)	D:00007E940	0007594	
exc isr	(enum enumeyp)	D.0000/E340	0007234	
exc_snoop				
flags	(char [19])	D:00007E980	0007EAA	
func2		1		I
func21			B::List f	unc2]
func22				
	1		► Step	📕 Over 🛛 🕂 Next 🖌 🥐 Return 🖉 Up
			addr/lin	e source
			15	5 1
🖌 B. Var AddWatch *			15	6 (*intptr)++;
			15	/ }
**\func2*	1. " Type:			11.6 20
			1.0	Void Tunc2()
symbol type	address (F_0004) (F_0001)		10	
fatatic (auto Int)	(F=0004)==(F=0001)	E62E		int autovar;
fetatic? (static int)	D:000056400000	505F	_	register int regvar;
rogyar (register in	+) 0.000030400000	504.5		static int fstatic?
regval (register in	KZ KZ			scatte int Istaticz;
			16	6 autovar - regvar - fstatic:
Ц			10	aucovai – regvai – rscacic,
			1	

To format the display of variables - global settings:

- 1. Choose Var menu > Format.
- 2. In the **SETUP.Var** window, make your settings. **Decimal** and **Hex** are useful global settings. TRACE32 applies your settings to all **Var.view** windows that you open *afterwards*.

To format the display of an individual variable:

- 1. At the command line, type: **var.view ast** (The variable **ast** is included in this demo.)
- 2. In the Var.view window, right-click ast, and then click Format. The Change Variable Format dialog opens.
- 3. Select the **Type** check box to display the variable **ast** with the complete type information.
- 4. Click Apply. The format of ast in the Var.view window is updated immediately.



5. For more complex variable select **TREE** in the **Change Variable Format** dialog box.



Modifying Variables

1. Double-click the variable value to modify the value. The **Var.set** command will be displayed in the command line. The short form of the command is \mathbf{v} or \mathbf{v}



2. Enter the new value directly after the equal sign and confirm with [ok].

Memory

Displaying Memory

- 1. To display a memory dump in a **Data.dump** window, do one of the following:
 - Choose View menu > Dump,
 - or click I Memory Dump on the toolbar,
 - or, at the TRACE32 command line, type: Data.dump
 You can also specify an address or symbol directly, e.g.: Data.dump flags
- 2. In the Data.dump dialog, enter the data item, e.g. flags
 - Alternatively click 🗾 to browse through the symbol database.
- 3. In the Browse Symbols window, double-click the symbol flags to select it, and then click OK.



In the following screenshot, the **Data.dump** window is called via the TRACE32 command line.



There are different ways to define an address range:

<start_address>--<end_address> (SD is an access class)

Data.dump SD:0x5530--SD:0x554F

<start_address>++<offset>

Data.dump cstr1++0x1f /Byte ;start at cstr1 plus the next 0x1f bytes

Modifying Memory

In a Data.dump window, double-click the value you want to modify.
 A Data.Set command for the selected address is displayed in the command line. The short form of the command is p.s or d.s

TRAC	CE32 ARM
	H ↓ ✔ ⊄ ▶ II 図 ② K? ◎ 圖 幽 III 🖾 鐍 鹵 🧶 🧎
	B::Data.dump cstrl /Byte addr ess 0 1 2 3 4 5 7 8 9 A B C D E F 01234567834850EF addr ess 0 1 2 3 4 5 7 8 9 A B C D E F 01234567834850EF F addr ess 0 1 2 3 4 5 7 4 72 69 65 67 31 constant 51 pcl
	SD:00005510 SD:00005510 SD:00005520 2A 15 44 4E 00 40 00 00 00 10 C9 00 00 00 00 00 00 00 00 00 00 00 00 00
B::D.5	S SD:0x5500 %LE

2. Enter the new value directly after **%LE**, and then confirm with **[ok]**. (**%LE** stands for Little Endian).

To save the window configuration for you next debug session use *Store Windows…* from the *Window* menu. *Store Windows…* generates a PRACTICE file, that includes all commands to reactivate your complete window configuration automatically.

app	Wi	ndow Help									
3 <u>7</u> 1		Cascade Tile Horizontally		A B::B::STOre * Win							×
k [Tile Vertically Arrange Icons		← → • ↑ <mark> </mark>	« T3	32	> scripts 🗸 🗸	Sear	ch scripts		Q
		Create Duplicate window	-	Organise 🔻 Ne	w fold	ler					?
_	×	Clear Windows on Page		🧊 3D Objects	^		Name		Date modified		Туре
poi		Chees Windows		Desktop			🛃 win_layout.cmm		23/01/2020 17:	05	CMM
		Load Windows		Documents							
		B::List.auto		Downloads Music							
		B::Register.view /SpotLight		Pictures							
ocat	e *	/		Videos							
				🏪 Local Disk (C:)	<					>
				File name:	win_	layo	put.cmm				~
				Save as type: Store Setting in PRACTICE Script (*.cmm)					~		
				∧ Hide Folders					Save	Cancel	

The saved window layout can be loaded again for the next debug session with the *Load Windows…* in the *Window* menu.

You can also add a call into your start-up file:

DO win_layout.cmm

The online help system consists of several documents. They are accessible as PDF-files directly from the TRACE32 software and can be found in the pdf directory.

A HELP	
? Contents 🛐 Index 🛱 Find 📴 Command Tree 🖡 Bookmarks	💾 Print
🕒 open all 🕒 close all 🕀 more 🕞 less 🗹 use filter: bdmarm;icrstr	n;
About the TRACE32 Online Help	
TRACE32 Glossary TRACE32 Debugger Getting Started TRACE32 Debugger Getting Started	
TRACE32 Documents TRACE32 Training	
TRACE32 Installation TRACE32 Technical Support	
TRACE32 Index TRACE32 Directory	
	~
	×

There are different ways to open the TRACE32 online help:

Help Topics button on the toolbar



• Help menu > Contents



HELP command in the command line



• Help button in the Welcome to TRACE32! dialog.

The help system is organized in a multilevel structured way. The screen below shows how to find this tutorial.

A HELP				- • ×
? Contents 📉 Index 🎒 Find	E: Command Tree	Bookmarks	💾 Print	
⊕ open all ⊖ close all ⊕ more	⊖ less 🗹 use f	ilter: bdmarm;icrstm;		
About the TRACE32 Online Help				
TRACE32 Glossary TRACE32 Debugger Getting Started				<u></u>
ICD Quick Installation Top				
⊞ I32Start ⊞ Establish Your Debug Session				
ICD Tutorial About the Tutorial				
■ Working with the Debugger				
■ Main Window of TRACE32 ■ TRACE32 Documents				
TRACE32 Training TRACE32 Installation				
■ TRACE32 Technical Support				
<pre>TRACE32 Index TRACE32 Directory</pre>				
				\sim
<				>

It is also possible to help for a single command. Enter the command into the command line, add a space and push F1.