

Training Basic Debugging

Release 09.2024



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Training	5
Debugger Training	5
Training Basic Debugging	1
System Concept	6
On-chip Debug Interface	7
Debug Features	7
TRACE32 Tools	8
On-chip Debug Interface plus On-chip Trace Buffer 1	0
On-chip Debug Interface plus Trace Port 1	2
NEXUS Interface 1	3
Starting a TRACE32 PowerView Instance1	4
Basic TRACE32 PowerView Parameters 1	4
Configuration File 1	4
Standard Parameters 1	5
Examples for Configuration Files 1	6
Additional Parameters 2	0
Application Properties (Windows only) 2	1
Configuration via T32Start (Windows only) 2	2
About TRACE32 2	3
Version Information 2	3
Prepare Full Information for a Support Email 2	4
TRACE32 PowerView	5
TRACE32 PowerView Components 2	5
Main Menu Bar and Accelerators 2	6
Main Tool Bar 2	8
Window Area 3	0
Command Line 3	3
Message Line 3	6
Softkeys 3	7
State Line 3	8
Registers	9
Core Registers 3	9
Display the Core Registers 3	9
Colored Display of Changed Registers 4	0

Modify the Contents of a Core Register	41
Special Function Register	42
Display the Special Function Registers	42
Details about a Single Special Function Register	45
Modify a Special Function Register	46
The PER Definition File	47
Memory Display and Modification	48
The Data.dump Window	49
Display the Memory Contents	49
Modify the Memory Contents	54
Run-time Memory Access	55
Colored Display of Changed Memory Contents	65
The List Window	66
Displays the Source Listing Around the PC	66
Displays the Source Listing of a Selected Function	67
Breakpoints	69
Breakpoint Implementations	69
Software Breakpoints in RAM	69
Software Breakpoints in FLASH	71
Onchip Breakpoints in NOR Flash	72
Onchip Breakpoints on Read/Write Accesses	75
Onchip Breakpoints by Processor Architecture	76
ETM Breakpoints for ARM or Cortex-A/-R	77
Breakpoint Types	80
Program Breakpoints	81
Read/Write Breakpoints	83
Breakpoint Handling	85
Breakpoint Setting at Run-time	85
Real-time Breakpoints vs. Intrusive Breakpoints	86
Break.Set Dialog Box	88
The HLL Check Box - Function Name	89
The HLL Check Box - Program Line Number	92
The HLL Check Box - Variable	94
The HLL Check Box - HLL Expression	96
Implementations	99
Actions	100
Options	104
DATA Breakpoints	108
Advanced Breakpoints	112
TASK-aware Breakpoints	113
Intrusive TASK-aware Breakpoint	113
Real-time TASK-aware Breakpoint	116

COUNTer	117
Software Counter	117
	117
On-chip Counter	120
CONDition	121
CMD	129
memory/register/var	132
Display a List of all Set Breakpoints	137
Delete Breakpoints	138
Enable/Disable Breakpoints	138
Store Breakpoint Settings	139
Debugging	140
Debugging of Optimized Code	140
Debugging of Optimized Code Basic Debug Control	140 143
Debugging of Optimized Code Basic Debug Control Sample-based Profiling	140 143 155
Debugging of Optimized Code Basic Debug Control Sample-based Profiling Program Counter Sampling	140 143 155 155
Debugging of Optimized Code Basic Debug Control Sample-based Profiling Program Counter Sampling Standard Procedure	140 143 155 155 156
Debugging of Optimized Code Basic Debug Control Sample-based Profiling Program Counter Sampling Standard Procedure Details	140 143 155 155 156 160
Debugging of Optimized Code Basic Debug Control Sample-based Profiling Program Counter Sampling Standard Procedure Details TASK Sampling	140 143 155 155 156 160 162

Version 05-Oct-2024

A single-core processor/multi-core chip can provide:

- An on-chip debug interface
- An on-chip debug interface plus an on-chip trace buffer
- An on-chip debug interface plus an off-chip trace port
- A NEXUS interface including an on-chip debug interface

Depending on the debug resources different debug features can be provided and different TRACE32 tools are offered.

The TRACE32 debugger allows you to test your embedded hardware and software by using the on-chip debug interface. The most common on-chip debug interface is JTAG.

A single on-chip debug interface can be used to debug all cores of a multi-core chip.

Debug Features

Depending on the processor architecture different debug features are available.

Debug features provided by all processor architectures:

- Read/write access to registers
- Read/write access to memories
- Start/stop of program execution

Debug features specific for a processor architecture:

- Number of on-chip breakpoints
- Read/write access to memory while the program execution is running
- Additional features as benchmark counters, triggers etc.

The TRACE32 debugger hardware always consists of:

- Universal debugger hardware
- Debug cable specific to the processor architecture

Debug Only Modules



Current module:

POWER DEBUG E40

Deprecated modules:

- POWER DEBUG INTERFACE / USB 3
- POWER DEBUG INTERFACE / USB 2



Current module:

• POWER DEBUG X50

Deprecated modules:

- POWER DEBUG PRO (USB 3 and 1 GBit Ethernet)
- POWER DEBUG II (USB 2 and 1 GBit Ethernet)
- POWER DEBUG / ETHERNET (USB 2 and 100 MBit Ethernet)

On-chip Debug Interface plus On-chip Trace Buffer

A number of single-core processors/multi-core chips offer in addition to the on-chip debug interface an onchip trace buffer.

On-chip Trace Features

The on-chip trace buffer can store information:

- On the executed instructions.
- On task/process switches.
- On load/store operations if supported by the on-chip trace generation hardware.

In order to analyze and display the trace information the debug cable needs to provide a **Trace License**. The Trace Licenses use the following name convention:

- <core>-TRACE e.g. ARM-TRACE
- or <core>-MCDS) e.g. TriCore-MCDS

He	lp
8	Contents
*	Index
Ë	Find
É	Tree
	TRACE32 PowerView User Manual
	Processor Architecture Manual
D	Debugger User Guide
F	MCDS User Guide
F.	Analyzer User Manual
D	Linux Awareness Manual
	Timing Analyzer User Manual
	Power Probe User Manual
G	Stimuli Generator User Manual
	Training Manuals
ñ	Demo Scripts
4	Welcome to TRACE32
	Lauterbach Homepage
	Support
A	About TRACE32

A B::VERSION	N		- • •
TRACE3	2 Powe	rView for ARM	^
MICROPRO	CESSOR D	EVELOPMENT SYSTEM	
Copyright (c) 1989-20	12 Lauterbach GmbH	
Software:		Interim Build (32-bit)	more
Software	Version:	S.2012.08.000038874	
Build:		38874.	08/2012
License:			more
Cable:	ARM	(Cortex ARM-TRACE	01/2014
Hardware:		PowerDebug-II via USB 2.0	more
Debug Ca	ble:	C12100165970 ARM Debug Cable V4d	
Environmen	it:	Windows 7	more
SYS:	C:\T32_A	ARM	
TMP:	C:\TMP		
CONFIG:	C:\TMP\a	andT32_1000004.t32	edit
		Close	

The display and the evaluation of the trace information is described in the following training manuals:

- "Training Arm CoreSight ETM Tracing" (training_arm_etm.pdf).
- "Training Cortex-M Tracing" (training_cortexm_etm.pdf).
- "Training AURIX Tracing" (training_aurix_trace.pdf).
- "Training Hexagon ETM Tracing" (training_hexagon_etm.pdf).
- "Training MPC5xxx/SPC5xx Nexus Tracing" (training_nexus_mpc5500.pdf).

A number of single-core processors/multi-core chips offer in addition to the on-chip debug interface a socalled trace port. The most common trace port is the TPIU for the ARM/Cortex architecture.

Off-chip Trace Features

The trace port exports in real-time trace information:

- On the executed instructions.
- On task/process switches.
- On load/store operations if supported by the on-chip trace generation logic.

The display and the evaluation of the trace information is described in the following training manuals:

- "Training Arm CoreSight ETM Tracing" (training_arm_etm.pdf)
- "Training Cortex-M Tracing" (training_cortexm_etm.pdf)
- "Training AURIX Tracing" (training_aurix_trace.pdf)
- "Training Hexagon ETM Tracing" (training_hexagon_etm.pdf)

NEXUS Interface

NEXUS is a standardized interface for on-chip debugging and real-time trace especially for the automotive industry.

NEXUS Features

Debug features provided by all single-core processors/multi-core chips:

- Read/write access to the registers
- Read/write access to all memories
- Start/stop of program execution
- Read/write access to memory while the program execution is running

Debug features specific for single-core processor/multi-core chip:

- Number of on-chip breakpoints
- Benchmark counters, triggers etc.

Trace features provided by all single-core processors/multi-core chips:

- Information on the executed instructions.
- Information on task/process switches.

Trace features specific for the single-core processor/multi-core chip:

• Information on load/store operations if supported by the trace generation logic.

The display and the evaluation of the trace information is described in "Training MPC5xxx/SPC5xx Nexus Tracing" (training_nexus_mpc5500.pdf).

Basic TRACE32 PowerView Parameters

This chapter describes the basic parameters required to start a TRACE32 PowerView instance.

The parameters are defined in the configuration file. By default the configuration file is named **config.t32**. It is located in the TRACE32 system directory (parameter **SYS**).

Configuration File

Open the file config.t32 from the system directory (default c:\T32\config.t32) with any ASCII editor.



The following rules apply to the configuration file:

- Parameters are defined paragraph by paragraph.
- The first line/headline defines the parameter type.
- Each parameter definition ends with an empty line.
- If no parameter is defined, the default parameter will be used.

Standard Parameters

Parameter	Syntax	Description
Host interface	PBI= <host_interface></host_interface>	Host interface type of TRACE32 tool hardware (USB or ethernet)
	PBI=ICD <host_interface></host_interface>	Full parameter syntax which is not in use.
Environment variables	OS= ID= <identifier> TMP=<temp_directory> SYS=<system_directory> HELP=<help_directory></help_directory></system_directory></temp_directory></identifier>	 (ID) Prefix for all files which are saved by the TRACE32 PowerView instance into the TMP directory (TMP) Temporary directory used by the TRACE32 PowerView instance (*) (SYS) System directory for all TRACE32 files (HELP) Directory for the TRACE32 help PDFs (**)
Printer definition	PRINTER=WINDOWS	All standard Windows printer can be used from TRACE32 PowerView
License file	LICENSE= <license_directory></license_directory>	Directory for the TRACE32 license file (not required for new tools)

	(*) In order to display source code information TRACE32 PowerView creates a copy of all loaded source files and saves them into the TMP directory.
$\overline{\mathbf{A}}$	(**) The TRACE32 online help is PDF-based.

Configuration File for USB

Single debugger hardware module connected via USB:

```
; Host interface
PBI=
USB
; Environment variables
OS=
ID=T32
                                 ; temporary directory for TRACE32
TMP=C:\temp
                                 ; system directory for TRACE32
SYS=C:\t32
HELP=C:\t32\pdf
                                 ; help directory for TRACE32
; Printer settings
PRINTER=WINDOWS
                                 ; all standard windows printer can be
                                 : used from the TRACE32 user interface
```

Multiple debugger hardware modules connected via USB:

```
; Host interface
PBI=
USB
                                 ; NODE name of TRACE32
NODE=training1
; Environment variables
OS=
ID=T32_training1
TMP=C:\temp
                                 ; temporary directory for TRACE32
SYS=C:\t32
                                 ; system directory for TRACE32
HELP=C:\t32\pdf
                                 ; help directory for TRACE32
; Printer settings
PRINTER=WINDOWS
                                 ; all standard windows printer can be
                                 : used from TRACE32 PowerView
```

Use the IFCONFIG command to assign a device name (NODE=) to a debugger hardware module. The manufacturing default device name is the serial number of the debugger hardware module:

- e.g. E18110012345 for a debugger hardware module with ethernet interface, such as PowerDebug PRO.
- e.g. C18110045678 for a debugger hardware module with USB interface only, such as PowerDebug USB 3.

	Control Control		
Veo Uscillator Frequency Counter Pulse Generator	ip address	- host ip address	Enter device name
Pulse Generator 2 Runtime Memory Map	ethernet address 00-C0-8A-80-57-98	host ethernet address	
 Flash Programming Choose Colors Interface Config Tools Japanese Menu 	device name MPC ethernet settings RARP BOOTP Ø DHCP (via device name) full duplex	statistics recv packets 11714. send packets 5219. kbytes 4369. collisions 0. retries 5. resyncs 0. errors 0. configuration: USB2 TEST	Save device name to debugger hardware module

IFCONFIG

Dialog to assign USB device name

Please be aware that USB device names are case-sensitive

TRACE32 allows to communicate with a POWER DEBUG INTERFACE USB from a remote PC. For an example, see "Example: Remote Control for POWER DEBUG INTERFACE / USB" in TRACE32 Installation Guide, page 46 (installation.pdf).

```
; Host interface
PBI=
NET
NODE=training1
; Environment variables
OS=
ID=T32
                                 ; temp directory for TRACE32
                                 ; system directory for TRACE32
SYS=C:\t32
HELP=C:\t32\pdf
                                 ; help directory for TRACE32
; Printer settings
PRINTER=WINDOWS
                                 ; all standard windows printer can be
                                 ; used from the TRACE32 user interface
```

Ethernet Configuration and Operation Profile

Frequency Counter	B::IFCONFIG		
Pulse Generator	ip address	– host ip address -	
Pulse Generator 2	10.2.0.208		
7 Runtime			
Memory Map	- ethernet address	– host ethernet ad	dress
Flash Programming	00-C0-8A-80-57-98		
Choose Colors			
Disterface Config	device name	statistics	
Dools	MPC	recv packets	11714.
Japanese Menu		send packets	5219.
	ethernet settings	kbytes	4369.
	RARP	collisions	0.
	ВООТР	retries	5.
	DHCP (via device name)	resyncs] 0.
	I full duplex	errors	0.
		configuration:	USB2
		TEST]
	Save to device	Close	

IFCONFIG

Dialog to display and change information for the Ethernet interface

Changing the font size can be helpful for a more comfortable display of TRACE32 windows.

```
; Screen settings
SCREEN=
FONT=SMALL
```

; Use small fonts

Display with normal font:

B::List		
Step 🕨 Ov	er 🛛 🛃 Diverge 🖉 Return 🖉 Up 📄 🕨 Go 🛛 🔢 Break 🛛 🎉 Mode	Find: demo.c
addr/line	code label mnemonic comment	
	char flags[SIZE+1];	A
	int sieve(void) /* sieve of erathostenes */	
410 NCD • 4 A 3 2 6 5 6 4	[1]	
NSK. HAJZOJOH	E3204030 STEVE: push {14-13,114}	
422 NSR:4A326568	<pre>for (i = 0 ; i <= SIZE ; flags[i++] = TRUE) ; E59FC17C</pre>	
	register int i, primz, k; int anzahl;	
420	anzahl = 0;	
NSR:4A32656C	E3A00000 mov r0,#0x0 ; anzah1,#0	
(22)		
422 NSR:44326570	1 = 0; 1 = 0; 1 = 5LZE; Tlags[1++] = IRUE); E1A01000 cpv r1 r0 i anzah]	-
ASIC TASEOSTO		

Display with small font:

📰 B::List										×
Step	• Over	🛃 Diverg	e 🖌 🖋 Retur	n 🙋 Up	Go	Break	Mode	Find:	demo.c	_
addr/line	code	label	mnemonic	c	omment					_
416 NSR:4A326564	char flag int sieve { E92D4030	s[SIZE+1]; (void) sieve:	/* sie push {r4-	ve of erathoste -r5.r14}	nes */					^
422 NSR:4A326568	E59FC17C	or (i = 0 egister in nt anzahl;	; i <= SIZE ldr r12 t i, primz,	; flags[i++] ,0x4A3266EC <;	= TRUE) ;					
420 NSR:4A32656C	E3A00000	nzahl = 0;	mov r0,/	≠0x0 ;	anzah1,#0					
422 NSR:4A326570	E1A01000	or (i = 0	; i <= SIZE cpy r1,	; flags[i++] ~0 ;	= TRUE) ; i,anzahl					
422 NSR:4A326574 NSR:4A326578 NSR:4A32657C	E3A02001 E7CC2001 E2811001	or (i = 0	; i <= SIZE mov r2,/ strb r2, add r1,	; flags[i++] ^{#0x1} ; [r12,+r1] r1,#0x1 ;	= TRUE) ; r2,#1 i,i,#1					
-	•								Þ	

The Properties window allows you to configure some basic settings for the TRACE32 software.

To open the Properties window, right-click the desired TRACE32 icon in the Windows Start menu.

Details	Novell Version	Previous	s Versions	
General	Shortcut Com	patibility	Security	
t t	32marm.exe			
Target type:	Application			
Target locatio	n: windows			
Target:	vindows\t32mam.exe	c J:\AND\POI	D\config.t32	— Configuration Fi
Start in:				Moulting Directo
Statt III.	0.000000			- working Directo
Shortcut key:	None			
Run:	Maximized			— Window Size
Comment:	Normal window			
	Maximized			
Open File	Location Change Ico	on Ad	vanced	

Definition of the Configuration File

By default the configuration file **config.t32** in the TRACE32 system directory (parameter **SYS**) is used. The option **-c** allows you to define your own location and name for the configuration file.

C:\T32_ARM\bin\windows\t32marm.exe -c j:\and\config.t32

Definition of a Working Directory

After its start TRACE32 PowerView is using the specified working directory. It is recommended not to work in the system directory.

```
PWD
```

TRACE32 command to display the current working directory

Definition of the Window Size for TRACE32 PowerView

You can choose between Normal window, Minimized and Maximized.

The basic parameters can also be set up in an intuitive way via T32Start.

A detailed online help for t32start.exe is available via the Help button or in "T32Start" (app_t32start.pdf).



If you want to contact your local Lauterbach support, it might be helpful to provide some basis information about your TRACE32 tool.

Version Information

lelp				
Contents				
Index				
) Find		A PUVERSION		
I ree		D.VERSION		
TRACE32 PowerView User Manual		TRACERS		
Processor Architecture Manual		TRACE32 P	owerview for ARM	
Debugger User Guide		MICROPROCESS		
MCDS User Guide		MICROPROCESS	OR DEVELOPMENT SYSTEM	
Analyzer User Manual		Copyright (c) 19	89-2014 Lauterbach GmbH	
Timing Analyzer User Manual		Software	Intorim Build (64-bit)	moro
Power Probe User Manual		Software.		more
Stimuli Generator User Manual		Builds	52010	05/2014
Training Manuals	•	Build.	55610.	05/2014
Demo Scripts		License:		more
Welcome to TRACE32		Cable: ARM	(Cortex ARM-TRACE)	12/2014
Lauterbach Homepage				
Support	•	Hardware:	PowerDebug USB 3.0	more
About TRACE32		Debug Cable:	C12100165970 ARM Debug	Cable V4d
		Environment:	Windows 7	more
		SYS: C:\T	T32_ARM	
		TMP: C:\l	Jsers\amartin\AppData\Local\Temp	
		CONFIG: C:\T	T32_ARM\config.t32	edit
		L		
			Close	

The VERSION window informs you about:

- 1. The version of the TRACE32 software.
- 2. The debug licenses programmed into the debug cable and the expiration date of your software warranty respectively the expiration date of your software warranty.
- 3. The serial number of the debug cable.

VERSION.view	Display the VERSION window.
VERSION.HARDWARE	Display more details about the TRACE32 hardware modules.
VERSION.SOFTWARE	Display more details about the TRACE32 software.

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

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

Help					
? Contents					
Index					
ji Eind					
E Tree					
TRACE32 PowerView User Manual	_				
n Processor Architecture Manual					
📷 Debugger <u>U</u> ser Guide					
🔂 MCDS User Guide					
💼 Analyzer User Manual					
📾 Timing Analyzer User Manual					
Power Probe User Manual					
🕞 Stimuli Generator User Manual					
📷 Training Manuals	•				
🛉 Demo Scripts	_				
A Welcome to TRACE32					
Lauterbach <u>H</u> omepage					
<u>S</u> upport	🖌 🖉 Systeminfo				
👠 About TRACE32	<u> O</u> nline Support				
	Contact Lauterback	n			
	🖉 Generate TR	ACE32 Support Information			
	Press th	e following button to get help on l	how to generate	Support Information:	
	Company:	Lauterbach	Department:	Training	
	Prefix:				
	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			
	Target CPU:	CortexA9			
	Hostsystem:	PC Windows 7 🔹			
	Compiler:	ARM			
	RealtimeOS:	None		Safe	Mode:
	L				
		Generate Support Information:	Save to Clip	oboard Save	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.

TRACE32 PowerView Components





The main menu bar provides all important TRACE32 functions sorted by groups.

For often used commands accelerators are defined.



A user specific menu can be defined very easily:

MENU.AddMenu <name> <command/></name>	Add a user menu
MENU.RESet	Reset menu to default
; user menu MENU AddMenu "Set PC to main"	"Pegister Set DC main"
MENU.Addmenia Set FC CO main	Register.set rt main
; user menu with accelerator MENU.AddMenu "Set PC to main,	ALT+F10" "Register.Set PC main"

TRACE32 PowerView	l l
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC5XXX User Window Help	
N ⊨ 🚣 ↓ 7 ℃ ▶ II 22 ? № 🗐 🕮 🔲 🐼 🐼 🗟 Set PC to main ALT+F10	
B::Data.List	
H Step H Over L Diverge ✓ Return C Up ► Go II Break	
main()	
int j; char * p;	
while (TRUE)	
592 start:	📏 User Menu
594 vtripplearray[1][0][0] = 2;	
355 vtrippiearray[0][1][0] = 5;	
B::MENU.AddMenu "Set PC to main, ALT+F10" "Register.Set pc main"	
	6
SF:40001138 \\diabc_int\diabc\main stopped at breakpoint HLL UP	



The main tool bar provides fast access to often used commands.

The user can add his own buttons very easily:

MENU.AddTool <tooltip_text> <tool_image> <command/></tool_image></tooltip_text>		Add a button to the toolbar
MENU.RESet		Reset menu to default
; <tooltip text=""> here: ; <tool image=""> here: ; <command/> here:</tool></tooltip>	Set PC to main button with capital Register.Set PC main	letters PM in black

TRACE32 PowerView	1
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC5XXX User Window Help	
▼ ▼ ▼ ↑ ↑ ↑ ↓ 2 2 2 2 2 2	
E: Data.List	User specific
H Step Nover L Diverge ✔ Return Up Go II Break	button
main()	
586 { int j;	
char * p;	
White (TRUE)	
592 start: 593 vtripplearray[0][0][0] = 1:	
vtripplearray[1][0][0] = 2;	
595 Vtr1pp1earray[0][1][0] = 5;	
B::MENU.AddTool "Set PC to main" "PM,X" "Register.Set PC main"	
[ok] previous	
SF:40001138 \\diabc_int\diabc\main stopped at breakpoint HLL UP	

Information on the <tool image> can be found in Help -> Contents

TRACE32 Documents -> IDE User Interface -> PowerView Command Reference -> MENU -> Programming Commands -> TOOLITEM.

All predefined TRACE32 icons can be inspected as follows:



TRACE32 PowerView for ARM			
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help			
▶ ♥ ↑ ੯ Ϛ ▶ Ⅱ 窓 & ₺5 ◎ 当 照 ■ 🕸 🕾 🕾 🔞 🍹 🏷 ヵ			
🔺 predefined TRACE32 icons - mouse click for displaying the icon name inside the message line 📃 💼 💼			
▲ ♀ ∓ ■ − ◊ ↓ ⊻ ★ → Ċ Ⅲ ↔ ೫ ⋭ ■ → ⊥ ⋫ ∓ − ⊌ ∰ ∷ ⋫			
፵▨੮ਸ਼ੋ ■ 」 ≝ ▼ ● 拳 ↓ ৫ ⊚ 44 ⋑ ⋑ Ж ≫) @ ∽ ♠ ∦ ⊗ 67 ि –			
V 📕 X 🗢 🗆 🚴 🐇 V ≑ ᆂ 🗸 📢 🖬 🎬 + I 🔶 🕄 🐴 🔺 🔺 🍪 🛀 💷			
A ■ Ø = ■ Ø 4 ▼ : ▼ Ø ? Ø ? &] I ◆ 6 H ■ ◆ A Ø Q M			
j) ﷺ 🖗 ■ 🖡 🖏 ▼ 🕂 🔆 ± ♦ ± 🕸 🖉 📕 🏾 + 💀 ⊨ 🖕 ▲ 🖓 🕵 Ξ			
図 = ↓ ▼ :::: ▼ :::: ▼ ::: ♥ :: ▼ ::: *:: ▼ ::: ▼ ::: ▼ ::: *:: *			
_ ∞ ☆ ॥ ■ 🔏 🥘 Ⴀ 🎾 🕲 주 🛨 🗷 🗸 🗶 🔵 🗇 🗇 🗠 🖄 🖼 🔶 🚽			
B::			
[:colorpurple]			
emulate trigger devices trace Data Var List other previous			
ST:00001BA4 \\thumble\arm\sieve+0x28 stopped MIX UP			

Or by following TRACE32 command:

ChDir.DO ~~/demo/menu/internal_icons.cmm

The predefined icons can easily be used to create new icons.

; overprint the icon **colorpurple** with the character **v** in **W**hite color Menu.AddTool "Set PC to main" "**v**,**W**,**colorpurple**" "Register.Set PC main"



For more complex changes to the main tool bar refer to **"Training Menu Programming"** (training_menu.pdf).

Videos about the menu programming can be found here: support.lauterbach.com/kb/articles/trace32-user-interface-customization

Save Page Layout

No information about the window layout is saved when you exit TRACE32 PowerView. To save the window layout use the *Store Windows to ...* command in the *Window* menu.



Script example:





Run the script to reactivate the stored window-configuration

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC5XXX Window Help	
N M M I + 4 C ▶ II ⊠ ? №	
N Step Nover L Diverge ✓ Return C Up Co II Break ∭Mode	
main()	The window header
586 { int j;	displays the command
char * p;	which was executed to
while (TRUE)	open the window
592 start:	
593 vtripplearray[0][0] = 1; 594 vtripplearray[1][0][0] = 2:	
595 vtripplearray[0][1][0] = 3;	
596 Vtripplearray[0][0][1] = 4;	
B:: B::List /SOrder	
[ok] Mark Track TOrder SOrder other previous	
SF:40001138 \\diabc_int\diabc\main stopped at breakpoint HLL UP	

By clicking with the right mouse button to the window header, the command which was executed to open the window is re-displayed in the command line and can be modified there



Command Structure

Device prompt: the default device prompt is **B::**. It stands for BDM which was the first on-chip debug interface supported by Lauterbach.

A TRACE32 command has the following structure:



Command Examples

Data	Command group to display, modify memory
Data.dump	Displays a hex dump
Data.Set	Modify memory
Data.LOAD.auto	Loads code to the target memory

Break	Command group to set, list, delete breakpoints
Break.Set	Sets a breakpoint
Break.List	Lists all set breakpoint
Break.Delete	Deletes a breakpoint

Each command can be abbreviated. The significant letters are always written in upper case letters.

Examples for the parameter syntax and the use of options will be presented throughout this training.



Enter the command to the command line. Add one blank. Push F1 to get the online help for the specified command.



- Message line for system and error messages
- Message Area window for the display of the last system and error messages
The softkey line allows to enter a specific command step by step. Here an example:

Select the command group, here Data.

B::						
trigger devices	trace	Data	Var	List	PERF	SYStem

Select the subcommand, here dump.

B::DATA.							
1							
[ok]	dump	View	Print	List	Set	Assemble	PROGRAM

Angle brackets request an entry from the user, here e.g. the entry of a <range> or an <address>.

B::DATA.DUMP
<u> </u>
[ok] <range> <address> options</address></range>

The display of the hex. dump can be adjusted to your needs by an option.

B::DATA.DUMP 0x10000x1	fff		
[ok] options			

Select the option formats to get a list of all format options.



Select a format option, here Byte.

B:: DATA.DUMP 0x10000x1fff /		
[ok] NoHex Decimal DecimalU	Hex Byte	Word Long

The command is complete now.

B::DATA	A.DUMP 0x1000-	-0x1fff /BYTE		
[ok]	options			



The Cursor field of the state line provides:

- Boot information (Booting ..., Initializing ... etc.).
- Information on the item selected by one of the TRACE32 PowerView cursors.

The **Debug** field of the state line provides:

- Information on the debug communication (system down, system ready etc.)
- Information on the state of the debugger (running, stopped, stopped at breakpoint etc.)

The **Mode** field of the state line indicates the debug mode. The debug mode defines how source code information is displayed.

- Asm = assembler code
- HII = programming language code/high level language
- Mix = a mixture of both

It also defines how single stepping is performed (assembler line-wise or programming language line-wise).

B::			
emulate trigger devices trace	Data Var List	other	previous
SF:40001138 \\diabc_int\diabc\main	stopped at breakpoint		HLL UP Mode
			Asm
			Mix
			✓ HII

The debug mode can be changed by using the **Mode** pull-down.

Core Registers

Display the Core Registers



Register.view

The option /SpotLight advises TRACE32 PowerView to mark changes.

```
Register.view /SpotLight ; The registers changed by the last
; step are marked in dark red.
; The registers changed by the
; step before the last step are
; marked a little bit lighter.
; This works up to a level of 4.
```

🗾 В::	Register	.view /SpotLigi	nt		[- 0	*
N _	R0	0	R8	4A326DB4	S	Stack	
Z _	R1	0	R9	0			
C _	R2	0	R10	3039			
V _	R3	0	R11	0			
Q _	R4	4A326D94	R12	0			
	R5	0	R13	4A326FF0			
0 _	R6	0	R14	00314D70			
1 _	R7	0	PC	4A326614			
2 _	SPSR	0	CPSR	01D3			-
•							<u>ار ا</u>

Establish /SpotLight as default setting



SETUP.Var %SpotLight	Establish the option SpotLight as default setting for - all Variable windows - Register window - PERipheral window - the HLL Stack Frame - Data.dump window

Modify the Contents of a Core Register



Register.Set <register> <value> Modify

Modify register

Display the Special Function Registers

TRACE32 supports a free configurable window to display/manipulate configuration registers and the on-chip peripheral registers at a logical level. Predefined peripheral files are available for most standard processors/chips.

Tree Display

The individual configuration registers/on-chip peripherals are organized by TRACE32 PowerView in a tree structure. On demand, details about a selected register can be displayed.

Change Frame CPU Registers Perpherals System Settings System Settings System Settings Set Clock Frequency Standalone Reset In Target Reset Reset CPU Registers	CPU Misc Trace Probe	e Perf									
CPU Registers Peripherals System Settings System Settings System Settings Standalone Reset In Target Reset Reset CPU Registers B::PER Core_Registers B::PER Core_Registers B::PER Core_Registers B::PER Core_Registers B::PER Core_Registers D::D_Registers Core_Registers D::D_Registers	Change Frame	•									
Periphetals Exception Control System Settings Clock Frequency Set Clock Frequency Standalone Reset In Target Reset Reset CPU Registers B:PER Core Registers Core Registers Core Registers Core Registers D Rev D D Rev D D D D D D D D D D D D D D D D D	CPU Registers										
Exception Control System Settings Clock Frequency Set Clock Frequency Standalone Reset In Target Reset Reset CPU Registers © Core Registers © CORE_CM2 @ CAM_PRM © INSTR_PRM © Core Registers © ID Registers I ID Registers I ID Registers © ID Registers	Peripherals										
System Settings Clock Frequency Standalone Reset In Target Reset Reset CPU Registers © Core Registers © ID Registers	🞾 Exception Control										
Set Clock Frequency Set Clock Frequency Standalone Reset In Target Reset Reset CPU Registers © Core Registers © PRCM © CORE_CM2 © CORE_CM2 © Core Registers © ID Registers MIDR 411FC092 IMPL 41 VAR 1 ARMV7 PART 0C09 REV 2	System Settings										
Set Clock Frequency Standalone Reset In Target Reset Reset CPU Registers © Core Registers © PRM © CORE_CM2 © CAM_PRM © CORE_CM2 © Core Registers © D Registers © ID Registers MIDR 411FC092 IMPL 41 VAR 1 ARMv7	Clock Frequency										
Standalone Reset In Target Reset Reset CPU Registers	Set Clock Frequency										
In Target Reset Reset CPU Registers	Standalone Reset										
Reset CPU Registers	In Target Reset										
B:PER Core Registers PRCM CORE_CM2 GORE_CM2 GORE_CM2 GORE_CM2 GORE_CM2 GORE_CM2 GORE Registers GORE R	Reset CPU Registers										
Core Registers PRCM CORE_CM2 BCAM_PRM Core Registers Core Registers ID Registers MIDR 411FC092 IMPL 41 VAR 1 ARMv7 PART 0C09 REV 2	S::PER							×			
PRCM BCORE_CM2 CAM_PRM Core Registers D Registers MIDR 411FC092 IMPL 41 VAR 1 ARCH ARMv7 PART 0C09 REV 2	⊞ <u>Core Registers</u>										
© <u>CORE_CM2</u> © <u>CAM_PRM</u> © <u>INSTR_PRM</u> © <u>Core Registers</u> © <u>ID Registers</u> MIDR <u>411FC092</u> IMPL <u>41</u> VAR <u>1</u> ARCH ARMv7 PART <u>0C09</u> REV <u>2</u>	■ PRCM										
B CAM_PRM ■ INSTR_PRM ■ <u>Core Registers</u> ■ <u>ID Registers</u> MIDR <u>411FC092</u> IMPL <u>41</u> VAR <u>1</u> ARCH <u>ARMv7</u> PART <u>0C09</u> REV <u>2</u>	€ CORE_CM2							_			
■ <u>INSTR_PRM</u> ■ <u>INSTR_PRM</u> ■ <u>Core Registers</u> ■ <u>ID Registers</u>	CAM_PRM	-	0								~
Core Registers ID Registers ID Registers MIDR 411FC092 IMPL 41 VAR 1 ARCH ARMv7 PART 0C09 REV 2	INSTR_PRM		DIIPEN								
□ ID Registers MIDR 411FC092 IMPL 41 VAR 1 ARCH ARMv7 PART 0C09 REV 2	< III		ore Regi	isters							-
MIDR 411FC092 IMPL 41 VAR 1 ARCH ARMv7 PART 0C09 REV 2		G	ID Regi	isters							_
			MIDR	411FC092	IMPL PART	41 0C09	VAR REV	1 2	ARCH	ARM∨7	
CTR 83338003 FORMAT ARMV7 CWG 3			CTR	83338003	FORMAT	ARM∨7	CWG	3 8 words			
L1POLICY Virtual IMINLINE 8 words					L1POLICY	Virtual	IMINLINE	8 words			
TLBTR 00000000 TLSIZE 00 DLSIZE 04 TLB_size 128			TLBTR	000000000000000000000000000000000000000	ILSIZE	00	DLSIZE	04	TLB_size	128	
MPIDR 80000000 U Multiprocessor ClusterID 0 CPUID 0 🔻			MPIDR	80000000	nu U	Multiprocessor	ClusterID	0	CPUID	0	-
		•		III		-				I	► a



Full Display

Sometimes it might be useful to expand the tree structure from the start.

B::PER B::Core Real Show Hide Show all Hide all B: INSTR_ DSP_CM	isters						Use the – select S	right mouse ar Show all	۱d
•	S::PER								3
	E Core Regi	sters							
	E TD Regi	stors							
	MIDR	411FC092	IMPL	41	VAR	1	ARCH	ARM∨7	
	CTR	83338003	FORMAT ERG	ARMV7 3	CWG DMINLINE	2 3 8 words			
	TCMTR TLBTR	00000000 00000402	L1POLICY ILSIZE	Virtual 00	IMINLINE DLSIZE	8 words 04	TLB_size	128	
	MPIDR MMFR0	80000000 00100103	nU U FCSE	Unified Multiprocessor Not supported	ClusterID ACR	0 Supported	CPUID TCM	0 Not supported	Ŧ
	•	III						•	

Commands:

```
        PER.view <filename> [<tree_item>]
        Display the configuration registers/on-chip peripherals
```

```
; Display all functional units in expanded mode
; , advises TRACE32 PowerView to use the default peripheral file
; * stands for all <tree-items>
PER.view , "*"
```

; Display the functional unit "ID Registers" within "Core Registers" ; in expanded mode

PER.view , "Core Registers, ID Registers"

-	B::PER.view ,	"Core Registers	,ID Registers"					
	Core Regi	sters						
	ID Regi	sters						
	MIDR	411FC093	IMPL	41	VAR	1	ARCH	ARM∨7
			PART	0C09	REV	3		
	CTR	83338003	FORMAT	ARM∨7	CWG	3		
			ERG	3	DMINLINE	8 words		
			L1POLICY	Virtual	IMINLINE	8 words		
	TCMTR	00000000						
	TLBTR	00000402	ILSIZE	00	DLSIZE	04	TLB size	128
			nU	Unified				
	MPIDR	80000000	U	Multiprocessor	ClusterID	0	CPUID	0
	MMERO	00100103	ECSE	Not supported	ACR	Supported	TCM	Not supported
			055	Not supported	CC CPUA	Supported	PMSA	Not supported -
1		111						
1.								

; Display the functional unit "DMA_Channel_0" within "sDMA_Module,sDMA"

; in expanded mode

PER.view , "sDMA_Module,sDMA,DMA_Channel_0"

B::PER.view , "sDMA_Module,sDMA,DMA_Channe	_0"		
🖻 sDMA_Module			
E <u>SDMA</u>			
E DWA Chappel 0			
DMA4 CCDN1 0 000083E8	CURRENT DESCRIPTOR NBR	83E8	
DMA4_CCENi_0 00550C3A	CURRENT_ELMNT_NBR	550C3A	
DMA4_CCFNi_0 0000F0C6	CURRENT_FRAME_NBR	F0C6	
DMA4_CCRi_0 0102A020	WRITE_PRIORITY	0	BUFFERING_DISABLE 0
	PREFETCH	0	SUPERVISOR 0
	BS	0	TRANSPARENT_COPY_ENABLE 1 -
			<u>ا</u> ا

The following command sequence can be used to save the contents of all configuration registers/on-chip peripheral registers to a file.

; PRinTer.FileType ASCIIE	; Select ASCII ENHANCED as output ; format ; (default output format)
PRinTer.FILE Per.lst	; Define Per.lst as output file
WinPrint.PER.view	; Save contents of all ; configuration registers/on-chip ; peripheral registers to the ; specified file

TRACE32 PowerView					- 0 ×
File Edit View Var Break Run	CPU Misc Trace Perf	Cov OMAP4430	app Window	Help	
M M ™ † 4. G ▶ II	ﷺ ? № 🖾 🕮	📕 🔂 🗟 🗟	8 🕲 1 🖉		
🛹 B::PER					
⊡ <u>Core Registers</u>					
⊞ ID Registers					
	figuration				
⊟ <u>Memory Management Unit</u>					
SCTLR 00C50078	TE ARM NMFI Disabled V 0x00000000 SW Disabled M Disabled	AFE Li EE Li I Di C Di	isabled ittle isabled isabled Instructior	TRE Disa RR Rand Z Disa A Disa	bled om bled bled
TTBR0 9C3F004A	TTBO 9C3F0000 RGN Back/alloca	ited	IRGN[1:0] S	Back/allocate Shared	d 📮
·					E. ₹
B::					
trigger devices trace	Data Var	List	PERF	other pre-	vious
C15:0000001 1212 Instruction	Cache Enable	system ready		N	IIX UP

The access class, address, bit position and the full name of the selected item are displayed in the state line; the full name of the selected item is taken from the processor/chip manual.

You can modify the contents of a configuration/on-chip peripheral register:

• By pressing the right mouse button and selecting one of the predefined values from the pulldown menu.

B::PER.view , "O	ore Registers,Memo	ry Manage	ment Unit"					×
Memory Ma	nagement Unit							
SCTLR	00C50078	TE	ARM	AFE	Disabled	TRE	Disabled	
		NMFI	Disabled	EE	Little	RR	Random	
		V	0x00000000	I	Disabled	7	Disabled	
		SW	Disabled	C	Disabled 🗸	Disabled	Disabled	
		М	Disabled			Enabled	2	
TTBR0	65B32FD9	TTB0	65B30000		IRGN[1:0] Back/no	t allocated	
		RGN	Back/not all	ocated	S	Nonshar	ed	
TTBR1	7BF7C042	TTB1	7BF7C000		IRGN[1:0] Back/a]	located	
		RGN	Noncacheable		S	Shared		
TTBCR	00000000	PD1	Enable PDO	Enable	N Of	F		-
•	III							•

By a double-click to a numeric value. A **PER.Set** command to change the contents of the selected register is displayed in the command line. Enter the new value and confirm it with return.

TRACE32 PowerView					
File Edit View Var Break Run CPU	Misc Trace Pe	erf Cov Ol	MAP4430app Wir	ndow Help	
M M 📅 + 4, G ▶ II 🕅	? № 🌚 🗮	1011 🔳 🤮	1 🗟 🗟 🔯	12	
🛹 B::PER					
□ <u>Core Registers</u>					^
⊞ ID Registers					
■ System Control and Config	uration				
SCTLR 00C50078 TE NMFI V V	ARM Disabled 0x00000000	AFE EE I	Disabled Little Disabled	TRE RR Z	Disabled Random Disabled
ACTLR 00000041 PARON FOZ	Disabled Disabled Disabled Disabled	EXCL DP1	Disabled Disabled	SMP DP2	1 Disabled
CPACR 00000000 FW ASEDIS CP10	Enabled No Denied	D32DIS	No	CP11	Denied 🗸
•					E. I
B:: PER.Set C15:0x201 %Long					
[ok] formats <td><string> op</string></td> <td>tions</td> <td></td> <td></td> <td>previous</td>	<string> op</string>	tions			previous
C15:00000201 Coprocessor Access Con	trol Register	system	ready		MIX UP

PER.Set.simple <address> <range> [%<format>] <value></value></format></range></address>	Mo chi
Data Set ~address ~range [% format] ~value	Mc

Modify configuration register/onchip peripheral Modify memory

Data.Set <address>|<range> [%<format>] <value>

Data.Set is equivalent to PER.Set.simple if the configuration register is memory mapped.

PER.Set.simple D:0xF87FFF10 %Long 0x0000b02

The layout of the PER window is described by a PER definition file.

The definition can be changed to fit to your requirements using the **PER** command group.

The path and the version of the actual PER definition file can be displayed by using:

VERSION.SOFTWARE

A B::VERSION.SOFTWARE	
Interim Build (32-bit) Software Version: S.2012.10.000040137 Build: 40137.	_
t32usbamd64.sys Jun 24 2010 USB C:\T32_ARM\fcc.t32 Oct 24 2012 Podbus (40134) C:\T32_ARM\bin\windows\t32marm.exe Oct 24 2012 Host Oct 24 2012 Operation System Oct 24 2012 Debugger C:\T32_ARM\fccarm.t32 Oct 24 2012 Controller C:\T32_ARM\peromap4430app.per	
Aug 51 2011 Del auto Per Pite	▼ h. ⊀

PER.view <filename>

Display the configuration registers/on-chip peripherals specified by *<filename>*

PER.view C:\T32_ARM\percortexa9mpcore.per

This training section introduces the most often used methods to display and modify memory:

- The **Data.dump** command, that displays a hex dump of a memory area, and the **Data.Set** command that allows to modify the contents of a memory address.
- The List.auto command, that displays the memory contents as source code listing.

A so-called **access class** is always displayed together with a memory address. The following access classes are available for all processor architectures:

P: 1000	Program address 0x1000
D: 6814	Data address 0x6814

For additional access classes provided by your processor architecture refer to your "Processor Architecture Manuals".

Display the Memory Contents

View Var Break Run CP	2			
Registers				
titi Dump	B::Data.dump			
📰 List Source 😼	Address / Expr	ession		
🔂 Watch				🗕 🚺 🗖 HLL
🛃 Referenced Var				
🔐 Locals	Width	Access	- Options	- Flag
😹 Stackframe with Locals	efault	efault	Track	Read
😹 Stackframe	© Byte	© F	☑ Orient	Write
Peripherals	Word		Ascii	
🛓 Symbols 🔹 🕨				
🚻 Groups	Cong		SpotLight	
🐇 Bookmarks				
Trace List	Ok			Cancel
📃 Message Area				

B::Data.dump				×	
Address / Expres	sion		7		
0x6814			- 👔 🗆 HL	_	
Width e default Byte Word	Access default E	Options Track Orient Ascii	Flag Read		
		E opoteight	Cancel		
	B::Data.dump (0	x6814) /DIALOG			
	C:0x6814	Find Me	odify	ong 👻	🗆 E 🔲 Track 🔍 Hex
	address	0	4 8	3 (0123456789ABCDEF
	SD: 00006810 SD: 00006820 SD: 00006820 SD: 00006830 SD: 00006850 SD: 00006850 SD: 00006870 SD: 00006880	83421780+06004 2010F820 54018 42010A04 87354 08000040 20014 6243200A 050402 02066040 50001 982A2100 40000 02254894 42830	185 038255Ci 080 2850648; C60 0801720; 624 42A1505; 2CD 1800414; 0F5 8080010; 801 8512040] 588 A080032;	0 A4404D68 2 A5DD1248 1 2423DC18 5 0820A090 2 410B0449 2 04B11560 0 10958806 4 C0484990	3:5783\$

Please be aware, that TRACE32 permanently updates all windows. The default update rate is 10 times per second.

If you enter an address range, only data for the specified address range are displayed. This is useful if a memory area close to memory-mapped I/O registers should be displayed and you do not want TRACE32 PowerView to generate read cycles for the I/O registers.

Conventions for address ranges:

- <start_address>--<end_address>
- <start_address>..<end_address>
- <start_address>++<offset_in_byte>
- <start_address>++<offset_in_word> (for DSPs)

Address / Exp 0x6814++0xf	ression —		
Width e default byte Word Long	Access default E	Options Flag Read	
Ok		Cancel	
	B::Data.dump (0	lx6814++0xf) /DIALOG	
	C:0x6814	jij Find Modify Lo	ng 🔹 🛛 E 🖓 Track 🖉 Hex 📝 ,
	address	0 4 8	C 0123456789ABCDEF
	SD:00006810 SD:00006820	+06004185 038255C0 2C10F820	H4404Jbb \$HVRSU\$£hMU4 ^ _55

Use \mathbf{i} to select any symbol name or label known to TRACE32 PowerView.	

B::Data.dump				
Address / Expre	ession ———			
			HLL	
Width	Access	Options		
 default 	 default 	Track Rea	d	
© Byte	©Ε	Vrite Vrite		
Word		🔽 Ascii		_
C Long		SpotLight		
	🛓 Browse Symbols			
Ok	114141*	t. .	Type: Symbols	
	symbol	type	address	
	sys_generate _ANSI_e2d	e_error	R : 000064BC A R : 000039D0	
	ast	(strtype1)	D:0000681400006827	
	aun	(struct un	tion1) D:00006A8000006A97	
	clib SigInt	(Int ())	R:000032E400003313 R:00005964 -	
	signie			

B::Data.dump			
Address / Expre	ssion		
ast			🗕 🚺 🗆 HLL
Width	- Access	- Ontions	Elag
 default 	 default 	Track	Read
© Byte	© E	🗹 Orient	Write
Word		📝 Ascii	
C Long		🔲 SpotLight	
	L] [
Ok			Cancel

By default an oriented display is used (line break at 2^x).

A small arrow indicates

the specified dump address.

B::Data.dump (a	ast) /DIALOG		- • •
D:0x6814	jiji Find	Modify Long 🔻 🗖 E	🔲 Track 🛛 Hex 📝 Ascii
address	0	4 8 C Ø1	23456789ABCDEF
SD:00006810	83421780	6004185 038255C0 A4404D68 87	BSSANRSUSENMOG
SD:00006820	2C10F820	4018080 28506482 A5DD1248 📲	3,885T2dP(H2B4
SD:00006830	42010A04	7354C60 08017201 2423DC18 🖡	ቪB ነL5% ቪዮ ቪፄ ቴሮ #\$ 📃 👻
SD:00006840	08000040	2014624 42A15055 0820A090 🛯	NB\$FA,UPAB86_B
SD:00006850	6243200A	150402CD 18004142 410B0449 🖡	Cb55F6BA03IF4A
SD:00006860	02066040	;00010F5 80800102 04B11560 Q`	ASE1NDS5888 18E
	4		

Data.dump 0x6814	; Display a hex dump starting at ; address 0x6814
Data.dump 0x68100x682f	; Display a hex dump of the ; specified address range
Data.dump 0x68100x682f	; Display a hex dump of the ; specified address range
Data.dump 0x6810++0x1f	; Display a hex dump of the ; specified address range
Data.dump ast	; Display a hex dump starting at ; the address of the label ast
Data.dump ast /Byte	; Display a hex dump starting at ; the address of the label ast in ; byte format

TRACE32 PowerView
File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app Window Help
▶ ▶ ↓ ↓ ↓ ↓
IIII B::Data.dump (ast) /DIALOG
D:0x4A326DCC 🛐 Find Modify Long 🔻 🗉 E 🗌 Track 🖉 Hex 🖉 Ascii
address 0 4 8 C 0123456789ABCDEF
NSD:4A326DC0 BC79F602 7927C8BD DF537F69+BC512D36 % y 5 6-0
NSD:44326DD0 0000303A 4A326DCC E1D/828C DAF4F549 :0005m23295174A
NSD:4A326DE0 00010101 01000101 01010001 00010100 HARDERUARURUHU
NSD:4A326000 B4072266 16C98450 783E8398 5006741 HHU37WF0198HAG08
NSD:4A326E10 186F436C C66B740F 69448476 638E885 1C65+K\$v\$D195:5
NSD:4A326E20 B46E638C 26273793 31AF8E28 BAEECB4C ေငက်ရှိ37 & ေန်ာန်းနှင့် 🗸 🔫
B::D.S.NSD:0x4A326DEC %LE %Long
[ok] formats <aata> <string options="" previous<="" td=""></string></aata>
NSD:4A326DEC \\demo\Global\mags+0x0C s/stem ready MIX UP
[ok] formats <data> <string< th=""> options previous NSD:4A326DEC \\demo\Global\mags+0x0C stem ready MIX UP </string<></data>

By a left mouse double-click to the memory contents a **Data.Set** command is automatically displayed in the command line, you can enter the new value and confirm it with return.

Data.Set <address>|<range> [%<format>] <value> [I<option>]

Data.Set 0x6814 0xaa	; Write 0xaa to the address ; 0x6814
Data.Set 0x6814 %Long 0xaaaa	; Write Oxaaaa as a 32 bit value to ; the address 0x6814, add the ; leading zeros automatically
Data.Set 0x6814 %LE %Long 0xaaaa	; Write Oxaaaa as a 32 bit value to ; the address Ox6814, add the ; leading zeros automatically
	; USE LITTLE Endian mode

TRACE32 PowerView updates the displayed memory contents by default only if the cores is stopped.

A TRACE32 PowerVie	2W		
File Edit View Va	ar Break Run CPU Misc Trace Perf	Cov OMAP4430app Window Help	
H H M + 4	ヽ Ϛ ▶ Ⅱ 🕅 ᠀ 🗞 🍩 📰 🗄	1월 📖 6월 6월 6월 🧐 🧎 🔑	
101 By Data during (ast)	K/\$\$\$\$\$\$\${/////////////////////////////		
D:0x4A326DCC	ji Find Modify Long	E Track V Hex V	Ascii
address	0 4 8	C 0123456789ABCDEF	
NSD:4A326DC0	BC79F602 7927C8BD DF537F69+BC5	12D36 x y b y i f S f 6 - Q c	
NSD:4A326DD0	0000303A 4A326DCC E1D7B28C DAF	4F549 :000 cm2Jc271I54A	
NSD:4A326DE0	00010101 01000101 01010001 000		+
NSD:4A3260F0	E2000101 648F4E08 0198A981 A86	46741 HHUZTNFC198HAGC8	
	•		E M
B::			
5.1.			
trigger devi	ices trace Data Var	List other prev	ious
	running	MIX	UP

A hatched window frame indicates that the information display is brozen because the core is executing the program.

A TRACE32 PowerVie	ew
File Edit View V	/ar Break Run CPU Misc Trace Perf Cov OMAP4430app Window Help
▶ ₩ ⊡ + 4	イ と ▶
111 태양 B::Data.dump (ast	a) /DIALOG
D:0x4A326DCC	Find Modify Long ▼ □ E □ Track ☑ Hex ☑ Ascii
address	0 4 8 C 0123456789ABCDEF
NSD:4A326DC0	BC79F602 7927C8BD DF537F69+BC512D36 ^x ₆ y ^c ₆ S yiFSF6-Q ^c _c
NSD:4A326DD0	0000303A 4A326DCC E1D7B28C DAF4F549 :0010102 25271 IF44
NSD:4A326DE0	E2000100 648E4E08 0198A981 A8646741 NSNEVNE de 498 Agd
N30.4A320010	4
<u>ل</u> ــــــــــــــــــــــــــــــــــــ	з
B::	
trigger dev	ices trace Data Var List other previous
NSR:4A326580 \\	demo\demo\sieve+0x1C stopped MIX UP

The plain window frame indicates that the information is updated, because the program execution is stopped. Various cores allow a debugger to read and write physical memory (not cache) while the core is executing the program. The debugger has in most cases direct access to the processor/chip internal bus, so no extra load for the core is generated by this feature.

Open the **SYStem** window in order to check if your processor architecture allows a debugger to read/write memory while the core is executing the program:



Please be aware that caches, MMUs, tightly-coupled memories and suchlike add conditions to the run-time memory access or at worst make its use impossible.

Restrictions

The following description is only a rough overview on the restrictions. Details about your core can be found in the **Processor Architecture Manual**.

Cache

If run-time memory access for a cached memory location is enabled the debugger acts as follows:

Program execution is stopped

The data is read via the cache respectively written via the cache.

Program execution is running

Since the debugger has no access to the caches while the program execution is running, the data is read from physical memory. The physical memory contains the current data only if the cache is configured as write-through for the accessed memory location, otherwise out-dated data is read.

Since the debugger has no access to the cache while the program execution is running, the data is written to the physical memory. The new data has only an effect on the current program execution if the debugger can invalidate the cache entry for the accessed memory location. This useful feature is not available for most cores.

MMU

Debuggers have no access to the TLBs while the program execution is running. As a consequence run-time memory access can not be used, especially if the TLBs are dynamically changed by the program.

In the exceptional case of static TLBs, the TLBs can be scanned into the debugger. This scanned copy of the TLBs can be used by the debugger for the address translation while the program execution is running.

Tightly-coupled Memory

Tightly-coupled memory might not be accessible via the system memory bus.

Usage

The usage of the non-intrusive run-time memory access has to be configured explicitly. Two methods are provided:

- Configure the run-time memory access for a specific memory area.
- Configure run-time memory access for all windows that display memory contents (not available for all processor architectures).

Configure the run-time memory access for a specific memory area:



A plain window frame indicates that the information is updated while the core is executing the program

If the E check box is enabled, the attribute E is added to the memory class:

EP: 1000	Program address 0x1000 with run-time memory access
ED: 6814	Data address 0x6814 with run-time memory access

Write accesses to the memory work correspondingly:

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app Window Help	
M M ⊷ + 4 ¢ ▶ II ⊠ 8 № 💿 🗄 🕮 📕 🚳 🚳 🚳 🕸 2 ⊘	
🗱 B::Data.dump (ast) /DIALOG	
ED:0x4A326DCC 👔 Find Modify Long 🔻 🗹 E 🗌 Track 🖉 Hex 🖉 Ascii	
address 0 4 8 C 0123456789ABCDEF	
ENSD:4A326DC0 BC79F602 7927C8BD DF537F69→BC512D36 5 yt 5 6-05	
ENSD:4A326DD0 0000303A 4A326DCC E1D/B28C DAF4F549 :0%%Em2365%EF%S%	
ENSD:4A326DE0 00010101 01000101 00010001 00010100 THATTATATATOTATA	
	Data dat via rup timo
B:: U.S ENSU: UX4A3200F0 TOLE TOLOTIG UXOA	
	memory access
[ok] formats <a href="https://www.databasecond-condition-condita-condita-condition-condition-condition-condition-condition-con</td> <td>(attribute F)</td>	(attribute F)
ENSD:4A326DF8 (\demo\Global_libspace_start+0x4 running MIX UP	(ambate _)

SYStem.MemAccess Enable	; Enable the non-intrusive ; run-time memory access
;	
Go	; Start program execution
Data.dump E:0x6814	; Display a hex dump starting at ; address 0x6814 via run-time ; memory access
Data.Set E:0x6814 0xAA	; Write 0xAA to the address ; 0x6814 via run-time memory ; access

Configure the run-time memory access for all windows that display memory (not available for all cores):



•	TRACE	32 Po	werVie	w for	Pow	erPC)													. 0	3	X				
File	Edit	Vie	ew Va	ır B	reak	Ru	n CPl	M	lisc	Trac	e F	robe	Р	erf	Cov	М	PC5	xxx	Win	dow	H	Help				
	l 🖬	L.	↓ 4	· ج		· II	12	Ŷ	N ?			1010 101	1	6	6	o' 6	ď	Ø	1 4	3						
010 10:	B::Da	ta.du	mp (fla	gs) /l	Byte /	/DIAL	.OG											-)[=	3)(6	28			•	
	D:0x40	0041	28		jΈ	ind		odify)		Byte	9	•	[E	E	Tra	ick	🔽 н	ex		Ascii				
	CD 4	add	ress	0	1	2	3 4	5	6	7	8	9	A	B	C	D	E	F								
	SD:4	000	4130	01	00	01	00 00	00	01	001	00	01	00	00	00	00	00	00				* =				
	SD:4	000	4140	00	00	00	00 00	00	00	00		00	00	00	00	00	00	00				*				
	SD:4	000	4160	00	00	00	00 00	00	00	00	00	00	00	00	00	00	00	00				*				
	SD:4 SD:4	000	4180	00	00	00	00 00	00	00	00	00	00	00	00	00	00	00	00								
	SD:4	000	4190	00	00	00	00 00	00	00	00	00	00	00	00	00	00	00	00				Ψ.				
1																						r				
-	B::PER																					X				
±.	Wakeı	up U	nit (WKP	U)																	_ ^				
±.	Perio	odic	Inte	erru	pt '	Time	er an	d Re	al	Tim	e I	nter	ru	ot	(PI	r_R	TI)					_				
	Syste	em T	imer	Mod	ule	(51	M)																			
	STM_C	R NT	0000	0000	0 CI	PS	00	RZ	0	TE	N (0														
			1 0		-																	E				
	STN	anne 1_CC	R0 00	000	001	CEN	1															-				
	STN	A_CI	R0 00	0000	001 874	CIF	1																			
			1 1																							
	E Cha	anne 1_CC	R1 00	0000	000	CEN	0															-				
	STN	1_CI	R1 00	0000	000	CIF	0							_								-				
Ľ																						, d				
B:	: D. S	SD:	0x400	0041	48	%BF	0x99											-					_			
[
	[ok]		form	ats		<dat< td=""><td>a></td><td><st< td=""><td>ring</td><td>></td><td>0</td><td>ption</td><td>s</td><td></td><td></td><td></td><td></td><td>prev</td><td>ious</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></st<></td></dat<>	a>	<st< td=""><td>ring</td><td>></td><td>0</td><td>ption</td><td>s</td><td></td><td></td><td></td><td></td><td>prev</td><td>ious</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></st<>	ring	>	0	ption	s					prev	ious							
SD	:4000	4148	\\dia	bc\Gl	obal	vtde	f2+0x8	ru	Innin	q									н		U	Р				
	0.000							100		EASINF.											1			I		

All windows that display memory have a plain window frame, because they are updated while the core is executing the program

Write access is possible for all memories while the core is executing the program

SYStem.MemAccess Enable	; Enable the non-intrusive ; run-time memory access
SYStem.Option.DUALPORT ON	; Activate the run-time memory ; access for all windows that ; display memory
	; this SYStem.Option is only ; available for some processor ; architectures
;	
Go	; Start program execution
Data.dump 0x6814	; Display a hex dump starting at ; address 0x6814 via run-time ; memory access
Data.Set 0x6814 0xAA	; Write 0xAA to the address ; 0x6814 via run-time memory ; access

If your processor architecture doesn't allow a debugger to read or write memory while the core is executing the program, you can activate an intrusive run-time memory access if required.

B::SYStem.view				x	
- Mode	_ MemAccess	Option	Option	٦Å	
© Down	O DAP	IMASKASM	DACR		
🔘 NoDebug	© TSMON3	IMASKHLL	MMUSPACES		
🔘 Go	🔘 RealMON	TURBO	MPU		
Attach	C TrkMON	🔲 BigEndian	CFLUSH	E	
💿 StandBy	C GdbMON	📝 ResBreak			
🔘 Up (StandBy)	Oenied	INTDIS		-	
🖲 Up	– CpuAccess –	DBGACK	C AMBA		
	💿 Enable 🔫	ShowError	NODATA	_	Cpuaccess Enable allows an
reset	O Denied	🗹 EnReset	EXEC		intrusive run-time memory acc
RESetOut	Nonstop	🔳 WaitReset	SPLIT	-	
•	III			E at	

If an intrusive run-time memory access is activated, TRACE32 stops the program execution periodically to read/write the specified memory area. Each update takes at least **50 us**.



The time taken by a short stop depends on various factors:

- The time required by the debugger to start and stop the program execution on a processor/core (main factor).
- The number of cores that need to be stopped and restarted.
- Cache and MMU assesses that need to be performed to read the information of interest.
- The type of information that is read during the short stop.

An intrusive run-time memory access is only possible for a specific memory area.

Enable the E check box to su the run-time memory access	witch ₅ to ON
TRACE32 PowerView File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app Window Help N H L L C C L II I I I I I I I I I I I I I	 A plain window frame indicates that the information is updated while the core(s) is executing the program

A red ${\rm S}$ in the state line indicates that a TRACE32 feature is activated that requires short-time stops of the program execution

Write accesses to the memory work correspondingly:

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app	Window Help
M M M I + √ C ▶ II 巡 ? № ◎ 副 盟 図 図 図	🕲 1 🖉
B::Data.dump (ast) /DIALOG	
ED:0x4A326DCC	Frack 🔽 Hex 🔽 Ascii
address 0 4 8 C 0123456789	9ABCDEF
ENSD:4A326DC0 BC79F602 7927C8BD DF537F69+BC512D36 5 ycbs yi	FSF6-Q ^E
ENSD:4A326DD0 0000303A 4A326DCC E1D/B28C DAF4F549 :0005m23c	
ENSD:4A326DE0 00010101 01000101 00010001 00010100 HHH0HH0HH0HH0H	
	0.00Agda8
	·
B::D.S ENSD:0x4A326DF8 %LE %Long 0x99	Data.Set via run-time
	memory appage with short
	memory access with short
[OK] Tormats <aata> <string> options</string></aata>	stop of the program
running S	MIX UP execution
	execution

SYStem.CpuAccess Enable	; Enable the intrusive ; run-time memory access
;	
Go	; Start program execution
Data.dump E:0x6814	; Display a hex dump starting at ; address 0x6814 via an intrusive ; run-time memory access
Data.Set E:0x6814 0xAA	; Write 0xAA to the address ; 0x6814 via an intrusive ; run-time memory access



Enable the option **SpotLight** to mark the memory contents changed by the last 4 single steps in orange, older changes being lighter.

B::Data.dump (flags) /SpotLight /DIALOG	- • ×
D:0x7E7C	👬 Find Modify Byte 🔹 🗆 E 🗌 Track 🖉 H	Hex 🛛 Ascii
address	0 1 2 3 4 5 6 7 01234567	
SD:00007E78	16 A5 3D 90+01 00 01 01 19=85855	*
SD:00007E80	01 01 01 01 01 01 01 01 01 11 11 11 11 1	E
SD:00007E88	01 01 01 01 01 01 01 0C AAAAAAA	-
SD:00007E90	00 47 03 4A 50 C2 90 A1 NG5JP589	
SD:00007E98	8A 01 01 85 33 15 99 B2 ลิลิลิธิ3535	
SD:00007EA0	1A 00 10 4B 6C 12 0D 02 ANAKIZES	
SD:00007EA8	16 06 2D 31 52 48 81 58 th-1RHix	-
	4	h. I⊀

Data.dump flags /SpotLight

- ; Display a hex dump starting at
- ; the address of the label flags
- ; Mark changes

Displays the Source Listing Around the PC

Registers	TRACE32 PowerView	
Dump	ile Edit View Var Break Run CPU Misc Trace Perf Cov OM	AP4430app Window Help
List Source	ਮ ਅ ਯੋ ↑ ੯ ਓ । ਮ ਕੋ ੈ ਲੋ 💿 🗐 🖼 🔳 😭	63 61 🕲 1 🖉
Natch 13		
Referenced Var	B::List	
ocals	N Step Nover Diverge Return C Up	o 📕 Break 🖄 Mode
Stackframe with Locals	addr/line code label mnemonic	comment
Stackframe	428 {	
Peripherals	NSR:4A326598 E1A01082 lsl r1,r2,#0x	+ 5, 1 ; r1,i,#1
Symbols 🕨	NSR:4A32659C E2813003 add r3,r1,#0x	3; r3,r1,#3
Groups	429 K = 1 + primz NSR:4A3265A0 E0821003 add r1.r2.r3	; ; r1.j.r3
Bookmarks	register int i, primz, k;	
Trace List	int anzahl;	
Message Area	420 anzahl = 0; NSR:4A3265A4 E3A04000 mov r4,#0x0	; r4,#0 -
B	trigger devices trace Data Var ot SR:44326598 \\demo\demo\sieve+0x34 stopped at breakpoint	ther previous

If MIX mode is selected for debugging, assembler and HLL information is displayed



information is displayed



```
List.auto [<address>] [/<option>]
```

Display source listing

List	; Display a source listing ; around the PC
List E:	; Display a source listing, ; allow scrolling while the ; program execution is running
List *	; Open the symbol browser to ; select a function for display
List func17	; Display a source listing of ; func17

Breakpoints

Videos about the breakpoint handling can be found here: support.lauterbach.com/kb/articles/using-breakpoints-in-trace32

Breakpoint Implementations

A debugger has two methods to realize breakpoints: Software breakpoints and Onchip breakpoints.

Software Breakpoints in RAM

The default implementation for breakpoints on instructions is a Software breakpoint. If a Software breakpoint is set the original instruction at the breakpoint address is patched by a special instruction (usually TRAP) to stop the program and return the control to the debugger.

The number of software breakpoints is unlimited.

(B::List)				_		
Step Nover	🛃 Diverge 🖌 🗸 Return	C Up	► Go II	Break 🕅 🏙 Mode	😹 t 🖘	
addr/line source	e					
674	if (f]	ags[i])			^	
676	{	primz = i +	i + 3;			
677		k = i + pri	mZ;			
0/0		{	- 512E)			
680 681		tla k +	gs[k] = F = primz:	ALSE;		
692		}	[
100	}	anzam++;				
	ł					
687	return anzahl;				-	
000 5					F	
Break Run CPU Misc	-					
Set						
A Method						
(* Wethod						
X Delete All						
T Trigger Bus						
OnChip Trigger						
Trigger Reset	🕲 B::Break.List					
rigger Keset	Delete All	le Al 🔘 Enable Al	I 🛇 Init	A Method	pre 🗣 Load.	🔯 Set
	address	type	method			
	address F:400013	type 38 Program	method SOFT	v ⊘ sieve	\19	

Breakpoints on instructions are called **Program** breakpoints by TRACE32 PowerView.



Please be aware that TRACE32 PowerView always tries to set an Onchip breakpoint, when the setting of a Software Breakpoint fails.

TRACE32 allows to set Software breakpoints to FLASH. Please be aware that the affected FLASH sector has to be erased and programmed in order to patch the break instruction used by the Software breakpoint. This usually takes some time and reduces the number of FLASH erase cycles. For details refer to "Software Breakpoints in FLASH" (norflash.pdf). Most core(s) provide a small number of Onchip breakpoints in form of breakpoint registers. These Onchip breakpoints can be used to set breakpoints to instructions in read-only memory like onchip or NOR FLASH.

(B::List)	
Step Nover	🛃 Diverge 🖌 Return 🗶 Up 🕨 Go 🔢 Break 🔛 Mode 😹 九 🔫
addr/line sou	rce
674	i if (flags[i])
	{
676	primz = 1 + 1 + 3;
678	$\kappa = 1 + primz;$ while ($\kappa \ll STZE$)
0/0	
680	flags[k] = FALSE;
681	k += primz;
683	anzah]++:
	}
	}
687	roturn anzahl:
688 }	recurn anzani,
4	
, _	
Break Run CPU Misc	


Since Software breakpoints are used by default for Program breakpoints, TRACE32 PowerView **can** be informed explicitly where to use Onchip breakpoints. Depending on your memory layout, the following methods are provided:

1. If the code is completely located in read-only memory, the default implementation for the Program breakpoints can be changed.

Set					
List	B::Break.METH	OD			
Method	Program	Read	Write		
Delete All	© AUTO	AUTO	OTUA		
	© SOFT	O SOFT	© SOFT		
Trigger Bus	© HARD	© HARD	© HARD		
OnChip Trigger	Onchip	Onchip	Onchip		
Trigger Reset			[
	Alpha	Beta	Charly	Delta	Echo
	O AUTO	AUTO	OTUA (I)	OTUA (I)	AUTO
	SOFT	© SOFT	© SOFT	© SOFT	© SOFT
	C HARD	C HARD	C HARD	C HARD	C HARD
	Onchip	Onchip	Onchip	Onchip	Onchip
	·				

Change the implementation of Program breakpoints to **Onchip**

Break.METHOD Program Onchip

Advise TRACE32 PowerView to implement Program breakpoints always as Onchip breakpoints

2. If the code is located in RAM and onchip/NOR FLASH you can define code ranges where Onchip breakpoints are used.

MAP.BOnchip <range></range>	Advise TRACE32 PowerView to implement Program breakpoints as Onchip breakpoints within the defined address range
MAP.List	Check your settings

MAP.BOnchip 0xA0000000++0x1FFFFF

MAP.BOnchip 0x0++0x1FFF

Check your settings as follows:

Mise: Trace Probe Perf Cov MI Image: Constraint of the second sec	For the specified address ranges Program breakpoints are implemented as Onchip breakpoints. For all other memory areas Software breakpoints are used.
Choose Colors Choose Col	Backback List address Lype bus bonch ip attributes A:0000000000001FFF bonch ip bonch ip bonch ip A:00000000A01FFFFF bonch ip bonch ip im

Onchip breakpoints can be used to stop the core at a read or write access to a memory location.

📰 (B::List.aut	to]							
Step	🔰 Over	Diverge	🖌 Return	🗶 Up	► Go	Break	Mod	e Find:
addr/1	ine source							
÷	689	for (i	= 0 ; i	<= SIZE ;	i++)			^
	691 693 694 695 697 698 700 ⊀	}	;f (fla & & & & ; } ↓ 0 0 0 1 1 1 1 1 1	as i i) Variable Add to Watch View in Windo Set Value Modify Value. Go Till Breakpoint Advanced Bree Breakpoints Display Memo Display Trace Grep in Source other	Window www akpoint ory efiles	ReadV Read Write Spot Alpha Beta Charly Delta Echo	Vrite	

ſ	🕲 B::Break.List 📃 🖬 📒	×
	X Delete All O Disable Al O Enable All O Init / Method Store Store Cod O	
	address type method	_
	C:400041284000413A write ONCHIP 🗸 🖉 flags	*
		Ŧ
	↓ ↓ ↓	

Refer to your Processor Architecture Manual for a detailed list of the available Onchip breakpoints.

For some processor architectures Onchip breakpoints can only mark **single addresses** (e.g Cortex-A9). Most processor architectures, however, allow to mark **address ranges** with Onchip breakpoints. It is very common that one Onchip breakpoint marks the start address of the address range while the second Onchip breakpoint marks the end address (e.g. MPC57xx).

The command **Break.CONFIG.VarConvert** (TrOnchip.VarConvert in older software versions) allows to control how range breakpoints are set for scalars (int, float, double).

Break.CONFIG.VarConvert ON	If a breakpoint is set to a scalar variable (int, float, double) the breakpoint is set to the start address of the variable. + Requires only one single address breakpoint. - Program will not stop on unintentional accesses to the variable's address space.
Break.CONFIG.VarConvert OFF	If a breakpoint is set to a scalar variable (int, float, double) breakpoints are set to all memory addresses that store the variable value.
	 + The program execution stops also on any unintentional accesses to the variable's address space. - Requires two onchip breakpoints since a range breakpoint is used.

The current setting can be inspected and changed from the **Break.CONFIG** window.

Example: the red line in the Data.View window shows the range of the Onchip breakpoint.

🔍 B::Data.View vir	nt			3
breakpoint	address	data value	symbol	
W	SD:4000406C	00 '\"	\\diabc\Global\vint	
	SD:4000406D	00 '\"	\\diabc\Global\vint+0x1	
	SD:4000406E	00 '\"	\\diabc\Global\vint+0x2	_
	SD:4000406F	00 '\"	\\diabc\Global\vint+0x3	Ŧ
J		•	4	

; Set an Onchip breakpoint to the start address of the variable vint Break.CONFIG.VarConvert ON Var.Break.Set vint /Write Data.View vint

; Set an Onchip breakpoint to the whole memory range address of the ; variable vint Break.CONFIG.VarConvert OFF Var.Break.Set vint /Write Data.View vin

Q B::Data.View vi	nt			×
breakpoint	address	data value	symbol	
W	SD:4000406C	00 '\'	\\diabc\Global\vint	
W	SD:4000406D	00 '\'	\\diabc\Global\vint+0x1	
W	SD:4000406E	00 '\'	\\diabc\Global\vint+0x2	
W	SD:4000406F	00 '\"	\\diabc\Global\vint+0x3	
	SD:40004070	00 '\"	\\diabc\Global\vlong	-
		•	III	•

A number of processor architectures provide only **bit masks** or **fixed range sizes** to mark an address range with Onchip breakpoints. In this case the address range is always enlarged to the **smallest bit mask/next allowed range** that includes the address range.

It is recommended to control which addresses are actually marked with breakpoints by using the **Break.List /Onchip** command:

Breakpoint setting:

Var.Break.Set str2 Break.List

🕲 B::Break.List				
X Delete All O Disable	All 🖲 Enable All 🛛 🛇 Init	🦉 Method 😰 Sto	ore 🛛 🔀 Load 🛛 🔞 Set]
addres	s type	method		
C:20005524-	-20005537 Write	ONCHIP 🗸 🕻	ð str2	*
				T
J	•			€ at

Break.List /Onchip

🕲 B::Break.List /Onchip						×
X Delete All O Disable All 🖲 Enable	All 🛇 Init	(> Method)	🖀 Store 🔀 Load.		🕸 Set	
address	type	method	onchip resource			
C:2000552020005537	Write	ONCHIP	01	V	(vppulong)(str2+0x13)	*
						-
_	4				•	

ETM Breakpoints for ARM or Cortex-A/-R

ETM breakpoints extend the number of available breakpoints. Some Onchip breakpoints offered by ARM and Cortex-A/-R cores provide restricted functionality. ETM breakpoints can help you to overcome some of these restrictions.

ETM breakpoints always show a break-after-make behavior with a rather large delay. Thus, use ETM breakpoints only if necessary.

	Program Breakpoints	Read/Write Breakpoints	Data Value Breakpoints
ARM7 ARM9	Onchip breakpoints: up to 2, but address range only as bit mask (Reduced to 1 if soft- ware breakpoints are used) ETM breakpoints: up to 2 exact address ranges	Onchip breakpoints: up to 2, but address range only as bit mask ETM breakpoints: up to 2 exact address ranges	Onchip Breakpoint: up to 2, but address range only as bit mask ETM breakpoints: up to 2 data value breakpoints for exact address ranges
ARM11	Onchip breakpoints:	Onchip breakpoints:	Onchip breakpoints:
	6, but only single	2, but only single	no data value breakpoints
	addresses	addresses	possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	up to 2 exact address	up to 2 exact address	up to 2 data value breakpoints
	ranges possible	ranges possible	for exact address ranges
Cortex-A5	Onchip breakpoints:	Onchip breakpoints:	Onchip breakpoints:
	3, but only single	2, but address range	no data value breakpoints
	addresses	only as bit mask	possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	up to 2 exact address	up to 2 exact address	up to 2 data value breakpoints
	ranges	ranges	for exact address ranges
Cortex-A7 Cortex-R7	Onchip breakpoints: 6, but only single addresses	Onchip breakpoints: 4, but address range only as bit mask	Onchip breakpoints: no data value breakpoints possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	up to 2 exact address	up to 2 exact address	up to 2 data value breakpoints
	ranges	ranges	for exact address ranges
Cortex-A8	Onchip breakpoints:	Onchip breakpoints:	Onchip breakpoints:
	6, but address range	2, but address range	no data value breakpoints
	only as bit mask	only as bit mask	possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	up to 2 exact address	up to 2 exact address	up to 2 data value breakpoints
	ranges	ranges	for exact address ranges

	Program Breakpoints	Read/Write Breakpoints	Data Value Breakpoints
Cortex-R4 Cortex-R5	Onchip breakpoints: 28, but address range only as bit mask	Onchip breakpoints: 18, but address range only as bit mask	Onchip breakpoints: no data value breakpoints possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	up to 2 exact address	up to 2 exact address	up to 2 data value breakpoints
	ranges	ranges	for exact address ranges
Cortex-A9	Onchip breakpoints:	Onchip breakpoints:	Onchip breakpoints:
Cortex-A15	6, but only single	4, but address range	no data value breakpoints
Cortex-A17	addresses	only as bit mask	possible
	ETM breakpoints:	ETM breakpoints:	ETM breakpoints:
	2 exact address ranges	—	—

	Program Breakpoints		Data Value Breakpoints		
Cortex-A3x Cortex-A5x Cortex-A6x Cortex-A7x Cortex-R82 Cortex-X Neoverse	Onchip breakpoints: 6, but only single addresses ETM breakpoints: 2 exact address ranges (more on request)	Onchip breakpoints: 4, but address range only as bit mask ETM breakpoints: —	Onchip breakpoints: no data value breakpoints possible ETM breakpoints: —		
Cortex-R52	Onchip breakpoints: 8, but only single addresses ETM breakpoints: up to 2 exact address ranges	Onchip breakpoints: 8, but address range only as bit mask ETM breakpoints: —	Onchip breakpoints: no data value breakpoints possible ETM breakpoints: —		

No ETM breakpoints are available for the Cortex-M family.

Please refer to the description of the ETM.StoppingBreakPoints command, if you want to use the ETM breakpoints.

TRACE32 PowerView provides the following breakpoint types for standard debugging.

Breakpoint Types	Possible Implementations			
Program	Software (Default) Onchip			
Read, Write, ReadWrite	Onchip (Default)			

Program Breakpoints

	E [B::List]
	▶ Step ▶ Over ▲ Diverge ✔ Return ▲ Up ▶ Go II Break ™ Mode ▲ III addr/line source III III IIII IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
	674 [if (flags[i])
	676 677 678 while (k <= SIZE)
Set a Program breakpoint by a left mouse	680 681 k += primz;
double-click to the instruction	anzan1++; }
	687 return anzahl;

The red program breakpoint indicator marks all code lines for which a Program breakpoint is set.

The program stops before the instruction marked by the breakpoint is executed (break before make).

	B::List				
	📕 Step 📑 Ov	er 🕹 Next	🖋 Return 🖉 🙋	Up 🕨 🕨 Go	🚺 Break 🛛 🖄 Mode
	addr/line	code la	bel mnemon	ic	comment
	426 NSR:4A326590 NSR:4A326594 NSR:4A326598	{ E7DC1002 E3510000 0A000016	if (flags ldrb cmp beq	[i]) r1,[r12,+r2] r1,#0x0 0x4A3265F8	; r1,#0
Disable the Program	428	1	pr	imz = i + i + 3;	
	NSR:4A32659C	E1A01082	lsl	r1,r2,#0x1	; r1,i,#1
breakpoint by a	NSR:4A3265A0	E2813003	add	r3,r1,#0x3	; r3,r1,#3
left mouse double-click	429 NCD • 4 A 3 2 6 5 A 4	E0821003	R add	= 1 + pr1mz;	. n1 i nnimz
to the red program	NSK:4A3203A4	regi	ster int i, pr anzahl;	imz, k;	, r1,1,pr1m2
		•	III		н. •
The program breakpoint					
indicator becomes grey.					

Break.Set <address>/Program [/DISable]

Set a Program breakpoint to the specified address. The Program breakpoint can be disabled if required.

Break.Set 0xA34f /Program	; set a Program breakpoint to ; address 0xA34f
Break.Set func1 /Program	; set a Program breakpoint to the ; entry of func1 ; (first address of function func1)
Break.Set func1+0x1c /Program	; set a Program breakpoint to the ; instruction at address ; func1 plus 28 bytes ; (assuming that byte is the ; smallest addressable unit)
Break.Set func11\7	; set a Program breakpoint to the ; 7th line of code of the function ; func11 ; (line in compiled program)
Break.Set func17 /Program /DISable	; set a Program breakpoint to the ; entry of func17 ; diable Program breakpoint
Break.List	; list all breakpoints

435

436

÷



🐼 Add to Watch Window

Advanced Breakpoint
 Breakpoints

Display Memory

Grep in Sourcefiles

Display Trace

other

ReadWrite

67

Read

Nrite

Spot

6 View in Window

중 Set Value... 중 Modify Value... ★ Go Till

🙆 Breakpoint...



anzahl++:

abcd++;

Core stops at

a write access

to the variable



Break.Set <address> | <range> /Read | /Write | /ReadWrite [/DISable]

; allow HLL expression to specify breakpoint Var.Break.Set <hll_expression> /Read | /Write | /ReadWrite [/DISable]

Break.Set 0x0B56 /Read
Break.Set ast /Write
Break.Set vpchar+5 /ReadWrite /DISable
Var.Break.Set flags /Write
Var.Break.Set flags[3] /Read
Var.Break.Set ast->count /ReadWrite /DISable
Break.List

Breakpoint Setting at Run-time



Software breakpoints

- If MemAccess Enable/NEXUS/DAP is enabled, Software breakpoints can be set while the core(s) is executing the program. Please be aware that this is not possible if an instruction cache and an MMU is used.
- If **CpuAccess** is enabled, Software breakpoints can be set while the core(s) is executing the program. If the breakpoint is set via CpuAccess the real-time behavior is influenced.
- If MemAccess and CpuAccess is Denied Software breakpoints can only be set when the program execution is stopped.

The behavior of **Onchip breakpoints** is core dependent. E.g. on all ARM/Cortex cores Onchip breakpoints can be set while the program execution is running.

TRACE32 PowerView offers in addition to the basic breakpoints (Program/Read/Write) also complex breakpoints. Whenever possible these breakpoints are implemented as real-time breakpoints.

Real-time breakpoints do not disturb the real-time program execution on the core(s), but they require a complex on-chip break logic.

If the on-chip break logic of a core does not provide the required features or if Software breakpoints are used, TRACE32 has to implement an intrusive breakpoint.



Intrusive breakpoint perform as follows:

Each stop to perform the check suspends the program execution for at least 1 ms. For details refer to "StopAndGo Mode" in TRACE32 Concepts, page 57 (trace32_concepts.pdf)

B::			
trigger devices trace Data	Var other	previous	
running	S	MIX UF	

The (short-time) display of a red S in the state line indicates that an intrusive breakpoint was hit.

There are two standard ways to open a Break.Set dialog.



sYmbol.INFO func11

- ; display symbol information
- ; for function func11

🔒 B::sYmbol.INFO func11	
🔒 Symbols 🛛 🏙 Dump 🖉 List 🔍 View 🛛 🗱 MMU	
address info	*
attr: FLE	
function	
\\diabc\diabc\func11	
P:40000BF040000C87 global static	
function info	
* size: 0. push: [] use: [R0,R1,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12]	
epilog: P:40000C74 exit: P:40000C84	
module info	
Tanguage: ELF-C	
source: I:\T32DEMO\POWERPC\55xx\code_0x40000020_data_0x40004000\diabc.c	
type	
(int ()) function (int)	
(int) signed integer (32 bits)	
	Ŧ

Function Name/HLL Check Box OFF

Program breakpoint is set to the function entry (first address of the function).

address / expre	ssion		
unc11		•	👔 🗆 HLL 🔹 🕈
суре	options		- method
Program	EXclude	Temporary	auto 👻
ReadWrite	NoMark	DISable	- action
Read		DISableHIT	stop 👻
Write	DATA		
🕽 default		•	≽ advanced
Ok	Add	Delete	Cancel

🕲 B::Break.List					×
X Delete All O Disable All 🖲 Ena	able All 🛛 🛇 Init	🖉 Method	.) 😤 st	ore) 😰 Load 🛛 📦 Set	
address	type	method			
F:40000BF0	Program	SOFT	V 🖉	func11	~
					Ŧ
	•				►

Break.Set func11

Function name/HLL Check Box ON (only for special use cases)

- If the on-chip break logic supports ranges for Program breakpoints, a Program breakpoint implemented as Onchip is set to the full address range covered by the function.
- If the on-chip break logic provides only bitmasks to realizes breakpoints on instruction ranges, a
 Program breakpoint implemented as Onchip is set by using the smallest bitmask that covers the
 complete address range of the function.
- otherwise this breakpoint is rejected with an error message.

B::Break.Set			- • X
- address / expres	sion		
func11		-	👔 🗹 HLL 🔫
type Program ReadWrite Read Write default	options EXclude NoMark	Temporary DISable DISableHIT	method auto action stop advanced
Ok	Add	Delete	Cancel

🕲 B::Break.List	
X Delete All O Disable All Enable All S Init	Method 😰 Store 🔀 Load 🔞 Set
address type	method
F:40000BF040000C87 Program	ONCHIP 🗸 🖉 func11 🛛 🔺
	E. 4

Var.Break.Set func11

sYmbol.INFO func10\7

- ; display debug information
- ; for 7th program line in
- ; function func10



Program Line Number/HLL Check Box OFF

Program breakpoint is set to the first assembler instruction generated for the program line number.

诊 B::Break.Set		-	- 🗆	×							
address / express	on			_							
func10\7		~	🧎 🗌 ні	L 🗆*							
type	options		- metho	d							
Program	EXclude	Temporary	auto	\sim							
○ ReadWrite	NoMark	DISable	- action								
○ Read	DeleteHIT	DISableHIT	stop	~							
⊖ Write	DATA										
		~	🛛 📚 ac	lvanced							
Ok	Add	Delete	Ca	ncel							
		📵 B::Break.List	t							-	
		🖉 Setup	💥 Delet	e All 🛛 🔿	Disable All	🔘 Ena	ble All	⊗ Init	😤 Store	😤 Load	😂 Set
		ado	P+4000	type	r am	method	1 19	func10\7			
							× 🖉	, anero ()			0
		J		<							÷., ۲

Break.Set func10\7

Program Line Number/HLL Check Box ON

- If the on-chip break logic supports ranges for Program breakpoints, a Program breakpoint implemented as Onchip is set to the full address range covered by all assembler instructions generated for the program line number.
- If the on-chip break logic provides only bitmasks to realizes breakpoints on instruction ranges, a Program breakpoint implemented as Onchip is set by using the smallest bitmask that covers the

complete address range of the program line.

• otherwise this breakpoint is rejected with an error message.

😻 B::Break.Set			— (×
address / expressio	n	~	<u>:</u> ⊡⊦	ILL	*
type Program ReadWrite Read Write	options EXclude NoMark DeleteHIT DATA	Temporary DISable DISableHIT	auto actio stop	n — advar	~ ~
Ok	Add	Delete	C	ance	I

	🤨 B::Break.List	t								K
	🖉 Setup	💥 Delete All	O Disable All	Enab	le All	⊗ Init	😤 Store	😤 Load	😢 Set	
	ado	dress	type	method						
	P:4000078	BC400007C7	Program	ONCHIP	1 🖉	func10\7				$\hat{\mathbf{v}}$
J			<						>	

sYmbol.INFO flags

; display symbol information

; for variable flags

🛓 B::sYmbol.INFO flags	3
Symbols 🕮 Dump 🗐 List 🔍 View 🗱 MMU	
address info	
attr: DATA	
variable	
\\sieve_cm\Global\flags	
D:2000550020005512 global static	E
type	
(char [19]) array (char, 19 bytes, 018)	
(char) unsigned integer (8 bits)	
4	т. Н

Variable/HLL Check Box OFF

Selected breakpoint (ReadWrite/Read/Write) is set to the start address of the variable.

🔯 B::Break.Set				
- address / expres	ssion			
flags		•		
type	options		- method	
Program	EXclude	Temporary	auto 💌	
© ReadWrite	🔲 NoMark	DISable	- action	
© Read		DISableHIT	stop 👻	
Write	- DATA			
Ø default		-	♦ advanced	
Ok	Add	Delete	Cancel	
🕲 B::Break.List				
X Delete All O Di	isable All 💿 Enable All	🛇 Init 🖉 Met	thod) 😰 Store) 💈	🛿 Load 🛛 💕 Set
add	ress typ	e meth		
	C:20005500 Wr1	Le UNCH		ys *
				H. (

Break.Set flags

Variable/HLL Check Box ON

- If the on-chip break logic supports ranges for Read/Write breakpoints, the specified breakpoint is set to the complete address range covered by the variable.
- If the on-chip break logic provides only bitmasks to realizes Read/Write breakpoints on address ranges, the specified breakpoint is set by using the smallest bitmask that covers the address range used by the variable.

🔯 B::Break.Set					
- address / expres	ssion				
flags		•			
type Program ReadWrite Read Write default	options EXclude NoMark - DATA	Temporary DISable DISableHIT	method auto • action stop • Ø advanced		
OK	Add	Delete	Cancel		
🕲 B::Break.List					×
X Delete All O Di	sable All 💿 Enable All	O Init Ø Met	thod) 😰 Store) 🛐	Load 💕 Set	
C:200055	0020005512 Wr	ite ONCH	HIP √ Ø flag	js	*
					► d

Var.Break.Set flags

sYmbol.INFO flags

; display symbol information

; for variable flags

🔒 B::sYmbol.INFO flags	X
🔒 Symbols 🔠 Dump 📰 List 🔍 View 🗱 MMU	_
address info	
attr: DATA	
variable	
<pre>\\sieve_cm\Global\flags</pre>	
D:2000550020005512 global static	E
type	
(char [19]) array (char, 19 bytes, 018)	
(char) unsigned integer (8 bits)	
	Ŧ
F	

Variable/HLL Check Box Must Be ON

If you want to use an HLL expression to specify the address range for a Read/Write breakpoint, the HLL check box has to be checked.

- If the on-chip break logic supports ranges for Read/Write breakpoints, the specified breakpoint is set to the complete address range covered by the HLL expression.
- If the on-chip break logic provides only bitmasks to realizes Read/Write breakpoints on address ranges, the specified breakpoint is set by using the smallest bitmask that covers the address range used by the HLL expression.

🙆 B::Break.Set				
address / expres	sion			
flags[3]		-	👔 🗸 HLL 🗆 *	
type Program ReadWrite Read Write default	options	Temporary DISable DISableHIT	method auto action stop advanced	
Ok	Add	Delete	Cancel	
🕲 B::Break.List				
C:2000550	able All 💿 Enable All ress ty 0320005503 Wr	<mark>⊘ Init </mark>	hod) 😰 Store) 🛐 nod IIP 🗸 🖉 flag] Load 🕲 Set
1	1			P

Set Program breakpoints the all function that match the defined name pattern.

•	🚹 🗆 HLL 🛛 🛚 🖌	Check *	to enable wildcard usage
Add Delete	method auto action stop advanced Cancel		
🖲 Enable All 🖉 Init 🛛 🖉 Mat	had Store QL	ad Bill Sot	
type method 0080 Program SOFT 0114 Program SOFT 0174 Program SOFT 0174 Program SOFT 02A8 Program SOFT 022A8 Program SOFT 0220 Program SOFT 020 Program SOFT 020 Program SOFT 020 Program SOFT 021 Program SOFT 021 Program SOFT 021 Program SOFT 022 Program SOFT 023 Program SOFT 024 Program SOFT 025 Progr	V Ø func2 V Ø func2a V Ø func2b V Ø func2b V Ø func2b V Ø func2b V Ø func2d V Ø func21 V Ø func23 V Ø func24 V Ø func25 V Ø func26 V Ø func27		
	ns clude Temporary Mark DISable DISableHIT A Add Delete Enable All O Init Met type method 080 Program SOFT 114 Program SOFT 174 Program SOFT 17	Ins method clude Temporary Mark DISable DISableHIT A DISableHIT A DISABLEHIT	HUL V* Check * ns method uuto u

Requires sufficient resources if Onchip breakpoints are used.

Break.SetPATtern func2*

address / expre	ssion				
sieve		•	1 HLL 🛛 *		
type Program ReadWrite Read Write default	options	Temporary DISable DISableHIT	method auto auto SOFT Onchip ♥ advanced	-	Implementation
Ok	Add	Delete	Cancel		

Implementation	
auto	Use breakpoint implementation as predefined in TRACE32 PowerView.
SOFT	Implement breakpoint as Software breakpoint.
Onchip	Implement breakpoint as Onchip breakpoint.



By default the program execution is stopped when a breakpoint is hit (action **stop**). TRACE32 PowerView provides the following additional reactions on a breakpoint hit:

Action (debug	Action (debugger)				
Spot	Spot The program execution is stopped shortly at a breakpoint hit to update the screen. As soon as the screen is updated, the program execution continues.				
Alpha	Set an Alpha breakpoint.				
Beta	Set a Beta breakpoint.				
Charly	Set a Charly breakpoint.				
Delta	Set a Delta breakpoint.				
Echo	Set an Echo breakpoint.				
WATCH	Trigger the debug pin at the specified event (not available for all processor architectures).				

Alpha, Beta, Charly, Delta and Echo breakpoint are only used in very special cases. For this reason no description is given in the general part of the training material.

Action (on-chip or off-chip trace)				
TraceEnable	Advise on-chip trace logic to generate trace information on the specified event.			
TraceON	Advise on-chip trace logic to start with the generation of trace information at the specified event.			
TraceOFF	Advise on-chip trace logic to stop with the generation of trace information at the specified event.			
TraceTrigger	Advise on-chip trace logic to generate a trigger at the specified event. TRACE32 PowerView stops the recording of trace information when a trigger is detected.			

A detailed description for the Actions (on-chip and off-chip trace) can be found in the following manuals:

- "Training Arm CoreSight ETM Tracing" (training_arm_etm.pdf).
- "Training Cortex-M Tracing" (training_cortexm_etm.pdf).
- "Training AURIX Tracing" (training_aurix_trace.pdf).
- "Training Hexagon ETM Tracing" (training_hexagon_etm.pdf).
- "Training MPC5xxx/SPC5xx Nexus Tracing" (training_nexus_mpc5500.pdf).

or with the description of the Break.Set command.

The information displayed within TRACE32 PowerView is by default only updated, when the core(s) stops the program execution.

The action Spot can be used to turn a breakpoint into a watchpoint. The core stops the program execution at the watchpoint, updates the screen and restarts the program execution automatically. Each stop takes **50 ... 100 ms** depending on the speed of the debug interface and the amount of information displayed on the screen.

Example: Update the screen whenever the program executes the instruction sieve\11.

🔯 B::Break.Set	
– address / expressi	on
sieve\11	▼ 🚺 🗆 HLL 💭 *
L	
type	options method
Program	EXclude Temporary
ReadWrite	NoMark DISable - action
Read	
Write	
⊖ vvrice ⊜ da£auti	
o default	▼ advanced
Ok	😮 B::Break.List
	🗶 Delete All 🔾 Disable All 💿 Enable All 💿 Init 🖉 Method 😨 Store 🔀 Load 🔞 Set
	address type method action
	T:2000156A Program SOFT SPOT 🗸 🖉 sieve\11

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app Wind	dow Help
│ Ŋ M ム₄ ↓ √ ℭ ▶ Ⅱ ⊠ ᠀ № ◎ 副 翊 📕 ଔ ଔ ଔ 👹 💈	19
📰 B::Data.List	
🗎 Step 🕞 Over 🔛 Diverge 🖌 Return 🖒 Up 📄 🕨 Go 🛛 🔢 Break	Mode
addr/line source	
	^ ^
426 1f (flags[1]) {	
428 429 k = i + primz	
430 while (k <= SIZE)	
432 flags[k] = FALS	5E; -
	► a
😸 B::Var.Local 🗖 🗉 🖾 📕 B::Register	- • •
sieve()	Stack 🔺
anzahl = 3 C C R2 4 R10 3039	
R5 5400000 R13 4A326FE4	
1 _ R7 4A326DCC PC 4A3265A0	_
CPSR 20000105	• •
p	
trigger devices trace Data Var other previou	IS
NSR:4A3265A0 \\demo\demo\sieve+0x3C spotted S	HLL UP
	1 111

spotted indicates a breakpoint with the action Spot

Break.Set sieve\11 /Spot



Temporary	OFF: Set a permanent breakpoint (default). ON: Set a temporary breakpoint. All temporary breakpoints are deleted the next time the core(s) stops the program execution.
DISable	OFF: Breakpoint is enabled (default). ON: Set breakpoint, but disabled.
DISableHIT	ON: Disable the breakpoint after the breakpoint was hit.

Temporary breakpoints are usually not set via the **Break.Set** dialog, but they are often used while debugging.

Examples:

Go Till



Go <address> [<address> ...]

; set a temporary Program breakpoint to ; the entry of the function func4 ; and start the program execution Go func4 ; set a temporary Program breakpoints to ; the entries of the functions func4, func8 and func9 ; and start the program execution Go func4 func8 func9

Go Till -> Write

B::List NSR:0x	4A3260	j24]							[- • •
Step	• Ove	r 🛛 🛃 Diverge	Return	Ċ Up	🕨 🕨 Go	II Break	Mode	Find	:	demo.c
addr/1	ine	source								
	373	abcd	= 0;							^
	375	whil	e (TRUE)							
t t	377 378 379 380 382 384	۲	vtripp vtripp vtripp vtripp func2(func2a	learray[learray[] learray[] learray[(); ();	0] [0] [0] [] [0] [0] [] [1] [0] [] [0] [1]	= 1; = 2; ✓ ariable ✓ Add to Wate ✓ View in Win ✓ Set Value ✓ Modify Valu ✓ Go Till	ch Window dow ie		ReadWrite	-
						 Breakpoint Advanced B Breakpoints Display Mer Display Trace 	reakpoint nory :e	• • •	Read Write	
						引 Grep in Sou other	rcefiles	•		

```
Var.Go <hll_expression> [/Write]
```

; set a temporary write breakpoint to the variable

; vtripplearray[0][1][0] and start the program execution

Var.Go vtripplearray[0][1][0] /Write

Go.Return and similar commands

[B::List]			
Step	🖬 Over	La Diverge 🖌 Return 🗶 Up 🕞 Go 🛛 🖬 Break 🖉 Mode Find:	demo.c
addr	/line sour	ce	
	void	func2a()	*
	ł	auto char autovar; /* char stack variable */ register char regvar; /* char register variable */	
	155	autovar = regvar = mstatic1:	
	156	autovar++;	
÷	158 159 160 }	<pre>for (regvar = 0; regvar < (char) 5 ; regvar++) vchar += regvar*autovar;</pre>	
		III	· · ·
1			*

Go.Return

- ; first Go.Return
- ; set a temporary breakpoint to the start of the function epilogue ; and start the program execution

Go.Return

- ; stopping at the function $\ensuremath{\mathsf{epilog}}$ first has the advantage that the
- ; local variables are still valid at this point.
- ; second Go.Return
- ; set a temporary breakpoint to the function return
- ; and start the program execution

Go.Return

🔒 B::sYmbol.INFO func2a)						
🗜 Symbols 🔛 Dump 🗮 List 🔍 View 🔀 MMU							
address info							
/\diabc\diabc\func2a							
P:4000011440000173 global static							
<pre>function info * size: 0. push: [] use: [R0,R1,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12]</pre>							
epilog: P:4000015C exit: P:40000170							
module info							
<pre>language: ELF-C producer: Diab Data, Inc:dcc Rel 4.0b:PPC603 source: I:\T32DEMO\POWERPC\55xx\code_0x40000020_data_0x40004000\diabc.c</pre>							
type							
(void) void							
· · · · · · · · · · · · · · · · · · ·							

DATA Breakpoints

The DATA field offers the possibility to combine a Read/Write breakpoint with a specific data value.

 DATA breakpoints are implemented as real-time breakpoints if the core supports Data Value Breakpoints (for details on your core refer to "Onchip Breakpoints by Processor Architecture", page 76).

TRACE32 PowerView indicates a real-time breakpoints by a full red bar.

TRACE32 PowerView allows inverted data values if this is supported by the on-chip break logic.

DATA breakpoints are implemented as intrusive breakpoints if the core does not support Data Value Breakpoints. For details on the intrusive DATA breakpoints refer to the description of the **Break.Set** command.

TRACE32 PowerView indicates an intrusive breakpoint by a hatched red bar.

TRACE32 PowerView allows inverted data values for intrusive DATA breakpoints.
Example: Stop the program execution if a 1 is written to flags[3].

B::Break.Set address / expres flags[3]	ssion		
type Program ReadWrite Read Write default	options	□ Temporary □ auto □ DISable □ action □ DISableHIT stop ▼ ■ advanced	
Ok	Add	B::Break.List X Delete All O Disable All Enable address C:2000550320005503 Write	All O Init @ Method Store SLoad SSter method data ONCHIP BYTE 0x1 v @ f1ags[3]

File Edit View Var Break Run CPU Misc Trace Perf Cov STM32F4x Window Help
N Step N Over L Diverge ✔ Return C Up D Go II Break 100 Mode 660 t
800 TOP (1 = 0; 1 <= SIZE; TTAGS[1++] = TRUE);
⊕ 802 for (i = 0; i <= SIZE; i++) { if (flags[i]) { } }
804 prime = i + i + 3;
806 while (k <= SIZE) { ▼
6d B::Var.View flags[3]
•[f]ags[3] = 1
X Delete All O Disable All O Enable All O Init (2 Method 23 Store ∑ Load 10 Set
C:2000550320005503 Write ONCHIP BYTE 0x1 V Ø flags[3]
B::
components trace Data Var List PERF other previous
ST:2000154E \\sieve_cm\sieve\sieve+0x26 stopped by r/w breakpoint

Var.Break.Set flags[3] /Write /DATA.auto 1.

Example: Stop the program execution if another value then 1 is written to flag[3].

🔯 B::Break.Set			
- address / expres	ssion		
flags[3]			🔻 📝 🔍 HLL
vpe ○ Program ○ ReadWrite ○ Read	options EXclude	Temporary DISable DISableHIT	implementation auto action stop
 Write default 	- DATA		Sector advanced
Ok	Add	Delete	Cancel

🕲 B::Break.List				
X Delete All O Disable All Enable	All 💿 Init 🖉 In	npl 😰 Store 😨 Load.	📦 Set	
address ty	/pes impl	data		
C:4000412B Wr	ite ONCHI	P BYTE !0x1	flags[3]	
				-
4				



Var.Break.Set flags[3] /Write /DATA.auto !1.

If an HLL expression is used TRACE32 PowerView gets the information if the data is written via a byte, word or long access from the symbol information.

If an address or symbol is used the user has to specify the access width, so that the correct number of bits is compared.



Break.Set 0x11dcf /Write /DATA.Word 1234.

🔯 B::Break.Set				ŋ		
- address / express	sion					If the educated button is pushed
		▼			1	additional input fields are provided
type	options		- method			
Program	EXclude	Temporary	auto 💌			
ReadWrite	NoMark	DISable	- action			
Read		DISableHIT	stop 🔻			
🔘 Write	DATA					
🔘 default		-				
Ok	Add	Delete	Cancel			
	memory / registe	r / var				
ProgramPass						
ProgramFail	- TASK		- COUNT			
MemoryReadWrite			· 1.			
MemoryRead	- MACHINE					
Memoryvvrite DesisterBeadWrite	CONDition					Advanced breakpoint input fields
	CONDITION		🔽 LILL 🔲 AfterDer			· · · · · · · · · · · · · · · · · · ·
	CMD					
Citegrater write			RESUME			

If OS-aware debugging is configured (refer to "**OS-aware Debugging**" in TRACE32 Concepts, page 34 (trace32_concepts.pdf)), TASK-aware breakpoints allow to stop the program execution at a breakpoint if the specified task/process is running.

TASK-aware breakpoints are implemented on most cores as intrusive breakpoints. A few cores support realtime TASK-aware breakpoints (e.g. ARM/Cortex). For details on the real-time TASK-aware breakpoints refer to the description of the **Break.Set** command.

Intrusive TASK-aware Breakpoint

Processing:



Each stop at the TASK-aware breakpoint takes at least 1.ms. This is why the red S is displayed in the TRACE32 PowerView state line whenever the breakpoint is hit.

Example: Stop the program execution at the entry to the function EE_oo_TerminateTask only if the task/process "Task3" is running.

address / express	sion		
EE_oo_Terminate	eTask	•	1 HLL 🛛 *
type	options		method
Program	EXclude	Temporary	auto 🔻
ReadWrite	NoMark	DISable	- action
Read		DISableHIT	stop 🔻
🗇 Write	DATA		
🖯 default		-	A advanced
Ok	Add	Delete	Cancel
	memory / regist	er / var	
ProgramPass			- 🚊 🗆 HLL
🖱 ProgramFail	- TASK		COUNT
MemoryReadWrite	"Task3"		▼ 1.
MemoryRead	- MACHINE		CORE
MemoryWrite			
🔿 RegisterReadWrite	- CONDition		
🛛 RegisterRead			🗷 HLL 🔲 AfteSter
🔿 RegisterWrite	- CMD		

ſ	😢 B::Break.List		
	X Delete All O Disable All Enable All O Init	Method 😰 Store	🔁 Load 🔞 Set
	address type V:40001080 Program	SOFT "Task3"	√ Ø EE_oo_TerminateTask ▲

Break.Set EE_oo_TerminateTask /Program /TASK "Task3"

TRACE32 PowerView
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX EE_cpu_0 Window Help
H H ¼ ↓ √ Ċ ▶ II ? K @ 圖 ֎ ֎ ❷ ፤ ≫
E\$\$\$\$\$
🕨 Step 📑 Over 🎿 Diverge 🖌 Return 🙋 Up 🕨 Go 🔢 Break 💥 Mode 🐼 🖿 Find
addr/line_codelabelmnemoniccomment
SV:40001080 182106F0 EE_oo_Te:e_stwu r1,-0x10(r1) ; r1,-16(r1)
register EE_FREG np_flags;
65 EE_ORTI_set_service_in(EE_SERVICETRACE_TERMINATETASK);
SV:40001086 1C8D8080 e_add16i r4,r13,-0x7F80 ; r4,r13,-32640
E_OS_RESOURCE if the task still occupy resources
E_OS_CALLLEVEL if called at interrupt level
🕲 þýð fesk Kýst / / / / / / / / / / / / / / / / / / /
🗶 Delete All 🔘 Disable All 🔘 Enable All 🚫 Init 🖉 Method 😨 Store 💱 Load 🛍 Set
ress type method task
V:40001080 Program SOFT "(other)" 🗸 🖉 EE_oo_TerminateTask
p;;
components trace Data Var List PERE SYStem other previous

The red S indicates that an intrusive breakpoint is used

TRACE32 PowerView						
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX EE_cpu_0 Window Help						
M 咪 ムム ↓ 4 ℃ と Ⅱ 巡 ? 校 ◎ 罰 遡 ■ 🐼 🗟 🚳 🧐 🧎 🖉						
🔄 B::List.auto						
🖹 Step 📑 Over 🛃 Diverge 🖌 Return 🖄 Up 🕨 Go 🛛 🖬 Break 🕅 Mode 🐻 📬 Find						
addr/line_code label mnemonic comment						
628{						
SV:40001080 182106F0 EE_oo_Te:e_stwu r1,-0x10(r1) ; r1,-16(r1)						
register EE_FREG np_flags;						
65 EE ORTI set service in(EE SERVICETRACE TERMINATETASK):						
sv:40001086 1C8D8080 e_add16i r4,r13,-0x7F80 ; r4,r13,-32640						
 E OS RESOURCE if the task still occupy resources						
E_OS_CALLLEVEL if called at interrupt level 🔹						
🕲 B::Break.List						
X Delete All O Disable All O Finable All O Tott / Method Store 9 Load 10 Set						
ress type method task						
V:40001080 Program SOFT "(other)" V 🖉 EE_oo_TerminateTask						
B::						
I components trace Data Var List PERE SYStem other previous						
SM0001000 Uncles terrinatic (other)						
autoarroa (Molo-munute Court) archhon ar niegrhouir						

Example for ARM9: Stop the program execution at the entry to the function Func_2 only if the taskF "main" is running (Onchip breakpoint).

🙆 B::Break.Set				
- address / express	sion			
Func_2			👻 📝 🗖 HLL	
type	options		implementation	
Program	EXclude	Temporary	Onchip 🔻	
ReadWrite	NOMARK	DISable	- action	
Read		DISableHIT	stop 🔻	
Write	DATA			
🔘 default		-		
Ok	Add	Delete	Cancel	
ProgramPace	memory / regis	ster / var		
ProgramEail				
 MemoryReadWrite 	- TASK		COUNT	
MemoryRead	"main"	•	1.	
MemoryWrite				
RegisterReadWrite	CONDition			
RegisterRead			I HLL	
RegisterWrite	- CMD			
			🔹 🗹 RESUME	
B::Break.List				
XDelete All O Disa	ble All 💿 Enable Al	l 🛛 🛛 Init 🖉 Im	ıpl 😰 Store 💈	🐉 Load 🚺 🔯 Set
a	ddress t	ypes impl	task TP "main"	Euro 2
	KI SOOSIZBC F			runc_2

TRACE32 PowerView ARM
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov eCos Window Help
H ⊨ ↓ √ Ċ ▶ II 郛 № ◎ ◎ ½ ≫
N Step N Over ↓ Next ✔ Return C Up ▶ Go II Break ﷺ Mode Find: addr/line source
135 Boolean Func_2 (Str_1_Par_Ref, Str_2_Par_Ref)
/* executed once */ /* Str_1_Par_Ref == "DHRYSTONE PROGRAM, 1'ST STRING" */ /* Str_2_Par_Ref == "DHRYSTONE PROGRAM, 2'ND STRING" */
Str_30 Str_1_Par_Ref; Str_30 Str_2_Par_Ref;
REG One_Thirty Int_Loc; Capital_Letter Ch_Loc;
B::
emulate trigger devices trace Data Var other previous
running HLL UP

Counters allow to stop the program execution on the *n* th hit of a breakpoint.

Software Counter

If the on-chip break logic of the core does not provide counters or if a Software breakpoint is used, counters are implemented as software counters.

Processing:



Each stop at a Counter breakpoint takes at least 1.ms. This is why the red S is displayed in the TRACE32 PowerView state line whenever the breakpoint is hit.

Example: Stop the program execution after the function sieve was entered 1000. times.

🔯 B::Break.Set			X
address / express	sion		
sieve		•	1 HLL 🗆 *
type Program ReadWrite Read Write default	options	Temporary DISable DISableHIT	method auto action stop
Ok	Add memory / regis	Delete	Cancel
ProgramFail	- TASK		COUNT
C MemoryReadWrite			- 1000.
 MemoryRead MemoryWrite 	MACHINE		CORE
RegisterReadWrite	- CONDition		
RegisterRead			🗹 HLL 🔲 AfteSte
RegisterWrite	- CMD		🗧 🗹 RESUME

🕲 B::Break.List							x
X Delete All O Disa	ble All 💿 Enable All	🛇 Init	Method	Store 28	🔀 Load	🙆 Set	
address	type	method	count				
F:400012A8	Program	SOFT	0./1000.	V Ø	sieve		*
	•					1	E ai

Break.Set sieve /COUNT 1000.

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX Window Help	
H ⊯ ™ ↑ 4, 5 ▶ II 🕅 5 16 ◎ ≅ 🛄 🙉 🗠 🦓 🧃 👘 🔭	
N Step N Over ↓ Diverge ✔ Return ♥ Up Go II Break M Mode C T	
addr/line source	
char flags[SIZE+1];	
int sieve() /* sieve of erathostenes */	
register int i, primz, k;	
ınt anzahl;	
668 anzahl = 0;	
🕲 B::Break.List	
X Delete All O Disable All O Enable All O Init / Method., Store., SLoad., 1	
address type method count	The cur
F:400012A8 Program SOFT /5./1000. 🗸 🖉 sieve	value is
	in the B
B::	window
components trace Data Var List PERF other previous	
running S HLL UP a	
	The red

The current counter value is displayed in the **Break.List** window

The red S indicates an intrusive breakpoint

File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPCSXXX Window Help N K ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	TRACE32 PowerView
N k ↓ ↓ ↓ ↓ ↓ L K N K ↓ ↓ ↓ ↓ ↓ L K N Step N Over A Diverge A Diverge A Return C Up Go I Break Mode A dir Char flags[SIZE+1]; int sieve() /* sieve of erathostenes */ 664 register int i, primz, k; int anzahl; 668 anzahl = 0; * * B:: B:: B::	File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX Window Help
B:List	│ Ŋ ⊮ ムム│ ↓ ✔ Ċ│ ▶ Ⅱ│ │ ? ㎏│ ◎│ 圖│ & ⊗ ⊗ ◎│ 🕲 ≛ ≫│
H Step K Over ▲ Diverge ✓ Return C Up Go II Break ™ Mode ▲ addr/line source	
addr/11ne source char flags[SIZE+1]; int sieve() /* sieve of erathostenes */ 664 register int i, primz, k; int anzahl; anzahl = 0; & * B::Break.List Image: Store Store Delete All O Disable All O Enable All O Init Method Store Store Address type F:400012A8 Program SOFT 0./1000. V Sieve *	N Step N Over M Diverge K Return C Up Go II Break Mode A To T
664 int sieve() /* sieve of erathostenes */ 664 register int i, primz, k; int anzahl; 668 anzahl = 0;	char flags[SIZE+1];
004 register int i, primz, k; int anzahl; 668 anzahl = 0; Image: BisBreak.List Image: Disable All Image: Disable Al	int sieve() /* sieve of erathostenes */
668 anzahl = 0;	register int i, primz, k; int anzahl;
Image: State	668 anzahl = 0;
B::Break.List M Delete All ○ Disable All ◎ Enable All ◎ Init ② Method ② Store ② Load ③ Set addr ess type method count F:400012A8 Program Store B:::	
Image: Section of the section of th	🕲 B::Break.List
address type method count F:400012A8 Program SOFT 0./1000. ✓ Ø sieve Image: Soft in the second s	X Delete All O Disable All O Enable All Init Method Store Store
B::	address type method count F:400012A8 <mark>Program SOFT 0./1000. √ ② sieve</mark>
8::	
	B::
components trace Data Var List PERF other previous	components trace Data Var List PERF other previous
SF:400012A8 \\diabc\diabc\sieve stopped at breakpoint HLL UP at	SF:400012A8 \\diabc\diabc\sieve stopped at breakpoint HLL UP

On-chip Counter

The on-chip break logic of some cores e.g. MPC55xx provides counters. They are used together with Onchip breakpoints.

Example: Stop the program execution after the function sieve was entered 1000. times.

😻 B::Break.Set									
- address / express	sion								
sieve		•	🧘 🗆 HLL 🔲 *						
type Program ReadWrite Read Write default	options EXclude NoMark	Temporary DISable DISableHIT	method Onchip action stop advanced						
Ok © ProgramPass	Add memory / registe	Delete	Cancel						
ProgramFail	- TASK		COUNT						
MemoryReadWrite			- 1000.						
MemoryRead	MACHINE		CORE						
MemoryWrite									
RegisterReadWrite	- CONDition								
RegisterRead			🗹 HLL 🔲 AfteSter						
RegisterWrite	- CMD								
			RESUME						
		🕲 B::Break.List							• •
		X Delete All	Disable All 💿 Enable All	⊗ Init	A Method	Store	SLoad	🙆 Set	
		address	type	method	count				

Break.Set sie	e /COUNT	1000.	/Onchip
---------------	----------	-------	---------

The counters run completely in real-time. No current counter value can be displayed while the program execution is running. As soon as the counter reached its final value, the program execution is stopped.

The program execution is stopped at the breakpoint only if the specified condition is true.

CONDition breakpoints are always intrusive.

Processing:



Each stop at a CONDition breakpoint takes at least 1.ms. This is why the red S is displayed in the TRACE32 PowerView state line whenever the breakpoint is hit.

Example: Stop the program execution on a write to flags[3] only if flags[12] is equal to 0 when the breakpoint is hit.

😻 B::Break.Set			
address / express	sion		
flags[3]		-	🦹 🗹 HLL 🗆 *
type Program ReadWrite Read Write	options	Temporary DISable DISableHIT	method auto • action stop •
O default		-	
Ok	Add	Delete	Cancel
[]	memory / registe	r / var	
ProgramPass			👻 🔝 🗆 HLL
ProgramFail	TASK		COUNT
MemoryReadWrite			· 1.
MemoryRead	MACHINE		CORE
MemoryWrite			
RegisterReadWrite	- CONDition		
RegisterRead	flags[12]==0		🔽 HLL 🔲 AfteSter
RegisterWrite	- CMD		
			🗘 💌 RESUME

ſ	🕲 B::Break.List					×
	X Delete All O Disable All 🖲 Enal	ole All 🚫 Init	A Method 😰 Store	Load	. 🔯 Set]
	address type	method	condition	a		
	C:4000412B Write	ONCHIP	flags[12]==0	√ Ø	flags[3]	
						-
	•					▶

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX Window Help	
N M ™ ↑ 4 G D II 28 5 K3 20 58 m 10 68 @ @ 69 5 %	
N Step 🕨 Over 🎝 Diverge 🖋 Return 🖄 Up 🕨 Go 🔢 Break 💥 Mode 🐼 🖿 🖓	
670 for (1 = 0; 1 <= SIZE; flags[1++] = IRUE);	
6/2 for (1 = 0; 1 <= SIZE; 1++)	
674 if (flags[i])	
676 primz = i + i + 3; 677 k = i + primz;	
🕲 B::Break.List	
🗶 Delete All 🔘 Disable All 💿 Enable All 💿 Init 🖉 Method) 😨 Store 😰 Load 🔞 Set	
address type method condition a C:4000412B Write ONCHIP flags[12]==0 √ Ø flags[3] ▲	
6d B:Var. View flags[3] flags[12]	The red S indicates
• flags[12] = 0	an intrusive breakpoint
B::	
components trace Data Var List PERF other previous	
SE:400012E4 \\diabc\diabc\sieve+0x3C stopped by r/w breakpoint S HLL UP	

Var.Break.Set flags[3] /Write /VarCONDition flags[12]==0

Example: "Break-before-make" Read/Write breakpoints only

Stop the program execution at a write access to the variable mstatic1 only if flags[12] is equal to 0 and mstatic1 is greater 0.

Perform an assembler single step because the processor architecture stops before the write access is performed.

🔯 B::Break.Set				
- address / expres	sion			
mstatic1		🔻 🔒 🗹 HLL		
L				
type	options	-implementation		
Program	EXclude Tempora	ry auto 🔻		
ReadWrite	NOMARK DISable	- action		
Read	DISableH	IT stop 🔻		
Write	DATA			
Ø default		▼		
Ok	Add Delete	Cancel		
ProgramPace	memory / register / var			
ProgramEail				
ProgramPad	TACK	COUNT		
Momon/Road				
MemoryKedu		· .		
	- CONDition			
	$(f _{2}ac[12] = -0)$ $\Re(mathing 1>0)$			After Oters, she sales
	(hays[12]==0)@@(histatic1>0.)			AfterStep checke
C Register write		🚊 🗹 RESUME		
	😵 B::Brea	k.List All O Disable All O Enable A	NI 🛛 Init 🎾 Impl	Store SLoad
		address	types impl	condition a
	N :-	4A326DA84A326DAB	Vrite ONCHIP	(†lags[12]==0)&&(mstatic1>0.) A
			•	1 1.

Var.Break.Set mstatic1 /Write /VarCONDition (flags[12]==0)&&(mstatic1>0)
/AfterStep

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Perf Cov OMAP4430app Window Help	
▶ ♥ ヤ ↑ ◀ ⊄ ▶ Ⅱ ૹ 3 ₺? ◎ 副 闘 🖓 🖓 🚳 👘 🕇 🗞	
📰 B::List	
N Step Nover Lucation And Antice Ant	Find:
addr/line_source	
H 140 for (regvar = 0; regvar < 5; regvar++) mstatic1 += regvar*autovar;	
143 fstatic += mstatic1:	
145 fstatic2 = 2*fstatic:	
	-
Ser B::Var.View mstatic1 flags[12]	• 33
• flags[12] = 0	÷
•	E. ₹
🕲 B::Break.List	
X Delete All O Disable All O Enable All O Init // Impl Store Store	
Address types impl condition N:4A326DA84A326DAB Write ONCHIP (flags[12]==0)&&(mstatic1>0.) A 🔺
	 ►
B::	
trigger devices trace Data Var List other previous	
NSR:4A3260D8 \\demo\demo\func2+0x30 stopped byr/w breakpoint S HLL	UP

The red S indicates an intrusive breakpoint

It is also possible to write register-based or memory-based conditions.

Examples: Stop the program executions on a write to the address flags if Register R11 is equal to 1.

🔯 B::Break.Set		X	
address / express	sion		
flags	▼ [1 HLL 🗆 *	
type Program ReadWrite Read Write default Ok ProgramPass ProgramFail MemoryReadWrite MemoryWrite RegisterReadWrite RegisterRead RegisterWrite	options EXclude Temporary NoMark DISable DISableHIT DATA Add Delete memory / register / var TASK MACHINE CONDition Register(R11)==0x1 CMD	method auto action stop Cancel Cancel Cancel U COUNT I. CORE CORE	Switch HLL OFF -> TRACE32 syntax can be used to specify the condition
	<u> </u>		

; stop the program execution at a write to the address flags if the ; register R11 is equal to 1 Break.Set flags /Write /CONDition Register(R11)==0x1

; stop program execution at a write to the address flags if the long ; at address D:0x1000 is larger then 0x12345 Break.Set flags /Write /CONDition Data.Long(D:0x1000)>0x12345 **Example:** Stop the program execution if an register-indirect call calls the function func3.



😻 B::Break.Set SF:0x4	40001150 /DIALOG	1	
address / express	ion		
SF:0x40001150		-	🚹 🗆 HLL 🛛 *
type Program ReadWrite Read Write default	options	Temporary DISable	method auto • action stop •
	Add	Delete	Cancel
ProgramPass			- 🚊 🗆 HLL
ProgramFail	TASK		COUNT
MemoryReadWrite			- 1.
MemoryRead	MACHINE		CORE
MemoryWrite			
RegisterReadWrite	- CONDition		
RegisterRead	Register(PC)=	=ADDRESS.OFFSET(fu	nc 📃 HLL 🗹 AfteSter
© RegisterWrite	CMD		🗘 🗷 RESUME

ſ	🕲 B::Break.List							×
	X Delete All O Disable	All 🖲 Enable All	⊗ Init	🖉 Method 😰 Store 💈	🛿 Load 🚺 💕 Set			
	address	type	method	condition		a		
	F:40001150	Program	SOFT	Register(PC)==ADDRE	SS.OFFSET(func3)	A 🗸 🖉	main\31+0x8	~
								-
		•						▶

Break.Set main\31+0x8 /CONDition Register(PC)==ADDRESS.OFFSET(func3)
/AfterStep

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Probe Per	Cov MPC5XXX Window Help
┝ ┡ ムム ↓ √ Ċ ▶ Ⅱ ? ㎏ ⑳ 圖	🐼 🗟 🗟 🕲 🧘 🔑
B::List	
N Step N Over L Diverge & Return C Up	► Go 📕 Break 🕅 Mode 🚱 🔁 😳 Find:
addr/line code label mnemonic	comment
static int func3()	/* simple function */
SF:40000310 9421FFF8 func3: stwu r1,-0	x8(r1) ; r1,-8(r1)
SF:40000314 7C0802A6 mflr r0	
233 return 5:	UC(r1); r0,12(r1)
SF:4000031C 38600005 li r3,0x	5 ; r3,5
234 }	06(-1) . =0.12(-1)
SF:40000320 8001000C 1W2 r0,0X	v (r1) ; r0,12(r1)
B::Break.List	
🗶 Delete All 🔘 Disable All 💿 Enable All 🚫 Init	Store 🔁 Load 🔞 Set
address type method condition	
P:40001130 Program Sorr Register(P	
	н (
R···	
components trace Data Var List	PERF SYStem Step other previous
SF:40000310 \\diabc\func3	stopped at breakpoint MIX UP

The field CMD allows to specify one or more commands that are executed when the breakpoint is hit.

Example: Write the contents of flags[12] to a file whenever the write breakpoint at the variable flags[12] is hit.

OPEN #1 outflags.txt	/Create	;	open	the	file	for	writing
----------------------	---------	---	------	-----	------	-----	---------

👸 B::Break.Set			The specified command(s) is executed
address / express	address / expression		whenever the breakpoint is hit. With RESUME
flags[12] type Program ReadWrite Read Write Here default	options EXclude Temporary NoMark DISable DISableHIT DATA	MHLL *	ON the program execution will continue after the execution of the command(s) is finished.
Ok ProgramPass ProgramFail MemoryReadWrite MemoryWrite RegisterReadWrite RegisterRead RegisterWrite	Add Delete memory / register / var TASK MACHINE CONDition CMD [WRITE #1 "flags[12]=" %Decimal V	Cance	The cmd field in the Break.List window informs the user which command(s) is associated with the breakpoint. R indicates that RESUME is ON.
B::Break.List Delete All O Disat address C:40004134	ole All @ Enable All @ Init @ Method type method cmd Write ONCHIP WRITE #1]	ad ♥ Set ecimal Var.VALUE(flags[12]) R V Ø flags[12]

Var.Break.Set flags[12] /Write /CMD "WRITE #1 ""flags[12]="" %Decimal Var.VALUE(flags[12])" /RESUME

 It is recommended to set RESUME to OFF, if CMD starts a PRACTICE script with the command DO commands are used that open processing windows like Trace.STATistic.Func, Trace.Chart.sYmbol or CTS.List
because the program execution is restarted before these commands are completed.

TRACE32 PowerView	
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC	SXXX Window Help
H M ™ ↑ € 5 ▶ II 🕅 5 K3 ◎ 🗉 🖼 🔲 🐄 🚳 🚳	1 🕲 1 🌽
B:List	
🗎 附 Step 📑 Over 🎿 Diverge 🖌 Return 🖿 Up 🕨 Go 🗌	🛚 Break 🕅 Mode 🐼 📬 Find: diabc.c
addr/line_codelabelmnemonic	comment
SF:400012E4 3BFF0001 addi r31,r31,0x1	; i,i,1
SF:400012E8 48FFFFE4 b 0X400012CC	; .L516
672 for (i = 0; i <= SIZE; i++)	
SF:400012F0 2C1F0012 .L522: cmpwi r31,0x12	; i,18
SF:400012F4 41810050 bgt 0x40001344	; .L51/ (-)
B:Break.List	
X Delete áll O Disable áll O Enable áll O Init / Method Store	Pload Miset
address type method cmd	r r
C:40004134 Write ONCHIP WRITE #1 "flags[12]="	<pre>%Decimal Var.VALUE(flags[12]) R / Ø flags[12]</pre>
•	
R::	
components trace Data Var List PERF	SYStem Step Go other previous
SF:400012E4 \\diabc\diabc\sieve+0x3C	stopped by r/w breakpoint MIX UP

The state of the debugger toggles between running and stopped

CLOSE #1

; close the file when you are done

Display the result:

File Edit View Var Break Run Cl		
🐠 Run Script		
🛃 Edit Script		
Search for Script		
😰 Open File		
😒 Load File		
Type File	r	
111 Dump File	B::TYPE I:\EVB\omap\omap4430\PandaBoard\linux\training\outflags.txt	
🚳 Stop Command	1. of 150.	
Printer Settings	f]ags[12]=1 f]ags[12]=1	A
🛕 Window Print	flags[12]=0	
🐻 Window Screenshot to File	f]ags[12]=1	
	flags[12]=0 flags[12]=1	
	flags[12]=0 flags[12]=1	
	flags[12]=0	
	f]ags[12]=1	
	flags[12]=0 flags[12]=1	-
	(E d

The on-chip break logic of some cores allows to combine data accesses and instructions to form a complex breakpoint (e.g. ARM or PowerArchitecture).

Preconditions

- Harvard architecture.
- The on-chip break logic supports a logical AND between Program and Read/Write breakpoints.

Advantageous

- Program breakpoints on address ranges are possible.
- Read/Write breakpoints on address ranges are possible.

Example: Stop the program execution when the function sieve writes a 1 to variable flags[3]. (If your core does not support this feature, the **radio buttons** (MemoryWrite, MemoryRead etc.) are grey.)

🞯 B::Break.Set			
- address / expres	sion	▼ 👔 🛛 HLL	 1. Define the address (range) of the instructions here
type Program ReadWrite Read Write default Ok ProgramPass ProgramFail MemoryReadWrite MemoryWrite RegisterReadWrite RegisterRead RegisterRead	options EXclude Temporary NOMARK DISable DISableHIT DATA 1. Add Delete memory / register / var flags[3] TASK CONDition	implementation action stop → advanced Cancel Cancel COUNT 1. AlteSte COUNT 1.	 2. Select MemoryWrite 3. Define the address (range) for the MemoryWrite accesses 4. Define the data value for the MemoryWrite accesses
B::Break.List Control Diss addre F:400012A8-	able All Enable All Init Imp ss types imp -40001367 MemoryWrite ONCHIP	pl] 😨 Store] 😨 Lo data BYTE 0x1	ad Sieve flags[3]

Var.Break.Set sieve /VarWrite flags[3] /DATA.auto 1.

TRACE32 PowerView
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MPC5XXX Window Help
ヱ 耳 卜 (ト 三 三 2 ~ 2 ~ 2 ~ 2 ~ 2 ~ 2 ~ 2 ~ 2 ~ 2 ~
B:List
► Step ► Over ▲ Diverge ✔ Return ▲ Up ► Go ■ Break ☑ Mode Find: addr/line_source
670 for (i = 0 ; i <= SIZE ; flags[i++] = TRUE) ;
B 672 for (i = 0; i <= SIZE; i++)
674 if (flags[i])
🕲 B::Break.List
X Delete All O Disable All O Enable All O Init // Impl Store SLoad Store
F:400012A840001367 MemoryWrite ONCHIP BYTE Ox1 sieve flags[3]
₩ B::Var.View flags[3]
• flags[3] = 1
B::
I trigger devices trace Data Var List PERF other previous
SF:400012E8 \\diabc\diabc\sieve+0x40 stopped byr/w breakpoint HLL UP

Exclude	(Advanced users only, not available on all cores)		
	The breakpoint is inverted.		
	by the inverting logic of the on-chip break logic		
	 by setting the specified breakpoint type to the following 2 address ranges 0x0(start_of_breakpoint_range-1) (end_of_breakpoint_range+1)end_of_memory 		
	The EXclude option applies only to Onchip breakpoints.		
	If the on-chip breakpoint logic does not provide an inverting logic, the core has to provide the facility to set the specified breakpoint type on 2 address ranges.		

Stop the program execution when code outside of the function sieve writes 1 to the variable flags[3].

🔯 B::Break.Set				
- address / express	address / expression			
sieve			🔻 📑 🗹 HLL	
type	– options ––––		implementation —	
Program	EXclude	Temporary	auto 👻	
ReadWrite	NOMARK	DISable	action	
Read		DISableHIT	stop 👻	
Write	DATA			
🔘 default	1.	_	A advanced	
Ok	Add	Delete	Cancel	
	- memory / regist	er / var		
ProgramPass	flags[3]		🔻 📑 🗹 HLL	
ProgramFail				
MemoryReadWrite	TASK		COUNT	
MemoryRead		-	1.	
MemoryWrite				
RegisterReadWrite	CONDition			
RegisterRead			🗹 HLL 🔲 AfteStep	
RegisterWrite	- CMD			
			💲 🗹 RESUME	

😢 B::Break.List				
🗶 Delete All 🗿 Disable All 💿 Enable All 💿 Init	🖉 Impl	Store	😨 Load 📦 Set	
address types	impl o	options	data	
F:400012A840001367 MemoryWrite	ONCHIP F	EXclude	BYTE 0x1	sieve flags[3]
4				

Var.Break.Set sieve /VarWrite flags[3] /DATA.auto 1. /EXclude

[B::List]			
Step	• Over	Diverge	e 🖌 Return 🙋 Up 🕨 Go 🔢 Break 🖉 Mode 🛛 Find:
breakpoint	a	dr/line s	i = func25():
		651	p = func26();
	÷	653	for (j = 0; j < 10; j++)
		655	sieve();
		658]	}
		c	thar flags[SIZE+1];
E mW		664 {	int sieve() /* sieve of er
			int anzahl;
E mW		668	anzahl = 0;
E mW		670	<pre>for (i = 0 ; i <= SIZE ; flags[i++] = TRUE</pre>
E mW	÷	672	for (i = 0 ; i <= SIZE ; i++)
			د
		i ne fu	nction sieve is marked with Exclude memoryWrit

The following command allows to check how the option EXclude is implemented.

Break.List /Onchip

Inverting logic of on-chip break logic:

🕲 B::Break:List /Onchip	
🗶 Delete All 🗿 Disable All 💿 Enable All 💿 Init 🖉 Impl 😰 Store 😒 Load) 📦 Set	
address types impl options data	onchip resource
R:0000180000001BFF MemoryWrite ONCHIP EXclude BYTE 0x1	(func23+0x4)(main\26+0x23) flags[3]
4	▶

Two address range breakpoints:

🕲 B::Break.List /Onchip			
X Delete All O Disable All O Enable All O Init // Impl	😰 Store 🔁 Load 🔯 Set	1	
address types impl opt	10ns data	C.0.0.0.0.70001250 [flags[2]	onchip resource
C:70001450FFFFFFF MemoryWrite ONCHIP	BYTE 0x1	C:0x700014500xFFFFFFFF flags[3]	A
		1	

If your TRACE32 PowerView does not accept the option EXclude, delete all other Onchip breakpoints, to make sure that enough resources are available.

Break Run CPU Misc	1				
🔯 Set					
🔁 List					
🖉 Method					
X Delete All					
-T- Trigger Bus	🕲 B::Break.List				×
ar OnChip Trigger	X Delete All O Disable All Enable All	🛛 Init 🎾 Impl 😭 Store	. 😒 Load 🔞 Set		
Trigger Reset	Address types NR:4A3261D8 Program	impl options SOFT DISableHIT	data	func4	*
	N:4A326FAB4A326FAB Write	ONCHIP	BYTE 0x1	flags[3]	

address	Address of the breakpoint
types	Type of the breakpoint
impl	Implementation of the breakpoint or disabled
action	Action selected for the breakpoint (if not stop)
options	Option defined for the breakpoint
data	Data value that has to be read/written to stop the program execution by the breakpoint
count	Current value/final value of the counter that is combined with a breakpoint
condition A (AfterStep)	Condition that has to be true to stop the program execution by the breakpoint A ON: Perform an assembler single step before condition is evaluated
cmd (command) R (resume)	Commands that are executed after the breakpoint hit R ON: continue the program execution after the specified commands were executed
task	Name of the task for a task-aware breakpoint
	Symbolic address of the breakpoint

Break.List [/<option>]

List all breakpoints



Break.Delete <address> <address_range> [/<type>] [/<implem.>] [/<option>]</option></implem.></type></address_range></address>	Delete breakpoint
Var.Break.Delete <hll_expression> [/<type>] [/<implem.>] [/<option>]</option></implem.></type></hll_expression>	Delete HLL breakpoint

Enable/Disable Breakpoints



Break.ENable [<address> <address_range>] [/<option>]</option></address_range></address>	Enable breakpoint
Break.DISable [<address> <address_range>] [/<option>]</option></address_range></address>	Disable breakpoint

😢 B::Break.List							×
X Delete All O Disable All O Ena	able All 🛛 🛛 Init	Impl	Store	Section Load	🙆 Set		
address t	ypes im	ol opt	Tons	data			
NR:4A3261D8 P	rogram SO	FT DIS	ableHIT			func4	
NR:4A32659C P	rogram SO	FT				sieve\6	
N:4A326FAB4A326FAB	/rite ON(CHIP		BYTE 0	x1	flags[3]	-
	•						•



// AndT32 Fri Jul 04 13:17:41 2003

B::

```
Break.RESet
Break.Set func4 /Program /DISableHIT
Break.Set sieve /Program
Var.Break.Set \\diabp555\Global\flags[3]; /Write /DATA.Byte 0x1;
```

ENDDO

STOre *<filename>* **Break** Generate a script for breakpoint settings

Debugging of Optimized Code

A video tutorial about debugging optimized code can be found here:

support.lauterbach.com/kb/articles/debugging-optimized-code-in-trace32

HLL mode and MIX mode debugging is simple, if the compiler generates a continuous block of assembler code for each HLL code line.

If compiler optimization flags are turned on, it is highly likely that two or more detached blocks of assembler code are generated for individual HLL code lines. This makes debugging laboriously.

TRACE32 PowerView displays a tree button, whenever two or more detached blocks of assembler code are generated for an HLL code line.



tree button

The following background information is fundamental if you want to debug optimized code:

- In HLL debug mode, the HLL code lines are displayed as written in the compiled program (source line order).
- In MIX debug mode, the target code is disassembled and the HLL code lines are displayed together with their assembler code blocks (target line order). This means if two or more detached blocks of assembler code are generated for an HLL code line, this HLL code line is displayed more than once in a MIX mode source listing.

The expansion of the tree button shows how many detached blocks of assembler code are generated for the HLL line (e.g. two in the example below).

Display source listing, display HLL code lines only.

List.Mix /Track Display source listing, display disassembled code and the assigned HLL code lines.

The blue cursor in the MIX mode display follows the cursor movement of the HLL mode display (Track option).



📑 [B::List.Mix /Trac	k]				
Step	Over Vext	Return 🕹 U	ip 🕨 🕨 Go	II Break Mode Find	diabc.c
672	for	$i = 0; i \le SI$	ZE ; i++)	comment	
SF:400012EC SF:400012E0	3BE00000 .L	514: li 522: cmpwi	r31,0x0	; i,0 : i.18	
SF:400012F4	41810050	bgt	0x40001344	; .L517 (-)	
674	1	if (<mark>flags[i</mark>])	12 16204	
SF:400012F8 SE:400012FC	3D804000 398C4128	l1s addi	r12,0x4000 r12 r12 0x4128	; r12,16384 • r12 r12 16680	
SF:40001300	7D8CF8AE	lbzx	r12,r12,r31	; r12,r12,i	
SF:40001304	2C0C0000 41820034	cmpwi	r12,0x0	; r12,0	
31.40001500	41020034	{ .	0,40001550	,	
676 SE+4000130C	7D9FFA14	primz add	= 1 + 1 + 3; r12 r31 r31	• r12 i i	
SF:40001310	3BCC0003	addi	r30,r12,0x3	; primz,r12,3	
677 SE+40001314	7EBEE214	k = i	+ primz;	· k i primz	
678	TUTZIT	while	(k <= SIZE)	, K, F, PF 102	
SF:40001318	2C1D0012 .L	520: cmpwi	r29,0x12	; k,18	
3F.4000151C	41010010		0X40001338	, .[]]] (-)	
680 SE • 40001 320	30804000	lic	flags[k] =	FALSE;	
SF:40001324	398C4128	addi	r12,r12,0x4000	; r12,r12,16680	
SF:40001328	39600000	li	r11,0x0	; r11,0	
SF:4000132C	/ DOCE9AE	STDX	k += primz:	; TII,TIZ,K	
SF:40001330	7FBDF214	add	r29,r29,r30	; k,k,primz	
SF:40001334	4866666	a {	0x40001318	; .L520	
683	2000001	anzah]++;	, anabl anabl 1	
SF:40001556	3B9C0001 .L:	addi	F20,F20,0X1	; dfizdfii,dfizdfii,1	
672	for SPEE0001	(i = 0 ; i <= SI	ZE ; i++)		
SF:40001340	4BFFFFB0	b	0x400012F0	; .L522	
		}			
			III		• • · · ·

List.Hll

To keep track when debugging optimized code, it is recommended to work with an HLL mode and a MIX mode display of the source listing in parallel.

List.Hll List.Mix /Track

Please be aware of the following:

If a Program breakpoint is set to an HLL code line for which two or more detached blocks of assembler code are generated, a Program breakpoint is set to the start address of each assembler block.

B::List.HII		×
📕 Step	Over 📕 Next 🖌 Return 🗶 Up 🕨 Go 🔢 Break 💆 Mode 🛛 Find: diabc	.c
addr/line	source	
664	int sieve() /* sieve of erathostenes */	*
	register int i, primz, k; int anzahl;	
668	anzahl = 0;	
670	for (i = 0 ; i <= SIZE ; flags[i++] = TRUE) ;	
⊞ 672	for (i = 0 ; i <= SIZE ; i++)	
674	if (flags[i])	
676 677	primz = i + i + 3; $k = i + primz;$	
678	while (k <= SIZE)	

😵 B::Break.List		
X Delete All O Disable All Enable All O Init	🖉 Impl	Store SLoad 📦 Set
address types	impl	
F:400012EC Program	SOFT	sieve\8
F:4000133C Program	SOFT	sieve\8 -
4		► <u></u>

There are local buttons in the List window for all basic debug commands

📰 B: List		
Step Nove	er 🛃 Diverge 🖋 Return 🙋 Up 🕨 Go 🔢 Break 🕅 Mode Find:	diabc.c
addr/line_so	burce	
	register int i, primz, k; int anzahl;	*
668	anzahl = 0;	
670	for (i = 0 ; i <= SIZE ; flags[$i\text{++}$] = TRUE) ;	
.	for (i = 0 ; i <= SIZE ; i++)	
674	if (flags[i])	
676	primz = i + i + 3;	
6//	k = 1 + primz;	•

Step	Single stepping (command: Step)
Over	Step over call (command Step.Over).
Diverge	Exit loops or fast forward to not yet stepped code lines. Step.Over is performed repeatedly.

More details on Step.Diverge

TRACE32 maintains a list of all assembler/HLL lines which were already reached by a Step. These reached lines are marked with a slim grey line in the List window.



The following command allows you to get more details:

List.auto /DIVERGE
[B::List /DIVERGE]	
📕 Step 📑 Over 🛃 Diver	rge 🖌 Return 🗶 Up 🕨 Go 🔢 Break 🖉 Mode 🛛 Find:
s state i addr/line	source
h stop 664	<pre>int sieve()</pre>
h done 668	anzahl = 0;
h done 670	<pre>for (i = 0 ; i <= SIZE ; flags[i++] = TRUE) ;</pre>
h done [672 672	for (i = 0 ; i <= SIZE ; i++) for (i = 0 ; i <= SIZE ; i++)
hit 674	if (flags[i])
676 677 678	{
680 681	<pre>flags[k] = FALSE; k += primz;</pre>
683	anzahl++; }
target 687 688	return anzahl;

Drag this handle to see the DIVERGE details

[B::List /DIVERGE]						
Step Nover	je 🖌 🖋 Return	🕑 Up	Go	Break	Mode	Find:
s state i addr/line	code 1	abel m	nemonic		comm	ient
a stop 602 a stop SF:40001148 a done SF:4000114C a done i SF:40001150 a stop SF:40001154	816D80C4 7D6803A6 4E800021 7C7F1B78	j = (l b m	*funcptr) wz utlr lrl m	<mark>();</mark> r11,-0x7F3 r11 r31,r3	3C(r13) ; j,	; r11,funcptr(r
a done 604 a done SF:40001158 a done SF:4000115C a done SF:40001160 a done SF:40001164 stop SF:40001168	7FE3FB78 38800002 38A00003 4BFFF23D 7C7F1B78	j = f "]] b m ""	unc5((in Ir i i i I r	t) j, (cha r3,r31 r4,0x2 r5,0x3 0x400003A0 r31,r3	ur) 2, (lo ; r3 ; r4 ; r5) ; fu ; j,	ng) 3); ;3 ;3 nc5 r3

Column layout	
S	Step type performed on this line a: Step on assembler level was started from this code line h: Step on HLL level was started from this code line
state	 done: code line was reached by a Step and a Step was started from this code line. hit: code line was reached by a Step. target: code line is a possible destination of an already started Step, but was not reached yet (mostly caused by conditional branches). stop: program execution stopped at code line.
i	indirect branch taken (return instructions are not marked).

Example 1: Diverge through function sieve.

1. Run program execution until entry to function sieve.

B::List /DI	VERGE]											
Step	Nover	Diverge	e 🖌 🖋 Return	C Up	► Go	II Break	Mode	Find:	sieve	d		
s state	i add	r/line s	ource			^						
		C	har flags	[SIZE+1];						~		
		i	int sieve()			/* sieve	of era	athostene	25 *		
		664 {		Prog	ram Address							
			in		ill							
				🔮 Breal	kpoint							
		668	an	zah 😢 <u>B</u> real	kpoints	•						
		670	fo	r 🌔 🚟 Displ	ay <u>M</u> emory	1ags	[i++] =	TRUE);			
	_	672	c.,	Book	mark							
	+	6/2	TO	r 🕻 🚮 Togg	jle Bookmark	++)						
		674		🔹 Set P	C <u>H</u> ere							
		676		🛃 Edit 🗄	Source	4.4	2.					
		677		🔒 View	Info	mz;	2,					
		678	r		whit le	(k <= SI	ZE)					
			< 🖂 🗐	B::List /DIVERG	E							[
,		-		Step	Over	Diverge 🖌	Return	t Up	► Go	II Brea	ak 🕅 Mode	Find: sie
			S	state i	addr/	line source	e					
						char 1	Flags [SIZE	+1];				
						int s	ieve()				/* sieve	of erath
				stop		664 {						
	·						registe	r int	ı, prımz	, k;		
stop indi	cates that	at the					The driz	arri,				
program	executio	n was				668	anzahl	= 0;				
stopped a	at this co	ode line	, III			670	for (i	= 0 :	i <= SI	ZE : fla	as[i++] =	TRUE) :
2.000000								. ,				, ,
					+	672	for (i	= 0;	i <= SI	ZE ; i++)	

2. Start a Step.Diverge command.

	📰 B::List /DI	VERGE							
h indicates that a Step	Step	🕨 🕨 🕹 🕹	likerg	e 🗸 Return	🔁 Up	Go 📔	🛛 🛛 🖉 Break	Mode	Find: sie
command in HLL mode was	s state	i ado	dr/line	ource	T75+11.				
started in this line			ľ	inar rraysį.	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,				
	h ston		664	nt sieve()				/* sieve	of erath
	in scop		001	reg int	ister int anzahl;	i, primz,	k;		
>	hit		668	anza	ahl = 0;				
hit indicates that this			670	for	(i = 0 ;	; i <= SIZ	E ; flags[i++] =	TRUE);
code line was reached by		+	672	for	(i=0;	i <= SIZ	E ; i++)		
Step command			674	ł	jf (flags[i])		
			676		۰.	primz	= i + i +	3;	
			678			κ = 1 while	+ primz; (k <= SIZ	Έ)	
	1			•					

3. Continue with Step.Diverge.

	B::List /DI	VERGE								- c
	► Step	🕨 🕨 🕹	Diverg	e 🖌 Return	Ċ Up	Go	II Break	Mode	Find:	siev
	s state	i ado	dr/line s	ource						
	h stop		664 (har flags[S nt sieve() regi int	IZE+1]; ster int anzahl;	i, primz,	k;	/* sieve	of er	atho
	h done		668	anza	h1 = 0;					
done indicates that the	hit		670	for	(i = 0;	; i <= SIZ	E ; flags	[i++] =	TRUE);
code line was reached by		±	672	for	(i = 0 ;	; i <= SIZ	E ; i++)			
a Step command and that			674	ĩ	if (flags[i	1)			
a Step command was started from this code line			676 677 678		l	primz k = i while {	= i + i + + primz; (k <= SIZ	3; ZE)		
	1			•						

	B::List /DI	VERGE					c
	Step	🕨 Over	verge 🖌 🖋 Return	🔁 Up 🚺 🕨 Go	Break	Mode F	ind: siev
	s state	i addr/li	ne source				
The tree button indicates that two or more detached blocks of	h stop	6	char flags[S int sieve() { regis	IZE+1]; ster int i, primz, anzahl;	k;	/* sieve of	eratho
assembler code are	h done	6	58 anzal	hl = 0;			
code line	hit	6	70 for	(i = 0 ; i <= SIZ	E ; flags <mark>[</mark>	i++] = TR	UE);
			72 for	(i = 0 ; i <= SIZ	E ; i++)		
		6	74	if (flags[i	1)		

4. Continue with Step.Diverge.



assembler code is marked as hit



5. Continue with Step.Diverge.



The not-reached code line is marked as **target**

6. Continue with Step.Diverge (several times).

	B::List /DIVERG	SE]		
	N Step	Over addr	Diver	rge ✔ Return Ć Up ► Go 🛛 Break 🕅 Mode
	 h stop	duu	664	<pre>int sieve() /* sieve { register int i, primz, k; int anzahl;</pre>
	h done		668	anzahl = 0;
	h done		670	for (i = 0 ; i <= SIZE ; flags[i++] =
All code lines are now	h done target	Γ	672 672	for (i = 0 ; i <= SIZE ; i++) for (i = 0 ; i <= SIZE ; i++)
either marked as done ,	h done		674	if (flags[i])
hit or target	h done h done h done		676 677 678	primz = i + i + 3; k = i + primz; while (k <= SIZE)
	h done hit		680 681	ہ flags[k] = FALs k+= primz:
	target		683	} anzahl++; }
	target		687 688	return anzahl; }
				K

7. Continue with Step.Diverge.

	B::List /DIVER	GE]		
	Step	🖬 Over	Diverg	rge) 🗸 Return 🜔 🕑 🕨 🕨 Go 🛛 🔢 Break 🛛 🖉 Mode 🛛 Find
	<u>s state i</u> h stop	addr/	<u>664</u>	<pre>char flags[SIZE+1]; int sieve() /* sieve of { register int i, primz, k; int anzahl;</pre>
	h done		668	anzahl = 0;
	h done		670	for (i = 0 ; i <= SIZE ; flags[i++] = TRUE
	h done target	Γ	672 672	for (i = 0 ; i <= SIZE ; i++) for (i = 0 ; i <= SIZE ; i++)
	h done		674	if (flags[i])
A code line former marked	h done h done h done		676 677 678	primz = i + i + 3; k = i + primz; while (k <= SIZE)
when it is reached	h done h done		680 681	flags[k] = FALSE; k += primz;
	hit		683	anzahl++;
	target		687 688	<pre>} return anzahl; } </pre>
	1			

When all reachable code lines are marked as **done**, the following message is displayed:

B:: no more reachable targets from this code						
trigger devices trace Data Var List						
SF:40001284 \\diabc\diabc\main+0x228						

The **DIVERGE marking** is cleared when you use the **Go.direct** command without address or the **Break** command while the program execution is stopped.

Example 2: Exit a loop.

DIVERGE marking is done whenever you single step.

If all code lines of a loop are marked as **done/hit**, a Step.Diverge will exit the loop

	B::List /DIV	ERGE		
	Step	Nover	Diverge	e 🗸 Return 🗶 Up 🕨 Go 🔢 Break 🖉 Mode Find:
	<u>s state</u>	i add	r/line s	register int i, primz, k;
	h dana			int anzahl;
	h done		670	anzani = 0; for $(i = 0, i = 5775, i = flore[iv] = TRUE)$;
ing is	h done	_	672	for $(1 = 0; 1 <= SIZE; 11ags[1++] = 1K0E)$,
you	target	L	672	for $(i = 0; i <= SIZE; i++)$
	h done		674	if (flags[i])
	h done h done		676 677	primz = i + i + 3; k = i + primz;
to ad as	h done		678	while (k <= SIZE)
50 85	h done hit		681	k += primz;
I	target		683	anzahl++;
				}
	target		687 688 }	return anzahl;
			co1 i	nt background() /* job for backgro
			09T {	register long count1, count2;
			1	in the second seco
	B::List /DI	VERGE]		
	► State	i add	Diverge	e ∉Return CUp ►Go II Break WMode Find:
			c	<pre>char flags[SIZE+1];</pre>
			i	int sieve() /* sieve of erath
	h stop		664 {	register int i, primz, k;
	h done		668	int anzani;
	h done		670	for (i = 0 : i <= SIZE : flags[i++] = TRUE) :
	h done	г	672	for (i = 0 ; i <= SIZE ; i++)
	target	L	672	for (i = 0 ; i <= SIZE ; i++)
	h done		6/4	1 (flags[1]) {
	h done		676 677	primz = 1 + 1 + 3; k = i + primz; while (k = 5775)
	h done		680	flags[k] = FALSE:
	h done		681	k += primz;
	hit		602	anzahl
	me		683	}
			683	} }



Step [<count>]Single stepStep.Change <expression>Step until <expression> changesStep.Till <condition>Step until <condition> becomes true,
<condition> written in TRACE32 syntaxVar.Step.Change <hll_expression>Step until <hll_expression> changesVar.Step.Till <hll_condition>Step until <hll_condition> becomes true,
<hll_condition> as allowed in used programming
language

Step 10.

Step.Change Register(R11)
Step.Till Register(R11)>0xAA
Var.Step.Change flags[3]
Var.Step.Till flags[3]==1

Step.Over	Step over call
Go [<address> <label>]</label></address>	Start program execution
Go.Next	Set a temporary breakpoint to the next code line and start the program execution
Go.Return	Set a temporary breakpoint to the return instruction and start the program execution
Go.Up [<level> <address>]</address></level>	Run program until it returns to the caller function

Program Counter Sampling

Task: get the percentage of time used by a high-level language function.

B::PERF.ListFunc							
Setup 🔛 Config 📭 Goto	. 📃 Detailed 🔍 View 👔	Profile 🛛 🛇	Init O DIS	able 💿 A	rm		
cover	age: 54.546% runtime	: 99.432%	covtime:	54.546%			
name	ratio	1% 2%	5%	10%	20%	50%	100
sieve	54.491%						
(other)	29.341%						
func10	7.784%						=
func9	1.796%						-
tunc13	1.197%						
main	1.197%						
tuncl	0.598%	+					
tunc2	0.598%	+					
tunc2a	0.598%	+					
tunc2c	0.598%	+					
func2d	0.598%	+					
funcl1	0.598%	+					
func1/	0.598%	+					Ψ.
		III					

Measurement procedure: The Program Counter is sampled periodically. This is implemented in two ways.

- **Snoop:** Processor architecture allows to read the Program Counter while the program execution is running.
- **StopAndGo:** The program execution is stopped shortly in order to read the Program Counter.

Steps to be taken:

1. Open the PERF configuration window.



		1		
	/			
🔑 B::PERF				
METHOD				commands
🔘 BusSnoop 🧕 🧕	StopAndGo 💿 Tr	ace 🔘 Snoop	O DCC	ListProgram
				ListTREE
state	Mode	scans done	- Sort	ListLine
OISable	PC		© OFF	ListFunc
OFF	© TASK	- curr.scan	O Address	ListModule
Arm	MEMory		SYmbol	ListFuncMod
	© PCTASK	- runtime	Ratio	ListLABEL
commands	PCMEMory			ListRange
Program		snoops/s	- SnoopAddress -	ListS10
⊗ Init	options		C:0x0	ListS100
E List	MMUSPACES	- snoop fails	- SnoopSize	ListS1000
RESet	STREAM		Byte 👻	ListS10000
AutoArm	L			ListDistriB
	RunTime	🖵 perf program file		ListVarState
			🛃 🔊	ListTASK

PERF.state

Display PERF configuration window

The PERF METHOD **Snoop** is automatically selected, if the processor architecture supports reading the Program Counter while the program execution is running.

The default METHOD for all other processor architectures is **StopAndGo**.

Remarks on the StopAndGo method

StopAnd Go means that the core is stopped periodically in order to get the actual Program Counter.

STREAM ON	The software running on the TRACE32 debug hardware initiates the periodic stops. This has the following advantages:
	Low intrusive (approx. 50. to 100.us)
	More samples per second are possible
STREAM OFF	The software running on the host initiates the periodic stops.
	More intrusive (1 ms in a worst case scenario)
	Less samples per second are possible

The display of a red **S** in the TRACE32 state line indicates that the program execution is periodically interrupted by the sample-based profiling.

B::							
trigger devices	trace	Data	Var	other	previo	ous	
	runnii	ng		S	MIX	UP	

TRACE32 tunes the sampling rate so that more the 99% of the run-time is retained for the actual program run (runtime). The smallest possible sampling rate is nevertheless 10 (snoops/s).

➢ B∷PERF				
BusSnoop) StopAndGo 🛛 🔘 Tr	ace 🔘 Snoop	© DCC	commands ListProgram
state	Mode	scans done	Sort	ListLine
OISable	PC		OFF	ListFunc
© OFF	© TASK	- curr.scan	O Address	ListModule
Arm	C MEMory		© sYmbol	ListFuncMod
L]	O PCTASK	- runtime	Ratio	ListLABEL
commands	PCMEMory	99.901%		ListRange
🛃 Program		- snoops/s	SnoopAddress	ListS10
O Init	options	11.	C:0x0	ListS100
🗾 List	MMUSPACES	– snoop fails –––	- SnoopSize	ListS1000
RESet	STREAM		Byte 👻	ListS10000
🗹 AutoArm				ListDistriB
[]	RunTime	perf program file		ListVarState
				ListTASK
		L		

2. Enable the sample-based profiling by selecting the OFF state.

METHOD				commands -
🔿 BusSnoop 🛛 🧕)StopAndGo 🛛 🔿 Tr	ace 💿 Snoop	O DCC	ListProgram
				ListTREE
state	Mode	scans done	Sort	ListLine
🔘 DISable	PC		OFF	ListFunc
OFF	C TASK	- curr.scan	Address	ListModule
🖱 Arm	C MEMory		SYmbol	ListFuncMod
	© PCTASK	- runtime	Ratio	ListLABEL
commands	PCMEMory			ListRange
Program		snoops/s	- SnoopAddress	ListS10
⊗ Init	options		C:0x0	ListS100
🗾 List	MMUSPACES	- snoop fails	- SnoopSize	ListS1000
RESet	STREAM		Byte 👻	ListS10000
🗷 AutoArm				ListDistriB
	RunTime	perf program file		ListVarState
			🚺 🔊	ListTASK

PERF.OFF

Enable the sample-based profiling

3. Open a result window by pushing the ListFunc button.

🔑 B::PERF								
METHOD				commands				
🔘 BusSnoop 🧕	StopAndGo	© Trace ○ Snoop	O DCC	ListProgram				
I				ListTREE				
- state	Mode	scans done	Sort	ListLine				
O DISable	PC		OFF	ListFunc				
OFF	© TASK	- curr.scan	O Address	ListModule				
O Arm	O MEMory		<u>avı</u>					
	© PCTASK	B::PERF.ListFunc						
commands	O PCMEMo	🌽 Setup 🔡 Config	📭 Goto 📃	Detailed 🔍 View 📗	Profile	⊗ Init	O DISable	O Arm
Program		name		ratio	194	2%	5%	1.0%
⊗ Init	– options –	(other)		0.000	%	2/0	370	10/0
E List	MMUSPA	func0 func1		0.000	% %			
RESet	STREAM	func2		0.000	%			=
AutoArm		func2a func2b		0.000	*6 *6			
	- RunTime -	func2c		0.000	%			
		func2a func3		0.000	76 Ж			
		func4		0.000	8			
		func6		0.000	76 %			
		func7		0.000	*			-
					/0			E a

PERF.ListFunc

Open an HLL function profiling window

4. Start the program execution and the sampling.

E:PERF.ListFunc		
🥬 Setup 🔡 Config 📭 Goto 📑 Detailed 🔍	View Profile 🛛 🛇 Init 🛛 O DISable 🔍 👁 Arm	
coverage: 54.546%	runtime: 99.432% covtime: 54.546%	
name	atio 1% 2% 5% 10% 20%	50% 100
sieve	54.491%	
(other)	29.341%	
func10	7.784%	
func9	1.796%	=
func13	1.197% -	
main	1.197% -	
func1	0.598% 🗲	
func2	0.598% +	
func2a	0.598% +	
func2c	0.598% +	
func2d	0.598%	
func11	0.598% 🗲	
func17	0.598% 🗲	-
		·

In-depth Result

Push the Detailed button, to get more detailed information on the result.

B::PERF.ListFunc								×
🌽 Setup 🔡 Config 📭 Goto 🗾 Detailed	View 📗	Profile 🛛 🛛 🛇	Init O DI	Sable 🛛 🔿 A	rm			
coverage: 100.000%	runtime:	99.994%	covtime:	100.000%				
name	ratio	1% 2%	5%	10%	20%	50%	100	
sieve	96.332%						_	
main	3.633%							m
(other)	0.034%	+						
START	0.000%							
background	0.000%							Ŧ
							F	

B::PERF.ListFu	unc ALL							×
🌽 Setup	Config	to 🗾 Detail	ed 🔍 View	v 🛛 🔟 Profi	le 🛛 🛇 Init	O DISable	O Arm	
name	time	watchtime	ratio	dratio	address		hits	
sieve	73.455s	76.409s	96.133%	75.000%	P:A10005F	8A100063F	7434.	
main	2.915s	76.409s	3.814%	25.000%	P:A10004A4	4A10005F7	295	· n
(other)	39.524ms	76.409s	0.051%	0.000%			4.	
START	0.000us	76.409s	0.000%	0.000%	P:A100000	0A1000009	0.	
•							•	

PERF.ListFunc ALL

Open a detailed HLL function profiling window

name	Function name			
time	Time in function			
watchtime	Time the function is observed			
ratio	Ratio of time spent by the function in percent			
dratio	Similar to Ratio, but only for the last second			
address	Function's address range			
hits	Number of samples taken for the function			

(other)

TRACE32 assigns all samples that can not be assigned to a high-level language function to **(other)**. Especially if the ratio for (other) is quite high, it might be interesting what code is running there. In this case pushing the button **ListLABEL** is recommended.



PERF.ListLABEL

Open a window for label-based profiling

If OS-aware debugging is configured (refer to "**OS-aware Debugging**" in TRACE32 Concepts, page 34 (trace32_concepts.pdf)), TASK information can be sampled.

Steps to be taken:

1. Open the PERF configuration window.



B::PERF				
METHOD				commands —
💿 BusSnoop 🛛 🧕)StopAndGo 🛛 🔘 Tr	ace 💿 Snoop	O DCC	ListProgram
				ListTREE
state	Mode	scans done	Sort	ListLine
DISable	PC		OFF	ListFunc
OFF	C TASK	- curr.scan	Address	ListModule
🛇 Arm	C MEMory		SYmbol	ListFuncMod
	© PCTASK	- runtime	Ratio	ListLABEL
commands	PCMEMory			ListRange
Program		- snoops/s	- SnoopAddress -	ListS10
⊗ Init	options		C:0x0	ListS100
E List	MMUSPACES	– snoop fails –	- SnoopSize	ListS1000
RESet	STREAM		Byte 👻	ListS10000
V AutoArm				ListDistriB
	RunTime	👝 perf program file		ListVarState
			🚺 🔊	ListTASK



Since every OS has a variable that contains the information which task/process is currently running, this variable has to be sampled while the program execution is running in order to perform TASK sampling.

TRACE32 fills the following fields when TASK mode is selected:

- the SnoopAddress field with the address of the variable.
- the SnoopSize field with the size of the variable.

The PERF METHOD **Snoop** is automatically selected, if the processor architecture supports reading physical memory while the program execution is running. For details refer to "**Run-time Memory Access**" in TRACE32 Concepts, page 45 (trace32_concepts.pdf)).

The default METHOD for all other processor architectures is **StopAndGo**.

PERF.Mode TASK

3. Enable sample-based profiling by switching to OFF state and open the result window by pushing the ListTask button.

₽ B::PERF								
METHOD				- commands				
🔘 BusSnoop 🔘	StopAndGo 🛛 🔘 Tra	ice 💿 Snoop	O DCC	ListProgram				
L				ListTREE				
state	Mode	scans done	Sort	ListLine				
O DISable	© PC		OFF	ListFunc				
OFF	TASK	- curr.scan	Address	ListModule				
© Arm	C MEMory		🔿 sYmbol	ListFuncMod				
	PCTASK	- runtime	Ratio	ListLABEL				
commands	PCMEMory			ListRange				
Program		- snoops/s	- SnoopAddress -	ListS10				
O Init	options		SD:0x400059I	ListS100				
E List	MMUSPACES	 snoop fails 	- SnoopSize	ListS1000				
RESet	STREAM		Byte 🔻	ListS10000				
AutoArm				ListDistriB				
	RunTime	perf program file		ListVarState				
				ListTASK				
B::PERF.L	istTASK							
Setup	Config	to	View IIII Pr	ofile 🛛 🛇 Init 🕻	O DISable	Arm		
name TASKE			ratio 1%	2%	5% 10%	20%	50%	100
TASKD								<u></u>
TASKC								
TASKA								
TASK4								
TASK2								
TASK1								
NO TASK								-
								►

PERF.OFF PERF.ListTASK Enable the sample-based profiling

4. Start the program execution and the sampling.

B::PERF.ListTASK											
Setup 🔛 Config 🕞 Goto 📑 Detailed	🔍 View 📗	Profile	⊗ Init O DI	Sable 🛛 🔘	Arm						
runtime: 76.397%											
name	ratio	1% 2	2% 5%	10%	20%	50%	100				
TASK0	92.391%						-				
TASK2	3.261%										
TASK3	2.174%										
TASKE	1.087%	-									
TASK4	1.087%	-									
TASKD	0.000%										
TASKC	0.000%										
TASKB	0.000%										
TASKA	0.000%										
TASK1	0.000%										
NO_TASK	0.000%						-				
		III									