

Trace Tutorial

Release 02.2024



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Debugger Getting Started	Þ
Trace Tutorial	1
History	3
About the Tutorial	3
What is Trace? Trace Use Cases	3 4
Trace Methods	5
Simulator Demo	6
Trace Configuration	7
Trace Recording	8
Displaying the Trace Results	10
Trace List	10
Displaying Function Run-Times	13
Graphical Charts	13
Numerical Statistics and Function Tree	14
Duration Analysis	15
Distance Analysis	16
Variable Display	17
Track Option	18
Searching Trace Results	19
Trace Save and Load	20

Version 04-Mar-2024

History

18-Jun-21 New manual.

About the Tutorial

This tutorial is an introduction to the trace functionality in TRACE32. It shows how to perform a trace recording and how to display the recorded trace information.

For simplicity, we use in this tutorial a TRACE32 Instruction Set Simulator, which offers a full trace simulation. The steps and features described in this document are however valid for all TRACE32 products with trace support.

The tutorial assumes that the TRACE32 software is already installed. Please refer to **"TRACE32 Installation Guide**" (installation.pdf) for information about the installation process.

Please refer to "**Debugger Tutorial**" (debugger_tutorial.pdf) for an introduction to debugging in TRACE32 PowerView.

What is Trace?

Trace is the continuous recording of runtime information for later analysis. In this tutorial, we use the term trace synonymously with core trace. A core trace generates information about program execution on a core, i.e. program flow and data trace. The TRACE32 Instruction Set Simulator used in this tutorial supports a full trace simulation including the full program flow as well as all read and write data accesses to the memory. A real core may not support all types of trace information. Please refer to your **Processor Architecture Manual** for more information.

Trace is mainly used in the following cases:

- 1. Understand the program execution in detail in order to find complex runtime errors more quickly.
- 2. Analysis of the code performance of the target code
- 3. Verification of real-time requirements
- 4. Code-coverage measurements

TRACE32 supports various trace methods. The trace method can be selected in the **Trace** configuration window, which can be opened from the menu **Trace** > **Configuration...**

🔑 B::Trace		
Onchip	• Analyzer • CAnalyzer • HAnalyzer • Integrator • Probe • Integrator • Integrator	

If a trace method is not supported by the current hardware/software setup, it is greyed out in the trace configuration window. **NONE** means that no trace method is selected.

We use in this tutorial the trace method **Analyzer**. Please refer to the description of the command **Trace.METHOD** for more information about the different trace methods.

We use in this tutorial a TRACE32 Simulator for Arm. The described steps are however valid for the TRACE32 Simulator for other core architectures.

To load a demo on the simulator, follow these steps:

- 1. Start the script search dialog from the menu File > Search for scripts...
- 2. Enter in the search field "compiler demo"

🛉 Search for scri	ipts							
Search	Selection	Manuals						
Example sear	ch: CortexA5? Flas	h						
compiler der	no		~	💥 🎁 Searc	h 25 demo files fo	ound.		
Filter								
None	⊖ Chip ⊖ Board							
	•							
Search for ne	west scripts at h	ttps://www.laute	rbach.com/scrip	ots.html				
		1	· · · ·					
CONFIG	te: Tree view	LIST CONFIG						
•	eg. Thee them							
Title				Chip)	Board		
Load the Pos	sition Indepen	dent Sieve De	emo	-		-	^	N
Load the Pos	sition Indepen le on a Mistra	dent SIM stin	nulus code	-	3530	_ Mistral		
Load the Dos	sition Indepen	dent Midi Der	no	UMAP	0000	MISCIAI		
Code Coverad	ne Example	dent minur bei		_		-		
Code Coveraç GNU C Examp	e for FLASH			EPXA	1 EPXA10 STM32F	:		
GNU C Examp	le for SRAM			ARM9	20T ARM966E-S E			
Load the Pos	sition Indepen	dent ITM stin	nulus Code	-		-	×	1
<							> .;	

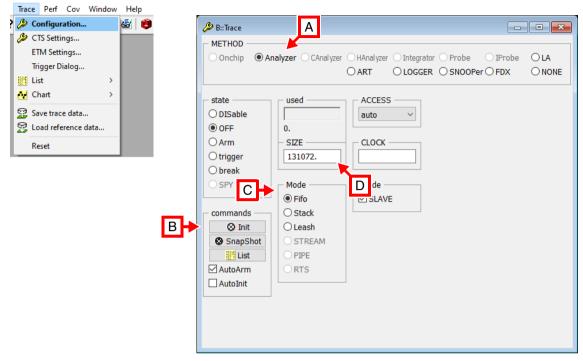
3. Select a demo from the list with a double click, a **PSTEP** window will appear. Press the "Continue" button.

[B::wr.CE]).PSTEP "C:\T32\demo\arm\c	ompiler\qnu\demo	sram.cmm	']				- • ×
🔊 Step	🛞 Over 🖉 Up	Dentinue	∰ Stop	🗐 Enddo	🗐 Skip	🕰 Macros	🕌 Edit	Breakpoints
	;; \$Id: demo_sram.cr	nm 17069 2021	-01-28 1	8:42:07z h	lohn \$			^
34 35 37	LOCAL ¶m ENTRY %LINE ¶m ¶m=STRing.LoWef WinCLEAR AREA.CLEAR	R("¶m")						_
39 40 41 42	SYStem.RESet Break.RESet TRANSlation.RESet symbol.RESet MAP.RESet							v

We will use here the demo "GNU C Example for SRAM".

In order to set up the trace, follow these steps:

1. Open the menu **Trace > Configuration...** The trace method Analyzer **[A]** should be selected per default. If this is not the case, select this trace method



- 2. Clear the contents of the trace buffer by pressing the **Init** button **[B]**.
- 3. Select the trace operation mode [C].

In mode **Fifo**, new trace records will overwrite older records. The trace buffer includes thus always the last trace cycles before stopping the recording.

In Mode **Stack**, the recording is stopped if the trace buffer is full. The trace buffer always includes in this case the first cycles after starting the recording.

Mode **Leash** is similar to mode **Stack**, the program execution is however stopped when the trace buffer is nearly full.

TRACE32 supports other trace modes. Some of these modes depend on the core architecture. Please refer to the documentation of the command **Trace.Mode** for more information.

We will keep here the default trace mode selection, which is Fifo.

4. The **SIZE** field **[D]** indicates the size of the trace buffer. As we are using a TRACE32 Simulator, the trace buffer is reserved by the TRACE32 PowerView application on the host. It is thus possible to increase the size of this buffer. If a TRACE32 trace hardware is used with a real chip, the size of the trace buffer is limited by the size of the memory available on the trace tool.

In order to have a longer trace recording, we will set the trace buffer size to 10000000.

The same configuration steps can be performed using the following PRACTICE script:

```
Trace.METHOD Analyzer
Trace.Init
Trace.Mode Fifo
Trace.SIZE 10000000.
```

Trace Recording

Press the Go button to start the program execution.

1	TRACE32 PowerVie	w for ARM 1 [SII	VI @]							
Fi	le Edit View Var	Break Run	CPU Misc T	race Perf Co	v ARM9	Window I	Help			
	N H 🖓 🕇 K	(€)	🕮 ? N ?		- 🐼 🗟	۲	1 🖉			
	B::List.auto					_				- • •
	🕨 Step 🛛 🛤 🕻	Over 🛃 Dive	erge 🖌 🖋 Return	n 🙋 Up	► Go	II Bre	ak 🛛 🎇 Mode	😸 t "	Find:	sieve.c
	addr/lin	e source								
	B∷List.auto ▶ Step ► (addr/lin	void (*ma	nHook)(void)attribu	ıte_ ((s	ection	(".data")))	= 0;		^

The trace recording is automatically started with the program execution. The state in the **Trace** window changes from **OFF** to **Arm [A]**. The **used** field displays the fill state of the trace buffer **[B]**.

METHOD Onchip Analyzer O CAnalyzer O HAnalyzer O Integrator O Pr	
Onchin Analyzer OCAnalyzer OHAnalyzer OIntegrator OPr	
Contemp Contemp Contempter Contempter	obe 🔿 IProbe 🔿 🗛 👘
	IOOPer O FDX O NONE
state used ACCESS	
O DISable auto V	
O OFF 10000000.	
Arm SIZE CLOCK	
O trigger 10000000.	
● Fifo SLAVE	
commands O Stack	
⊗ Init O Leash	
SnapShot O STREAM	
List O PIPE	
AutoArm	
AutoInit	
C	

In order to stop the trace recording, stop the program execution with the **Break** button. The state in the trace window changes to **OFF**.

	🔺 TRACE32 PowerView for ARM 1 [SIM @]	
1	File Edit View Var Break Run CPU Misc Trace Perf Cov ARM9 Window Help	
	Ŋ⋫⊾ ↓✔⊄ ▶ <mark>Ⅲ</mark> ?№ ◎ ≝ ﷺ 🛑 🚳 📾 🔮 💈 🎾	
ſ	📰 B::List.auto	- • •
	🕨 Step 📫 Over 🛃 Diverge 🖋 Return 🙋 Up 🕨 Go 🔢 Break 💹 Mode 🚳 九 👎 Find:	sieve.c
Ш	addr/line source	
	<pre>void (*monHook)(void)attribute ((section (".data"))) = 0;</pre>	^

The trace recording is automatically started and stopped when starting and stopping the program execution because of the **AutoArm [C]** setting in the **Trace** window, which is per default enabled. The trace recording can also be started/stopped manually while the program execution is running using the radio buttons **Arm** and **OFF** of the **Trace** window **[A]**.

TRACE32 offers different view for displaying the trace results. This document shows some examples.

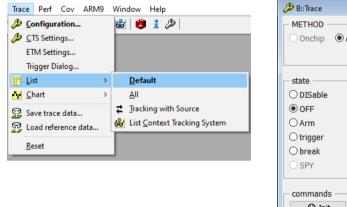
Please note that the trace results can only be displayed if the trace state in the **Trace** window is **OFF**. It is not possible to display the trace results while recording.

The caption of a TRACE32 window includes the TRACE32 command that can be executed in the TRACE32 command line or in a PRACTICE script to open this window, e.g. here **Trace.List**

::Trace.List	t									x
🔑 Setup	Ø	onfig	🔒 Goto	🛐 Find	Chart	📕 Profile	🔼 MIPS	More	Less	
record	run	addre	ss	cycle	data	symbol			ti.back	
-00000040 -00000039		str	r2,[r3] D:00007FCC T:000006E8	wr-long fetch	15643BC7 0423	\\sieve\s \\sieve\s	ieve\vlong ieve\func2	d+0x40	0.100us 0.100us	< II >
223		1s1	r3.r4.#	0x10	: r3.re		-			^
-00000038	<		T:000006ÉA	fetch	OCIB	\\s1eve\s	ieve\func2	d+0x42	0.100us	
3:: Trace. L [ok]		cord>	<recordrange></recordrange>	<tim< td=""><td></td><td></td><td></td><td></td><td></td><td></td></tim<>						

Trace List

A list view of the trace results can be opened from the menu Trace > List > Default. The same window can be opened from the Trace configuration window by pressing the List button.



🔑 B::Trace		
Onchip Onchip	Analyzer O CAnalyzer	HAnalyzer Integrator Probe IProb OART OLOGGER OSNOOPer OFDX
state O DISable O OFF O Arm O trigger	used	ACCESS
O break O SPY	Mode	Mode
SnapShot SnapShot ∐List AutoArm	O Leash O STREAM O PIPE O RTS	
AutoInit		J •
<		

The **Trace.List** window displays the recorded trace packets together with the corresponding assembler and source code.

B::Trace.Lis	t							×
🔑 Setup	🔑 Config 🔒 Goto	👘 Find	Chart	📕 Profile	K MIPS	More	Less	
record	run address	cycle o	data	symbol			ti.back	
-02791017	T:0000137A	fetch F	F91E	\\sieve\s	ieve\main+	0xD6	0.100us	~
713	Г. –	func2b();						=
	bl 0x5B8		; func2b					_
-02791016	Т:000005В8	fetch	B590	<pre>\\sieve\s</pre>	ieve\func2	b	0.100us	~
192	void func2b(voi { push {r4,r7,							^
-02791015	D:00007FAC		40000000	\\sieve\s	ieve\stra1	+0xA8	0.100us	
-02791014	D:00007FB0	wr-long (00007FC0	\\sieve\s	ieve\stra1	+0xAC	0.100us	
-02791013	D:00007FB4						0.100us	
-02791012	Т:000005ВА		B083	<pre>\\sieve\s</pre>	ieve\func2	b+0x2	0.100us	
	sub sp,#0x0							
-02791011	T:000005BC		AF00	//sieve/s	ieve\func2	b+0x4	0.100us	
-02791010	add r7,sp,# T:000005BE		1000	\\ciewe\c	ieve\func2	h. 0v6	0.100us	
-02/ 91010				tack varia	able */	0+0x0	0.10005	- 1
	registe	r long reg	gvar; /	* long re	gister var	iable */		
196	autovar 1dr r3,0x5F	= regvar 0	= mstati	c1;				
-02791009	D:00005F0	rd-long (000067C0	\\sieve\s	ieve\func2	b+0x38	0.100us	
-02791008	T:000005C0	fetch	681C	<pre>\\sieve\s</pre>	ieve\func2	b+0x8	0.100us	×
	<						>	

In our case, trace packets are program fetches (cycle fetch) or data accesses (e.g. wr-long and rd-long for 32bit write and read accesses). Each trace packet has a record number displayed in the **record** column. The record number is a negative index for **Fifo** mode.

As we are using a Simulator, each assembly instruction has an own trace packet. This is not the case with a real hardware trace.

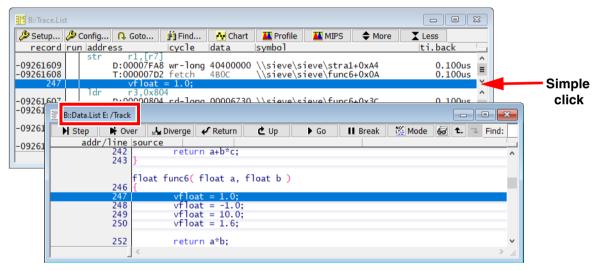
The displayed information can be reduced using the **Less** button. By pressing **Less** three times, only the high-level source code is displayed. This can be reverted using the **More** button.

🎾 Setup	🔑 Config	🔒 Goto	Find	Chart	📕 Profile	📕 MIPS	More	Less	
record								ti.back	
187		for (regvar = 0			5 ; regva	°++)	1.700us	
188	Γ		vchar +=	= regvar*a	autovar;			0.600us	s
187		for (regvar = 0	: regvar	< (char)	5 : regva	·++)	1.700us	s
188	l i			regvar*a				0.600us	s
187		for (regvar = 0	: regvar	< (char)	5 : regvar	·++)	1.700us	s
188		101 (regvar*a		s , regra		0.600us	
1.07		<i>c i</i>			(1) >	-		1 700	
187 188		TOP (regvar = 0	; regvar : regvar*a		5 ; regva	°++)	1.700us 0.600us	
100	Ι Γ		venar +-	- regvar "a	aucovar,			0.0000	2
187		for (regvar = 0); regvar	< (char)	5 ; regva	·++)	1.700us	
189	}		-					0.600us	
713	Г		func2b()	;				0.800us	s
	[void	func2b(vo	(bi						
192		14110210(10	(a)					0.200us	s
		long a regist	utovar; er long re	/* long s gvar; /	tack vari /* long re	able */ gister van	iable */		
196		autova	r = regvar	= mstati	c1:			0, 600us	s
197		autova			,			0.600us	s

A double click on a line with an assembly instruction or high-level source code opens a List window showing the corresponding line in the code.

🔑 Setup	🎾 Config 📭 Goto 👘 Find 🚺 Chart 🛛 🗮 Profile 🛛 🗮 MIPS 🔶 More 🛛 🗶 Less	
record 187	for (regvar = 0; regvar < (char) 5 ; regvar++) 1.700us	
188	vchar += regvar *autovar; 0.600us =	
		Daub
187 189	for (regvar = 0; regvar < (char) 5 ; regvar++) 1.700us	Doub
		clic
B::List	T:0x59A	23
► Ste	p 뵭 Over 🛃 Diverge 🖌 Return 🙋 Up 🕨 Go 🔢 Break 💥 Mode 🐻 📬 ৃ Find:	_
	ine source	
	1.87 for (regvar = 0; regvar < (char) 5 ; regvar++)	~
	188 vchar += regvar*autovar;	
	22 }	
	void func2b(void)	
	192 {	_
	long autovar; /* long stack variable */ register long regvar; /* long register variable */	
	register long register variable /	
	L96 autovar = regvar = mstatic1;	
	97 autovar++;	

Using the TRACE32 menu Trace > List > Tracing with Source, you get a Trace.List and a List /Track window. When doing a simple click on a line in the Trace.List window, the List window will automatically display the corresponding code line.



The timing information (see **ti.back** column) is generated in this case by the TRACE32 Instruction Set Simulator. With a real core trace, timestamps are either generated by the TRACE32 trace hardware or by the onchip trace module.

TRACE32 supports nested and flat function run-time analysis based on the trace results. Please refer to the video "Flat vs. Nesting Function Runtime Analysis" for an introduction to function run-time analysis in TRACE32:

support.lauterbach.com/kb/articles/trace-based-profiling

Graphical Charts

By selecting the menu **Trace > Chart > Symbols**, you can get a graphical chart that shows the distribution of program execution time at different symbols. The displayed results are based on a flat analysis:

🔑 Setup	iii Groups	E Config	🔒 Goto	🔒 Goto	👘 Find	∙⊡• In	•⊡• Out	🖸 Full			
			-510.860ms	-51	10.840ms	-51	0.820	ms	-510.800ms	-510.7	80ms
		ress 🛛					_				
		her) 🗈					<u> </u>				
	lobal\ac									- · · ·	
	ee754_rem_										
	2iz_from_t										
	∖aeabi_										
	i2d_from_t										
	al\aeabi										
	nul_from_t		<u>. </u>							·	
	∖aeabi_									. 💻	
	sub_from_t										
	l∖aeabi_										
	eve\Global										
	\kerne]										
	nul_from_t										
	add_from_t										
	sub_from_t										
	sieve\ fun										
	add_from_t										
	liv_from_t					.					
	∖aeabi_					.					
_aeab1_dr	mul_from_t	numb 💽					. II		I . I I		

The corresponding nesting analysis can be displayed using the menu **Perf > Function Runtime > Show as Timing**.

🖢 Setup 🛛 🚺	Groups	E Config		🗘 Goto.	(🕽 Got	o	👘 Fi	nd	• D•]	n	▶ □• O(ut 🗉	E Full							
	r	ange 🖬	.500)ms –	615.0	000ms	5 -	614.5	00ms	-61	.4.	000m	s -	613.	500m	s	-613	. 000ms	; -6	612.	5
\\sieve\	sieve\s	ubst 💀		i uiu				i www	1				ļ.								
\\sieve\s									I				i IIIIII				, III III,				
		oot) 🖬 –	╋	╋	╶═╴╢	· .	╋╟┼┼		╞╌┛╫	· .	-04			╼	. -	╢╌┼┼				╢┼┼╴	+
\\sieve\									.												- 11
sieve\ test					1.11				. I []]					1				- I. .			- 11
	sieve\ f													Į.							- 11
	sieve\ f				, II.				, II					ļ	. I.	III.		. 4			
\\sieve\s					. 1				. 1					1							
\\sieve\s					. 1													. L.			
\\sieve\s																					11
\\sieve\s	ieve\ fu	ınc2d 🚯								- I			· · ·								11
ieve\ init_	linked_	list 🖬									-11		Ľ .		·)=					([]	11
\\sieve\	sieve\f	unc4	111				111				- 11		Ľ .			1111				Ш.	11
\\sieve\	sieve\f	unc3	11								1		· · ·			111				111	11
\\sieve\	sieve\f	unc 5 🚯	11								- ii		· · ·			111				ШĽ	11
\\sieve\	sieve\f	unc6 🗰	11				11	1.1			Ĩ		· ·			111				11	11
	sieve\f			1.1				1.1.1.1.1.1			- 1		1 I I I							- 11 H	- 11

The **In** and **Out** buttons can be used to zoom in/out. Alternatively, you can select a position in the window and then use the mouse wheel to zoom in/out.

The menu entry **Perf > Function Runtime >Show Numerical** displays numerical statistics for each function with various information as total run-time, minimum, maximum and average run-times, ratio, and number of function calls.

E::Trace.STATistic.FUNC								- • •
🌽 Setup 🏥 Groups 🔡	Config 🔒 G	oto 🗾 Deta	iled 🗵 Nestin	g 🙀 Chart				
	funcs: 36.	to	tal: 1.00)0s				
range	total	min	max	avr	count	intern%	1% 2%	5%
\\sieve\sieve\ subst	44.506ms	2.600us	3.300us	3.200us	13909.	4.450%		A
\\sieve\sieve\ encode	136.855ms	98.400us	98.400us	98.400us	1391.	9.234%		
(root)	1.000s	-	1.000s	-	-	9.855%		
\\sieve\sieve\ sieve	83.321ms	59.900us	59.900us	59.900us	1391.	8.332%		
sieve\test_cond_instr	11.267ms	3.300us	4.800us	4.050us	2782.	1.126%	-	
\\sieve\sieve\ func2	31.019ms	22.300us	22.300us	22.300us	1391.	2.142%		
\\sieve\sieve\ func1	22.386ms	2.300us	2.300us	2.300us	9733.	2.238%		
\\sieve\sieve\ func2a	20.865ms	15.000us	15.000us	15.000us	1391.	2.086%		
\\sieve\sieve\func2b	15.162ms	10.900us	10.900us	10.900us	1391.	1.516%		
\\sieve\sieve\func2c	195.675ms	138.600us	142.800us	140.672us	1391.	19.567%		
\\sieve\sieve\func2d	25.866ms	18.600us	18.600us	18.600us	1391.	2.586%		~
	<							>:
-								

Further display options can be selected by doing a right mouse click on a specific function.

B::Trace.STATistic.FUNC									×
🔑 Setup 🏥 Groups 🚦	Config 🔒 🤇	Goto \Xi Detailed	d 🗵 Nesting	g 🙀 Chart					
	funcs: 36.	tota	l: 1.00	0s					
range		min ma	IX	avr	count	intern%	1% 2	% 59	%
\\sieve\sieve\ subst	44.506ms		3.300us	3.200us	13909.	4.450%			~
\\sieve\sieve\encode	136.855	Statistic	400us	98.400us	1391.	9.234%			=
(root) \\sieve\sieve\sieve	1.00(83.321		000s 900us	- 59.900us	1391.	9.855% 8.332%			
sieve\test_cond_instr	11.26	List First	800us	4.050us	2782.	1.126%	_		_
\\sieve\sieve\func2	31.01	List Last	300us	22.300us	1391.	2.142%			
\\sieve\sieve\ func1	22.38	List Extreme	300us	2.300us	9733.	2.238%		-	
\\sieve\sieve\func2a	20.86		000us	15.000us	1391.	2.086%			
\\sieve\sieve\func2b	15.16	Goto Extreme	900us 800us	10.900us	1391.	1.516%	—		
<pre>\\sieve\sieve\func2c \\sieve\sieve\func2d</pre>	195.67	Bookmark Extreme	600us	140.672us 18.600us	1391. 1391.	19.567%		_	- ,
	<	Linkage					1		> .:
		Parents							
	F	Children	В						
	F	Duration Analysis							
	é	Findall Duration							
	F	Distance Analysis							
	é	Findall Distance							
		here	>						

Parents [A] displays for example a caller tree for the selected function. By doing a right mouse click on func1 and selecting **Parents**, we see the run-times of the functions func2 and func9, which have called func1 in the trace recording.

🔑 Setup	iii Groups	Config	Goto	E Detailed	Nesting	Chart					
		funcs: 5.		total	: 22.386ms						
	range	e tree	to	otal	min	max	avr	total%	1%	2%	5%
\\sieve\s	ieve\ func	L ∣⊟ func1		22.386ms		2.300us 2.300us	2.300us 2.300us	100.000% 42.874%			
\\sieve\s	(root	2 ⊢⊟ func2) └── (ro		9.598ms 9.598ms			2.300us	42.874%			_
\\sieve\s		9 └── func9		12.788ms			2.300us	57.125%			—
	(root)) ' (ro	ot)	12.788ms	2.300us	2.300us	2.300us	57.125%			
		<									>

Children [B] displays the run-times of the functions called by the selected function, for example here the function subst called by the function encode.

B::Trace.STAT.ChildTREE T:0x1194	
🔑 Setup 👖 Groups 🔡 Config 🔃 Goto 🛒 Detailed 🖉 Nesting	😾 Chart
funcs: 2. total: 136.	855ms
range tree	total min max avr inter
\\sieve\sieve\encode \\sieve\sieve\subst └── subst	136.855ms 98.400us 98.400us 98.400us 67.4 44.506ms 2.600us 3.300us 3.200us 32.5
<	

A function call tree view of all function recorded in the trace can be displayed using the menu entries **Perf** > **Function Runtime** > **Show as Tree** or **Perf** > **Function Runtime** > **Show Detailed Tree**.

Setup	iii Groups	E Config	Goto	🗾 Detailed	Nesting	Chart			
		fun	cs: 40.	to	tal: 1.	000s			
		range tre	e			total	min	max	avr
		root) 🗉 (1.000s	-	1.000s	-
\\siev	/e\sieve\e		encode			136.855ms	98.400us	98.400us	98.400us
\\sie	eve\sieve\	subst	— subst			44.506ms	2.600us	3.300us	3.200us
\\sie	eve\sieve\	sieve	sieve			83.321ms	59.900us	59.900us	59.900us
	est_cond_		test_con	d_instr		11.267ms	3.300us	4.800us	4.050us
	eve\sieve\		func2			31.019ms	22.300us	22.300us	22.300us
	eve\sieve\		— func1			9.598ms	2.300us	2.300us	2.300us
	/e\sieve\f		func2a			20.865ms	15.000us	15.000us	15.000us
	/e\sieve\f		func2b			15.162ms	10.900us	10.900us	10.900us
	/e\sieve\f		func2c			195.675ms	138.600us	142.800us	140.672us
	/e\sieve\f		func2d			25.866ms	18.600us	18.600us	18.600us
	nit_linked		_init_lin	ked_list		95.215ms	68.500us	68.500us	68.500us
	eve\sieve\		func4			11.815ms	8.500us	8.500us	8.500us
	eve\sieve\		func3			1.668ms	1.200us	1.200us	1.200us
	eve\sieve\		func5			4.448ms	3.200us	3.200us	3.200us
\\sie	eve\sieve\	tunc6	func6			9.730ms	7.000us	7.000us	7.000us

Duration Analysis

By doing a right mouse click on a function in the numerical statistics window (**Trace.STATistic.Func**) then selecting **Duration Analysis**, you get an analysis of the function run-times between function entry and exit including the time spent in called subroutines, e.g. here for the function subst (P:0x114C corresponds to the start address of the subst function):

	E B::Trace.STAT.Fu	uncDURation P:0>	(114C								×
	🔑 Setup 📗	Chart 🛛 🏮 Z	oom 🗗	Zoom	Full						
	•		3909.(1) 1.000s	avr: in:	3.200us 44.506ms	min: out:	2.600us 955.494ms	max: ratio:	3.300us 4.450%		
		count	ratio	1%	2%	5%	10%	20%	50%	100	
ľ	< 2.600us 2.650us	0.	0.000%								^
	2.700us	0.	0.000%								
	2.750us 2.800us	0.	0.000%								
	2.850us	0.	0.000%								
	2.900us	0.	0.000%								
	2.950us	0.	0.000%								
	3.000us 3.050us	0.	0.000%								
	3.100us	0. 0.	0.000%								
	3.150us	0.	0.000%								
	3.200us	0.	0.000%								
	3.250us 3.300us	4172.	29.997% 0.000%								
	3.350us	8345.	60.001%								
	3.400us	0.	0.000%								
	•	0.	0.000%								\sim
		<								>	

The time interval can be changed using the **Zoom** buttons.

By doing a right mouse click on a function in the numerical statistics window (**Trace.STATistic.Func**) then selecting **Distance Analysis**, you can get run-times between two consecutive calls of the selected function, e.g. here for the function subst (P:0x114C corresponds to the start address of the subst function):

	B::Trace.STAT.A	duressoistar	ICE PROX I	140									x
ß	🖉 Setup 📗	📊 Chart	🗘 Zoon	n 🛛 🏝 🕻	Zoom	🗘 F	Full						
		samples total:		13907. .000s	avr: in:		883us 672ms		8.700us 328.000us		636.400us : 99.967%		
	up to	count		tio	1%	2%		5%	10%	20%	50%	100	
<	0.000us 50.000us			0.000% 0.005%								_	^
	100.000us		0.	0.000%									
	150.000us 200.000us			0.000% 0.000%									
	250.000us			0.000%									
	300.000us			0.000%									
	350.000us 400.000us			0.000% 0.000%									
	450.000us			0.000%									
	500.000us			0.000%									
	550.000us 600.000us			0.000% 0.000%									
	650.000us			9.994%									
	700.000us			0.000%									
	750.000us 800.000us			0.000% 0.000%									
	000.000us			0.000%									
-		<			1							3	> .

The **Trace.ListVar** command allows to list recorded variables in the trace. If the command is used without parameters all recorded variables are displayed:

Trace.ListVar

6 /	B::Trace.List	tVar						- • ×
ß	Setup	f	Goto	👸 Find	🖕 Draw			
1	1895263	rur	addre	ess	cycle	data	var	ti.back
	99999990			D:00007FA4	1 rd-long	00000001	stral[8].word = 0x1	0.200us 🔨
	9999988						stra1[7].right = 0x6754	0.200us 🛓
-09	9999983			D:00007FA				0.500us 💻
	9999981			D:0000675	5 wr-byte	6F	x1[1] = 111	0.200us 🗡
-09	9999979			D:00007FA4	1 rd-long	00000001	stral[8].word = 0x1	0.200us ^
-09	9999976			D:00007FA4	4 wr-long	0000002	stral[8].word = 0x2	0.300us 🗸
		<						>

You can optionally add one or multiple variables as parameters.

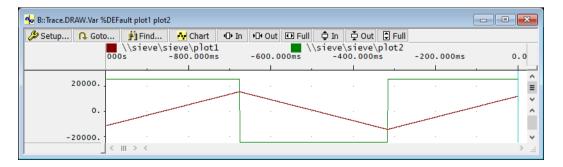
Example: display all accesses to the variables plot1 and plot2

```
Trace.ListVar %DEFault plot1 plot2
```

🛃 B::Trace.Lis	tVar %DEFault	t plot1 plot2						×
🔑 Setup	🔒 Goto	👘 Find	<mark>4</mark> Draw					
4173	run addre	ess	cycle	data	var		ti.back	
-09998147 -09998136 -09998121 -09990973 -09990962 -09990947	<	D:0000672C D:0000672C D:000067CC D:0000672C D:0000672C D:000067CC	wr-word wr-word rd-word wr-word	D120 61A8 D120 D15C	plot1 = plot2 = plot2 = plot1 = plot1 = plot2 =	-12000 25000 -12000 -11940	1.100us 1.500us 714.800us 1.100us 1.500us	а III > < > П >

The **Draw** button can then be used to plot the displayed variables graphically against time. This corresponds to the following TRACE32 command:

```
Trace.DRAW.Var %DEFault plot1 plot2
```



Please refer for more information about the **Trace.DRAW** command to "**Application Note for Trace.DRAW**" (app_trace_draw.pdf). The **/Track** options allows to track windows that display the trace results. You just need to add the **/Track** option after the command that opens a trace window, e.g.

Trace.List /Track

The cursor will then follow the movement in other trace windows, e.g. **Trace.Chart.Func**. Default is time tracking. If no time information is available, tracking to record number is performed.

TRACE32 windows that displays the trace results graphically, e.g. **Trace.Chart.Func**, additionally accept the **/ZoomTrack** option. If the tracking is performed with another graphical window, the same zoom factor is used in this case.

Trace.Chart.Func /ZoomTrack

The Find button allows to search for specific information in the trace results.

🔑 Setup	B	Config	🔒 Goto	iji Find	Chart	📕 Profile	🚺 MIPS	More	Less		
record	rur	addre		cycle	data	symbol			ti.bao	:k	
00000040 00000039		str	r2,[r3 D:00007FC T:000006E] C wr-long 8 fetch	15643BC7 0423	\\sieve\s \\sieve\s	ieve\vlong ieve\func2	d+0x40		L00us L00us	
223 00000038		lsl		#0x10	; r3,re	< 5] ; re gvar,#16 \\sieve\s	-	d+0x42	0.1	L00us	

Example 1: find the first call of function func2

- 1. Enter "func2" under address / expression
- 2. Select Program under cycle
- 3. Press the **Find First** button. The next entries to func2 in the trace can then be found using the **Next** button

🛱 Trace Find -		_
Expert Cycle Group Changes Signal address / expression func2	Direction - O Up	
Cycle Data	Find Up Find Down]
Find First Find Next Find All Find Here Clear	Cancel	

Example 2: Find all write accesses to the variable mstatic1 with the value 0x0

- 1. Enter "mstatic1" under address / expression
- 2. Select Write under cycle
- 3. Enter **0x0** under **Data**
- 4. Press the Find All button

🛱 Trace Find —	
Expert Cycle Group Changes Signal address / expression mstatic1	Direction O Up
Cycle Data Ox0 L V	Find Up Find Down
Find First Find Next Find All Find Here Clear	Cancel

Please refer to "Application Note for Trace.Find" (app_trace_find.pdf) for more information about Trace.Find.

The recorded trace can be stored in a file using the command **Trace.SAVE**, e.g.

Trace.SAVE file.ad

The saved file can then be loaded in TRACE32 PowerView using the command Trace.LOAD

```
Trace.LOAD file.ad
```

The TRACE32 trace display windows will show in this case a LOAD message in the low left corner

🔑 Setup	B	Config< Q Goto		👸 Find	Chart	🔼 Profile	📕 MIPS	More	Less	
record	rur	n addre	ss	cycle	data	symbol			ti.back	
-00000040 -00000039		str	r2,[r3] D:00007FCC T:000006E8	wr-long fetch	15643BC7 0423	\\sieve\si \\sieve\si	ieve\vlong ieve\func2] 2d+0x40	0.100us 0.100us	
223		lsl		0x10	: r3.re	< 5 <mark>] ; re</mark> g gvar,#16 \\sieve\si		0d±0x42	0.100us	-

Please note that TRACE32 additionally allows to export/import the trace results in different formats. Refer to the documentation of the command groups **Trace.EXPORT** and **Trace.IMPORT** for more information.