

Application Note for the SNOOPer Trace

Release 09.2023

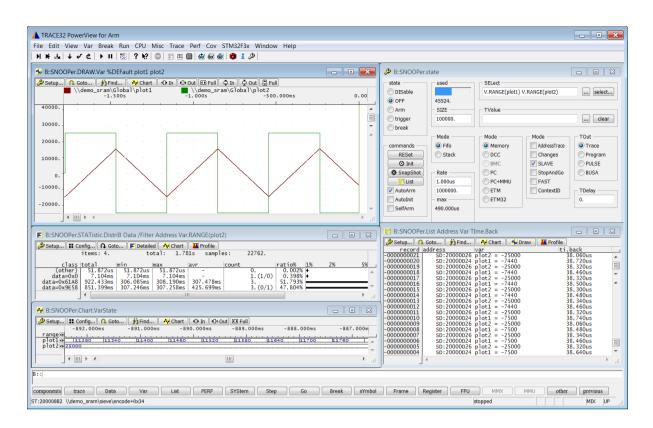


TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Documents	Þ
Trace Application Notes	
Software Traces	
Application Note for the SNOOPer Trace	1
History	3
Introduction	4
SNOOPer Trace Configuration	6
Sampling the Memory Logging a Single Variable Logging only Data Changes Logging Multiple Variables Display the SNOOPer Trace Results List of Recorded Samples Graphical Display of SNOOPer Trace Results Statistical Distributions SNOOPer Trace Trigger Sampling the Program Counter Setup Display Options	 8 10 12 13 14 14 16 18 19 20 21 21
Sampling the Program Counter and the Current Task	21 24
Data Sampling via Debug Communication Channel	27
Sampling Benchmark Counters	29
Sampling ETM Counters	33
Save and Load	37

Version 09-Oct-2023

History

14-Feb-2018 Mayor rework on application note.



The SNOOPer trace is part of TRACE32 trace framework and is designed to collect samples periodically while the program execution is running.

The SNOOPer can be used for:

 Sampling the Memory: the SNOOPer trace allows to sample the content of up to 16 data items while the program execution is running. This feature is especially useful for variable monitoring if the on-chip trace logic can not generate data trace information or if the TRACE32 tool in use is just a debugger with no trace capabilities.

The sampling works non-intrusively, thus without stopping the program execution, if the on-chip debugging interface provides run-time memory access. Otherwise, the debugger will periodically stop the program execution to read the selected memory (StopAndGo mode). Please refer to "Run-time Memory Access" (glossary.pdf) and "StopAndGo Mode" (glossary.pdf).

Memory sampling is only recommended for variables whose sizes are smaller or equal to the core data bus width and which change with a lower frequency than the achievable SNOOPer trace frequency. Because of the low achievable sampling rates, the intrusive StopAndGo mode is not recommended.

- Sampling the Program Counter: the SNOOPer trace allows to periodically sample the actual program counter. This works non-intrusively if the on-chip debugging interface supports one of the following characteristics:
 - The program counter is memory-mapped and the on-chip debugging interface provides runtime memory access (e.g. TriCore).

- The on-chip debugging interface provides the possibility to sample the program counter (e.g. EDPCSR for Cortex-A/R (Armv8), Quick Access for RH850).

Otherwise, the SNOOPer trace will shortly stop the program execution to read the current program counter and resume again.

- **Data Sampling via Debug Communication Channel (DCC)** if the on-chip debugging interface includes DCC capability.
- Sampling Benchmark Counters if the target processor provides benchmark counters.
- Sampling ETM Counters for Arm processors with an Embedded Trace Macrocell.
- SFT Software Trace via LPD4 Debug Port for RH850 processors. Please refer to "RH850 Debugger and Trace" (debugger_rh850.pdf) for more information.

The sampling rate depends heavily on the sampling object (memory, PC...) and the target processor. If the SNOOPer trace works non-intrusively then the rate is generally in the range of microseconds. The intrusive StopAndGo mode is however much more slower with a sampling rate in the range of milliseconds. The sampling rate might be increased by a higher JTAG clock (SYStem.JtagClock <frequency>). Please refer to your processor/chip manual to find out what the maximum JTAG clock can be.

The collected data is stored with timestamp information into a buffer allocated by the TRACE32 PowerView software on the host. The size of the buffer can be set up by the user and is only limited by the resources of the host. To achieve high SNOOPer trace frequencies, the sampling is performed by the software running on the TRACE32 Debug Module where the collected sampled are stored on a temporary buffer. The results are streamed to the host during recording or read by TRACE32 PowerView after the recording is stopped.

The SNOOPer trace is part of the TRACE32 trace framework. To configure the SNOOPer trace:

1. On the TRACE32 main menu bar, choose **Trace** menu > **Configuration**:



2. Under **METHOD**, click the radio option **SNOOPer**.

Þ	B::Tr	ace.state		- • ×		
METHOD Analyzer Onchip OART OLOGGE SNOOPer DEDX OLA Integrator Probe DProbe						
state DISable OFF	used	SELect		select		
○ Arm ○ trigger ○ break	- SIZE	TValue		clear		
	Mode	Mode	Mode	- TOut		
commands RESet S Init	 Fifo Stack 	Memory DCC BMC	AddressTrace Changes SLAVE	Trace Program PULSE		
SnapShot	Rate	OPC	StopAndGo	OBUSA		
List	10.000ms	O PC+MMU				
✓ AutoArm	100.			TDelay		
AutoInit	- max			0.		
SelfArm	0.000us					

Or execute the following commands on the TRACE32 command line:

```
Trace.state
Trace.METHOD SNOOPer
```

Alternatively, execute the SNOOPer.state command:

SNOOPer.state

All commands relative to the SNOOPer trace can be executed using the Trace command group (e.g. Trace.List) after selecting the SNOOPer method in the **Trace.state** window or using SNOOPer command group (e.g. SNOOPer.List). The second form is especially useful if the SNOOPer trace should be used together with a different trace method. In this application note, the **SNOOPer** command group will be used.

The following steps are needed to configure the SNOOPer trace:

1. Reset the SNOOPer trace to its default settings using the **RESet** button [**A**] from the **SNOOPer.state** window or using the command **SNOOPer.RESet**.

	B::SNOOPer.st	ate		(- • ×
	 state DISable OFF 	— used ———	SELect		select
	Arm	- SIZE	C alue		clear
	_ commands	- Mode	- Mode Memory	AddressTrace	Out Out Orace
A	RESet	Stack	O DCC	Changes	Program PULSE
	SnapShot	Rate	PC PC+MMU	StopAndGo	BUSA
	AutoArm AutoInit	1000000. - max			TDelay0.
	SelfArm	0.000us	D		

- 2. You can increase the SNOOPer trace buffer size in the **SIZE** input box [**B**] or using the command **SNOOPer.SIZE**. The size is specified in number of records (samples).
- 3. Select the Fifo or Stack mode [C]. This can also be set using the commands SNOOPer.Mode Fifo or SNOOPer.Mode Stack. In Fifo mode, if the SNOOPer trace is full, new collected samples will overwrite older records. Therefore the SNOOPer trace memory always contains the last samples before stopping the trace. In Stack mode however, if the SNOOPer trace is full the recording will be stopped so that the trace buffer always contains the first samples after starting the trace.

The SNOOPer trace operation mode is set per default to Fifo.

 Set the SNOOPer trace sampling rate in the Rate input box [D] or using the command SNOOPer.Rate. The rate can be specified as time interval (e.g. 10us) or as number of samples per seconds.

The sampling rate is set per default set to 1.us (1000000 samples/s). The defined rate is however not guaranteed.

5. Select the sampling object [E].

Further configurations may be needed depending on the selected sampling object. This will be explained in details for each sampling object in the following chapters.

The settings done in the **SNOOPer.state** window can be saved in the format of a PRACTICE script to an external file using the **STOre** command or to the clipboard using the **ClipSTOre** command.

STOre <file> SNOOP</file>	Create a batch to restore the SNOOPer trace settings
ClipSTOre SNOOP	Provide the commands to restore the SNOOPer trace settings in the cliptext

The typical use case of the SNOOPer trace is variable monitoring. The SNOOPer trace can be used for this purpose the on-chip trace logic can not generate data trace information or if the TRACE32 tool in use is just a debugger with no trace capabilities. Up to 16 data items (e.g. HLL variables) can be monitored using the SNOOPer trace.

B::SNOOPer.List Var TIme.Back / 🗖 🖲 🔀		B::SNOOPer.DRAW /ZoomTrack
🖉 Setup 🔃 Goto 👘 Find 🙌 Chart 📥 Draw 🛄 Profile]	🔑 Setup 📭 Goto 🟥 Find 🕂 Chart 🕕 In 🔎 🖽 Full 🇢 In 🌩 🕃 Full
record var ti.back	1	0.000ms -70.000ms -60.000ms
-0000000250 mstatic1 = -450410872 38.680us		d.any
-0000000249 mstatic1 = -450410872 38.440us	a	600000000.
-0000000248 mstatic1 = -450410872 38.700us = -0000000247 mstatic1 = -450410872 38.700us = -0000000246 mstatic1 = -450410872 38.460us		500000000.
-0000000245 mstatic1 = -450410872 38.440us - 0000000244 mstatic1 = -450410872 38.680us		4000000000. • · · · · · · · · · · · · · · · · · ·
-000000243 mstatic1 = 12 38.700us -0000000242 mstatic1 = -659552286 38.700us		
-0000000241 mstatic1 = -659552286 38.700us		
-0000000240 mstatic1 = -659552286 38.680us		
-0000000239 mstatic1 = -659552286 38.700us		
-0000000238 mstatic1 = -659552286 38.460us	-	
-0000000237 mstatic1 = -659552286 38.460us		

The memory sampling is non-intrusive if the following conditions are met:

- The processor architecture in use allows the debugger to read memory while the program execution is running.
- Run-time memory access is enabled in TRACE32.

Depending on the above conditions, TRACE32 checks/un-checks the **StopAndGo** check box in the **SNOOPer.state** window automatically as soon as a sampling address is selected.

c1)	select
Mode	
Mode	TOut
	TOUL
AddressTrace Changes SI AVF	Trace Program PULSE BUSA TDelay 0.

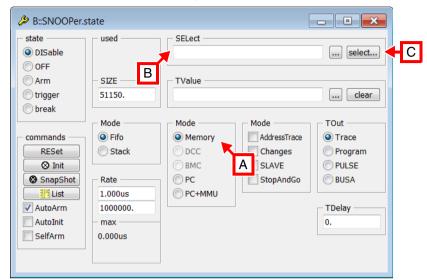
It is not recommended to force the StopAndGo option when memory access on run-time is possible.

If the **StopAndGo** mode is used, a red S will then appear in the state line while recording.

SPE MMU TRANSlation CACHE CORE APU FLASH FLASHFIL	E PFR	other p	previous
running	S	HLL	UP

To set up the SNOOPer trace for memory sampling, the **Memory** radio option [**A**] has to be selected under **Mode**. This option is selected per default after resetting the SNOOPer trace.

Moreover, the variable or memory address of interest needs to be specified under SELect [B]:



- 1. In the SNOOPer.state window, click the select... button [C] to open the SNOOPer.SELect dialog.
- 2. In the SNOOPer.SELect dialog, click the button [E] to get a list of all variables.

B::SNOOP.SEL /DIALOG	
SELect	
Ok Add	E

3. Select the variable you are interested in from the **Browse Symbols** window.

🛓 Browse Symbo	ls	- • •
* * *	1 Type :	Variables 🔻 🗌 Source
symbol	type	address
funcptr	(int (*)())	D:400040D4400040D7
main		
mstatic1	(static int)	D:400040584000405B
sieve str2 str6 stra1 stra2	(struct struct2) (struct struct6) (structarray) (struct struct4 [5][5])	D:400044384000444B D:400044504000445F D:400041C84000428F D:400042904000441F *
<u> </u>		

The above steps can be achieved using the following command sequence:

```
SNOOPer.RESet ; Reset the SNOOPer configuration to
; its default settings
SNOOPer.Mode Memory ; Set the Memory mode
SNOOPer.SELect %Long mstatic1 ; select the 32bit variable mstatic1
```

To inform the debugger about the width of the sampling address, you need to use the %<format> option:

SNOOPer.SELect %Word plot1 ; select the 16bit variable plot1

Alternatively, you can use for variables the Var.RANGE PRACTICE function:

SNOOPer.SELect Var.RANGE(plot1)

If neither the %<format> option nor the Var.RANGE() PRACTICE function is used, the SNOOPer trace will only sample one byte from the given sampling address. For more information, please refer to the documentation of the SNOOPer.SELect command.

After selecting the sampling address, the SNOOPer trace will automatically switch to the OFF state which means that it is ready for sampling.

B::SNOOP.List		
🖉 Setup 📭 Goto 👔 Find 🚺 Chart 骨 Draw	Profile	
record run address cycle data	symbol	ti.back 🔄
-0000000013 NSD:403031A8 snoop	80E08D0A \\sieve_arm\sieve\mstatic1	275.720us 🔺
-0000000012 NSD:403031A8 snoop	80E08D0A \\sieve_arm\sieve\mstatic1	275.780us
-0000000011 NSD:403031A8 snoop	89A60F78 \\sieve_arm\sieve\mstatic1	275.680us 🗐
-0000000010 NSD:403031A8 snoop	EA22AA32 \\sieve_arm\sieve\mstatic1	275.700us 🚽
-0000000009 NSD:403031A8 snoop	EA22AA32 \\sieve_arm\sieve\mstatic1	277.060us
-0000000008 NSD:403031A8 snoop	OF7D5030 \\sieve_arm\sieve\mstatic1	275.700us 🔺
-0000000007 NSD:403031A8 snoop	0F7D5030 \\sieve_arm\sieve\mstatic1	275.700us
-0000000006 NSD:403031A8 snoop	AA62721A \\sieve_arm\sieve\mstatic1	275.720us
-0000000005 NSD:403031A8 snoop	00000000 \\sieve_arm\sieve\mstatic1	277.060us
-0000000004 NSD:403031A8 snoop	523AE728 \\sieve_arm\sieve\mstatic1	275.700us
-0000000003 NSD:403031A8 snoop	8887EEC2 \\sieve_arm\sieve\mstatic1	275.700us 😑
-0000000002 NSD:403031A8 snoop	8887EEC2 \\sieve_arm\sieve\mstatic1	275.720us 📟
-0000000001 NSD:403031A8 snoop	883FDA2A \\sieve_arm\sieve\mstatic1	16.145ms 🔻

The **SNOOPer.List** window displays the time between two consecutive samples which can give an idea about the actual used sampling rate. Moreover, the longest sampling interval for the current trace contents is displayed in the **max** field of the **SNOOPer.state** window.

The **Mode Changes** can be used, if the read variable content should only be stored to the SNOOPer trace when it has changed.

B::SNOOPer.sta	ate			- • •		
state DISable OFF Arm trigger	used 16101. SIZE 500000.	SELect V.RANGE(mstati	ic1)	select		
© break	Mode	- Mode	Mode	Clear		
commands RESet	 Fifo Stack 	Memory DCC	AddressTrace	 Trace Program 		
SnapShot	Rate	BMC PC PC+MMU	SLAVE	© PULSE ◎ BUSA		
AutoArm	B::SNOOPer.Lis	t	🕂 Chart 🛛 😽 Drav	v Profile		
SelfArm	record run -000000021 -000000020 -000000019 -000000018 -0000000016 -0000000014 -0000000014 -0000000014 -0000000012 -0000000011 -0000000012 -0000000012 -0000000001 -0000000009 -000000009 -000000009 -000000009		cycle data is snoop is snoop i4 snoop snoop	Symt 0000000C \dd A87DB722 \dd 0000000C \dd 3D6DB80 \dd 0000000C \dd A83B4FB0 \dd 0000000C \dd 3A866C9A \dd	<pre>bol mmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1 emmo_sram/sieve/mstatic1</pre>	ti.back 577.310us 38.870us 576.860us 38.360us 577.080us 38.620us 577.880us 38.600us 576.980us 38.600us 576.980us 38.600us 576.760us 38.840us 577.180us 38.840us

SNOOPer.Mode Changes ON

If you use the Add button in the SNOOPer.SELect dialog, additional variables can be selected.

🔻 🗾 🗹 HLL
Cancel

This can be achieved by specifying multiple variables in series using the **SNOOPer.SELect** command:

```
; select the 16bit variables plot1 and plot2 SNOOPer.SELect %Word plot2 %Word plot2
```

B::SNOOPer.List		- • •
🖉 Setup 🔃 Goto 👘 Find 🙌 Chart	🖶 Draw 🔛 Profile	
record run address cycle	data symbol	ti.back
-0000000015 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.752us 🔒
-0000000014 0 NSD:40302E5C snoop	2940 \\sieve_ram_arm_v7\Global\plot1	275.600us 🔤
-0000000013 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.280us 📃
-0000000012 0 NSD:40302E5C snoop		275.460us 🚽
-0000000011 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.760us
-0000000010 0 NSD:40302E5C snoop	2850 \\sieve_ram_arm_v7\Global\plot1	275.700us 🔺
-0000000009 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.260us
-000000008 0 NSD:40302E5C snoop	27D8 \\sieve_ram_arm_v7\Global\plot1	275.648us
-0000000007 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.452us
-000000006 0 NSD:40302E5C snoop	2760 \\sieve_ram_arm_v7\Global\plot1	275.720us 😑
-0000000005 0 NSD:40302EBC snoop	9E58 \\sieve_ram_arm_v7\Global\plot2	275.000us 📟
-0000000004 0 NSD:40302E5C snoop	26E8 \\sieve_ram_arm_v7\Global\plot1	275.640us 🝸
•		. In the second s
-		

Please be aware that the debugger reads one variable after the other. As a result the maximum sampling rate is always a multiple of the variables logged e.g. 3 variable, 3 times of max. sampling rate. Moreover, losses are inevitable if the monitored data items are changed at a higher rate by the application program.

List of Recorded Samples

Open the **SNOOPer.List** window to display a list of the recorded samples. The **SNOOPer.List** window can be opened using the **List** button from the **SNOOPer.state** window or using the command.

SNOOPer.List

B::SNOOPer.List				
Image: Constraint of the second in			m_v7\sieve\mstatic1 275.680us m_v7\sieve\mstatic2 276.760us m_v7\sieve\mstatic1 275.680us m_v7\sieve\mstatic2 275.320us m_v7\sieve\mstatic1 275.680us m_v7\sieve\mstatic2 275.880us m_v7\sieve\mstatic2 275.680us	
-000000004 0 NSD:403 -000000003 0 NSD:403	302EB4 snoop 302EB0 snoop 302EB4 snoop 302EB0 snoop	00000022 \\sieve_ram_ar B3338BFA \\sieve_ram_ar 00000022 \\sieve_ram_ar C708AEOA \\sieve_ram_ar	m_v7\sieve\mstatic1 277.380us m_v7\sieve\mstatic2 275.240us m_v7\sieve\mstatic1 21.359ms	-

The **SNOOPer.List** window displays per default for each recorded sample the following information:

- **run**: displays the core number for SMP systems. This column is empty for single core processors.
- address: this column displays the sampling address.
- **cycle**: the cycle type is always snoop.
- **data**: the sampled data value in hexadecimal.
- **symbol**: symbolic information with path and offset of the sampled address.
- **ti.back**: time relative to the previous record.

The **ti.back** values can give an idea about the actual used sampling rate. Moreover, the longest sampling interval for the current trace contents is displayed in the **max** field of the **SNOOPer.state** window. Please note that in case the sampling has been started just after resuming the execution, the first ti.back values can be especially large. The same thing applies for the last ti.back value if the sampling has been stopped when halting the CPU. These values are thus not used when computing the longest sampling rate.

The different columns in the window can be rearranged by changing the order of the **SNOOPer.List** parameters. Moreover, other columns can be added to the window. You can use for example the keyword **Var** to display the recorded variable in its HLL representation or **Time.Zero** to display the time relative to the start of the sampling. Please refer to the documentation of the **SNOOPer.List** command for a complete list of the different possible parameters.

Using the following command for instance, the recorded variable is listed it its HLL representation together with the time relative to the previous record:

B::SNOOPer.List Var TIm	e.Back	
Setup 📭 Goto	🐴 Find 🔤 👬 Chart	🔂 Draw 📕 Profile
record var	1	ti.back
-0000000005 vdoub1e -0000000004 vdoub1e -0000000003 vdoub1e -0000000002 vdoub1e	<pre>2 = -6.5500286323999 2 = 1.6000000000000 2 = 754.05847060000 2 = 2.26217540859999 2 = 4.52435081560000 2 = 7.54058469160000</pre>	97.980us 97.760us 1.157ms 97.440us 97.720us
		F a

You can rearrange the column layout by changing the order of the parameters:

SNOOPer.List	TIme.Back	Var	Data	
--------------	-----------	-----	------	--

- ; rearrange the column layout to
- ; fit your requirements

B::SNOOPer.List TIme.Back Var Data	
Setup 📭 Goto 🎁 Find 💁 Chart 💁 Draw	Profile
record ti.back var	data
-0000000005 97.760us vdouble = 1.6000000000	000001 3FF99999999999A
-0000000004 1.157ms vdouble = 754.058470600	000002e+6 41C67902734CCCCD
-0000000003 97.440us vdouble = 2.26217540859	777777792+7 41EUUAC100100000
-0000000002 97.720us vdouble = 4.52435081560	000004e+9 41F0DAC1D5F9999A
-0000000001 97.640us vdouble = 7.54058469160	000004e+9 41FC17430F39999A ^
T0000000000 97.180us vdouble = 1.60000000000	000001 3FF99999999999A 🔻
	a I

Or display the default parameters together with the time relative to the start to the sampling:

SNOOPer.List DEFault TIme.Zero

B::SNOOPer.List DEFault TIme.Zero		
🔑 Setup 📭 Goto 👘 Find 🙌 Chart	🖕 Draw 🔛 🏧 Profile	
record run address cycle		ti.back ti.zero
-0000021142 SD:20000D84 snoo		sram\sieve\mstatic1 38.400us 15.164s 🔒
-0000021141 SD:20000D84 snoo		sram\sieve\mstatic1 38.660us 15.164s 🔚
-0000021140 SD:20000D84 snoo		sram\sieve\mstatic1 38.6600s 15.164s
-0000021139 SD:20000D84 snoo		sram\sieve\mstatic1 38.420us 15.164s 📿
-0000021138 SD:20000D84 snoo		sram\sieve\mstatic1 38.660us 15.164s
-0000021137 SD:20000D84 snoo		sram\sieve\mstatic1 38.400us 15.164s 🔺
-0000021136 SD:20000D84 snoo		sram\sieve\mstatic1 38.420us 15.164s 📰
-0000021135 SD:20000D84 snoo	00001452 \\demo_s	sram\sieve\mstatic1 38.420us 15.164s 🚟
-0000021134 SD:20000D84 snoo	00001452 \\demo_	sram\sieve\mstatic1 38.420us 15.164s 🍸

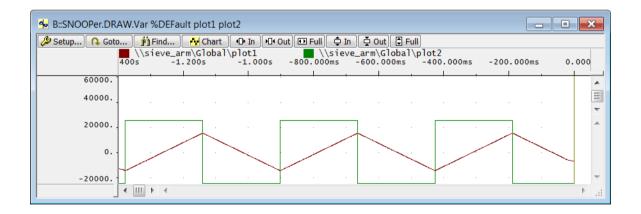
You can use the **Draw** button from the **SNOOPer.List** window to display the sampled data values graphically. Please refer to the documentation of the <trace>.DRAW command group for more information.

B::SNOOP.List Var	B::SNOOPer.DRAW
🖉 Setup 📭 Goto 🎒 Find ᠬ Char 😽 Draw 🗳 Profile	🔑 Setup 🕞 Goto 🎁 Find 🔂 Chart 🛛 🕩 In 🕩 Out ⊡ Full 🇅 In 💆 Out 🗄 Full
record var	0ms -20.000ms -10.000ms
-0000000017 mstatic1 = -1657364150	d.any
-0000000016 mstatic1 = -1657364150 -0000000015 mstatic1 = -1657364150	700000000.
-0000000014 mstatic1 = -1657364150 -0000000013 mstatic1 = -1051136456	600000000.
-0000000012 mstatic1 = -1051136456 -0000000011 mstatic1 = -1051136456	500000000.
-0000000010 mstatic1 = -1051136456 -0000000009 mstatic1 = 1322400882	400000000.
-000000008 mstatic1 = 1322400882 -0000000007 mstatic1 = 1322400882	300000000. Та "{ съ да Г с с б с с Д с с д с), д с
-0000000006 mstatic1 = 1322400882 -0000000005 mstatic1 = 1661507824	
-0000000004 mstatic1 = 1661507824 -0000000003 mstatic1 = 1661507824	
-0000000002 mstatic1 = 1661507824 -0000000001 mstatic1 = 1661507824	0

The **SNOOPer.DRAW.Var** command visualizes e.g. one or more HLL variables in one graphical chart. Using this command, you do not need to specify the display format and the access width of the variables. Moreover, you can superimpose multiple variable in one single graph.

Example: If we display now the results of the plot1 and plot2 variables using the **SNOOPer.DRAW.Var** command, we get the following graph:

SNOOPer.DRAW.Var %DEFault plot1 plot2 ; superimpose variables



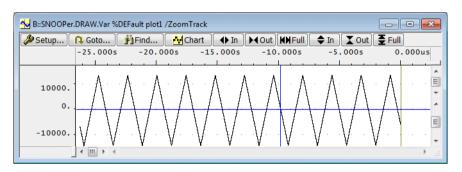
Displaying all variables in one single graph doesn't always make sense, especially if they have different value ranges. In this case, it makes more sense to display each variable in a separate window. By adding the **/ZoomTrack** Option to the **SNOOPer.DRAW.Var** command, a time and zoom synchronisation can be established between the graphical display windows:

SNOOPer.DRAW.Var %DEFault plot1 /ZoomTrack

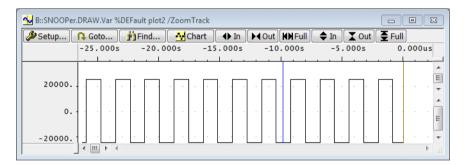
- ; the option ZoomTrack
- ; establishes time- and
- ; zoom-synchronisation
- ; between display windows

SNOOPer.DRAW.Var %DEFault plot2 /ZoomTrack

Active window



Windows with the option **/ZoomTrack** are time- and zoom-synchronized to the cursor in the active window



TRACE32 additionally allows to display the SNOOPer trace results as statistical distributions.

Using the **SNOOPer.STATistic.DistriB** command it is possible to display a distribution statistic of the sampled data values.

Example: We sample the element with index three of the flags array of type char (flags[3]). We can use the following command to display a statistical distribution of the sampled data values:

; display the statistical distribution of a variable value over the time

; Data advise the command to analyze the recorded data information

; Address informs the command for which address the data

; should be analyzed

SNOOPer.STATistic.DistriB Data /Filter Address Var.RANGE(flags[3])

B::SNOOPer.STATistic.DistriB Data /Filter Address Var.RANGE(flags[3]) Setup Config Config <											
class t (other)	0.000us	min 0.000us	-	avr -	count 0.	ratio% 0.000%	1% 2%	5%	10%	20%	
data=0x0 data=0x1	1.416s 64.788ms	359.914us 38.180us	20.284ms 407.696us	871.312us 39.845us		95.624% 4.375%		_			-

These results can also be displayed as time chart using the command SNOOPer.Chart.DistriB e.g.:

; display a time chart of the variable values SNOOPer.Chart.DistriB Data /Filter Address Var.RANGE(flags[3])

➡ B::SNOOPer.Chart.DistriB Data /Filter Ad	dress Var.RANGE(flags[3])	
Setup iii Groups Config	Goto 🛱 Find 🕩 In 🕫 Out 💽 Full	
s -760.000ms class 🛞	-750.000ms -740.000ms -730.000ms	s -720.000ms
(other) data=0x0 data=0x1 () () () () () () () () () () () () () (· · · · · · · · · · · · · · · · · · ·
• III • •		

Using the command **SNOOPer.Chart.VarState**, you can additionally have a graphical representation in time for the taking values of the sampled addresses.

M	B::SN	IOOPer.Chart.VarState			
🖉 Setup 🚦 Config 📭 Go	to 🎒 Find 🔣 Chart	◆ In ◆ Out KN Full			
range plot1	128332	-200.000ms -150.000ms	-100.000ms	-50.000ms	0.000
plot2	-25000	[25000]-25000		25000	
] < _ > <				>

The SNOOPer trace can be used to trigger an action on a specific data value. The trigger can be set by specifying a trigger value and a trigger. Please be aware that the time interval between the trigger event (program writes specified data value to variable) and the triggering by TRACE32 is relatively large. At least max. sampling rate plus reactions time by TRACE32. Thus the trigger can only indicate that the trigger event has taken place. Which instruction initiated the trigger event can not usually be determined.

B::SNOOPer				- • ×
State DISable OFF	used	SELect V.RANGE(mstati	c1)	select
C Arm trigger	SIZE 51150. A	TValue		clear
	Mode	_ Mode	Mode	TOut
commands RESet Sinit SnapShot AutoArm AutoInit SelfArm	Fifo Stack Rate 1.000us 1000000. max 0.000us	Memory C C BMC PC PC PC+MMU ETM ETM32	AddressTrate	 ○ Trace ○ Program ○ PULSE ○ BUSA

To stop for instance the program execution when a certain data value is sampled by the SNOOPer Trace:

- 1. Enter the trigger value in the **Tvalue** field **[A]** of the **SNOOPer.state** window.
- 2. Select the trigger action Program under TOut [B].
- 3. Start the program execution.

The program execution will be stopped as soon as the given value is sampled by the SNOOPer trace.

B::S	SNOOP.Li	st						- • ×
🔑 Seti	up 🔒	Goto	Find	Chart	🖕 Draw	📕 Profile		
r	record r			cycle	data		symbol	ti.back
-00000			NSD:40303			4F3AD148	<pre>\\sieve_arm\sieve\mstatic1</pre>	276.420us 🔒
-00000			NSD:40303	1A8 snoop			<pre>\\sieve_arm\sieve\mstatic1</pre>	278.100us 🔤
-00000			NSD:40303	1A8 snoop			<pre>\\sieve_arm\sieve\mstatic1</pre>	276.400us 📃
-00000			NSD:40303				\\sieve_arm\sieve\mstatic1	276.460us 👻
T00000	000000		NSD:40303	1A8 snoop		00000000	<u>\\sieve_arm\sieve\mstatic1</u>	276.160us
		<u>ا</u>						E. 4



The SNOOPer trigger can only indicate that the trigger event has been taken. It is generally not possible to determine the instruction that initiated the trigger event. The reaction time needed by the debugger to execute the trigger action is approximately 2x the sampling rate. The SNOOPer trace allows to monitor the actual program counter. This mode can be used e.g. for

- Sample-based flat run-time analysis. Please also consider using the **PERF** command group for this purpose.
- Post-mortem debugging: if the target system crashes, it is generally not possible to halt the processor in order to find the location of the crash. However, it is often still possible in such situations to sample the program counter. In this case, the SNOOPer trace can give valuable information about the location of the crash.

E B::SNOOPer.ST	E B::SNOOPer.STATistic.sYmbol										
🥬 Setup 🚺 Groups 🚼 Config 🖪 Goto 🛒 Detailed 📑 Tree 🛛 🕂 Chart 🛛 🗮 Profile											
	items: 34.	t	otal: 4.5	16s sampl	es: 1171()1.					
			1				le or	201	50/	1.007	
address		min	max	avr	count		1%	2%	5%	10%	
(other)	0.000us	-	-	-	0.	0.000%					
ram\sieve\ main	354.838ms	-	-	-	9192.	7.857%				•	
m\sieve\func10	1.356s	-	-	-	35412.	30.031%					_
m\sieve\func21	8.779ms	-	-	-	231.	0.194%	+				=
am\sieve\ subst	233.813ms	-	-	-	5979.	5.177%					
am\sieve\ sieve	699.123ms	-	-	-	18319.	15.481%					
am\sieve\func2	167.000ms	-	-	-	4368.	3.698%					
m\sieve\func2d	93.705ms	-	-	-	2455.	2.075%					
initLinkedList	314.223ms	-	-	-	8120.	6.958%					
am\sieve\ func8	274.038ms	-	-	-	6885.	6.068%					
am\sieve\func9	97.995ms	-	-	-	2548.	2.170%					
m\sieve\func13	161.030ms	-	-	-	4183.	3.565%					
m\sieve\ encode	304.768ms	-	-	-	7772.	6.748%					
am\sieve\ func1	50.791ms	-	-	-	1319.	1.124%					-
m\sieve\ func2b	90.012ms	-	-	-	2373.	1.993%			_		
	<			111							▶

Sampling the program counter works non-intrusively if the on-chip debugging interface supports one of the following characteristics:

- The program counter is memory-mapped and the on-chip debugging interface provides real-time memory access (e.g. TriCore)
- The on-chip debugging interface provides the possibility to sample the program counter on runtime (e.g. EDPCSR for Cortex-A/R (Armv8), Quick Access for RH850).

Otherwise, the SNOOPer trace will shortly stop the program execution to read the current program counter and resume again. A red S will then appear in the state line while recording.

SPE MMU TRANSlation CACHE CORE APU	FLASH FLASHFILE PFR	other previous
	running	HLL UP

SNOOPer.Mode PC

To record the program counter with the SNOOPer trace, you only need to select the **PC** radio option [**A**] in the **SNOOPer.state** window under **Mode** or execute the following command:

; Set the PC mode

- D X B::SNOOPer.state state used SELect OISable ... select... OFF O Arm SIZE TValue () trigger ... clear 51150. O break Mode Mode Mode TOut commands Fifo Memory AddressTrace Trace RESet Stack ODCC Changes Program SLAVE O PULSE ⊗ Init BMC 🔘 PC 🔶 A SnapShot StopAndGo BUSA Rate O PC+MMU 📃 🔛 List 1.000us ✓ AutoArm 1000000. TDelay В AutoInit max 0. SelfArm 0.000us

If sampling the program counter on run-time is not possible, the **StopAndGo** check box [**B**] will be automatically selected in the **SNOOPer.state** window. Manually setting the **StopAndGo** option is not recommended.

Display Options

The **SNOOPer.List** window displays a list of the recorded program counter values. The **SNOOPer.List** window can be opened using the **List** button from the **SNOOPer.state** window or using the command

SNOOPer.List

B::SNOOPe	er.List								×
🖉 Setup 🔒	Goto	🛉 Find 🔤 🗛	Chart	🗧 Draw	📕 Profile				
record	run add	ress	cycle	data		symbol		ti.back	
-000000014	0	R:40300F20				\\sieve_ram_arm_v7\s	sieve\func13	1.417ms	
-000000013	1	R:40301F10				m_arm_v7\sieve\back	cground+0x40	1.403ms	
-000000012	0	R:40300754				\\sieve_ram_arm_v7\s		1.402ms	=
-0000000011	1	R:40301F10				m_arm_v7\sieve\back			
-0000000010	0	R:40301514	snoop			v7\sieve\init_linke			-
-0000000009	1	R:40301F10				m_arm_v7\sieve\back			-
-0000000008	0	R:4030257C				rm_v7\Global\aea			
-0000000007	1	R:40301F10				m_arm_v7\sieve\back			
-0000000006	0	R:40301EB0				ve_ram_arm_v7\sieve			
-0000000005	1	R:40301F10				m_arm_v7\sieve\back			=
-0000000004	0	R:40301650				\\sieve_ram_arm_v7\s		1.402ms	-
-000000003	1	R:40301F10	snoop			m_arm_v7\sieve\back	cground+0x40	1.404ms	Ŧ
								Þ	

The **SNOOPer.List** window displays per default for each recorded sample the following information:

- **run**: displays the core number for SMP systems. This column is empty for single core processors.
- **address**: the sampled program counter value.
- cycle: snoop.
- **data**: this column is empty.
- **symbol**: the symbolic information with path and offset corresponding to the sampled program counter value.
- **ti.back**: time relative to the previous record.

The **ti.back** values can give an idea about the actual used sampling rate. Moreover, the longest sampling interval for the current trace contents is displayed in the **max** field of the **SNOOPer.state** window. Please note that in case the sampling has been started just after resuming the execution, the first ti.back values can be especially large. The same thing applies for the last ti.back value if the sampling has been stopped when halting the CPU. These values are thus not used when computing the longest sampling rate. Please also note that the used sampling rate in the example of the screen shot above is about 1.4ms although the sampling was non-intrusive. This is due to the fact that on some SMP systems the PC sampling is slower than for single core.

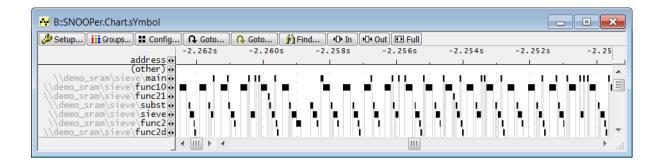
The different columns in the window can be rearranged by changing the order of the **SNOOPer.List** parameters. Moreover, other columns can be added to the window. You can use for example the keyword **TIme.Zero** to display the time relative to the start of the sampling. Please refer to the documentation of the **SNOOPer.List** command for a complete list of the different possible parameters.

Example:

SNOOPer.List TIme.Back Address sYmbol TIme.Zero

🗾 B::SNOOPe	er.List TIme.Ba	ck Address sYmbol	TIme.Zero			×
🔑 Setup 🛛	🖡 Goto 🤇 🏥 F	ind 🚺 🚺 Chart	🖕 Draw 🛛 🌉 Profile]		
record	ti.back	address	symbol		ti.zero	
-000000009	442.940us		<pre>\\sieve\sieve\enc</pre>		8.677s	
-0000000008	442.900us		\\sieve\sieve\mai		8.677s	
-0000000007	443.720us		\\sieve\Global\		8.678s	=
-0000000006	444.200us		\\sieve\sieve\fun		8.678s	-
-0000000005	441.980us		sieve\Global\a		8.679s	
-0000000004	444.060us		\\sieve\sieve\fun		8.679s	-
-000000003	442.060us	T:000011D8	\\sieve\sieve\enc	ode+0x70	8.680s	Ξ
-0000000002	444.300us		\\sieve\sieve\sie		8.680s	
-0000000001	11.286ms	R:00003E68	\\sieve\Global\	.cmpdf2+0x8	8.691s	-
	•					h.

Additionally to the **SNOOPer.List** window, other display options are available. By selecting the **Chart** button from the **SNOOPer.List** window, the SNOOPer trace results can be displayed as a time chart. The corresponding command is **SNOOPer.Chart.sYmbol**.



Please be aware that the displayed charts are based on periodically collected samples and thus not 100% accurate.

If the target processor has a memory management unit (MMU) and a target operating system (e.g. Linux) is used, several processes/tasks can run at the same logical addresses. In this scenario, the logical address sampled by the SNOOPer trace is not sufficient to assign the sampled PC to a program location. For a clear assignment, the information about the current task is also required. The **PC+MMU** mode can be used for this purpose: with every sample, the SNOOPer trace will read the actual program counter and the memory address containing the information about the current task. This mode is however always intrusive since the current task and the program counter have to be read exactly at the same time which cannot be achieved without stopping the program execution.

B::SNOOPer.s	tate			- • ×
state	used	SELect		
DISable				select
OFF	0.775	Theba		
O Arm	- SIZE	TValue		
🔘 trigger	51150.			clear
🔘 break]
	Mode	Mode	Mode	TOut
commands	Fifo	Memory	AddressTrace	Trace
RESet	Stack	O DCC	Changes	Program
S Init		🖉 вмс	SLAVE	O PULSE
SnapShot	Rate	O PC	StopAndGo	O BUSA
List	1.000us	O PC+MMU		
🗸 AutoArm	1000000.			TDelay
AutoInit	- max	-		0.
SelfArm	0.000us			

Example: A Linux OS is running on a target with a Cortex-A9 core. The sampled program counter values are in the user space. Due to the fact that different user tasks can run on the same virtual addresses, these addresses cannot be assigned to distinct program addresses.

B::SNOOP.L	.ist						×
setup ᠺ	Goto 🎁	Find Chart	🖕 Draw	Profile			
record	run addre	ss	cycle	data	symbol	ti.back	
-000000014			snoop			395.140us	
-000000013		R:0000:00009864	snoop			394.300us	
-0000000012		R:0000:00009864				395.680us	Ξ
-0000000011		R:0000:000097F4				393.820us	-
-0000000010		R:0000:0000983C				393.720us	
-0000000009		R:0000:00009864				395.320us	-
-0000000008		R:0000:000097F4				393.880us	
-0000000007		R:0000:00009890	snoop			394.800us	
-0000000006		R:0000:000097F4				394.140us	
-0000000005		R:0000:000097F4	snoop			393.800us	
-0000000004		R:0000:0000989C	snoop			394.920us	
-000000003		R:0000:00009890	snoop			393.820us	=
-0000000002		R:0000:000097F4	snoop			395.680us	
-0000000001		R:0000:0000988C	snoop			15.323ms	Ψ.
	•					•	

The **PC+MMU** mode will be used to additionally read the current task with every sampled program counter. Since an OS Awareness is loaded in TRACE32, the SNOOPer trace automatically knows how to sample the current task.

SNOOPer.Mode PC+MMU

[;] Sample the PC and the current task

The sampled program counter values are now assigned to the process symbols. The OS Awareness gets the space ID (e.g. 0x5F in the screenshot below) and thus the process from the sampled task identifier, the so-called task magic number.

B::SNOOP.List		
🌽 Setup [♀ Goto] 👘 Find] 🗛 Chart]	cycle data symbol snoop \process1\sieve\sieve snoop \process1\sieve\sieve	ti.back +0x94 +0x80 +0x4C +0x4C +0x4C +0x4C +0x42 +0x28 +0x28 +0x28 +0x28 +0x74 +0x67 ms +0x6C +0x6 +0x6 +0x6 +0x6 +0x8 +0x7 +0x8 +0x7 +0x8 +0x7 +0x8 +0x7 +0x8
-0000000006 NUR:005F:00009830 -0000000005 NUR:005F:000098A0 -0000000004 NUR:005F:00009854 -0000000002 NUR:005F:00009838 -0000000002 NUR:005F:00009838 -0000000001 NUR:005F:0000980C	snoop \\process1\sieve\sieve snoop \\process1\sieve\sieve snoop \\process1\sieve\sieve snoop \\process1\sieve\sieve snoop \\process1\sieve\sieve	+0x60 31.432ms +0x00 32.158ms +0x84 31.475ms +0x90 31.828ms +0x68 32.613ms

The intrusive **StopAndGo** mode is used. This can be clearly seen by comparing the **ti.back** values between the first and second screen shot of the **SNOOPer.List** windows.

Sampling the Context ID Register

For Arm processors supporting reading the Context ID register on run-time (e.g. Cortex-A15), by enabling the **SNOOPer.Mode ContextID** mode, the Context ID register can be sampled instead of the memory address containing the current task identifier (task magic number). This way, the sampling can be achieved without disturbing the program run-time.

Example:

SNOOPer.Mode PC+MMU	;	Sample the PC and the current task	
SNOOPer.Mode ContextID ON	;	Sample the Context ID register	

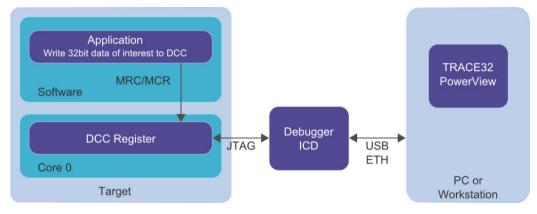
The **SNOOPer.List** window displays in the data column for each record the magic of the task corresponding to the sampled program counter. The magic number is task unique identifier used by the OS Awareness and is generally the address of the task control block. The magic numbers of all running tasks care displayed in the **TASK.List.tasks** window. In case no OS Awareness is loaded, the value of the sampled Context ID register is displayed in the data column.

B::SNOOP	er.List						×
🔑 Setup 🚺	🕽 Goto 📄 Find 🛛 👬 Cha	rt 🛛 📥 Drav	v 🛛 🔼 Profile]			
record	run address	cycle	data	symbol		ti.back	
000000088	R:02D6:00008D3	4 owner		DAOCO\Global		490.880us	
000000087	R:02D6:0000902	8 owner			aeabi_ddiv+0x10C	490.400us	-
000000086	R:02D6:0000900				aeabi_ddiv+0xEC	490.000us	=
000000085	R:02D6:00008FB				aeabi_ddiv+0x9C	490.420us	-
000000084	R:02D6:00008FD				aeabi_ddiv+0xBC	491.280us	
000000083	R:02D6:00008FA				aeabi_ddiv+0x8C	490.740us	-
000000082	R:02D6:00008FE				aeabi_ddiv+0xCC	488.860us	
000000081	R:02D6:000089B				bal\adddf3+0xB4	491.900us	
080000000	R:02D6:00008FE				aeabi_ddiv+0xCC	489.140us	
0000000079	R:02D6:00008FD				aeabi_ddiv+0xBC	490.420us	
000000078	R:02D6:000089F				bal\adddf3+0xEC	486.940us	Ξ
0000000077	R:02D6:00008CB				lobal\aeabi_dmul	492.940us	
000000076	R:02D6:00008FB	8 owner	EDE	12BCO\Global	aeabi_ddiv+0x9C	490.700us	Ψ.
							▶

The **D**ebug **C**ommunication **C**hannel - short **DCC** - is a characteristic of the on-chip debugging support. It allows to pass information between the application program on the target and the debugger. For details refer to your CPU manual.

If the SNOOPer trace uses the DCC, the following basic steps are required:

- The application program on the target writes a 32 bit information to the corresponding registers of the DCC.
- The debugger on the other side checks the DCC registers in a defined sampling rate and enters the received information into the SNOOPer trace buffer.



In order to check whether your CPU provides a DCC, check if the **DCC** radio option is available under **Mode** in the **SNOOPer.state** window and that it can be selected:

B::SNOOPer.s	state			- • ×
state DISable	used	SELect		select
OFF	0.			
C Arm	- SIZE	TValue		
🔘 trigger	51150.			clear
🔘 break				
	Mode	Mode	Mode	TOut
commands	Fifo	Memory	AddressTrace	Trace
RESet	Stack	DCC	Changes	Program
Ø Init		O BMC	SLAVE	O PULSE
SnapShot	Rate	© PC	StopAndGo	O BUSA
List	1.000us	O PC+MMU	FAST	
AutoArm	1000000.	© ETM	ContextID	- TDelay
AutoInit	- max	© ETM32		0.
SelfArm	0.000us			

or enter the following command:

SNOOPer.Mode DCC

; If your debugger accepts this command, DCC ; is provided by your CPU

The application program has to provide the data of interest. This requires that special code is added to the application program. An example for the Arm architecture can be found in the TRACE32 demo folder under ~~/demo/arm/etc/snooper_dcc. You can also get this demo by sending an e-mail to support@lauterbach.com.

The data that should be sampled by the SNOOPer trace is written to the DCC registers using the following function:

```
/* SnoopData may be called by the application */
void SnoopData(unsigned int data) {
    while (T32_TsMon_SendStatus()); //get status of the com-channel
    T32_TsMon_SendWord(data); //if it's free, send data to channel
}
```

If you plan to use the SNOOPer via DCC, you have to be aware of the following:

- 1. New information can only be passed by the application program to the DCC if the debugger has already read the previous written information. The function T32_TsMon_SendStatus() in the above example checks the status of the DCC. This behavior allows the user to select one of the following strategies:
 - If DCC is not ready for the next 32 bit information, the application program can wait until DCC is ready and pass the information then. This way no information is lost, but waiting will consume CPU time.
 - If DCC is not ready for the next 32 bit information, the application program can ignore the current 32 bit information and continue the program execution. This way information might be lost, but the CPU doesn't spend CPU time to wait until DCC is ready.

The fastest possible sampling rate by the debugger is approximately 50 µs.

2. For an SMP system, the demo code that writes to the DCC registers has to run on the first core.

The **SNOOPer.List** window displays for each recorded sample the sampled data value together with the time relative to the last record.

	🗾 B::SNOOPer.	List		
	🔑 Setup 🔒	Goto 🏽 👘 Find 🛛 💀 Chart	🖕 Draw 🛛 🌉 Profile	
	record r	un address cycle	data symbol	ti.back
	-0000000012	C:00000000 snoop	00000F80	749.120us 🔒
	-0000000011	C:00000000 snoop	00000F81	749.346us
ŀ	-0000000010	C:00000000 snoop	00000F82	750.334us 😑
	-000000009	C:00000000 snoop	00000F83	748.100us 👻
	-0000000008	C:00000000 snoop	00000F84	749.340us
	-0000000007	C:00000000 snoop	00000F85	748.226us 🔺
	-000000006	C:00000000 snoop	00000F86	750.094us
ŀ	-0000000005	C:00000000 snoop	00000F87	749.640us
ŀ	-0000000004	C:00000000 snoop	00000F88	750.266us
ŀ	-000000003	C:00000000 snoop	00000F89	748.400us =
ŀ	-0000000002	C:00000000 snoop	00000F8A	750.100us 🧮
ŀ	-0000000001	C:00000000 snoop	00000F8B	750.734us 🝸
		< i		b. ∢

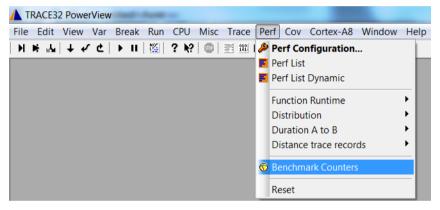
Benchmark counters are on-chip counters that count specific hardware events e.g. the number of executed instructions or number of cache misses. Please refer to your **Processor Architecture Manual** to check if your target processor supports benchmark counters.

The SNOOPer trace can be used to record benchmark counters periodically. This is done non-intrusively if the target system allows to read these counters while the program execution is running. Otherwise, the intrusive **StopAndGo** mode is used. Several benchmark counters can be sampled at the same time. All counters are read simultaneously in one step.

The benchmark counters can be configured in TRACE32 using the **BMC** (**B**ench**M**ark **C**ounter) command group.

To configure the SNOOPer trace for benchmark counter sampling, the following steps need to be done:

1. Open the **BMC.state** window from the TRACE32 menu **Perf > Benchmark Counters**. This menu is only visible if benchmark counters are provided by the selected chip.



2. Select the counters that should be sampled from the **BMC.state** window. You can select one or several counters.

🕲 B::BMC.state				- • ×
Control RESet S Init	AutoInit	SNOOPer Hist	SELect PMN0 TREE art SYmbol SYmbol	
counter name	event	size value	ratio	ratio value o
	ICMISS (Instruction Cache OFF (Disable Benchmarkcou OFF (Disable Benchmarkcou	32BIT nter) 32BIT nter) 32BIT nter) 32BIT nter) 32BIT nter) 16BIT nter) 16BIT nter) 32BIT	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	· ·
•		III		E. ▲

Alternatively, you can assign the event of interest to the benchmark counter using the following PRACTICE commands.

```
BMC.<counter1> <event1> ; assign event of interest to
; the event counter
BMC.<counter2> <event2> ; several assignments possible
...
```

The syntax of the commands is architecture-specific. Please refer to your **Processor Architecture Manual** for more information.

Example (Arm):

BMC.PMN0 ICMISS	; assign instruction cache miss ; counter to first event counter
BMC.PMN0 DCMISS	; assign data cache miss counter to ; second event counter

 Configure the SNOOPer trace for benchmark counter recording by selecting the SnoopSet check box in the BMC.state window or by selecting the BMC mode from the SNOOPer.state window. When the SnoopSet option is selected in BMC.state, the BMC mode is automatically selected in SNOOPer.state and vice-versa.

S::BMC.state		
control profile (Marcological PRO	file SNOOPer SELect SNOOPer IIIList SNOOPSet PROfileChart SYmbol SYmbol	
counter name event ── CLOCKS	ل B::SNOOPer	
BETM1 OFF (Disable Benchman PMN3 OFF (Disable Benchman PMN3 OFF (Disable Benchman PMN3 OFF (Disable Benchman PMN4 OFF (Disable Benchman PMN5 OFF (Disable Benchman BETM1 OFF (Disable Benchman BETM2 OFF (Disable Benchman BETM2 OFF (Disable Benchman BL2CNT0 OFF (Disable Benchman BL2CNT1 OFF (Disable Benchman	state used SELect DISable 0. TValue OFF 0. TValue trigger 51150. TValue break Mode Mode Mode Memory Addresst Stack DCC Changes Stack PC StopAnc Ilit 1.000us PC+MMU AutoInit max ETM32	ace TOut Tout Tout O Trace Program ○ PULSE ○ BUSA
	SelfArm 0.000us	

The respective TRACE32 commands are

BMC.SnoopSet ON

and

```
SNOOPer.Mode BMC
```

Example (TriCore): We can use the following PRACTCE script to sample data cache / data buffer hits and misses on a TriCore processor:

BMC.RESet	; reset BMC configuration
BMC.M1CNT DATA_X_HIT	; count data cache / data buffer ; hits
BMC.M2CNT DATA_X_CLEAN	; count data cache / data buffer ; misses
BMC.SnoopSet ON	; configure the SNOOPer trace for ; event counter recording

The **SNOOPer.List** window displays for each sample the following information:

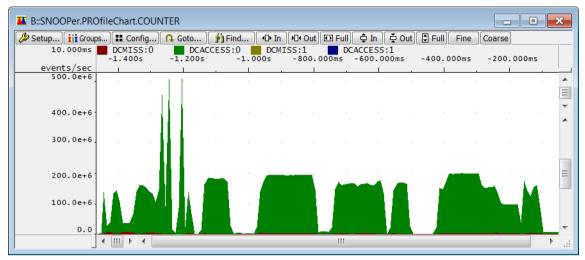
- **core**: core number. This column is only visible for SMP systems.
- <counter>: sampled counter values where <counter> is
 - **bmc**<**x**>: benchmark counter with index <**x**> e.g. bmc0, bmc1...
 - **fbmc**<**x**>: delta bmc<**x**> divided by delta time.
 - architecture specific counter name e.g. m1cnt, m2cnt, m3cnt for TriCore
- **ti.back**: time relative to the previous sample.

B::SNOOPer.List										×
🔑 Setup 📭 Goto	🧌 🎁 Find	Cha	art 🛛 🖕 Dra	aw 🛛 🌉 Pro	ofile					
record core	bmc0	bmc1	bmc2	bmc3	fbmc0	fbmc1	fbmc2	fbmc3	ti.back	
-000000019 1	2FE6E31D	12F1FC45	0000235B	0003A6B8	0D6FA4EC	054ED016	00000000	00000000	460.140us	
-000000018 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.440us	
-000000017 1		12F29BA5						00000000	458.220us	=
-000000016 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.480us	-
-000000015 1	2FEA0C85	12F33BEF	0000235B	0003A6B8	0D702A20	054EBA6E	00000000	00000000	460.820us	
-000000014 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	460.820us	*
-000000013 1	2FEBA048	12F3DB6B	0000235B	0003A6B8	0D6FE66F	054EBF6C	00000000	00000000	458.500us	
-000000012 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	459.000us	
-0000000011 1	2FED35AC	12F47B8D	0000235B	0003A6B8	0D6FAC8D	054EB379	00000000	00000000	460.380us	
-000000010 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	459.260us	
-0000000009 1	2FEEC8ED	12F51AD1	0000235B	0003A6B8	0D701353	054EA97B	00000000	00000000	457.900us	Ξ
-000000008 0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.760us	
-0000000007 1	2FF05E24	12F5BADF	0000235B	0003A6B8	0D702017	054ECE3A	00000000	00000000	460.120us	Ŧ
•									Þ	

If you change the number of the assigned benchmark counters, then you need to refresh the **SNOOPer.List** window so that it gets adjusted to the new configuration.

Please be aware that the debugger reads all counters at once. So the number of read counter has nearly no impact on the maximum sampling rate.

The **SNOOPer.PROfileChart.COUNTER** command can be used to display a graphical profile statistic of the sampled benchmark counter values. The result is a stacked graph i.e. the total number of events/s at a given time represent the sum of the events for all counters at that time.



The TRACE32 SNOOPer trace allows to sample Embedded Trace Macrocell (ETM) counters on Arm processors.

The number of available ETM counters [A] is displayed in the ETM.state window that can be accessed from the TRACE32 PowerView menu Trace > ETM Settings.



The ETM.state window also displays the current values of the ETM counters [B].

You can display the ETM counter registers from the ETM.state window using the Register button [C] or using the command

```
ETM.Register "Counter"
```

Please do not change the register values manually since they are programmed by the debugger.

B::ETM.Register				- • ×
<u>Address Comparators</u>				·
Data Comparators				
				≡
© Counter CENABLE1 00023787 ECOND yes BF A	RB external	RBS 16 RA acomp	RAS 8	
CRELOADI 000037C0 BF A IVALUEI 0000FFFF COUNT 0000FFFF CVALUEI 0000FFFF COUNT 0000FFFF	RB external	RBS 16 RA counter	RAS 1	
CENABLE2 000207C0 ECOND yes BF A	RB acomp		RAS 1	
CRELOAD2 000037C1 BF A IVALUE2 0000FFFF COUNT 0000FFFF CVALUE2 0000FFFF COUNT 0000FFFF	RB external	RBS 16 RA counter	RAS 2	-
	III			<u>ب</u>

The SNOOPer trace can be used to sample periodically the values of the ETM counters. Two special modes are provided for this purpose:

- **ETM**: sample the 16bit value of the first ETM counter.
- ETM32: two ETM counters are used to have a 32bit counter.

B::SNOOPer.s	tate			- • ×
state DISable	used	SELect		select
OFF Arm	0. SIZE	- TValue		
🔘 trigger	51150.			clear
🔘 break				
	Mode	Mode	Mode	TOut
commands	Fifo	Memory	AddressTrace	Trace
RESet	Stack	O DCC	Changes	Program
S Init		ВМС	SLAVE	O PULSE
SnapShot	Rate	○ PC	StopAndGo	O BUSA
List	1.000us	PC+MMU	FAST	
🗸 AutoArm	1000000.	ETM	ContextID	- TDelay
AutoInit	- max	— ЕТМ32		0.
SelfArm	0.000us		_	

You can assign an ETM counter to a breakpoint using the **Break.Set** option **/BusCount**. On a breakpoint hit, instead of stopping the program execution, the ETM counter will then be increased. This can be used for instance to count the number of calls of a certain HLL function or program address:

Break.Set <address> /BusCount

The SNOOPer trace will then give you a statical distribution of the function calls over the time.

Sampling the ETM counters can be set up using the following PRACTICE script:

SNOOPer.RESet ; reset the SNOOPer
SNOOPer.Mode ETM ; select mode ETM or ETM32
Break.Set myFunc /BusCount

The **SNOOPer.List** window displays the following information:

- core: core number. This column is only visible for SMP systems
- etm1: value of the ETM counter
- **fetm1**: delta etm1 divided by delta time
- **ti.back**: time relative to the previous sample

B::SNOOPe	er.List				×
🔑 Setup 🔃 🗛	ioto)	👘 Find	Chart 4	🗸 Draw 🛛 🌉 Pro	file
record	core	etm1	fetm1	ti.back	
000000011	0	00000081	00000000	446.280us	
-000000013	1	0000014D	00000000	446.200us	
000000011	0	00000081	00000000	447.960us	=
-0000000011	1	0000014D	00000000	448.020us	Ŧ
-0000000010	0	00000081		446.920us	
-0000000009	1	0000014D	00000000	447.400us	-
000000000	0	00000081		448.920us	
000000000	1	0000014D	00000000	449.180us	
000000000	0	00000081	00000000	446.600us	Ξ
000000000	1	0000014D	00000000	445.900us	
-0000000004	0	00000081	00000000	447.340us	Ψ.
	•			Þ	.41

The **SNOOPer.PROfileChart.COUNTER** command can be used to display a graphical profile statistic of the sampled counter values over the time. The window displays per default the values of **fetm1**.

Setup iii Group	 		_				Ť	Find	i)		In][•]	Out	t] 💽	l Ful		‡ In	j	ž 0	ut)	Ş F	ull	F	ine	C	oars	e		
	etn	n1:0 -3		00s	1:1		3.0	00s			-	2.5	500s	5		- ;	2.00)0s			-1	. 5	00s			-1	1.0	00s	
events/sec			1		 							1					1					1							
3000.0.			ŀ																										
2500.0			ŀ																										
2000.0			ŀ			ľ																							
1500.0			ŀ																										
1000.0			ŀ																										
500.0			l.				÷		d.	- It																			

You can display the etm1 values instead using the following command

SNOOPer.PROfileChart.COUNTER etm1

etml:0 etml:1 4.000s -3.500s -2.500s -2.000s -1.500s -1.000s events/sec						_ _			(.	🛉 Find		_		up iii Group
1500000.0	-500.	000s	-1.0	.500s	-1.	.000s	- 2	.500s	-	-3.000s				
		<u> </u>									 		I .	
1000000.0											 			1500000.0
											 			1000000.0
500000.0.	4-1								T					500000.0.

NOTE:Advanced setup of the ETM counters is possible using the ETM Programming
dialog accessible from the TRACE32 menu Trace > Trigger Dialog...
Please refer to "Arm ETM Programming Dialog" (trace_arm_etm_dialog.pdf)
for detailed information.

It is possible to save the SNOOPer trace results to a file for postprocessing using the command **SNOOPer.SAVE** e.g.

SNOOPer.SAVE file.ad

The file can then be loaded in TRACE32 PowerView using the command SNOOPer.LOAD:

SNOOPer.LOAD file.ad

The **SNOOPer.List** window will display the loaded SNOOPer trace data. The message "LOAD" is displayed in red at the lower left of the window to indicate that the displayed data is loaded from a file.

B::SNC	OPer.List		- • ×
🔑 Setup	. 🗛 Goto 🎁 Find 🔂 🗛 Chart	🖶 Draw 🔛 Profile	
rec ru	n address cycle data	symbol	ti.back
-0025	R:0000093C snoop	\\sieve\sieve\func8+0xD0	99.908ms 🔒
-0024	R:00001396 snoop	\\sieve\sieve\main+0x12A	110.457ms
-0023	R:00001080 snoop	ve\sieve\init_linked_list+0x44	70.281ms
-0022	R:00001618 snoop	<pre>\\sieve\sieve\sieve+0x38</pre>	1.424s 👻
-0021	R:00001080 snoop	ve\sieve\init_linked_list+0x44	74.476ms
-0020	R:000015F8 snoop	\\sieve\sieve\sieve+0x18	100.984ms 🔺
-0019	R:000010CE snoop	ve\sieve\init_linked_list+0x92	110.239ms
-0018	R:00003644 snoop	\\sieve\G]oba]\adddf3+0xA0	109.915ms
-0017	R:00000D02 snoop	\\sieve\sieve\func13+0x56	109.958ms
-0016	R:000011BE snoop	<pre>\\sieve\sieve\encode+0x56</pre>	109.834ms
-0015	R:00001108 snoop	ve\sieve\init_linked_list+0xCC	110.156ms
-0014	R:00000636 snoop	\\sieve\sieve\func2c+0x36	109.744ms
-0013	R:00003450 snoop	\\sieve\Global\divsi3+0x0C	110.185ms
-0012	R:00000E3E snoop	\\sieve\sieve\func22+0x2	109.963ms
-0011	R:000034F8 snoop	\\sieve\Global\divsi3+0xB4	110.048ms
-0010	R:00000998 snoop	\\sieve\sieve\func8+0x12C	100.015ms
-0009	R:00001290 snoop	\\sieve\sieve\main+0x24	110.061ms Ξ
-0008	R:00001614 snoop	\\sieve\sieve\sieve+0x34	109.987ms
-0007	R:00000BCA snoop	<pre>\\sieve\sieve\func10+0x142 \\sieve\sieve\sieve+0x36</pre>	109.932ms 110.044ms
-0005	R:00001616 snoop		109.975ms
-0003	R:0000106A snoop R:00003618 snoop	ve\sieve\init_linked_list+0x2E \\sieve\Global\adddf3+0x74	109.852ms
-0004	R:000013E8 shoop	\\sieve\sieve\main+0x17C	109.052ms 109.993ms
-0002	R:0000012 snoop	\\sieve\sieve\func14+0x6	110.339ms
_0001	R:00001506 snoop	\\sieve\sieve\main+0x29A	70.325ms
LOAD		((_ · · · · · / / · · · · · · · · · · · ·	t. ∢

The SNOOPer trace results can also be exported to a file as comma-separated values using the following commands:

PRinTer.FILE snoop_plot1.lst	; specify documentation file name
PRinTer.FileType CSV	; specify comma-separated value as ; output format
WinPrint.SNOOPer.List	; save result of the command ; SNOOPer.List to file