

# Application Note for the SNOOPer Trace

MANUAL

# Application Note for the SNOOPer Trace

---

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

[TRACE32 Documents](#) .....



[Trace Application Notes](#) .....



[Software Traces](#) .....



[Application Note for the SNOOPer Trace](#) .....

1

[History](#) .....

3

[Introduction](#) .....

4

[SNOOPer Trace Configuration](#) .....

6

[Sampling the Memory](#) .....

8

    Logging a Single Variable

10

    Logging only Data Changes

12

    Logging Multiple Variables

13

    Display the SNOOPer Trace Results

14

        List of Recorded Samples

14

        Graphical Display of SNOOPer Trace Results

16

        Statistical Distributions

18

        SNOOPer Trace Trigger

19

[Sampling the Program Counter](#) .....

20

    Setup

21

    Display Options

21

    Sampling the Program Counter and the Current Task

24

[Data Sampling via Debug Communication Channel](#) .....

27

[Sampling Benchmark Counters](#) .....

29

[Sampling ETM Counters](#) .....

33

[Save and Load](#) .....

37

# Application Note for the SNOOPer Trace

---

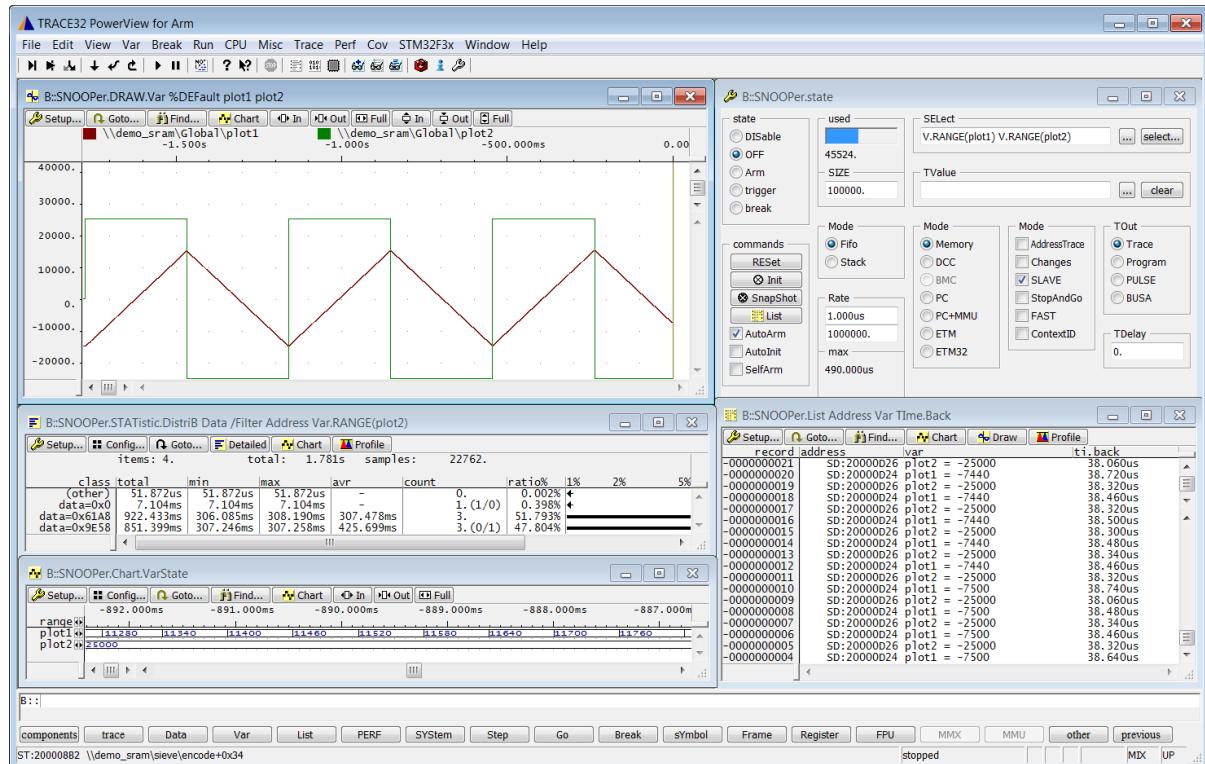
Version 06-Jun-2024

## History

---

14-Feb-2018 Major rework on application note.

# Introduction



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 run-time 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.

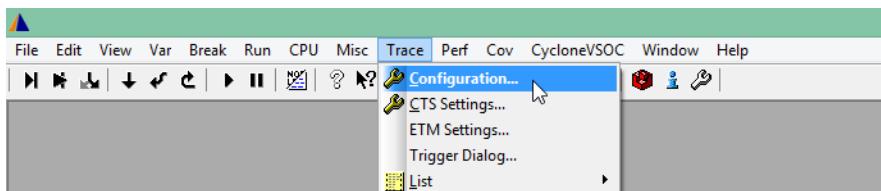
The sampling rate depends heavily on the sampling object (memory, PC...) and the target processor. If the SNOOPer trace works non-intrusively then the rate is generally in the range of microseconds. The intrusive StopAndGo mode is however much more slower with a sampling rate in the range of milliseconds. 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 samples 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.

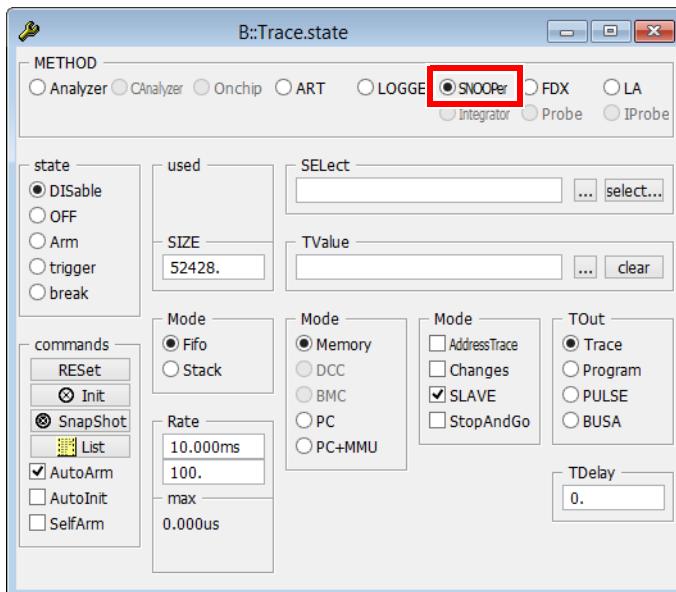
# SNOOPer Trace Configuration

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**.



Or execute the following commands on the TRACE32 command line:

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

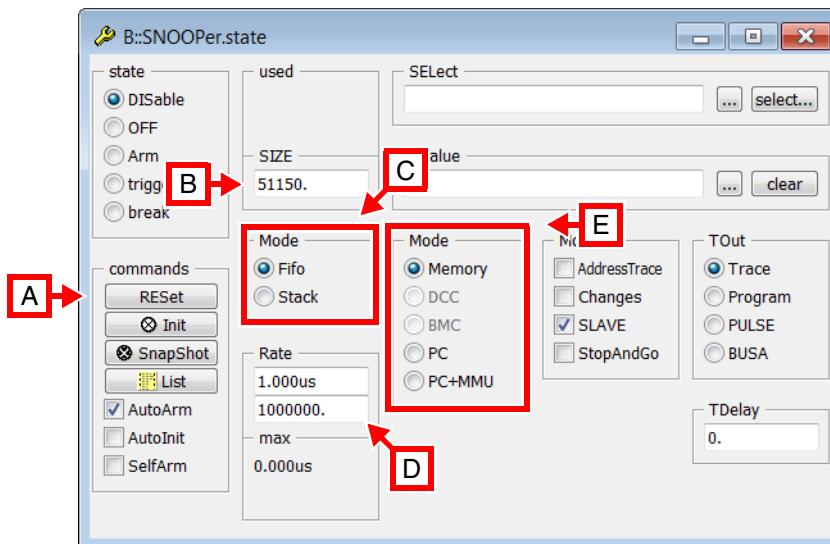
Alternatively, execute the **SNOOPer.state** command:

```
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**.



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.

4. 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**

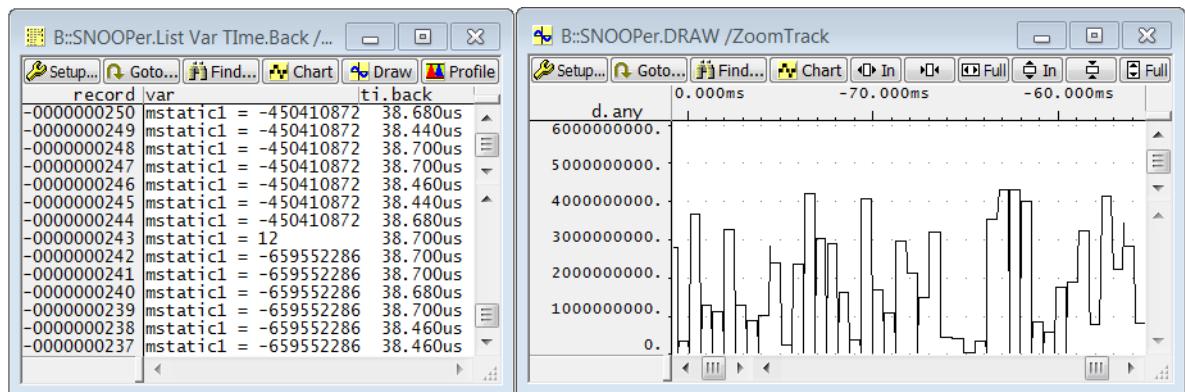
Create a batch to restore the SNOOPer trace settings

**ClipSTOre SNOOP**

Provide the commands to restore the SNOOPer trace settings in the cliptext

# Sampling the Memory

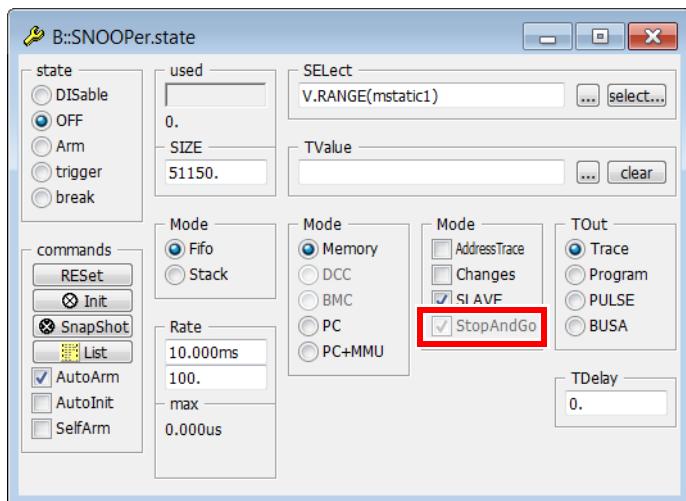
The typical use case of the SNOOPer trace is variable monitoring. The SNOOPer trace can be used for this purpose the on-chip trace logic can not generate data trace information or if the TRACE32 tool in use is just a debugger with no trace capabilities. Up to 16 data items (e.g. HLL variables) can be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



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

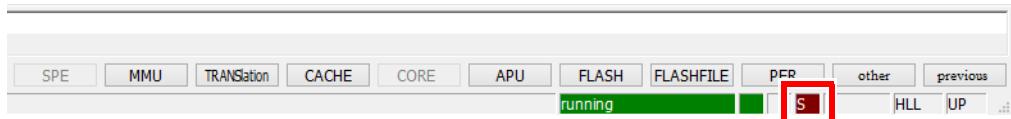
- The processor architecture in use allows the debugger to read memory while the program execution is running.
- Run-time memory access is enabled in TRACE32.

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.



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

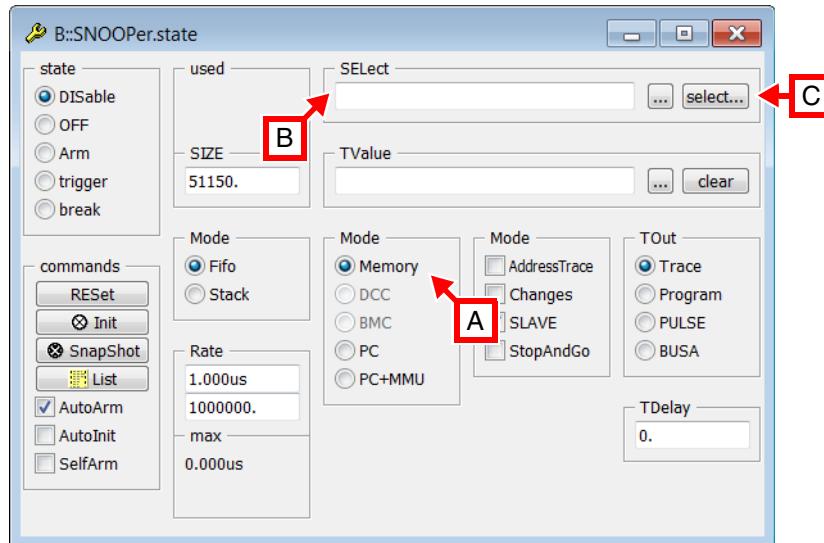
If the **StopAndGo** mode is used, a red S will then appear in the **state line** while recording.



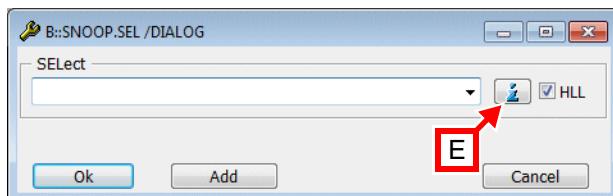
# Logging a Single Variable

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.

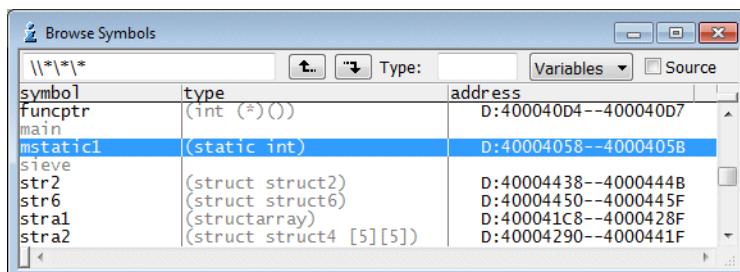
Moreover, the variable or memory address of interest needs to be specified under **SElect** **[B]**:



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



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



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.SESelect %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:

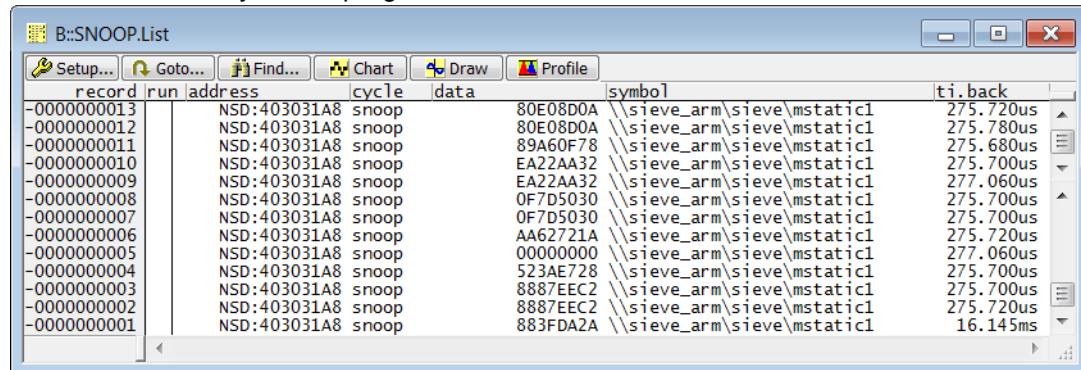
```
SNOOPer.SESelect %Word plot1 ; select the 16bit variable plot1
```

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

```
SNOOPer.SESelect 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.SESelect** command.

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



The screenshot shows the SNOOPer.List window with the following data:

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	0F7D5030	\\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.

# Logging only Data Changes

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

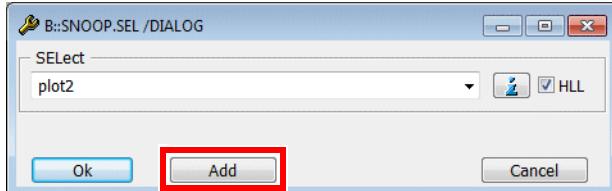
The image shows two windows of the SNOOPer software. The top window is titled 'B::SNOOPer.state' and contains configuration settings for a trace. The 'Mode' section is expanded, showing options for 'AddressTrace', 'Changes' (which is checked and highlighted with a red box), and 'SLAVE'. The bottom window is titled 'B::SNOOPer.List' and displays a table of trace data. The table has columns: record, run, address, cycle, data, symbol, and ti.back. The data shows multiple entries for address SD:20000084, with the 'symbol' column indicating various memory locations and the 'ti.back' column showing times ranging from 38.360us to 577.180us.

record	run	address	cycle	data	symbol	ti.back
-0000000021		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	577.310us
-0000000020		SD:20000084	snoop	A87DB722	\demo_sram\sieve\mstatic1	38.870us
-0000000019		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	576.860us
-0000000018		SD:20000084	snoop	3D66DE80	\demo_sram\sieve\mstatic1	38.360us
-0000000017		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	577.080us
-0000000016		SD:20000084	snoop	A3688F8A	\demo_sram\sieve\mstatic1	38.620us
-0000000015		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	577.880us
-0000000014		SD:20000084	snoop	059F2AF8	\demo_sram\sieve\mstatic1	38.600us
-0000000013		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	576.980us
-0000000012		SD:20000084	snoop	3DD6D8B2	\demo_sram\sieve\mstatic1	38.620us
-0000000011		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	576.720us
-0000000010		SD:20000084	snoop	A8384FB0	\demo_sram\sieve\mstatic1	38.600us
-0000000009		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	576.760us
-0000000008		SD:20000084	snoop	3A8C6C9A	\demo_sram\sieve\mstatic1	38.840us
-0000000007		SD:20000084	snoop	0000000C	\demo_sram\sieve\mstatic1	577.180us

## SNOOPer.Mode Changes ON

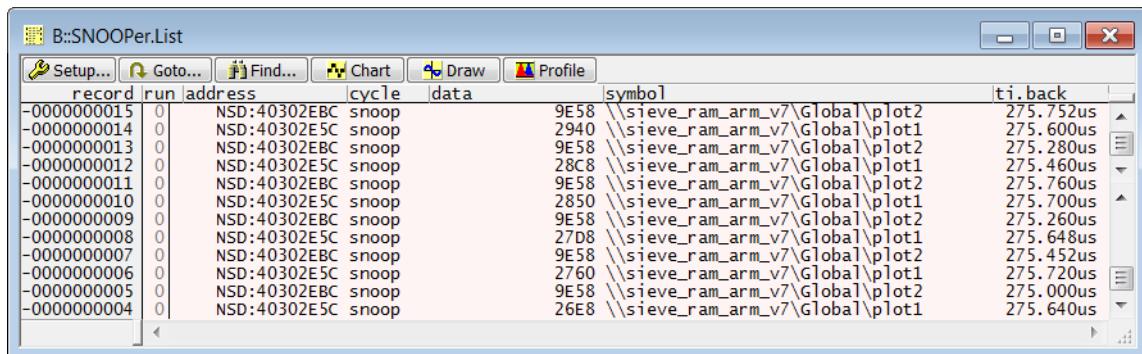
# Logging Multiple Variables

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



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 plot1
```



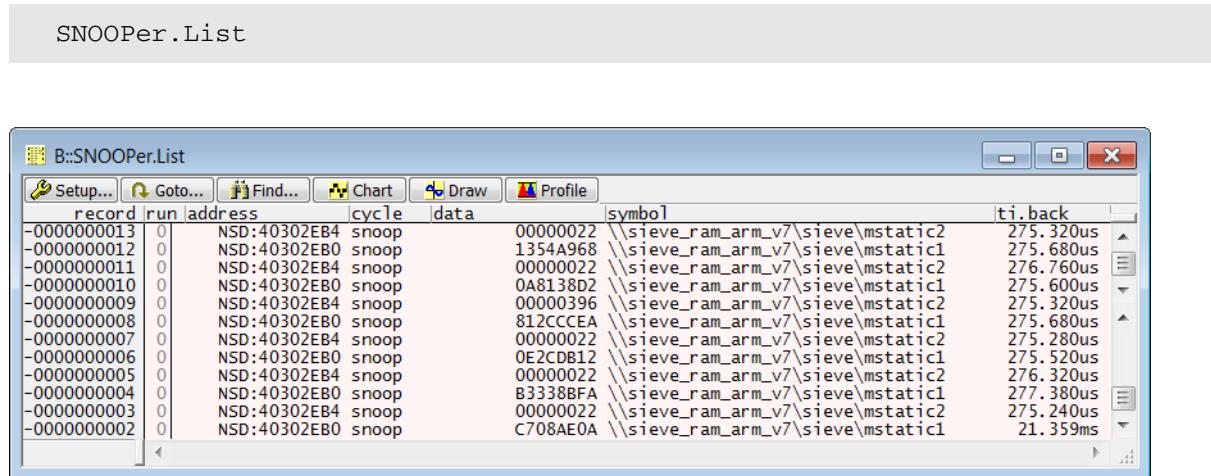
record	run	address	cycle	data	symbol	ti.back
-0000000015	0	NSD:40302EBC	snoop	9E58	\\sieve_ram_arm_v7\Global\plot2	275.752us
-0000000014	0	NSD:40302E5C	snoop	2940	\\sieve_ram_arm_v7\Global\plot1	275.600us
-0000000013	0	NSD:40302EBC	snoop	9E58	\\sieve_ram_arm_v7\Global\plot2	275.280us
-0000000012	0	NSD:40302E5C	snoop	28C8	\\sieve_ram_arm_v7\Global\plot1	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
-0000000008	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
-0000000006	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

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.

# Display the SNOOPer Trace Results

## 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.



The screenshot shows the SNOOPer.List window with the title bar 'SNOOPer.List'. The window contains a table with the following columns: record, run, address, cycle, data, symbol, and ti.back. The data table is as follows:

record	run	address	cycle	data	symbol	ti.back
-0000000013	0	NSD:40302EB4	snoop	00000022	\sieve_ram_arm_v7\sieve\mstatic2	275.320us
-0000000012	0	NSD:40302EB0	snoop	1354A968	\sieve_ram_arm_v7\sieve\mstatic1	275.680us
-0000000011	0	NSD:40302EB4	snoop	00000022	\sieve_ram_arm_v7\sieve\mstatic2	276.760us
-0000000010	0	NSD:40302EB0	snoop	0A8138D2	\sieve_ram_arm_v7\sieve\mstatic1	275.600us
-0000000009	0	NSD:40302EB4	snoop	000000396	\sieve_ram_arm_v7\sieve\mstatic2	275.320us
-0000000008	0	NSD:40302EB0	snoop	812CCCEA	\sieve_ram_arm_v7\sieve\mstatic1	275.680us
-0000000007	0	NSD:40302EB4	snoop	00000022	\sieve_ram_arm_v7\sieve\mstatic2	275.280us
-0000000006	0	NSD:40302EB0	snoop	0E2CDB10	\sieve_ram_arm_v7\sieve\mstatic1	275.520us
-0000000005	0	NSD:40302EB4	snoop	00000022	\sieve_ram_arm_v7\sieve\mstatic2	276.320us
-0000000004	0	NSD:40302EB0	snoop	B3338BFA	\sieve_ram_arm_v7\sieve\mstatic1	277.380us
-0000000003	0	NSD:40302EB4	snoop	00000022	\sieve_ram_arm_v7\sieve\mstatic2	275.240us
-0000000002	0	NSD:40302EB0	snoop	C708AE0A	\sieve_ram_arm_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 **TTime.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 with its HLL representation together with the time relative to the previous record:

```
SNOOPer.List Var TIme.Back
```

```
; list the recorded variable in
; its HLL representation together
; with the time relative to the
; previous record
```

record	var	ti.back
-0000000006	vdouble = -6.5500286323999	97.980us
-0000000005	vdouble = 1.600000000000000	97.760us
-0000000004	vdouble = 754.058470600000	1.157ms
-0000000003	vdouble = 2.26217540859999	97.440us
-0000000002	vdouble = 4.52435081560000	97.720us
-0000000001	vdouble = 7.54058469160000	97.640us

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
```

record	ti.back	var	data
-0000000005	97.760us	vdouble = 1.600000000000000	3FF999999999999A
-0000000004	1.157ms	vdouble = 754.0584706000002e+6	41C67902734CCCCD
-0000000003	97.440us	vdouble = 2.26217540859999e+9	41E0DAC1D6133333
-0000000002	97.720us	vdouble = 4.524350815600004e+9	41F0DAC1D5F9999A
-0000000001	97.640us	vdouble = 7.540584691600004e+9	41FC17430F39999A
T0000000000	97.180us	vdouble = 1.6000000000000001	3FF999999999999A

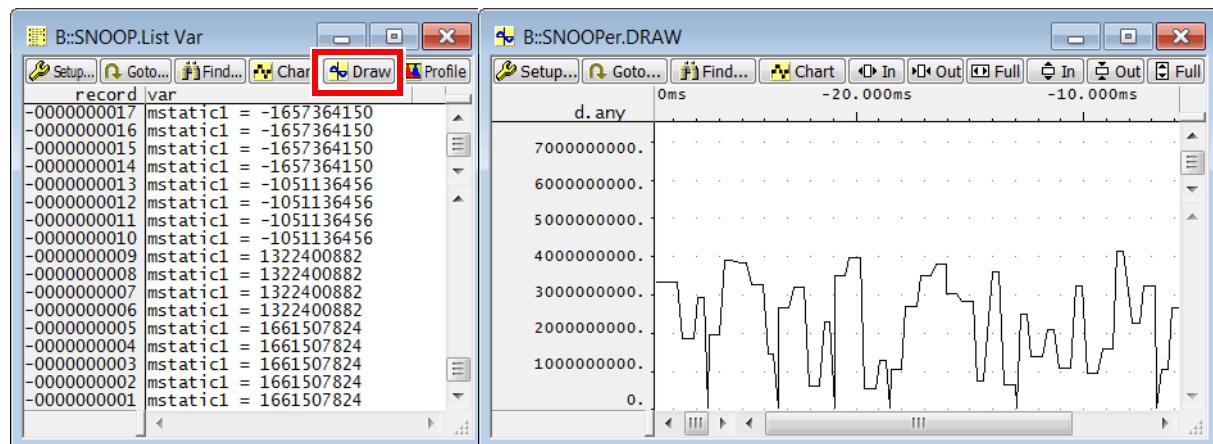
Or display the default parameters together with the time relative to the start to the sampling:

```
SNOOPer.List DEFault TIme.Zero
```

record	run	address	cycle	data	symbol	ti.back	ti.zero
-0000021142		SD:20000D84	snoop	00000C36	\\demo_sram\\sieve\\mstatic1	38.400us	15.164s
-0000021141		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.660us	15.164s
-0000021140		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.420us	15.164s
-0000021139		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.420us	15.164s
-0000021138		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.660us	15.164s
-0000021137		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.400us	15.164s
-0000021136		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.420us	15.164s
-0000021135		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.420us	15.164s
-0000021134		SD:20000D84	snoop	00001452	\\demo_sram\\sieve\\mstatic1	38.420us	15.164s

# Graphical Display of SNOOPer Trace Results

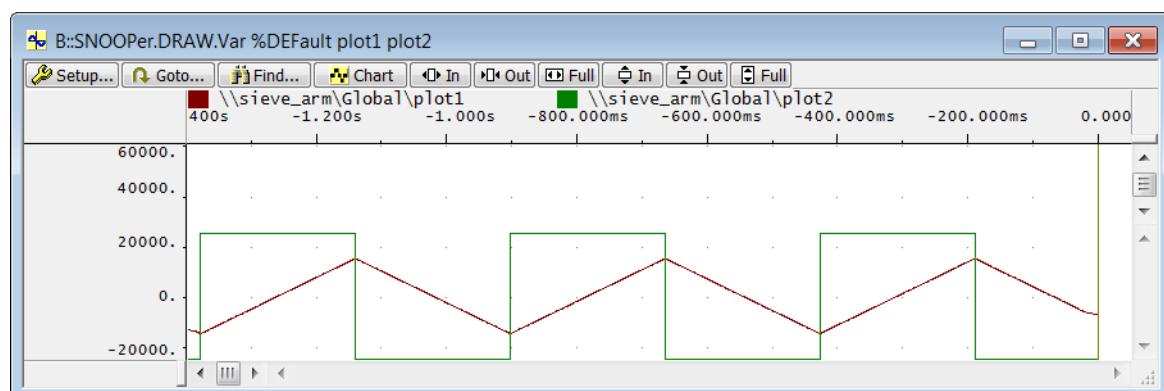
You can use the **Draw** button from the **SNOOPer.List** window to display the sampled data values graphically. Please refer to the documentation of the [\*\*<trace>.DRAW\*\*](#) command group for more information.



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
```

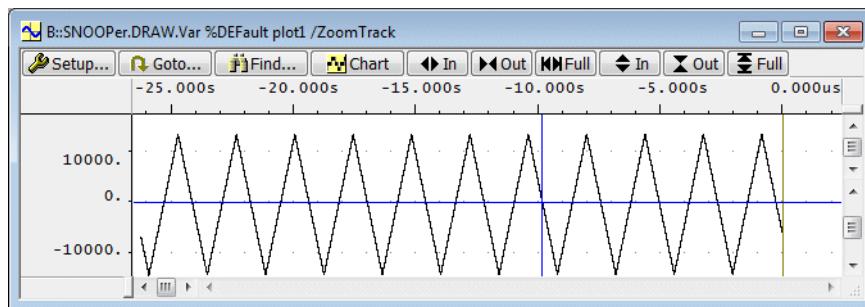


Displaying all variables in one single graph doesn't always make sense, especially if they have different value ranges. In this case, it makes more sense to display each variable in a separate window. By adding the **/ZoomTrack** Option to the **SNOOPer.DRAW.Var** command, a time and zoom synchronisation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

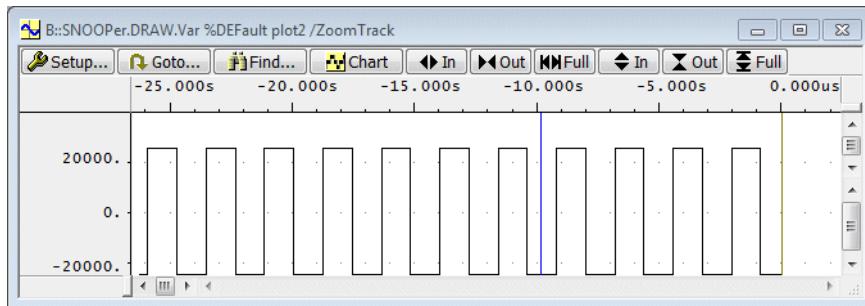
```
SNOOPer.DRAW.Var %DEFault plot1 /ZoomTrack      ; the option ZoomTrack
; establishes time- and
; zoom-synchronisation
; between display windows

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

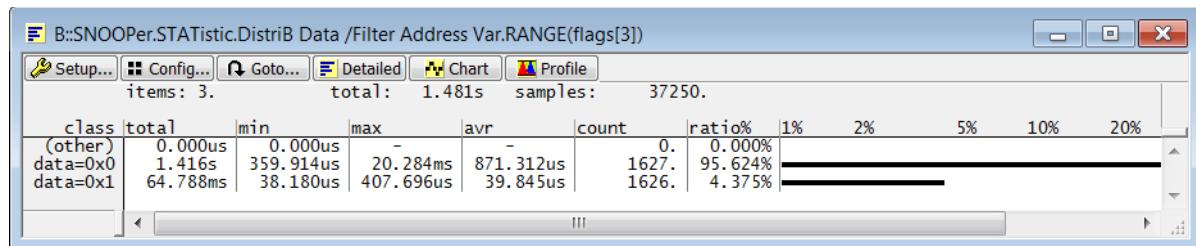


TRACE32 additionally allows to display the SNOOPer trace results as statistical distributions.

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

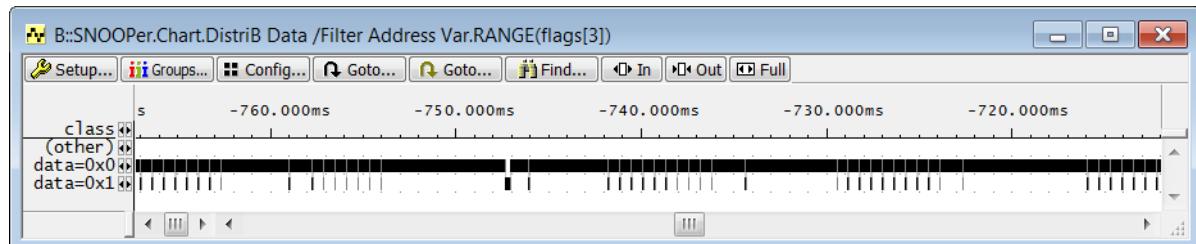
**Example:** We sample the element with index three of the flags array of type char (flags[3]). We can use the following command to display a statistical distribution of the sampled data values:

```
; 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])
```

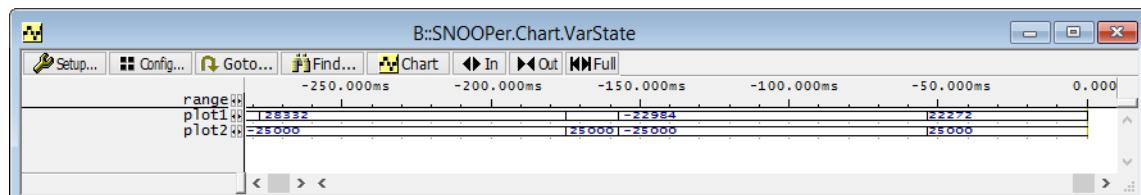


These results can also be displayed as time chart using the command **SNOOPer.Chart.DistriB** e.g.:

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

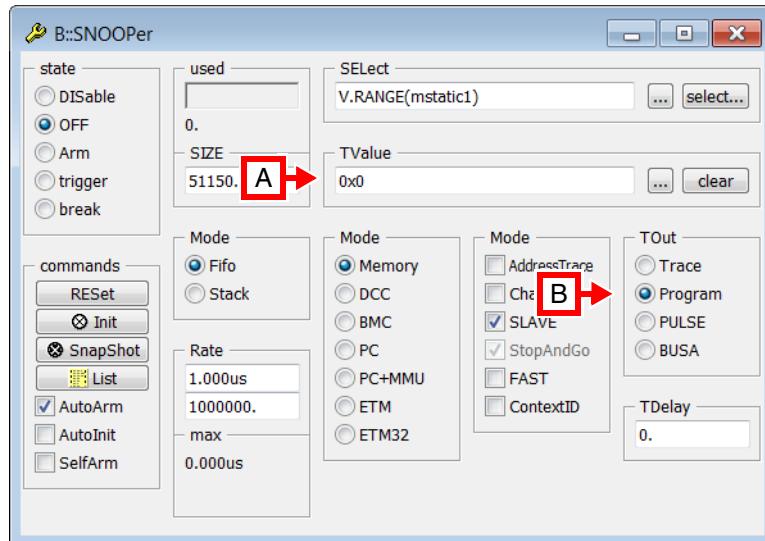


Using the command **SNOOPer.Chart.VarState**, you can additionally have a graphical representation in time for the taking values of the sampled addresses.



# SNOOPer Trace Trigger

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.



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.

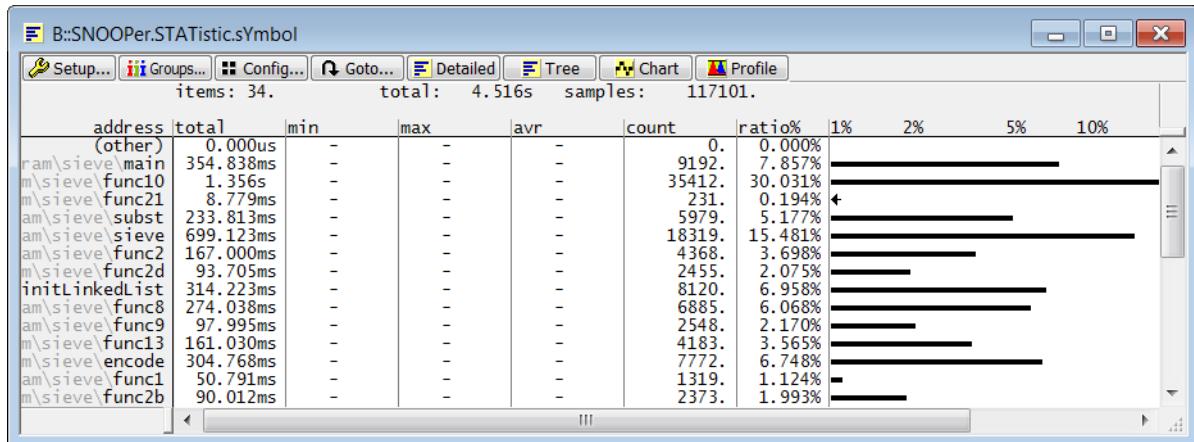
record	run	address	cycle	data	symbol	ti.back
-0000000004		NSD:403031A8	snoop	4F3AD148	\\sieve_arm\\sieve\\mstatic1	276.420us
-0000000003		NSD:403031A8	snoop	6786FE22	\\sieve_arm\\sieve\\mstatic1	278.100us
-0000000002		NSD:403031A8	snoop	6786FE22	\\sieve_arm\\sieve\\mstatic1	276.400us
-0000000001		NSD:403031A8	snoop	72CCEB80	\\sieve_arm\\sieve\\mstatic1	276.460us
T000000000000		NSD:403031A8	snoop	00000000	\\sieve_arm\\sieve\\mstatic1	276.160us

	<p>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.</p>
---	---

# Sampling the Program Counter

The SNOOPer trace allows to monitor the actual program counter. This mode can be used e.g. for

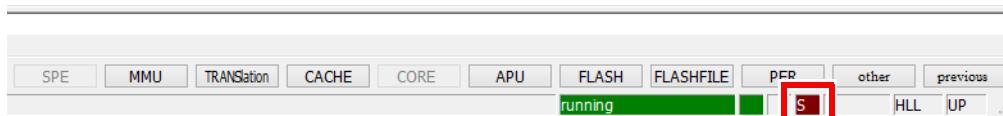
- Sample-based flat run-time analysis. Please also consider using the **PERF** command group for this purpose.
- Post-mortem debugging: if the target system crashes, it is generally not possible to halt the processor in order to find the location of the crash. However, it is often still possible in such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



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 run-time (e.g. EDPCSR for Cortex-A/R (Armv8), Quick Access for RH850).

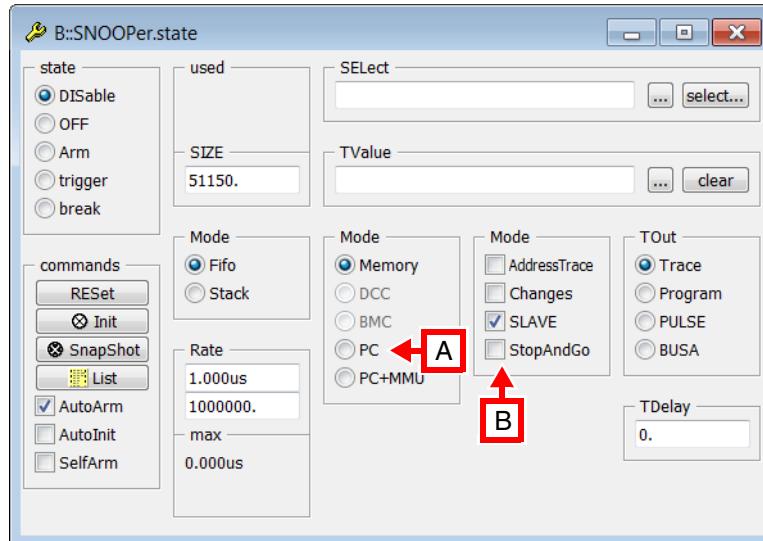
Otherwise, the SNOOPer trace will shortly stop the program execution to read the current program counter and resume again. A red S will then appear in the **state line** while recording.



# Setup

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:

SNOOPer.Mode PC ; Set the PC mode



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

record	run	address	cycle	data	symbol	ti.back
-0000000014	0	R:40300F20	snoop		\\sieve_ram_arm_v7\\sieve\\func13	1.417ms
-0000000013	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.403ms
-0000000012	0	R:40300754	snoop		..\\sieve_ram_arm_v7\\sieve\\func8	1.402ms
-0000000011	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.405ms
-0000000010	0	R:40301514	snoop		..v7\\sieve\\init_linked_list+0xA8	1.407ms
-0000000009	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.402ms
-0000000008	0	R:4030257C	snoop		..rm_v7\\Global\\_aeabi_dmul+0x20	1.400ms
-0000000007	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.406ms
-0000000006	0	R:40301EB0	snoop		..ve_ram_arm_v7\\sieve\\sieve+0x90	1.402ms
-0000000005	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.399ms
-0000000004	0	R:40301650	snoop		\\sieve_ram_arm_v7\\sieve\\subst	1.402ms
-0000000003	1	R:40301F10	snoop		..m_arm_v7\\sieve\\background+0x40	1.404ms

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

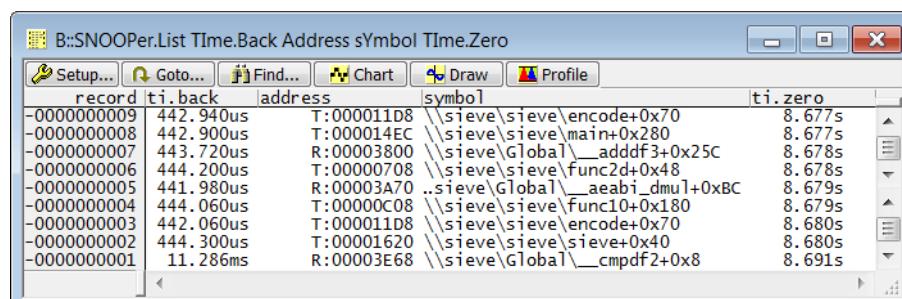
- **run**: displays the core number for SMP systems. This column is empty for single core processors.
- **address**: the sampled program counter value.
- **cycle**: snoop.
- **data**: this column is empty.
- **symbol**: the symbolic information with path and offset corresponding to the sampled program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

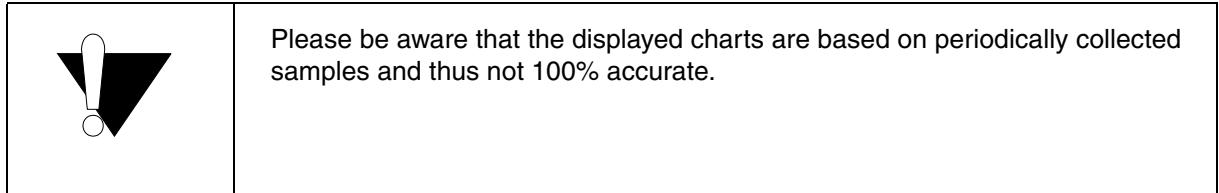
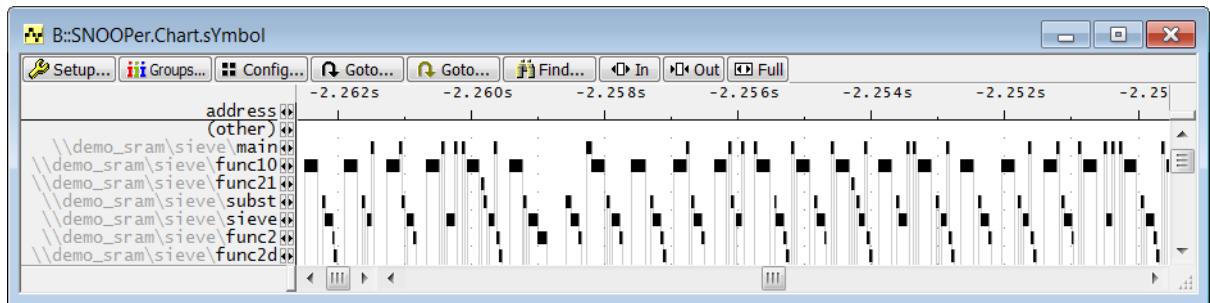
### Example:

```
SNOOPeR.List TIme.Back Address sYmbol TIme.Zero
```



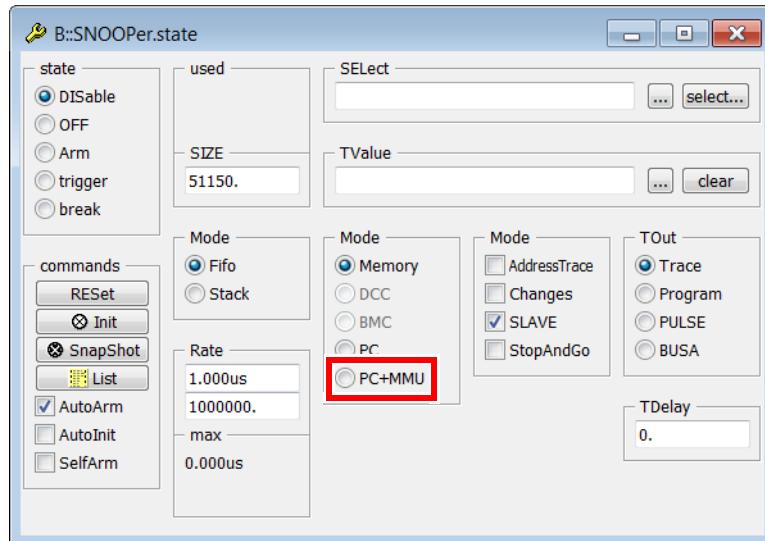
record	ti.back	address	symbol	ti.zero
-0000000009	442.940us	T:000011D8	\sieve\sieve\encode+0x70	8.677s
-0000000008	442.900us	T:000014EC	\sieve\sieve\main+0x280	8.677s
-0000000007	443.720us	R:00003800	\sieve\Global\_\_adddf3+0x25C	8.678s
-0000000006	444.200us	T:00000708	\sieve\sieve\func2d+0x48	8.678s
-0000000005	441.980us	R:00003A70	..sieve\Global\_\_aeabi_dmul+0xBC	8.679s
-0000000004	444.060us	T:00000C08	\sieve\sieve\func10+0x180	8.679s
-0000000003	442.060us	T:000011D8	\sieve\sieve\encode+0x70	8.680s
-0000000002	444.300us	T:00001620	\sieve\sieve\sieve+0x40	8.680s
-0000000001	11.286ms	R:00003E68	\sieve\Global\_\_cmpdf2+0x8	8.691s

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**.



# Sampling the Program Counter and the Current Task

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.



**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.

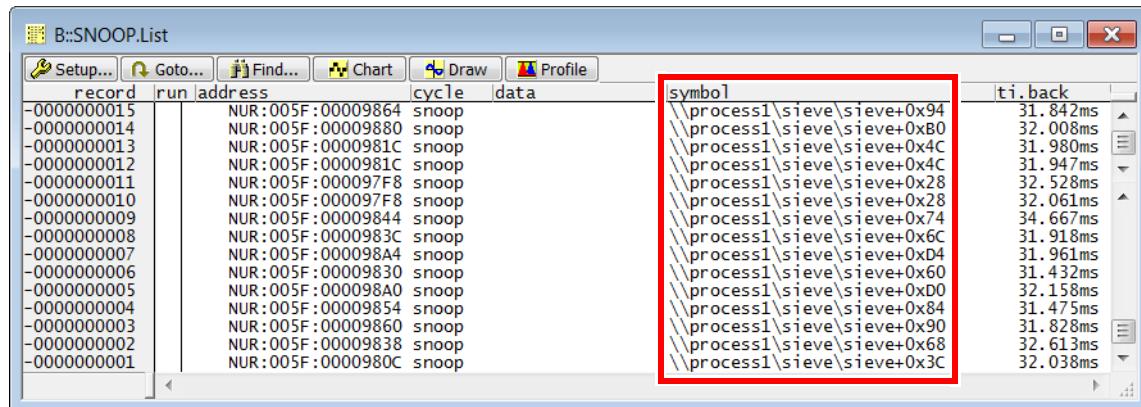
record	run	address	cycle	data	symbol	ti.back
-00000000014		R:0000:00009890		snop		395.140us
-00000000013		R:0000:00009864		snop		394.300us
-00000000012		R:0000:00009864		snop		395.680us
-00000000011		R:0000:000097F4		snop		393.820us
-00000000010		R:0000:0000983C		snop		393.720us
-00000000009		R:0000:00009864		snop		395.320us
-00000000008		R:0000:000097F4		snop		393.880us
-00000000007		R:0000:00009890		snop		394.800us
-00000000006		R:0000:000097F4		snop		394.140us
-00000000005		R:0000:000097F4		snop		393.800us
-00000000004		R:0000:0000989C		snop		394.920us
-00000000003		R:0000:00009890		snop		393.820us
-00000000002		R:0000:000097F4		snop		395.680us
-00000000001		R:0000:0000988C		snop		15.323ms

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.



The screenshot shows a software interface titled 'B:SNOOP.List'. The main window contains a table with columns: record, run, address, cycle, data, symbol, and ti.back. The 'symbol' column is highlighted with a red box. The 'ti.back' column shows times in milliseconds. The data in the 'symbol' column is as follows:

symbol	ti.back
\\process1\\sieve\\sieve+0x94	31.842ms
\\process1\\sieve\\sieve+0xB0	32.008ms
\\process1\\sieve\\sieve+0x4C	31.980ms
\\process1\\sieve\\sieve+0x4C	31.947ms
\\process1\\sieve\\sieve+0x28	32.528ms
\\process1\\sieve\\sieve+0x28	32.061ms
\\process1\\sieve\\sieve+0x74	34.667ms
\\process1\\sieve\\sieve+0x6C	31.918ms
\\process1\\sieve\\sieve+0xD4	31.961ms
\\process1\\sieve\\sieve+0x60	31.432ms
\\process1\\sieve\\sieve+0xD0	32.158ms
\\process1\\sieve\\sieve+0x84	31.475ms
\\process1\\sieve\\sieve+0x90	31.828ms
\\process1\\sieve\\sieve+0x68	32.613ms
\\process1\\sieve\\sieve+0x3C	32.038ms

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

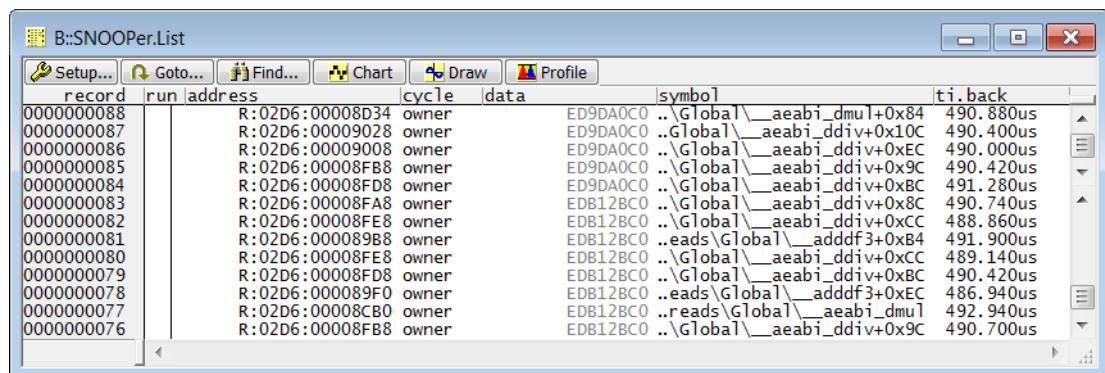
For Arm processors supporting reading the Context ID register on run-time (e.g. Cortex-A15), by enabling the **SNOOPer.Mode ContextID** mode, the Context ID register can be sampled instead of the memory address containing the current task identifier (task magic number). This way, the sampling can be 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 are 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:SNOOPer.List



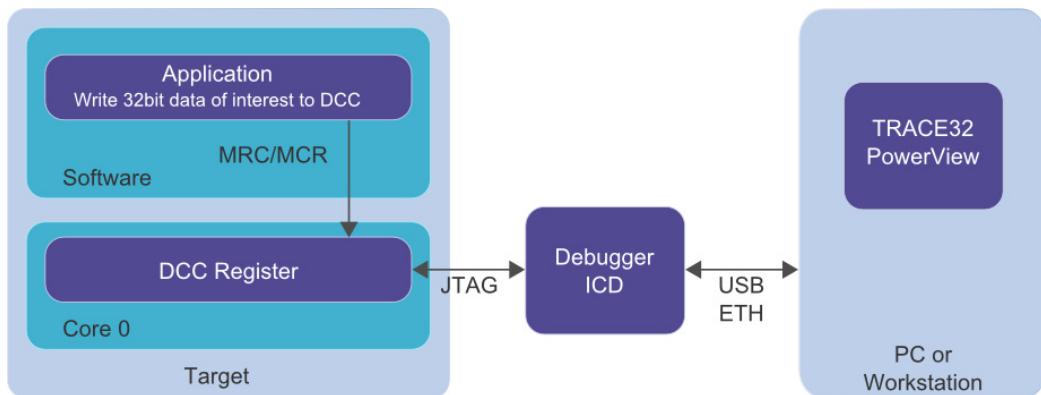
record	run	address	cycle	data	symbol	ti.back
0000000088		R:02D6:00008D34	owner	ED9DA0C0	..\Global\__aeabi_dmul+0x84	490.880us
0000000087		R:02D6:00009028	owner	ED9DA0C0	..\Global\__aeabi_ddiv+0x10C	490.400us
0000000086		R:02D6:00009008	owner	ED9DA0C0	..\Global\__aeabi_ddiv+0xEC	490.000us
0000000085		R:02D6:00008FB8	owner	ED9DA0C0	..\Global\__aeabi_ddiv+0x9C	490.420us
0000000084		R:02D6:00008FD8	owner	ED9DA0C0	..\Global\__aeabi_ddiv+0xBC	491.280us
0000000083		R:02D6:00008FA8	owner	EDB12BC0	..\Global\__aeabi_ddiv+0x8C	490.740us
0000000082		R:02D6:00008FE8	owner	EDB12BC0	..\Global\__aeabi_ddiv+0xCC	488.860us
0000000081		R:02D6:000089B8	owner	EDB12BC0	..\eads\Global\__adddf3+0xB4	491.900us
0000000080		R:02D6:00008FE8	owner	EDB12BC0	..\Global\__aeabi_ddiv+0xCC	489.140us
0000000079		R:02D6:00008FD8	owner	EDB12BC0	..\Global\__aeabi_ddiv+0xBC	490.420us
0000000078		R:02D6:000089F0	owner	EDB12BC0	..\eads\Global\__adddf3+0xEC	486.940us
0000000077		R:02D6:00008CB0	owner	EDB12BC0	..\reads\Global\__aeabi_dmul	492.940us
0000000076		R:02D6:00008FB8	owner	EDB12BC0	..\Global\__aeabi_ddiv+0x9C	490.700us

# Data Sampling via Debug Communication Channel

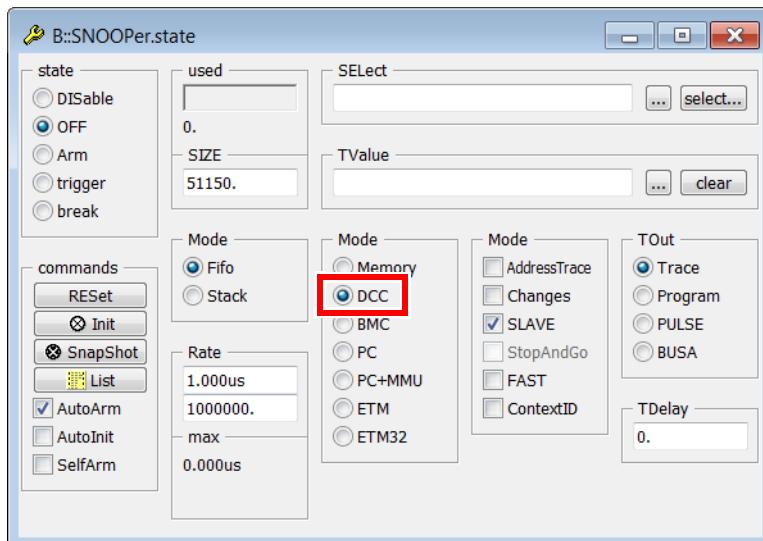
The **Debug Communication Channel** - short **DCC** - is a characteristic of the on-chip debugging support. It allows to pass information between the application program on the target and the debugger. For details refer to your CPU manual.

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:



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](mailto: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
}
```

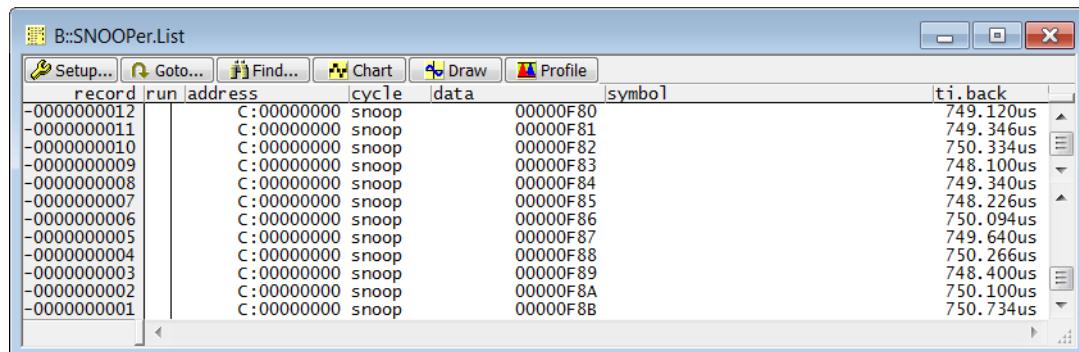
If you plan to use the SNOOPer via DCC, you have to be aware of the following:

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  $\mu$ s.

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.



record	run	address	cycle	data	symbol	ti.back
-0000000012		C:00000000	snoop	00000F80		749.120us
-0000000011		C:00000000	snoop	00000F81		749.346us
-0000000010		C:00000000	snoop	00000F82		750.334us
-0000000009		C:00000000	snoop	00000F83		748.100us
-0000000008		C:00000000	snoop	00000F84		749.340us
-0000000007		C:00000000	snoop	00000F85		748.226us
-0000000006		C:00000000	snoop	00000F86		750.094us
-0000000005		C:00000000	snoop	00000F87		749.640us
-0000000004		C:00000000	snoop	00000F88		750.266us
-0000000003		C:00000000	snoop	00000F89		748.400us
-0000000002		C:00000000	snoop	00000F8A		750.100us
-0000000001		C:00000000	snoop	00000F8B		750.734us

# Sampling Benchmark Counters

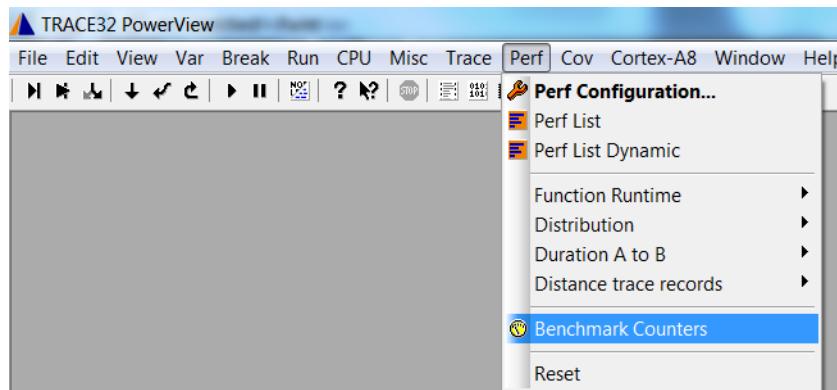
Benchmark counters are on-chip counters that count specific hardware events e.g. the number of executed instructions or number of cache misses. Please refer to your [Processor Architecture Manual](#) to check if your target processor supports benchmark counters.

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.

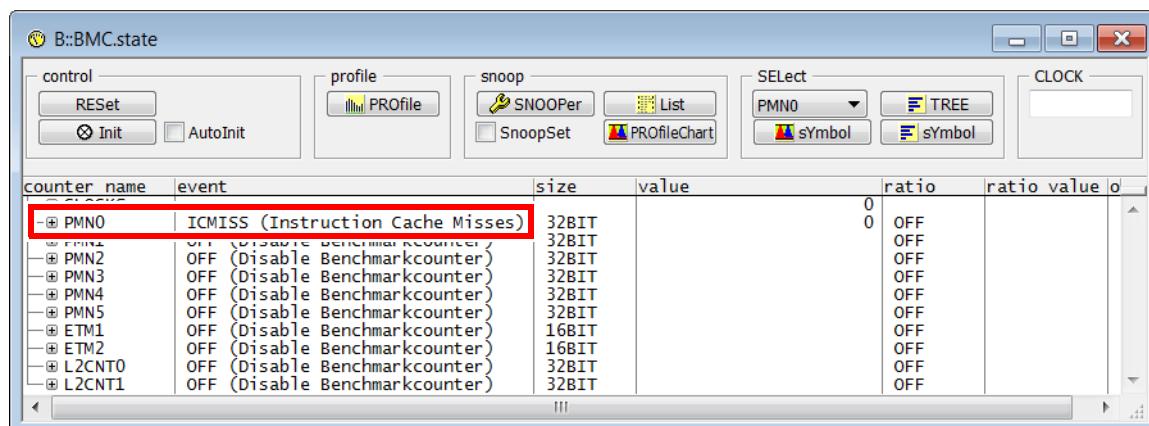
The benchmark counters can be configured in TRACE32 using the **BMC** (BenchMark Counter) command group.

**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.



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

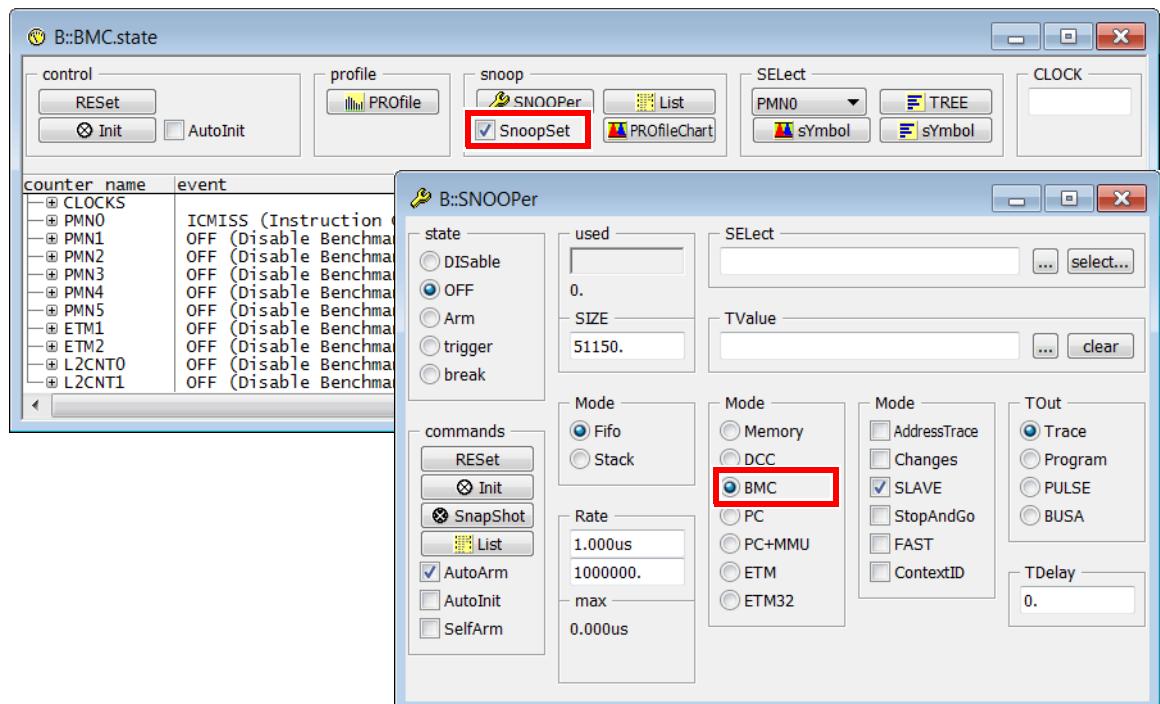
```
BMC.<counter1> <event1> ; assign event of interest to  
; the event counter  
  
BMC.<counter2> <event2> ; several assignments possible  
...
```

The syntax of the commands is architecture-specific. Please refer to your [Processor Architecture Manual](#) for more information.

#### Example (Arm):

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

3. 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.



The respective TRACE32 commands are

```
BMC.SnoopSet ON
```

and

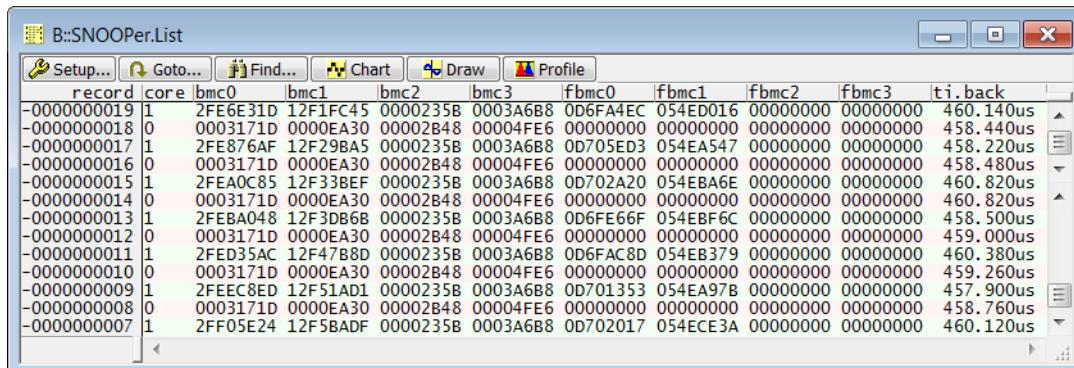
SNOOPer.Mode BMC

**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.



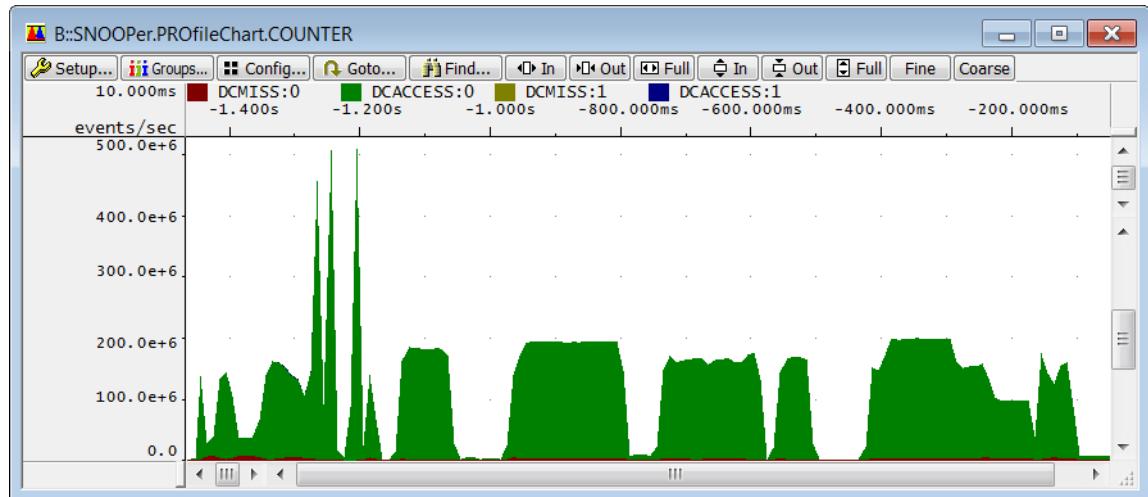
The screenshot shows a Windows application window titled "B:SNOOPer.List". The window has a menu bar with "Setup...", "Goto...", "Find...", "Chart", "Draw", and "Profile". The main area is a data grid with the following columns: record, core, bmc0, bmc1, bmc2, bmc3, fbmc0, fbmc1, fbmc2, fbmc3, and ti.back. The data grid contains approximately 20 rows of sampled counter values. The "core" column shows values 1 and 0, indicating two cores. The "ti.back" column shows values ranging from 458.440us to 460.900us.

record	core	bmc0	bmc1	bmc2	bmc3	fbmc0	fbmc1	fbmc2	fbmc3	ti.back
-0000000019	1	2FE6E31D	12F1FC45	00002358	0003A688	066FA4EC	054E016	00000000	00000000	460.140us
-0000000018	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.440us
-0000000017	1	2FE876AF	12F29BA5	00002358	0003A688	0705ED3	054EA547	00000000	00000000	458.220us
-0000000016	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.480us
-0000000015	1	2FEAC085	12F338EF	00002358	0003A688	0702A20	054EBA6E	00000000	00000000	460.820us
-0000000014	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	460.820us
-0000000013	1	2FEB0408	12F3DB6B	00002358	0003A688	06F6E6F	054EBF6C	00000000	00000000	458.500us
-0000000012	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	459.000us
-0000000011	1	2FED35AC	12F47B8D	00002358	0003A688	0D6FAC8D	054EB379	00000000	00000000	460.380us
-0000000010	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	459.260us
-0000000009	1	2FFEC8ED	12F51A01	00002358	0003A688	0701353	054EA97B	00000000	00000000	457.900us
-0000000008	0	0003171D	0000EA30	00002B48	00004FE6	00000000	00000000	00000000	00000000	458.760us
-0000000007	1	2FF05E24	12F5BADF	00002358	0003A688	0D702017	054ECE3A	00000000	00000000	460.120us

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.

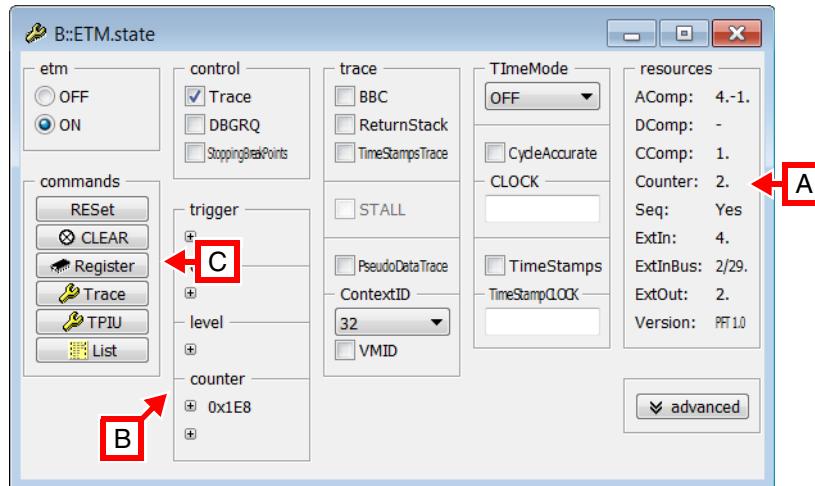
The **SNOOPer.PROfileChart.COUNTER** command can be used to display a graphical profile statistic of the sampled benchmark counter values. The result is a stacked graph i.e. the total number of events/s at a given time represent the sum of the events for all counters at that time.



# Sampling ETM Counters

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**.

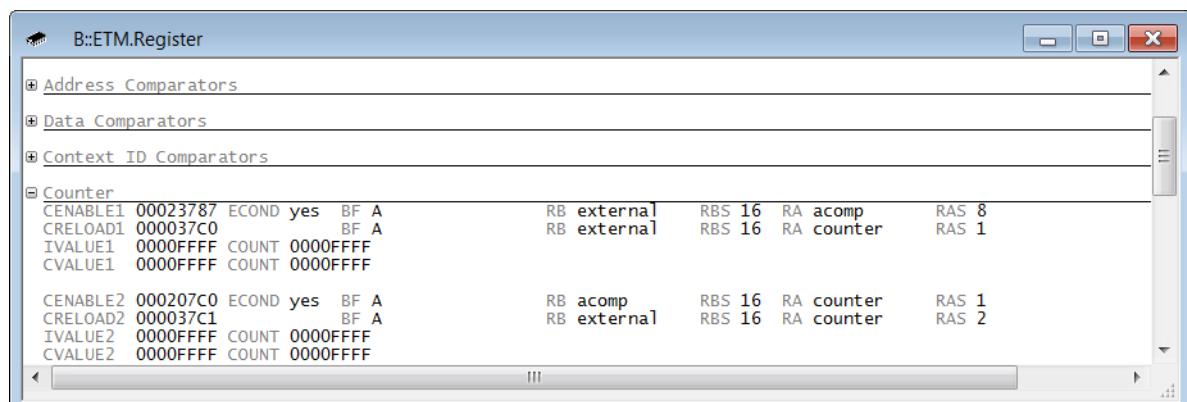


The **ETM.state** window also displays the current values of the ETM counters [B].

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

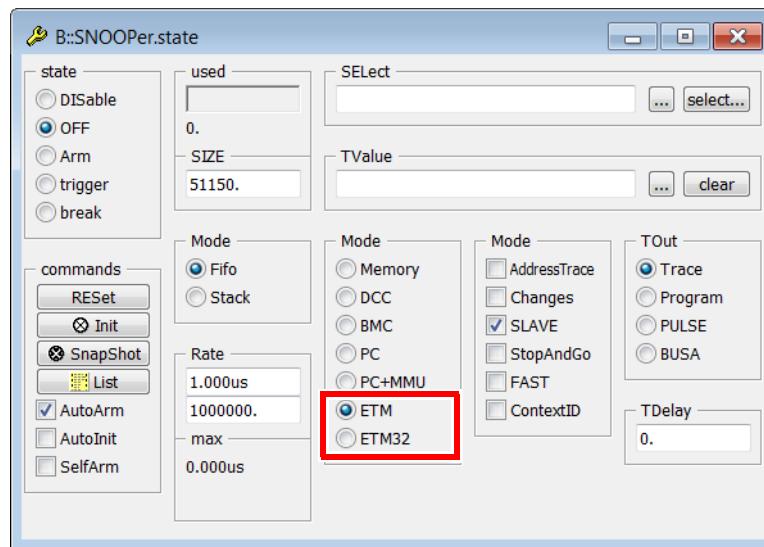
```
ETM.Register "Counter"
```

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



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.



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:

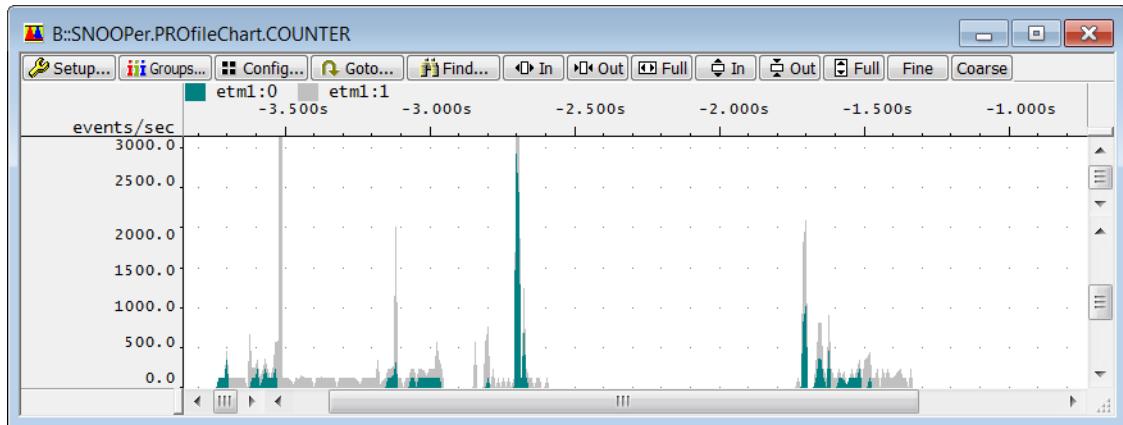
```
SNOOPer.RESet ; reset the SNOOPer
SNOOPer.Mode ETM ; select mode ETM or ETM32
; SNOOPer.Mode ETM32
Break.Set myFunc /BusCount
```

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

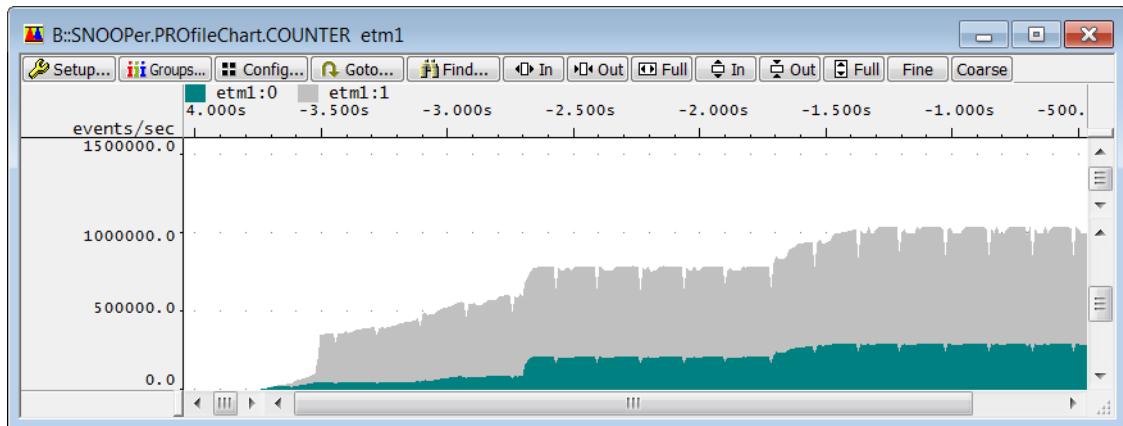
record	core	etm1	fetm1	ti.back
-0000000014	0	00000081	00000000	446.280us
-0000000013	1	0000014D	00000000	446.200us
-0000000012	0	00000081	00000000	447.960us
-0000000011	1	0000014D	00000000	448.020us
-0000000010	0	00000081	00000000	446.920us
-0000000009	1	0000014D	00000000	447.400us
-0000000008	0	00000081	00000000	448.920us
-0000000007	1	0000014D	00000000	449.180us
-0000000006	0	00000081	00000000	446.600us
-0000000005	1	0000014D	00000000	445.900us
-0000000004	0	00000081	00000000	447.340us

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**.



You can display the **etm1** values instead using the following command

```
SNOOPer.PROfileChart.COUNTER etm1
```



**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.

# Save and Load

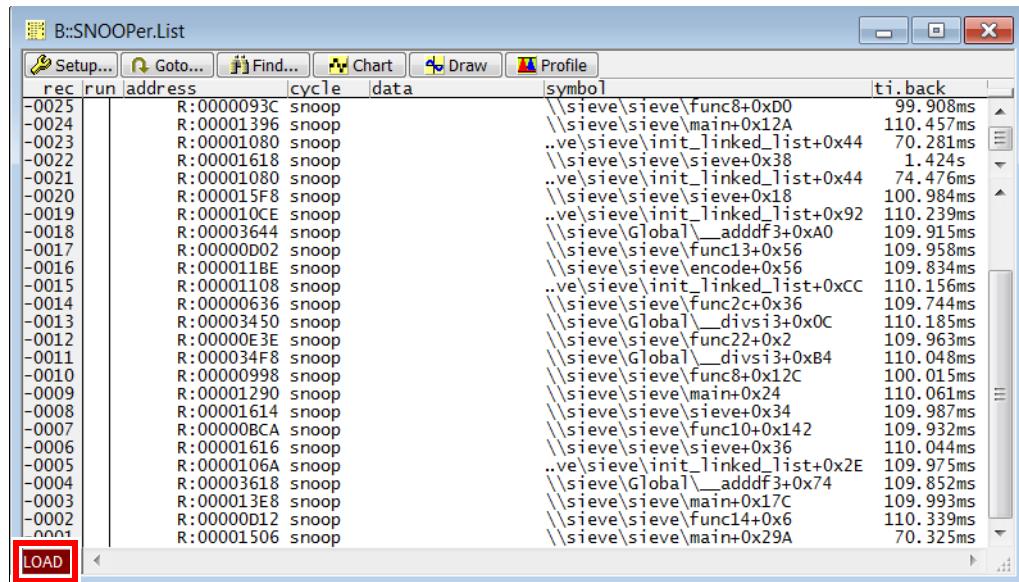
It is possible to save the SNOOPer trace results to a file for postprocessing using the command **SNOOPer.SAVE** e.g.

```
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.



rec	run	address	cycle	data	symbol	ti.back
-0025		R:0000093C	snoop		\sieve\sieve\func8+0x00	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		\sieve\sieve\sieve+0x38	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\Global\_\_adddf3+0xA0	109.915ms
-0017		R:00000D02	snoop		\sieve\sieve\func13+0x56	109.958ms
-0016		R:000011BE	snoop		\sieve\sieve\encode+0x56	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+0xC	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
-0008		R:00001614	snoop		\sieve\sieve\sieve+0x34	109.987ms
-0007		R:000000BCA	snoop		\sieve\sieve\func10+0x142	109.932ms
-0006		R:00001616	snoop		\sieve\sieve\sieve+0x36	110.044ms
-0005		R:0000106A	snoop		..ve\sieve\init_linked_list+0x2E	109.975ms
-0004		R:00003618	snoop		\sieve\Global\_\_adddf3+0x74	109.852ms
-0003		R:000013E8	snoop		\sieve\sieve\main+0x17C	109.993ms
-0002		R:00000D12	snoop		\sieve\sieve\func14+0x6	110.339ms
-0001		R:00001506	snoop		\sieve\sieve\main+0x29A	70.325ms

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
```