LAUTERBACH
DEVELOPMENT TOOLS

Trace Tutorial

Release 02.2024

MANUAL

Trace Tutorial

Trace Tutorial

Version 04-Mar-2024

History

18-Jun-21 New manual.

About the Tutorial

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

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

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

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

What is Trace?

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

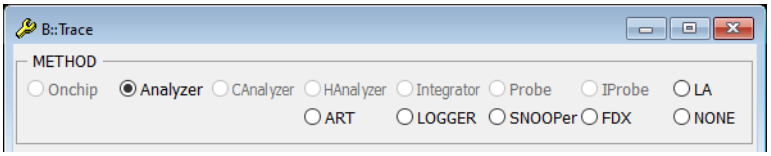
Trace Use Cases

Trace is mainly used in the following cases:

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

Trace Methods

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



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

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

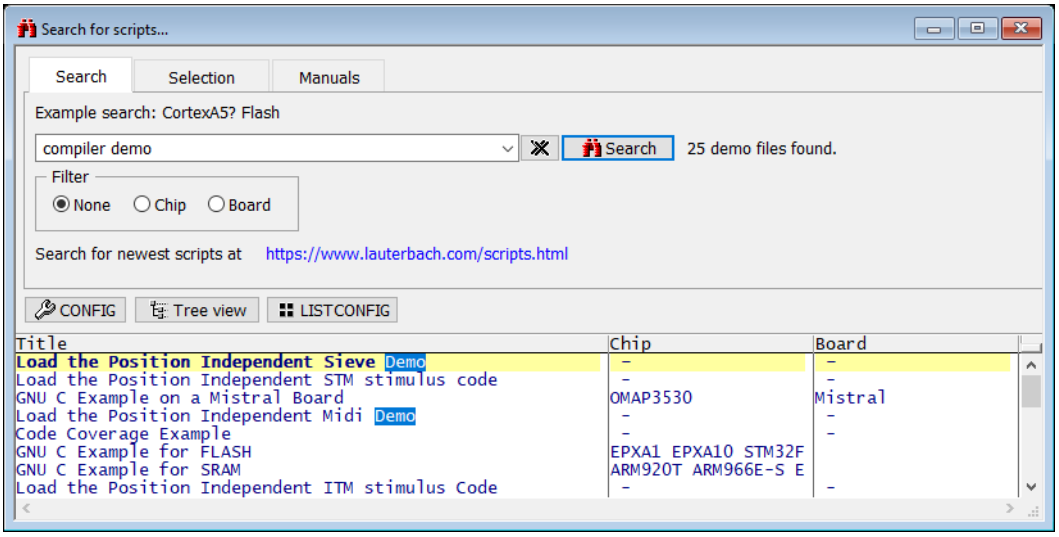
Simulator Demo

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

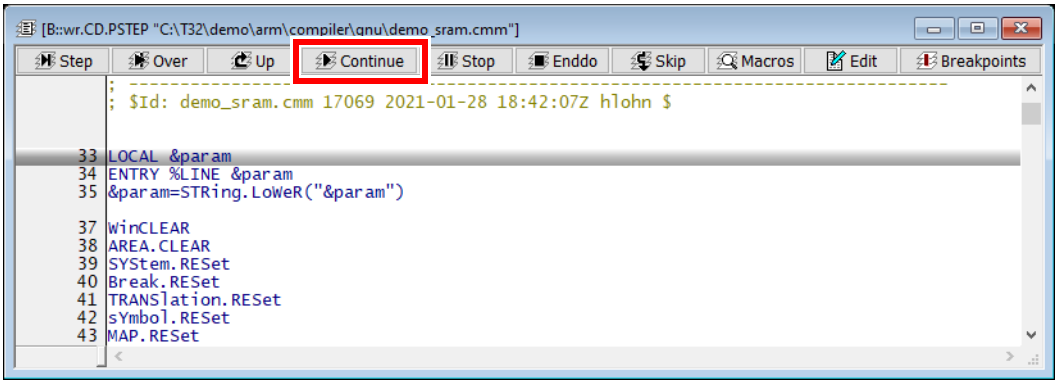
To load a demo on the simulator, follow these steps:

1. Start the script search dialog from the menu **File > Search for scripts…**

2. Enter in the search field “compiler demo”



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

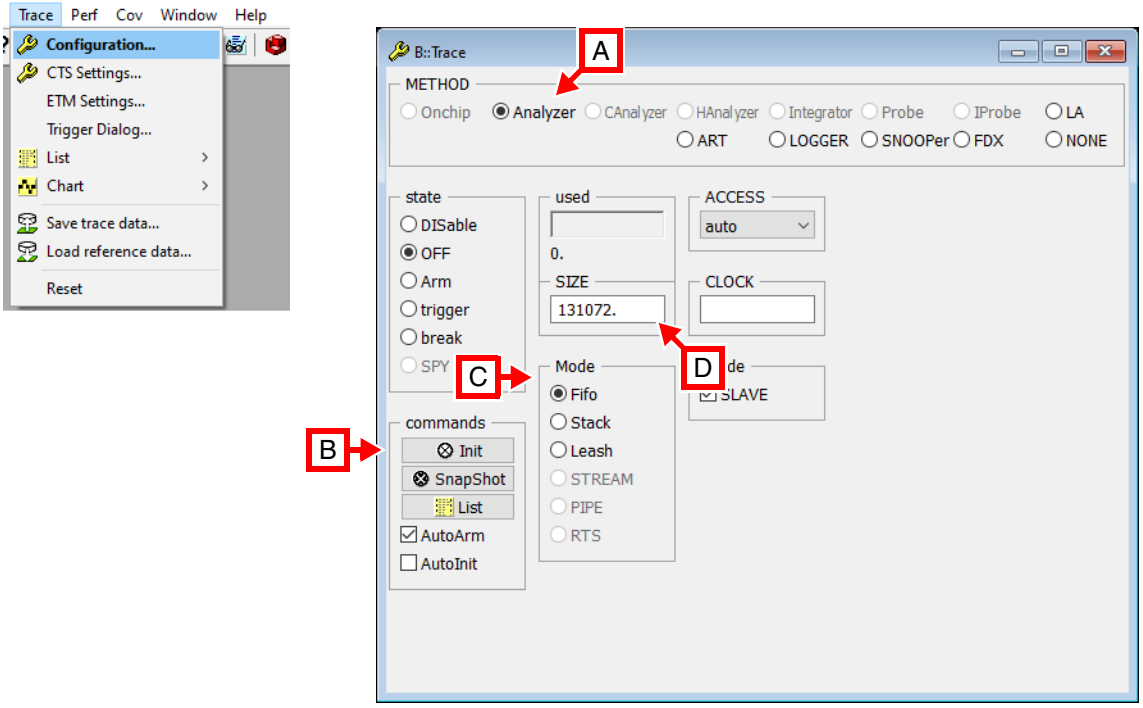


We will use here the demo **“GNU C Example for SRAM”**.

Trace Configuration

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

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



2. Clear the contents of the trace buffer by pressing the **Init** button **[B]**.

3. Select the trace operation mode **[C]**.

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

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

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

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

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

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

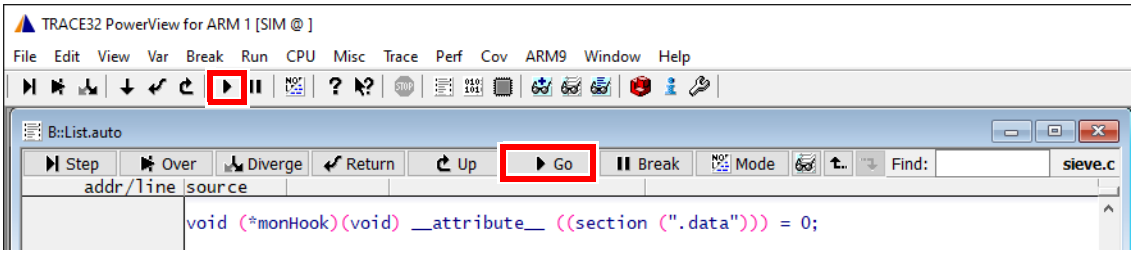
   In order to have a longer trace recording, we will set the trace buffer size to 10000000.

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

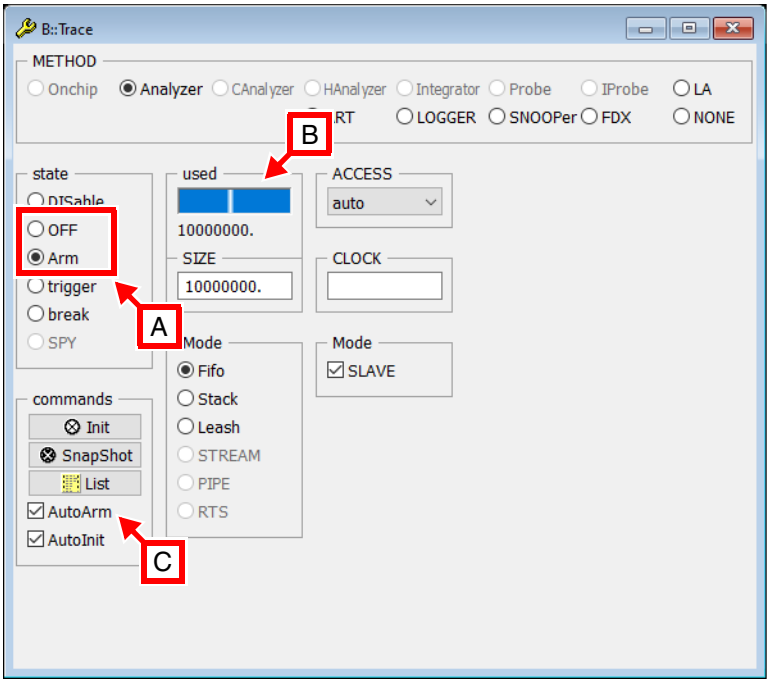
```
Trace.METHOD Analyzer
Trace.Init
Trace.Mode Fifo
Trace.SIZE 10000000.
```

Trace Recording

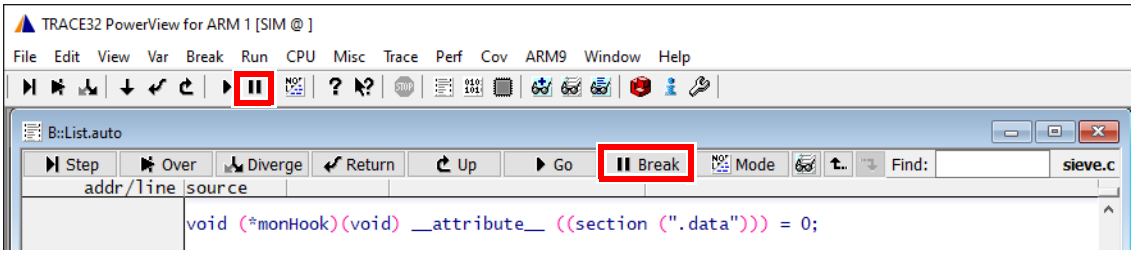
Press the **Go** button to start the program execution.



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



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



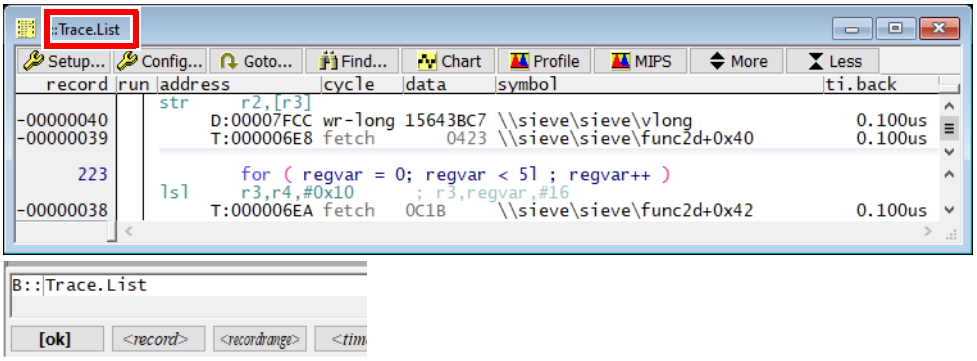
The trace recording is automatically started and stopped when starting and stopping the program execution because of the **AutoArm [C]** setting in the **Trace** window, which is per default enabled. The trace recording can also be started/stopped manually while the program execution is running using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

Displaying the Trace Results

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

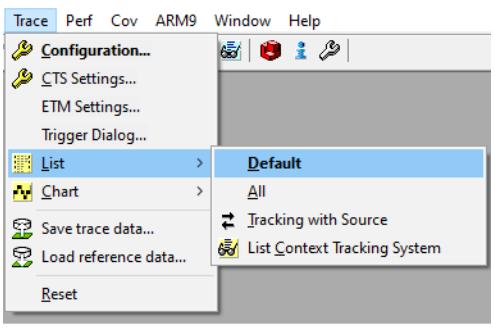
Please note that the trace results can only be displayed if the trace state in the **Trace** window is **OFF**. It is not possible to display the trace results while recording.

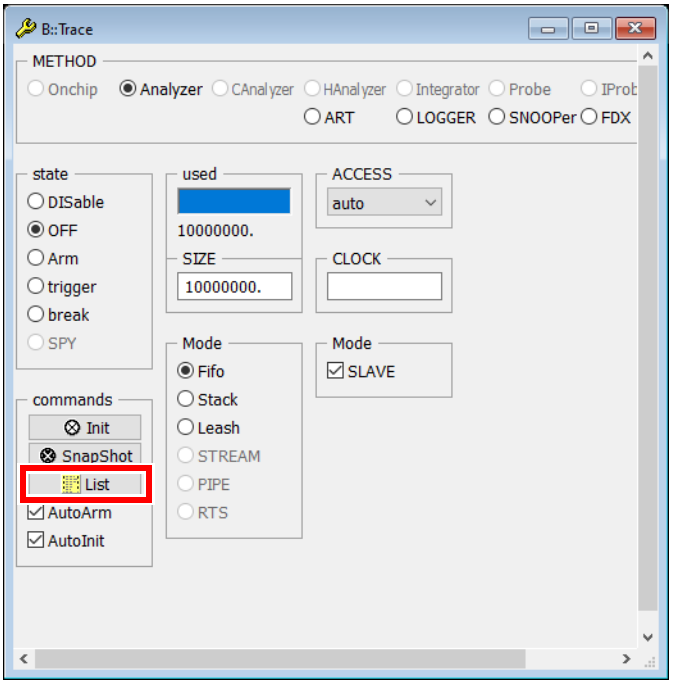
The caption of a TRACE32 window includes the TRACE32 command that can be executed in the TRACE32 command line or in a PRACTICE script to open this window, e.g. here **Trace.List**



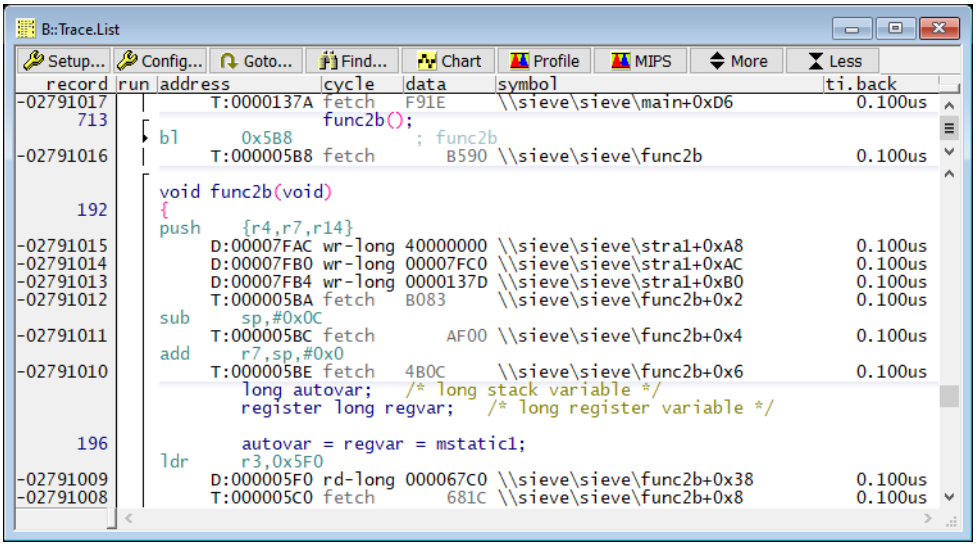
Trace List

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





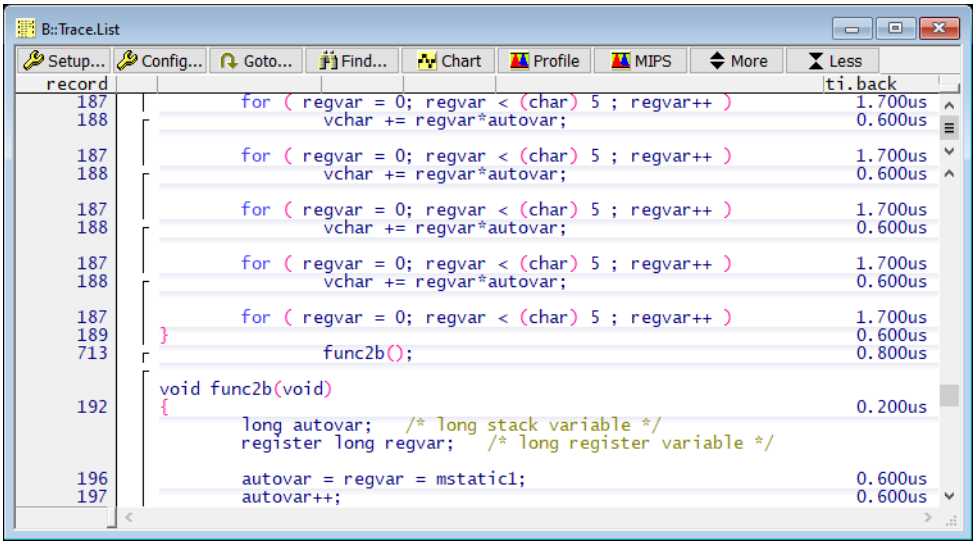
The **Trace.List** window displays the recorded trace packets together with the corresponding assembler and source code.



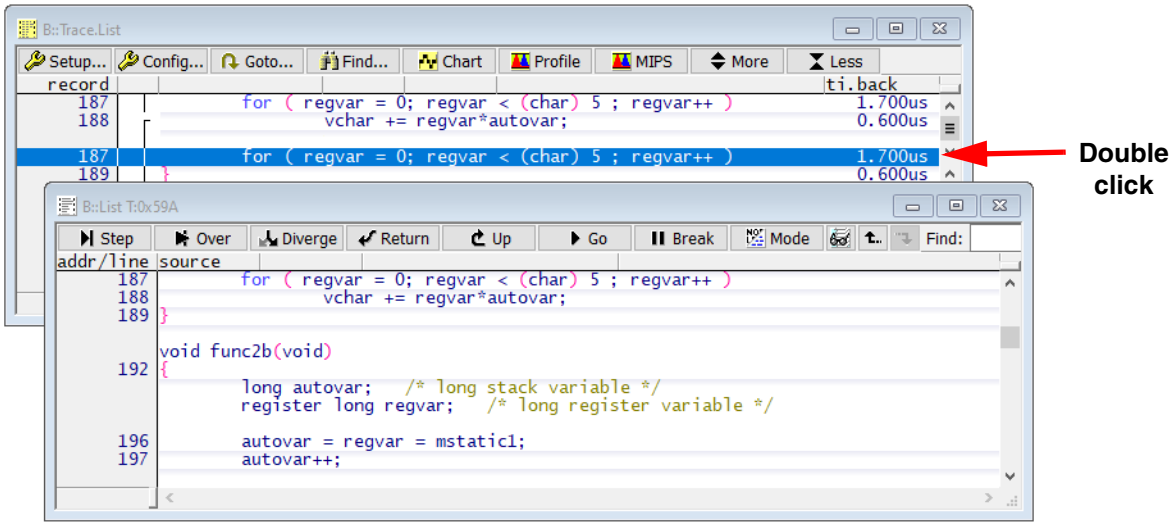
In our case, trace packets are program fetches (cycle fetch) or data accesses (e.g. wr-long and rd-long for 32bit write and read accesses). Each trace packet has a record number displayed in the **record** column. The record number is a negative index for **Fifo** mode.

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

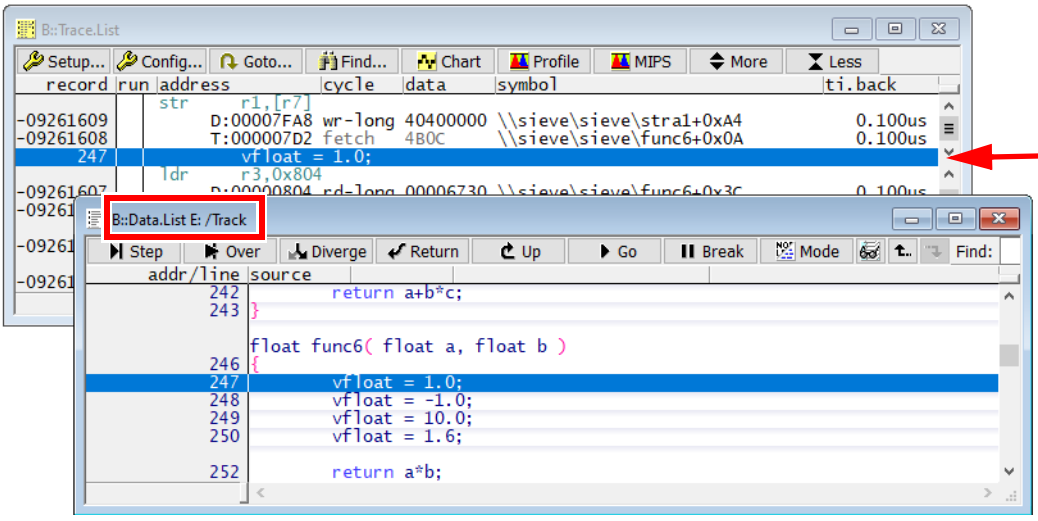
The displayed information can be reduced using the **Less** button. By pressing **Less** three times, only the high-level source code is displayed. This can be reverted using the **More** button.



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



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



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

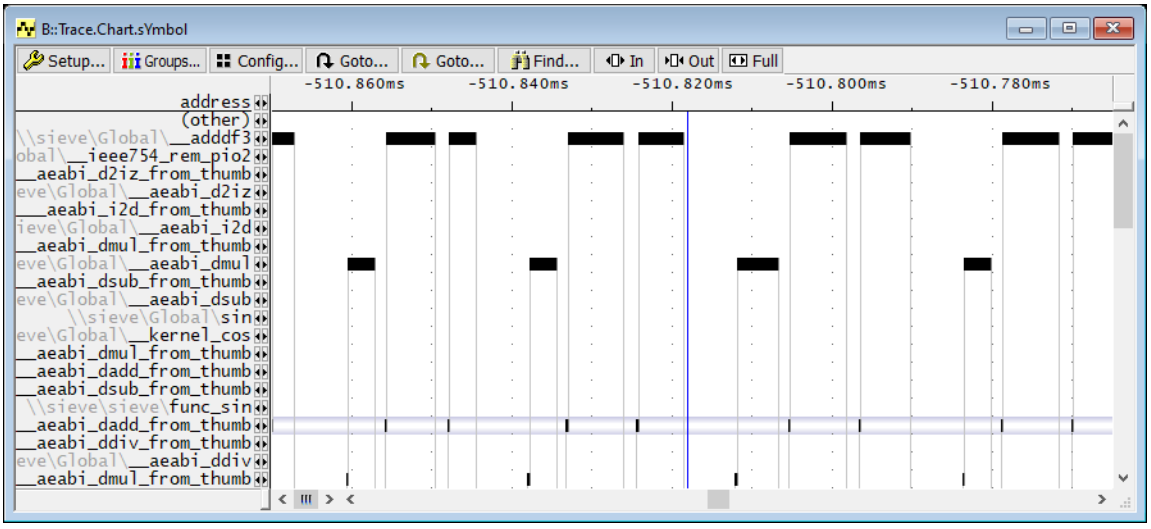
Displaying Function Run-Times

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

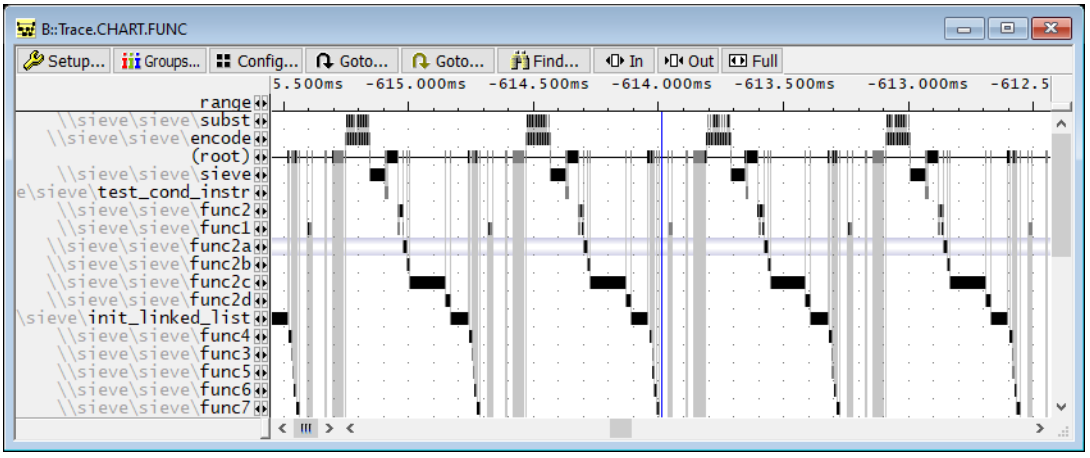
support.lauterbach.com/kb/articles/trace-based-profiling

Graphical Charts

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

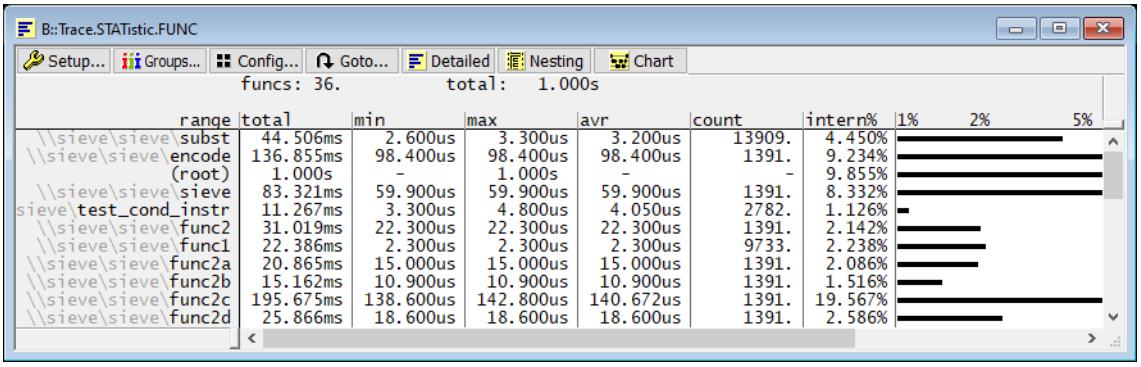


The corresponding nesting analysis can be displayed using the menu **Perf > Function Runtime > Show as Timing**.

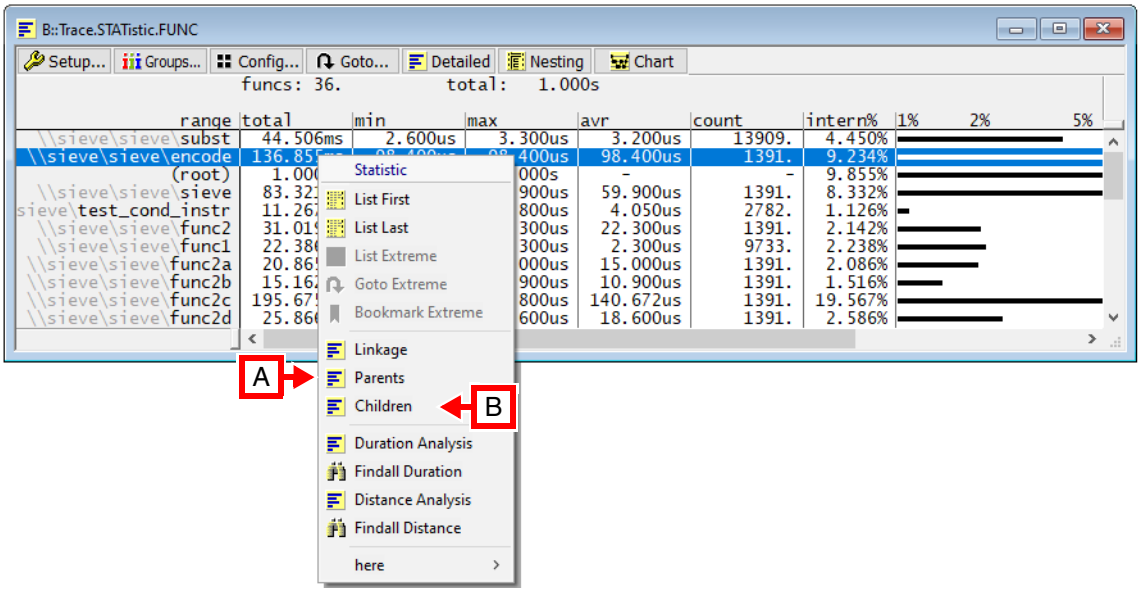


The **In** and **Out** buttons can be used to zoom in/out. Alternatively, you can select a position in the window and then use the mouse wheel to zoom in/out.

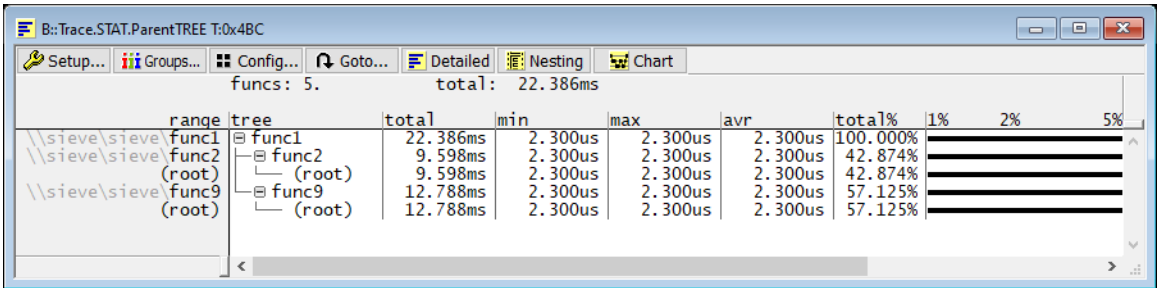
The menu entry **Perf > Function Runtime >Show Numerical** displays numerical statistics for each function with various information as total run-time, minimum, maximum and average run-times, ratio, and number of function calls.



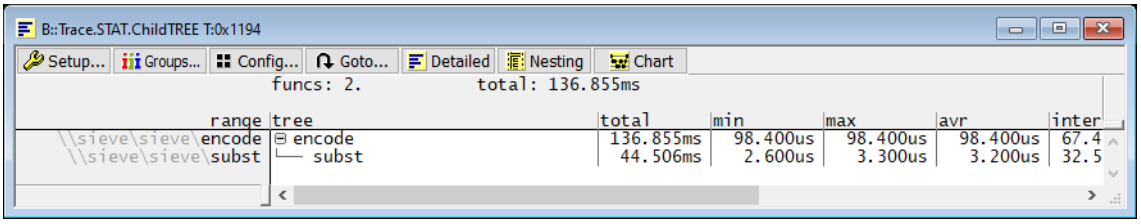
Further display options can be selected by doing a right mouse click on a specific function.



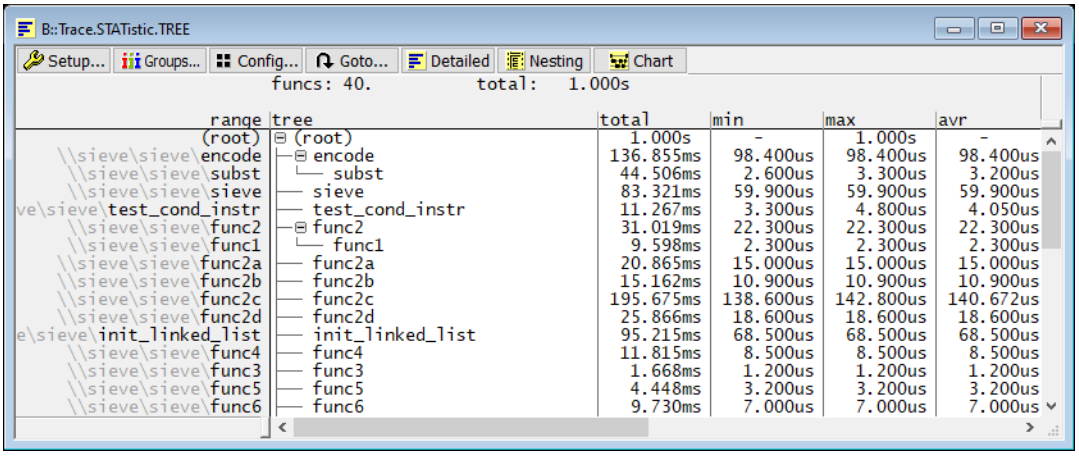
Parents [A] displays for example a caller tree for the selected function. By doing a right mouse click on func1 and selecting **Parents**, we see the run-times of the functions func2 and func9, which have called func1 in the trace recording.



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

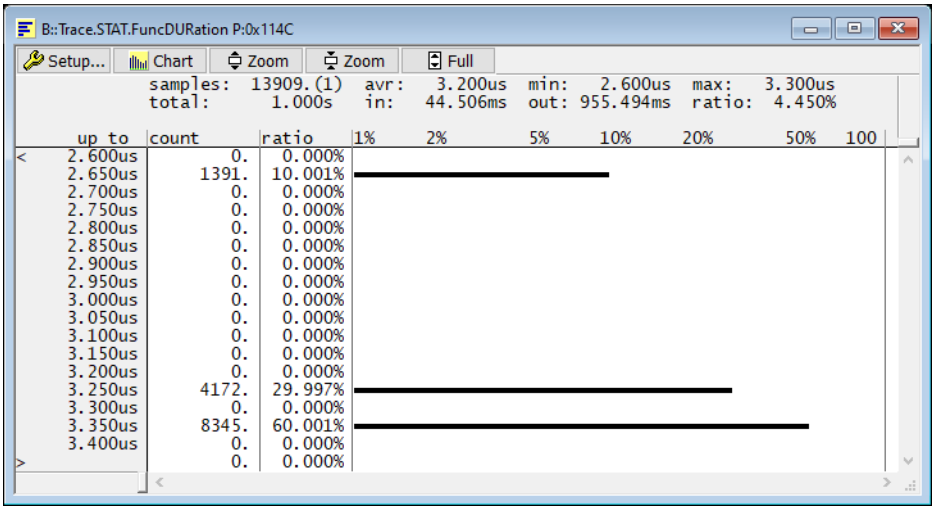


A function call tree view of all function recorded in the trace can be displayed using the menu entries **Perf > Function Runtime > Show as Tree** or **Perf > Function Runtime > Show Detailed Tree**.



Duration Analysis

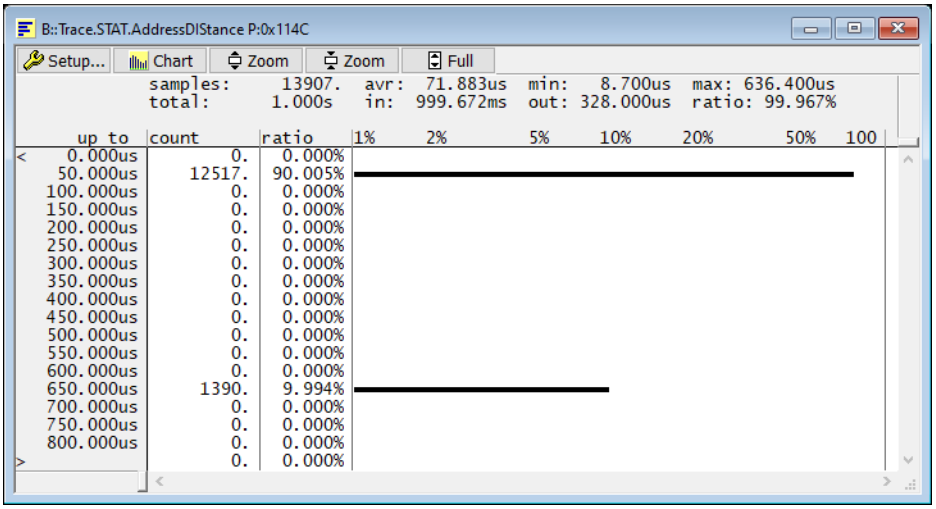
By doing a right mouse click on a function in the numerical statistics window (**Trace.STATistic.Func**) then selecting **Duration Analysis**, you get an analysis of the function run-times between function entry and exit including the time spent in called subroutines, e.g. here for the function subst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



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

Distance Analysis

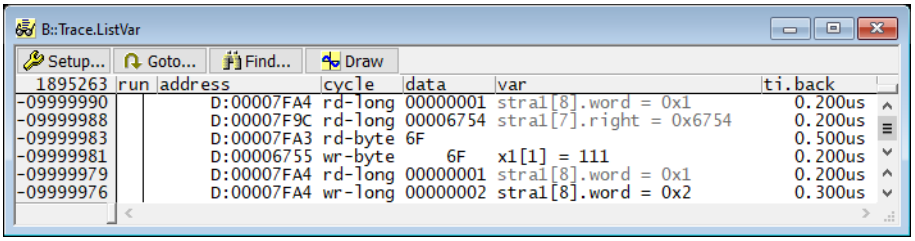
By doing a right mouse click on a function in the numerical statistics window (**Trace.STATistic.Func**) then selecting **Distance Analysis**, you can get run-times between two consecutive calls of the selected function, e.g. here for the function subst (P:0x114C corresponds to the start address of the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



Variable Display

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

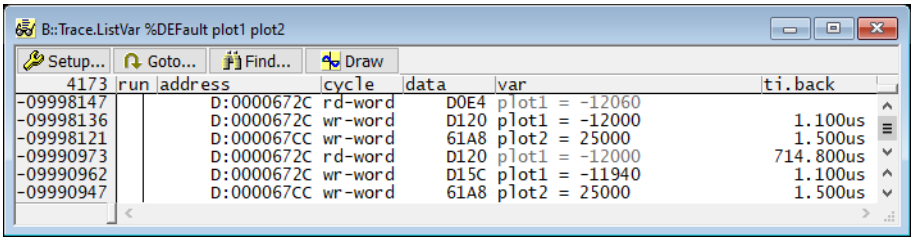
```
Trace.ListVar
```



You can optionally add one or multiple variables as parameters.

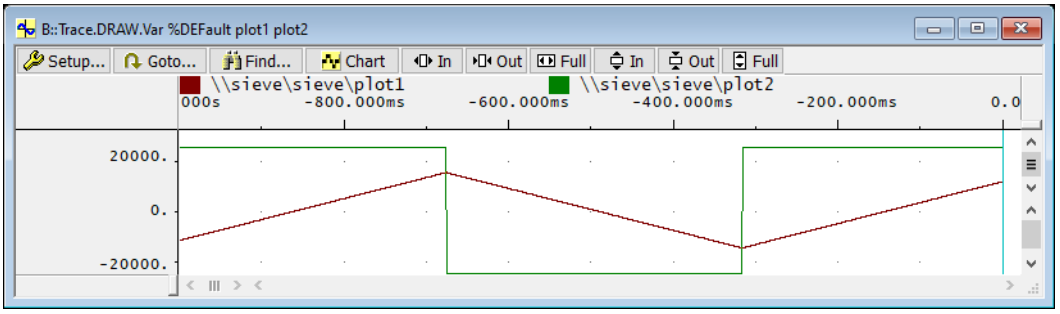
Example: display all accesses to the variables plot1 and plot2

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



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

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



Please refer for more information about the **Trace.DRAW** command to **"Application Note for Trace.DRAW"** (app_trace_draw.pdf).

Track Option

The **/Track** options allows to track windows that display the trace results. You just need to add the **/Track** option after the command that opens a trace window, e.g.

```
Trace.List /Track
```

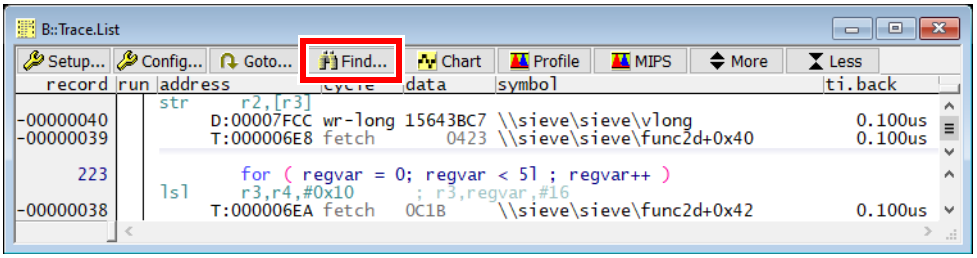
The cursor will then follow the movement in other trace windows, e.g. **Trace.Chart.Func**. Default is time tracking. If no time information is available, tracking to record number is performed.

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

```
Trace.Chart.Func /ZoomTrack
```

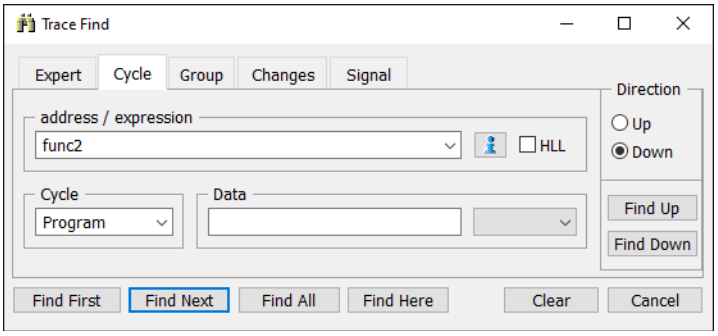
Searching Trace Results

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



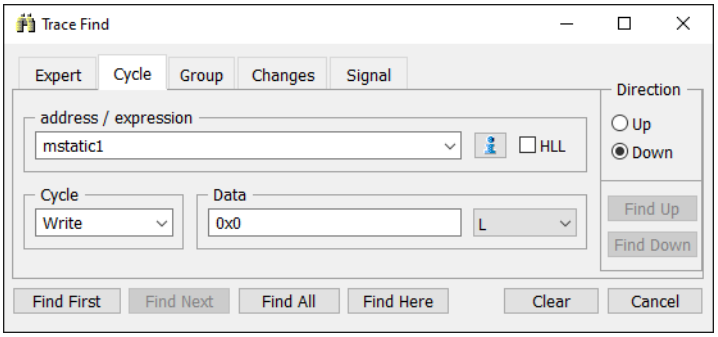
Example 1: find the first call of function func2

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



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

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



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

Trace Save and Load

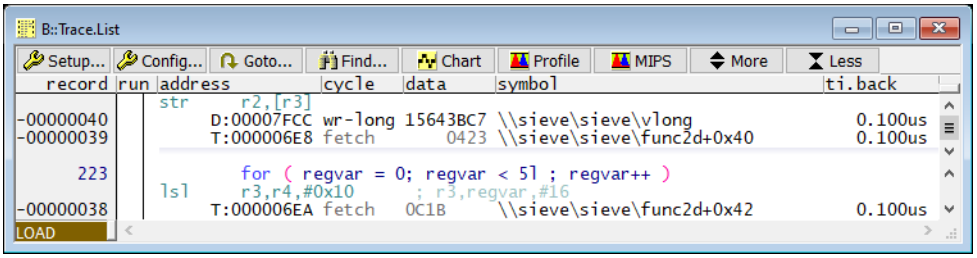
The recorded trace can be stored in a file using the command **Trace.SAVE**, e.g.

```
Trace.SAVE file.ad
```

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

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

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



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