LAUTERBACH
DEVELOPMENT TOOLS

MicroTrace for Cortex-M User's Guide

Release 02.2024

MANUAL

MicroTrace for Cortex-M User's Guide

MicroTrace for Cortex-M User's Guide

Version 04-Mar-2024

TRACE32 Products for Cortex-M

Lauterbach offers different tool configurations for debugging and tracing of Cortex-M cores. This chapter presents the individual configurations and their main applications briefly.

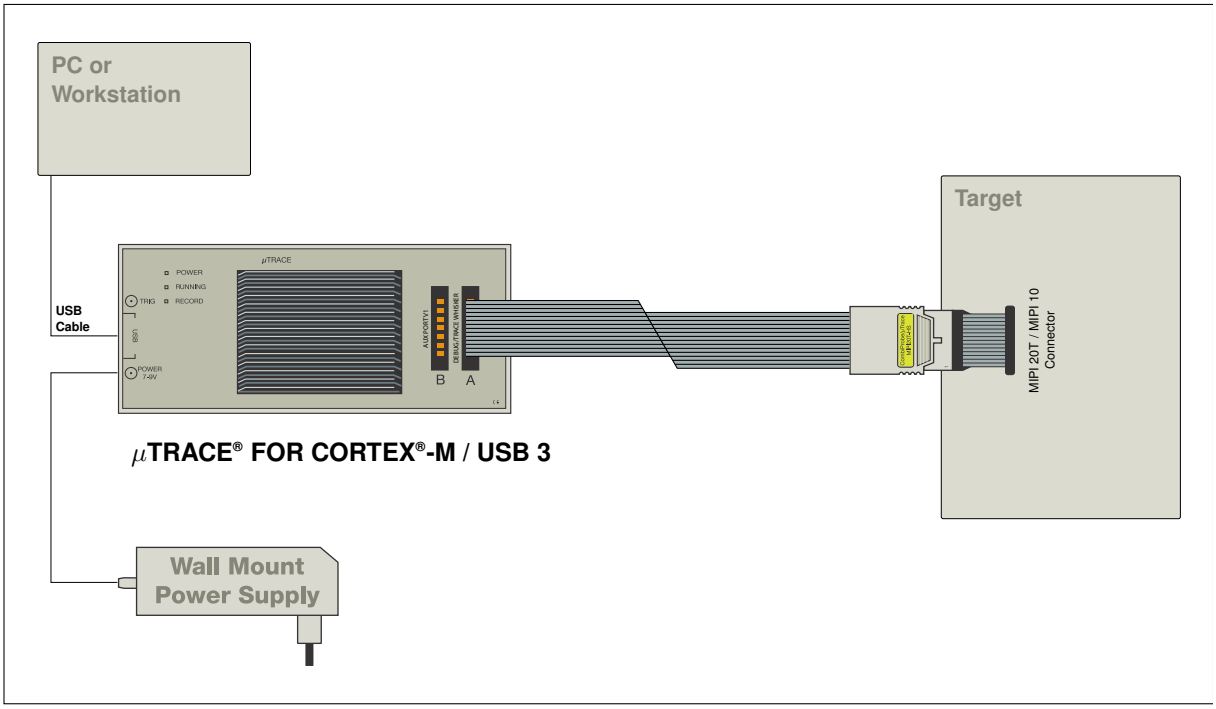
The following configurations are provided:

- µTrace (with MIPI20T-HS Whisker)
- PowerDebug and CombiProbe (with MIPI20T-HS Whisker)
- PowerDebug and CombiProbe (with CombiProbe MIPI34 Whisker)
- PowerDebug and Debug Cable
- PowerDebug and PowerTrace (X-License)

The following older combination is no longer recommended, but still supported:

- µTrace (with CombiProbe MIPI34 Whisker)

µTrace (MicroTrace) (with MIPI20T-HS Whisker)



You have chosen the all-in-one debug and off-chip trace solution developed by Lauterbach especially for Cortex-M processors.

The combination of µTrace (MicroTrace) and MIPI20T-HS whisker supports:

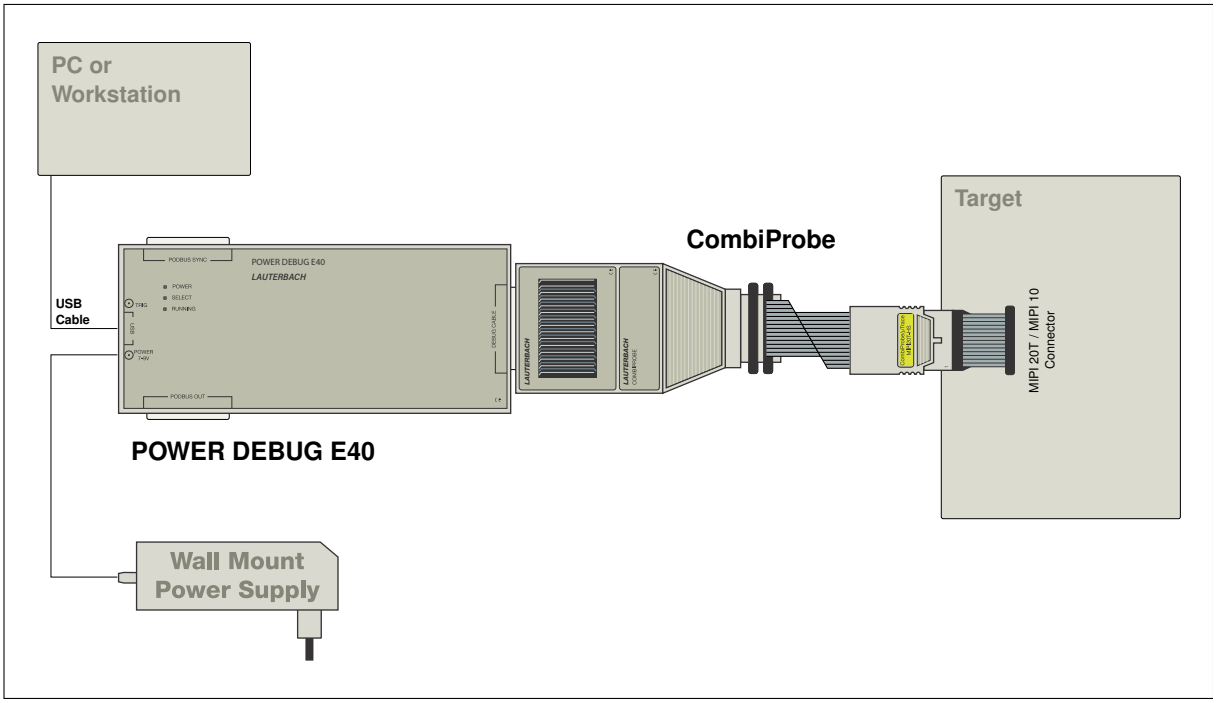
- Debugging via JTAG (IEEE 1149.1), SWD (Serial Wire Debug) or cJTAG (IEEE 1149.7) at clock rates up to 100 MHz
- Debug connectors MIPI20T and MIPI10 (without adapter), Arm-20 (with included adapter)
- Parallel trace using ETM/TPIU continuous mode with up to 4 data pins and bit rates of up to 400 Mbit/s per pin
- SWV (Serial Wire Viewer) / SWO (Serial Wire Output) trace at port rates up to 200 Mbit/s
- Automatic configuration and advanced diagnostics of electrical parameters of the used trace port

This combination requires TRACE32 R.2018.09 or newer.

Please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf) for all Cortex-M specific debug features.

This manual describes the basic setups and all Cortex-M specific trace features.

PowerDebug and CombiProbe (with MIPI20T-HS Whisker)



You have chosen a debug and off-chip trace solution for your processor which is tailor-made for the Cortex-M, but provides you with greater flexibility than the all-in-one debug and trace solution µTrace (MicroTrace). The combination of CombiProbe and MIPI20T-HS whisker supports:

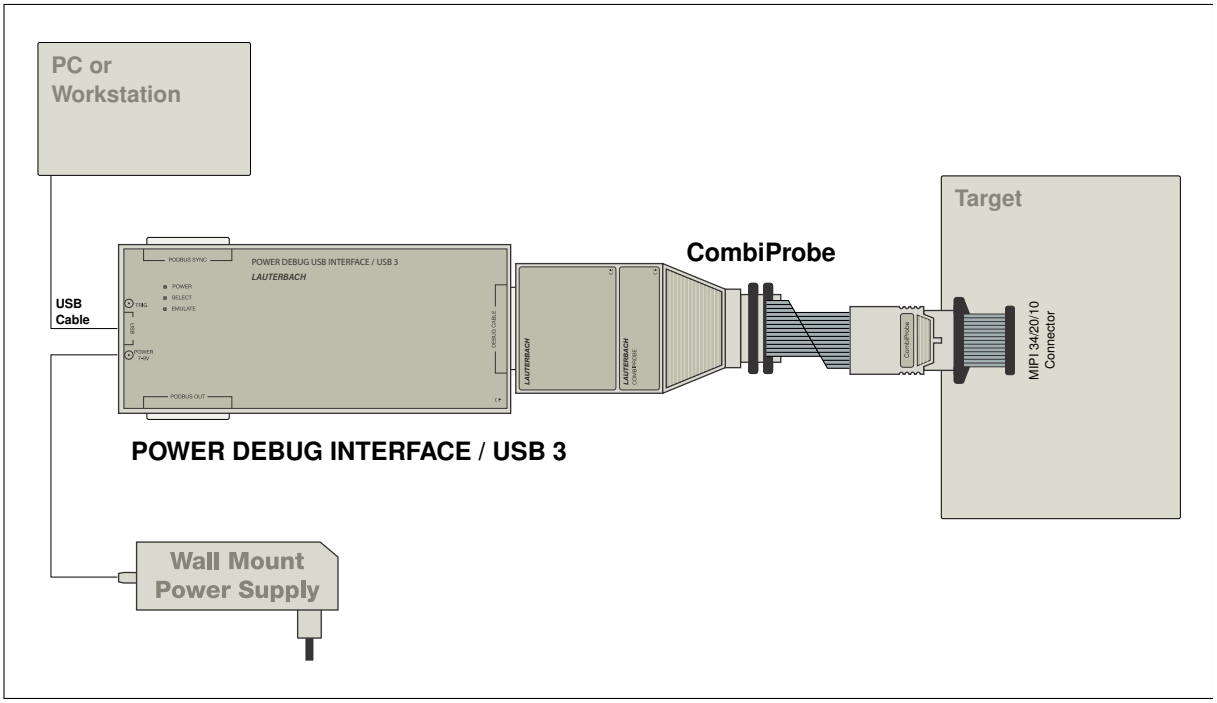
- Debugging via JTAG (IEEE 1149.1), SWD (Serial Wire Debug) or cJTAG (IEEE 1149.7) at clock rates up to 100 MHz
- Debug connectors MIPI20T and MIPI10 (without adapter), Arm-20 (with included adapter)
- Parallel trace using ETM/ITM in TPIU continuous mode with up to 4 data pins and bit rates of up to 400 Mbit/s per pin
- SWV (Serial Wire Viewer) / SWO (Serial Wire Output) trace at port rates up to 200 Mbit/s
- Automatic configuration and advanced diagnostics of electrical parameters of the used trace port
- Optional logic analyzer extension TRACE32 Mixed-Signal Probe.
- Debugging of CPU types other than Cortex-M (e. g. Cortex-A/R)
- Debugging two chips with two separate debug connectors (using a second whisker cable)

This combination requires TRACE32 R.2018.09 or newer.

For all Cortex-M specific debug features, please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf).

For all Cortex-M specific trace features, please refer to **"CombiProbe for Cortex-M User's Guide"** (combiprobe_cortexm.pdf).

PowerDebug and CombiProbe (with CombiProbe MIPI34 Whisker)



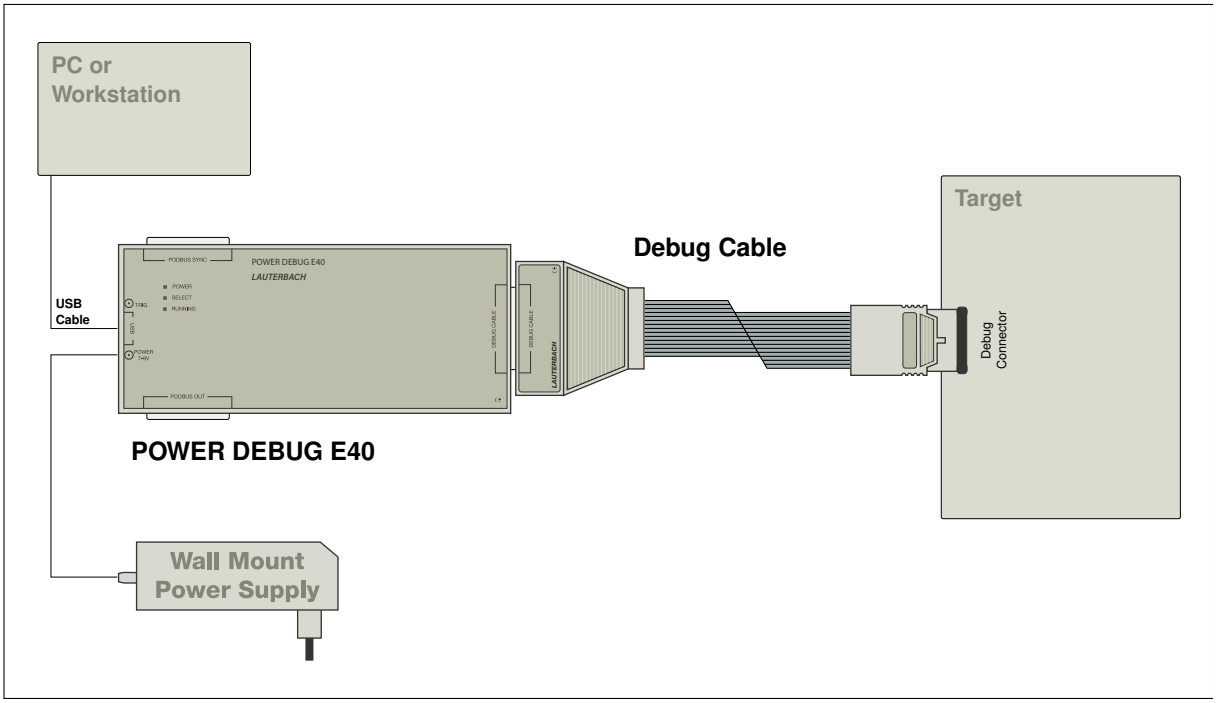
This solution is outdated.

You have chosen a debug and off-chip trace solution for your processor which is tailor-made for the Cortex-M, but provides you with greater flexibility than the all-in-one debug and trace solution µTrace (MicroTrace). The combination of CombiProbe and MIPI34 whisker supports:

- Debugging via JTAG (IEEE 1149.1), SWD (Serial Wire Debug) or cJTAG (IEEE 1149.7) at clock rates up to 100 MHz
- Debug connectors MIPI34, MIPI20D, MIPI20T and MIPI10 (without adapter), Arm-20 (with included adapter)
- Parallel trace using ETM/ITM/STM either in TPIU continuous mode or without a TPIU with up to 4 data pins and bit rates of up to 200 Mbit/s per pin
- SWV (Serial Wire Viewer) / SWO (Serial Wire Output) trace at port rates up to 100 Mbit/s
- Optional logic analyzer extension (PowerProbe or PowerIntegrator)
- Debugging of CPU types other than Cortex-M (e. g. Cortex-A/R)
- Debugging two chips with two separate debug connectors (using a second whisker cable)

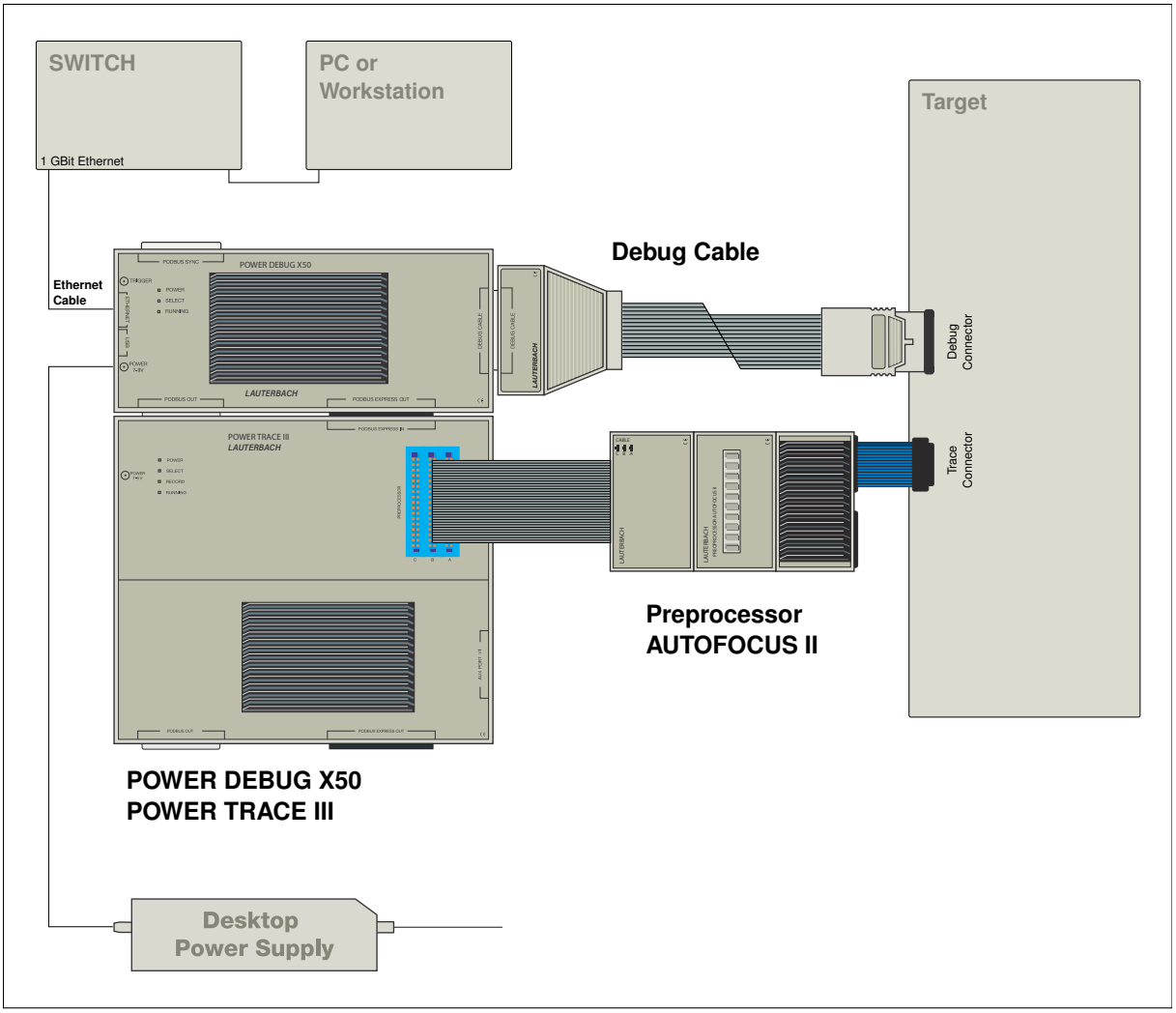
For all Cortex-M specific debug features, please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf).

For all Cortex-M specific trace features, please refer to **"CombiProbe for Cortex-M User's Guide"** (combiprobe_cortexm.pdf).



You have chosen a pure debug solution because your processor has no off-chip trace option or you have no interest in off-chip tracing.

For all Cortex-M specific debug features, please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf).



You have the TRACE32 high-end debug and off-chip trace solution for your processor and it is likely that your Cortex-M is part of a complex SoC.

For all Cortex-M specific debug features, please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf).

For all Cortex-M specific trace features, please refer to **"Training Cortex-M Tracing"** (training_cortexm_etm.pdf).

Keywords

These keywords are used in the following:

Keywords	Description
Serial Wire	ARM's Serial Wire Debug (SWD) port and ARM's Serial Wire Viewer (SWO) output.
Trace	• Support for ARM's Instrumentation Trace Macrocell (ITM) and Embedded Trace Macrocell (ETM), both exported by a Trace Port Interface Unit (TPIU) in 4/2/1-bit continuous mode (with enabled Formatter). • Support for ARM's Instrumentation Trace Macrocell (ITM), exported via ARMs SerialWireOutput in UART mode.

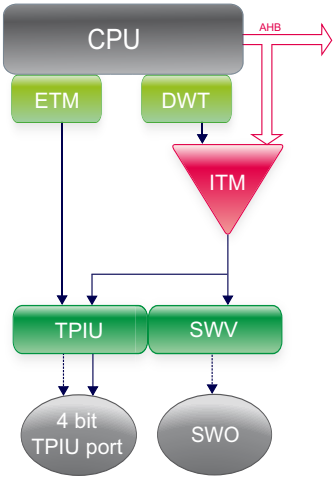
CoreSight Components

Cortex-M processors may include the CoreSight components **Instrumentation Trace Macrocell** (ITM) and **Embedded Trace Macrocell** (ETM). Components in detail:

- 4-bit ETMv3 in Continuous mode for Cortex-M3/M4/M7.
- 4-bit ETMv4 in Continuous mode for Cortex-M7.
- Onchip trace buffer (ETB).
- ITM over TPIU for Cortex-M3/M4/M7.
- ITM over Serial Wire Output for Cortex-M3/M4/M7.

Overview of Cortex-M CoreSight Components

This overview focuses on the CoreSight components which implement trace support for the Cortex-M3, the Cortex-M4, and Cortex-M7. These components are shown in the figure below.



The DWT (Data Watchpoint and Trace) unit and the SWV mode (Serial Wire Viewer) are features, which are implemented in the Cortex-M3, the Cortex-M4, and Cortex-M7.

As you can see from the above figure, the ITM is a regular memory-mapped peripheral for the CPU, accessible via the AHB (Advanced High-performance Bus).

All CoreSight components are now explained in more detail.

Embedded Trace Macrocell (ETM) Overview

The Cortex-M3/M4/M7 can be connected to an *Embedded Trace Macrocell* (ETM). The ETM is an *optional* component.

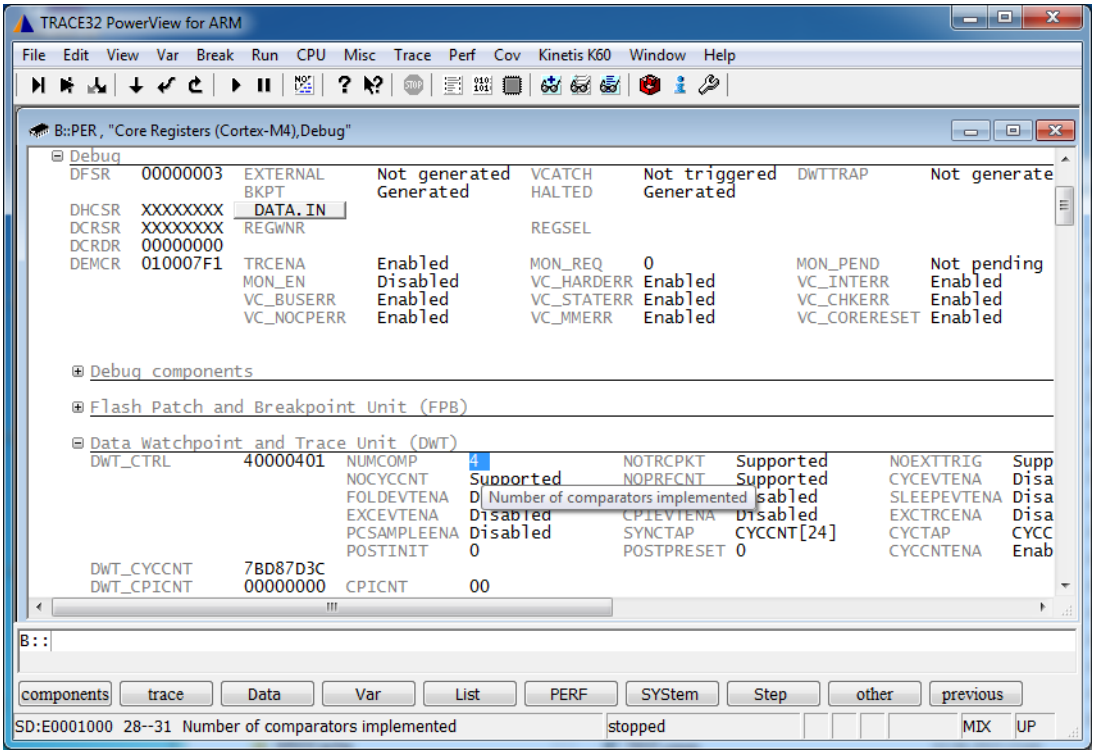
The ETM is a very simple ETM, which only can generate information about the instruction execution sequence. Specifically the Cortex-M ETM:

- Does *not* support any kind of data tracing.
- Does *not* contain comparators to filter information of interest out of the instruction execution sequence.
- Does *not* support cycle accurate tracing.
- Does *not* support to trace a ContextID.

Data Watchpoint and Trace (DWT) Unit Overview

To add more trace features, the Cortex-M3/M4/M7 can contain an optional DWT unit. The DWT unit is able to monitor data accesses and the program counter of the CPU. The DWT offers the following features:

- It contains comparators, which can trigger several actions if a match occurs.



The NUMCOMP field in the control register for the DWT shows the number of available comparators for your core.

- It can count specific types of CPU cycles. The DWT can emit events indicating that a counter wrapped (which happens every 256 counted cycles). By analyzing these events the debugger can present statistics about the distribution of the different types of CPU cycles.
- It can emit information about data accesses if the access is matched by a comparator.
- It can halt the CPU or trigger the ETM if an access is matched by a comparator.
- It can emit information about the current program counter value at regular intervals.
- It can emit information about Interrupt Service Routine entries and exits.

Instrumentation Trace Macrocell (ITM) Overview

The *Instrumentation Trace Macrocell* (ITM) for the Cortex-M3/M4/M7 has two main functions:

- It is used by the DWT to emit data to an external debug and trace tool.
- It can be used by software running on the CPU to emit data to an external debug and trace tool.

The ITM appears to software running on the CPU as a memory mapped peripheral. By writing to the corresponding memory range, the software can send data to an external debug and trace tool.

Trace Port Interface Unit (TPIU) Overview

The Cortex-M *Trace Port Interface Unit* (TPIU) is responsible for exporting data from the ETM and ITM through pins of the chip. The TPIU has two different modes of operation:

- **Trace Port mode**: In this mode, the TPIU uses one clock and up to 4 data pins to synchronously export data. In this mode the TPIU always enables its internal Formatter. This means that data from the ETM and ITM will be encapsulated into the Formatter protocol. The Formatter merges ETM and ITM data into a single stream of bytes.
- **Serial Wire Viewer mode**: In this mode, the TPIU uses a single signal (Serial Wire Output, SWO) to asynchronously export data. In this mode the TPIU solely outputs ITM data; it is not possible to output ETM data. The internal Formatter is disabled; so the ITM data is not encapsulated into the Formatter protocol.

Serial Wire Viewer in general refers to outputting ITM data via the asynchronous Serial Wire Output signal.

Embedded Trace Buffer

Instead of exporting data from the ETM and ITM through pins off chip, data can also be routed to the Embedded Trace Buffer (ETB). The ETB content is read by TRACE32 through the debug connection.

Connectors

The export of the trace data depends on the debug and trace connector.

For parallel trace, we recommend the 20-pin debug and trace variant specified by the MIPI alliance and ARM. Lauterbach refers to this connector as **MIPI20T**:

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS|TMSC|SWDIO
GND	3	4	TCK|TCKC|SWCLK
GND	5	6	TDO|-|SWO
GND (KEY)	-	8	TDI
GND	9	10	RESET-
GND	11	12	TRC CLK
GND	13	14	TRC DATA[0]
GND	15	16	TRC DATA[1]
GND	17	18	TRC DATA[2]
GND	19	20	TRC DATA[3]

If parallel trace is not required, you can also use the smaller **MIPI10** variant, which is a subset of MIPI20T:

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS|TMSC|SWDIO
GND	3	4	TCK|TCKC|SWCLK
GND	5	6	TDO|- |SWO
GND (KEY)	-	8	TDI
GND	9	10	RESET-

Both can be used for SWV trace if supported by the target. In this case, the debug protocol is SWD and the TDO line is used for trace.

Setting up Parallel Trace

To set up parallel trace, several steps are required.

Configuring the Correct Port Type

Since the trace data pins are at different locations for different connectors, you should inform TRACE32 about the used connector type. For details on the connectors refer to **"Connectors"**, page 14.

```
; the default connector is MIPI20T (with trace pins)
SYStem.CONFIG CONNECTOR MIPI20T

; for all other connectors, specify MIPI34
SYStem.CONFIG CONNECTOR MIPI34
```

Changing the connector type is only possible in the **System.Mode Down** state, before connecting to the target.

Connecting to the Target and Configure Trace-related Components

Connect to the target as usual. Please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf) for details.

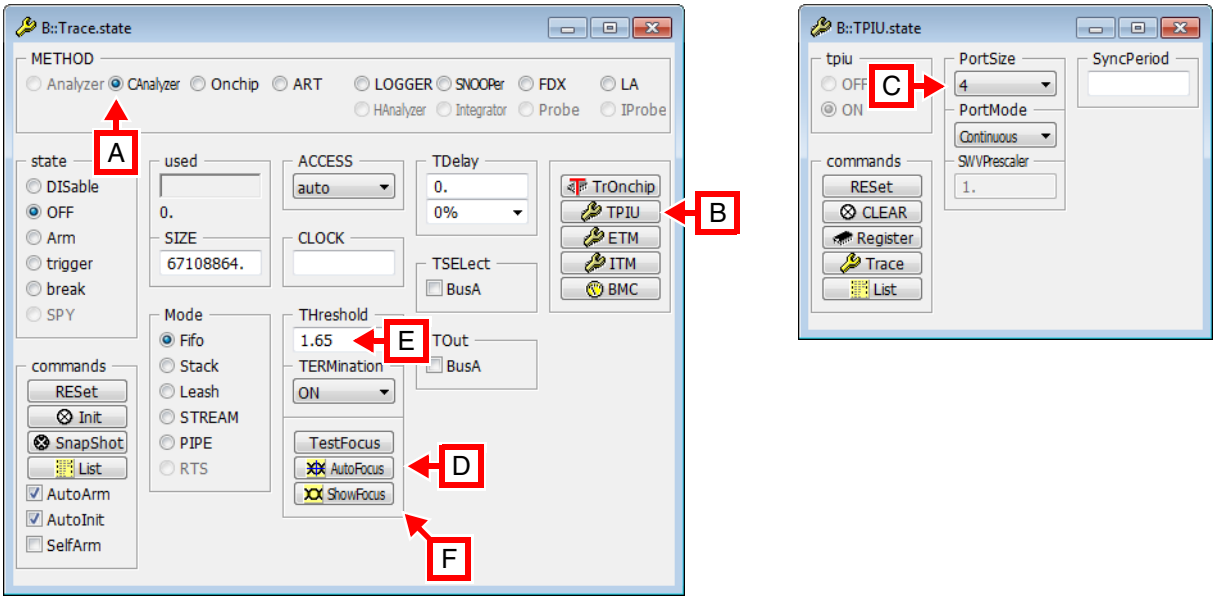
Depending on the target chip and target board, some additional configuration may be required. This configuration can be done either by the software running on the target or through the debugger.

- Ensure that the trace logic on the chip, especially the TPIU, is clocked using an appropriate clock source. The clock to the TPIU determines the bit rate of the data pins. Because the trace uses double data rate (DDR) signalling, the frequency on the TRACECLK pin will be one half of the TPIU input clock.
- Configure on-chip pin multiplexing, so that the trace data and clock lines are routed to the correct pins.
- If the chip I/Os have configurable drive strength and slew rate, configure these appropriately. If in doubt, use the highest available values.
- If the trace data and clock lines are shared with other peripherals on the target, make sure this periphery does not drive to the lines. If possible, remove jumpers or solder bridges to completely disconnect the periphery from the trace lines.

Please check the documentation of your chip and/or board for further details.

The trace features of the µTrace (MicroTrace) are controlled with the **CAnalyzer** command group and the **CAnalyzer.state** window. Since **Trace** is the default alias for CAnalyzer if the µTrace (MicroTrace) is used, you can also use the command group **Trace** to control the trace features:

Trace	Command group for the trace features of the µTrace (MicroTrace)



To configure the trace settings:

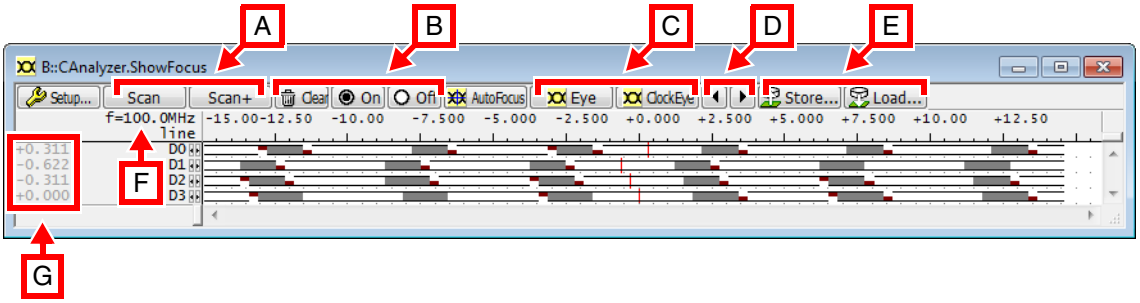
1. Make sure **CAnalyzer** is selected as METHOD, see [**A**].
2. Open the **TPIU.state** window, see [**B**].
3. Select the number of trace data lines you have connected to the debugger (1, 2 or 4), see [**C**].
4. Click **AutoFocus**, see [**D**].

   This configures the target to generate a test pattern that is captured by your µTrace (MicroTrace) to automatically determine the best threshold voltage and data delays.

   The operation takes approximately three seconds. If successful, the message “Analyzer data capture o.k.” will be printed to the message line at the bottom of the TRACE32 window.
5. Review or change the threshold voltage for the trace data and clock lines in the **THreshold** input box, see [**E**].
6. To view or change the timing parameters, click the **ShowFocus** button, see [**F**].

If the AutoFocus operation was successful, you are ready to use the off-chip trace. In this case, you can skip the rest of this section.

To diagnose any problems that were detected by the **CAnalyzer.AutoFocus** command or to check the signal quality, open the **CAnalyzer.ShowFocus** window:



The data area of the **CAnalyzer.ShowFocus** windows displays each channel in a separate row. The horizontal axis is the time offset from the clock edge, measured in nanoseconds. The gray areas indicate the offsets where a change of the data value was detected. Red bars mean that the data value only changed on falling or rising edges, not both.

A Click **Scan** to perform a new measurement. Existing results are discarded.
Scan+ performs the same operation, but combines the new measurement with the existing data.

B Use the **Clear**/**On**/**Off** buttons to manually clear the measured data or enable/disable the capture. This can be done even while trace data is being recorded, providing a way to detect rare glitches on the data lines.

C The **Eye** and **ClockEye** buttons open the data and clock eye windows described below.

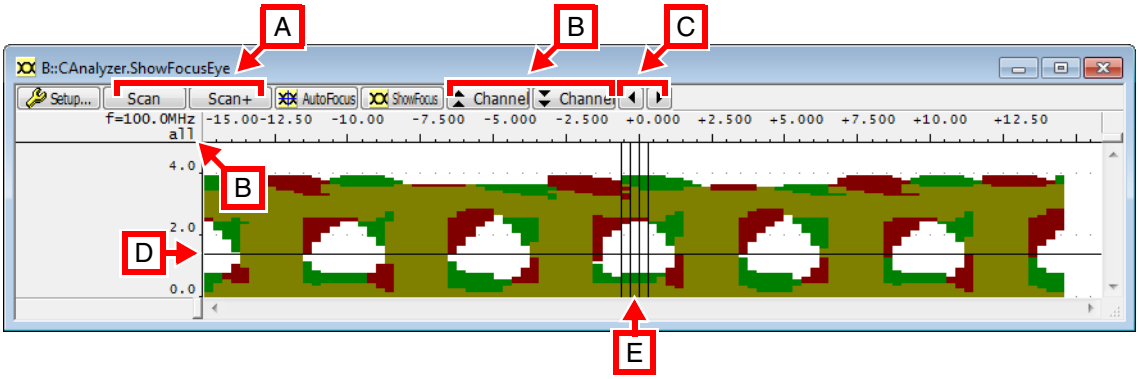
D Use the arrow buttons to move the sample points for all channels at the same time.

E Use the **Store...** button to create a PRACTICE script (*.cmm) with all current electrical settings. This script can be executed using the **DO** command or the **Load...** button. The contents of the script may also be copied to your start-up script to replace the AutoFocus command. This can speed up the start-up process.

F This shows the detected TRACECLK frequency. Note that the bit rate of the data lines is twice this frequency.

G Displays the current sample delays. The delays can be changed by double-clicking in the data area of the window or using the small buttons next to the channel names. Positive delays mean that the data line is sampled after the corresponding clock edge.

The **CAnalyzer.ShowFocusEye** window provides even more detailed information:



As in the **CAnalyzer.ShowFocus** window, the horizontal axis is the time offset from an edge of the TRACECLK signal. The vertical axis is the voltage and always ranges from 0 to 5 V. In white areas, the data signal was always stable. In green, red and yellow areas, the signal changed in response to rising clock edges, falling clock edges, or both.

A The **Scan** and **Scan+** behave like their counterparts in the **CAnalyzer.ShowFocus** window. However, the operation is more thorough and takes a few seconds. During the measurement, the threshold voltage is changed temporarily, so a scan is not possible while trace data is being captured.

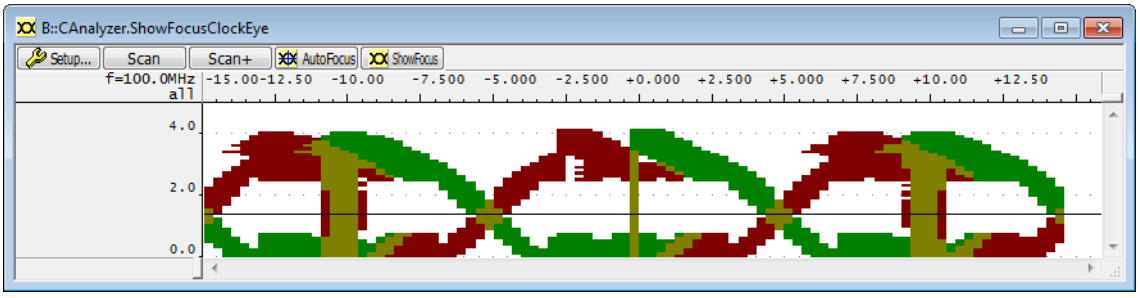
B Use the **Channel** buttons to switch between individual channels. By default, all channels are overlaid on each other. The current channel is indicated at the top left corner of the window.

C The arrow buttons change the sample times for all visible channels.

D The horizontal line indicates the current threshold voltage. This setting can be changed in the **Trace.state** window or with the **CAnalyzer.THreshold** command.

E The vertical lines show the sample point of each visible channel.

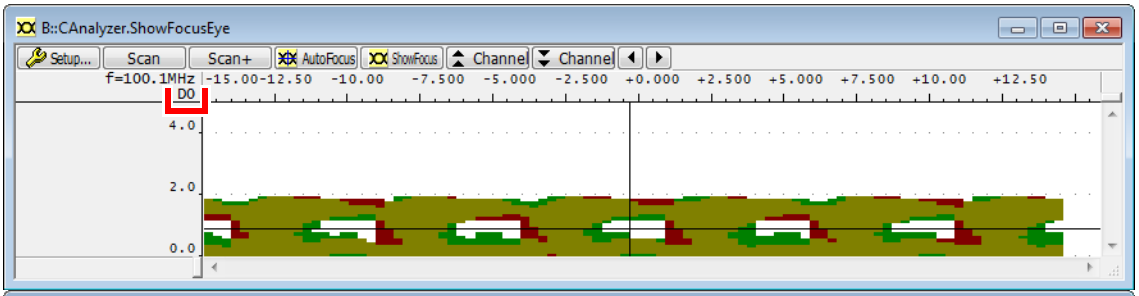
The **CAnalyzer.ShowFocusClockEye** window can be used to analyze the TRACECLK line:



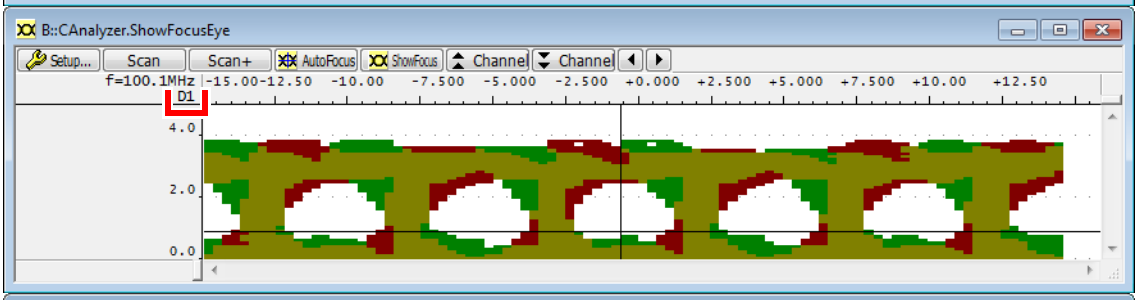
Note that the capture of this data is triggered by the clock line itself, so there should always be a vertical yellow line at offset 0. The information displayed in this window can be useful to check the duty cycle and jitter of the clock signal.

In this scenario, the **CAnalyzer.ShowFocusEye** view for the D0 line differs significantly from the other lines D1, D2, and D3:

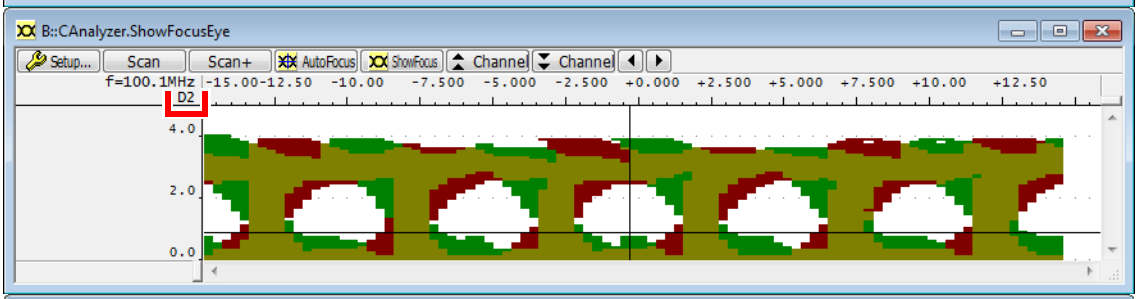
D0



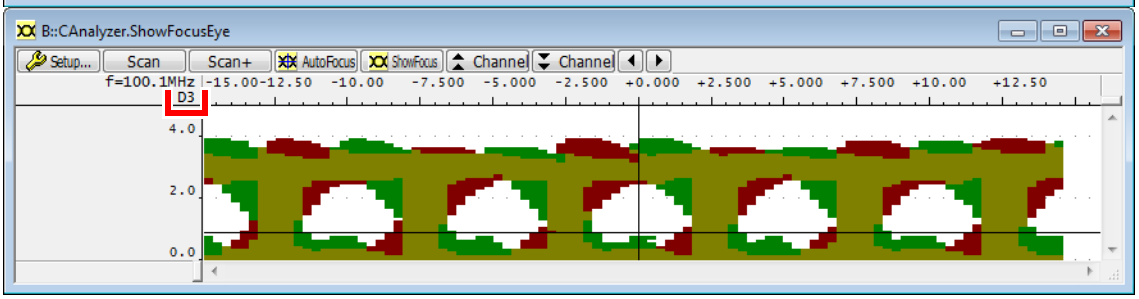
D1



D2

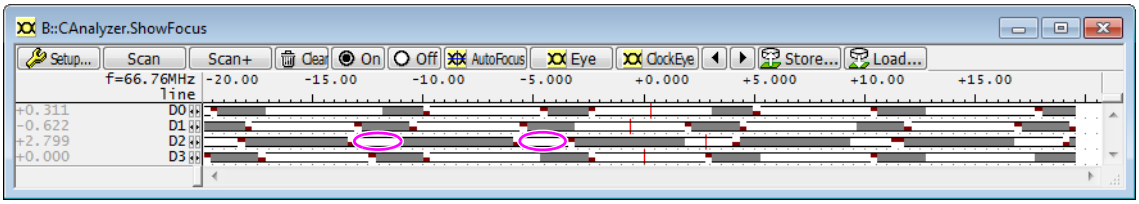


D3



In this case, the D0 line was shared with another IC on the target. Because this IC was not properly disabled, it was driving the D0 line to ground.

In this scenario, the **CAnalyzer.ShowFocus** window reveals an unusually large delay on the D2 line:

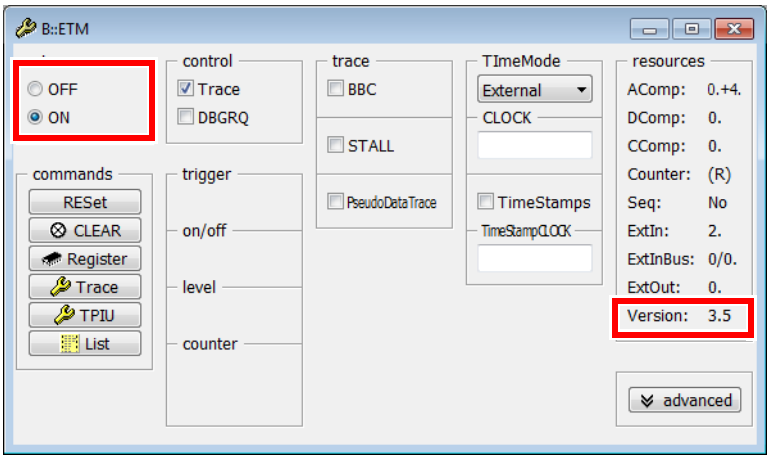


In this case, it was possible to get an error-free recording at a frequency of 66 MHz because the sample points were set appropriately. However, at higher frequencies, there was no longer a usable data eye for the D2 signal.

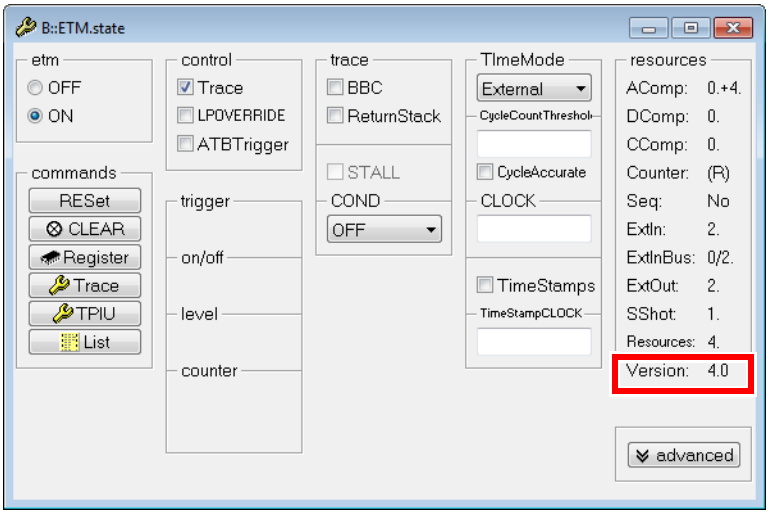
The delay was caused by a long connection on a custom adapter board.

The TRACE32 PowerView GUI controls the ETM via the **ETM** window respectively via the command group:

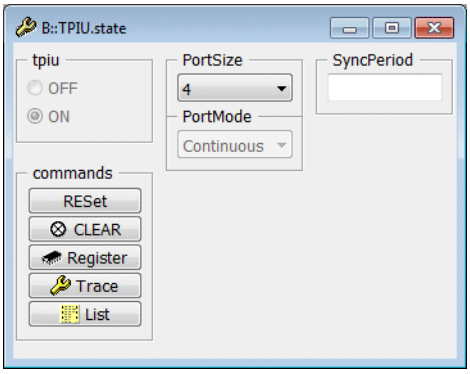
ETM	Command group to control the ETM



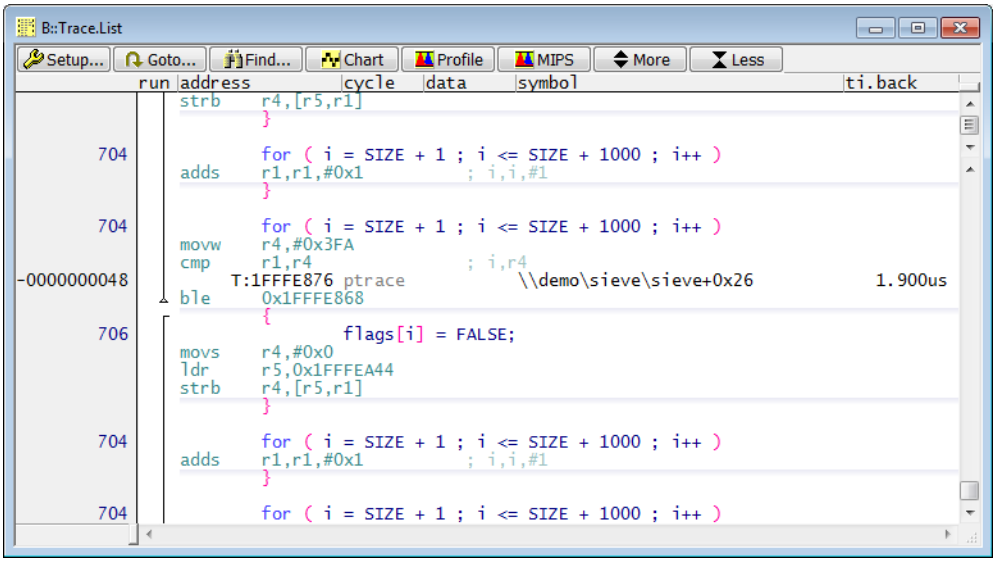
Most Cortex-M chips include an ETMv3, only the Cortex-M7 includes a ETMv4 with slightly different features.



The TPIU is controlled via the **TPIU** command group. The **TPIU.PortSize** command allows you to set up the number of used trace data pins.



The ETM is ON be default, so the µTrace (MicroTrace) can immediately record ETM data exported via the TPIU port. After recording the data, you can display the recorded instruction execution sequence with the **Trace.List** command and window.



NOTE: As long as the ETM is turned on, the **Trace.List** command will display ETM data.

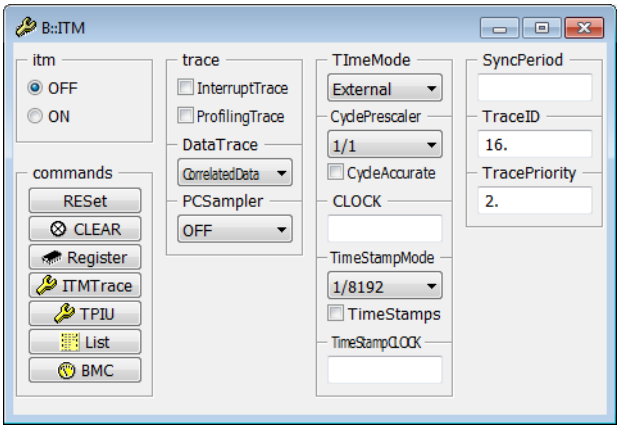
- If the ETM is turned off and you only record ITM data, the **Trace.List** command will display ITM data.
- If you record both ETM and ITM data, you can use the **ITMCAnalyzer.List** command to view the ITM data.

Of course the recorded ETM data can be used to run any kind of analysis supported by the TRACE32 PowerView GUI. This includes for example:

- A **chart display** of the executed functions
- A **run-time statistics** of the executed functions

A DWT unit is a CoreSight component and can be attached to the Cortex-M. The TRACE32 PowerView GUI controls the DWT via the **ITM** window and via the command group:

ITM	Command group to control the DTW



To display the recorded ITM data (which encapsulates DWT data), you can always use the command **ITMCAnalyzer.List**. If the ETM is turned off, then the **Trace.List** command will also display ITM data.

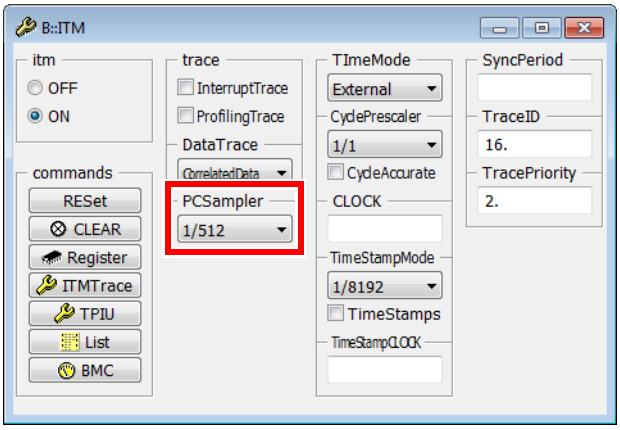
The comparators of the DWT are programmed by setting special on-chip breakpoints. The following DWT features are supported:

- PCSampler
- Interrupt Trace
- Tracing data accesses
- Cycle accurate trace

These features are described in the following sections.

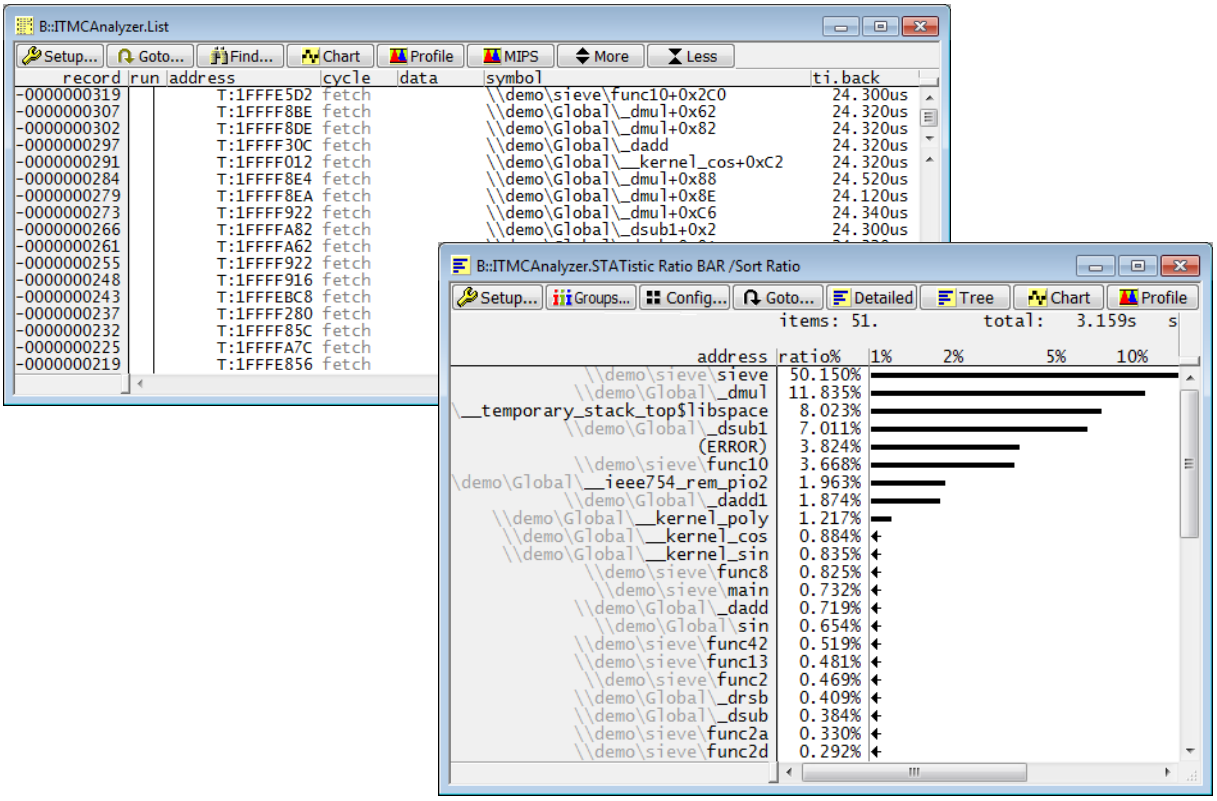
The DWT can output the current value of the program counter at regular intervals. The intervals are specified in clock cycles. The frequency with which the program counter is emitted via the ITM can be selected from the **PCSampler** drop-down list in the **ITM** window or entered at the command line:

ITM.PCSampler *<rate>*




```
ITM.PCSampler 1/512                    ; configure DTW to emit PC all
                                       ; 512 clock cycles
```

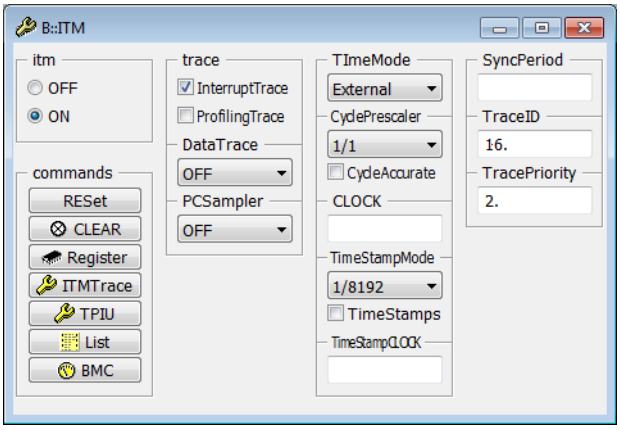
Sampling the PC can be used to get a statistical analysis of the distribution of run-time of the various functions by using the **ITMCAnalyzer.STATistic** command.

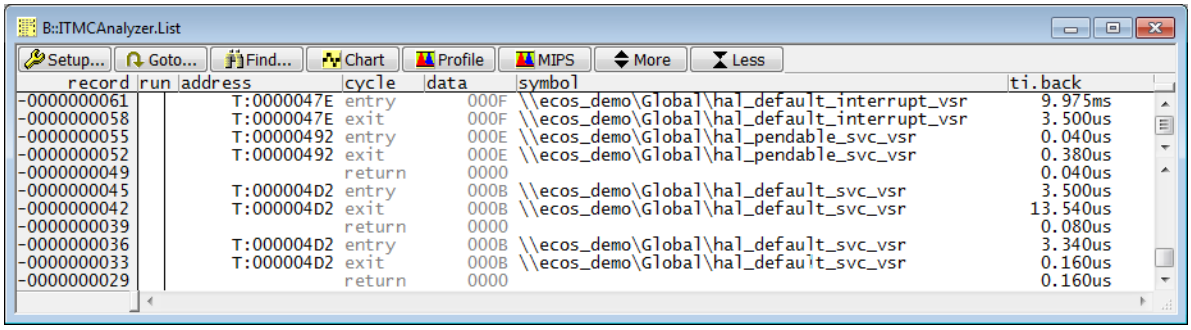



```
; List command to display recorded information
ITMCAnalyzer.List
; Statistic command, result sorted by ratio
ITMCAnalyzer.STATistic Ratio BAR /Sort Ratio
```

The DWT offers the possibility to emit information about interrupt service routine entries and exits. To use this feature of the DWT, select the **InterruptTrace** check box in the **ITM** window or type at the command line:

```
ITM.InterruptTrace ON
```



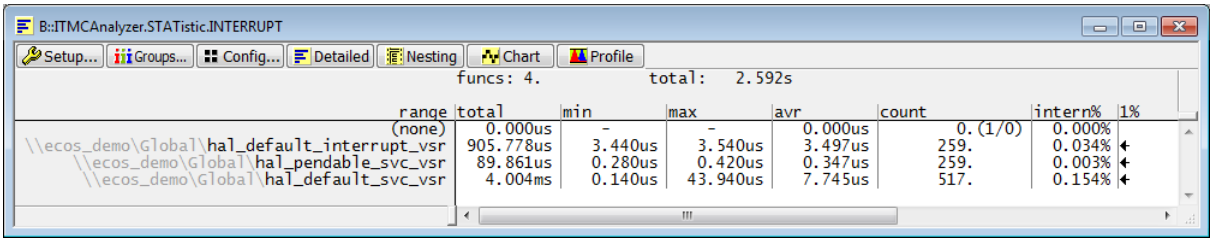


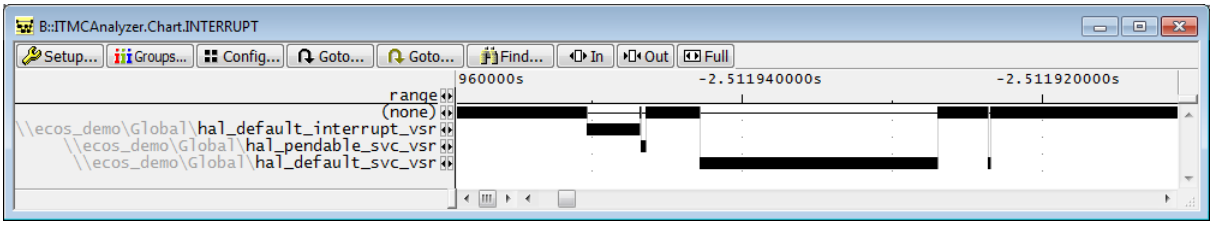
If your application is mainly interrupt driven, you can get a quite precise analysis of the run time of the different interrupt service routines by using this DWT feature.

cycle information	
entry	Interrupt entry
exit	Interrupt entry
return	Return to normal program execution

The following commands allow an interrupt analysis:

Trace.STATistic.INTERRUPT	Interrupt statistic
Trace.Chart.INTERRUPT	Interrupt time chart





Tracing Data Accesses

To trace data accesses to specific memory locations, you first have to configure the DWT to emit information about data accesses. This can be achieved via the **DataTrace** drop-down list in the **ITM** window or via the command line:

> **ITM.DataTrace** *<param>*

Tracing only Write Accesses

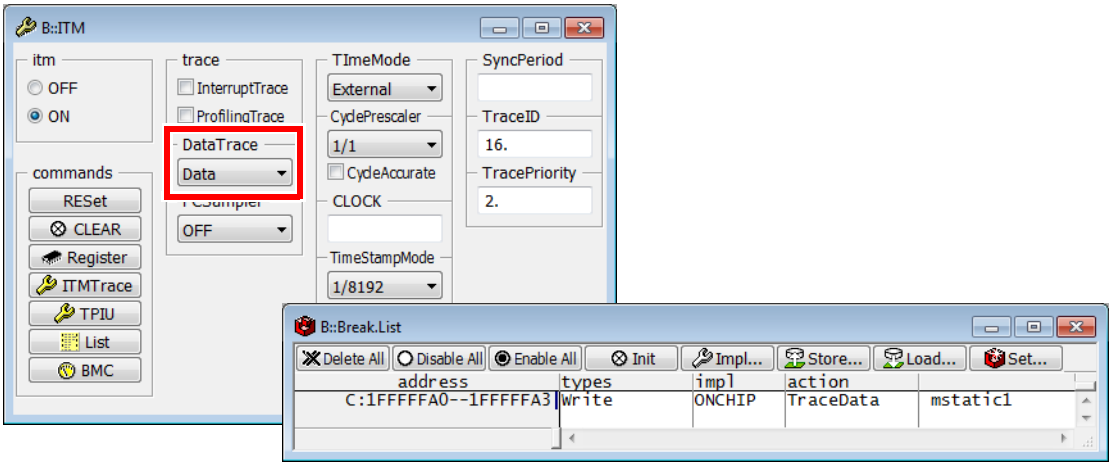
If you want to trace accesses to a single global variable, the **Data** setting is the most useful one. Select **Data** from the **Data Trace** drop-down list or type at the command line:

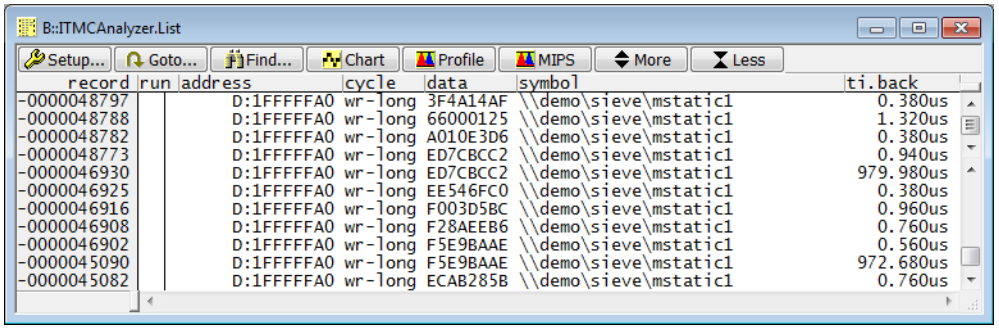
```
ITM.DataTrace Data
```

The second step is to program the DWT comparators. In the TRACE32 PowerView GUI, the mechanism to configure the DWT comparators is to define a `TraceData` breakpoint for the address or address range for which you want to record information about accesses.

```
Break.Set mstatic1 /Write /TraceData
```

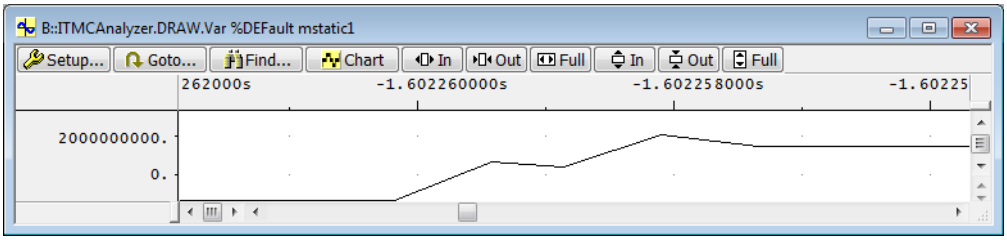
Please be aware the older Cortex-M3 (r1p1 and earlier) were not able to separate read from write accesses.





The following command allows a graphical display of the data value over the time:

```
ITMCAnalyzer.DRAW.Var %DEFault mstatic1
```



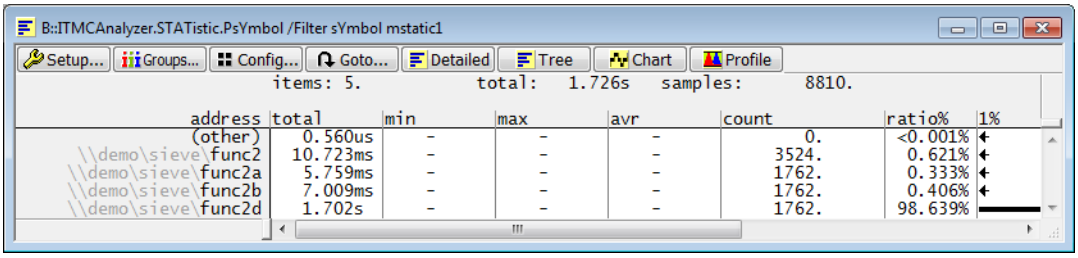
Tracing Data Accesses and the PC

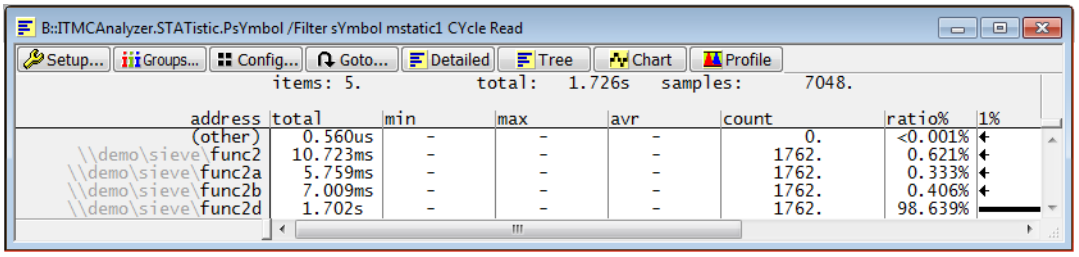
If you like to know which function accesses a variable do the following:

```
ITM.DataTrace DataPC

Break.Set mstatic1 /ReadWrite /TraceData
```

These analysis commands can be used:

```
; display an access statistic for the variable mstatic1
ITMCAnalyzer.STATistic.PsYmbol /Filter sYmbol mstatic1

; display a read access statistic for the variable mstatic1
ITMCAnalyzer.STATistic.PsYmbol /Filter sYmbol mstatic1 CYcle Read
```





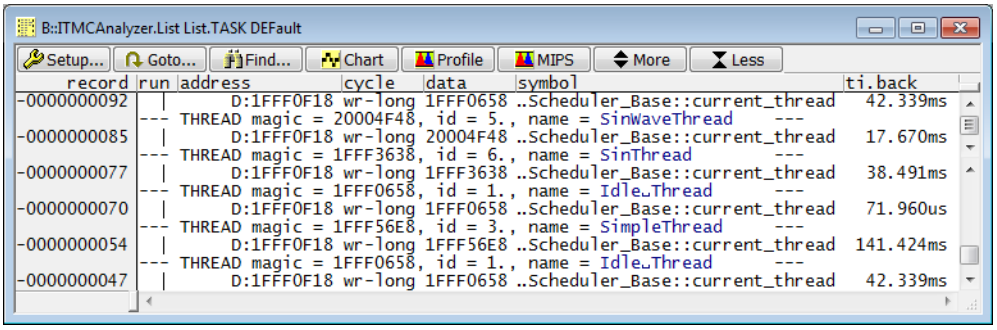
Tracing Task Switches

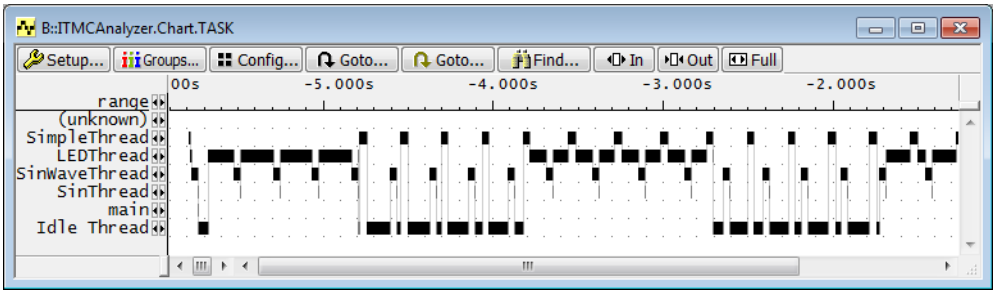
If an OS is running on your target, **OS-aware debugging** has to be configured in order to use OS-aware tracing.

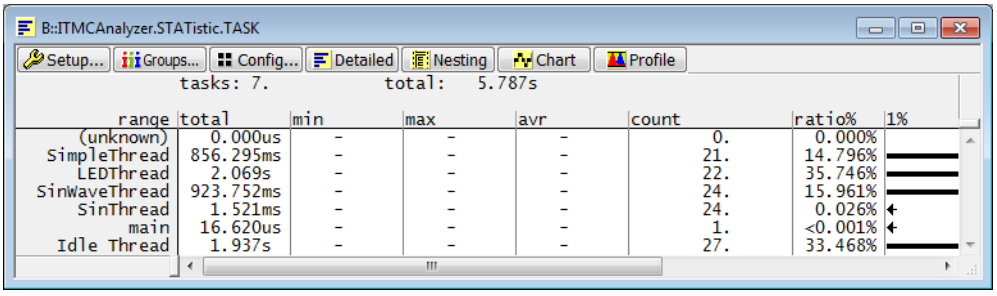
The OS writes the ID of the current task to variable that contains the information which task is currently running. If this write access is traced, the task behavior can be analyzed. TRACE32 PowerView uses a generic function to identify this variable.

TASK.CONFIG(magic)	Returns the address of the variable that contains the information which task/process is running.

```
ITM.DataTrace Data

Break.Set TASK.CONFIG(magic) /Write /TraceData
```







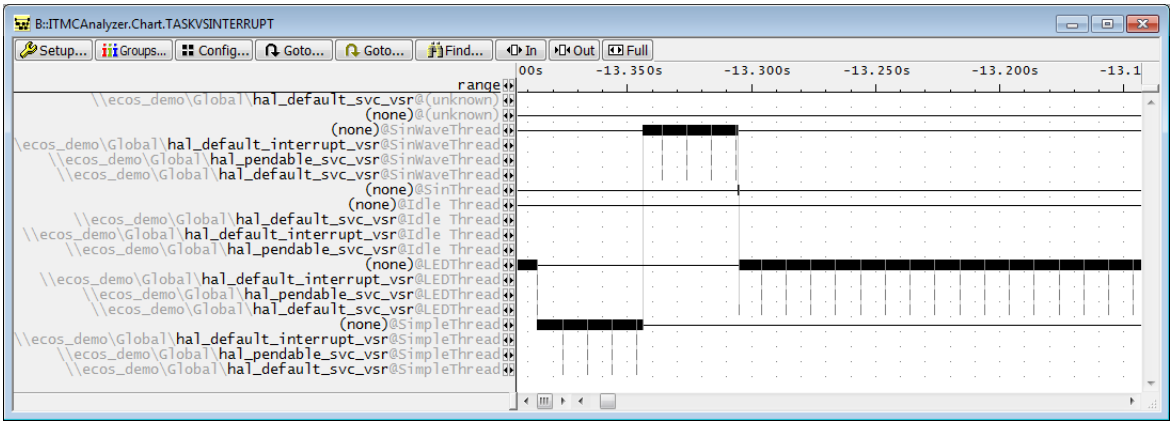

```
ITMCAnalyzer.List List.TASK DEFault

ITMCAnalyzer.Chart.TASK

ITMCAnalyzer.STATistic.TASK
```

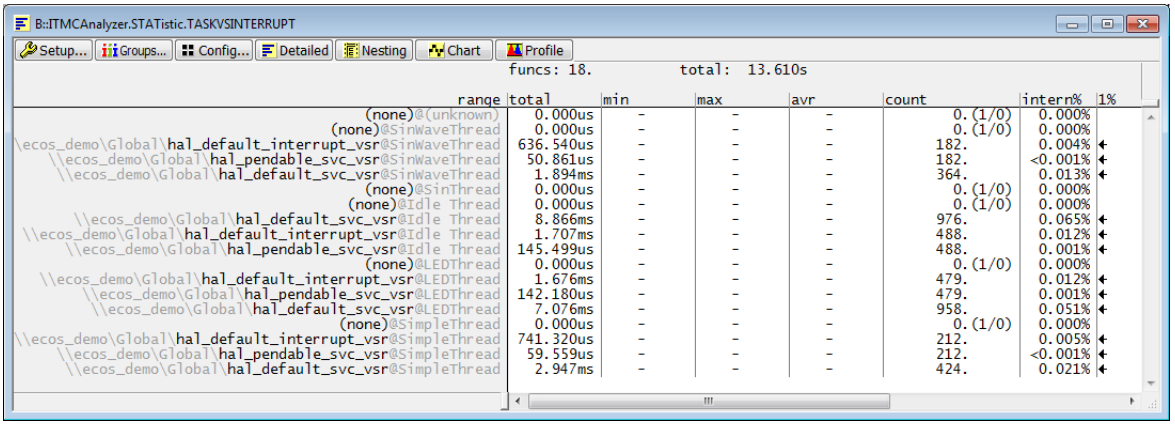
The following setup allows you to trace interrupts and task switches.

```
ITM.InterruptTrace ON

ITM.DataTrace Data

Break.Set TASK.CONFIG(magic) /Write /TraceData
```

The following commands allow you to analyze the run-time behavior of your system:

```
ITMCAnalyzer.Chart.TASKVSINTERRUPT

ITMCAnalyzer.STATistic.TASKVSINTERRUPT
```





(none)@*<task_name>* indicates the task was running.

<interrupt>@*<task_name>* indicates that *<interrupt>* interrupted the task *<task_name>*.

The DWT exports its data via the ITM. The ITM can add a timestamp to the data which is based on the CPU clock. The TRACE32 PowerView GUI can use one of the following timestamp sources:

- Either the external timestamps generated by the µTrace (MicroTrace)
- Or the timestamps of the ITM

The default is to use the externally generated timestamps of the µTrace (MicroTrace).

To enable timestamp generation of the ITM and to use the timestamps of the ITM, select the **CycleTrace** check box in the **ITM** window or type at the command line:

```
ITM.CycleTrace ON
```

Additionally you have to configure the clock rate of the ITM timestamp counter. The following command is available to configure this clock rate:

CAnalyzer.CLOCK *<frequency>*	Configures the debugger for the CPU clock frequency of the target.

If you are using the TPIU to export data, then the ITM timestamp counter is clocked with the CPU clock. So if your CPU runs at for example 64Mhz, you have to configure the ITM timestamp rate as 64Mhz with the command:

```
CAnalyzer.CLOCK 64Mhz
```

Merging ETM and DWT Data

The Cortex-M allows to use the ETM and DWT (transmitted via the ITM) in parallel. The TPIU will merge the information from ETM and ITM (containing the DWT data) into a single stream of bytes. This stream is then exported by the TPIU pins and recorded by the µTrace (MicroTrace).

Preconditions for merging the information:

1. The chip has both CoreSight components: (a) a DWT and (b) an ETM.
2. There *must not* be a single assembler instruction which accesses both: (a) Memory locations which are traced with the DWT and (b) memory locations which are not traced with the DWT.
   The reason is that under these circumstances the correlation between DWT data access information and ETM program flow becomes ambiguous.

If the preconditions are fulfilled, the DWT can be configured to emit information about the Program Counter value for a data access. The information from the DWT about data accesses can then be merged with the information about program flow from the ETM.

To configure the DWT and to merge the data flow and the program flow, select **CorrelatedData** from the **DataTrace** drop-down list in the **ITM** window (see Figure 1). Alternatively, type at the command line:

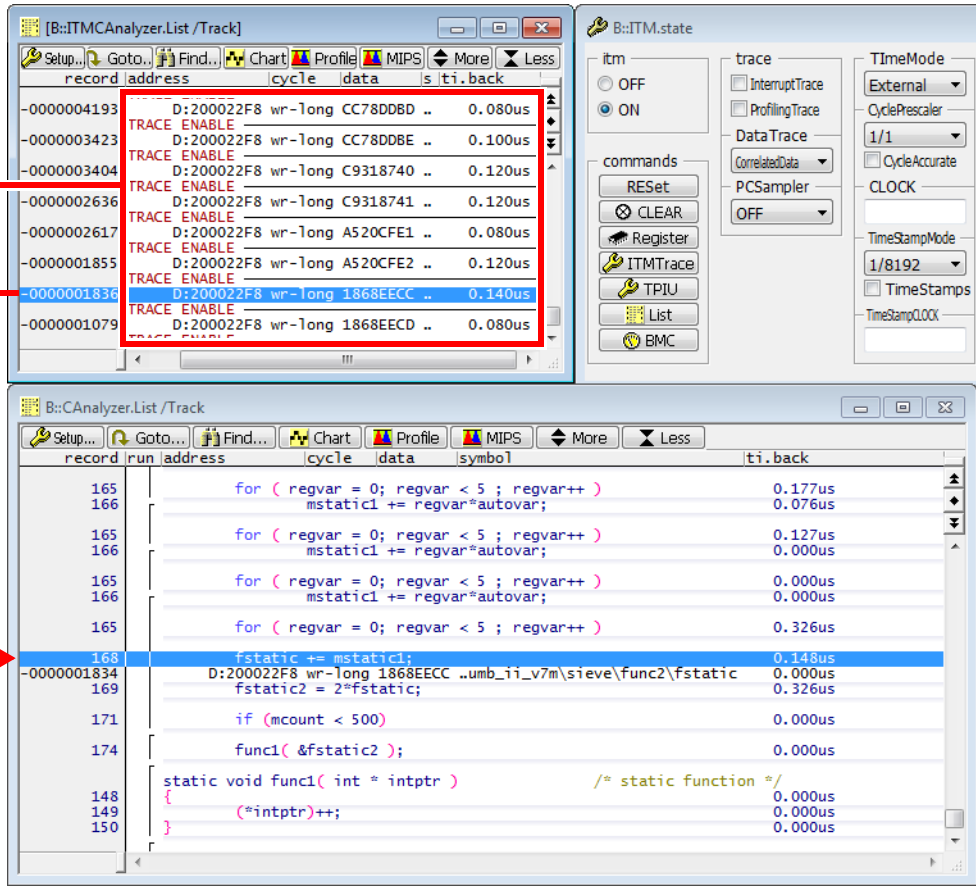
```
ITM.DataTrace CorrelatedData
```

To display the merged information in TRACE32, use the command **Trace.List** or **CAnalyzer.List**

Figure 1: A Merged Program Flow and Data Trace

Data flow from the DWT via ITM)

The program flow from the ETM (see HLL lines) is merged with the data flow from the DWT.



The TRACE32 PowerView GUI supports to enable and analyze the counters included in the DWT. The DWT counters emit information about:

- **CYC**, Cycle Counter: This counter counts the total number of CPU clock cycles. The counter will be used in conjunction with the CPU clock frequency to calculate the running times.
- **CPI**, This counter counts the total number of cycles per instruction *after* the first cycle. For example: If an instruction takes 5 cycles, the CPI counter will be incremented by 4. The slower this counter increases, the more instructions per cycle are executed.
- **EXC**, Exception Counter: This counter counts the number of cycles spent in interrupt processing specific operations. The counter counts the overhead incurred because of interrupts (like entry sequences which put registers onto the stack, exit sequences which restore registers from the stack, etc.).
- **SLP**, Sleep Counter: This counter counts the number of FCLK cycles the CPU spent sleeping.
- **LSU**, Load and Store Unit Counter: This counter counts the number of cycles spent in load and store instructions *after* the first cycle. For example: If a load instruction takes 4 cycles, the LSU counter will be incremented by 4. The slower this counter increases, the more instructions per cycle are executed.
- **FLD**, Fold Counter: In certain situations, the Cortex-M core is able to spent zero clock cycles for an instruction. Such instructions are called *folded* instructions. The FLD counter counts the number of folded instructions.

Additionally by analyzing the counters you can extract a MIPS (million instructions per second) over time.

To enable the DWT counters, select the **ProfilingTrace** check box in the **ITM** window (see Figure 2) or type at the command line:

```
ITM.ProfilingTrace ON
```

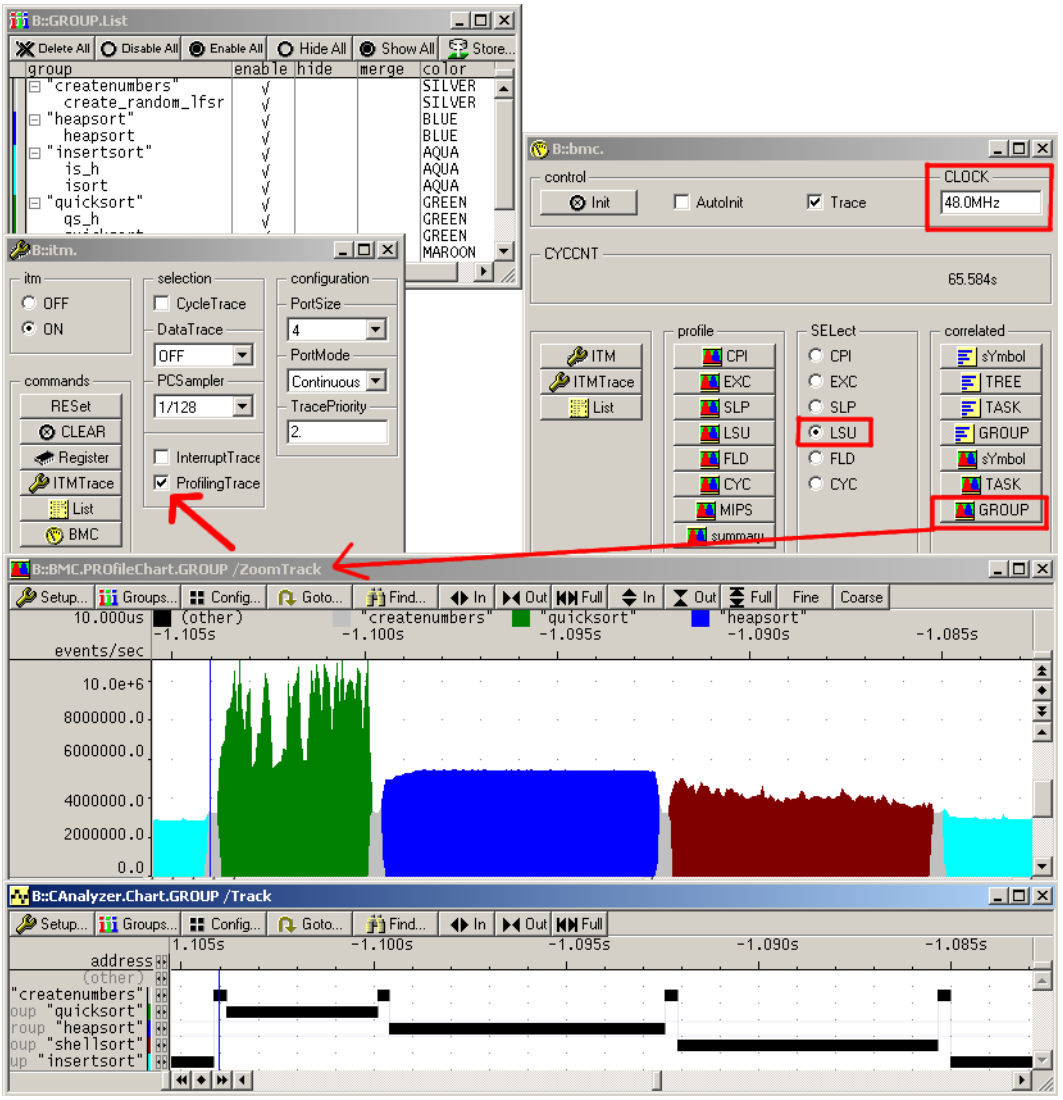
To get meaningful numbers, it is recommended to first group your program into interesting sections using this command group:

GROUP	Helps to structure application programs to ease the debugging process and the evaluation of the trace contents.

The TRACE32 PowerView GUI is then able to draw different counter rates over time and correlate the counter rates to the different sections you defined. You can select the counters via the **BMC** window (BenchMark-Counter); in this window, you also have to specify the CPU clock frequency. As an example, see Figure 2.

In this example, the *quicksort* algorithm produces the highest rate for the **LSU** counter. This means that the bottleneck for this algorithm is the access to memory where the data is stored; the CPU spends more cycles waiting for memory than in all other algorithms.
This is a *good* sign; it means that the code is very optimized, so that the CPU itself does not have to execute many non-load/store instructions.

Figure 2: LSU Counter Rate (in Events/Second) for Different Sorting Algorithms



Serial Wire Debug Port (SWDP) and Serial Wire Viewer (SWV)

This chapter describes how to configure the µTrace (MicroTrace) to use the Serial Wire Debug protocol and Serial Wire Viewer.

As mentioned before, Arm offers the Serial Wire Debug protocol, which uses only two pins. TRACE32 can debug a chip with the Serial Wire Debug protocol by using the following commands in this sequence:

```
SYStem.CONFIG DEBUGPORTTYPE SWD
SYStem.Up
```

If a chip is debugged with Serial Wire Debug protocol (instead of JTAG), you can put the TPIU in Serial Wire Viewer mode, which makes it possible to receive ITM data asynchronously via the Serial Wire Output signal. The Serial Wire Output signal is currently expected to be multiplexed with the unused JTAG TDO pin.

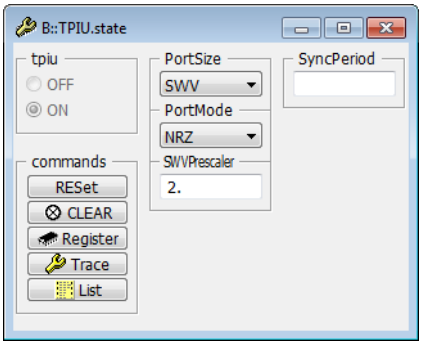
To put the TPIU into Serial Wire Viewer mode, the ETM has to be disabled via the **ETM** window, since the TPIU *only* supports ITM in Serial Wire Viewer mode. Alternatively, type at the command line:

```
ETM.OFF
```

When the ETM is disabled, you can switch the TPIU to Serial Wire Viewer mode. To do this, select **SWV** from the **PortSize** drop-down list in the **TPIU.state** window (see Figure below) or type at the command line:

```
TPIU.PortSize SWV
```

Figure 3: How to select Serial Wire Viewer mode in the TPIU.state window



The bit rate of the asynchronous Serial Wire Output signal is derived by dividing the CPU frequency. The frequency divider can be configured in the **TPIU.state** window via the **SWVPrescaler** input field, see Figure above. In this example, **2.** is selected, which stands for a bit rate of half the CPU frequency.

Alternatively, use the command **TPIU.SWVPrescaler**:

```
TPIU.SWVPrescaler 2.
```

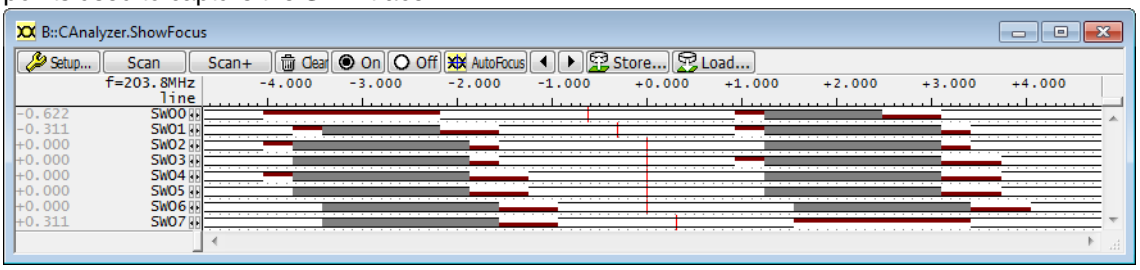
The µTrace (MicroTrace) has to know the bit rate of the asynchronous Serial Wire Output signal to sample the data correctly. Currently up to 200 MHz are supported (100 MHz with the MIPI34 whisker). You might need to choose a larger divider to remain in the allowed range. You can specify the bit rate manually with the following command:

CAnalyzer.TraceCLOCK *<frequency>*	Configures the frequency of the trace port, in this case the bit rate of the Serial Wire Output signal.

To auto-detect the bit rate, click the **AutoFocus** button in the **CAnalyzer** window or type at the command line:

```
CAnalyzer.AutoFocus
```

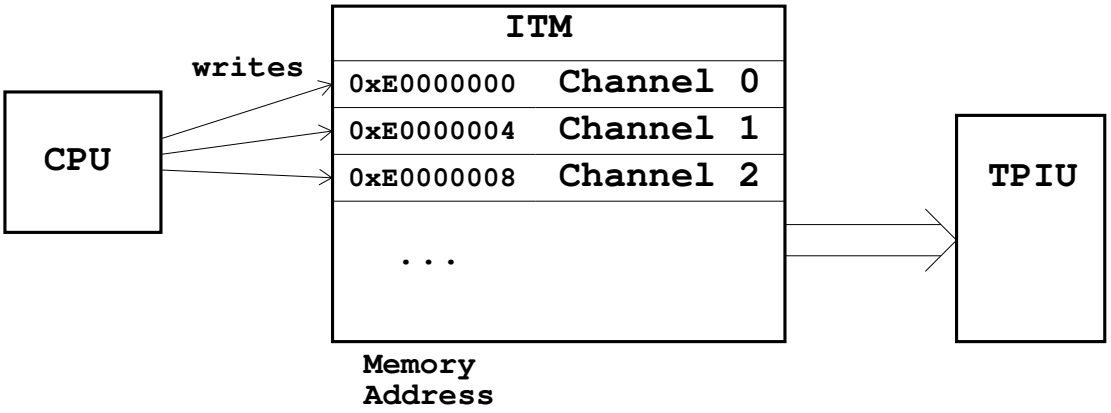
If you use the MIPI20T-HS whisker, you can also use the ShowFocus button to view and change the sample points used to capture the SWV trace:



The ITM offers another important feature to the user: It enables the CPU to emit data via the TPIU port or the Serial Wire Output signal.

The ITM provides 32 channels to the CPU. These ITM channels appear as memory-mapped peripheral in the CPUs memory space. The CPU can emit data through one channel by simply writing to the corresponding memory location. See Figure 4.

Figure 4: Memory Map for CPU accesses to the ITM Channels



The ITM acts as an extremely easy-to-use output device to send out debug related data. For example, it is very easy to implement debug “printf” like functionality by simply writing the “printf” strings to an ITM channel.

Of course using the ITM in this manner requires that you modify your software to access the ITM. So you have to *instrument* your software to output data you are interested in; this is the reason why the module is called *Instrumentation Trace Macrocell.*

Because the ITM already offers 32 channels it is also easy to implement thread safe code, by simply assigning one channel per thread.

Other usage models of the ITM can also be easily implemented. For example, you could use Channel 0 to log Function Entries by emitting the address of the function through channel 0.

Here is an example source code, which shows C macros to access the ITM from software.

```
static volatile unsigned int *ITM_BASE = (volatile unsigned int *)0xE0000000;
#define ITM_TRACE_D8(_channel_,_data_) { \
     volatile unsigned int *_ch_=ITM_BASE+(_channel_); \
     while ( *_ch_ == 0); \
     (*((volatile unsigned char *)(_ch_)))=(_data_); \
}

#define ITM_TRACE_D16(_channel_,_data_) { \
     volatile unsigned int *_ch_=ITM_BASE+(_channel_); \
     while ( *_ch_ == 0); \
     (*((volatile unsigned short *)(_ch_)))=(_data_); \
}

#define ITM_TRACE_D32(_channel_,_data_) { \
     volatile unsigned int *_ch_=ITM_BASE+(_channel_); \
     while ( *_ch_ == 0); \
     *_ch_ = (_data_); \
}

/* ... */
ITM_TRACE_D32(0,value); /* send a 32 bit value over ITM channel 0 */
```

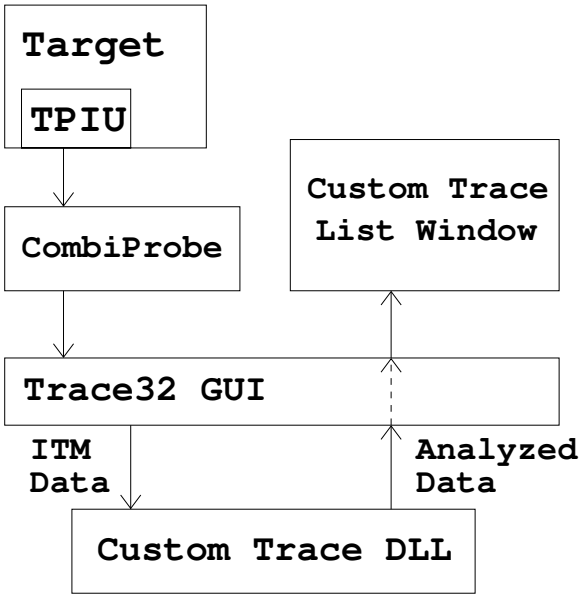
If you use the ITM to emit data from your software, the biggest challenge for the TRACE32 PowerView GUI is to present the data in a useful manner.

Since the TRACE32 PowerView GUI does not know how to interpret the data emitted by your software, the TRACE32 PowerView GUI can only show you the raw data from the ITM but not more.

To allow for a more useful representation of the data, the TRACE32 PowerView GUI offers an open API which makes it possible to load a user implemented custom DLL to the TRACE32 PowerView GUI. The DLL will then be used to analyze the payload of the ITM data (so the data emitted by the Software). The data produced by the DLL can then be displayed by the TRACE32 PowerView GUI.

Figure 5 shows the concept.

Figure 5: Concept how a Custom DLL analyses ITM Data

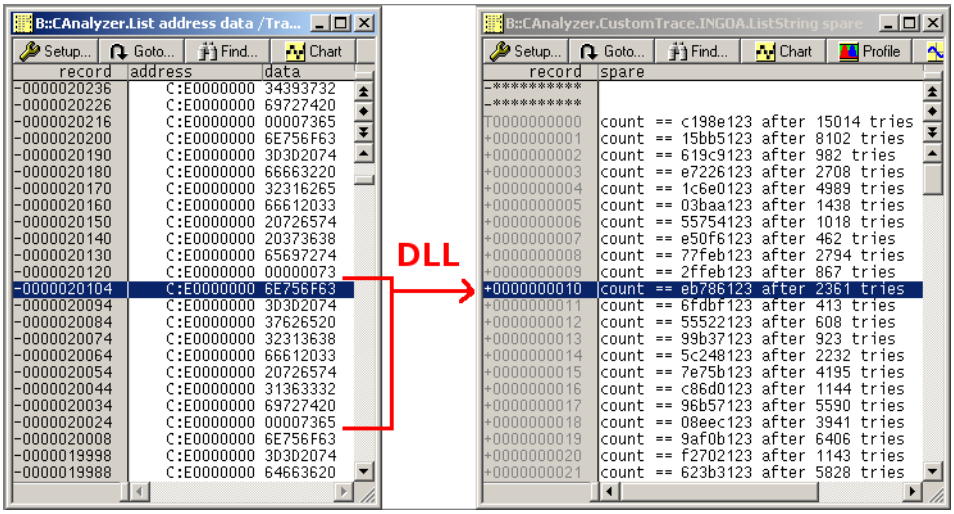


Here is a simple “printf” example in which the source code calls a “printf” function to output strings via ITM. A custom DLL is used by the TRACE32 PowerView GUI to decode the payload of the ITM data.

```
if ( (d1&0xFFF) == 0x123) {
     ITM_printf("count == %08x after %d tries",d1,numInt);
     numInt=0;
}
```

The source code produces the ITM data shown on the left of Figure 6. The right hand side is generated by a custom DLL loaded into the TRACE32 PowerView GUI.

Figure 6: A Simple “printf” Custom DLL Decoding Example



A custom trace demo is included in your TRACE32 installation. The demo runs under Windows and Linux in the TRACE32 Instruction Set Simulator.

NOTE:	Before you can run the demo under Linux, navigate to `~~/demo/customtrace/pipe_dll/` and compile with `make -f makefile.linux`

To access the custom trace demo in TRACE32:

1. Run this command `B::CD.PSTEP ~~/demo/customtrace/pipe_dll/*`
2. Select one of the following *_demo.cmm files:
   - dll_stp_demo.cmm
   - dll_csstm_demo.cmm
   - dll_itm_demo.cmm (for details about these files, refer to the readme.txt)

   The selected PRACTICE script file opens in the **CD.PSTEP** window in single step mode.
3. To run the demo, click **Continue**. The result will look similar to Figure 6.

dll_stp_demo.cmm	Raw STPv2 - MIPI **S**ystem **T**race **P**rotocol version 2
dll_csstm_demo.cmm	CoreSight STM - CoreSight **S**ystem **T**race **M**acrocell output via a CoreSight TPIU in CONTINUOUS mode
dll_itm_demo.cmm	CoreSight ITM - CoreSight **I**nstrumentation **T**race **M**acrocell output via a CoreSight TPIU in CONTINUOUS mode

On-the-fly Transfer of ITM and ETM Data

Recording ITM data usually does not require a high bandwidth; especially when dealing only with ITM data generated by software.

The low bandwidth makes it possible to transfer the data recorded by the µTrace (MicroTrace) on-the-fly to the TRACE32 PowerView GUI on the PC, while the recording is in progress.

The µTrace (MicroTrace) supports three different use cases for on-the-fly transfer of trace data:

1. Extending the Recording Size
2. Feeding Your Own Applications with Trace Data
3. Real-time Profiling with the ETM

The three use cases are discussed in the following sections.

When the µTrace (MicroTrace) transfers data on-the-fly, it behaves like a FIFO: The data coming from the target is buffered in the memory of the µTrace (MicroTrace) and is then transferred to the PC.

The same mechanism works for ETM data, as long as the bandwidth of the generated ETM data is not too high.

The on-the-fly transfer to the PC allows to process the data on-the-fly and to store the data to hard disk, while the recording is in progress.

Extending the Recording Size

The first use case is to stream the trace data to disk to extend the recording size, if you want more than the 128MiB256MiB the µTrace (MicroTrace) has internally.

To enable this use case, select **STREAM** mode in the **CAnalyzer** window or type at the command line:

```
CAnalyzer.Mode STREAM
```

In **STREAM** mode, all data will be transferred to your hard disk on-the-fly. The following commands are available for storing the trace data to a file on your hard disk:

<trace>.STREAMCompression	Select compression mode for streaming.
<trace>.STREAMFileLimit	Set size limit for streaming file.
<trace>.STREAMFILE	Specify your own streaming file.

When you open a **CAnalyzer.List** window, the TRACE32 PowerView GUI will access the recorded data stored on your hard disk.

The second major use case is supported by PIPE mode. To enable this use case, select **PIPE** mode in the **CAnalyzer** window or type at the command line:

```
CAnalyzer.Mode PIPE
```

In this use case, data is on-the-fly transferred to the PC and processed. Processing in this context means, that the TRACE32 PowerView GUI will decode the TPIU formatter and ITM protocol into a more general format. This preprocessed data can then be:

- Stored into a file
- Send to a “Named Pipe”
- Passed to a user implemented DLL

If the data is stored into a file, the data can be further processed later on by your own software. The TRACE32 PowerView GUI allows to filter the ITM data by channels; it is possible to open up to eight files, each of which receives data from a different set of ITM channels. The command to store the data to a file is:

CAnalyzer.WRITE *<file>* **/ChannelID** *<range_or_mask>*

If you have a software application which wants to receive and process data from the ITM on-the-fly, you can open up a “Named Pipe”. The TRACE32 PowerView GUI can send the ITM data to up to eight different Named Pipes; as with files the TRACE32 PowerView GUI can be configured to only send a specific set of channels to each of the Named Pipes. The command to send the data to a Named Pipe is:

CAnalyzer.PipeWRITE *<named_pipe>* **/ChannelID** *<range_or_mask>*

Probably the most flexible approach is to load your own DLL into the TRACE32 PowerView GUI. The TRACE32 PowerView GUI will pass all received ITM data to your DLL. In the DLL you can filter and distribute the data in any manner you like.

You can combine all three possibilities, so you can *in parallel*:

- Store the ITM data to a file
- Send it to a Named Pipe
- Pass it to a DLL

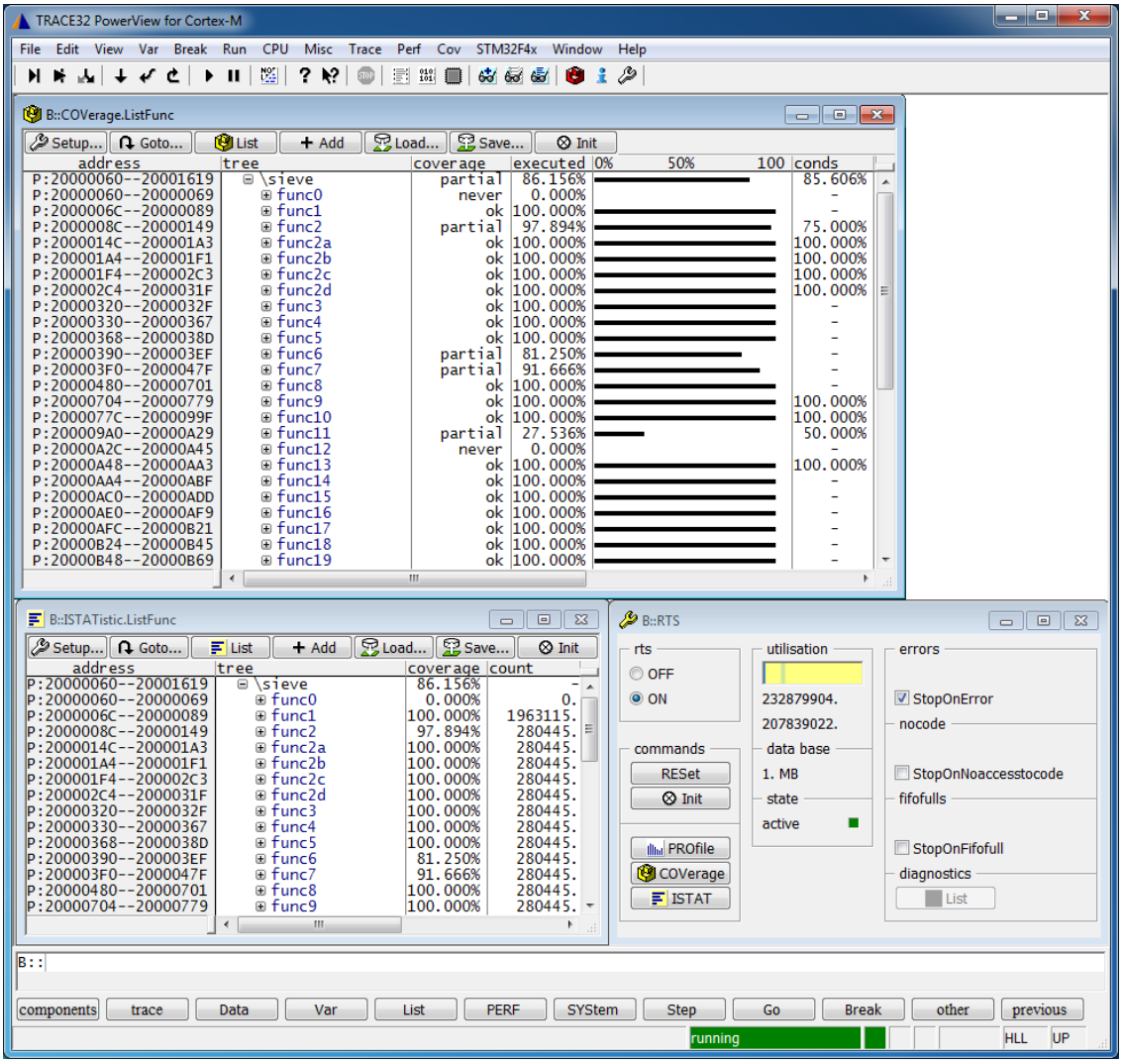
Note that the TRACE32 PowerView GUI treats DLLs for custom trace processing and DLLs for PIPE mode processing in exactly the same manner. Conceptually there is no difference: In both cases the TRACE32 PowerView GUI will feed the ITM data to your DLL; your DLL analyses the data and either passes it back to the TRACE32 PowerView GUI (in the case of a custom trace DLL) or it sends it to another application.

This means that if you use a custom trace DLL in PIPE mode, the processing will take place while the recording is in progress. If you stop the recording, the processing will already be finished and the processed data can be viewed instantaneously.

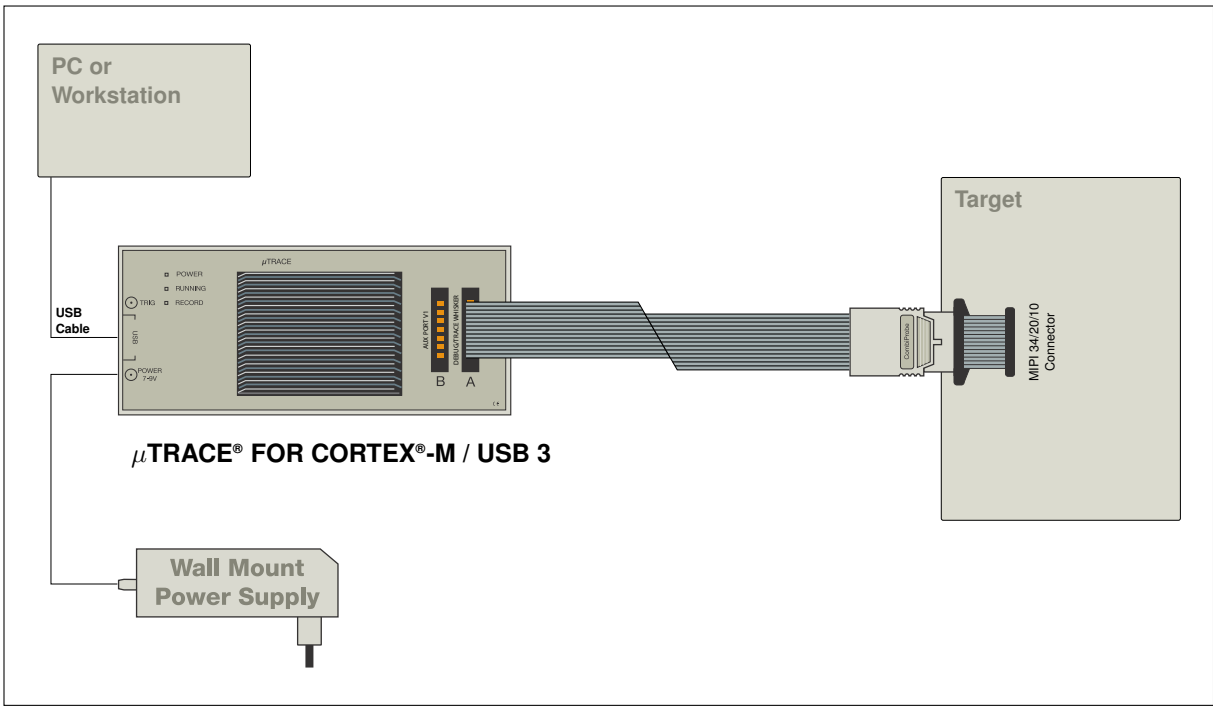
The third use case is to analyze ETM data on-the-fly: If the Cortex-M core implements an ETM, the µTrace (MicroTrace) can be used to transfer and analyze ETM data on-the-fly (e.g. while the core is executing its program).

By analyzing the ETM data on-the-fly, the TRACE32 PowerView GUI can visualize (for example) how often each function has been executed and if there are code parts which have not been executed at all. The analysis of ETM data takes place without stopping or influencing the Cortex-M core. For details, please refer to the **RTS** command group.

Figure 7: Real-Time Profiling of ETM data



µTrace (MicroTrace) (with CombiProbe MIPI34 Whisker)



You have chosen the all-in-one debug and off-chip trace solution developed by Lauterbach especially for Cortex-M processors.

The combination of µTrace (MicroTrace) and MIPI34 whisker supports:

- debugging via JTAG (IEEE 1149.1), SWD (Serial Wire Debug) or cJTAG (IEEE 1149.7) at clock rates up to 100 MHz
- debug connectors MIPI-20T, MIPI-10, MIPI-34 and MIPI-20D (without adapter), ARM-20 and TI-14 (with included adapter)
- parallel trace using ETM/TPIU continuous mode with up to 4 data pins and bit rates of up to 200 Mbit/s per pin.
- SWV (Serial Wire Viewer) / SWO (Serial Wire Output) trace at port rates up to 100 Mbit/s

Please refer to **"Cortex-M Debugger"** (debugger_cortexm.pdf) for all Cortex-M specific debug features.

This manual describes the basic setups and all Cortex-M specific trace features.

Deprecated Connectors

The following connectors are rarely used and not supported by the MIPI20T-HS whisker. To use them, you currently need to use the older MIPI34 whisker (limited to 100 Mbit/s per pin). Contact Lauterbach support for alternative solutions.

34-Pin Debug, SWO and Trace Connector

The signals RTCK, DBGRQ/EMU0, DBGACK/EMU1 are not supported by µTrace (MicroTrace).

Signal	Pin	Pin	Signal
VREF DEBUG	1	2	TMS
GND	3	4	TCK
GND	5	6	TDO
(KEY) GND	7	8	TDI
GND	9	10	RESET-
GND	11	12	RTCK
GND	13	14	BCE
GND	15	16	TRST-
GND	17	18	TRIGIN
GND	19	20	TRIGOUT
GND	21	22	TRC CLK
GND	23	24	TRC DATA0
GND	25	26	TRC DATA1
GND	27	28	TRC DATA2
GND	29	30	TRC DATA3
GND	31	32	TRC EXT
GND	33	34	VREF TRACE

20-Pin Debug and SWO Connector

The signals RTCK, DBGRQ/EMU0, DBGACK/EMU1 are not supported by µTrace (MicroTrace).

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS|TMSC|SWDIO
GND	3	4	TCK|TCKC|SWCLK
GND	5	6	TDO|-|SWO
GND (KEY)	-	8	TDI
GND	9	10	RESET-
GND	11	12	RTCK
GND	13	14	TRST- PULLDOWN
GND	15	16	TRST-
GND	17	18	DBGRQ (EMU0)
GND	19	20	DBGACK (EMU1)