Training Intel® Processor Tracing

Release 09.2023

MANUAL

Training Intel® Processor Tracing

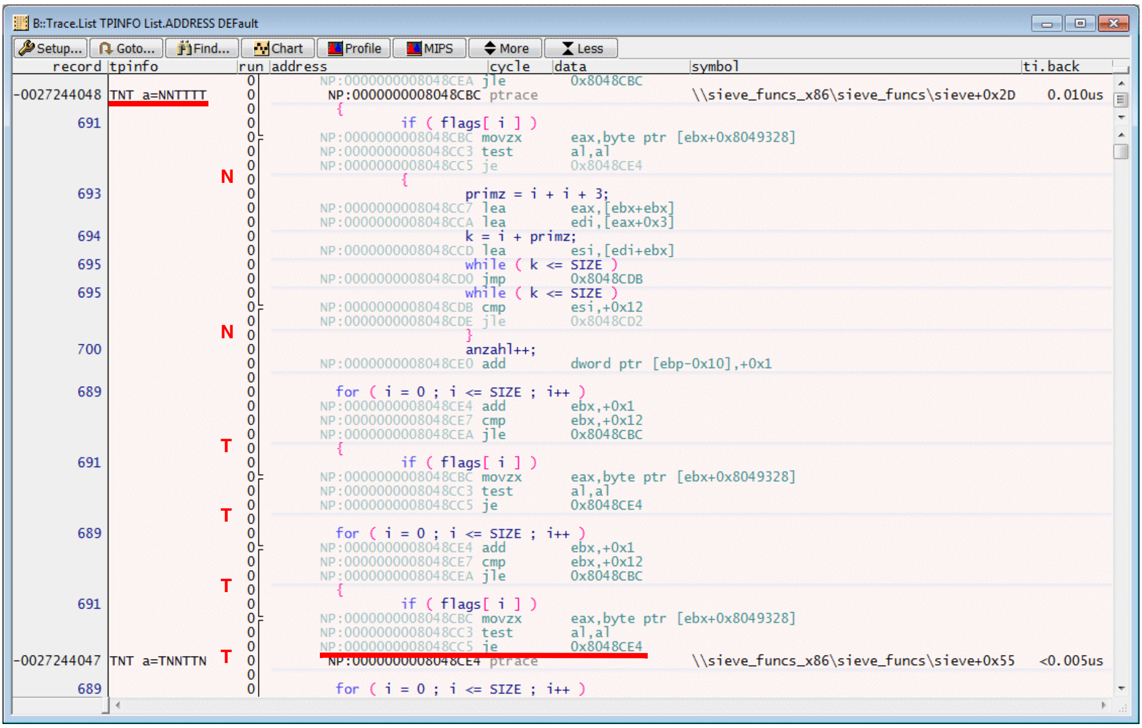
Version 10-Oct-2023

Basic Trace Packets

To enable a trace tool to reconstruct the instruction execution sequence the following trace packets are generated:

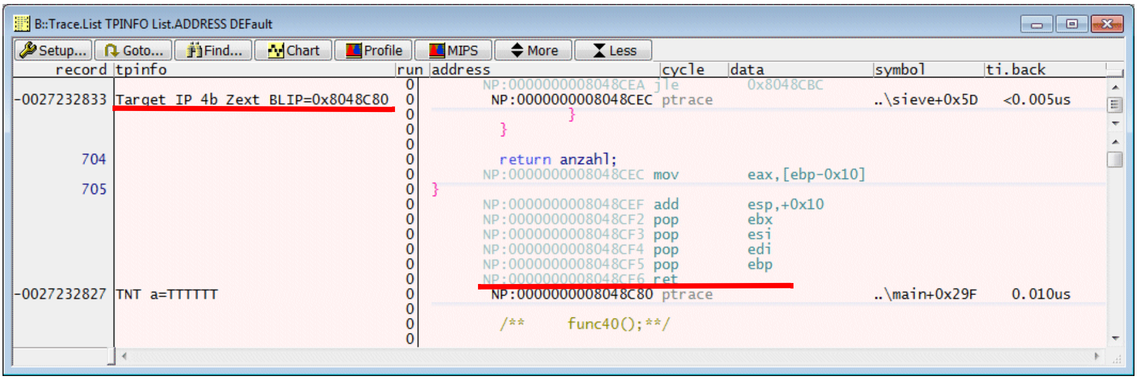
TNT packets

Taken Not Taken packets track the direction of up to 6 conditional branches. Since the address at which the program execution continues when the branch was taken is part of the source code TNT packets provide sufficient information to reconstruct the instruction execution sequence.



Target IP packets

Ret instructions, register indirect calls and similar instructions as well as exception and interrupts cause the generation of a Target IP packet. Since the address at which the program execution continues is only known at run-time, a Target IP packet contains this address fully or in a compressed format.

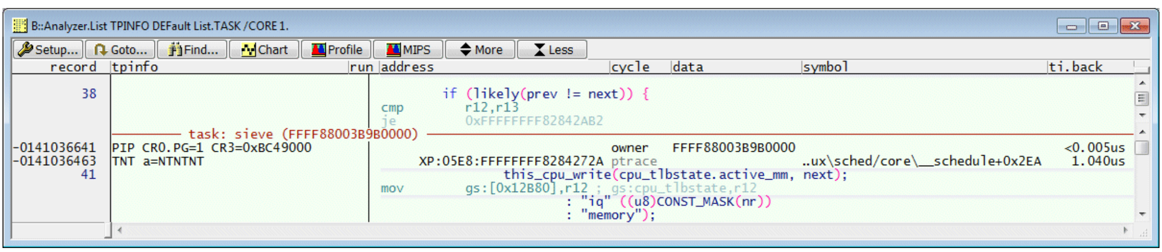


OS-Aware Tracing

Paging Information Packet (PIP)

x86/x64 processors have a CR3 control register that contains the Process Context Identifier (PCID). On every context switch the corresponding PCID is loaded to CR3.

Intel® PT generates a Paging Information Packet (PIP) when a write to CR3 occurs.



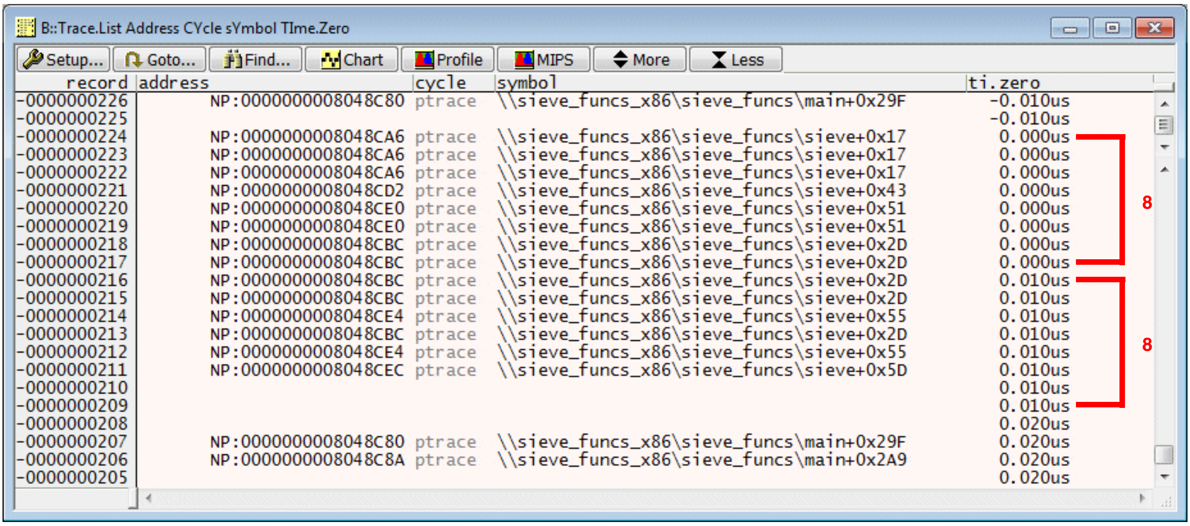
Tool Timestamp (POWER TRACE II / POWER TRACE III only)

POWER TRACE II / POWER TRACE III timestamps the trace information when it is stored in its trace buffer.

The resolution of the POWER TRACE II / POWER TRACE III timestamp is 5 ns.

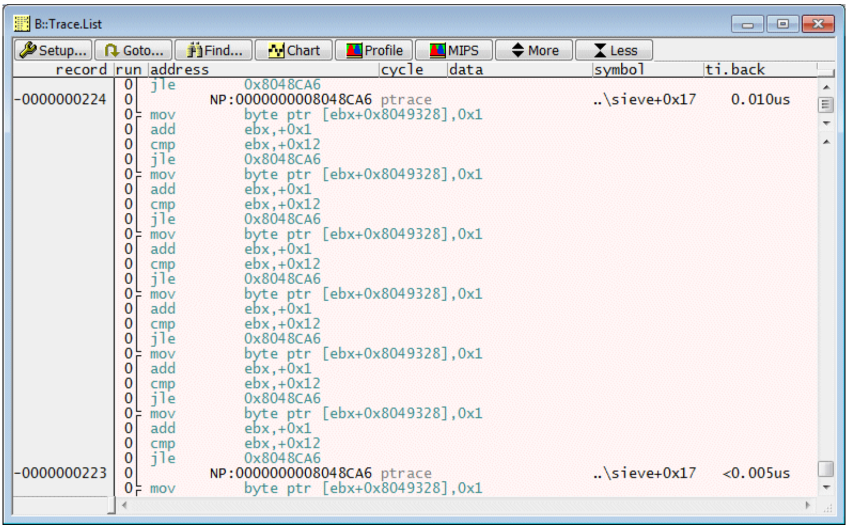
8 trace record have always an identical timestamp. There are two reasons for this:

- The TRACE32 recording technology.
- The smallest Intel® PT packet is one byte.



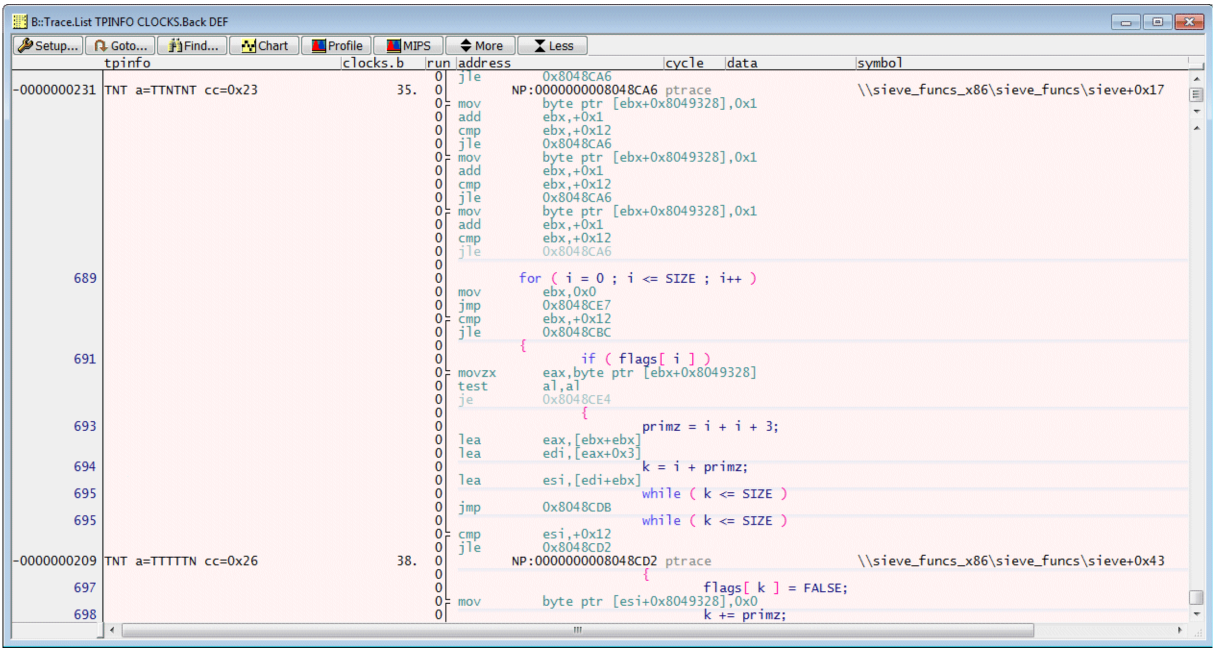
record	address	cycle	symbol	ti.zero
-0000000226	NP:000000008048C80	ptrace	\\sieve_funcs_x86\sieve_funcs\main+0x29F	-0.010us
-0000000225				-0.010us
-0000000224	NP:000000008048CA6	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x17	0.000us
-0000000223	NP:000000008048CA6	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x17	0.000us
-0000000222	NP:000000008048CA6	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x17	0.000us
-0000000221	NP:000000008048CD2	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x43	0.000us
-0000000220	NP:000000008048CE0	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x51	0.000us
-0000000219	NP:000000008048CE0	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x51	0.000us
-0000000218	NP:000000008048CBC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x2D	0.000us
-0000000217	NP:000000008048CBC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x2D	0.000us
-0000000216	NP:000000008048CBC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x2D	0.010us
-0000000215	NP:000000008048CBC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x2D	0.010us
-0000000214	NP:000000008048CE4	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x55	0.010us
-0000000213	NP:000000008048CBC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x2D	0.010us
-0000000212	NP:000000008048CE4	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x55	0.010us
-0000000211	NP:000000008048CEC	ptrace	\\sieve_funcs_x86\sieve_funcs\sieve+0x5D	0.010us
-0000000210				0.010us
-0000000209				0.010us
-0000000208				0.020us
-0000000207	NP:000000008048C80	ptrace	\\sieve_funcs_x86\sieve_funcs\main+0x29F	0.020us
-0000000206	NP:000000008048C8A	ptrace	\\sieve_funcs_x86\sieve_funcs\main+0x2A9	0.020us
-0000000205				0.020us

In the standard trace display timestamp information is displayed for the first record with the new timestamp. All following records with an identical timestamp show <0.005us.



CycleAccurate Tracing

If configured Intel® PT can generate cycle count information. The cycle count information indicates how much core clocks it took to execute a program section.



Cycle accurate tracing requires up to 2 times more bandwidth.

Synchronization Time

Not implemented yet.

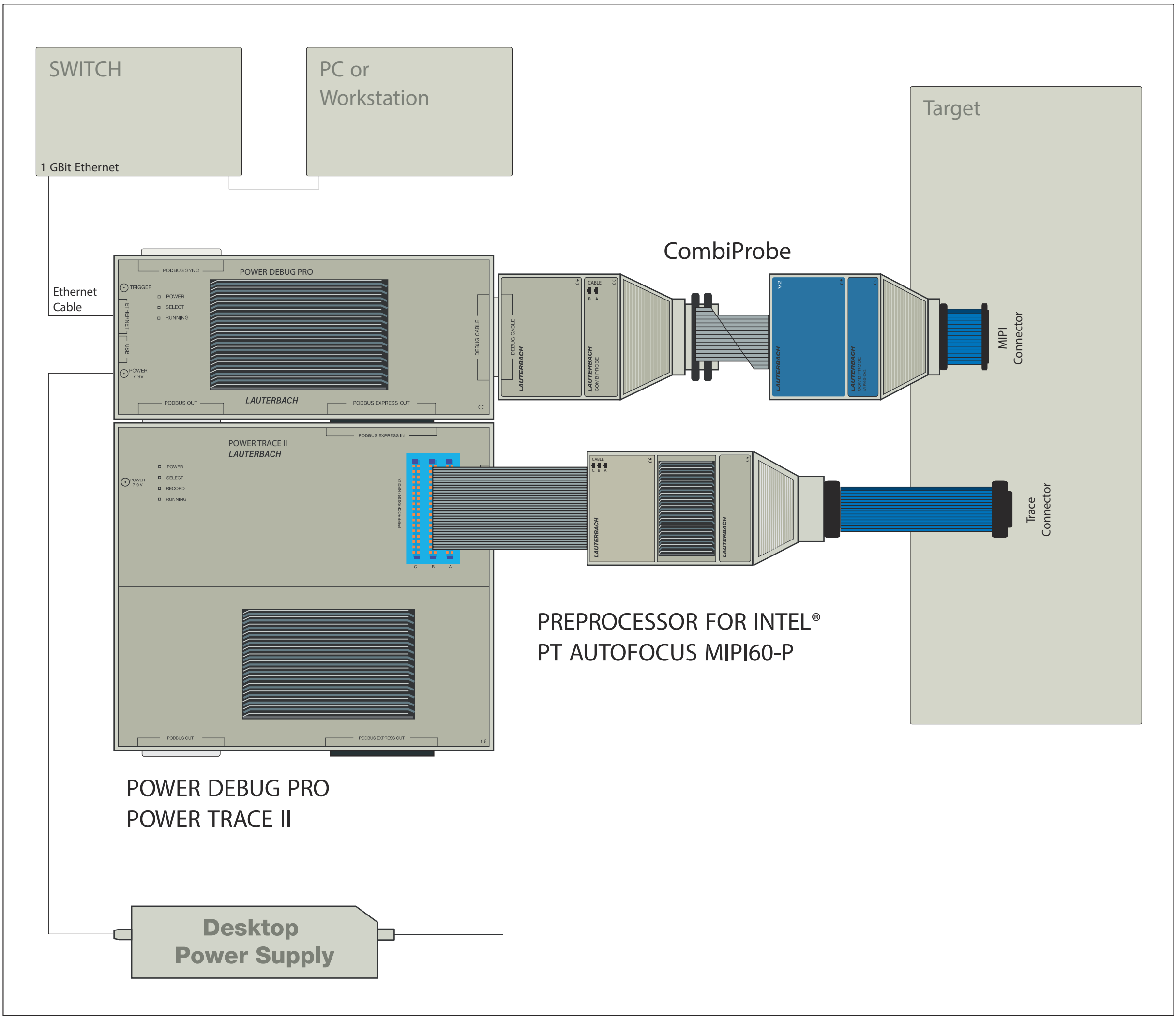
Off-chip Trace

Recording the trace information exported via a PTI (Parallel Trace Interface) requires:

- A POWER TRACE II hardware (1 GByte, 2 GByte or 4 GByte of trace memory) or a POWER TRACE III hardware (4 GByte or 8 GByte of trace memory)

  TRACE32 PowerView uses the name **Analyzer** to refer to the trace memory within POWER TRACE II / POWER TRACE III.

- An *Preprocessor for Intel® Atom™ AUTOFOCUS 600 MIPI*



The following configuration steps are required for off-chip tracing:

1. **Configure Parallel Trace Interface on target.**

Configuration is required for:

- PTI port size

- PTI frequency

- GPIO pins used for PTI

The following commands are provided for this purpose:

; write *<value>* to the configuration register addressed by **A:***<physical_address>*
; in the specified *<format>*
PER.Set.simple **A:***<physical_address>* **%***<format>* *<value>*

; write *<value>* to the memory location addressed by **A:***<physical_address>*
; in the specified *<format>*
Data.Set **A:***<physical_address>* **%***<format>* *<value>*

Data.Set is equivalent to **PER.Set.simple** if the configuration register is memory mapped.

The **access class** **A:** allow to use the physical address for the write operations.

```
Per.Set.simple A:0xf9009000 %Long 0x3e715

Data.Set A:0xf9009000 %Long 0x3e715
```

Please refer to your chip manual for the physical addresses of the configuration registers.

2. **Configure TRACE32 for a PTI that exports STP (System Trace Protocol) packets.**

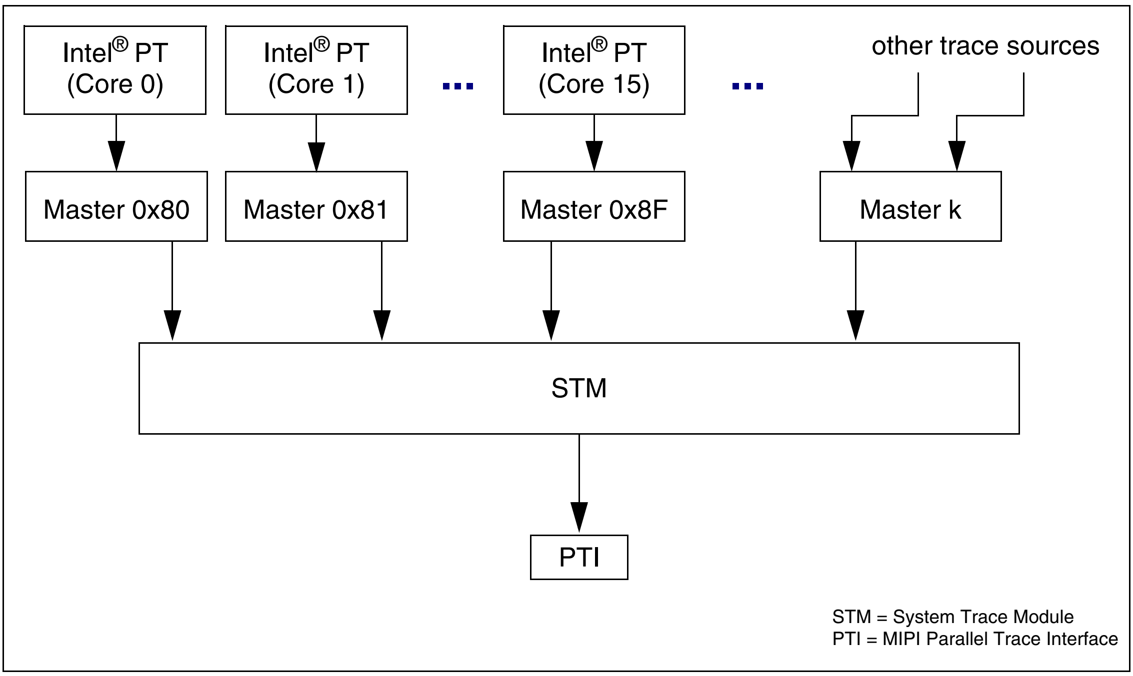
```
SYStem.CONFIG STM Mode STP64          ; inform TRACE32 that your
                                      ; chip provides a STM that
                                      ; generated 64-bit STPv1
                                      ; packets

STM.PortSize 16.                      ; inform TRACE32 that your
                                      ; PTI size is 16 pins
```

3. **Inform TRACE32 which core traces you want to analyze.**

IPT.TraceID *<value>* | *<bitmask>*	Specify which masters/channels (that produce Intel® PT trace information) you want to analyze with the help of TRACE32.

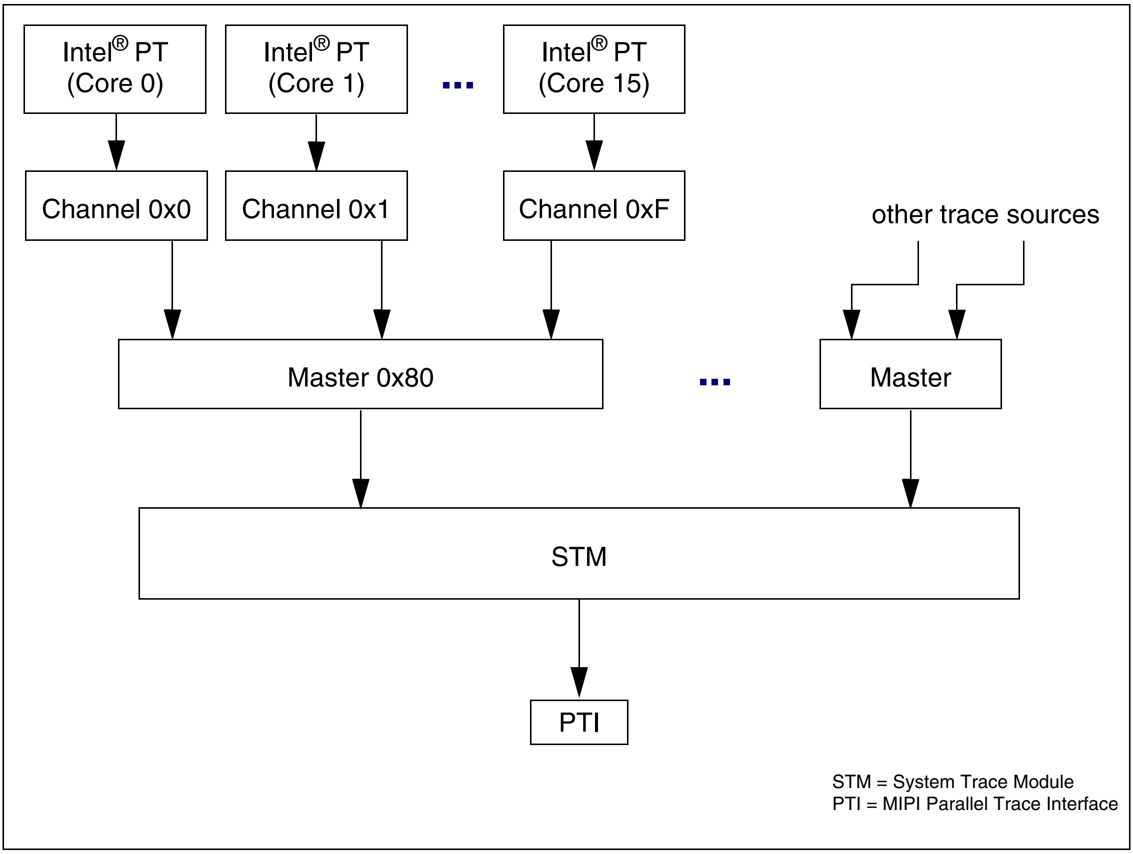
<value>	*<value>* is a 32-bit number. The first 16 bits represent the master ID, the last 16 bits represent the channel ID.
<bitmask>	bitmask representation of *<value>*

Example 1: Each core has its own master ID.




```
IPT.TraceID 0x00800000        ; master ID 0x80 is used to export Intel®
                              ; PT trace information for core 0

IPT.TraceID 0x008x0000        ; master ID 0x80, 0x81, 0x82 … are used to
                              ; export Intel® PT trace information
                              ; master ID 0x80 represents core 0
                              ; the other master IDs consecutively
                              ; represent core 1 to core 15
```

Example 2: Each core has its own channel ID, all cores use the same master ID.




```
IPT.TraceID 0x0080000x        ; master ID 0x80 is used to export Intel®
                              ; PT
                              ; trace information
                              ; channel ID 0x0 represents core 0
                              ; the other channel IDs consecutively
                              ; represent core 1 to core 15
```

4. **Enable Intel® PT on the target and allow TRACE32 to configure it.**

```
IPT.ON
```

5. **Calibrate the *Preprocessor for Intel® Atom™ AUTOFOCUS 600 MIPI* for recording.**

TRACE32 supports three methods of generating outputs on the trace lines for calibration.

- On-chip test pattern generator (not tested yet).
- Test executable provided by Lauterbach.
- Application program.

Please be aware that TRACE32 PowerView displays “**Analyzer data capture o.k.**” only if:

- All trace lines toggled while calibration is performed.
- There are no short circuits between the trace lines.
- An error-free trace decoding was possible.

Test executable provided by Lauterbach

In order to use the test executable provided by Lauterbach for calibration, the following command sequence is recommended.

```
; example for a free-running clock (Tangier)

AREA.view                             ; open TRACE32 Message AREA
                                      ; to observe calibration
                                      ; results

Analyzer.THreshold VCC                ; advise TRACE32 to use
                                      ; 1/2 VCC as threshold level
                                      ; for the trace signals

Analyzer.AutoFocus /NoTHreshold       ; start the calibration by
                                      ; using test executable

                                      ; advise TRACE32 to keep
                                      ; the threshold level
```

A manual setup is required if your target is using a gated clock. Refer to **“Manual Setup”** in AutoFocus User’s Guide, page 18 (autofocus_user.pdf) for assistance.

Application program

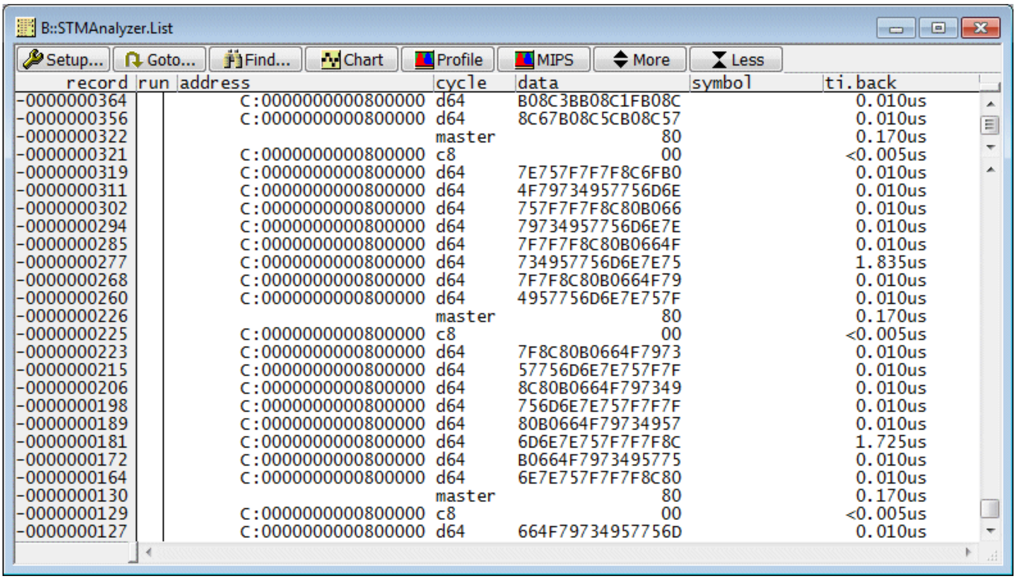
In order to use the application program for calibration, the following command sequence is recommended.

```
; example for a free-running clock (Tangier)

AREA.view                               ; open TRACE32 Message AREA
                                        ; to observe calibration
                                        ; results

Data.LOAD.Elf demo_x86.elf /PlusVM      ; download application program
                                        ; to the target,
                                        ; in order to perform trace
                                        ; decoding while the
                                        ; application program is
                                        ; running, the program code
                                        ; has to be copied to the
                                        ; TRACE32 Virtual Memory

Go                                      ; start the execution of the
                                        ; application program

Analyzer.THreshold VCC                  ; advise TRACE32 to use
                                        ; 1/2 VCC as threshold level
                                        ; for the trace signals

Analyzer.AutoFocus /NoTHreshold         ; start the calibration

                                        ; advise TRACE32 to keep
                                        ; the threshold level

WAIT 1.s                                ; wait 1 second

Break                                   ; stop the program execution
```

A manual setup is required if your target is using a gated clock. Refer to **“Manual Setup”** in AutoFocus User’s Guide, page 18 (autofocus_user.pdf) for assistance.

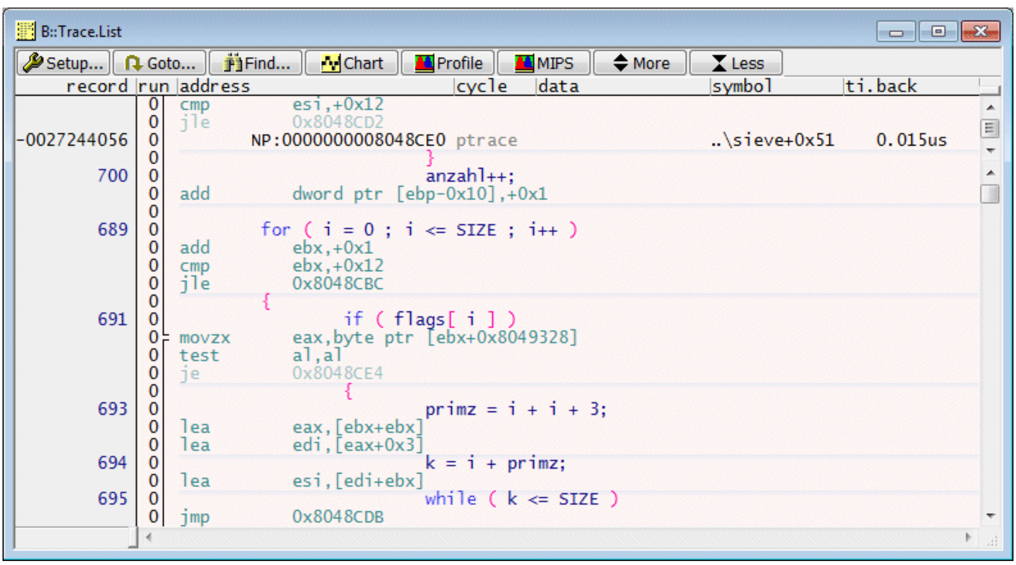
After a successful configuration of the off-chip tracing the following command can be used to inspect the STP packet stream:

STMAnalyzer.List	Display STP packet stream recorded to POWER TRACE II / POWER TRACE III.

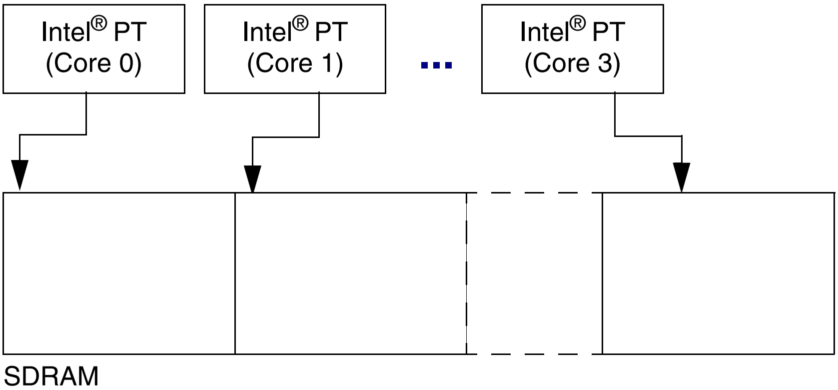


The Intel® PT based core traces can be displayed by the following command:

Analyzer.List	Display all core trace information decoded out of the STP packet stream.



If the Intel® PT trace information is routed to SDRAM, a fixed amount of memory is assigned to each core. The max. SDRAM size per core is currently 4 MByte.



Configure TRACE32

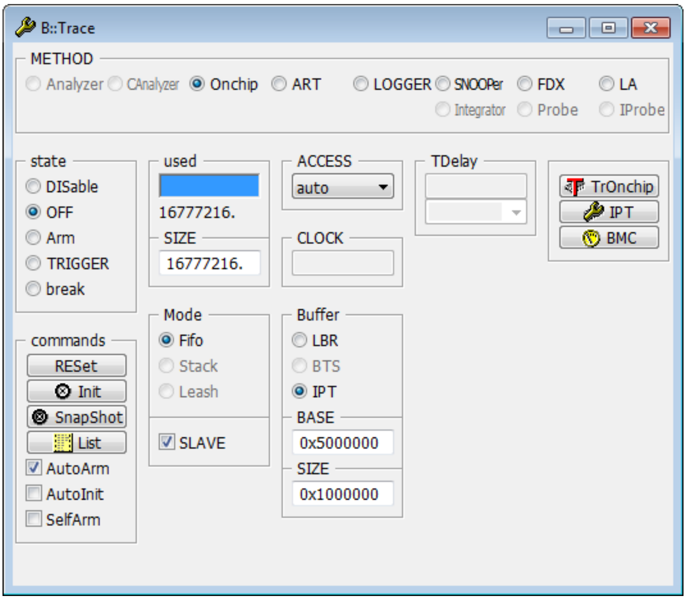
1. **Advise TRACE32 to read the trace information from SDRAM.**

```
Trace.METHOD Onchip
```

TRACE32 reads the onchip trace via JTAG.

2. **Provide further details on the SDRAM configuration to TRACE32.**

```
Onchip.Buffer IPT                  ; inform TRACE32 that the SDRAM
                                   ; provides Intel® PT trace
                                   ; information

Onchip.Buffer BASE 0x5000000       ; inform TRACE32 that the SDRAM
                                   ; allocated for Intel® PT trace
                                   ; starts at address 0x5000000

Onchip.Buffer SIZE 0x1000000       ; inform TRACE32 that the SDRAM
                                   ; allocated for Intel® PT trace has
                                   ; a size of 16 MByte
```



3. **Enable Intel® PT on the target and allow TRACE32 to configure it.**

```
IPT.ON
```

If the command **Onchip.List** is used, TRACE32 merges the Intel® PT traces from the individual cores as follows:

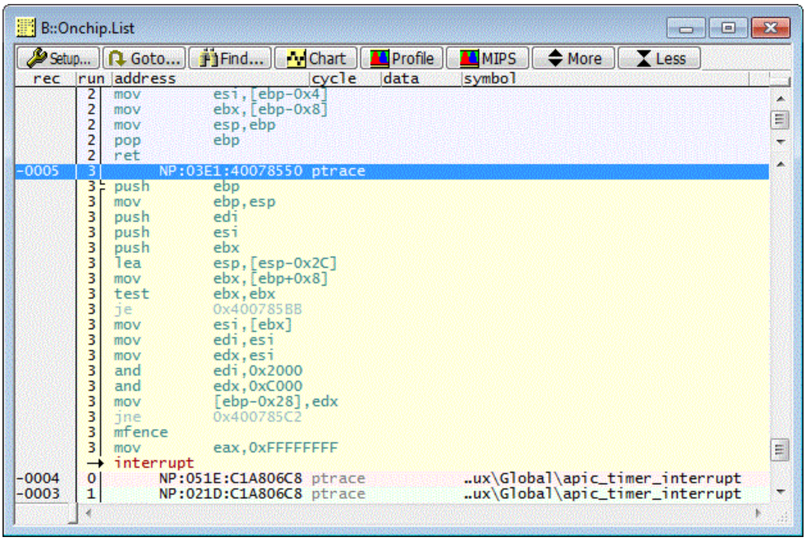
SDRAM block core 0	SDRAM block core 1	SDRAM block core 2	SDRAM block core 3
Intel® PT packet 1	Intel® PT packet 1	Intel® PT packet 1	Intel® PT packet 1
Intel® PT packet 2	Intel® PT packet 2	Intel® PT packet 2	
Intel® PT packet 3	Intel® PT packet 3	Intel® PT packet 3	
Intel® PT packet 4	Intel® PT packet 4	Intel® PT packet 4	
Intel® PT packet 5	Intel® PT packet 5	Intel® PT packet 5	
	Intel® PT packet 6	Intel® PT packet 6	
	Intel® PT packet 7		

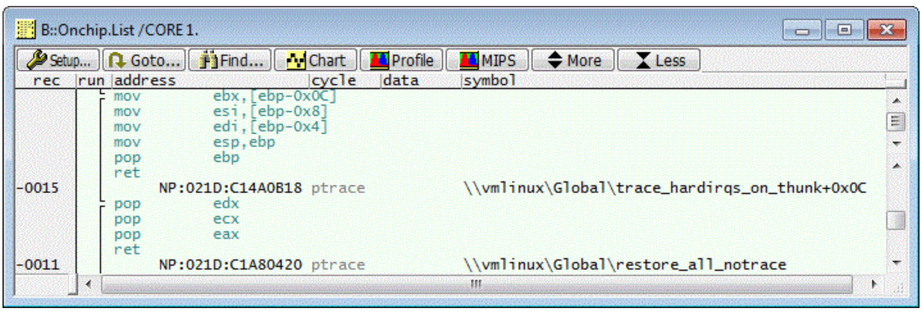
Onchip.List

Intel® PT packet 1 of core 0
Intel® PT packet 1 of core 1
Intel® PT packet 1 of core 2
Intel® PT packet 1 of core 3
Intel® PT packet 2 of core 0
Intel® PT packet 2 of core 1
Intel® PT packet 2 of core 2
Intel® PT packet 3 of core 0

This procedure will change with the decoding of synchronisation packets.

```
Onchip.List                              ; display trace listing for all
                                         ; cores

Onchip.List /CORE 1                      ; display trace listing for core 1
```



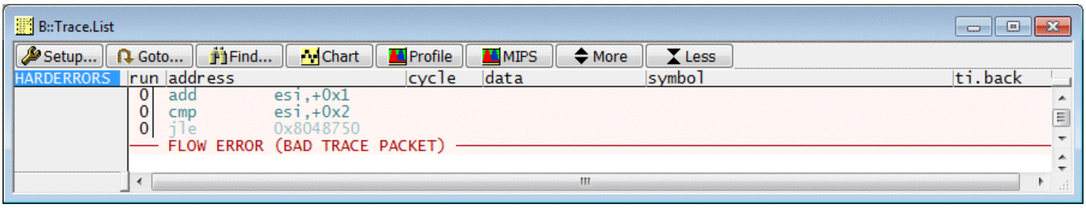


ERRORS

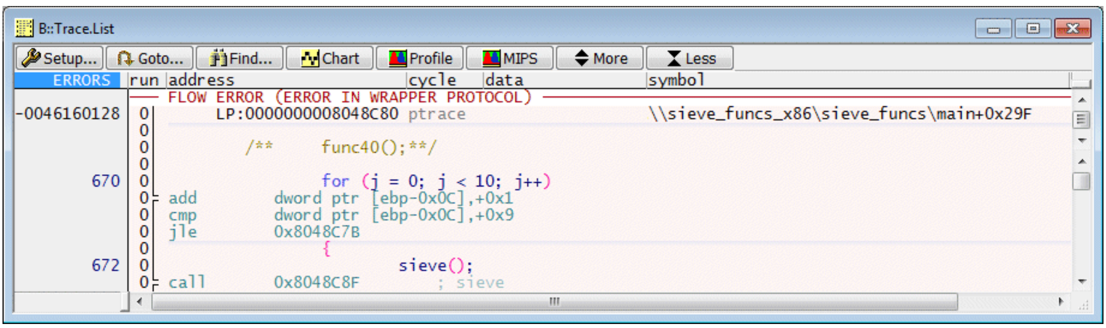
If the trace contains ERRORS, please try to set up a proper trace recording before you start to evaluate or analyze the trace contents.

ERRORS can be caused by the following:

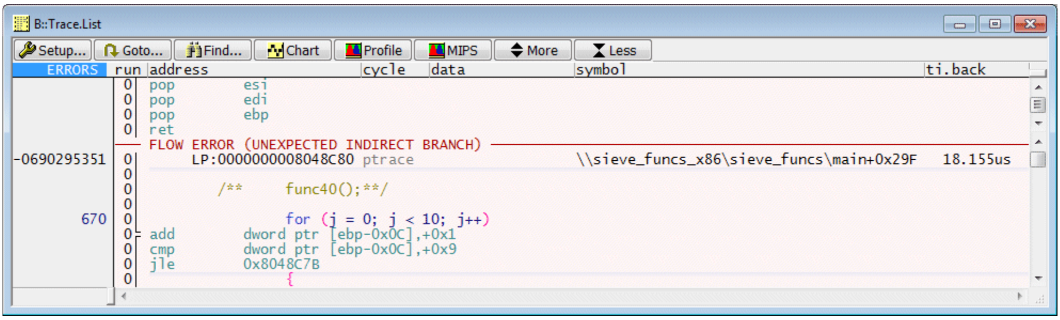
- TRACE32 detected an invalid trace packet. TRACE32 additionally displays the error indicator HARDERROR, if it is likely that the error was caused by pin problems.



- TRACE32 could not decode the packet.



- The trace information is not consistent with the program code in the target memory.

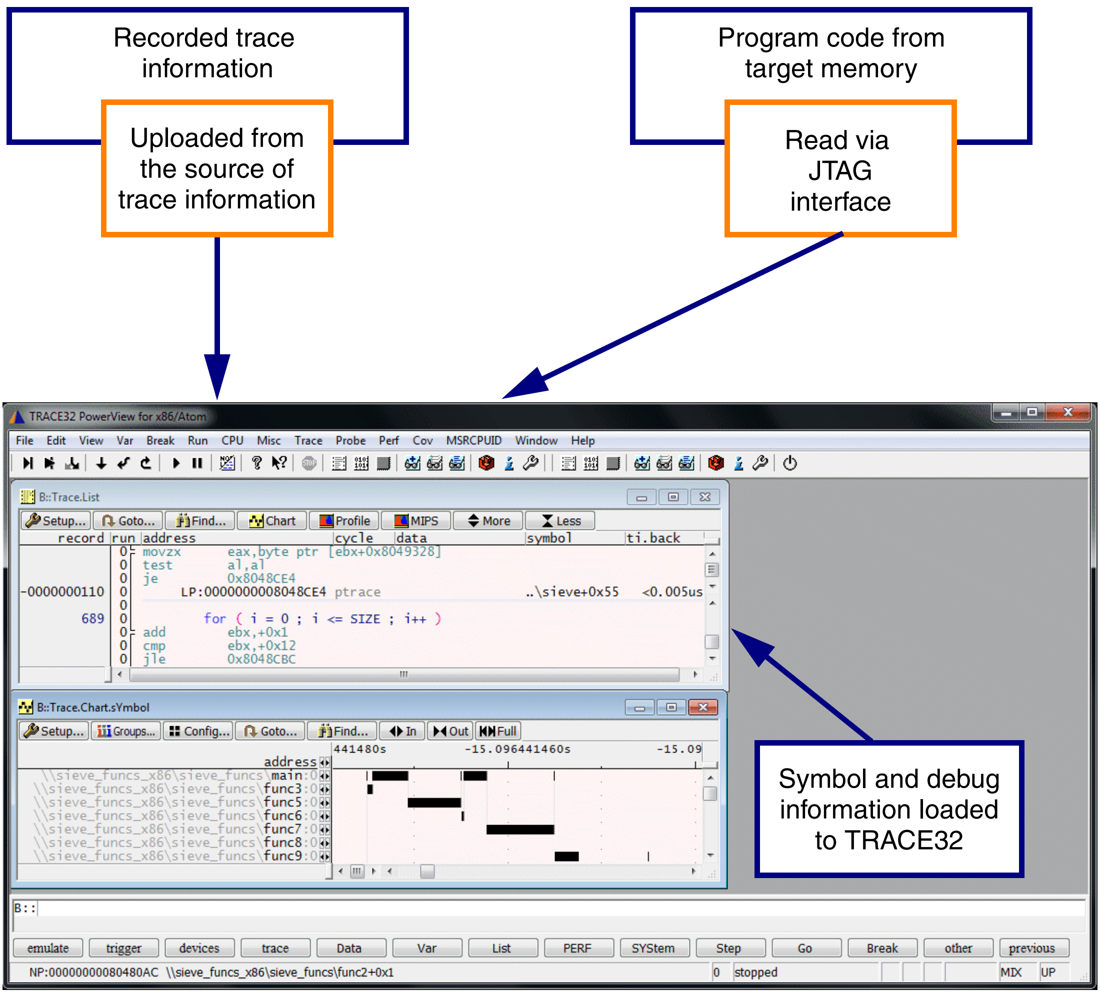


Background: In order to provide an intuitive trace display the following sources of information are merged:

- The trace information recorded.

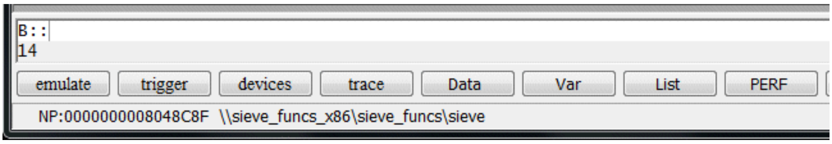
- The program code from the target memory read via the JTAG interface.

- The symbol and debug information already loaded to TRACE32.

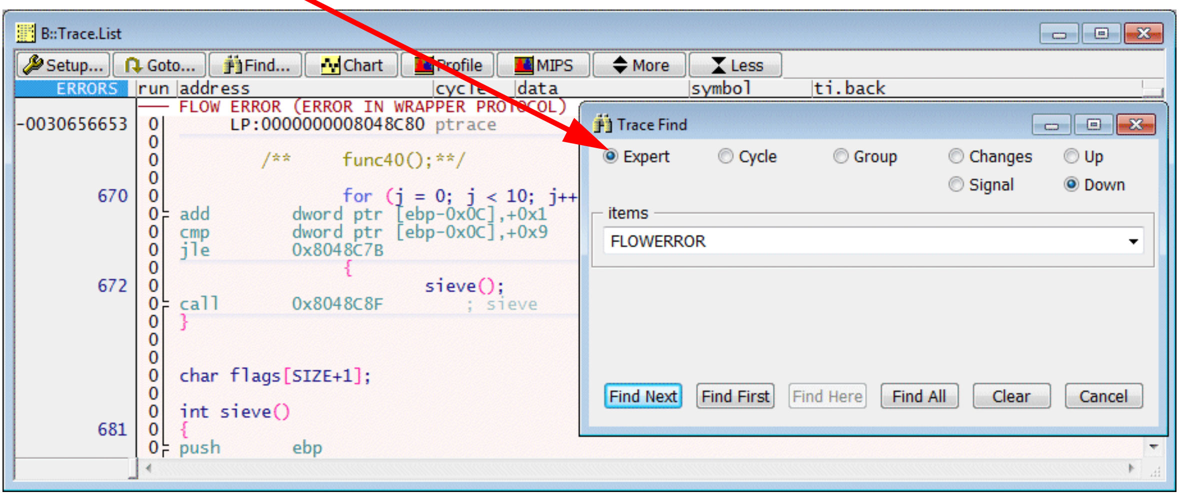


The TRACE32 function **Trace.FLOW.ERROR()** returns the number of ERRORS as a hex. number.

```
PRINT %Decimal Trace.FLOW.ERRORS()     ; display the number of ERRORS
                                       ; as a decimal number in the
                                       ; TRACE32 PowerView Message Line
```



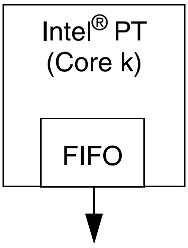
To find ERRORS in the trace use the keyword FLOWERROR on the Expert page of the **Trace Find** dialog.



Trace.Find FLOWERROR

TARGET FIFO OVERFLOW

Inside each Intel® PT generation module trace packets are queued to a FIFO buffer in order to send them out to the STM/SDRAM.



If trace packets are generated faster than can be sent out, the FIFO buffer can overflow and trace packets are lost.

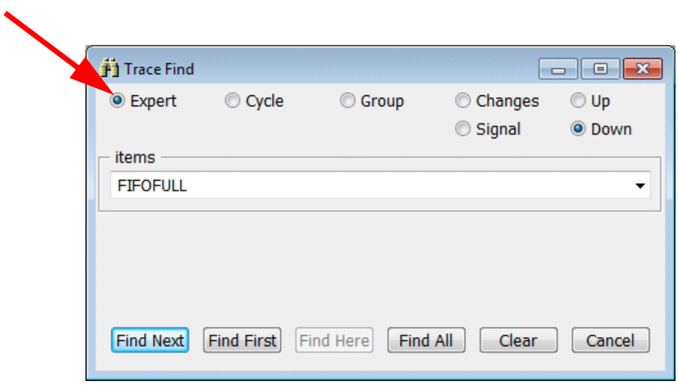
The affected Intel® PT generates a Buffer Overflow packet (FUP.OVF) to indicate that its FIFO is full and trace packets are no longer generated.

A Asynchronous Flow Update packet, that provides the address of the next instruction that will be executed, is generated to indicate that the packet generation now continues.

The TRACE32 function **Trace.FLOW.FIFOFULL()** returns the number of TARGET FIFO OVERFLOWs as a hex. number.

```
PRINT %Decimal Trace.FLOW.FIFOFULL()        ; display the number of TARGET
                                            ; FIFO OVERFLOWs as a decimal
                                            ; number in the TRACE32
                                            ; PowerView Message Line
```

To find TARGET FIFO OVERFLOWs in the trace use the keyword FIFOFULL on the Expert page in the **Trace Find** dialog.



Trace.Find FIFOFULL

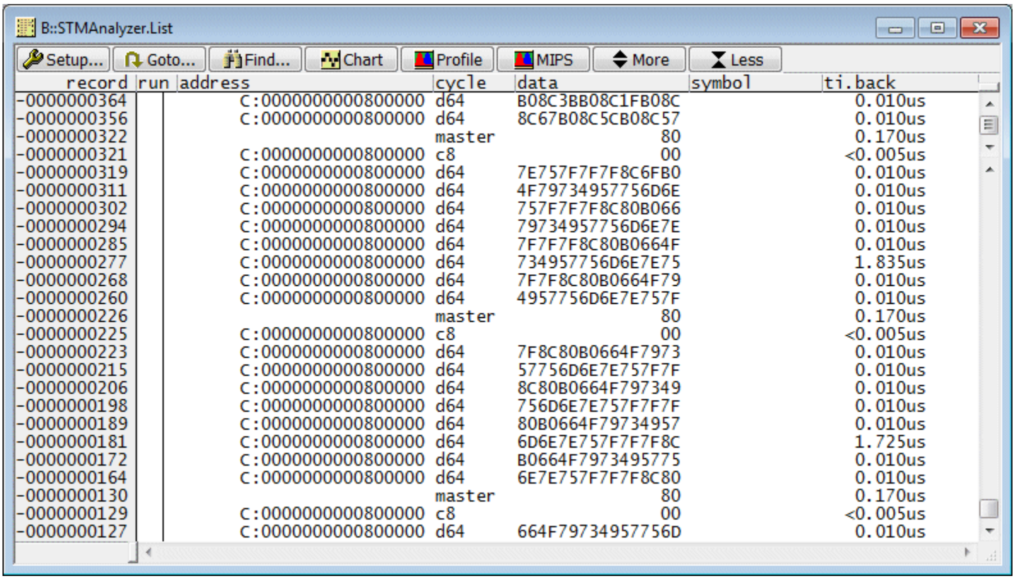
TARGET FIFO OVERFLOWs are strictly speaking not errors. They can occur in normal operation.

Since gaps in the instruction execution sequence are likely to disturb the nesting trace analyses, TRACE32 explicitly points them out.

SystemTrace

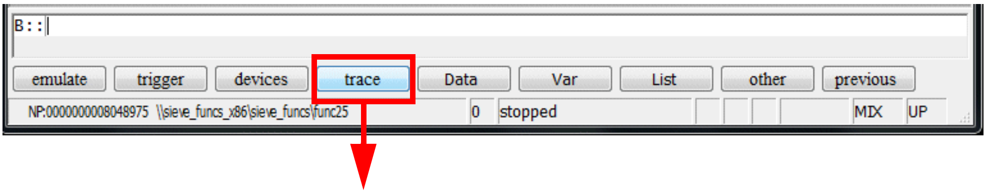
Depending on where the STP packets are stored, the following TRACE32 command groups can be used to analyze and display these packets:

- **STMAnalyzer.**<*sub_cmd*>, if the STP packets are stored in the trace memory provided by POWER TRACE II / POWER TRACE III.

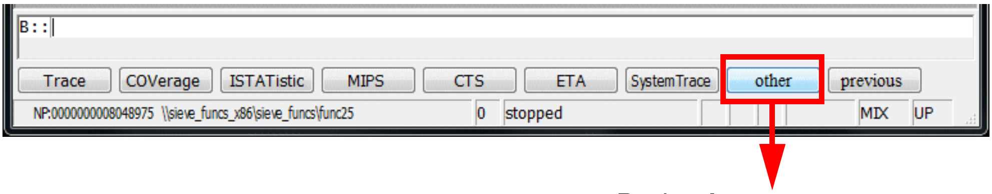


- **STMLA.**<*sub_cmd*>, if the STP packets were recorded without a TRACE32 trace tool, and if they were loaded to TRACE32 PowerView for analysis.

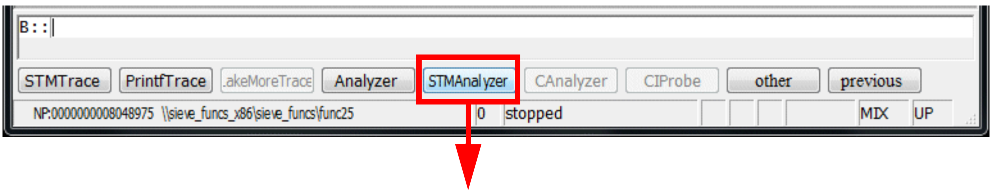
The command groups usable in your current configuration can be get from the TRACE32 PowerView Softkey line.



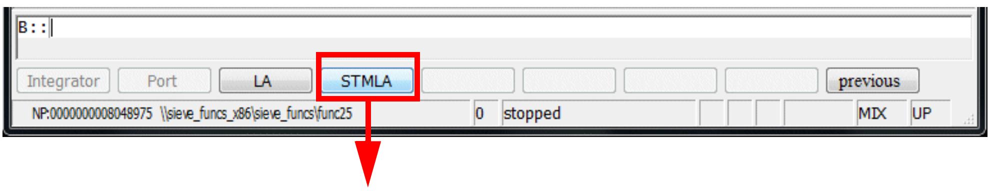
Push **trace** to get access to all command groups that analyze trace information.



Push **other** to see more command groups.



POWER TRACE II / POWER TRACE III is used in the current configuration, so the command group STMAnalyzer is enabled.



The command group STMLA is always enabled.

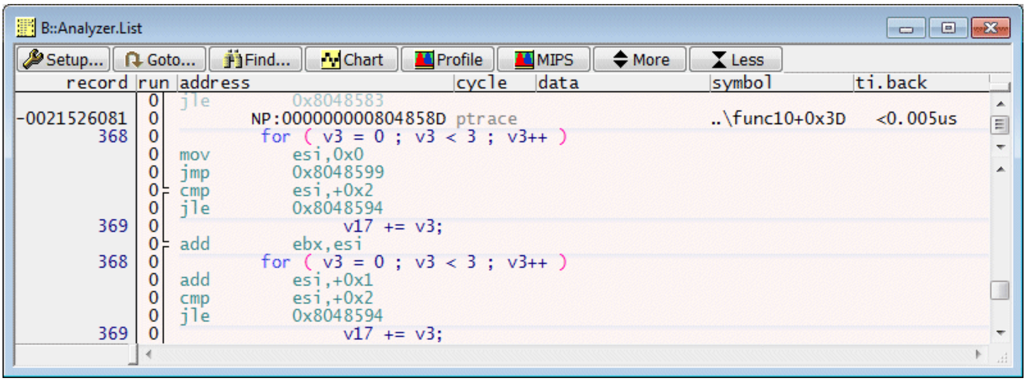
TRACE32 PowerView offers the following abstraction, since most *<sub_cmd>* are identical for all command groups:

SystemTrace.METHOD Analyzer | LA

```
SystemTrace.METHOD Analyzer            ; inform TRACE32 PowerView that the
                                       ; STP packets are stored in POWER
                                       ; TRACE II

SystemTrace.List                       ; List STP packet stream
```

Depending on where the trace packets are stored, the following TRACE32 command groups can be used to analyze and display the core trace information:

- **Analyzer.**<*sub_cmd*>, if the STP packets are stored in the trace memory provided by POWER TRACE II / POWER TRACE III.



- **Onchip.**<*sub_cmd*>, if the Intel® PT trace packets are stored in the target SDRAM.
- **LA.**<*sub_cmd*>, if the trace packets were recorded without a TRACE32 trace tool, and if they were loaded to TRACE32 PowerView for analysis.

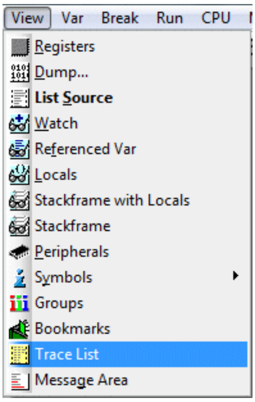
TRACE32 PowerView offers the following abstraction, since most <*sub_cmd*> are identical for all command groups:

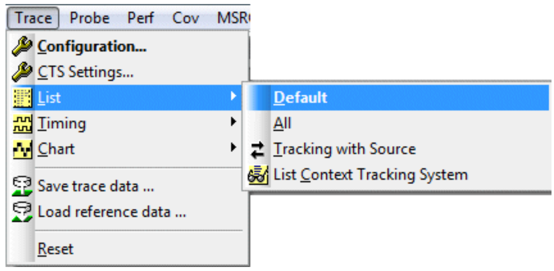
Trace.METHOD Analyzer | Onchip | LA

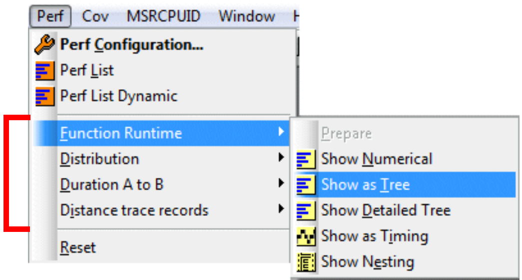
```
Trace.METHOD Analyzer                  ; inform TRACE32 PowerView that the
                                       ; trace packets are stored in POWER
                                       ; TRACE II

Trace.List                             ; List core trace information
```

Selecting the trace METHOD has the following additional consequences:

All **Trace.**<*sub_cmd*> commands offered in the TRACE32 PowerView menu apply to the selected trace METHOD.







TRACE32 is advised to use the trace information from the trace specified by METHOD as source for the trace evaluations of the following command groups:

COVerage.<*sub_cmd*>	Trace-based code coverage
ISTAT.<*sub_cmd*>	Detailed instruction analysis
MIPS.<*sub_cmd*>	MIPS analysis

Influencing Factors on the Trace Information

The main influencing factor on the trace information is the **Intel® PT**. It specifies what type of trace information is generated for the user.

Basics about the trace messages are described in **“Protocol Description”**, page 6.

Advanced setting can be found in **“Trace Control by Filters”**, page 68.

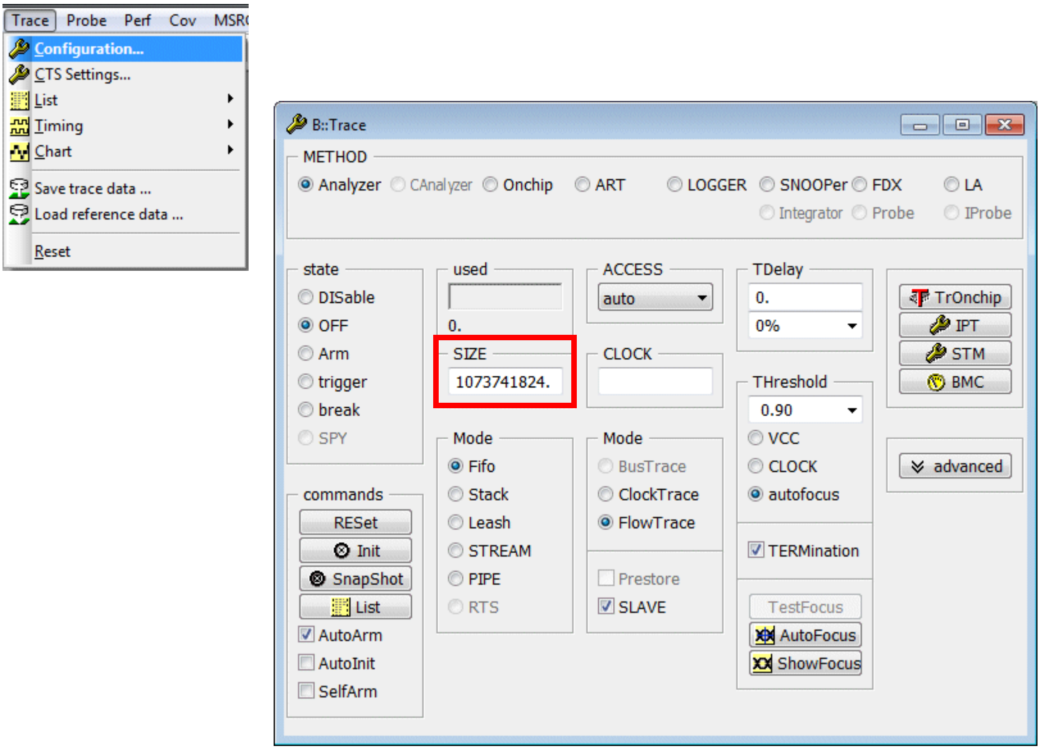
Another important influencing factor are the settings in the **TRACE32 Trace Configuration** window. They specify how much trace information can be recorded and when the trace recording is stopped.

Settings in the TRACE32 Trace Configuration Window

The Mode settings in the Trace Configuration window specify how much trace information can be recorded and when the trace recording is stopped.

The following modes are provided, if the **Trace.METHOD Analyzer** is selected:

- **Fifo, Stack, Leash Mode:** allow to record as much trace records as indicated in the **SIZE** field of the Trace Configuration window.



- **STREAM Mode:** STREAM mode specifies that the trace information is immediately streamed to a file on the host computer. STREAM mode allows a trace memory size of several T Frames.

- **PIPE Mode:** PIPE mode specifies that the trace information is immediately streamed to a named pipe on the host computer.

  PIPE mode creates the path to convey trace raw data to an application outside of TRACE32 PowerView. The named pipe has to be created by the receiving application before TRACE32 can connect to it.

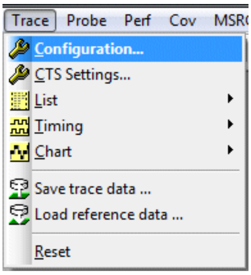
Trace.Mode PIPE	
Trace.PipeWrite *<pipe_name>*	Connect to named pipe
Trace.PipeWrite \\.\pipe*<pipe_name>*	Connect to named pipe (Windows)
Trace.PipeWrite	Disconnect from named pipe

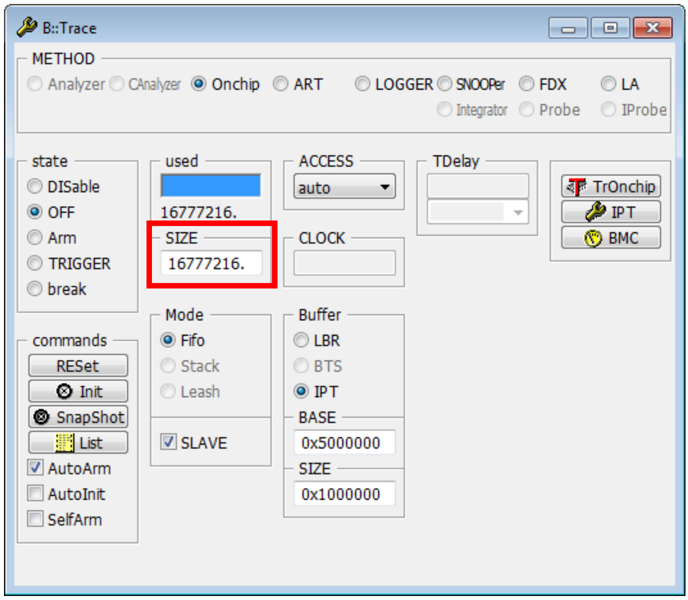
```
…

Trace.Mode PIPE                        ; switch trace to PIPE mode

Trace.PipeWRITE \\.\pipe\pproto00      ; connect to named pipe
                                       ; (Windows)

…

Trace.PipeWRITE                        ; disconnect from named pipe
```

STP packets (no timestamp) are conveyed in PIPE mode.

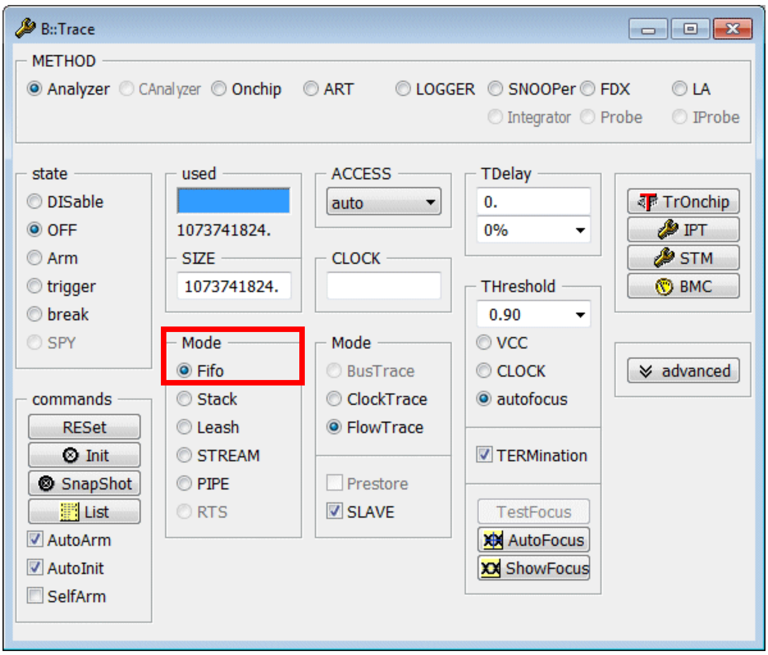
If the **Trace.METHOD Onchip** is selected only Fifo mode can be used:

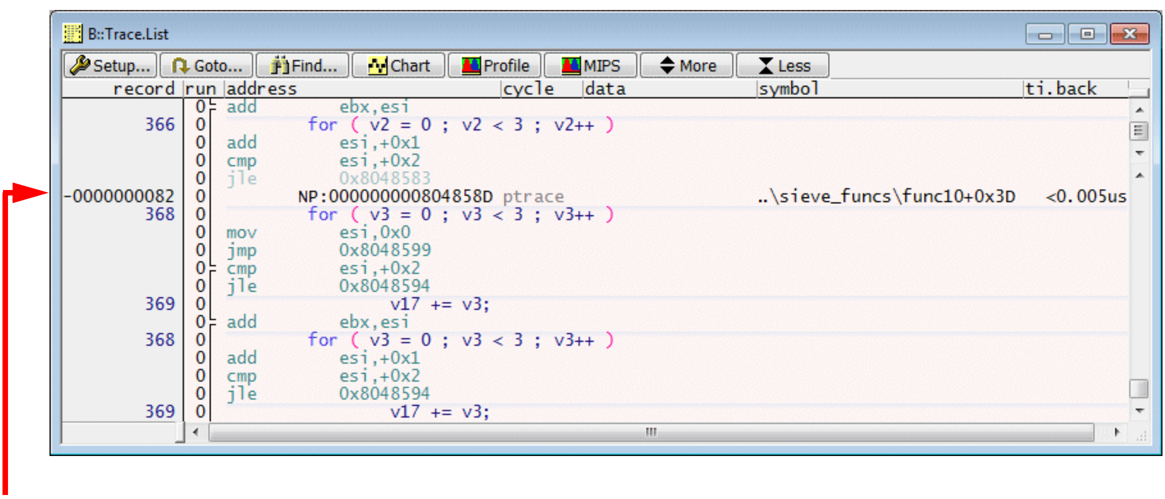
- **Fifo:** allows to record as much trace records as indicated in the **SIZE** field of the Trace Configuration window.





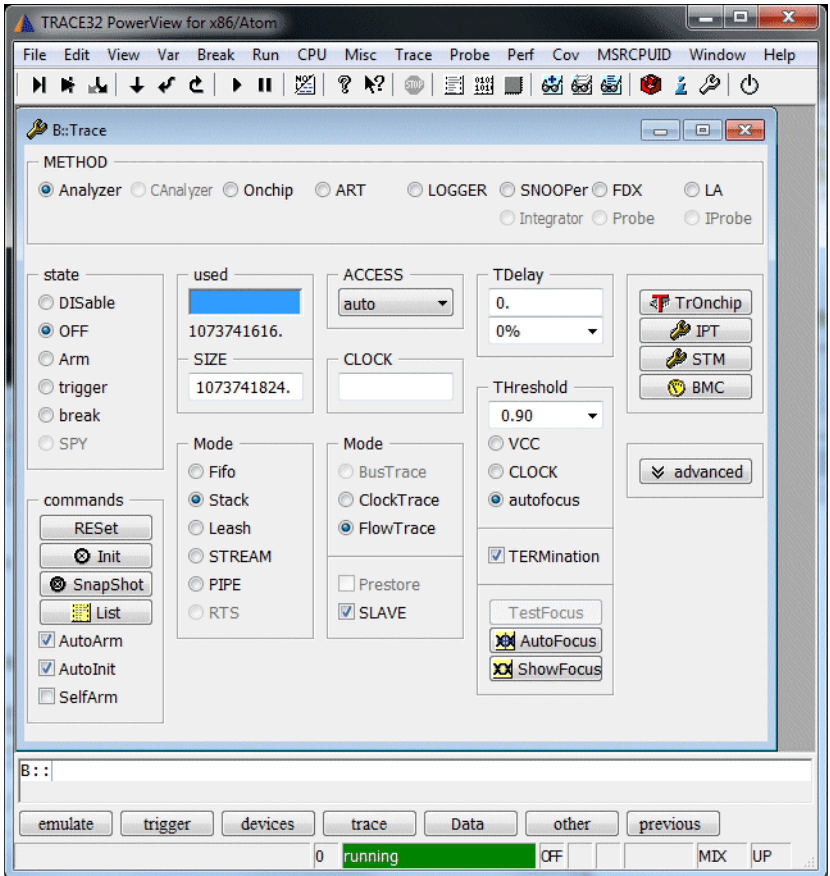
```
Trace.Mode Fifo                                 ; default mode

                                                ; when the trace memory is full
                                                ; the newest trace information will
                                                ; overwrite the oldest one

                                                ; the trace memory contains all
                                                ; information generated until the
                                                ; program execution stopped
```





In **Fifo** mode negative record numbers are used. The last record gets the smallest negative number.

```
Trace.Mode Stack                              ; when the trace memory is full
                                              ; the trace recording is stopped

                                              ; the trace memory contains all
                                              ; information generated directly
                                              ; after the start of the program
                                              ; execution
```

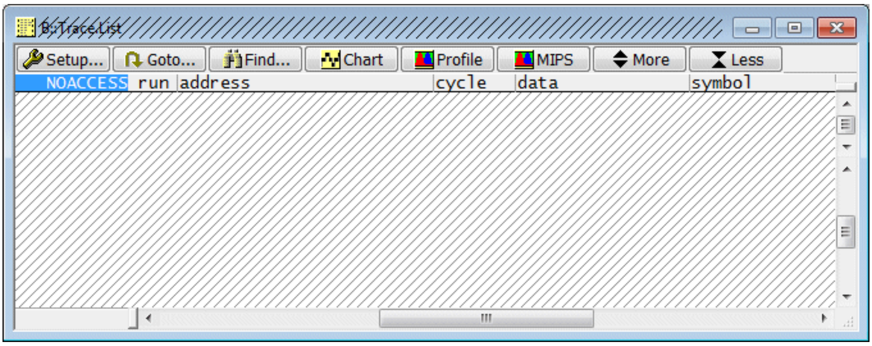


The trace recording is stopped as soon as the trace memory is full (OFF state)

Green **running** in the Debug State Field indicates that program execution is running

OFF in the Trace State Field indicates that the trace recording is switched off

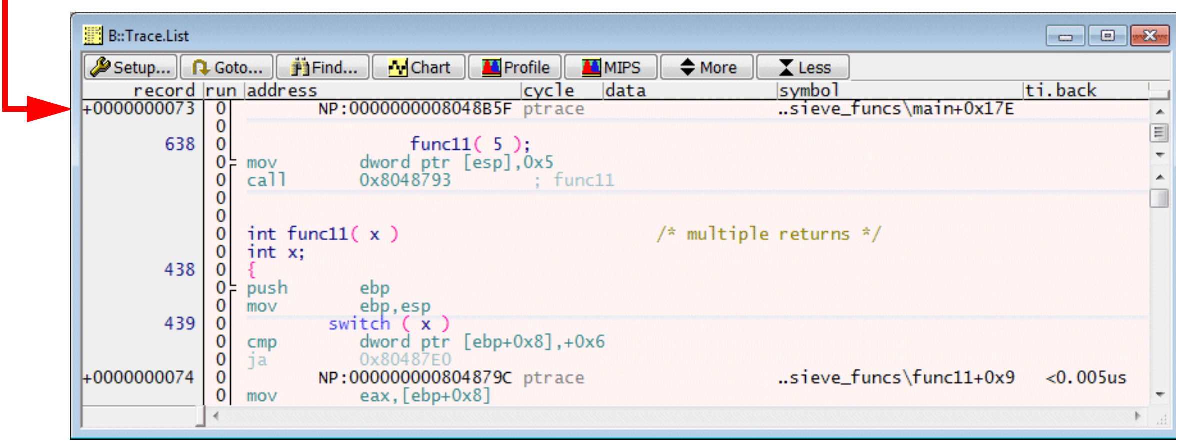
TRACE32 needs to read the program code from the target memory in order to display the core trace information. This is not possible while the program execution is running. This is the reason why the **Trace.List** window indicates **NOACCESS**.



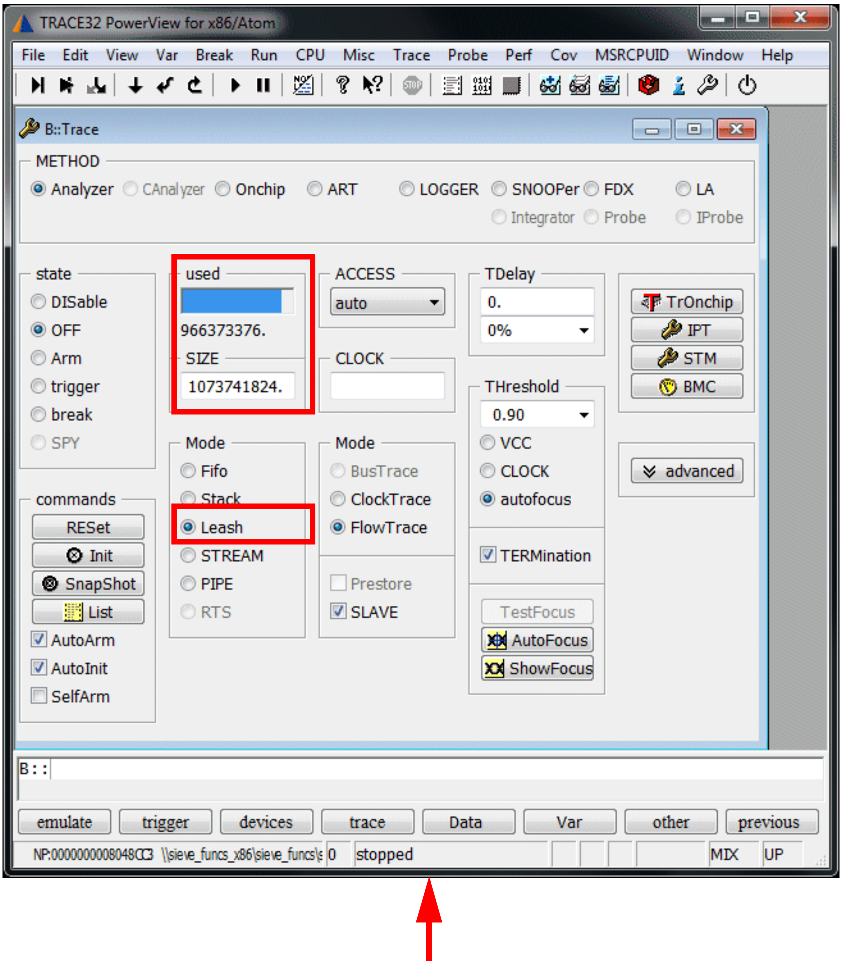
Stop the program execution to allow TRACE32 to read the program code from the target. Or if you need to display the core trace information while the program execution is running, load a copy of the program code to the **TRACE32 Virtual Memory**.

Data.LOAD.Elf *<file>* **/PlusVM**	Load the program code to the target and to the TRACE32 Virtual Memory.

Since the trace recording starts with the program execution and stops, when the trace memory is full, positive record numbers are used in **Stack** mode. The first record in the trace gets the smallest positive number.



```
Trace.Mode Leash                          ; when the trace memory is nearly
                                          ; full the program execution is
                                          ; stopped

                                          ; Leash mode uses the same record
                                          ; numbering scheme as Stack mode
```



The program execution is **stopped** as soon as the trace buffer is nearly full.

Since stopping the program execution when the trace buffer is nearly full requires some logic/time, **used** is smaller then the maximum **SIZE.**

```
Trace.Mode STREAM                          ; stream the recorded trace
                                           ; information to a file on the host
                                           ; computer

                                           ; STREAM mode uses the same record
                                           ; numbering scheme as Stack mode
```

The trace information is immediately streamed to a file on the host computer after it was placed into the trace memory. This procedure extends the size of the trace memory to several T Frames.

- STREAM mode requires a 64-bit host computer and a 64-bit TRACE32 executable to handle the large trace record numbers.

By default the streaming file is placed into the TRACE32 temp. directory (**OS.PresentTemporaryDirectory()**).

The command **Trace.STREAMFILE** *<file>* allows to specify a different name and location for the streaming file.

```
Trace.STREAMFILE d:\temp\mystream.t32      ; specify the location for
                                           ; your streaming file
```

TRACE32 stops the streaming when less then 1 GByte free memory is left on the drive by default.

The command **Trace.STREAMFileLimit** *<+/- limit in bytes>* allows a user-defined free memory limitation.

```
Trace.STREAMFileLimit 5000000000.          ; streaming file is limited to
                                           ; 5 GByte

Trace.STREAMFileLimit -5000000000.         ; streaming is stopped when less
                                           ; the 5 GByte free memory is left
                                           ; on the drive
```

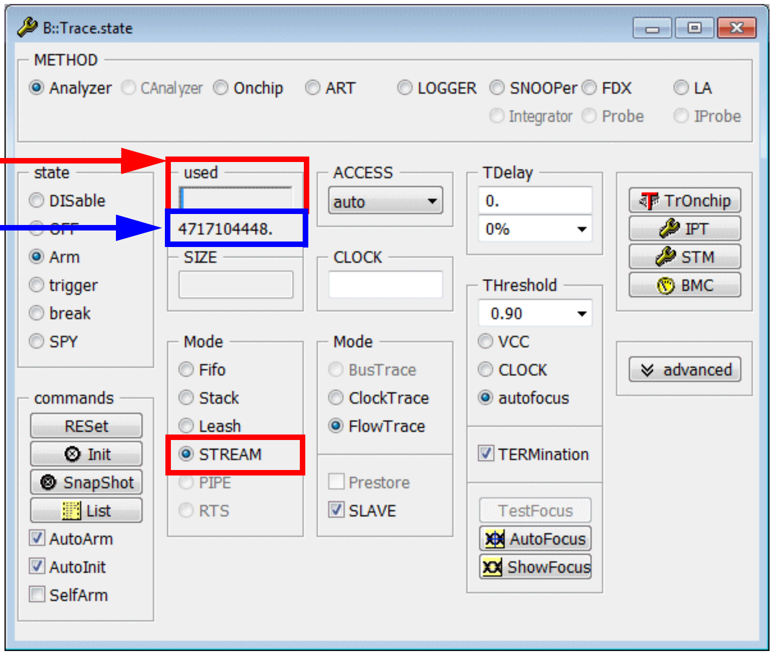
Please be aware that the streaming file is deleted as soon as you de-select the STREAM mode or when you exit TRACE32.

At high data rates your host computer might have problems saving the trace data to the streaming file. The command **Trace.STREAMCompression** allow to configure a better compression.

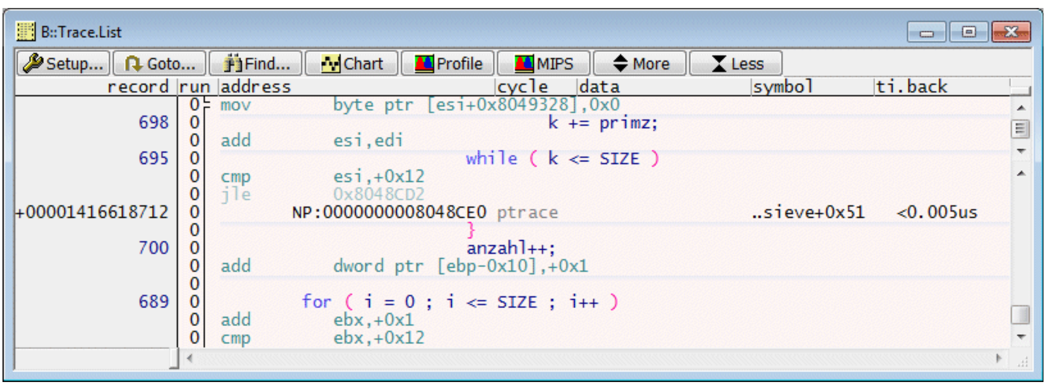
```
Trace.STREAMCompression HIGH
```

In STREAM mode the **used** field is split:

Number of records buffered by the trace memory of POWER TRACE II / POWER TRACE III



Number of records saved to streaming file



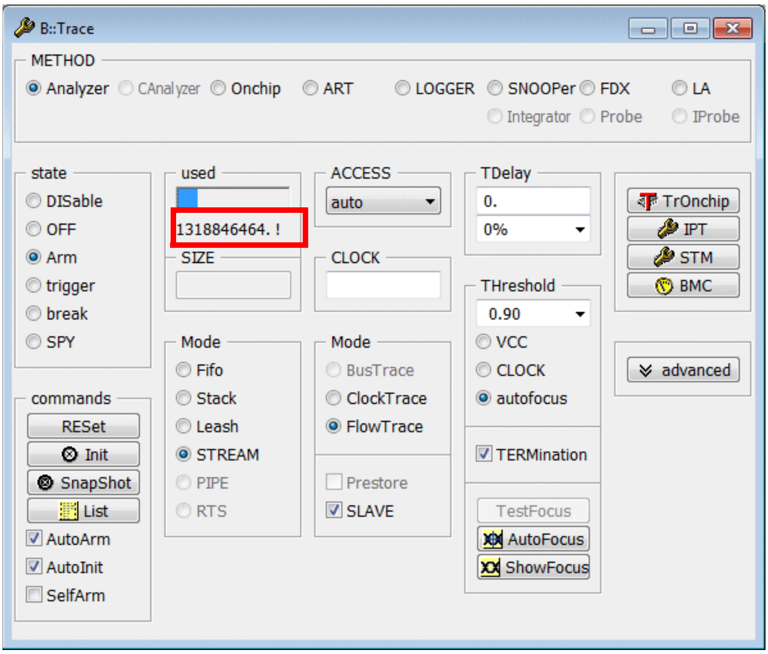
STREAM mode can generate very large record numbers

STREAM mode can only be used if the average data rate at the trace port does not exceed the maximum transmission rate of the host interface in use. Peak loads at the trace port are intercepted by the memory in POWER TRACE II / POWER TRACE III, which can be considered to be operating as a large FIFO.

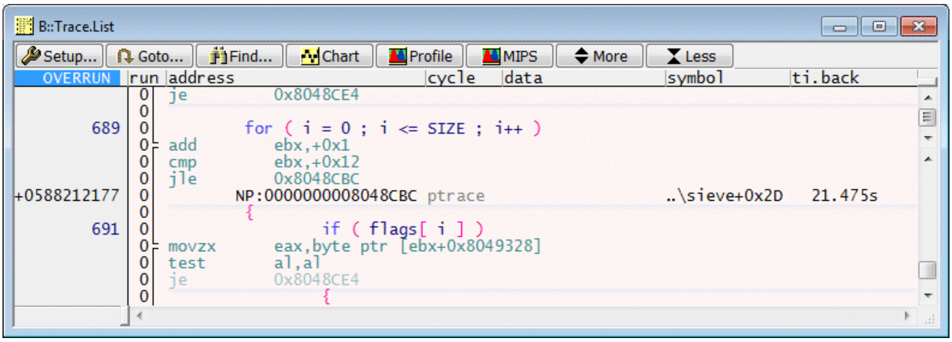
If the average data rate at the trace port exceeds the maximum transmission rate of the host interface in use, a **PowerTrace FIFO Overrun** occurs. TRACE32 stops streaming and empties the POWER TRACE II / POWER TRACE III FIFO. Streaming is re-started after the POWER TRACE II / POWER TRACE III FIFO is empty.

A **PowerTrace FIFO Overrun** is indicated as follows:

1. A **!** in the **used** area of the Trace Configuration window indicates an overrun of the POWER TRACE II / POWER TRACE III FIFO.



2. The OVERRUN is indicated in all trace display windows.



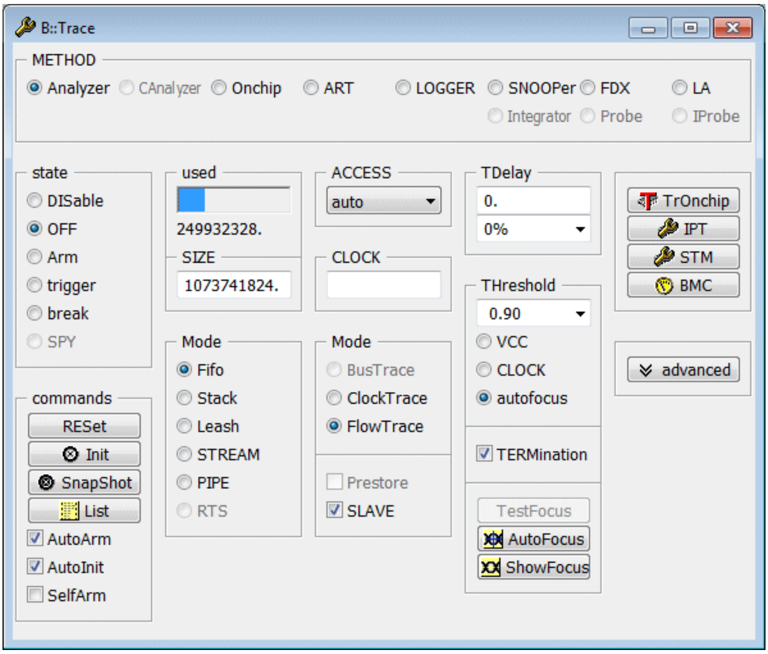
OVERRUNs are not visible at record level.

A large **ti.back** value (**tool timestamp** only) can be considered as an OVERRUN indicator.

```
Trace.FindAll TIme.Back 10.s--50.s     ; find all trace records with
                                       ; a timestamp between 10.s and
                                       ; 50.s
```

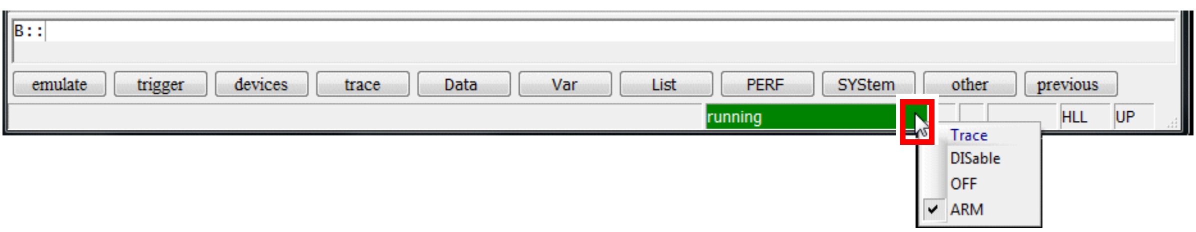
States of the Trace

The trace buffer can either sample or allows the read-out for information display.



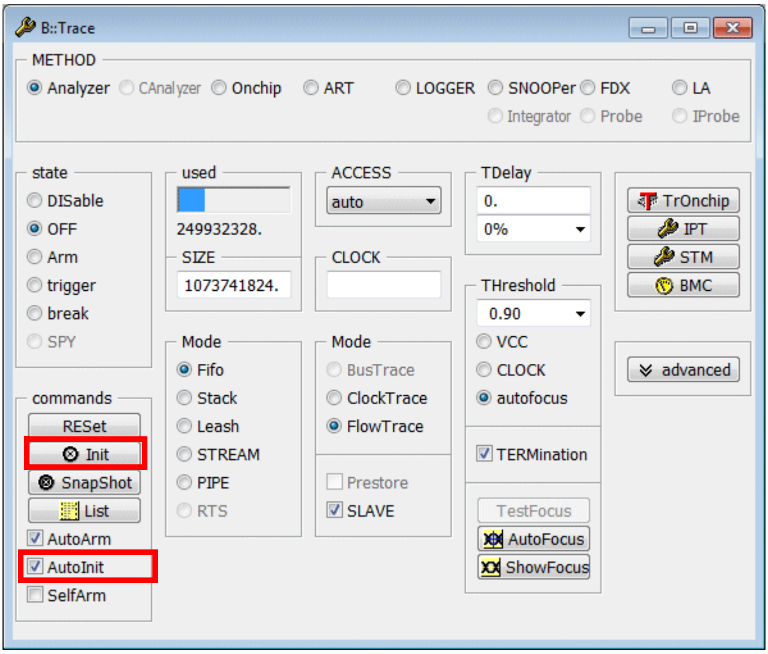
States of the Trace	
DISable	The trace is disabled.
OFF	The trace is not sampling. The trace contents can be analyzed and displayed.
Arm	The trace is sampling. There is no access to the trace contents.

The current state of the trace is always indicated in the **Trace State** field of the TRACE32 state line.



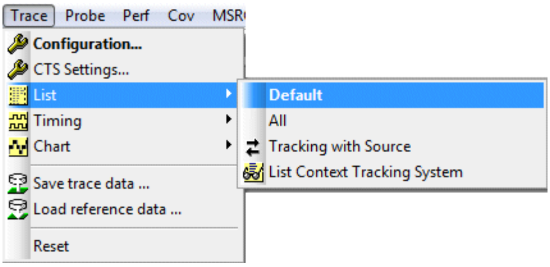
Since Intel® PT does not provide a mean to indicate a trigger, the Trace states **trigger** and **break** are never reached.

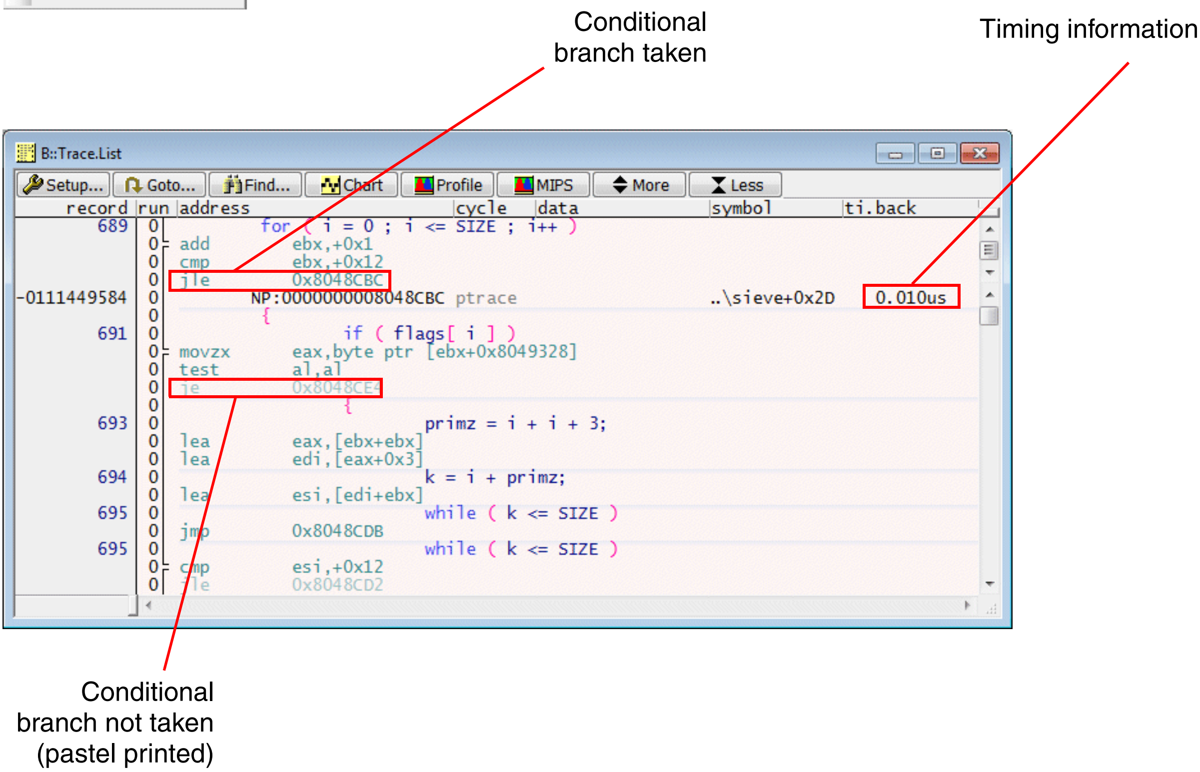
The AutoInit Command



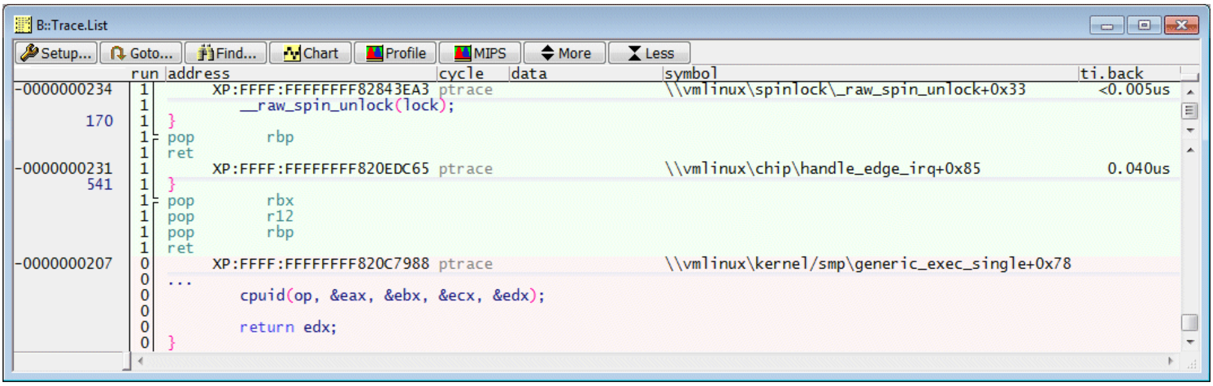
Init Button	Clear the trace memory. All other settings in the Trace configuration window remain valid.
AutoInit CheckBox	ON: The trace memory is cleared whenever the program execution is started (Go, Step).

Default Listing

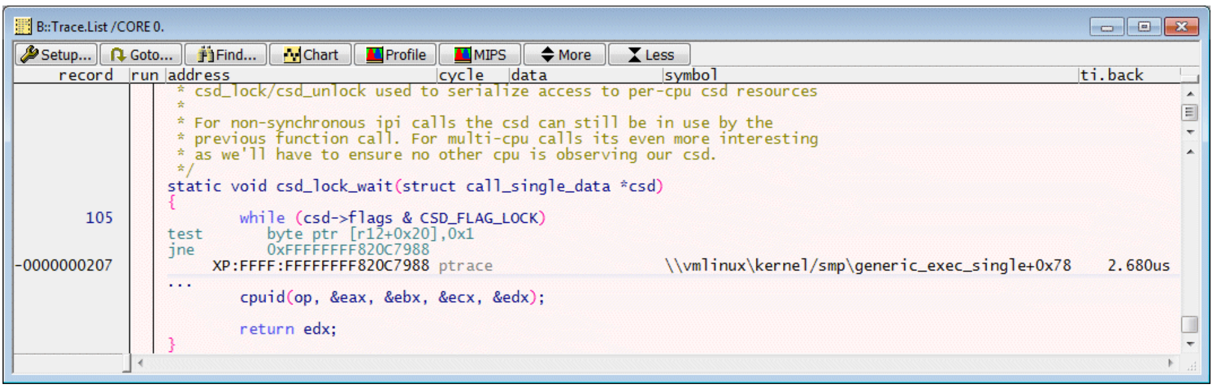


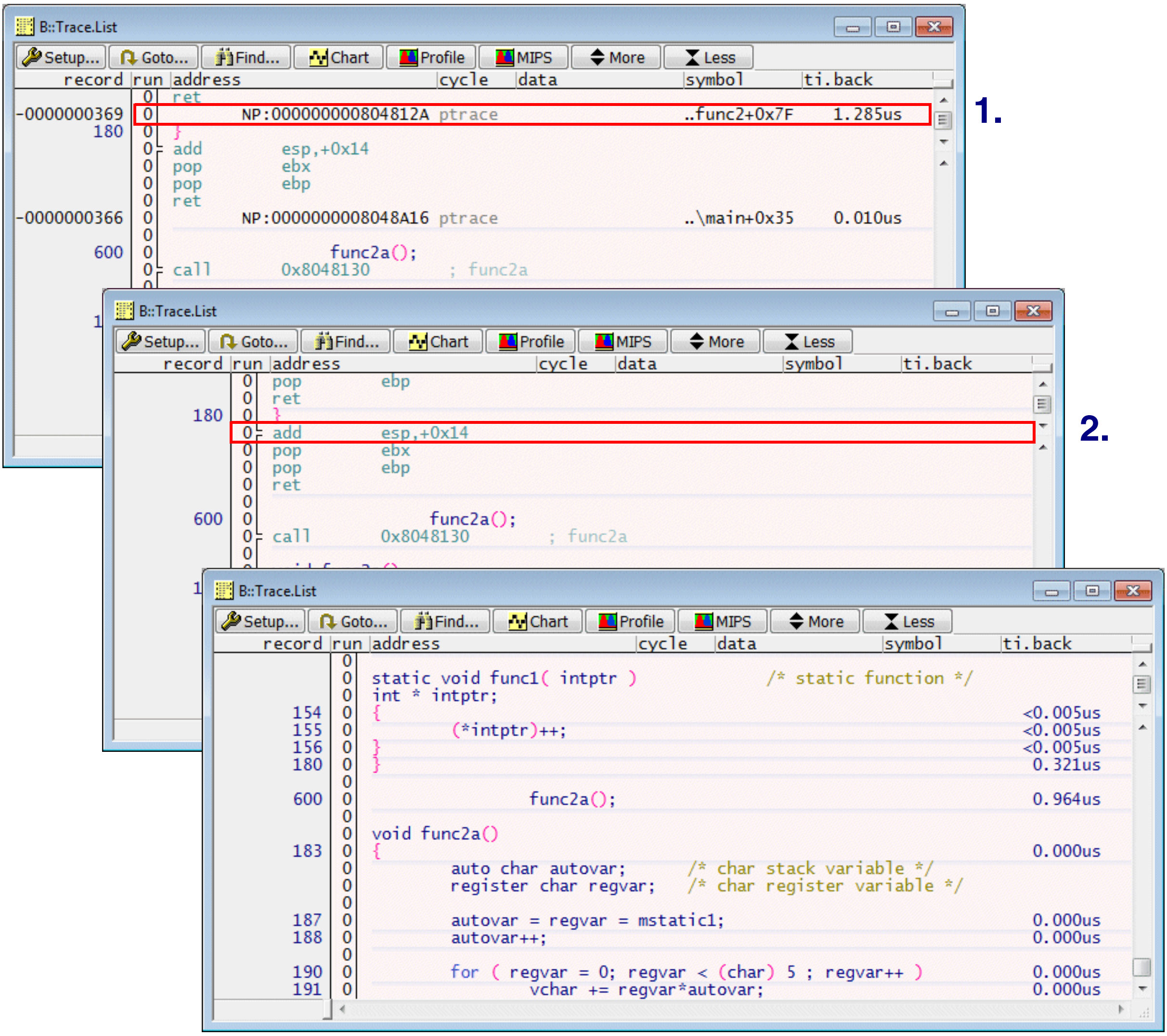


The trace information for all cores is displayed by default in the **Trace.List** window. The column run and the coloring of the trace information are used for core indication.



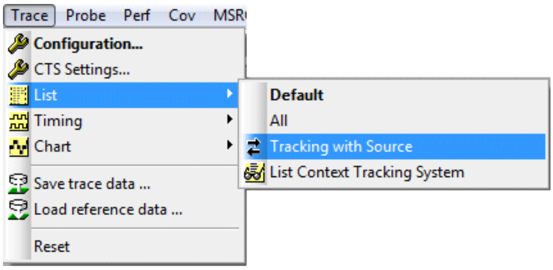
Trace.List /CORE *<n>* The option CORE allows a per core display of the trace information.

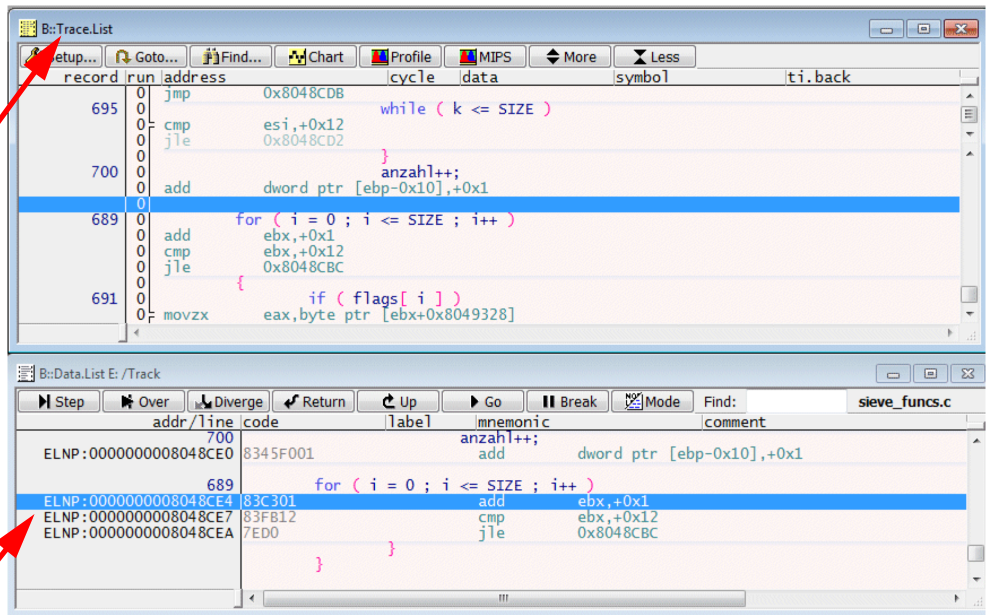




1. time Less	Suppress the display of the ptrace packets.
2. time Less	Suppress the display of the assembly code.

The ***More*** button works vice versa.

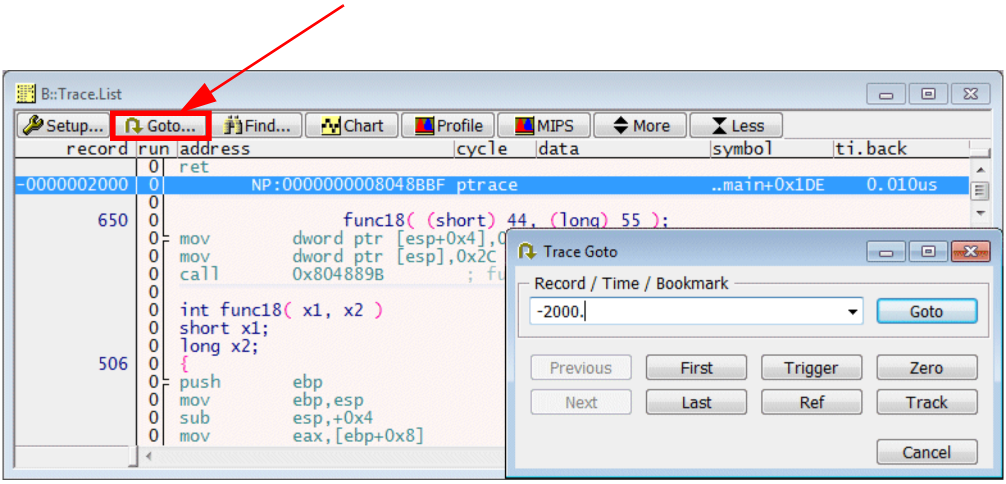




Active Window

All windows opened with the ***/Track*** option follow the cursor movements in the active window

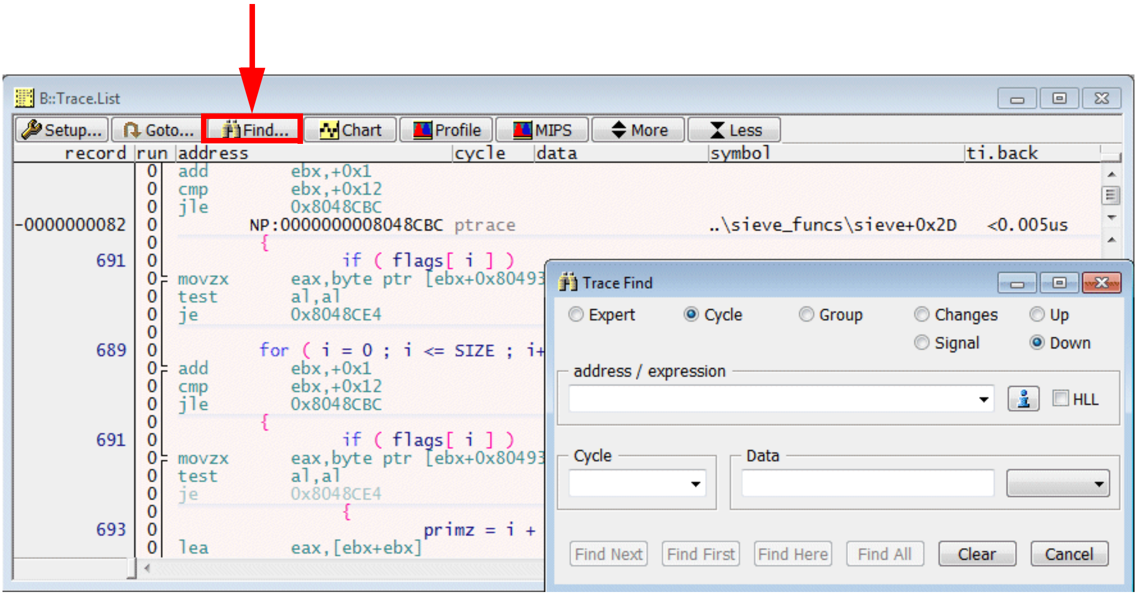
Tracking between the Trace Listing and the Source Listing is based on the program addresses.



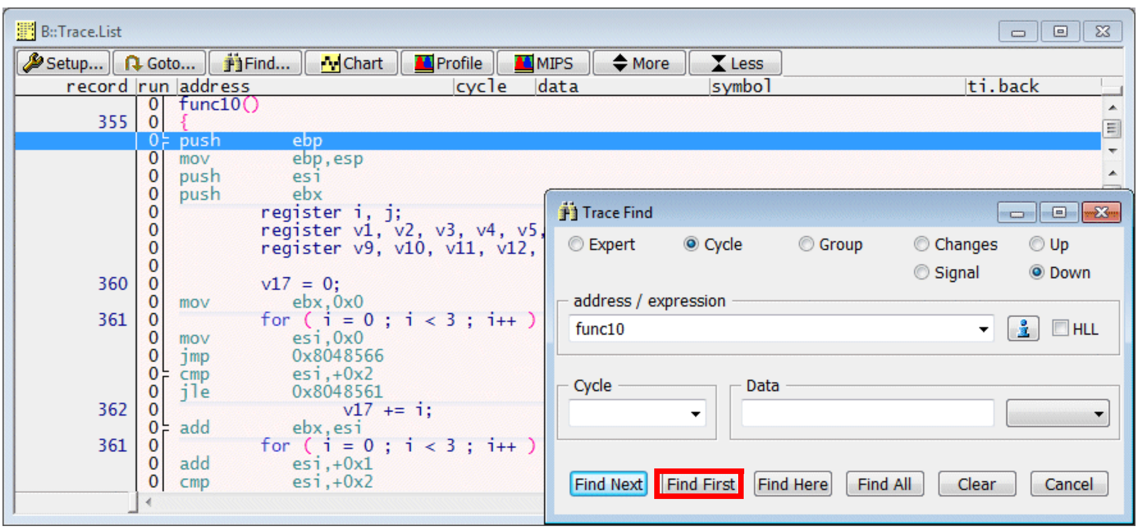
Pg ↑	Scroll page up.
Pg ↓	Scroll page down.
Ctrl - Pg ↑	Go to the first record sampled in the trace buffer.
Ctrl - Pg ↓	Go to the last record sampled in the trace buffer.

Find a Specific Record

The **Trace.List** window provides a “**Find…**” button to open the **Trace Find** dialog. The **Trace Find** dialog allows to search for events of interest in the trace.



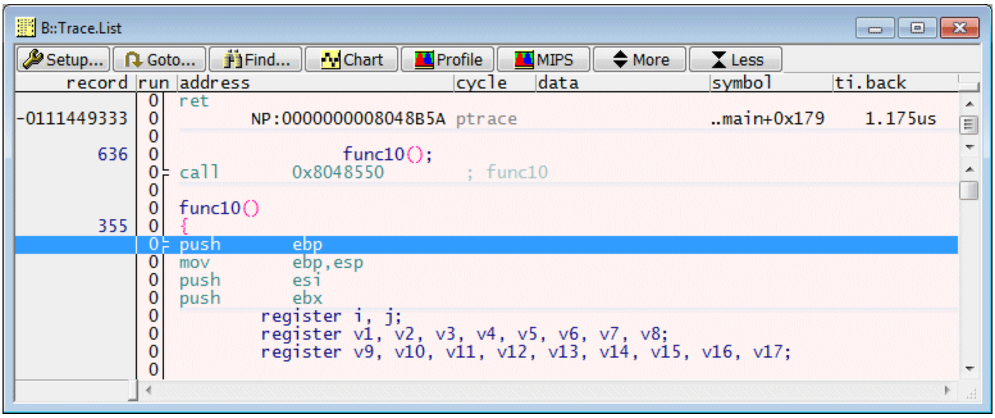
Example: Find the entry to the function func10.



A detailed description of the **Trace Find dialog** can be found in **“Application Note for Trace.Find”** (app_trace_find.pdf).

Display Items

Default Display Items



- **Column *record***

  Displays the record numbers

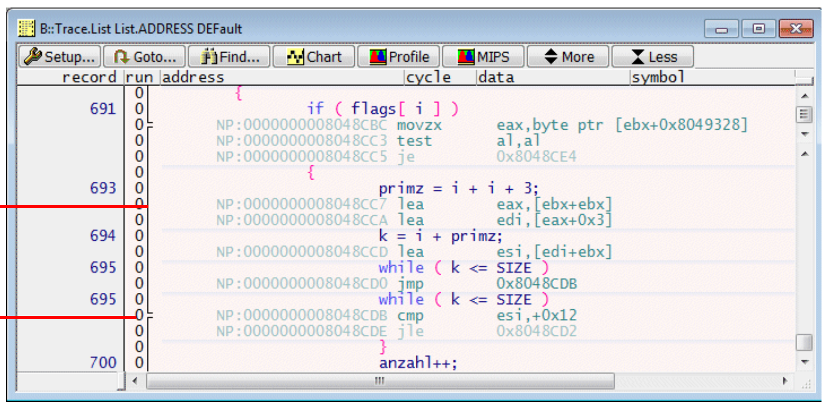
- **Column *run***

  The column run displays some graphic element to provide a quick overview on the instruction execution sequence.

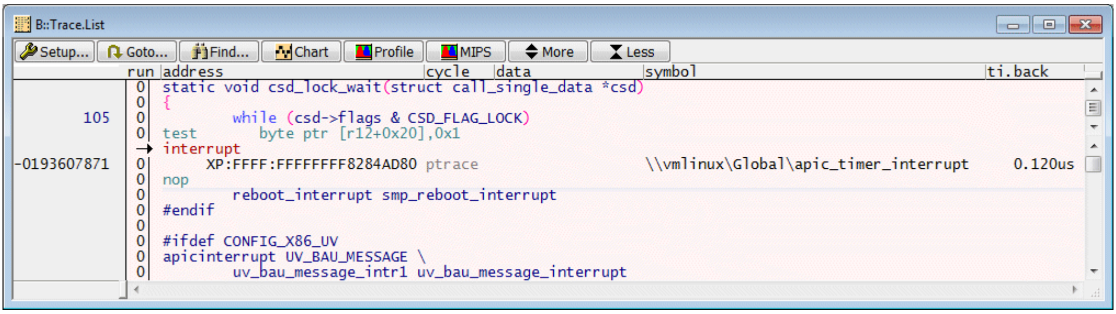
```
Trace.List List.ADDRESS DEFault
```

Sequential program execution (solid line)

Non-sequential program execution (broken line)



The column run also indicates Interrupts and TRAPs.

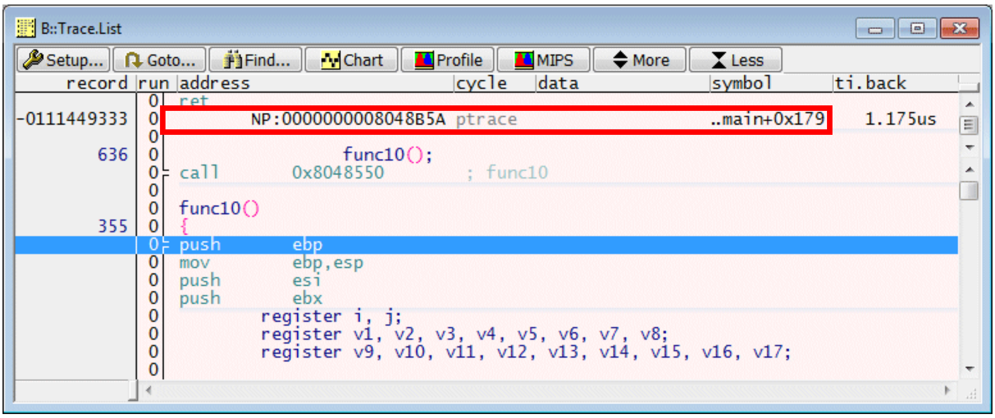


- **Column *cycle***

  The main cycle type is:

  - ptrace (program trace information)

- **Column *address/symbol***



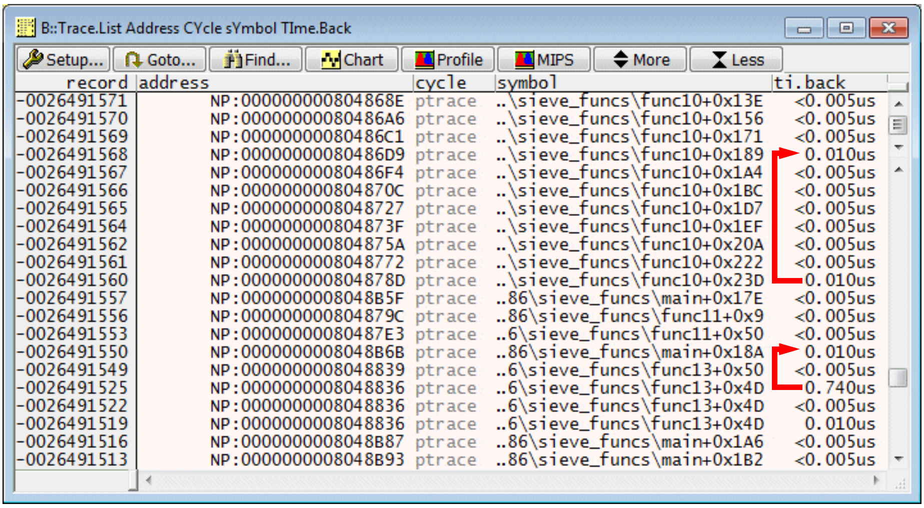
The **address column** shows the following information:
<access class>:<address>

Access Classes	
NP	Program address in 32-bit Protected Mode
XP	Program address in 64-bit mode

Information on the other available access classes can be found in **“Intel® x86/x64 Debugger”** (debugger_x86.pdf).

The **symbol column** shows the corresponding symbolic address.

- **Column *ti.back***



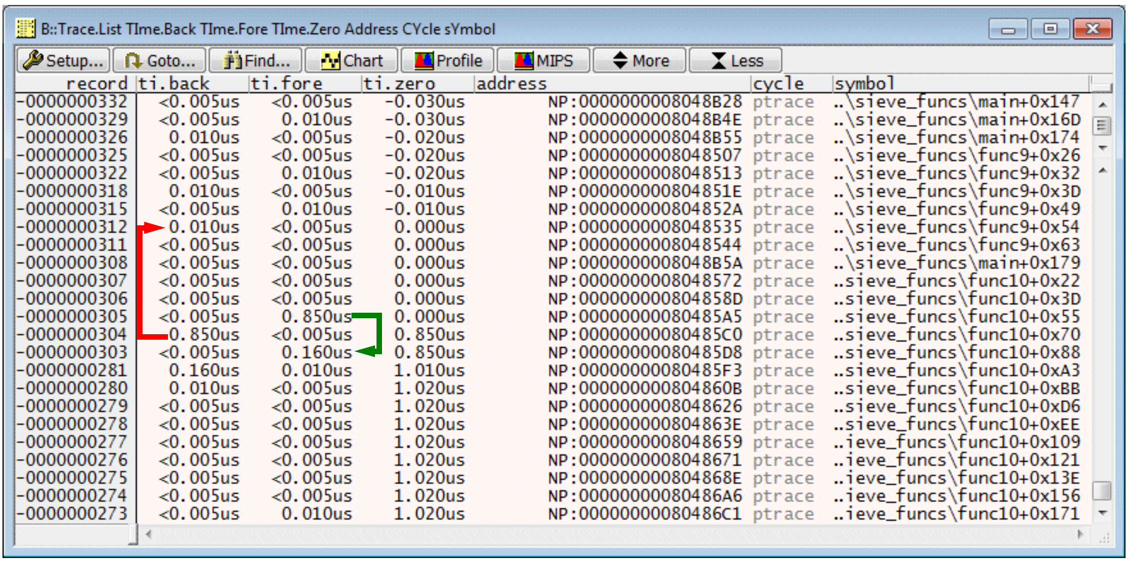
The **ti.back column** shows the time distance to the previous timestamped record.

For details on the TRACE32 tool timestamp refer to **“Tool Timestamp (POWER TRACE II / POWER TRACE III only)”**, page 8.

Time Information

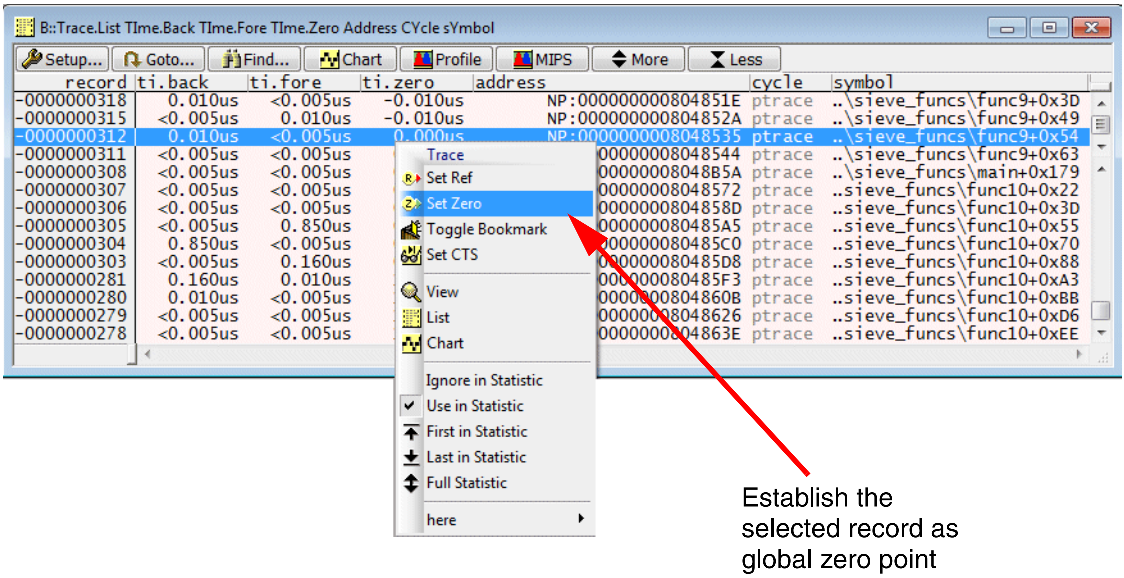
TIme.Back	Time relative to the previous record (red)
TIme.Fore	Time relative to the next record (green).
TIme.Zero	Time relative to the global zero point.

```
Trace.List TIme.Back TIme.Fore TIme.Zero Address CYcle sYmbol
```



record	ti.back	ti.fore	ti.zero	address	cycle	symbol
-0000000332	<0.005us	<0.005us	-0.030us	NP:00000000008048B28	ptrace	..\sieve_funcs\main+0x147
-0000000329	<0.005us	0.010us	-0.030us	NP:00000000008048B4E	ptrace	..\sieve_funcs\main+0x16D
-0000000326	0.010us	<0.005us	-0.020us	NP:00000000008048B55	ptrace	..\sieve_funcs\main+0x174
-0000000325	<0.005us	<0.005us	-0.020us	NP:00000000008048507	ptrace	..\sieve_funcs\func9+0x26
-0000000322	<0.005us	0.010us	-0.020us	NP:00000000008048513	ptrace	..\sieve_funcs\func9+0x32
-0000000318	0.010us	<0.005us	-0.010us	NP:0000000000804851E	ptrace	..\sieve_funcs\func9+0x3D
-0000000315	<0.005us	0.010us	-0.010us	NP:0000000000804852A	ptrace	..\sieve_funcs\func9+0x49
-0000000312	0.010us	<0.005us	0.000us	NP:00000000008048535	ptrace	..\sieve_funcs\func9+0x54
-0000000311	<0.005us	<0.005us	0.000us	NP:00000000008048544	ptrace	..\sieve_funcs\func9+0x63
-0000000308	<0.005us	<0.005us	0.000us	NP:00000000008048B5A	ptrace	..\sieve_funcs\main+0x179
-0000000307	<0.005us	<0.005us	0.000us	NP:00000000008048572	ptrace	..sieve_funcs\func10+0x22
-0000000306	<0.005us	<0.005us	0.000us	NP:0000000000804858D	ptrace	..sieve_funcs\func10+0x3D
-0000000305	<0.005us	0.850us	0.000us	NP:000000000080485A5	ptrace	..sieve_funcs\func10+0x55
-0000000304	0.850us	<0.005us	0.850us	NP:000000000080485C0	ptrace	..sieve_funcs\func10+0x70
-0000000303	<0.005us	0.160us	0.850us	NP:000000000080485D8	ptrace	..sieve_funcs\func10+0x88
-0000000281	0.160us	0.010us	1.010us	NP:000000000080485F3	ptrace	..sieve_funcs\func10+0xA3
-0000000280	0.010us	<0.005us	1.020us	NP:0000000000804860B	ptrace	..sieve_funcs\func10+0xBB
-0000000279	<0.005us	<0.005us	1.020us	NP:00000000008048626	ptrace	..sieve_funcs\func10+0xD6
-0000000278	<0.005us	<0.005us	1.020us	NP:0000000000804863E	ptrace	..sieve_funcs\func10+0xEE
-0000000277	<0.005us	<0.005us	1.020us	NP:00000000008048659	ptrace	..ieve_funcs\func10+0x109
-0000000276	<0.005us	<0.005us	1.020us	NP:00000000008048671	ptrace	..ieve_funcs\func10+0x121
-0000000275	<0.005us	<0.005us	1.020us	NP:0000000000804868E	ptrace	..ieve_funcs\func10+0x13E
-0000000274	<0.005us	<0.005us	1.020us	NP:000000000080486A6	ptrace	..ieve_funcs\func10+0x156
-0000000273	<0.005us	0.010us	1.020us	NP:000000000080486C1	ptrace	..ieve_funcs\func10+0x171

Set the Global Zero Point (PowerTrace II only)



ZERO.RESet (tool timestamp only)	TIme.Zero is the zero point of the timestamp counter commonly used by all TRACE32 hardware modules.
ZERO.offset *<time>*	TIme.Zero is the zero point of the timestamp counter commonly used by all TRACE32 hardware modules minus the specified *<time>*.

```
PRINT Trace.RECORD.TIME(-99.)                ; print the timestamp of
                                             ; record -99.

ZERO.offset Trace.RECORD.TIME(-99.)          ; specify the time of record
                                             ; -99. as global zero point
```

- **Cycle Accurate Tracing Pros.**

  Provides how much core clocks it took to execute a program section.

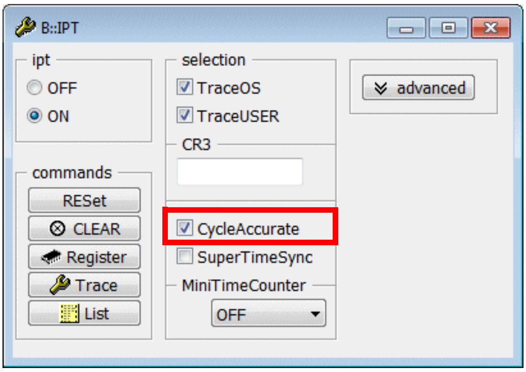
  Allows to synchronize traces from different trace sources if Time Synchronization packets are available (not implemented yet).

- **Cycle Accurate Tracing Cons.**

  Cycle accurate tracing requires up 2 times more bandwidth.

Cycle accurate tracing and changing core clock while recording

Cycle accurate tracing has to be enabled in the IPT configuration window.

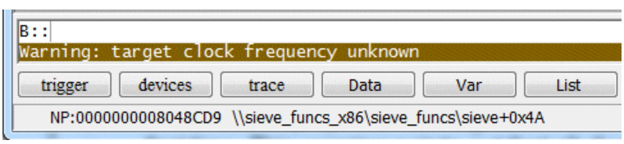


IPT.CycleAccurate ON	Advise Intel® PT to generate cycle count information.

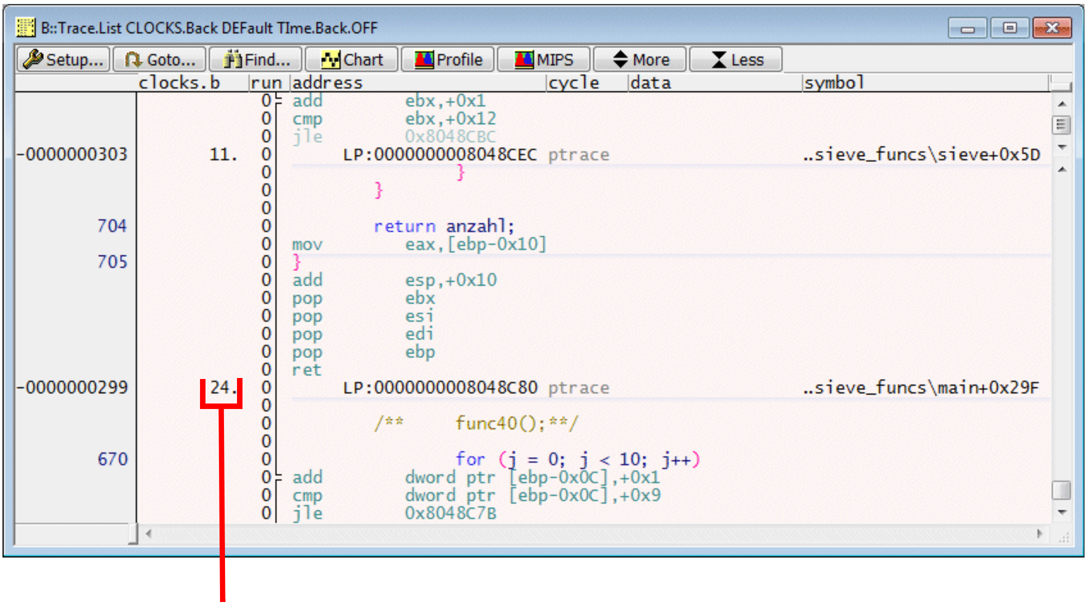
```
; advise TRACE32 to display a trace listing with
; cycle count information (CLOCKS.Back)
; advise TRACE32 to suppress the display
; of the timestamp information (TIme.Back.OFF)
Trace.List CLOCKS.Back DEFault TIme.Back.OFF
```

The following command allows to specify this display as default for the **Trace.List** window.

```
SETUP.ALIST CLOCKS.Back DEFault TIme.Back.OFF
```

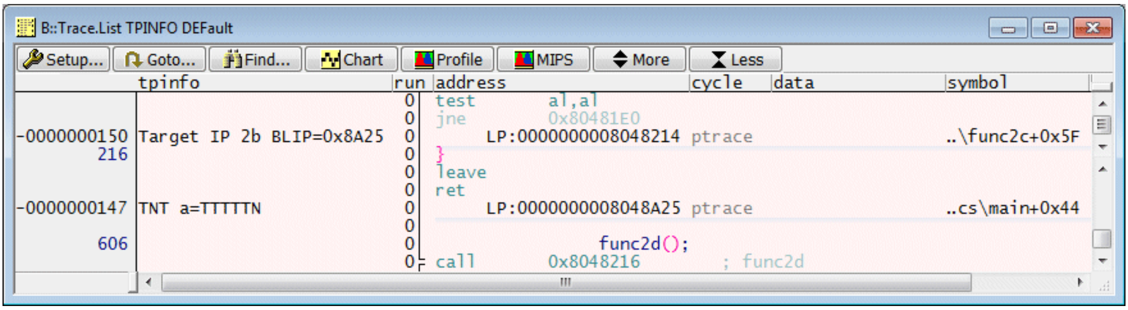


TRACE32 displays the warning above when the recorded trace information is analyzed and displayed the first time. This warning points out that all displayed time information (TIme.Back, TIme.Zero) might be inaccurate.



Cycle count information
relative to the previous record

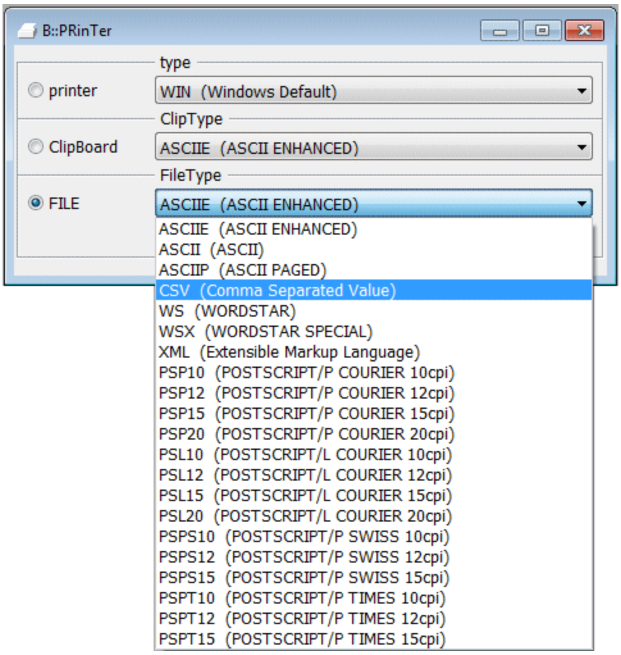
```
; advise TRACE32 to display a trace listing with the decoded trace packet
; (TPINFO)
Trace.List TPINFO DEFault
```



Belated Trace Analysis

There are several ways for a belated trace analysis:

1. Save a part of the trace contents into a file (ASCII, CSV or XML format) and analyze this trace contents outside of TRACE32 PowerView.



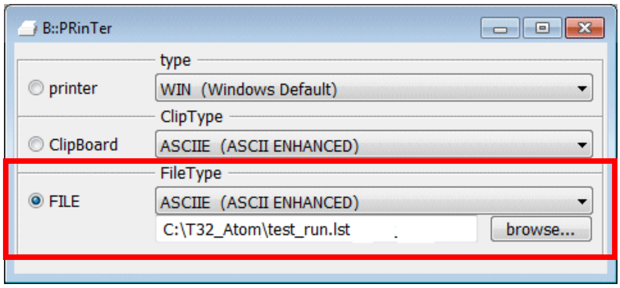
2. Save the trace contents in a compact format into a file. Load the trace contents at a subsequent date into a TRACE32 Instruction Set Simulator and analyze it there.

3. Export the STP byte stream to postprocess it with an external tool.

Save the Trace Information to an ASCII File

Saving a part of the trace contents to an ASCII file requires the following steps:

1. Select **Printer Setting…** in the **File** menu to specify the file name and the output format.




```
PRinTer.FileType ASCIIE                 ; specify output format
                                        ; here enhanced ASCII

PRinTer.FILE test_run.lst               ; specify the file name
```

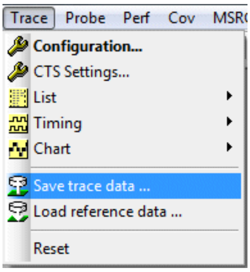
2. It might make sense to save only a part of the trace contents into the file. Use the record numbers to specify the trace part you are interested in.

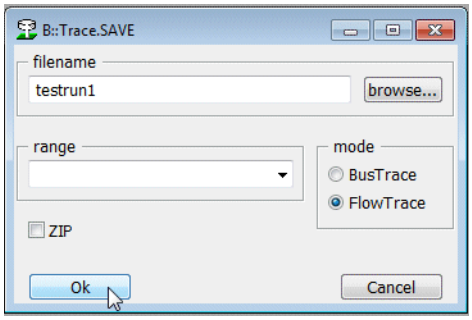
   TRACE32 provides the command prefix **WinPrint.** to redirect the result of a display command into a file.

```
; save the trace record range (-8976.)--(-2418.) into the
; specified file
WinPrint.Trace.List (-8976.)--(-2418.)
```

3. Analyze the result outside of TRACE32.

1. **Save the contents of the trace memory into a file.**

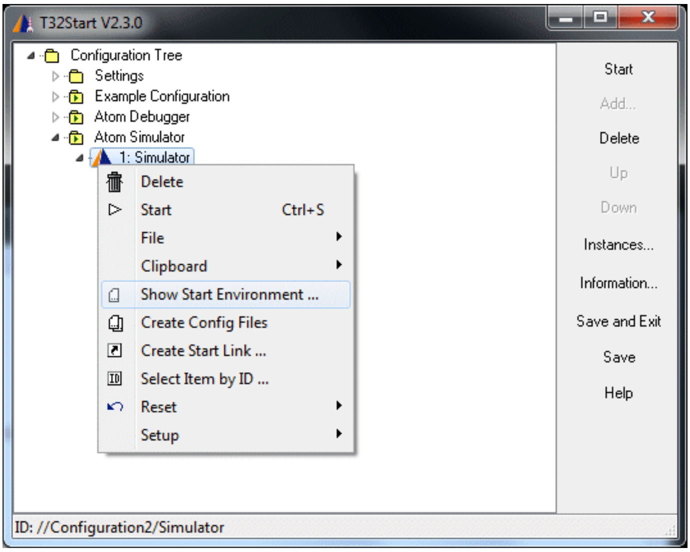


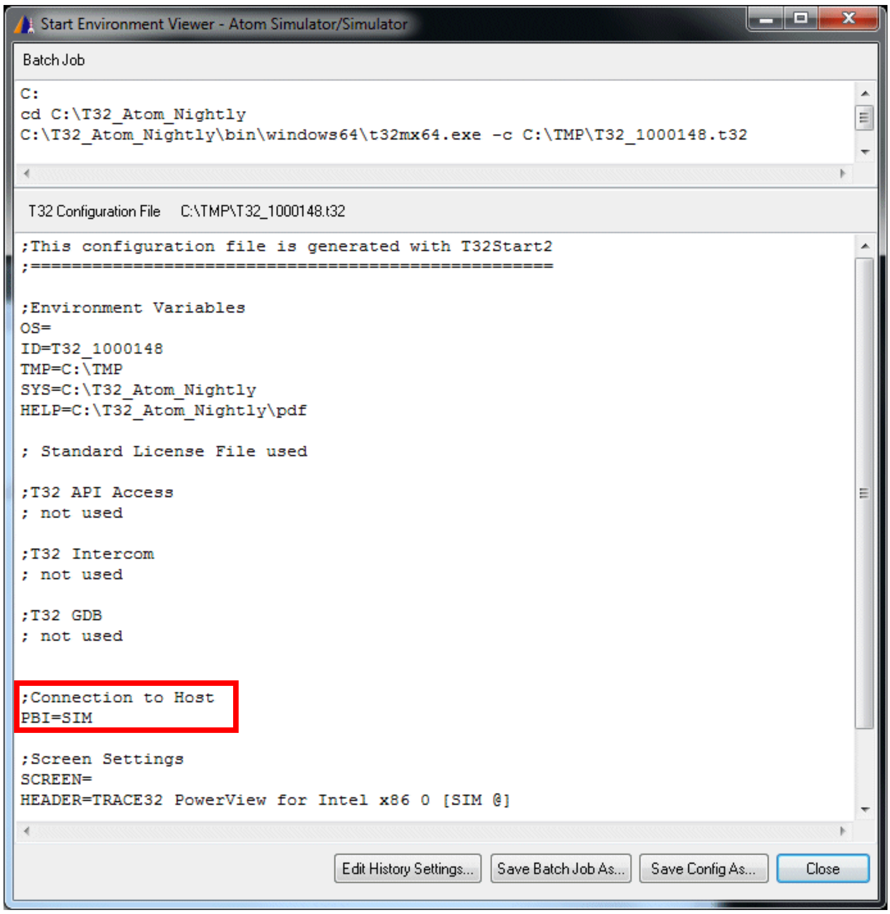


The default extension for the trace file is **.ad**.

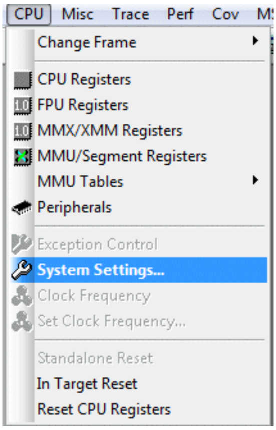
```
Trace.SAVE testrun1
```

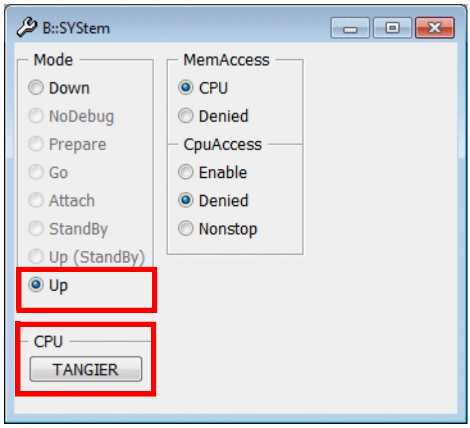
2. Start a TRACE32 Instruction Set Simulator (PBI=SIM).



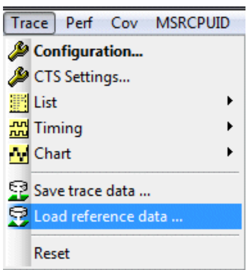


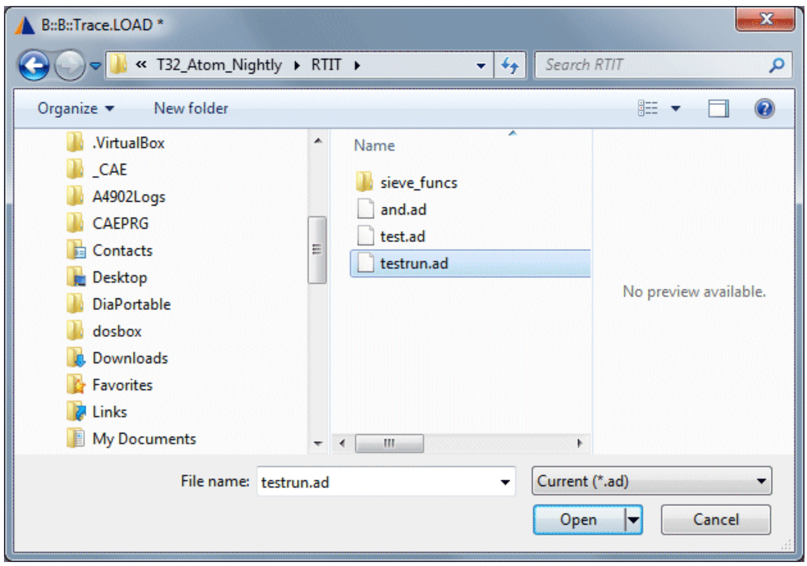
3. **Select your target CPU within the simulator. Then establish the communication between TRACE32 and the simulator.**





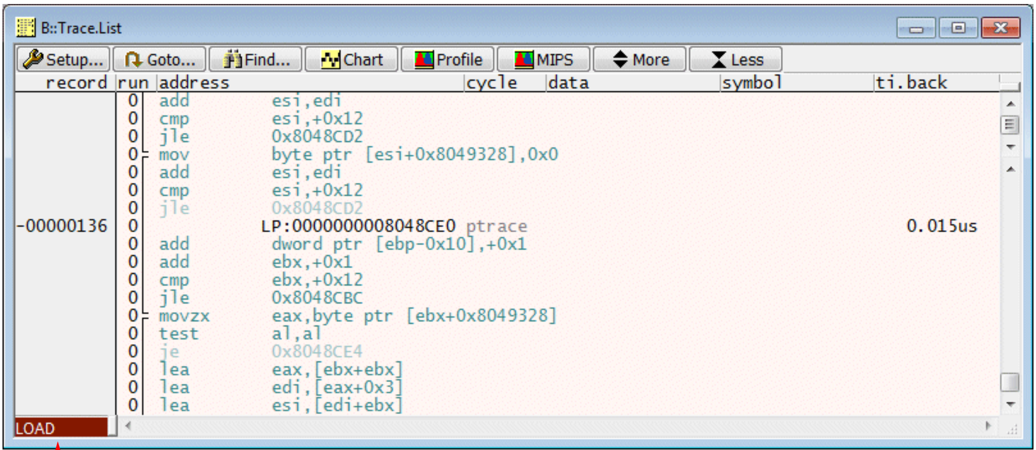
4. **Load the trace file.**






```
Trace.LOAD testrun
```

5. **Display the trace contents.**

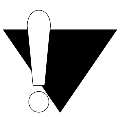


LOAD indicates that the source for the trace information is the loaded file.

6. **Load symbol and debug information if you need it.**

```
Data.LOAD.Elf sieve_funcs_x86.elf /NoCODE
```

The TRACE32 Instruction Set Simulator provides the same trace display and analysis commands as the TRACE32 debugger.

	Postprocessing of recorded trace information with the TRACE32 Instruction Set Simulator becomes more complex if an operating system that uses dynamic memory management to handle processes/task is used (e.g. Linux).

Script version

Save the trace contents in the recording TRACE32 instance:

```
Trace.SAVE testrun.ad
```

Prepare the TRACE32 Instruction Set Simulator for off-line processing of the trace information:

```
SYStem.CPU TANGIER

SYStem.Up

Trace.LOAD testrun.ad

Data.LOAD.Elf sieve_funcs_x86.elf /NoCODE

Trace.List
```

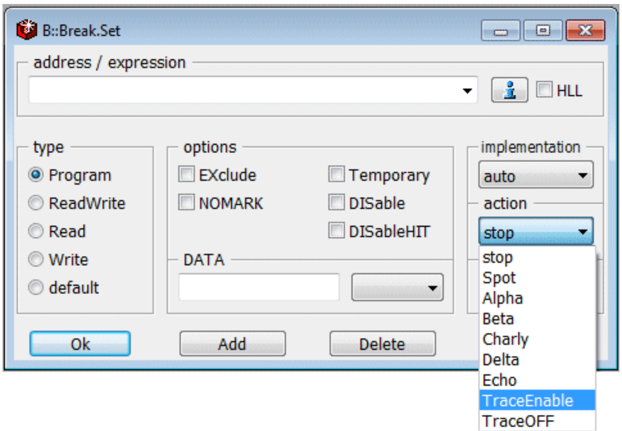
Trace.EXPORT.TracePort *<file>*	Export trace raw data (no timestamps).

```
SystemTrace.EXPORT.TracePort mytest1.ad
```

Trace Control by Filters

Intel® PT provides 2 address ranges for trace control. The smallest range size is 4 bytes.

TRACE32 PowerView provides access to these address ranges by the **action** field in the **Break.Set** dialog.



The 2 address ranges can be used for the following purposes:

TraceEnable advises Intel® PT to generate program flow information for the specified address range only.

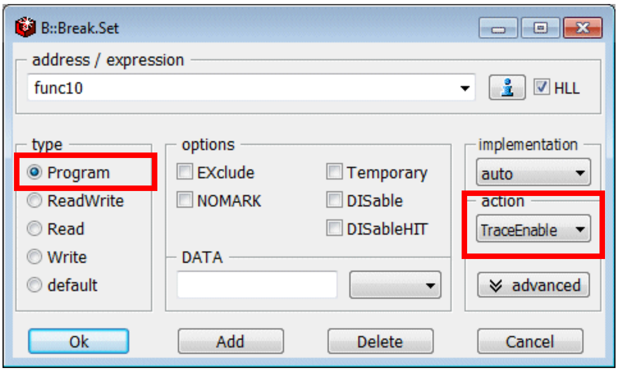
TraceOFF advises Intel® PT to stop the generation of program flow information as soon as a specified address range is reached.

Both filters are programmed to all Intel® PT in an SMP configuration.

TraceEnable

Example 1: Advise Intel® PT to generate program flow information only for function func10.

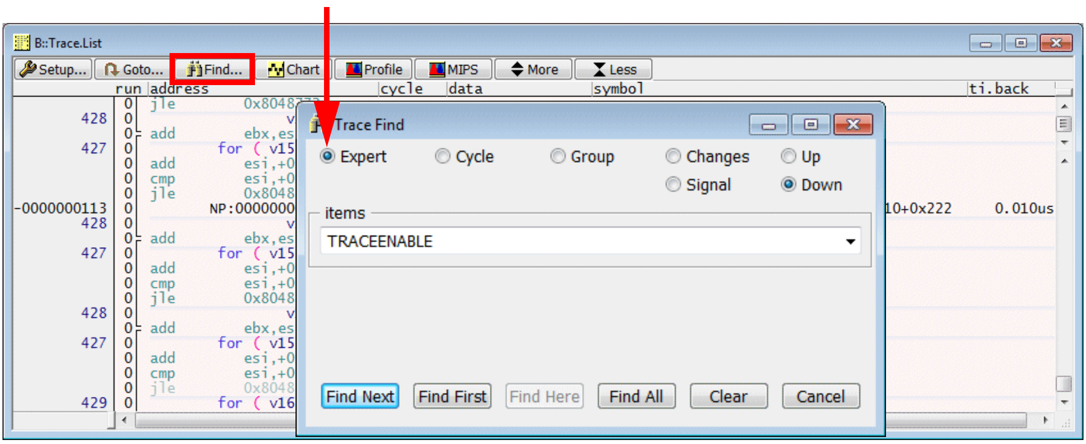
1. **Set a Program breakpoint to the address range of func10 and select the action TraceEnable.**

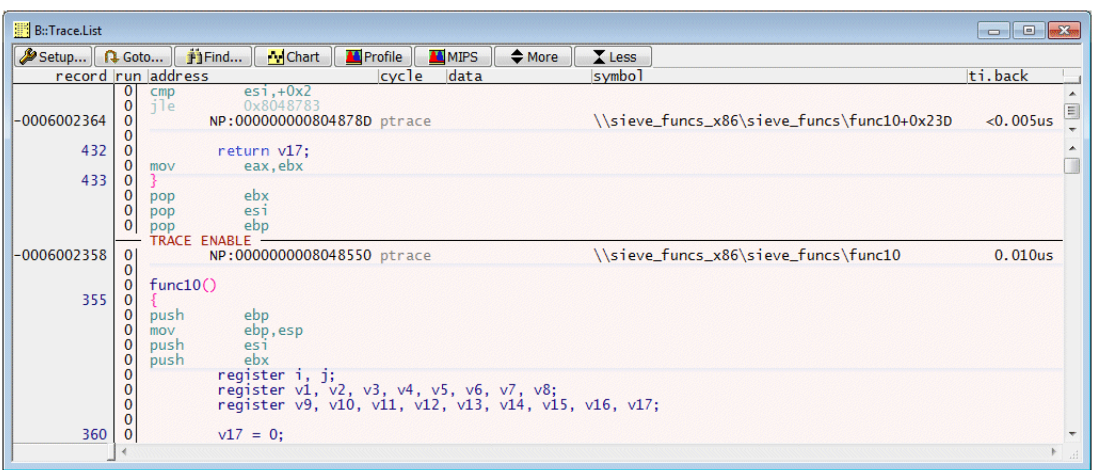


2. **Start and stop the program execution.**

3. **Display the result.**

TRACE ENABLE indicates the start of the message generation. It might be necessary to search for it.

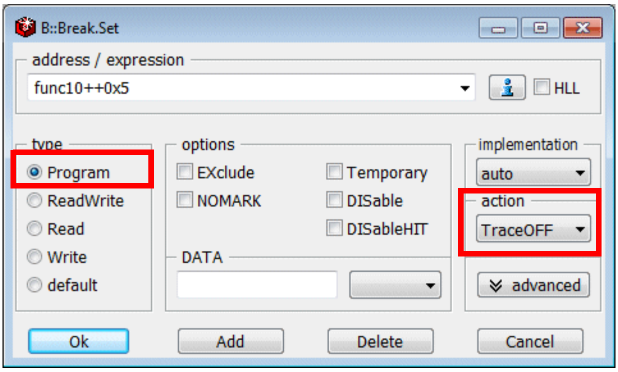




```
Break.Delete /ALL

Var.Break.Set func10 /Program /TraceEnable

Go

…

Break

Trace.List
```

Example 2: Advise Intel® PT to stop the generation of program flow information as soon as function func10 is entered.

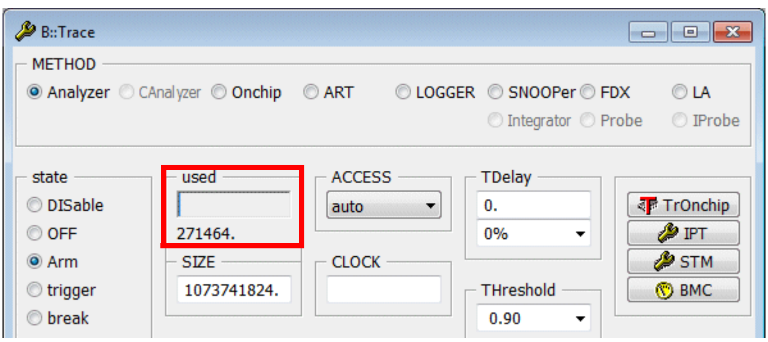
1. **Set a Program breakpoint to the start address of func10 plus 5 bytes and select the action TraceOFF.**



2. **Start the program execution.**

TRACE32 has, unfortunately, no way to detect that Intel® PT stopped the generation of trace information.

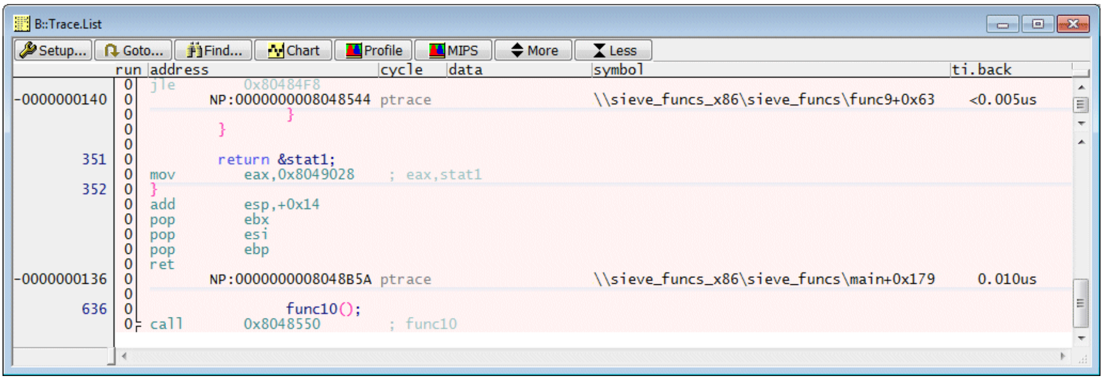
Off-chip trace: Since the Analyzer is recording STP packets, **used** could increase, because other trace sources continue generating STP packets.



Onchip trace: TRACE32 can not read the filling level of the onchip trace while recording.

3. **Stop the program execution.**

4. **Display the result.**




```
Break.Delete /ALL

Break.Set func10++0x5 /Program /TraceOFF

Go

…

Break

Trace.List
```

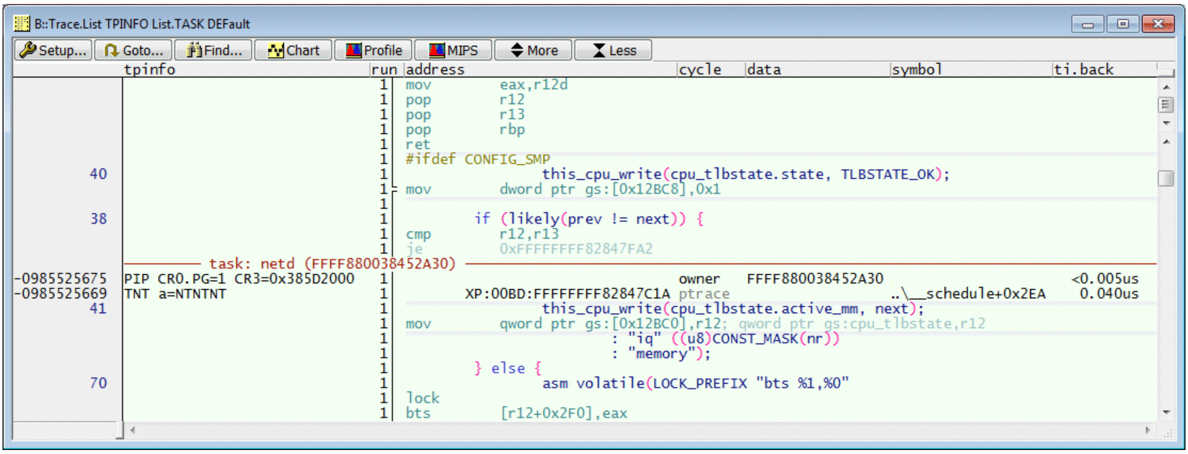
OS-Aware Tracing

OS-aware tracing requires that OS-aware debugging is configured. For more information refer to **"OS-aware Debugging"** (glossary.pdf).

Process Switch Packets

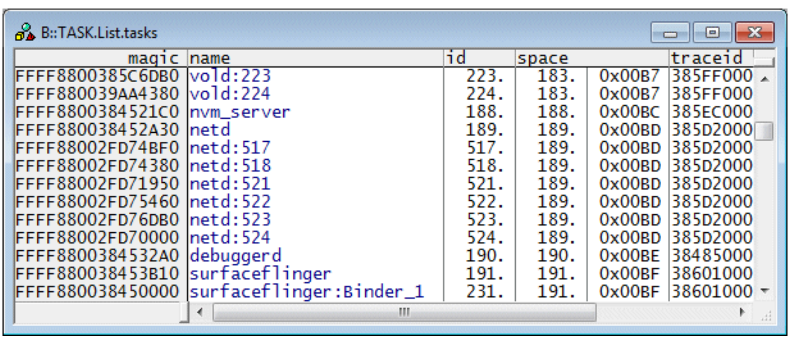
x86/x64 processors have a CR3 control register that contains the Process Context Identifier (PCID). On every context switch the corresponding PCID is loaded to CR3.

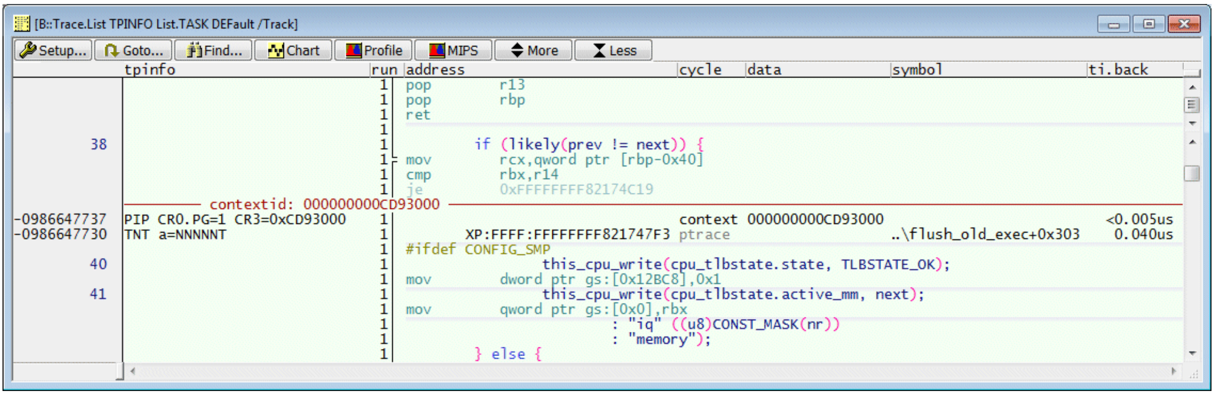
Intel® PT generates a Paging Information Packet (PIP) when a write to CR3 occurs.



TRACE32 names the cycle type **owner** if the PCID loaded to CR3 can be assigned to a process.

The command **TASK.List.tasks** can be used the check all assignments currently known to TRACE32. The **traceid** represents the PCID in this display.





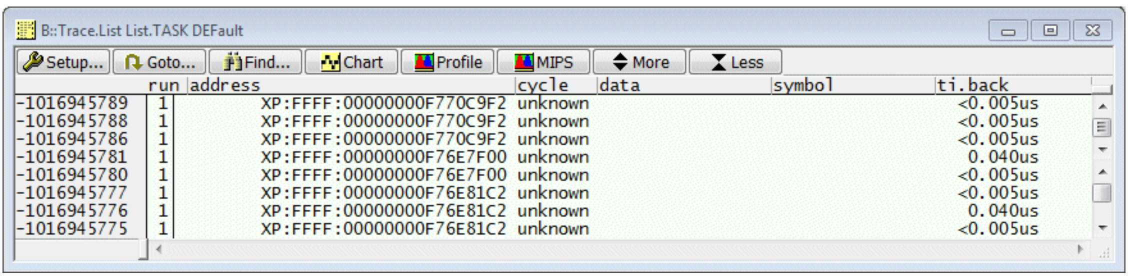
TRACE32 names the cycle type **context** if the PCID loaded to CR3 can not be assigned to a process.

The fact that the PCID can not be assigned to a process results in the following,:

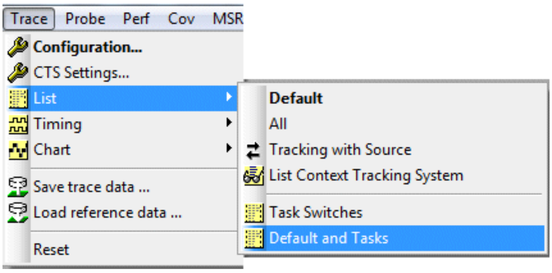
- Since TRACE32 does not require the PCID to decode trace information for the common address range, full trace decoding is possible.

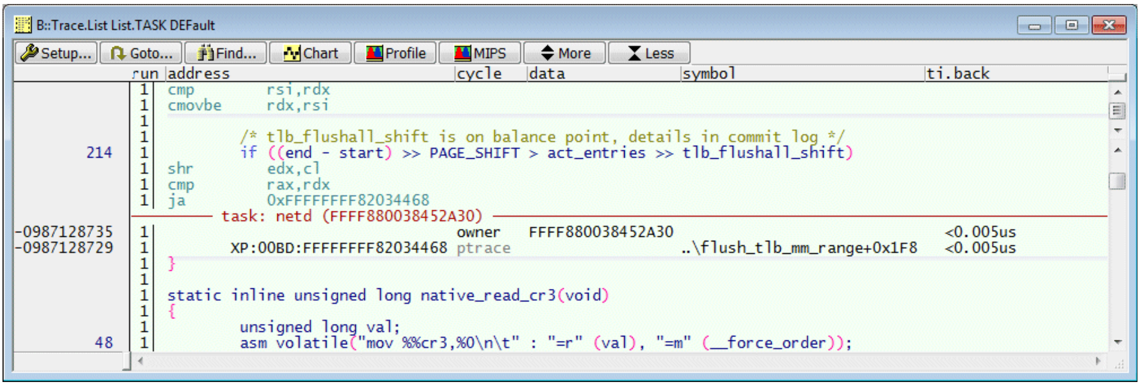
```
; command in the setup for the OS Awareness
TRANSlation.COMMON 0xffff880000000000--0xffffffffffffffff
```

- For all other address ranges a decoding of the trace information is not possible. The cycle type **unknown** is used for trace information that can not be decoded.



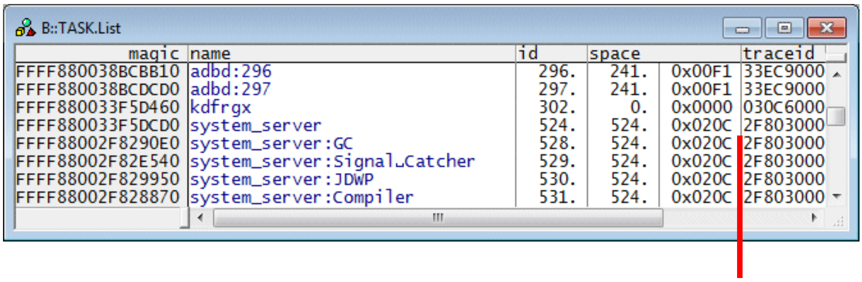
NOTE:	The Real-Time Instruction Trace (RTIT), doesn't feature the process switching packets. If multiple user space applications are traced, it is only possible to decode the trace packets of the kernel. The cycle type unknown is used for the user space trace packets. For decoding the trace packets of a user application, it is necessary to filter the process of interest using the CR3 filter. RTIT was implemented on very few devices, then it was extended to the Intel Processor Trace which supports the process switching trace. The RTIT trace is also covered by TRACE32 using the IPT command group.



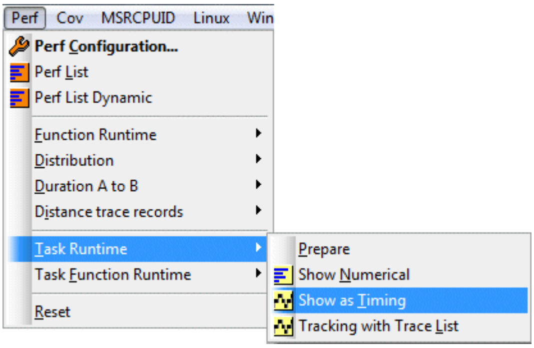


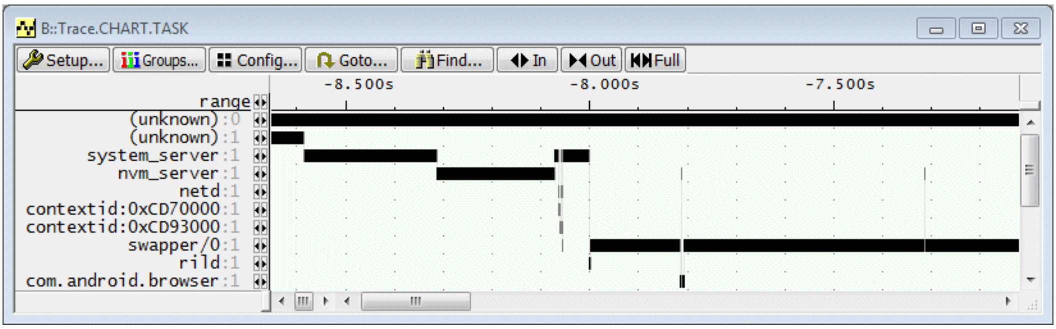

```
Trace.List List.TASK DEFault          ; display trace listing with
                                      ; decoded task switch information
```

NOTE: This is a process switch analysis, since Paging Information Packets (PIP) only indicate process switches, but no thread switches.



Threads do not have their own traceid

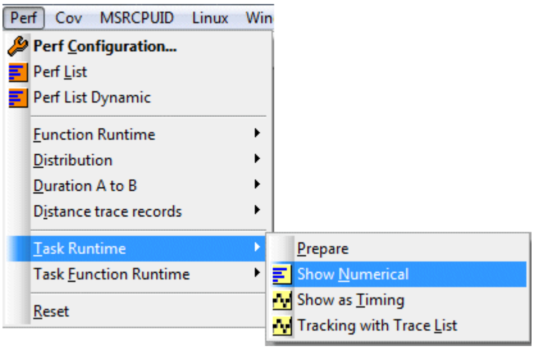


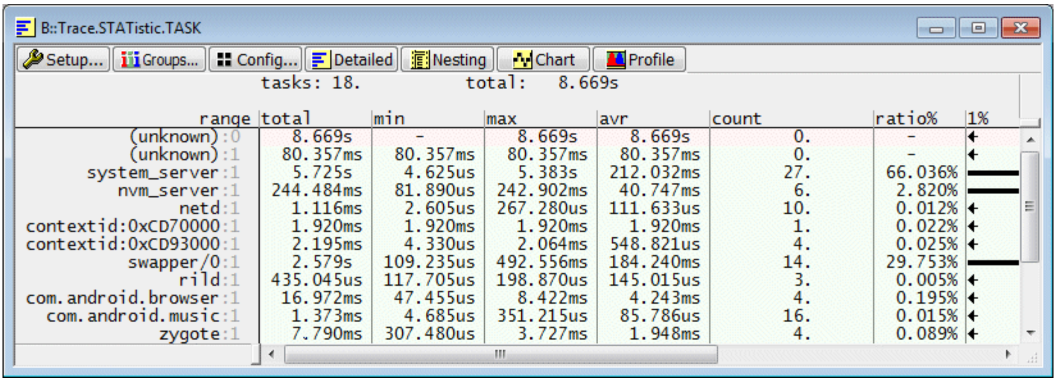


contextid:<*trace_id*> indicates process switches for which the <*trace_id*> can not be assigned to a process.

Trace.Chart.TASK [/SplitCORE]	Display process time chart - graphical display - split the results per core - sort the results per recording order
Trace.Chart.TASK /MergeCORE	Display process time chart - graphical display - merge the results of all cores Trace information is analyzed independently for each core. The time chart summarizes these results to a single result.

The recording time before the first Paging Information Packet (PIP) is assigned to the **(unknown)** task.

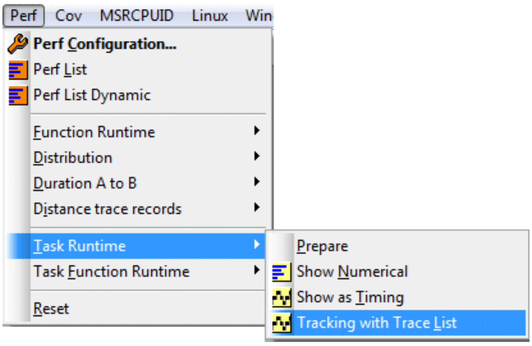




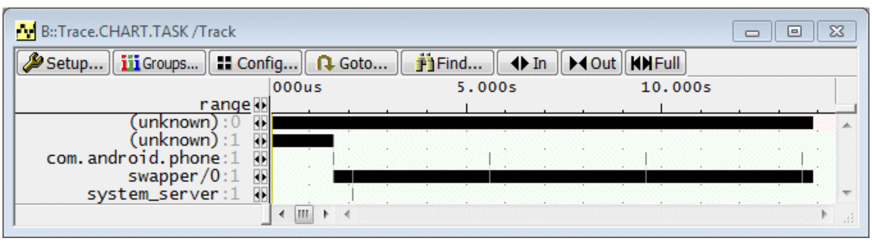
Trace.STATistic.TASK [/SplitCORE]	Process runtime statistic - numerical display - split the results per core - sort the results per recording order
Trace.STATistic.TASK /MergeCORE	Process runtime statistic - numerical display - merge the results of all cores Trace information is analyzed independently for each core. The statistic summarizes these results to a single result.

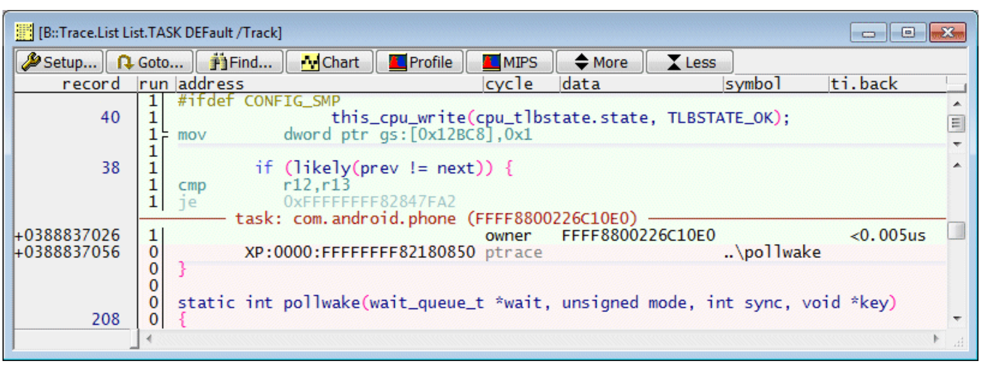
Find Process Switches in the Trace

1. **Open a process time chart window and a trace listing with decoded process switch information. Link both windows by using the /Track option.**

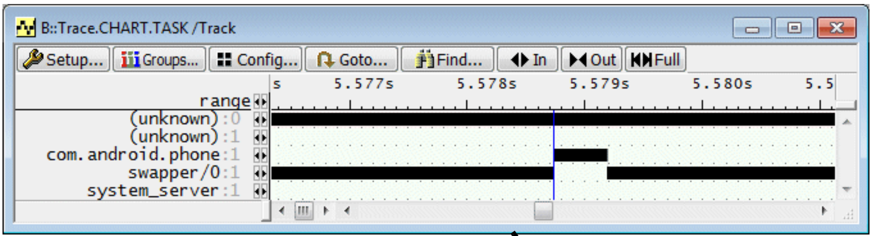



```
Trace.Chart.TASK /Track                         ; open process time chart
                                                ; window

Trace.List List.TASK DEFault /Track             ; open a default trace
                                                ; listing that includes
                                                ; process information

; both windows use the /Track option
; a window opened with the /Track option follows the cursor movement
; of the active window

; tracking between trace windows is based on the timestamp
; information
```

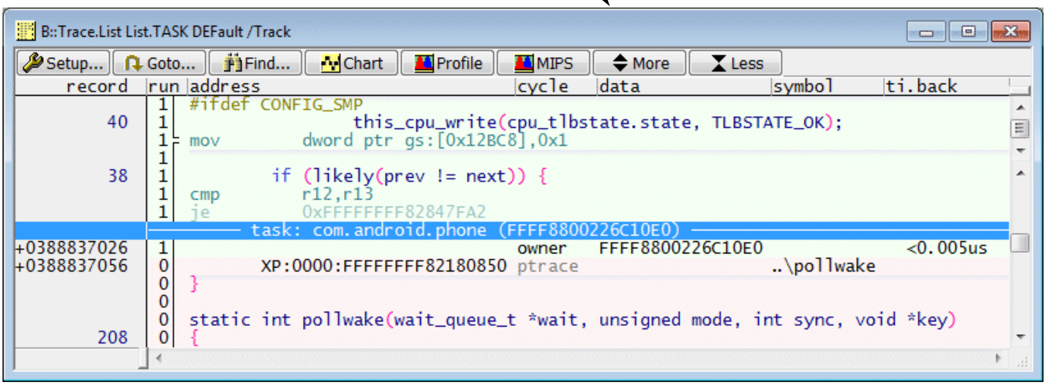




2. **Use the arrow keys of the process of interest to move to next state change.**



The trace listing follows the cursor movement in he Trace.Chart.TASK window



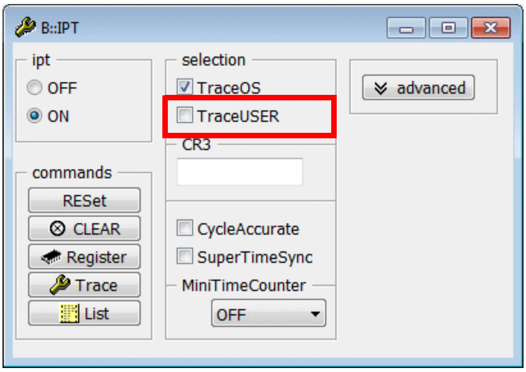
Filtering by Privilege Level

Intel® PT can be advised to generate program flow information only for:

- privilege level 0
- all privilege levels greater than 0

Example: Advise Intel® PT to generate only program flow information for privilege level 0.

1. **Uncheck TraceUSER in the IPT configuration window.**



2. **Start and stop the program execution.**
3. **Display the result.**

```
IPT.state

IPT.TraceUSER OFF

Go

…

Trace.List
```

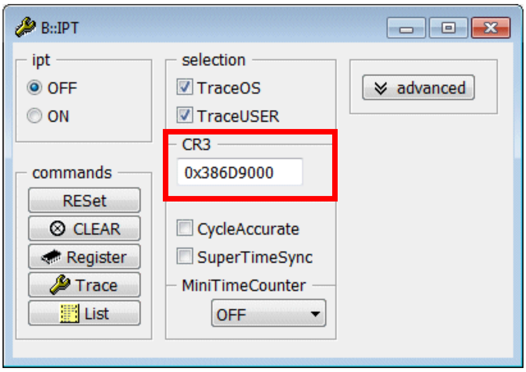
Filtering by Process

Intel® PT can be advised to generate program flow information only for a process of interest.

Example: Advise Intel® PT to generate only program flow information for the process “logcat”.

1. **Program the CR3 filter via the IPT window.**

```
IPT.state                                       ; open IPT configuration
                                                ; window
```

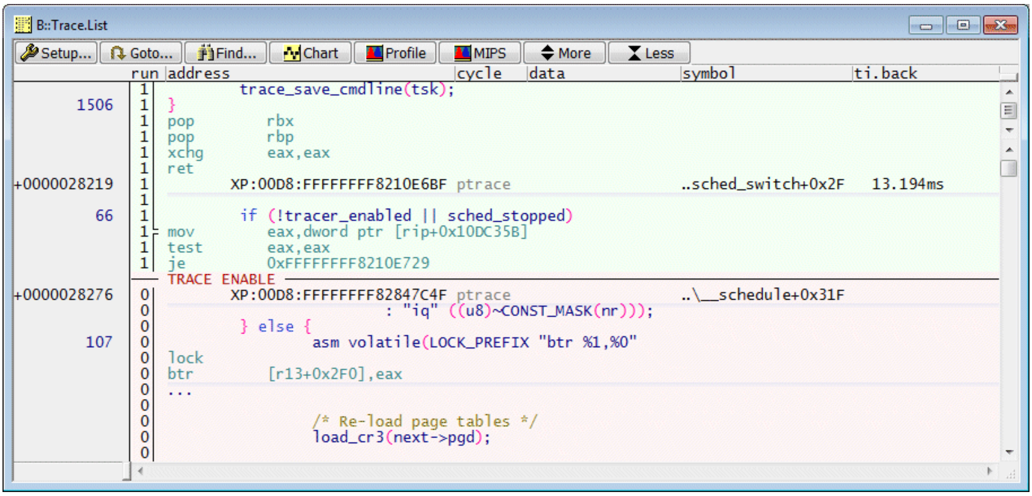



```
; specify the <trace_id> of the process "logcat"
IPT.CR3 0x386D9000

; IPT.CR3 TASK.PROC.NAME2TRACEID("logcat")
```

TASK.PROC.NAME2TRACEID(*<process_name>*)	Returns the *<trace_id>* of specified process.

2. **Start and stop the program execution.**

3. **Display the result.**

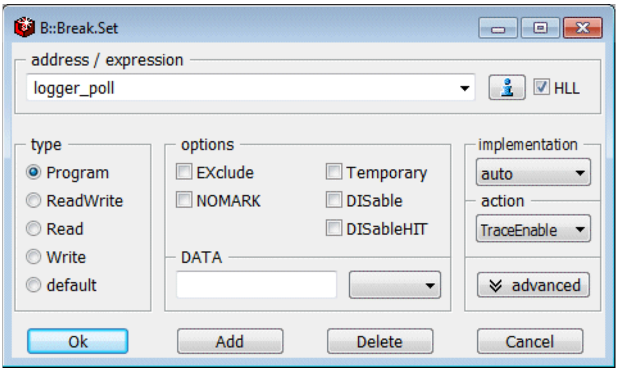


TRACE ENABLE indicates the re-start of the program flow trace generation.

Please be aware that TRACE32 decodes all trace information for the process specified in the command **IPT.CR3** *<trace_id>*. Intel® PT does not generate Paging Information Packet (PIP) in this scenario.

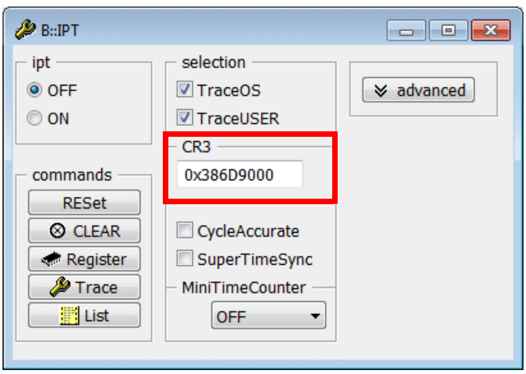
Example: Advise Intel® PT to generate only program flow information when the function “logger_poll” is running in the context of the process “logcat”.

1. **Set a Program breakpoint to the address range of the function logger_poll and select the action TraceEnable.**



2. **Program the CR3 filter via the IPT window.**

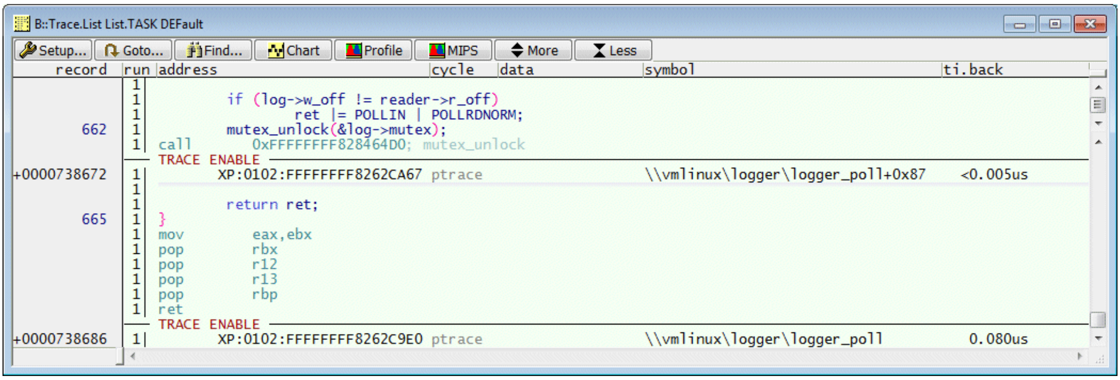
```
IPT.state                                    ; open IPT configuration
                                             ; window
```




```
IPT.CR3 TASK.PROC.NAME2TRACEID("logcat")
```

3. **Start and stop the program execution.**

4. **Display the result.**



TRACE ENABLE indicates the re-start of the program flow trace generation.

Belated Analysis

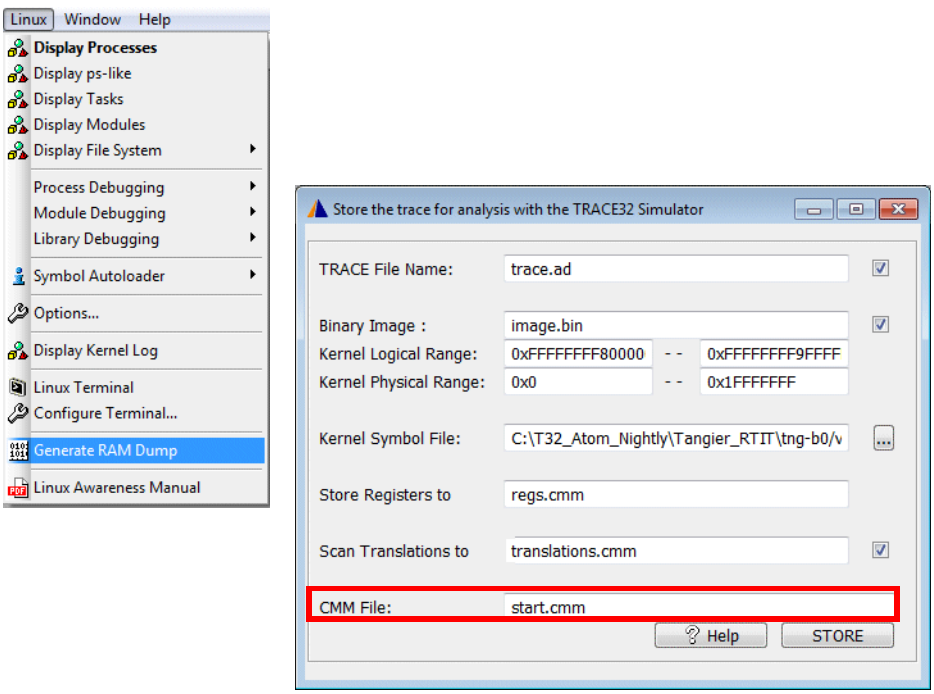
Postprocessing of recorded trace information with the TRACE32 Instruction Set Simulator requires complex preparations if an operating system that uses dynamic memory management to handle processes is used (e.g. Linux).

The following information has to be store after recording and re-loaded to the TRACE32 Instruction Set Simulator:

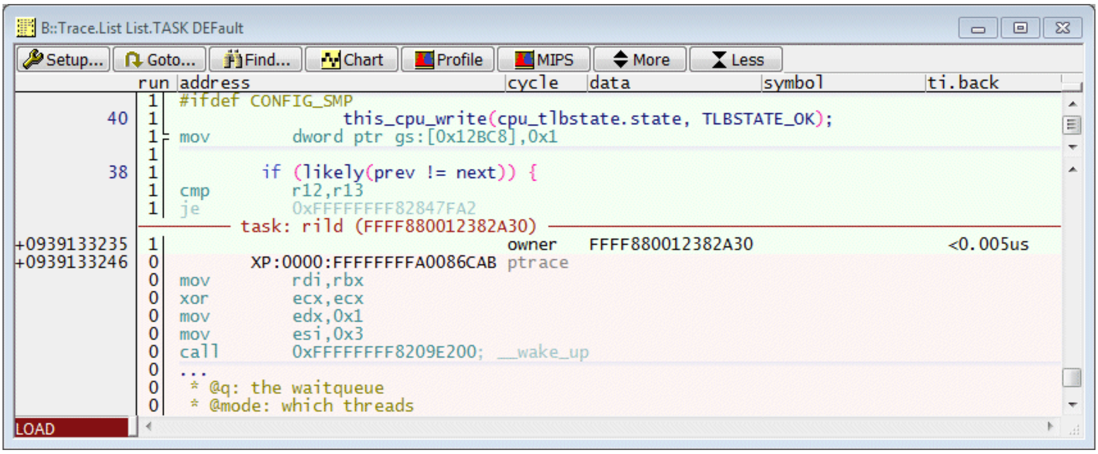
- The recorded trace information
- The whole kernel address space (code and data)
- The core registers
- All MMU-related registers
- The settings of the Debugger Address Translation (TRACE32 command group: **TRANSlation**)

Example for Linux

The **Generate RAM Dump** command in the **Linux** menu provides a store framework. It generates a **CMM file** that summarizes all commands for the TRACE32 Instruction Set Simulator.



If you start a TRACE32 Instruction Set Simulator and run the generated script, the recorded trace information can be analyzed there. Please be aware that additional settings might be necessary e.g. the specification of the search paths for the C/C++ sources.

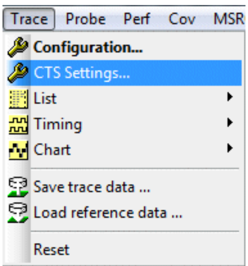


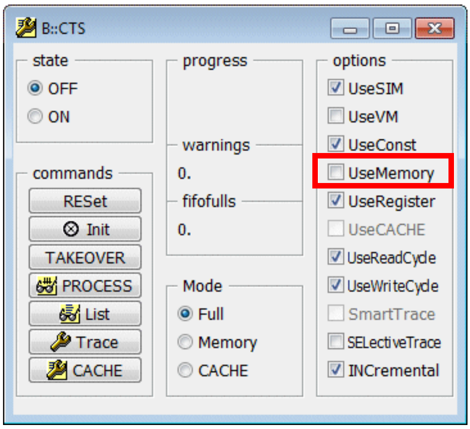
Trace-based Debugging (CTS)

Trace-based debugging allows to re-run the recorded program section within TRACE32 PowerView.

Setup

Since Intel® PT does not provide any information on read/write accesses, **UseMemory** has to be unchecked. A full explanation on this is given later in the chapter **"CTS Technique"**, page 93.

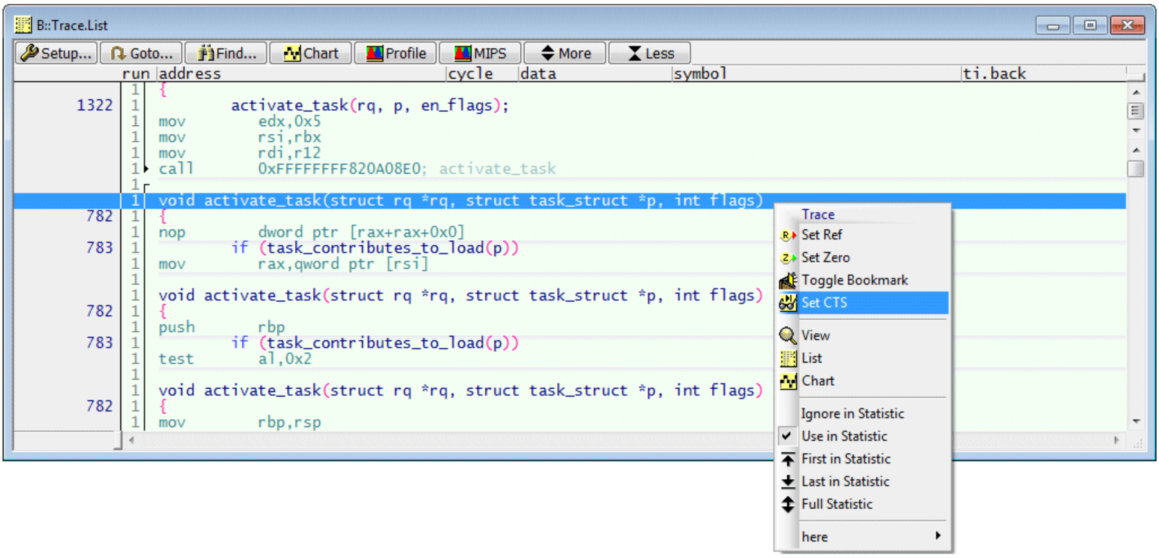




CTS.UseMemory OFF

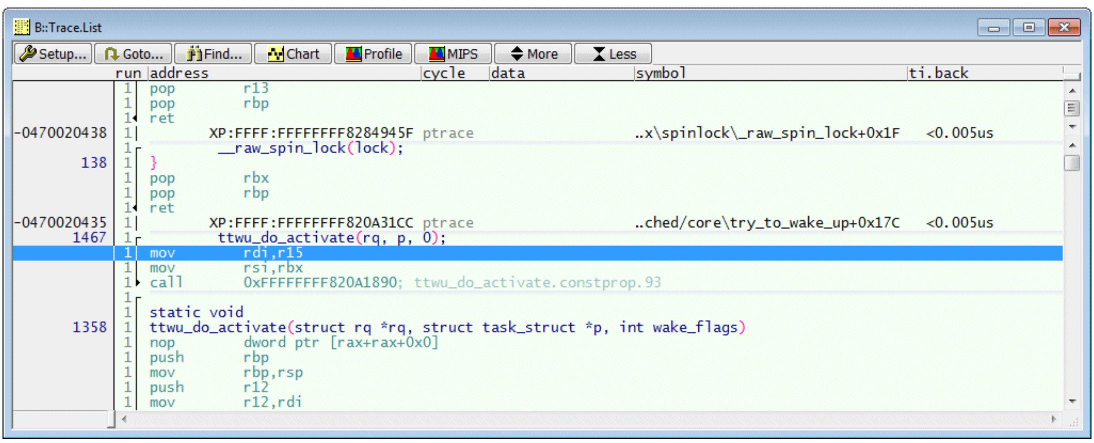
Get Started

Specify the starting point for the trace re-run by selecting **Set CTS** from the Trace pull-down menu. The starting point in the example below is the entry to the function **activate_task** executed by core 1.



Selecting **Set CTS** has the following effect:

- TRACE32 PowerView will use the preceding trace packet as starting point for the trace re-run.



- The TRACE32 PowerView GUI does no longer show the current state of the target system, but it shows the target state as it was, when the starting point instruction was executed. This display mode is called CTS View.

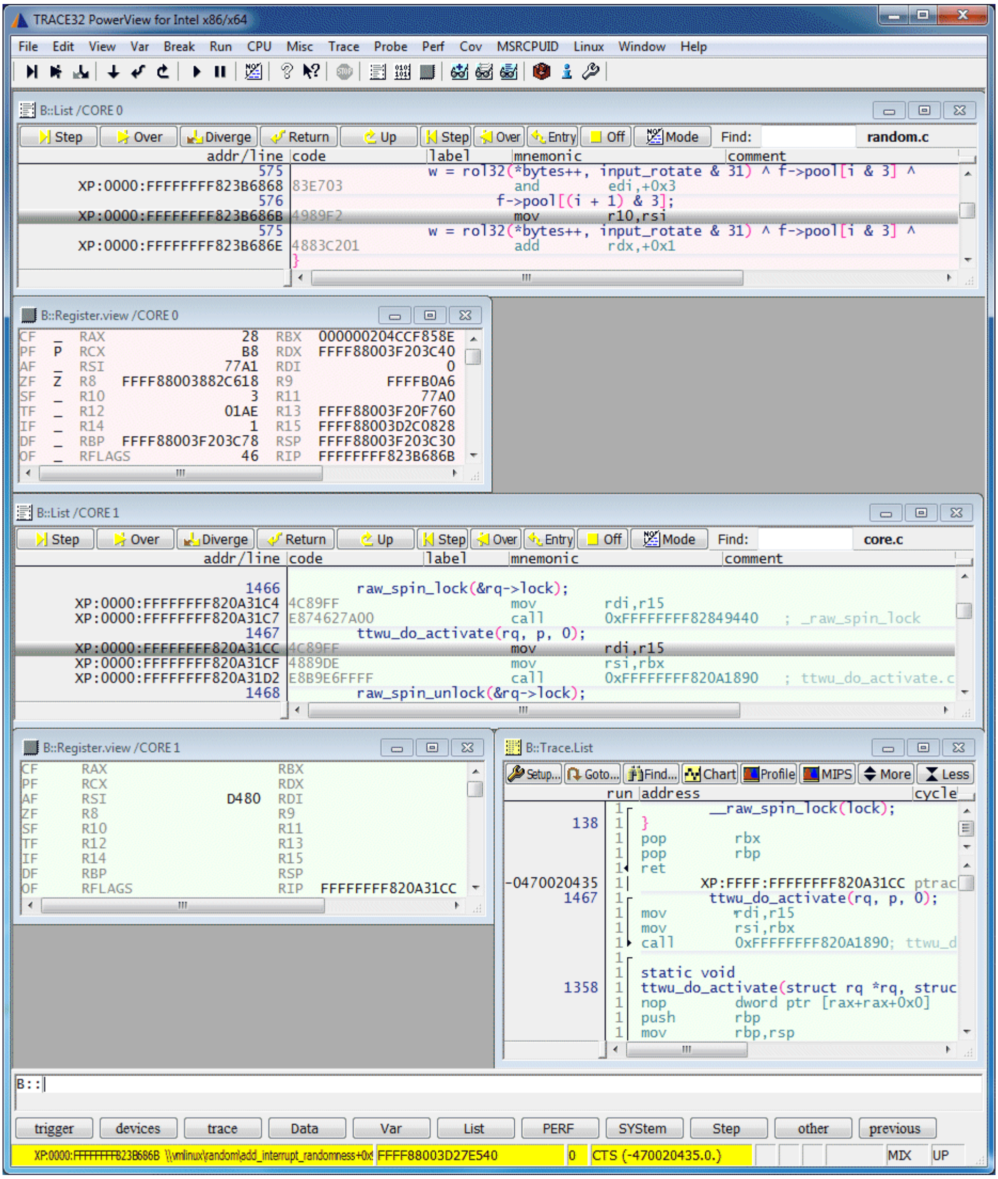
  CTS View means:

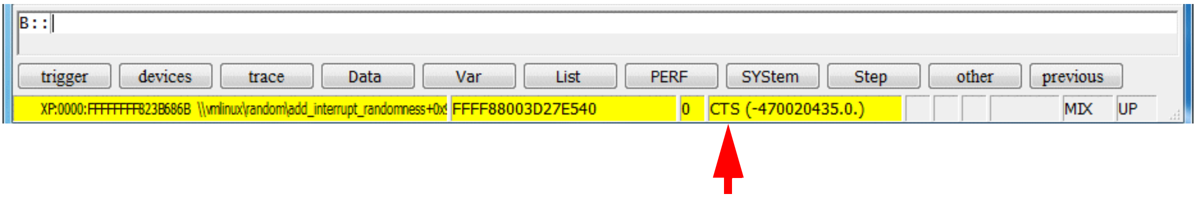
  - The instruction pointers of all cores are set to the values they had when the starting point instruction was executed.

  - The content of the core registers of all cores is reconstructed (as far as possible) to the values they had when the starting point instruction was executed. If TRACE32 can not reconstruct the content of a register it is displayed as empty.

  TRACE32 PowerView uses a yellow look-and-feel to indicate CTS View.

  The **Off** button in the source listing can be used to switch off the CTS View.

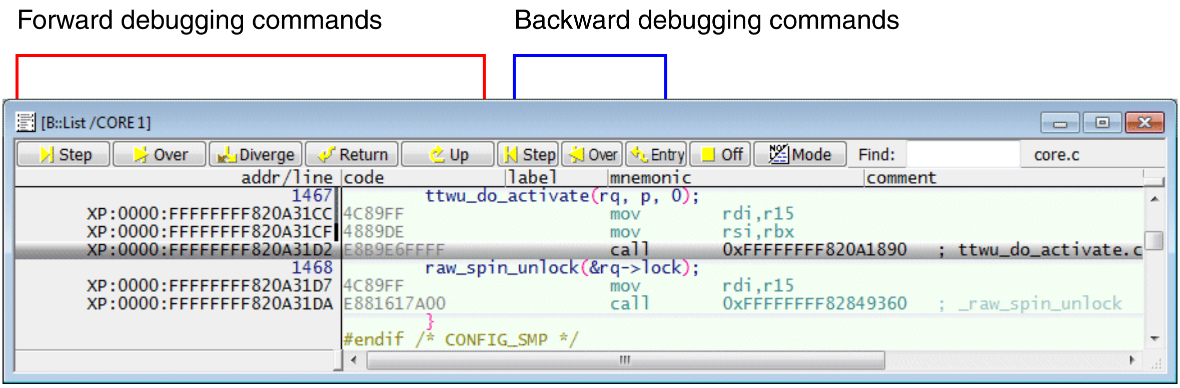




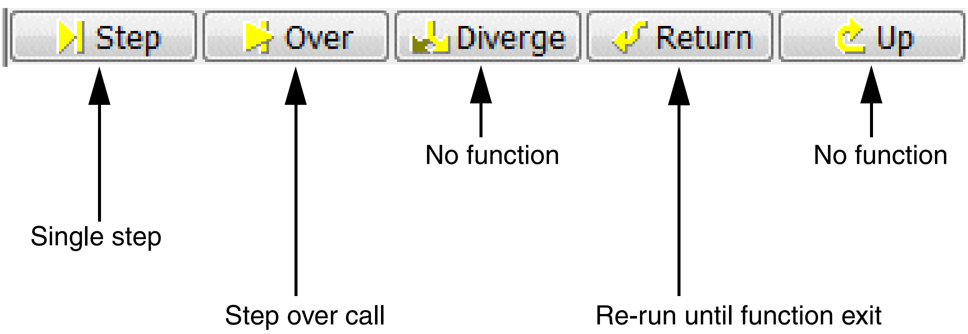
TRACE32 PowerView displays the state of the target as it was when the instruction of the trace record -470020435.0 was executed

Forward and Backward Debugging

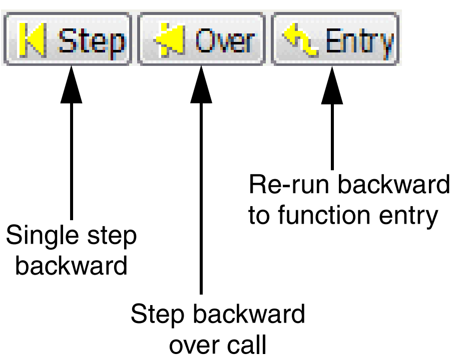
Now you can start to re-run the recorded program section within TRACE32 PowerView by forward or backward debugging.

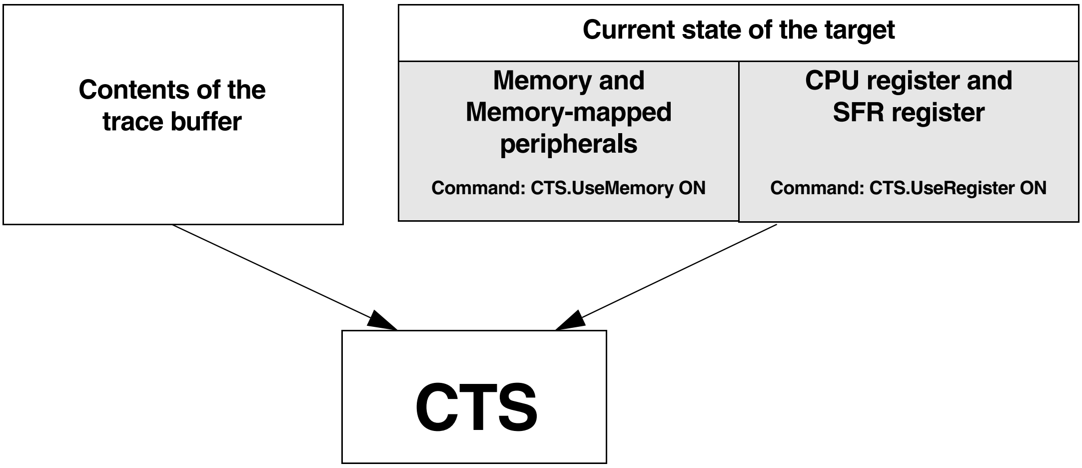


Forward Debugging



Backward Debugging





CTS.UseMemory ON	Default setting within TRACE32

If **CTS.UseMemory** is ON and TRACE32 detects that a memory address was not changed by the recorded program section, TRACE32 PowerView displays the current content of this memory in CTS display mode.

Since Intel® PT does not provide any information on read/write accesses and since most read/write accesses are done by using an indirect address, TRACE32 can not detect which memory content was changed. This is the reason why **CTS.UseMemory** has to be set to OFF.

If **CTS.UseMemory** is switch OFF, but your memory contains constants, you can configure TRACE32 to use these constants by the following commands:

MAP.CONST *<address_range>*

CTS.MapConst ON

CTS.UseRegister ON	Default setting within TRACE32

If **CTS.UseRegister** is ON and TRACE32 detects that a register was not changed by the recorded program section, TRACE32 PowerView displays the current content of this register in CTS display mode.

CTS.UseRegister has to be set to OFF, if you are using **Stack** mode for tracing.

Function Run-Time Analysis - Basic Concept

Software under Analysis (no OS, OS or OS+MMU)

For the use of the function run-time analysis it is helpful to differentiate between three types of application software:

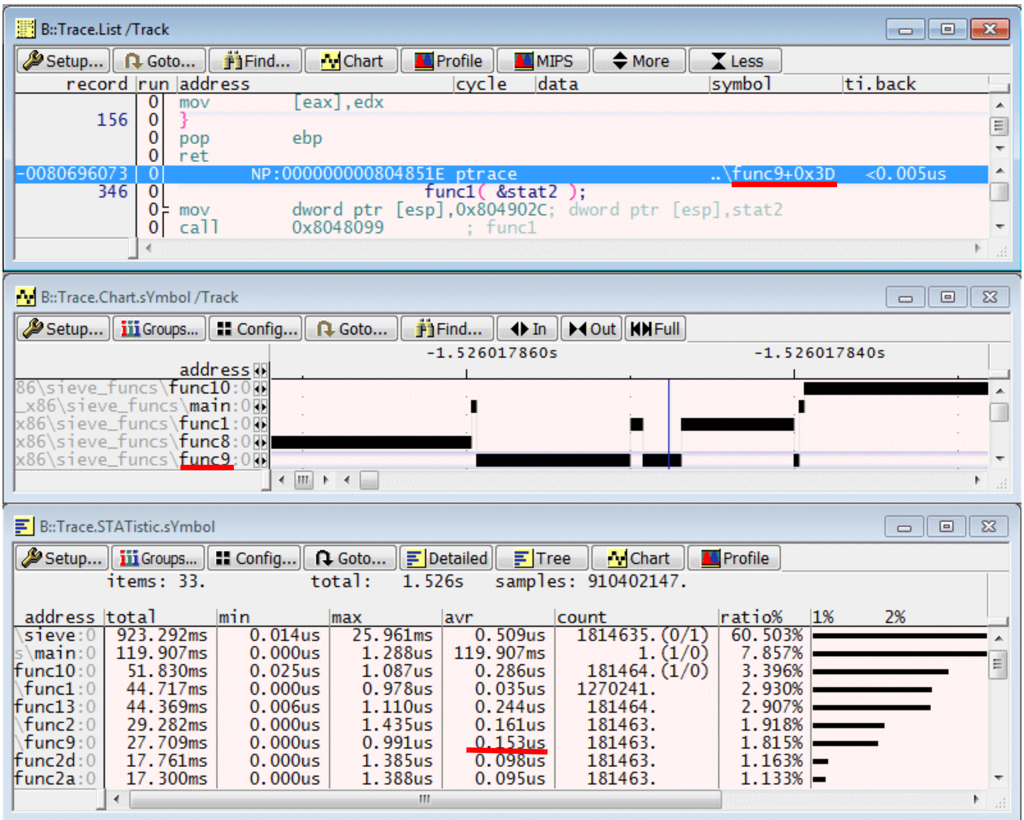
1. Software without operating system (abbreviation: **no OS**)
2. Software that includes an operating system (abbreviation: **OS**)
3. Software with an operating system that uses dynamic memory management to handle processes/tasks (**OS+MMU**).

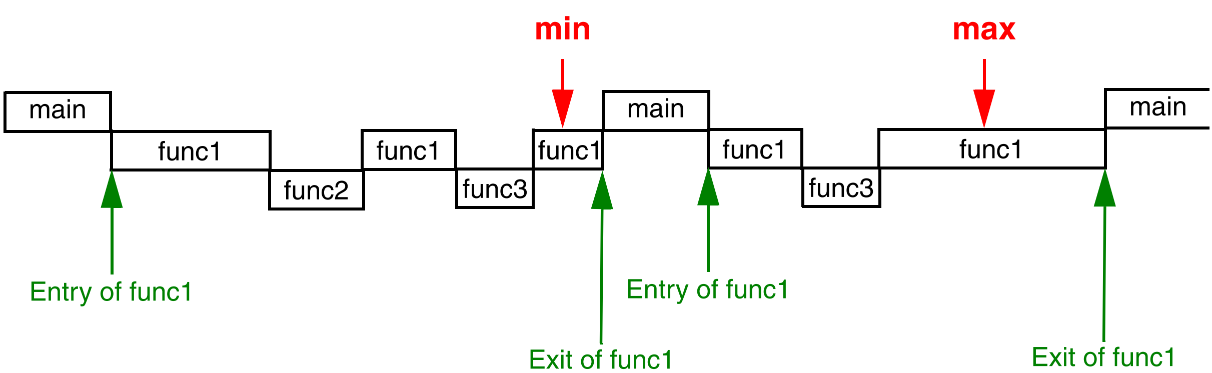
Flat vs. Nesting Analysis

TRACE32 provides two methods to analyze function run-times:

- Flat analysis
- Nesting analysis

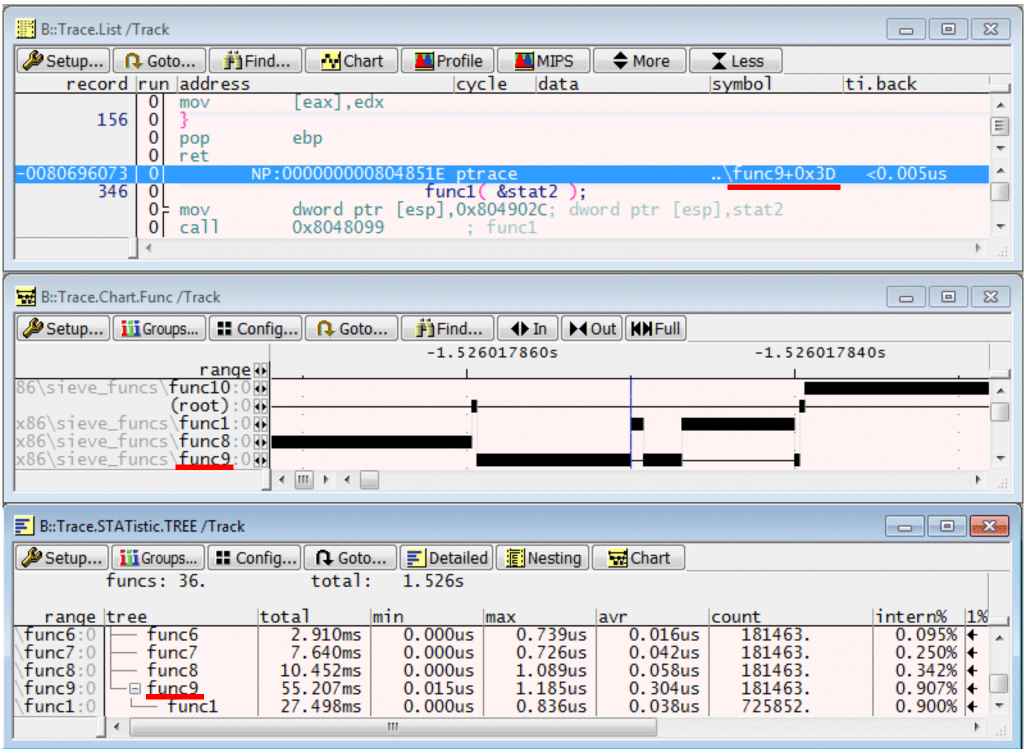
The flat function run-time analysis bases on the symbolic instruction addresses of the trace entries. The time spent by an instruction is assigned to the corresponding function/symbol region.

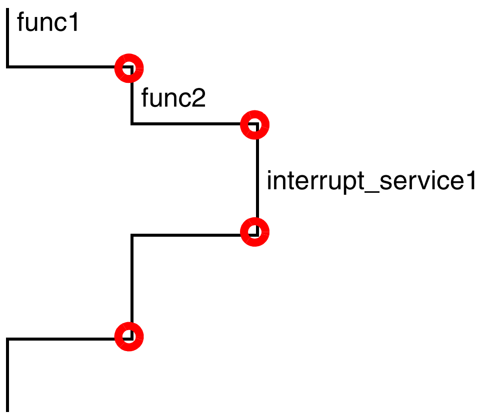




min	shortest time continuously in the address range of the function/symbol region
max	longest time continuously in the address range of the function/symbol region

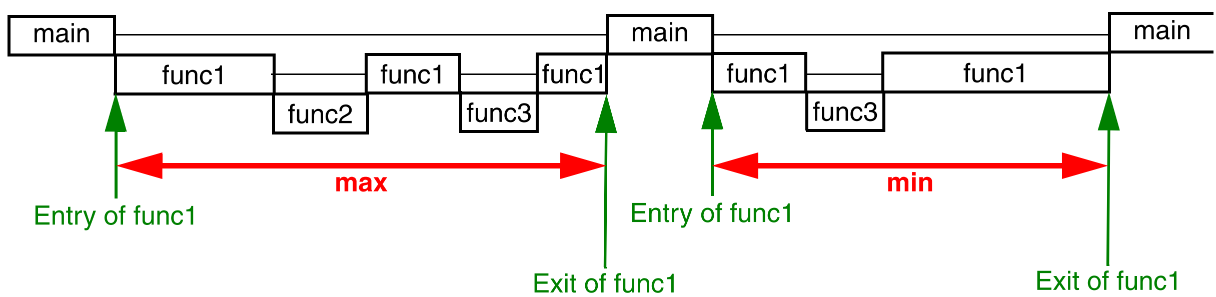
The function nesting analysis analyses only high-level language functions.





In order to display a nesting function run-time analysis TRACE32 analyzes the structure of the program execution by processing the trace information. The focus is put on the transition between functions (see picture above). The following events are of interest:

1. **Function entries**
2. **Function exits**
3. **Entries to interrupt service routines**
4. **Exits of interrupt service routines**
5. **Entries to TRAP handlers**
6. **Exits of TRAP handlers**



min	shortest time within the function including all subfunctions and traps
max	longest time within the function including all subfunctions and traps

Summary

The nesting analysis provides more details on the structure and the timing of the program run, but it is much more sensitive then the flat analysis. Missing or tricky function entries/exits may require additional setups before nesting analysis can be used.

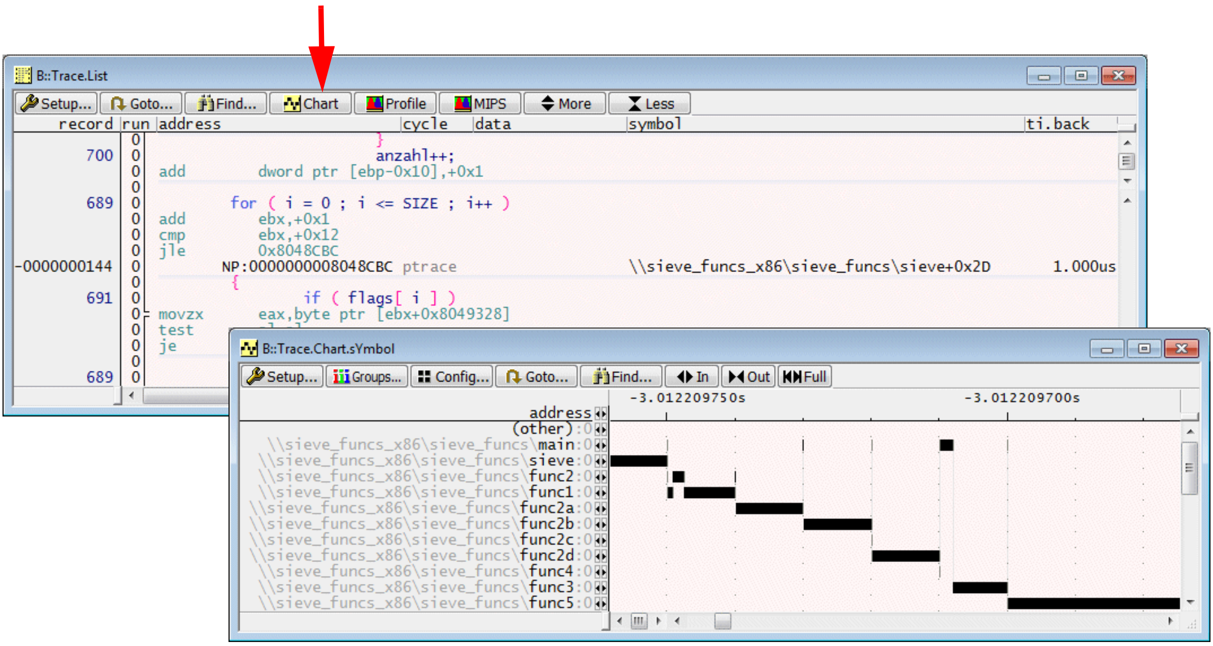
Flat Function-Runtime Analysis

NOTE:	As long a TRACE32 does not support Synchronisation Time, cycle accurate tracing should be disabled for all kind of runtime measurement.

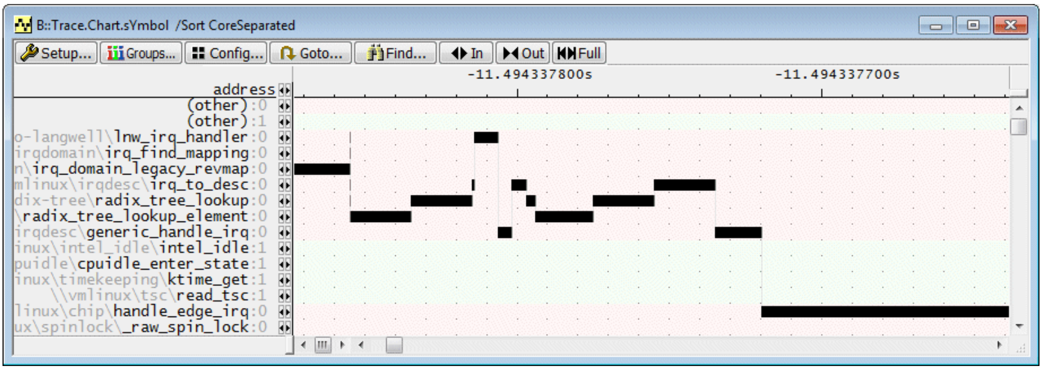
Function Time Chart

Default Time Chart

Pushing the **Chart** button in the **Trace.List** window
opens a **Trace.Chart.sYmbol** window

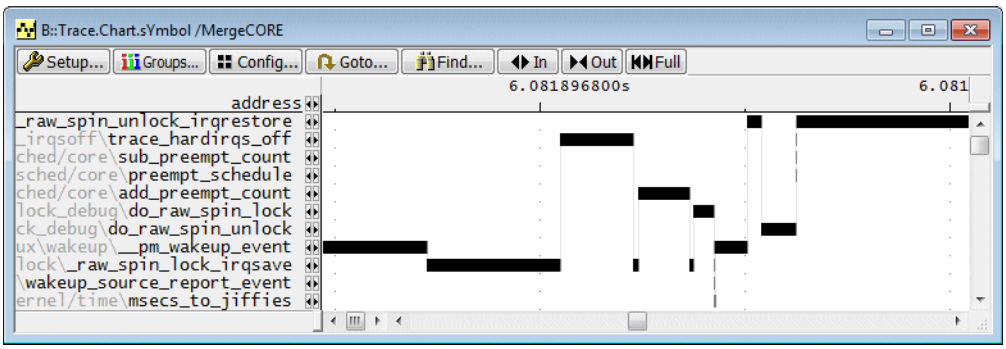


Trace.Chart.sYmbol [/SplitCore /Sort CoreTogether]	Flat function run-time analysis - graphical display - split the result per core - sort results per core and then per recording order



Trace.Chart.sYmbol [/SplitCORE] /Sort CoreSeparated

Flat function run-time analysis
- graphical display
- split the result per core
- sort the results per recording order

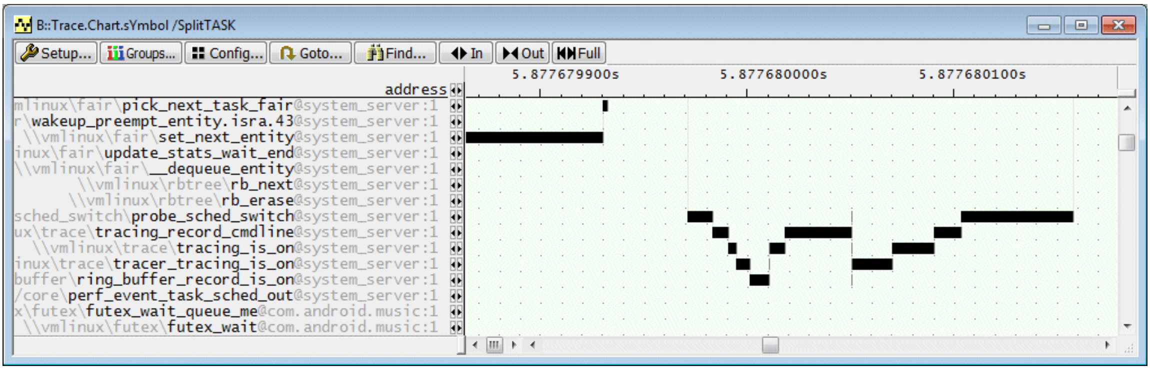


Trace.Chart.sYmbol /MergeCORE

Flat function run-time analysis
- graphical display
- merge the results of all cores

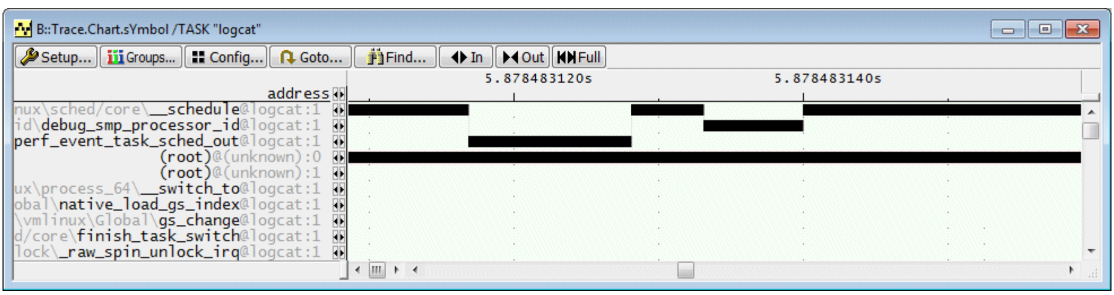
Trace information is analyzed independently for each core. The time chart summarizes these results to a single result.

Trace.Chart.sYmbol /SplitTASK	Display function time chart including process information (OS, OS+MMU only)

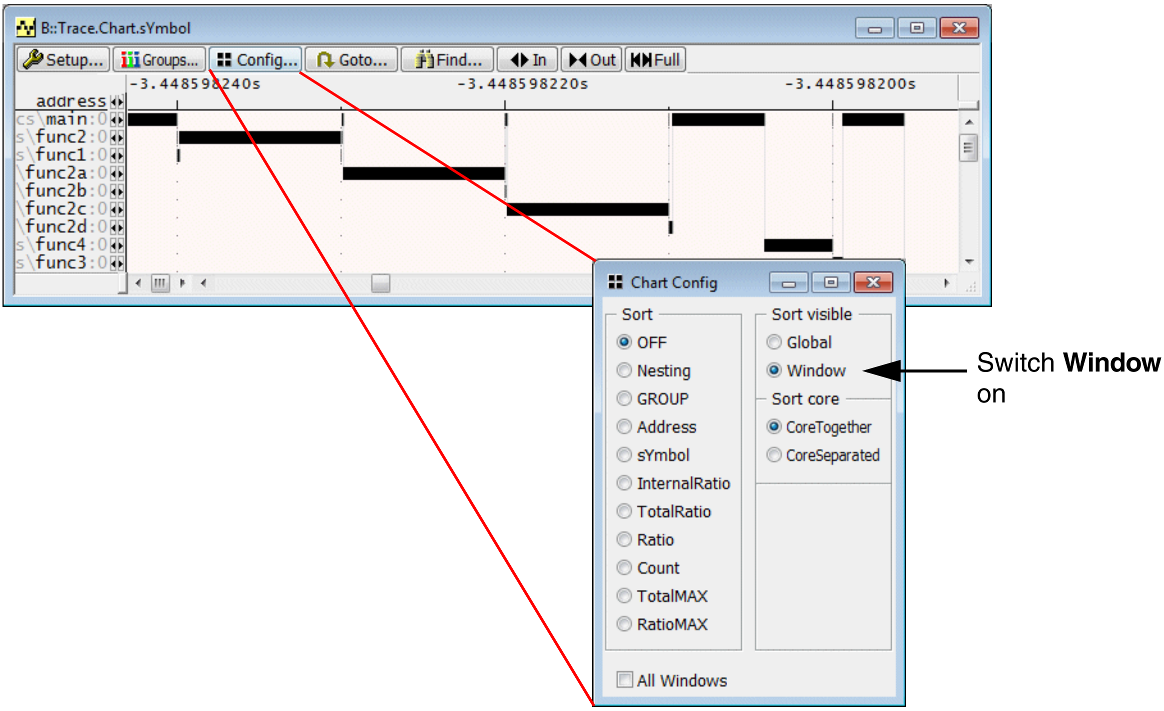


@<*task_name*>	Process name information
@(unknown)	Function was recorded before first process switch information was recorded

Trace.Chart.sYmbol /TASK <*name*>	Display function time chart for specified process (OS, OS+MMU only)



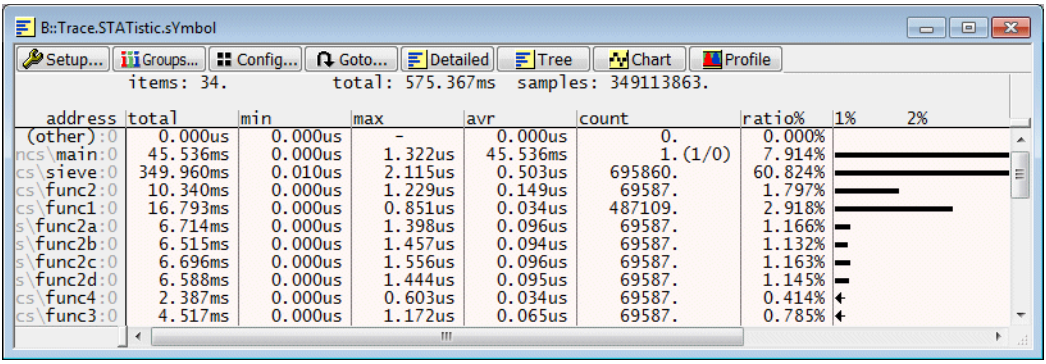
@<*task_name*>	Process name information
(root)@(unknown)	Everything outside of the specified process.



If **Window** in the **Sort visible** field is switched ON in the **Chart Config** window, the functions that are active at the selected point of time are visualized in the scope of the **Trace.Chart.sYmbol** window. This is helpful especially if you scroll horizontally.

Function Run-time Statistic

Analog to the timing diagram there is also a numerical analysis.



B::Trace.STATistic.sYmbol

Setup... Groups... Config... Goto... Detailed Tree Chart Profile

items: 34. total: 575.367ms samples: 349113863.

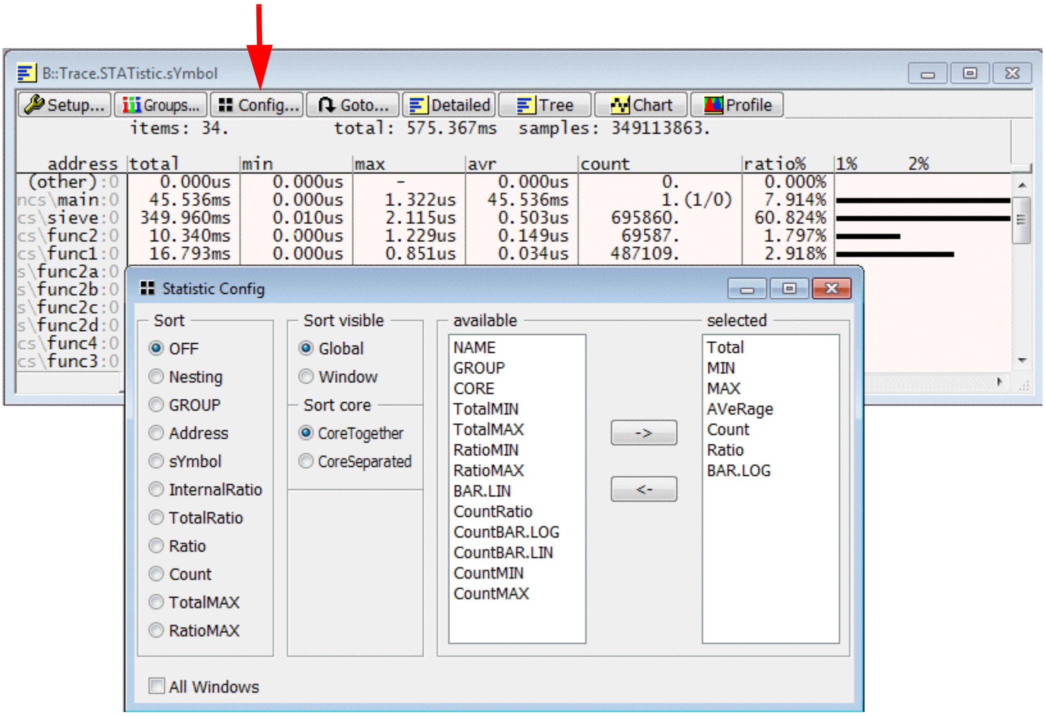
address	total	min	max	avr	count	ratio%
(other):0	0.000us	0.000us	-	0.000us	0.	0.000%
ncs\main:0	45.536ms	0.000us	1.322us	45.536ms	1.(1/0)	7.914%
cs\sieve:0	349.960ms	0.010us	2.115us	0.503us	695860.	60.824%
cs\func2:0	10.340ms	0.000us	1.229us	0.149us	69587.	1.797%
cs\func1:0	16.793ms	0.000us	0.851us	0.034us	487109.	2.918%
s\func2a:0	6.714ms	0.000us	1.398us	0.096us	69587.	1.166%
s\func2b:0	6.515ms	0.000us	1.457us	0.094us	69587.	1.132%
s\func2c:0	6.696ms	0.000us	1.556us	0.096us	69587.	1.163%
s\func2d:0	6.588ms	0.000us	1.444us	0.095us	69587.	1.145%
cs\func4:0	2.387ms	0.000us	0.603us	0.034us	69587.	0.414%
cs\func3:0	4.517ms	0.000us	1.172us	0.065us	69587.	0.785%

survey	
item	number of recorded functions/symbol regions
total	time period recorded by the trace
samples	total number of recorded changes of functions/symbol regions (program flow continuously in the address range of a function/symbol region)

function details	
address	function/symbol region name (here per core) **(other)** program sections that can not be assigned to a function/symbol region
total	time period in the function/symbol region during the recorded time period
min	shortest time continuously in the address range of the function/symbol region
max	longest time continuously in the address range of the function/symbol region
avr	average time continuously in the address range of the function/symbol region (calculated by total/count)

count	number of new entries (start address executed) into the address range of the function/symbol region
ratio	ratio of time in the function/symbol region with regards to the total time period recorded

Pushing the **Config** button provides the possibility to specify a different column layout and a different sorting criterion for the address column.
By default the functions/symbol regions are sorted by their recording order.



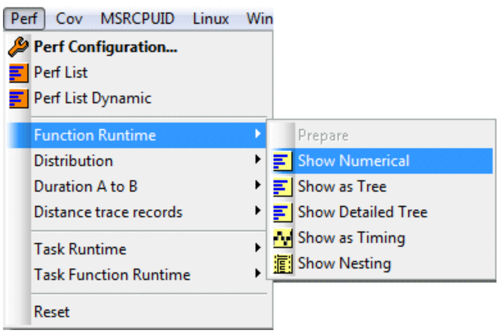
Further Commands

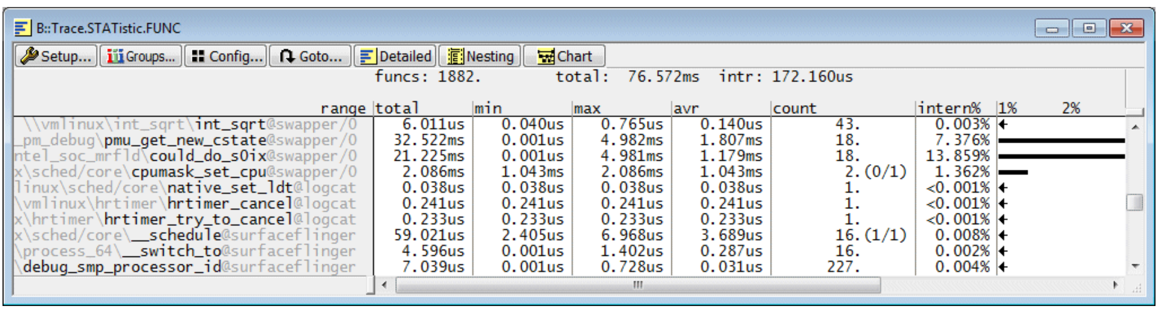
Trace.PROfileChart.sYmbol	Display dynamic program behavior graphically.
MIPS.PROfileChart.sYmbol	Display MIPS for all program symbols graphically.
MIPS.STATistic.sYmbol	Display MIPS for all program symbols numerically.

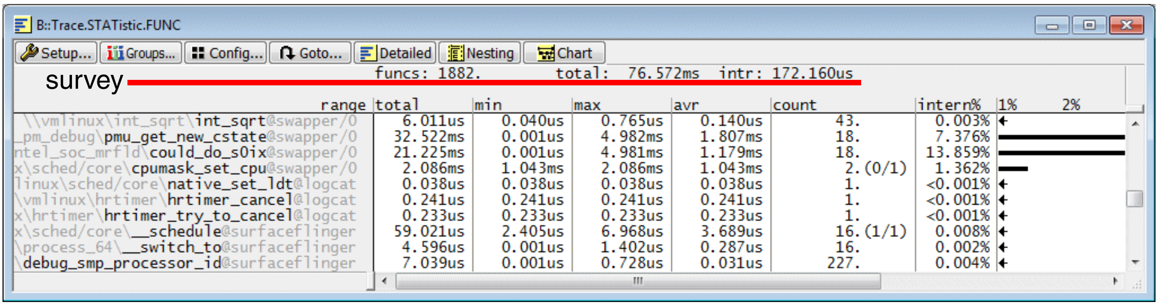
Nesting Function Analysis OS

Function nesting analysis for OS requires that OS-aware debugging is configured. For more information refer to **“OS-aware Debugging”** (glossary.pdf).

Trace.STATistic.Func	Nesting function run-time analysis - numeric display - core information is discarded exceptions are the @(unknown) task and the @(interrupt) task



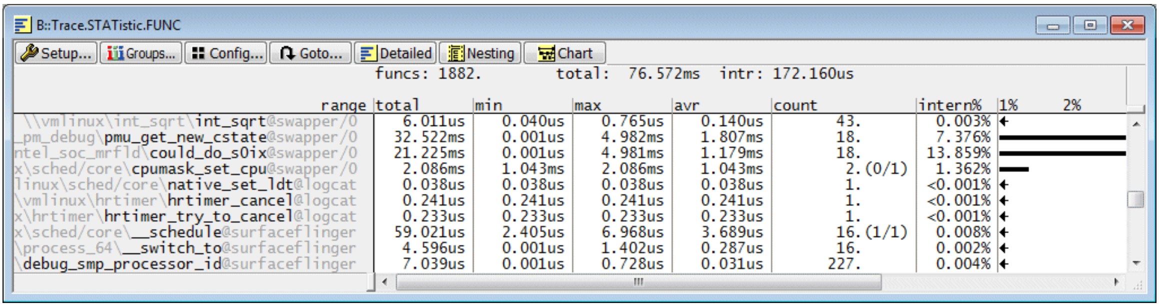




survey	
func	number of functions in the trace
total	total measurement time
intr	total time in interrupt service routines

survey (issue indication)	
stopped: *<time>*	The analyzed trace recording contains program stops. *<time>* indicates the total time the program execution was stopped.
<number> **problems**	The nesting analysis contains problems. Please contact **support@lauterbach.com**.
<number> **workarounds**	The nesting analysis contains issues, but TRACE32 found solutions for them. It is recommended to perform a sanity check on the proposed solutions.
stack overflow at *<record>*	The nesting analysis exceeds the nesting level 200. It is highly likely that the function exit for an often called function is missing. The command **Trace.STATistic.TREE** can help you to identify the function. If you need further help please contact **support@lauterbach.com**.
stack underflow at *<record>*	The nesting analysis exceeds the nesting level 200. It is highly likely that the function entry for an often executed function is missing. The command **Trace.STATistic.TREE** can help you to identify the function. If you need further help please contact **support@lauterbach.com**.

range Column



columns	
range (NAME)	function name, sorted by their recording order as default

```
\\vmlinux\hrtimer\hrtimer_cancel@logcat
```

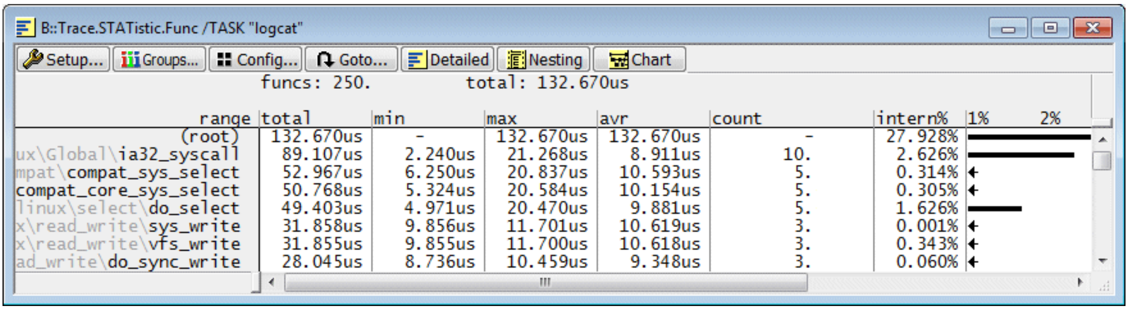
HLL function **hrtimer_cancel** running in process **@logcat**.

Please be aware that no core information is provided for processes and their functions.

Nesting function run-time analysis can also be performed per process.

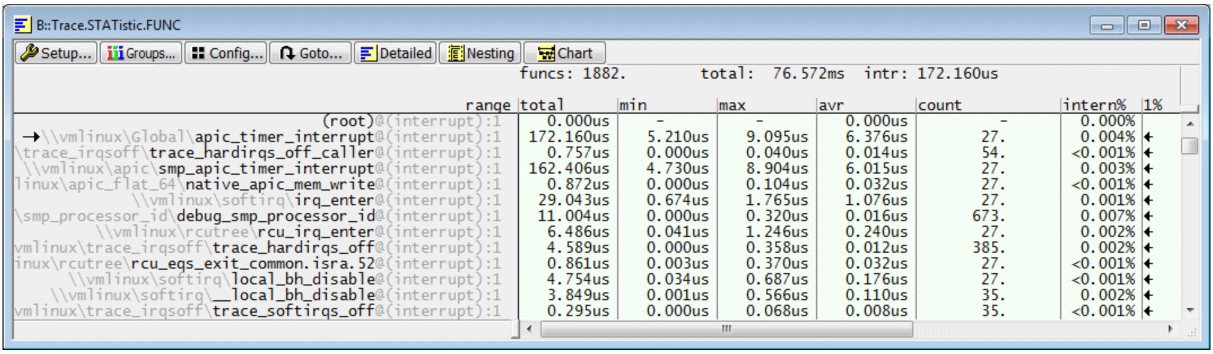
Trace.STATistic.Func /TASK *<task_magic>* | *<task_name>* | *<task_id>*

```
Trace.STATistic.Func /TASK "logcat"
```



Interrupt Functions

Interrupt service routines are assigned to the @(interrupt) task. Core information is provided for the @(interrupt) task.



B::Trace.STATistic.FUNC

Setup... Groups... Config... Goto... Detailed Nesting Chart

funcs: 1882. total: 76.572ms intr: 172.160us

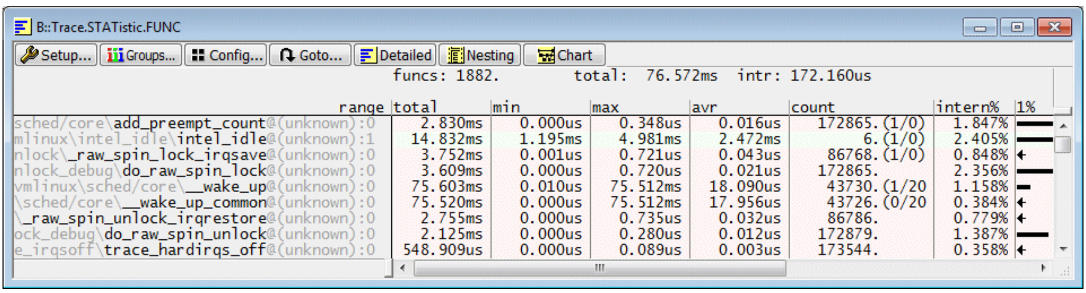
range	total	min	max	avr	count	intern%	1%
(root)@(interrupt):1	0.000us	-	-	0.000us	-	0.000%	
→\\vmlinux\Global\apic_timer_interrupt@(interrupt):1	172.160us	5.210us	9.095us	6.376us	27.	0.004%	
\trace_irqsoff\trace_hardirqs_off_caller@(interrupt):1	0.757us	0.000us	0.040us	0.014us	54.	<0.001%	
\\vmlinux\apic\smp_apic_timer_interrupt@(interrupt):1	162.406us	4.730us	8.904us	6.015us	27.	0.003%	
linux\apic_flat_64\native_apic_mem_write@(interrupt):1	0.872us	0.000us	0.104us	0.032us	27.	<0.001%	
\\vmlinux\softirq\irq_enter@(interrupt):1	29.043us	0.674us	1.765us	1.076us	27.	0.001%	
\smp_processor_id\debug_smp_processor_id@(interrupt):1	11.004us	0.000us	0.320us	0.016us	673.	0.007%	
\\vmlinux\rcutree\rcu_irq_enter@(interrupt):1	6.486us	0.041us	1.246us	0.240us	27.	0.002%	
vmlinux\trace_irqsoff\trace_hardirqs_off@(interrupt):1	4.589us	0.000us	0.358us	0.012us	385.	0.002%	
inux\rcutree\rcu_eqs_exit_common.isra.52@(interrupt):1	0.861us	0.003us	0.370us	0.032us	27.	<0.001%	
\\vmlinux\softirq\local_bh_disable@(interrupt):1	4.754us	0.034us	0.687us	0.176us	27.	<0.001%	
\\vmlinux\softirq__local_bh_disable@(interrupt):1	3.849us	0.001us	0.566us	0.110us	35.	0.002%	
vmlinux\trace_irqsoff\trace_softirqs_off@(interrupt):1	0.295us	0.000us	0.068us	0.008us	35.	<0.001%	

An arrow before the interrupt function indicates the function executed after the interrupt occurred:

`→\\vmlinux\Global\apic_timer_interrupt@(interrupt):1`

The unknown Task

All function recorded before the first process switch is recorded are assigned to the @(unknown) task. Core information is provided for the @(unknown) task.

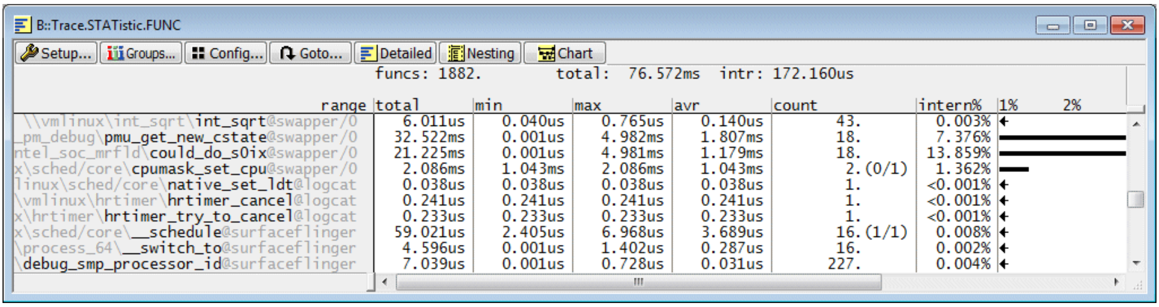


B::Trace.STATistic.FUNC

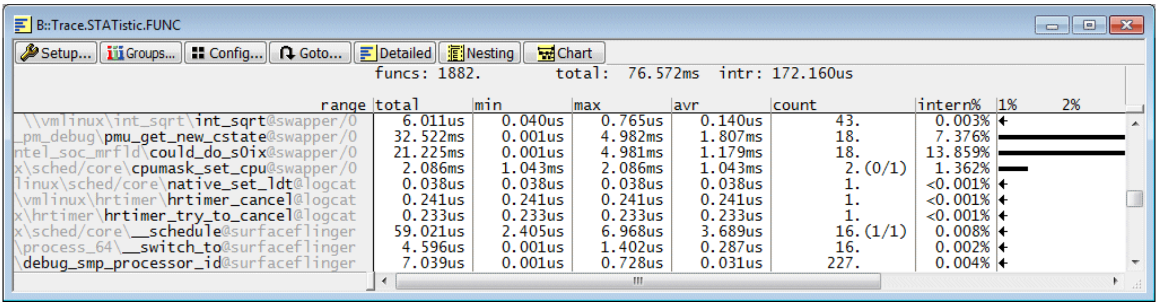
Setup... Groups... Config... Goto... Detailed Nesting Chart

funcs: 1882. total: 76.572ms intr: 172.160us

range	total	min	max	avr	count	intern%	1%
sched/core\add_preempt_count@(unknown):0	2.830ms	0.000us	0.348us	0.016us	172865.(1/0)	1.847%	
mlinux\intel_idle\intel_idle@(unknown):1	14.832ms	1.195ms	4.981ms	2.472ms	6.(1/0)	2.405%	
nlock_raw_spin_lock_irqsave@(unknown):0	3.752ms	0.001us	0.721us	0.043us	86768.(1/0)	0.848%	
nlock_debug\do_raw_spin_lock@(unknown):0	3.609ms	0.000us	0.720us	0.021us	172865.	2.356%	
vmlinux\sched/core__wake_up@(unknown):0	75.603ms	0.010us	75.512ms	18.090us	43730.(1/20	1.158%	
\sched/core__wake_up_common@(unknown):0	75.520ms	0.000us	75.512ms	17.956us	43726.(0/20	0.384%	
_raw_spin_unlock_irqrestore@(unknown):0	2.755ms	0.000us	0.735us	0.032us	86786.	0.779%	
ock_debug\do_raw_spin_unlock@(unknown):0	2.125ms	0.000us	0.280us	0.012us	172879.	1.387%	
e_irqsoff\trace_hardirqs_off@(unknown):0	548.909us	0.000us	0.089us	0.003us	173544.	0.358%	



columns (cont.)	
total	total time within the function
min	shortest time between function entry and exit, time spent in interrupt service routines is excluded No **min** time is displayed if a function exit was never executed.
max	longest time between function entry and exit, time spent in interrupt service routines is excluded
avr	average time between function entry and exit, time spent in interrupt service routines is excluded



columns (cont.)	
count	number of times within the function

If function entries or exits are missing, this is displayed in the following format:

<times within the function >. (*<number of missing function entries>***/***<number of missing function exits>*).

3671.(0/1)

Interpretation examples:

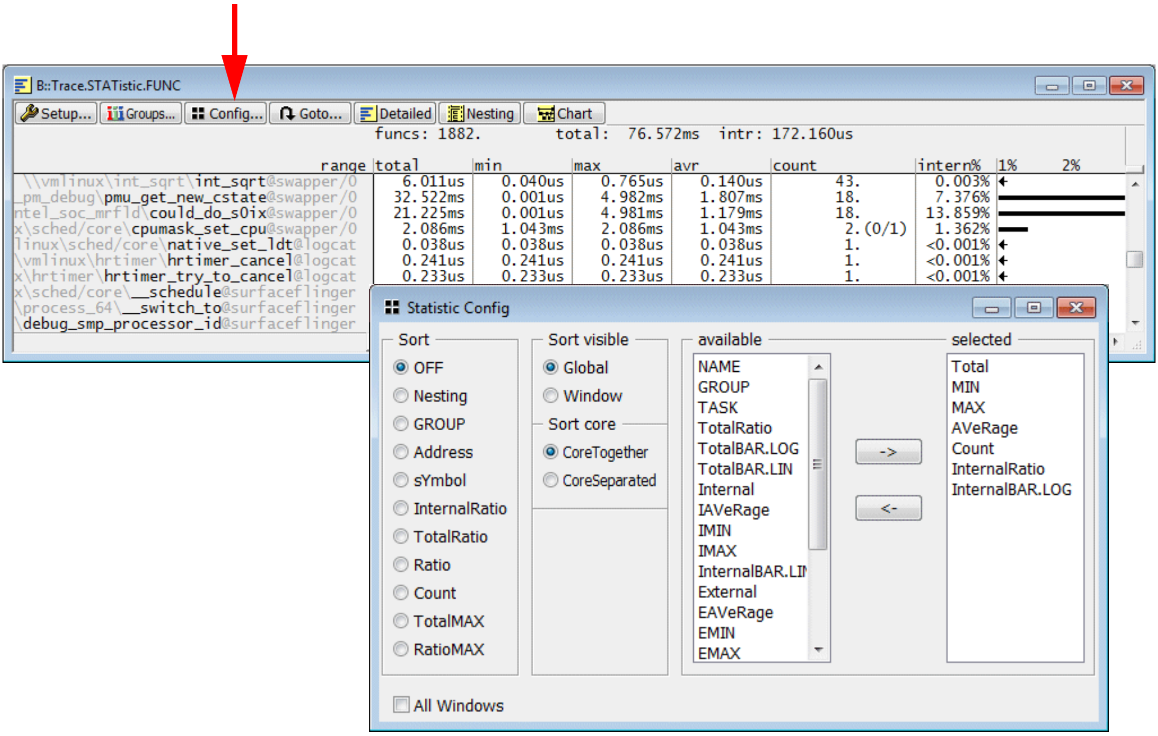
1. 2. (2/0): 2 times within the function, 2 function entries missing
2. 4. (0/3): 4 times within the function, 3 function exits missing
3. 11. (1/1): 11 times within the function, 1 function entry and 1 function exit is missing.

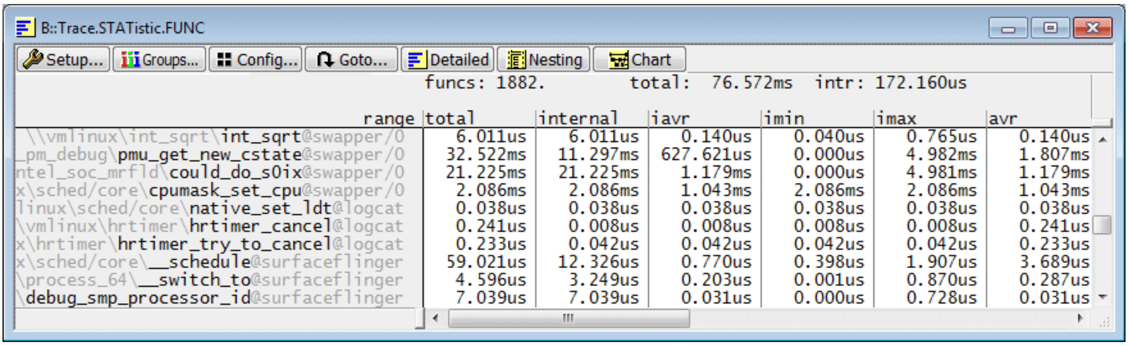
	If the number of missing function entries or exits is higher the 1 the analysis performed by the command **Trace.STATistic.Func** might fail due to nesting problems. A detailed view to the trace contents is recommended.

columns (cont.)	
intern% (InternalRatio, InternalBAR.LOG)	ratio of time within the function without subfunctions, TRAP handlers, interrupts (net time)

Net Results

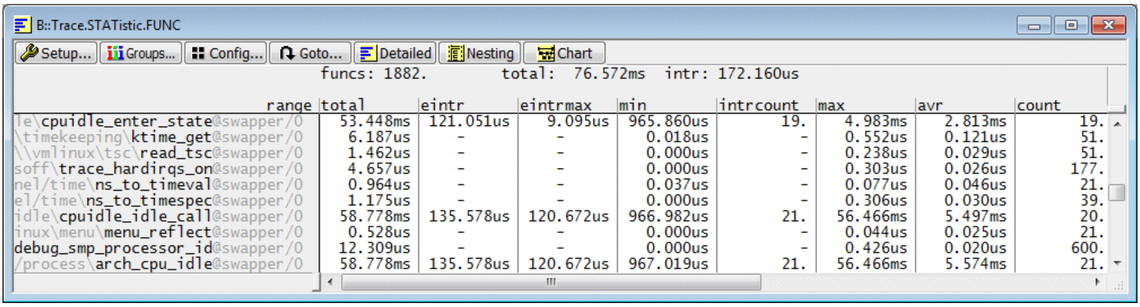
Pushing the **Config…** button allows to display additional columns





columns (cont.) - times only in function	
Internal	total time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IAVeRage	average time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IMIN	shortest time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IMAX	longest time spent in the function between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
InternalRatio	*<Internal time of function>/<Total measurement time>* as a numeric value.
InternalBAR	*<Internal time of function>/<Total measurement time>* graphically.

Interrupt Details



B::Trace.STATistic.FUNC

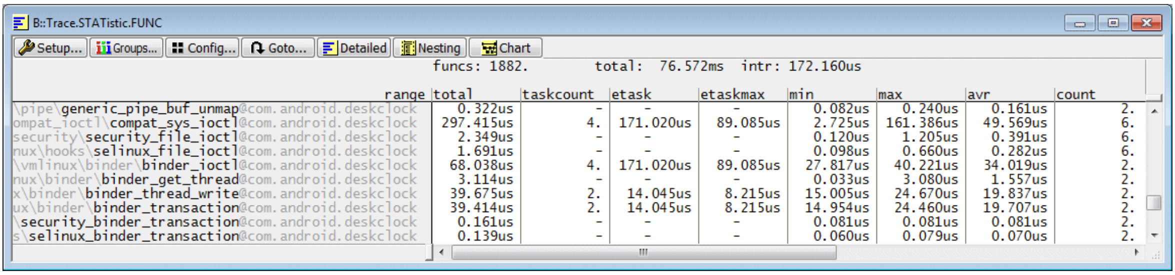
Setup... Groups... Config... Goto... Detailed Nesting Chart

funcs: 1882. total: 76.572ms intr: 172.160us

range	total	eintr	eintrmax	min	intrcount	max	avr	count
le\cpuidle_enter_state@swapper/0	53.448ms	121.051us	9.095us	965.860us	19.	4.983ms	2.813ms	19.
\timekeeping\ktime_get@swapper/0	6.187us	-	-	0.018us	-	0.552us	0.121us	51.
\\vmlinux\tsc\read_tsc@swapper/0	1.462us	-	-	0.000us	-	0.238us	0.029us	51.
soff\trace_hardirqs_on@swapper/0	4.657us	-	-	0.000us	-	0.303us	0.026us	177.
nel/time\ns_to_timeval@swapper/0	0.964us	-	-	0.037us	-	0.077us	0.046us	21.
el/time\ns_to_timespec@swapper/0	1.175us	-	-	0.000us	-	0.306us	0.030us	39.
idle\cpuidle_idle_call@swapper/0	58.778ms	135.578us	120.672us	966.982us	21.	56.466ms	5.497ms	20.
inux\menu\menu_reflect@swapper/0	0.528us	-	-	0.000us	-	0.044us	0.025us	21.
debug_smp_processor_id@swapper/0	12.309us	-	-	0.000us	-	0.426us	0.020us	600.
/process\arch_cpu_idle@swapper/0	58.778ms	135.578us	120.672us	967.019us	21.	56.466ms	5.574ms	21.

columns (cont.) - interrupt times	
ExternalINTR	total time the function was interrupted
ExternalINTRMAX	max. time one function pass was interrupted
INTRCount	number of interrupts that occurred during the function run-time

Time in Other Tasks



B::Trace.STATistic.FUNC

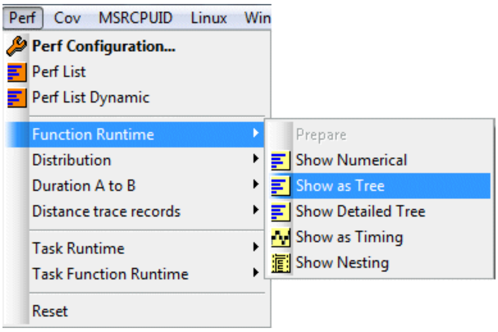
Setup... Groups... Config... Goto... Detailed Nesting Chart

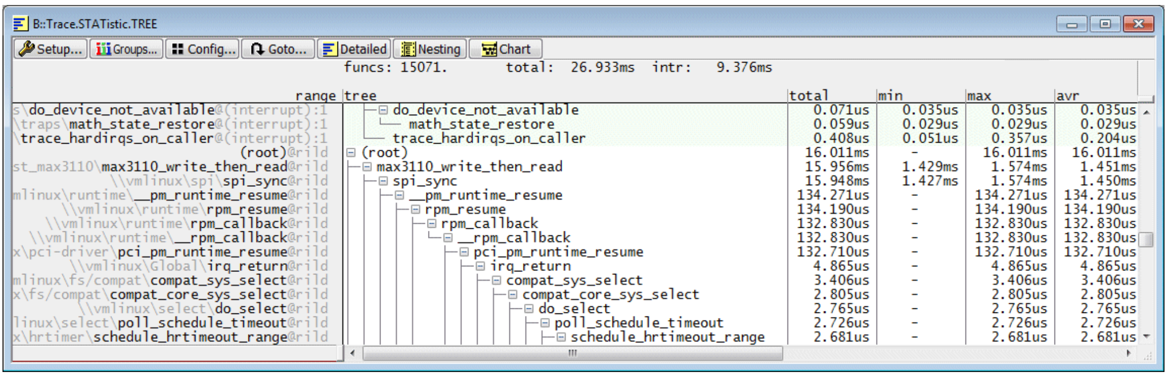
funcs: 1882. total: 76.572ms intr: 172.160us

range	total	taskcount	etask	etaskmax	min	max	avr	count
\pipe\generic_pipe_buf_unmap@com.android.deskclock	0.322us	-	-	-	0.082us	0.240us	0.161us	2.
ompat_ioctl\compat_sys_ioctl@com.android.deskclock	297.415us	4.	171.020us	89.085us	2.725us	161.386us	49.569us	6.
security\security_file_ioctl@com.android.deskclock	2.349us	-	-	-	0.120us	1.205us	0.391us	6.
nux\hooks\selinux_file_ioctl@com.android.deskclock	1.691us	-	-	-	0.098us	0.660us	0.282us	6.
\vmlinux\binder\binder_ioctl@com.android.deskclock	68.038us	4.	171.020us	89.085us	27.817us	40.221us	34.019us	2.
nux\binder\binder_get_thread@com.android.deskclock	3.114us	-	-	-	0.033us	3.080us	1.557us	2.
x\binder\binder_thread_write@com.android.deskclock	39.675us	2.	14.045us	8.215us	15.005us	24.670us	19.837us	2.
ux\binder\binder_transaction@com.android.deskclock	39.414us	2.	14.045us	8.215us	14.954us	24.460us	19.707us	2.
\security_binder_transaction@com.android.deskclock	0.161us	-	-	-	0.081us	0.081us	0.081us	2.
s\selinux_binder_transaction@com.android.deskclock	0.139us	-	-	-	0.060us	0.079us	0.070us	2.

columns - process related information	
TASKCount	number of tasks that interrupt the function/task
ExternalTASK	total time in other tasks
ExternalTASKMAX	max. time 1 function/task pass was interrupted by a task

Trace.STATistic.TREE — Nesting function run-time analysis - tree display





It is also possible to get a task/process-specific tree.

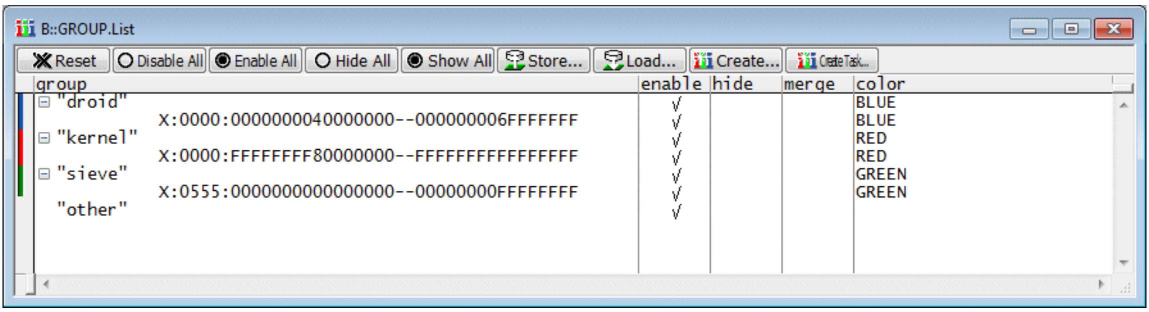
```
Trace.STATistic.TREE /TASK "rild"
```

GROUPs for OS-aware Tracing

TRACE32 PowerView provides the GROUP command to structure the trace evaluation.

If you use a target OS such a Linux, the following groups are created by the Lauterbach scripts and Lauterbach OS menus:

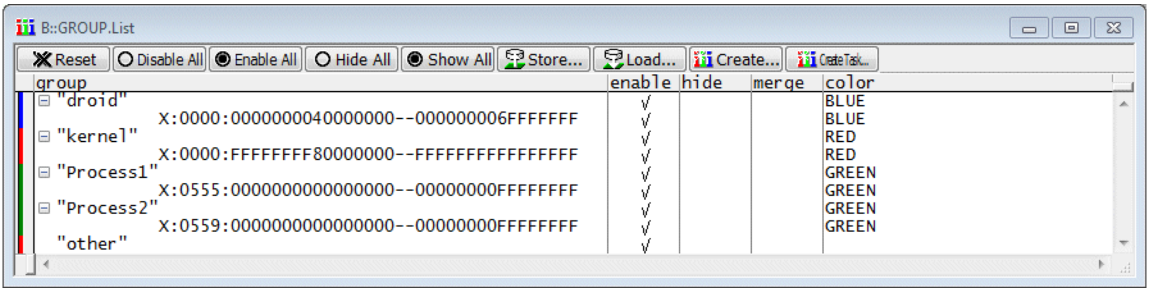
- A GROUP “kernel”, color RED, to mark the OS kernel.
- A GROUP “droid”, color BLUE, to mark virtual machine byte code e.g. Android/Dalvik.
- A GROUP *<process_name>* per process, color GREEN.
- A GROUPs *<module_name>* per kernel module, color YELLOW.



A group can have the following statuses:

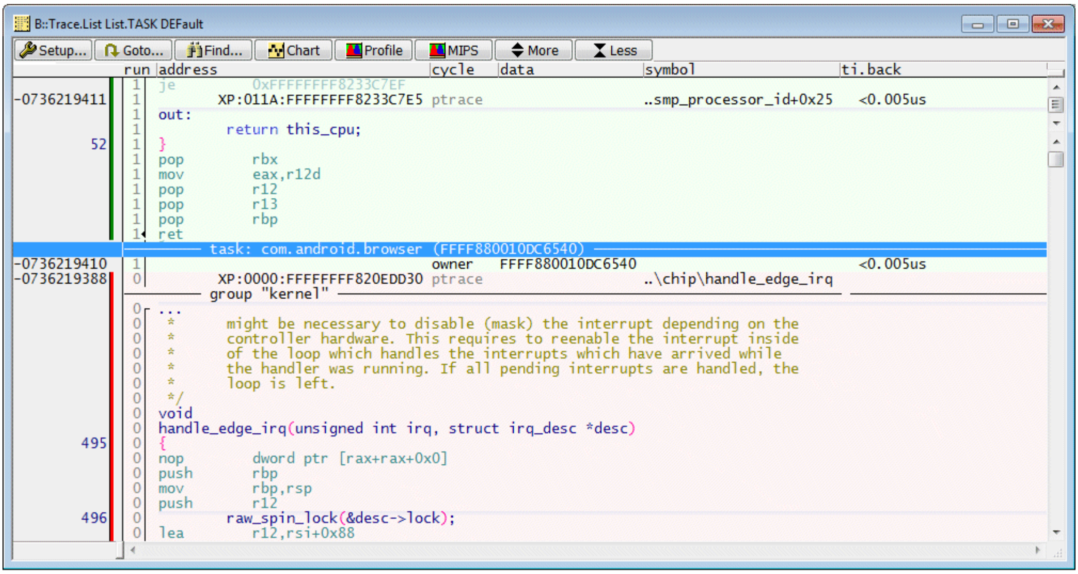
- enable
- enable + merge
- enable + hide

GROUP Status ENable

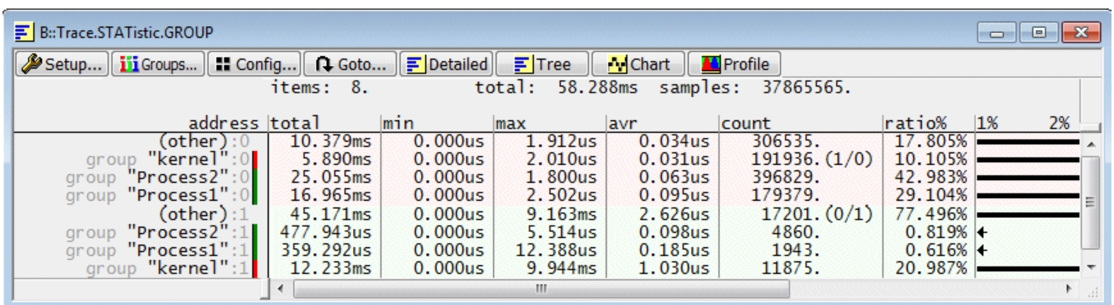


If a GROUP is enabled:

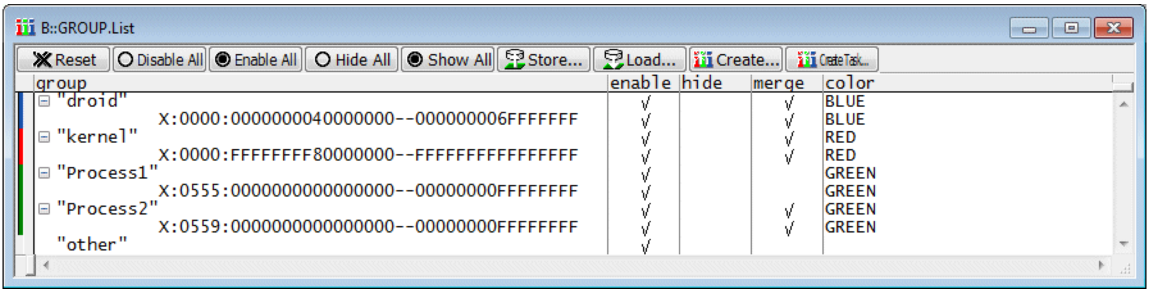
- The trace information recorded for the group members is marked with the color assigned to the group.



- Group-based trace analyses commands are provided e.g. **Trace.STATistic.GROUP**.

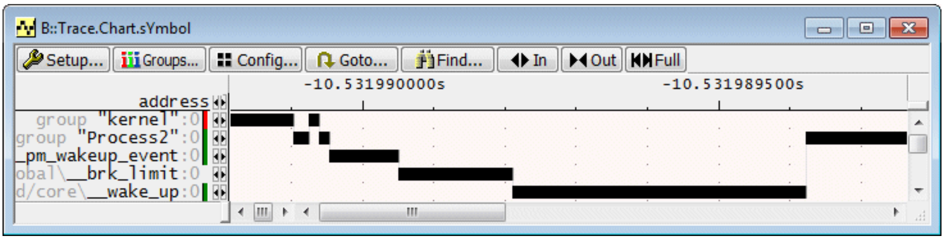


GROUP Status ENable+Merge

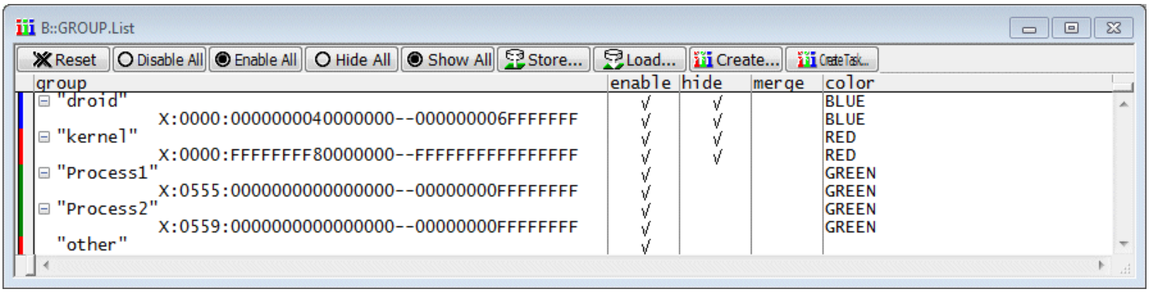


If a GROUP is enabled and merge is checked:

- The group represents its members in all trace analysis windows. No details about group members are displayed.

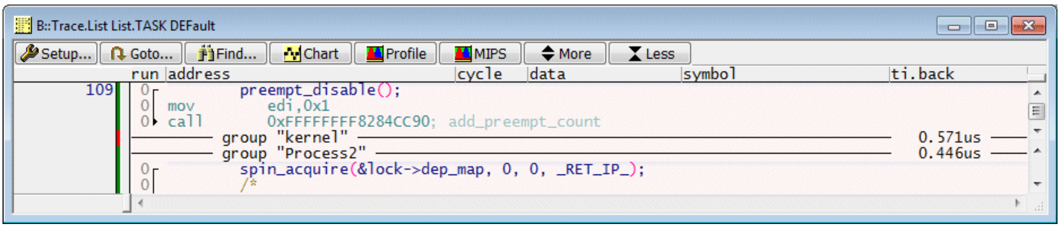


GROUP Status Enable+HIDE



If a GROUP is enabled and hide is checked:

- The group represents its members in all trace analysis windows. No details about group members are displayed (same as merge checked).
- The trace information recorded for the group members is hidden in the **Trace.List** window.



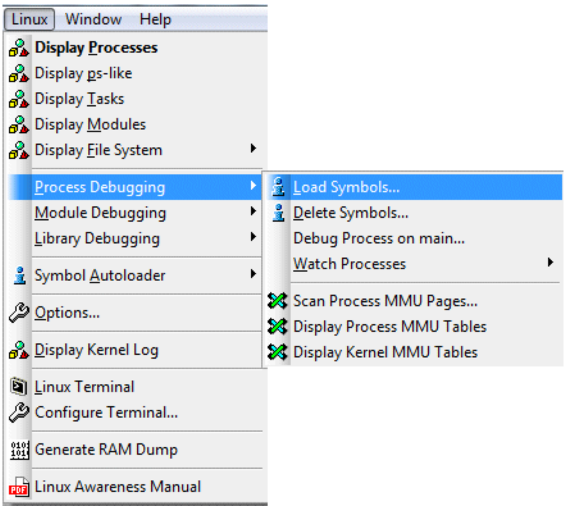
GROUP Creation

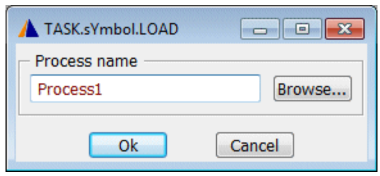
The GROUPs “kernel” and “droid” are typically created in the start-up script that sets up the OS-aware debugging.

GROUP.Create *<group_name> <address_range>* **/***<color>*

```
GROUP.Create "kernel" XP:0xffffffff80000000--0xffffffffffffffff /RED

GROUP.Create "droid"  XP:0x0000000040000000--0x000000006FFFFFFF /BLUE
```

<process> or *<module>* GROUPs are created when their symbol information is loaded.





For more details about GROUPs refer to the **GROUP** command group.