

# Training Intel® Processor Tracing

Release 09.2024

MANUAL



# Training Intel® Processor Tracing

---

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Training .....	
Training Intel® x86/x64 .....	
Training Intel® Processor Tracing .....	1
Protocol Description .....	6
Basic Trace Packets	6
OS-Aware Tracing	7
Time Information	8
Tool Timestamp (POWER TRACE II / POWER TRACE III only)	8
CycleAccurate Tracing	10
Synchronization Time	10
Trace Configuration .....	11
Off-chip Trace	11
SDRAM Trace	18
Trace Errors	22
ERRORS	22
TARGET FIFO OVERFLOW	25
TRACE32 Abstractions	27
SystemTrace	27
(Core) Trace	29
Displaying the Trace Contents .....	31
Influencing Factors on the Trace Information	31
Settings in the TRACE32 Trace Configuration Window	32
States of the Trace	43
The AutoInit Command	44
Basic Display Commands	45
Default Listing	45
Basic Formatting	47
Correlating the Trace Listing with the Source Listing	48
Browsing through the Trace Buffer	49
Find a Specific Record	50
Display Items	51
Default Display Items	51
Further Display Items	54
Belated Trace Analysis	60

Save the Trace Information to an ASCII File	61
Postprocessing with TRACE32 Instruction Set Simulator	62
Export STP Byte Stream	67
<b>Trace Control by Filters</b> .....	<b>68</b>
TraceEnable	69
TraceOFF	71
<b>OS-Aware Tracing</b> .....	<b>73</b>
Process Switch Packets	73
Program Flow and Process Switches	75
Process Runtime Analysis	76
Time Chart	77
Statistic	78
Find Process Switches in the Trace	79
OS-aware Filtering	81
Filtering by Privilege Level	81
Filtering by Process	82
Filter on Function executed by Process	84
Belated Analysis	86
Example for Linux	86
<b>Trace-based Debugging (CTS)</b> .....	<b>88</b>
Setup	88
Get Started	89
Forward and Backward Debugging	92
Forward Debugging	92
Backward Debugging	92
CTS Technique	93
<b>Function Run-Time Analysis - Basic Concept</b> .....	<b>94</b>
Software under Analysis (no OS, OS or OS+MMU)	94
Flat vs. Nesting Analysis	94
Basic Knowledge about Flat Analysis	95
Basic Knowledge about Nesting Analysis	96
Summary	98
<b>Flat Function-Runtime Analysis</b> .....	<b>99</b>
Function Time Chart	99
Default Time Chart	99
Core Options	100
TASK Options	101
Function Run-time Statistic	103
Further Commands	104
<b>Nesting Function Analysis OS</b> .....	<b>105</b>
Survey	106

range Column	107
Default Results	110
Net Results	112
Interrupt Details	114
Time in Other Tasks	115
Tree Display	116
<b>Structure your Trace Evaluation .....</b>	<b>117</b>
GROUPs for OS-aware Tracing	117
GROUP Status ENable	118
GROUP Status ENable+Merge	119
GROUP Status Enable+HIDE	120
GROUP Creation	121

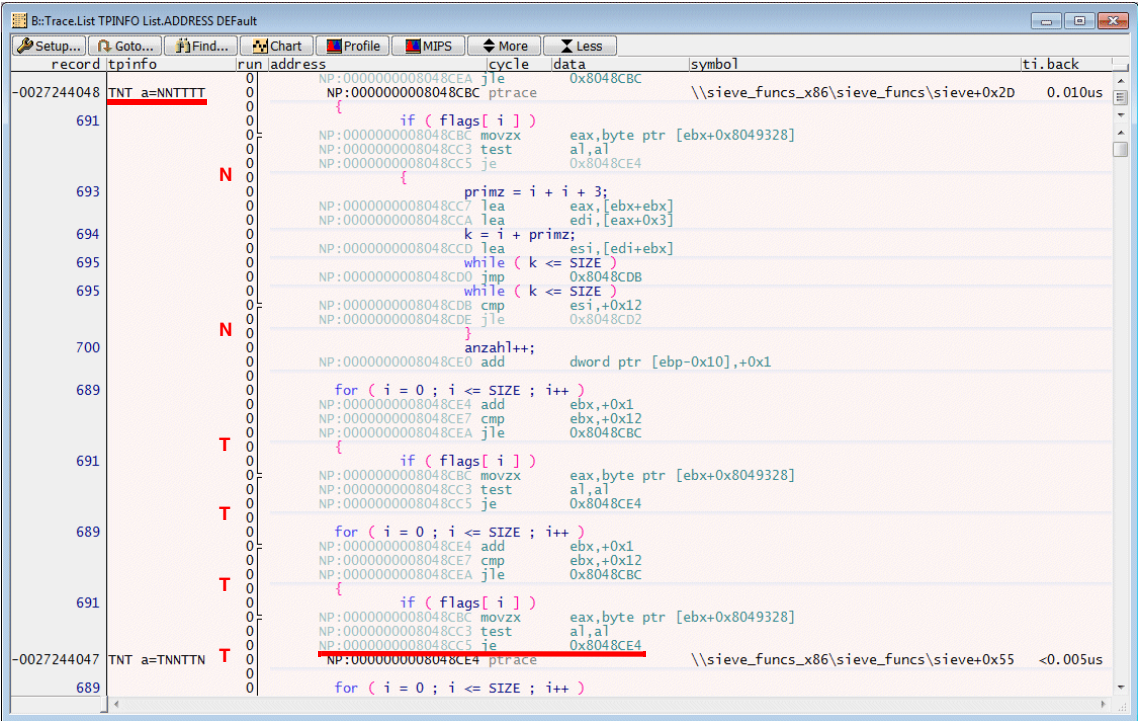


Basic Trace Packets

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.

record	tpinfo	run address	cycle	data	symbol	ti.back
-0027232833	Target IP 4b Zext BLIP=0x8048C80	NP:0000000008048CEA	0	ptrace	0x8048CBC	.. \sieve+0x5D <0.005us
704			0			
705			0			
-0027232827	TNT a=TTTTTT	NP:0000000008048C80	0	ptrace	.. \main+0x29F	0.010us

## 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.

record	tpinfo	run address	cycle	data	symbol	ti.back
-0141036641	task: sieve (FFFF88003B9B0000)	XP:05E8:FFFFFFFF8284272A	0	ptrace	.. \ux\sched\core\_schedule+0x2EA	<0.005us
-0141036463	PIP CR0, PG=1 CR3=0x8C49000		0			1.040us
41	TNT a=NTNTNT		0			

# Time Information

## 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.

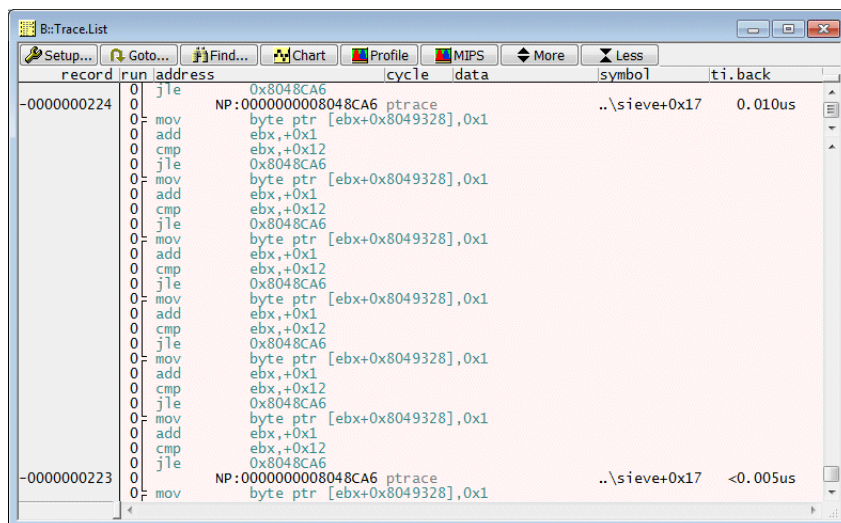
8 trace record have always an identical timestamp. There are two reasons for this:

- 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



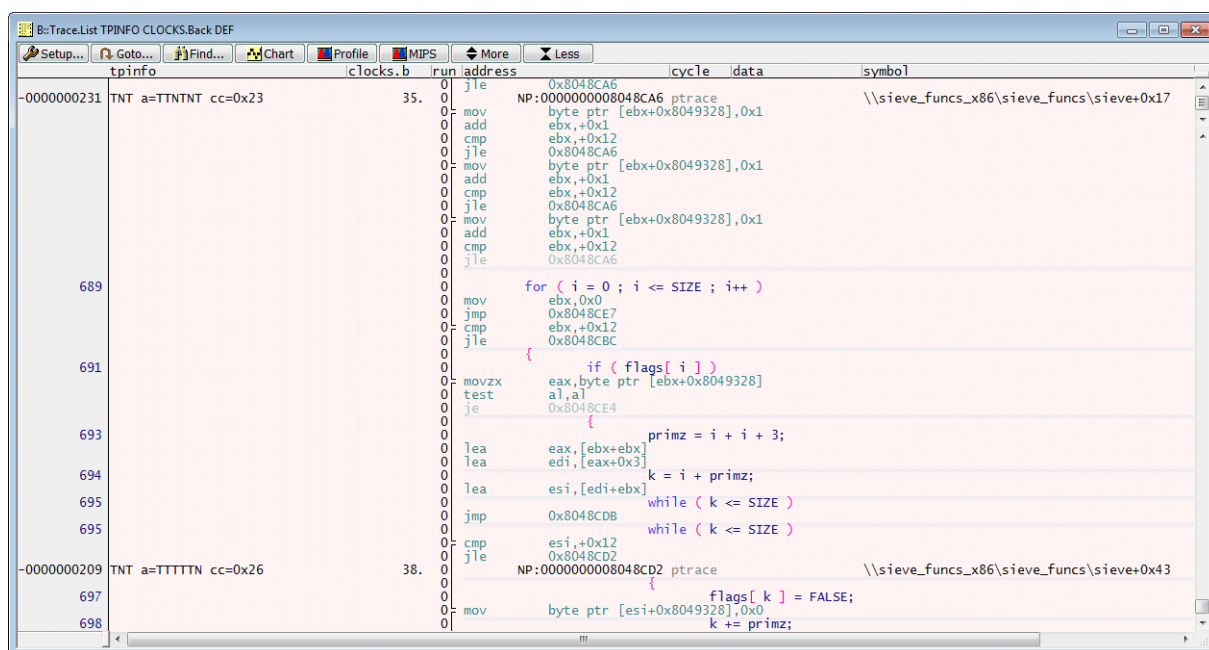
In the standard trace display timestamp information is displayed for the first record with the new timestamp. All following records with an identical timestamp show <0.005us.



record	run	address	cycle	data	symbol	ti.back
-0000000224	0	jle 0x8048CA6				
	0	NP:0000000008048CA6 ptrace			..\sieve+0x17	0.010us
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	mov byte ptr [ebx+0x8049328],0x1				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CA6				
	0	NP:0000000008048CA6 ptrace			..\sieve+0x17	<0.005us
-0000000223	0	mov byte ptr [ebx+0x8049328],0x1				

## 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.



The screenshot shows the Intel PT Trace List window with the following columns: tpinfo, clocks.b, run, address, cycle, data, and symbol. The trace data is as follows:

tpinfo	clocks.b	run	address	cycle	data	symbol
-0000000231	TNT a=TTNTNT cc=0x23	35.	0 jle 0x8048CA6			
		0	NP:000000008048CA6 ptrace			\\sieve_funcs_x86\sieve_funcs\sieve+0x17
		0	mov byte ptr [ebx+0x8049328],0x1			
		0	add ebx,+0x1			
		0	cmp ebx,+0x12			
		0	jle 0x8048CA6			
		0	mov byte ptr [ebx+0x8049328],0x1			
		0	add ebx,+0x1			
		0	cmp ebx,+0x12			
		0	jle 0x8048CA6			
		0	mov byte ptr [ebx+0x8049328],0x1			
		0	add ebx,+0x1			
		0	cmp ebx,+0x12			
		0	jle 0x8048CA6			
689		0				
		0	for ( i = 0 ; i <= SIZE ; i++ )			
		0	mov ebx,0x0			
		0	jmp 0x8048CE7			
		0	cmp ebx,+0x12			
		0	jle 0x8048C8C			
691		0	{			
		0	if ( flags[ i ] )			
		0	movzx eax,byte ptr [ebx+0x8049328]			
		0	test al,al			
		0	je 0x8048CE4			
693		0	{			
		0	primz = i + i + 3;			
		0	lea eax,[ebx+ebx]			
		0	lea edi,[eax+0x3]			
694		0	k = i + primz;			
		0	lea esi,[edi+ebx]			
695		0	while ( k <= SIZE )			
		0	jmp 0x8048CDB			
695		0	while ( k <= SIZE )			
		0	cmp esi,+0x12			
		0	jle 0x8048CD2			
-0000000209	TNT a=TTTTTN cc=0x26	38.	NP:000000008048CD2 ptrace			\\sieve_funcs_x86\sieve_funcs\sieve+0x43
		0	{			
		0	flags[ k ] = FALSE;			
697		0	mov byte ptr [esi+0x8049328],0x0			
698		0	k += primz;			
		0				

Cycle accurate tracing requires up to 2 times more bandwidth.

## Synchronization Time

Not implemented yet.

# Trace Configuration

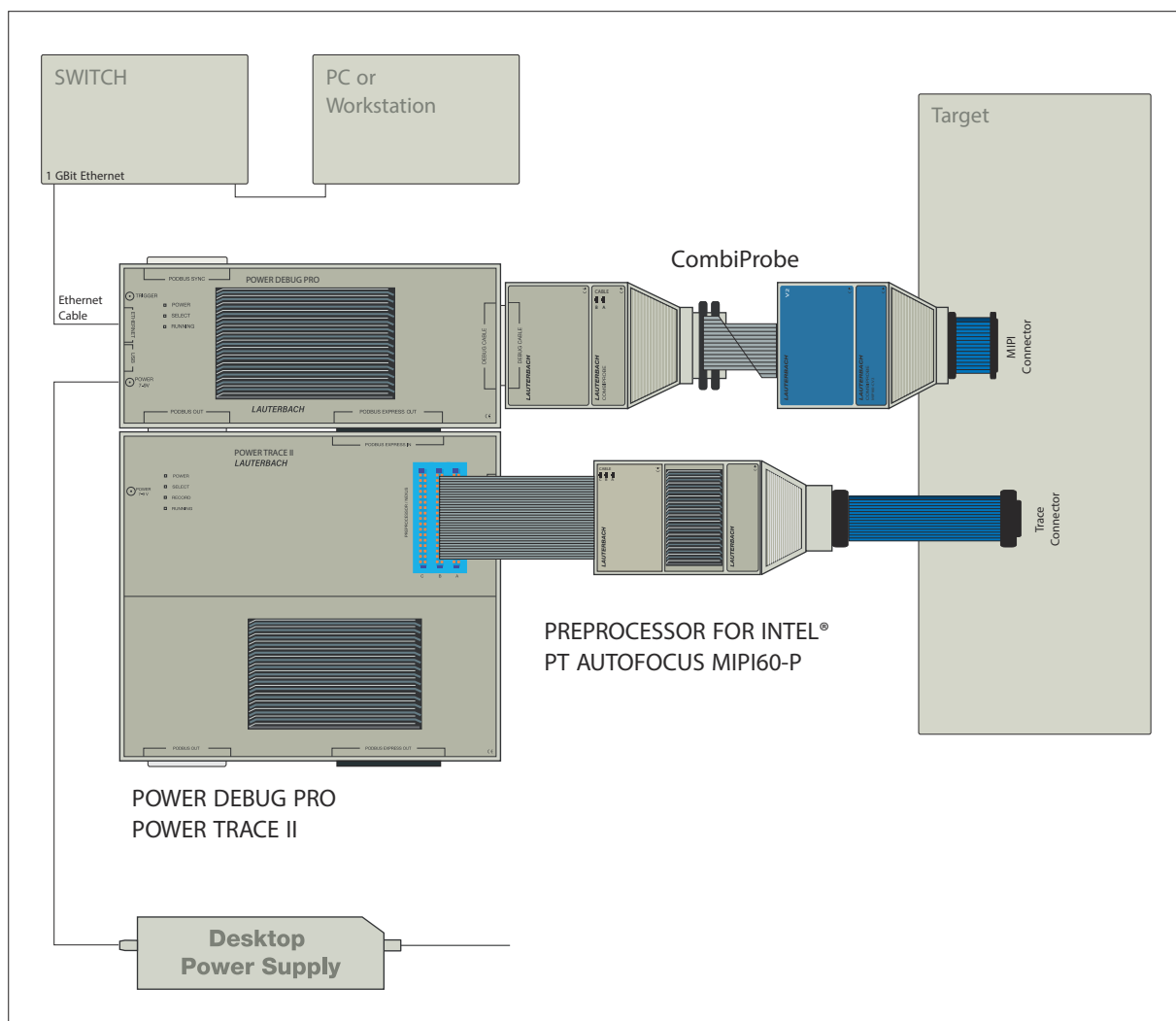
## Off-chip Trace

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
```

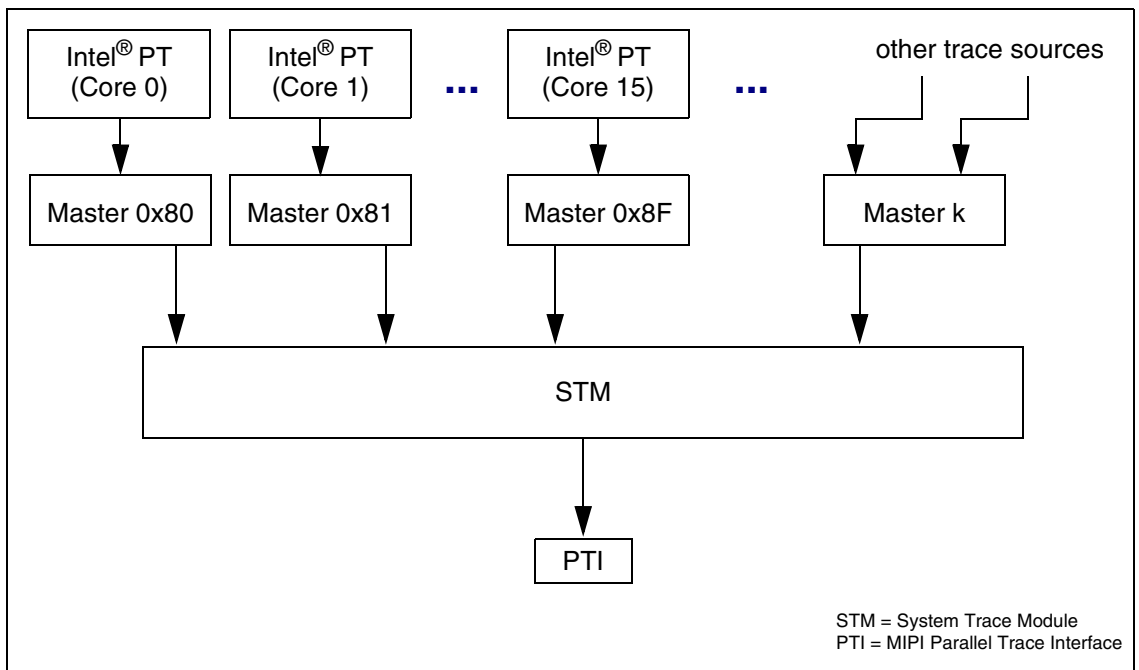
### 3.

### **IPT.TraceID** *<value>* | *<bitmask>*

Specify which masters/channels (that produce Intel® PT trace information) you want to analyze with the help of TRACE32.

<b>&lt;value&gt;</b>	<b>&lt;value&gt;</b> is a 32-bit number. The first 16 bits represent the master ID, the last 16 bits represent the channel ID.
<b>&lt;bitmask&gt;</b>	bitmask representation of <b>&lt;value&gt;</b>

**Example 1:** Each core has its own master ID.



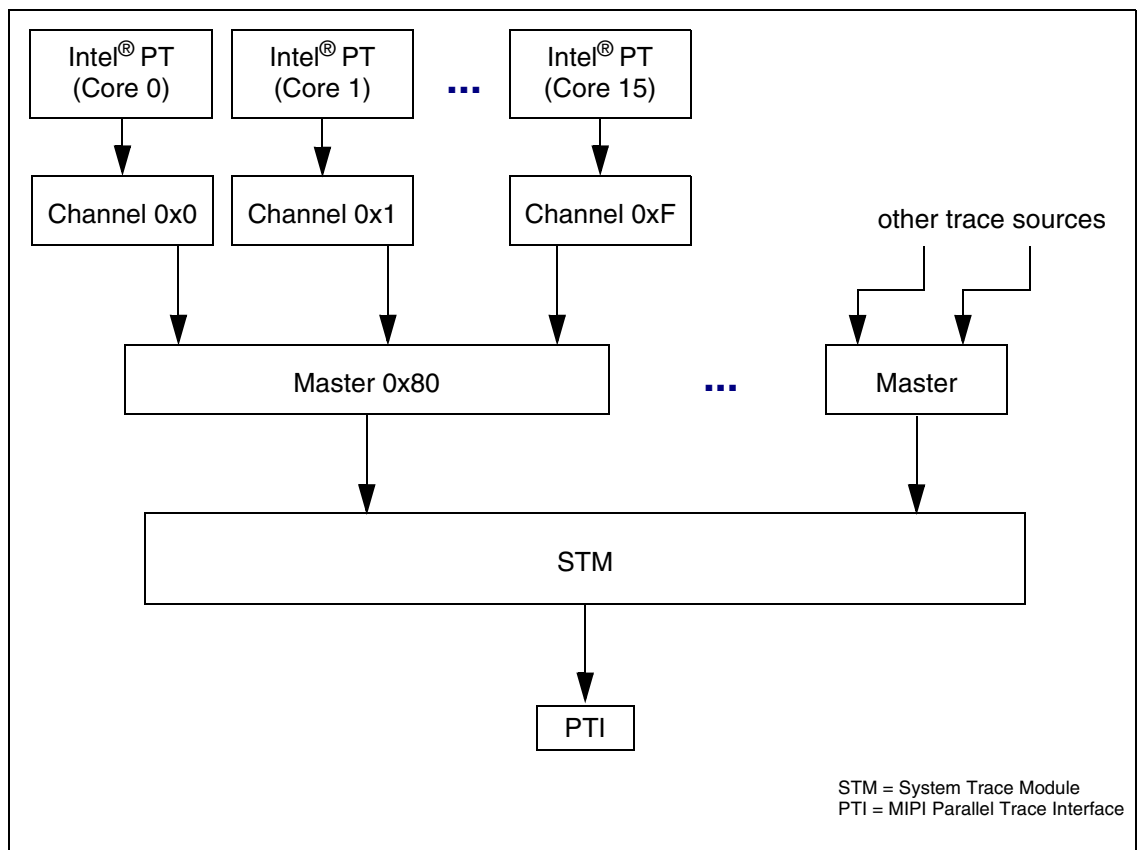
```

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
```

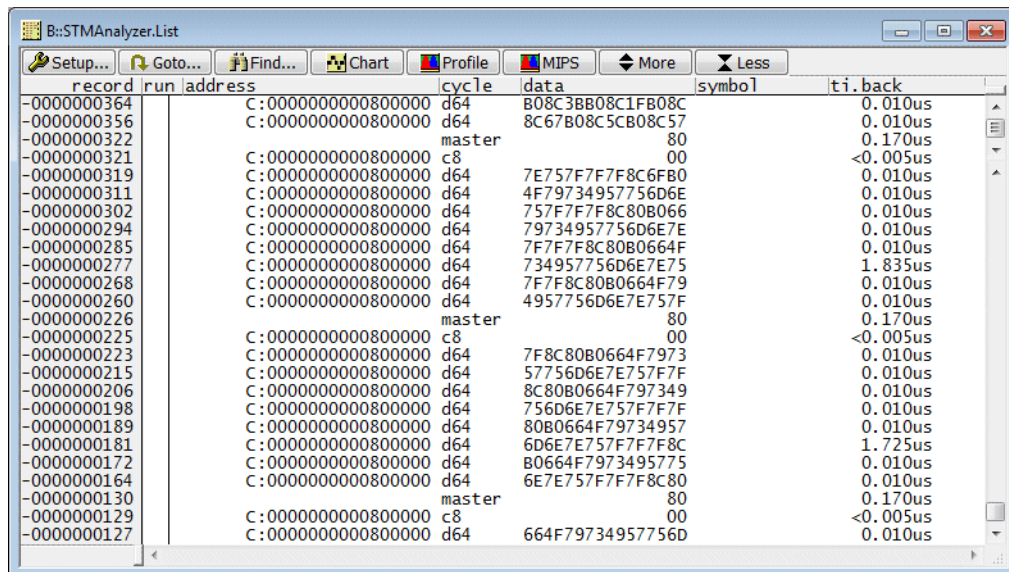
A manual setup is required if your target is using a gated clock. Refer to [“Manual Setup”](#) in AutoFocus User’s Guide, page 18 (autofocus\_user.pdf) for assistance.



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

## STMAalyzer.List

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

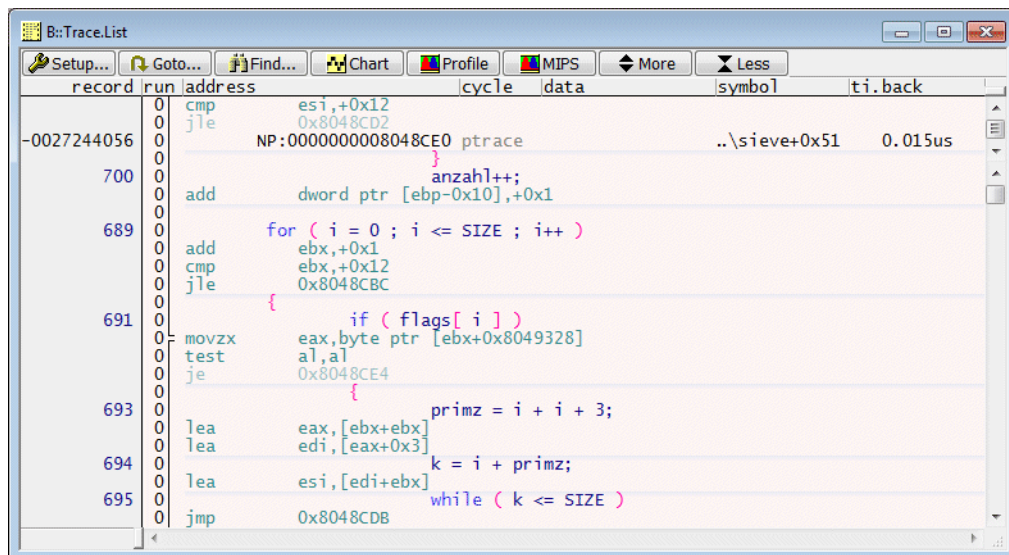


record	run	address	cycle	data	symbol	ti.back
-0000000364		C:000000000800000	d64	B08C3BB08C1FB08C		0.010us
-0000000356		C:000000000800000	d64	8C67B08C5CB08C57		0.010us
-0000000322			master	80		0.170us
-0000000321		C:000000000800000	c8	00		<0.005us
-0000000319		C:000000000800000	d64	7E757F7F7F8C6FB0		0.010us
-0000000311		C:000000000800000	d64	4F79734957756D6E		0.010us
-0000000302		C:000000000800000	d64	757F7F7F8C80B066		0.010us
-0000000294		C:000000000800000	d64	79734957756D6E7E		0.010us
-0000000285		C:000000000800000	d64	7F7F7F8C80B0664F		0.010us
-0000000277		C:000000000800000	d64	734957756D6E7E75		1.835us
-0000000268		C:000000000800000	d64	7F7F8C80B0664F79		0.010us
-0000000260		C:000000000800000	d64	4957756D6E7E757F		0.010us
-0000000226			master	80		0.170us
-0000000225		C:000000000800000	c8	00		<0.005us
-0000000223		C:000000000800000	d64	7F8C80B0664F7973		0.010us
-0000000215		C:000000000800000	d64	57756D6E7E757F7F		0.010us
-0000000206		C:000000000800000	d64	8C80B0664F797349		0.010us
-0000000198		C:000000000800000	d64	756D6E7E757F7F7F		0.010us
-0000000189		C:000000000800000	d64	80B0664F79734957		0.010us
-0000000181		C:000000000800000	d64	6D6E7E757F7F7F8C		1.725us
-0000000172		C:000000000800000	d64	B0664F7973495775		0.010us
-0000000164		C:000000000800000	d64	6E7E757F7F7F8C80		0.010us
-0000000130			master	80		0.170us
-0000000129		C:000000000800000	c8	00		<0.005us
-0000000127		C:000000000800000	d64	664F79734957756D		0.010us

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

## Analyzer.List

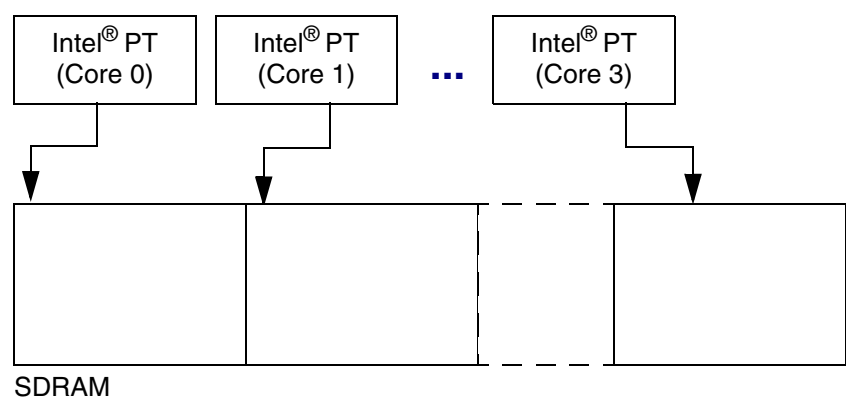
Display all core trace information decoded out of the STP packet stream.



record	run	address	cycle	data	symbol	ti.back
-0027244056	0	cmp esi,+0x12				
	0	jle 0x8048CD2				
	0	NP:0000000008048CE0 ptrace			..\sieve+0x51	0.015us
700	0	add dword ptr [ebp-0x10],+0x1				
689	0	for ( i = 0 ; i <= SIZE ; i++ )				
	0	add ebx,+0x1				
	0	cmp ebx,+0x12				
	0	jle 0x8048CBC				
691	0	{ if ( flags[ i ] )				
	0	movzx eax,byte ptr [ebx+0x8049328]				
	0	test al,al				
	0	je 0x8048CE4				
693	0	{ primz = i + i + 3;				
	0	lea eax,[ebx+ebx]				
	0	lea edi,[eax+0x3]				
694	0	k = i + primz;				
695	0	lea esi,[edi+ebx]				
	0	while ( k <= SIZE )				
	0	jmp 0x8048CDB				

# SDRAM Trace

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
```

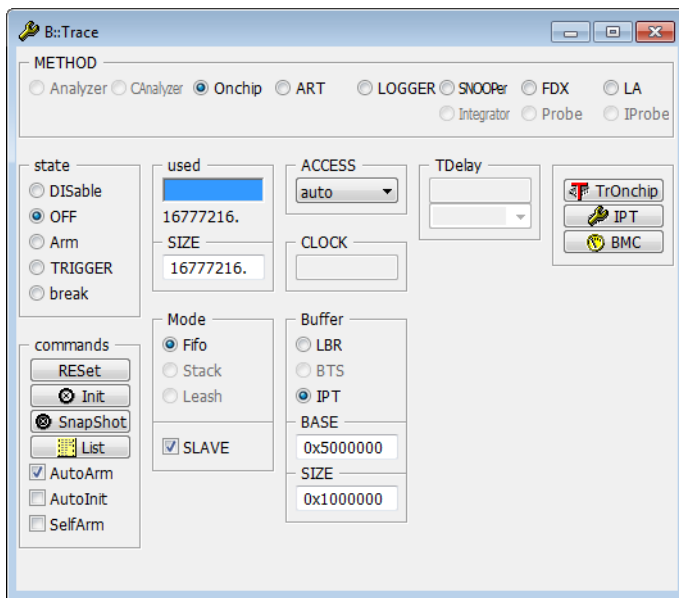
TRACE32 reads the onchip trace via JTAG.

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

Intel® PT packet 1
Intel® PT packet 2
Intel® PT packet 3
Intel® PT packet 4
Intel® PT packet 5

SDRAM block  
core 1

Intel® PT packet 1
Intel® PT packet 2
Intel® PT packet 3
Intel® PT packet 4
Intel® PT packet 5
Intel® PT packet 6
Intel® PT packet 7

SDRAM block  
core 2

Intel® PT packet 1
Intel® PT packet 2
Intel® PT packet 3
Intel® PT packet 4
Intel® PT packet 5
Intel® PT packet 6

SDRAM block  
core 3

Intel® PT packet 1

**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

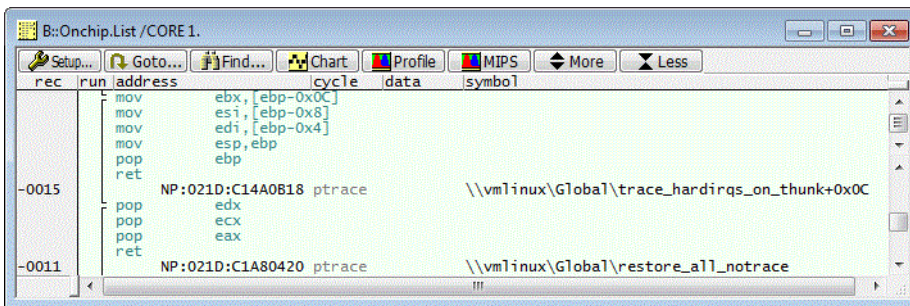
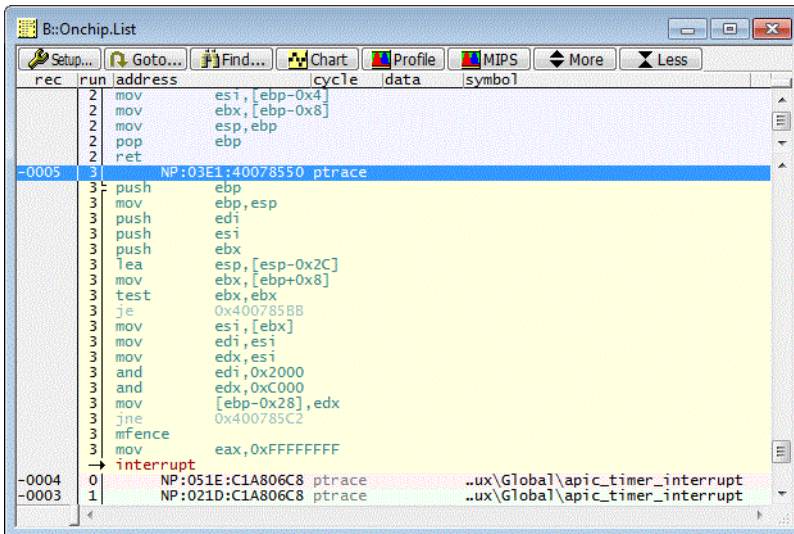
This procedure will change with the decoding of synchronisation packets.

```

Onchip.List                                ; display trace listing for all
                                           ; cores

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

```

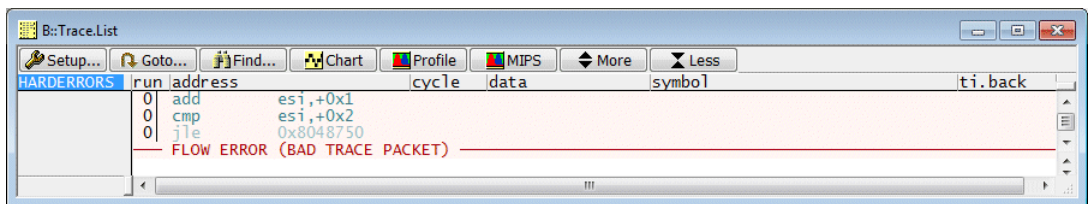


## ERRORS

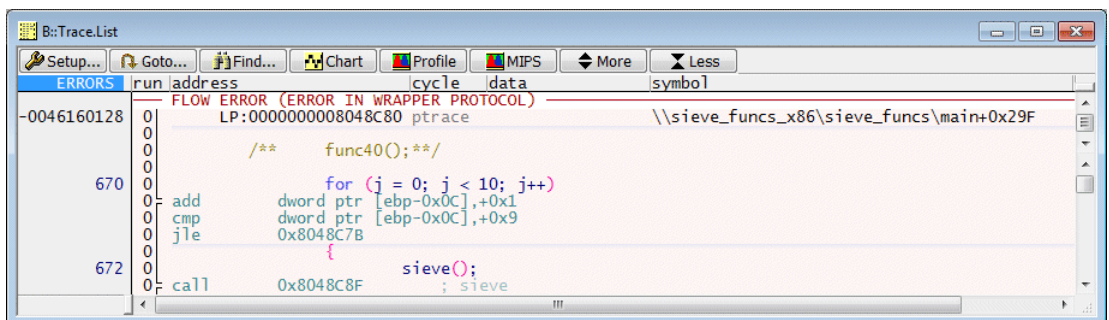
If the trace contains **ERRORS**, please try to set up a proper trace recording before you start to evaluate or analyze the trace contents.

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.

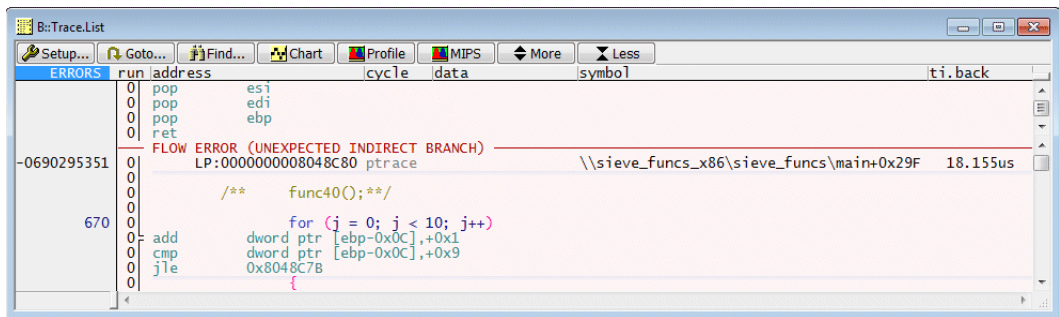


- TRACE32 could not decode the packet.



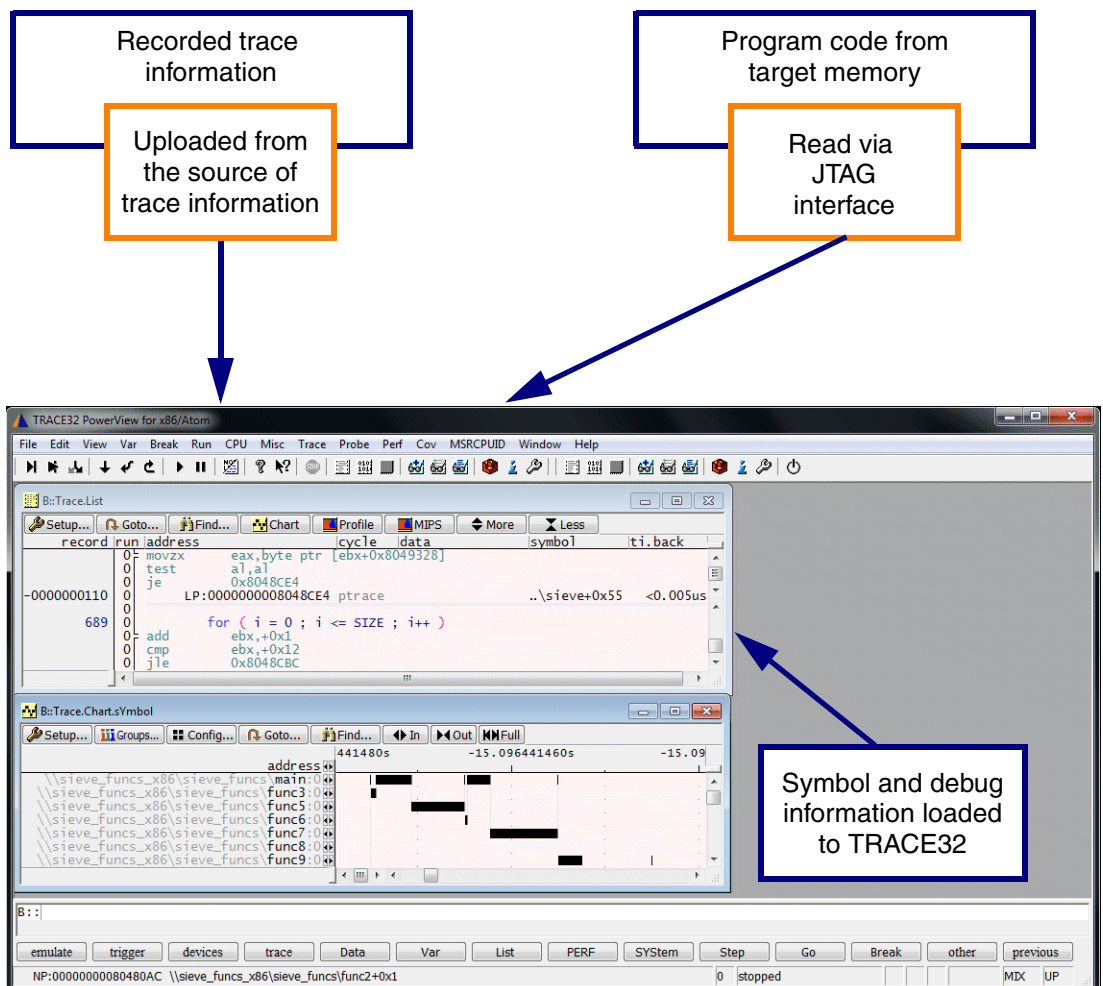


- The trace information is not consistent with the program code in the target memory.



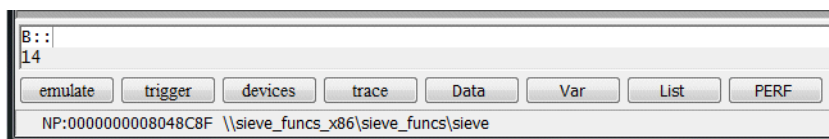
Background: In order to provide an intuitive trace display the following sources of information are merged:

- 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.

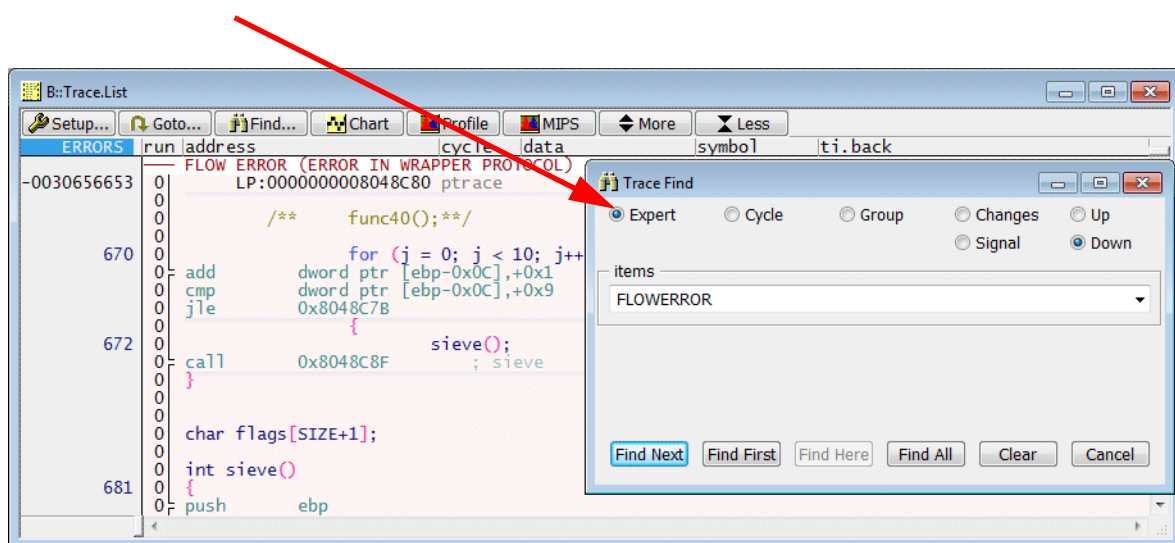


The TRACE32 function **Trace.FLOW.ERROR()** returns the number of ERRORS as a hex. number.

```
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.

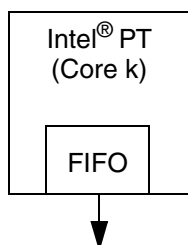


### Trace.Find FLOWERROR



# 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.

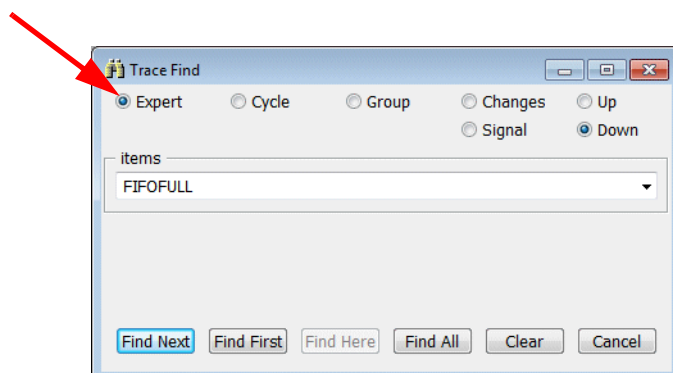
The affected Intel® PT generates a Buffer Overflow packet (FUP.OVF) to indicate that its FIFO is full and trace packets are no longer generated.

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

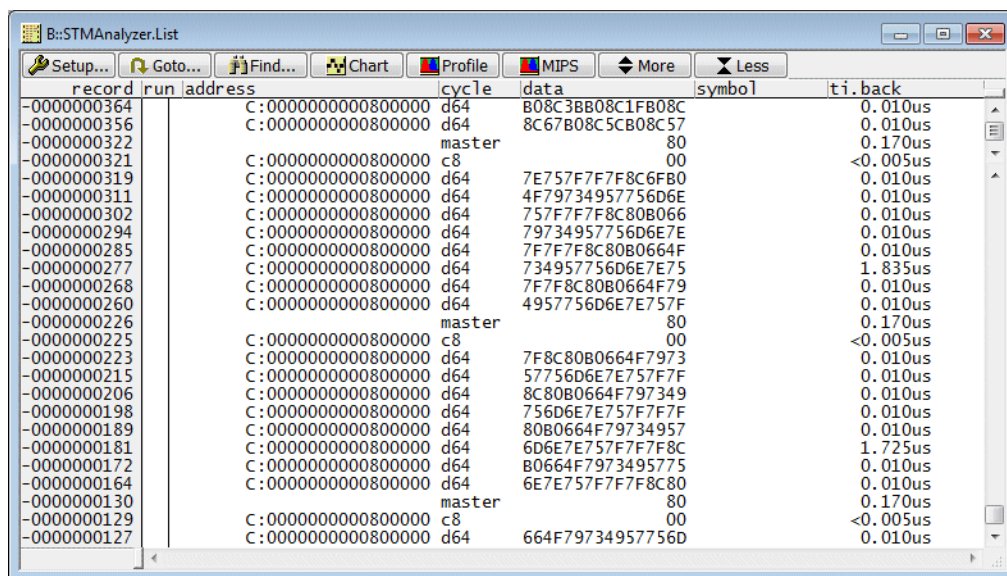
TARGET FIFO OVERFLOWS are strictly speaking not errors. They can occur in normal operation.

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:

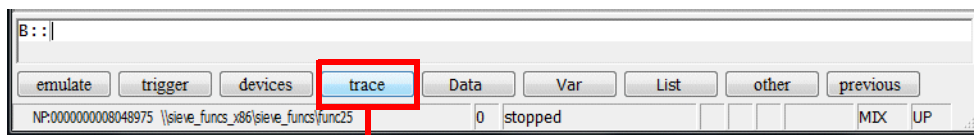
- **STMAalyzer.<sub\_cmd>**, if the STP packets are stored in the trace memory provided by POWER TRACE II / POWER TRACE III.



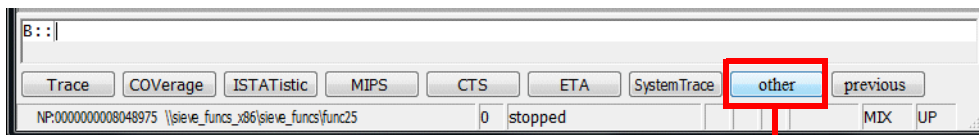
record	run	address	cycle	data	symbol	ti.back
-0000000364		C:0000000000800000	d64	B08C3BB08C1FB08C		0.010us
-0000000356		C:0000000000800000	d64	8C67B08C5CB08C57		0.010us
-0000000322			master	80		0.170us
-0000000321		C:0000000000800000	c8	00		<0.005us
-0000000319		C:0000000000800000	d64	7E757F7F7F8C6FB0		0.010us
-0000000311		C:0000000000800000	d64	4F79734957756D6E		0.010us
-0000000302		C:0000000000800000	d64	757F7F7F8C80B066		0.010us
-0000000294		C:0000000000800000	d64	79734957756D6E7E		0.010us
-0000000285		C:0000000000800000	d64	7F7F7F8C80B0664F		0.010us
-0000000277		C:0000000000800000	d64	734957756D6E7E75		1.835us
-0000000268		C:0000000000800000	d64	7F7F8C80B0664F79		0.010us
-0000000260		C:0000000000800000	d64	4957756D6E7E757F		0.010us
-0000000226			master	80		0.170us
-0000000225		C:0000000000800000	c8	00		<0.005us
-0000000223		C:0000000000800000	d64	7F8C80B0664F7973		0.010us
-0000000215		C:0000000000800000	d64	57756D6E7E757F7F		0.010us
-0000000206		C:0000000000800000	d64	8C80B0664F797349		0.010us
-0000000198		C:0000000000800000	d64	756D6E7E757F7F7F		0.010us
-0000000189		C:0000000000800000	d64	80B0664F79734957		0.010us
-0000000181		C:0000000000800000	d64	6D6E7E757F7F7F8C		1.725us
-0000000172		C:0000000000800000	d64	B0664F7973495775		0.010us
-0000000164		C:0000000000800000	d64	6E7E757F7F7F8C80		0.010us
-0000000130			master	80		0.170us
-0000000129		C:0000000000800000	c8	00		<0.005us
-0000000127		C:0000000000800000	d64	664F79734957756D		0.010us

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

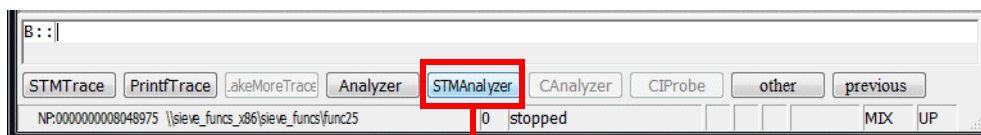
The command groups usable in your current configuration can be get from the TRACE32 PowerView Softkey line.



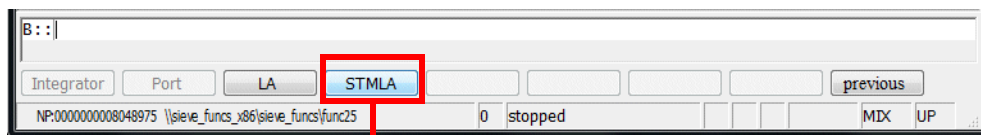
Push **trace** to get access to all command groups that analyze trace information.



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



POWER TRACE II / POWER TRACE III is used in the current configuration, so the command group STMAAnalyzer 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:

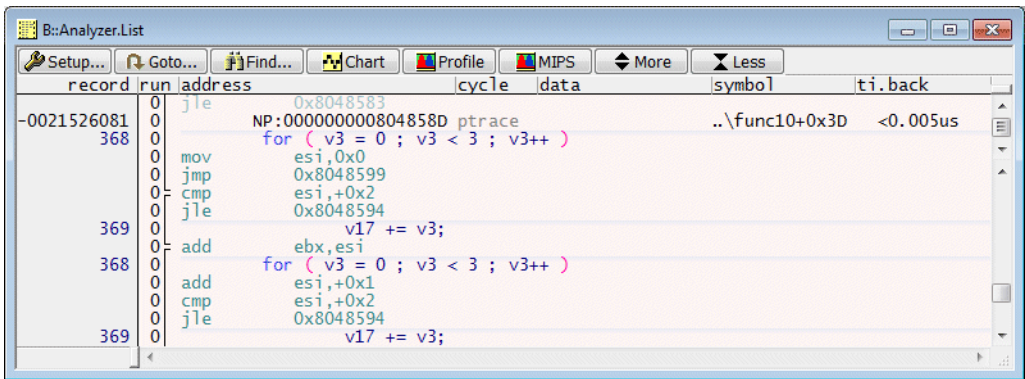
### SystemTrace.METHOD Analyzer | LA

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

# (Core) Trace

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.

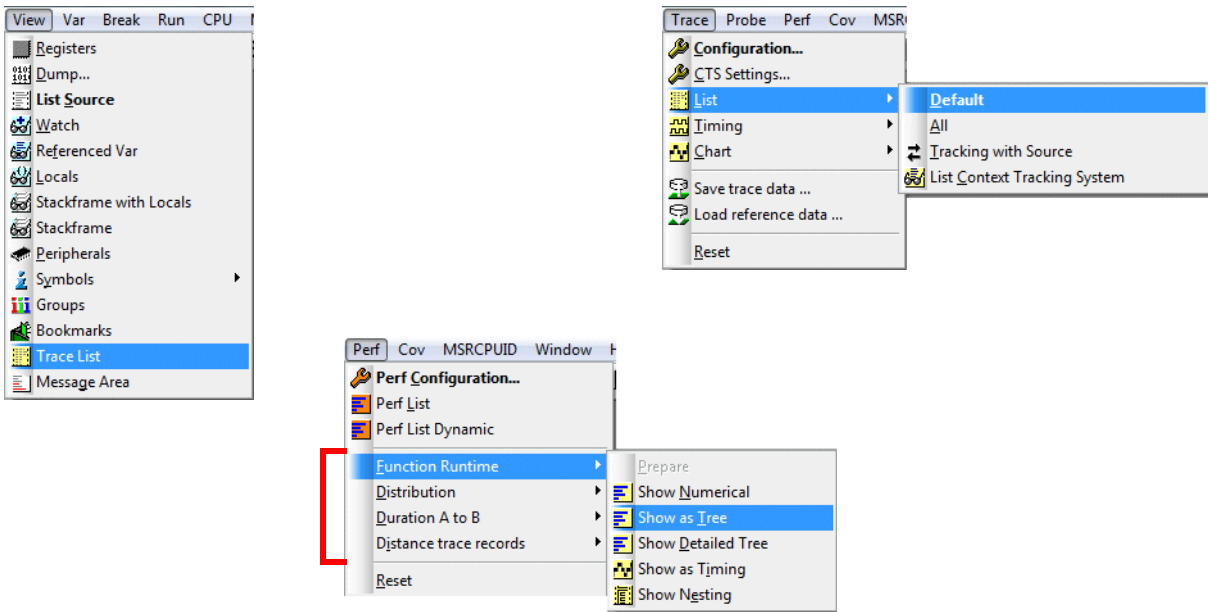
TRACE32 PowerView offers the following abstraction, since most <sub\_cmd> are identical for all command groups:

## 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:

<b>COVERAGE.&lt;sub_cmd&gt;</b>	Trace-based code coverage
<b>ISTAT.&lt;sub_cmd&gt;</b>	Detailed instruction analysis
<b>MIPS.&lt;sub_cmd&gt;</b>	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.

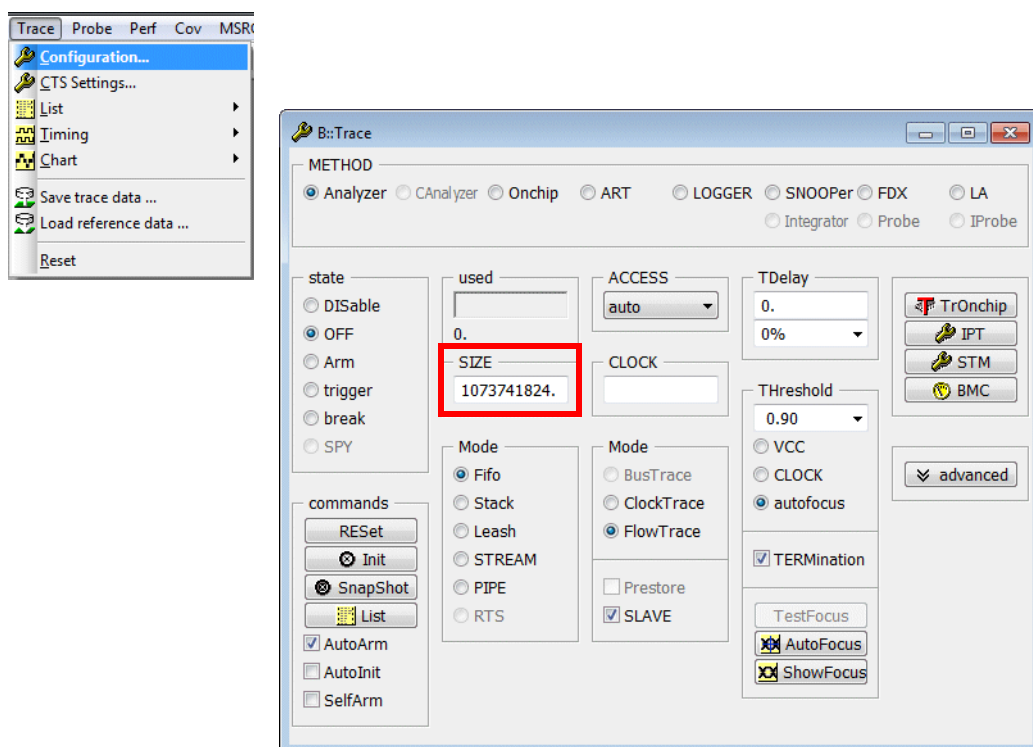
Another important influencing factor are the settings in the **TRACE32 Trace Configuration** window. They specify how much trace information can be recorded and when the trace recording is stopped.

# 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.

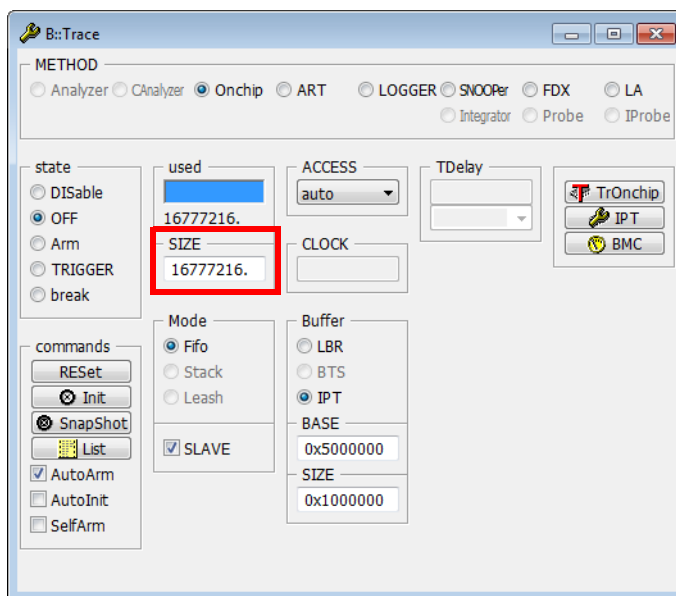
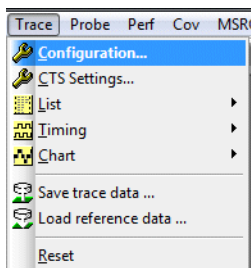
<b>Trace.Mode PIPE</b>	
<b>Trace.PipeWrite</b> <pipe_name>	Connect to named pipe
<b>Trace.PipeWrite</b> \\.\pipe\<pipe_name>	Connect to named pipe (Windows)
<b>Trace.PipeWrite</b>	Disconnect from named pipe

```
...  
  
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:

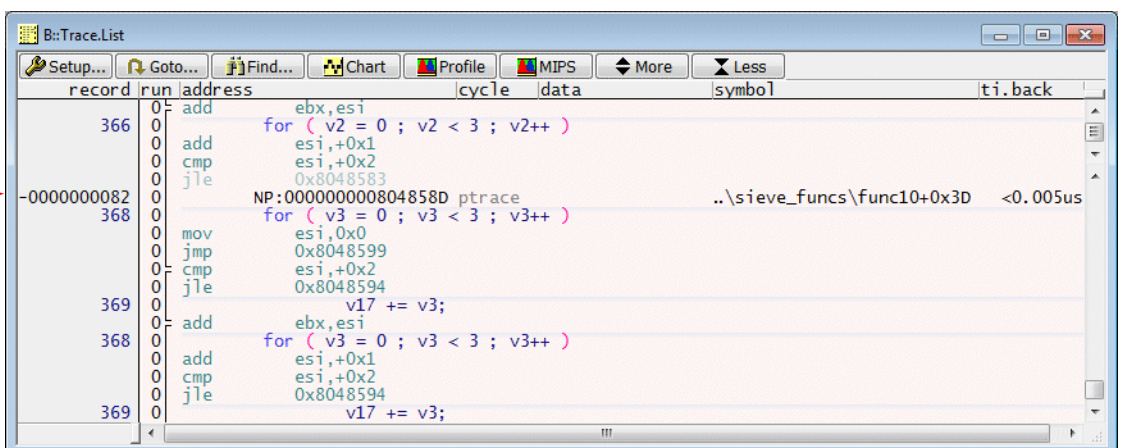
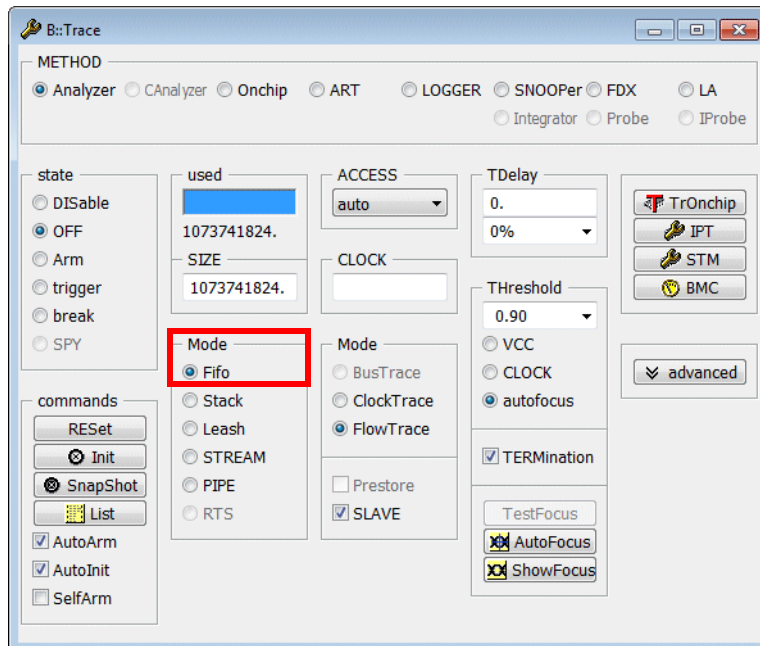
- **Fifo**: allows to record as much trace records as indicated in the **SIZE** field of the Trace Configuration window.



```
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
```

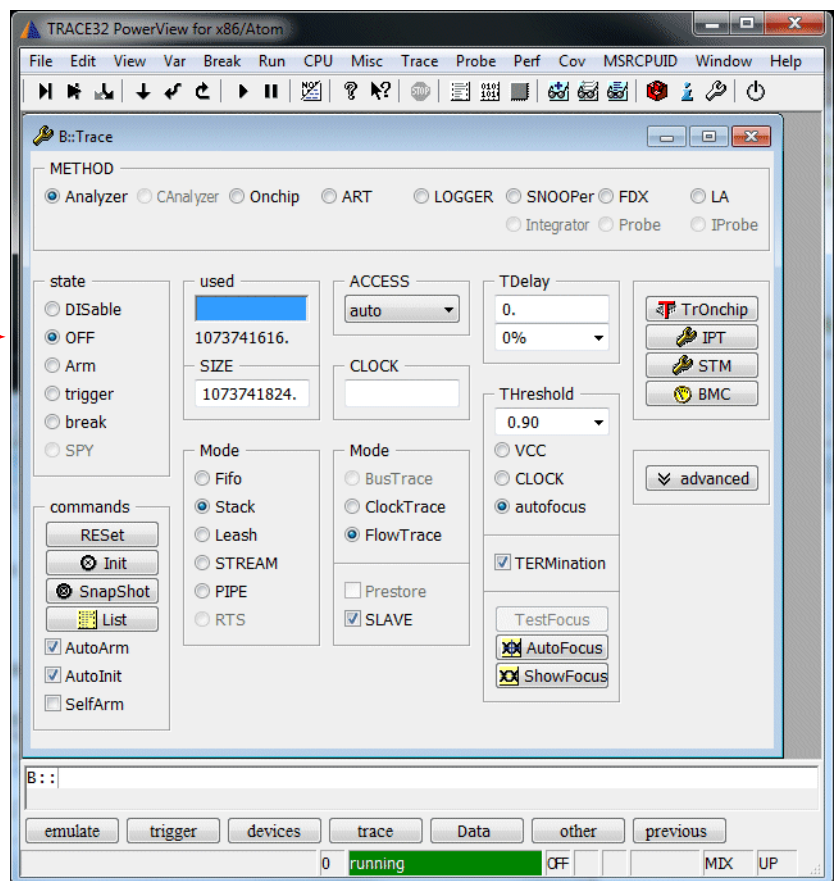


In **Fifo** mode negative record numbers are used. The last record gets the smallest negative number.

```
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
```

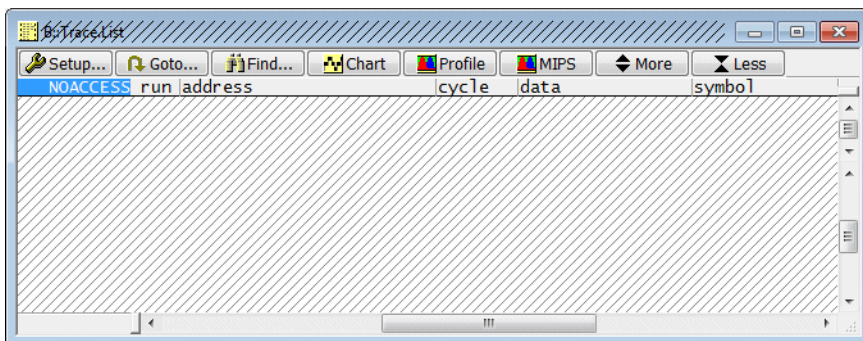
The trace recording is stopped as soon as the trace memory is full (OFF state)



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**.

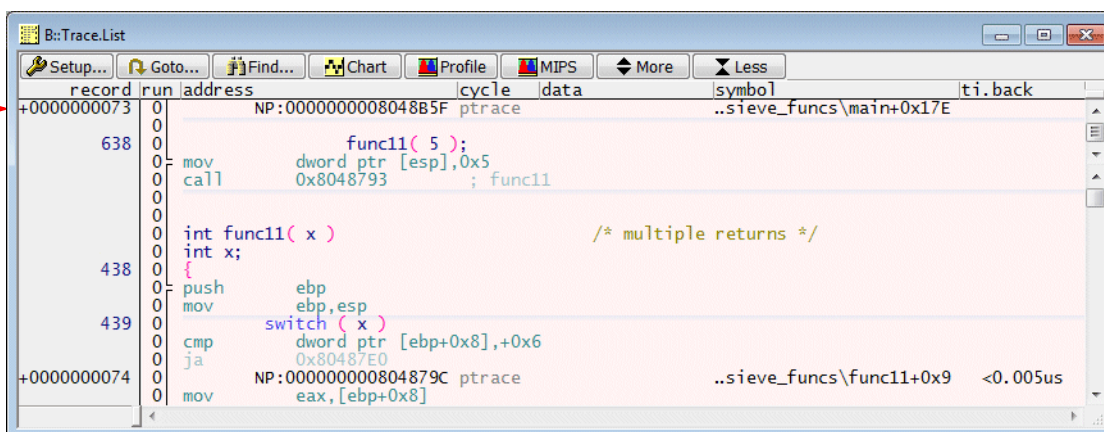


Stop the program execution to allow TRACE32 to read the program code from the target. Or if you need to display the core trace information while the program execution is running, load a copy of the program code to the **TRACE32 Virtual Memory**.

**Data.LOAD.Elf <file> /PlusVM**

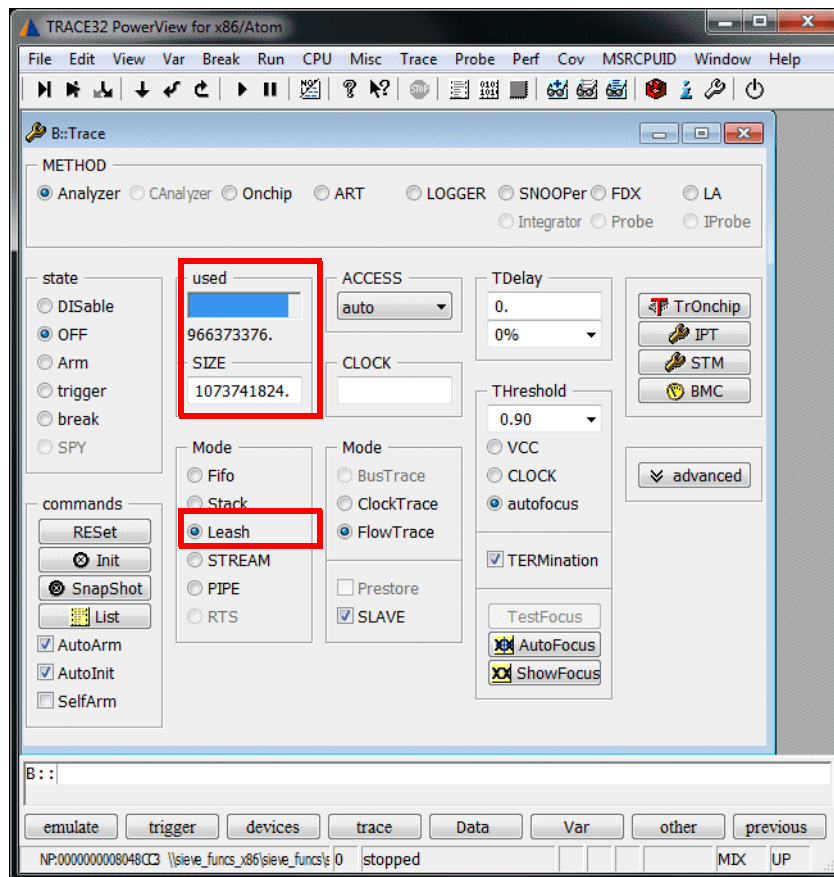
Load the program code to the target and to the TRACE32 Virtual Memory.

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.



## Leash Mode (Analyzer only)

```
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 than 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 than 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
```

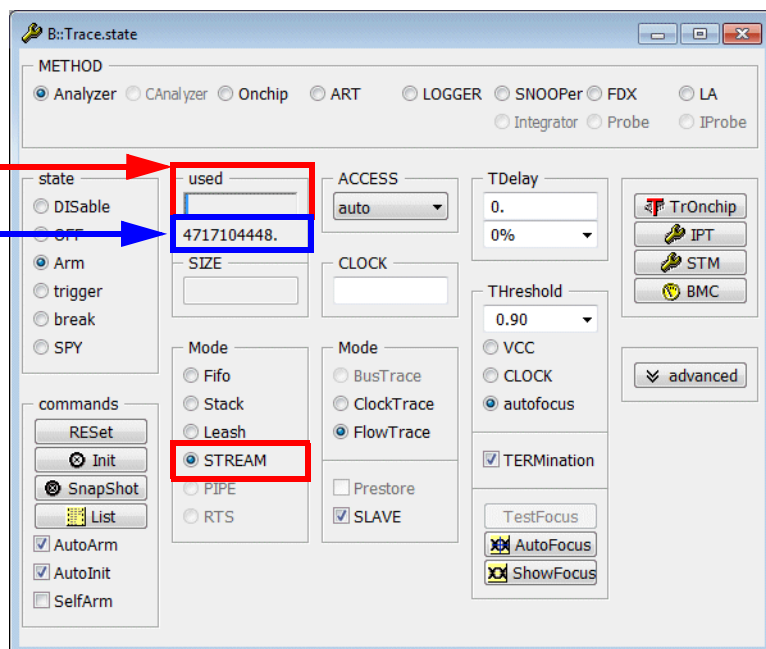
Please be aware that the streaming file is deleted as soon as you de-select the STREAM mode or when you exit TRACE32.

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.

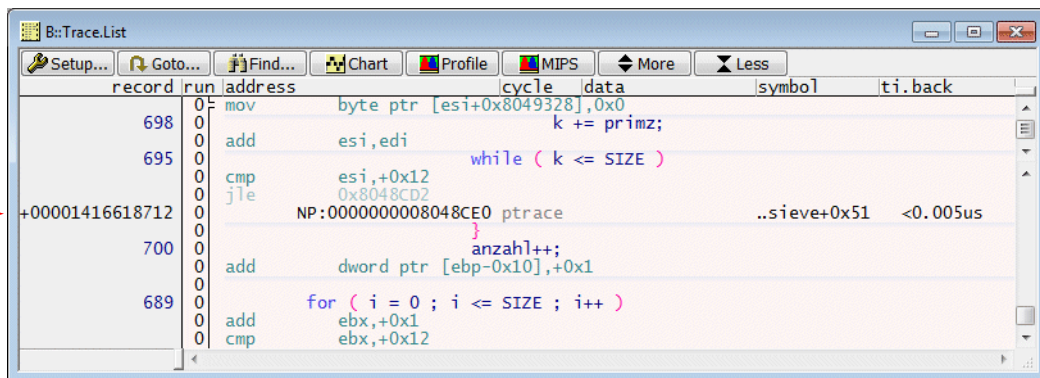
Trace.STREAMCompression HIGH

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

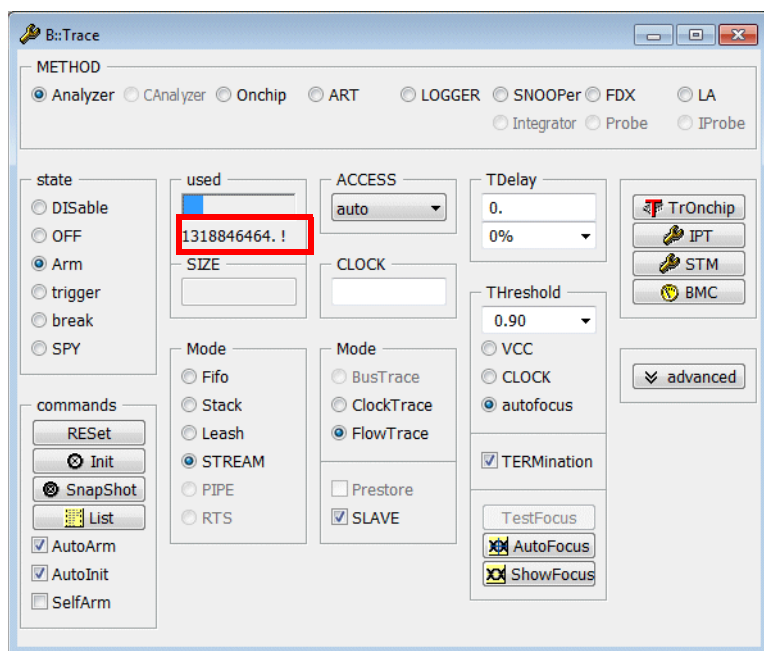


STREAM mode can only be used if the average data rate at the trace port does not exceed the maximum transmission rate of the host interface in use. Peak loads at the trace port are intercepted by the memory in POWER TRACE II / POWER TRACE III, which can be considered to be operating as a large FIFO.

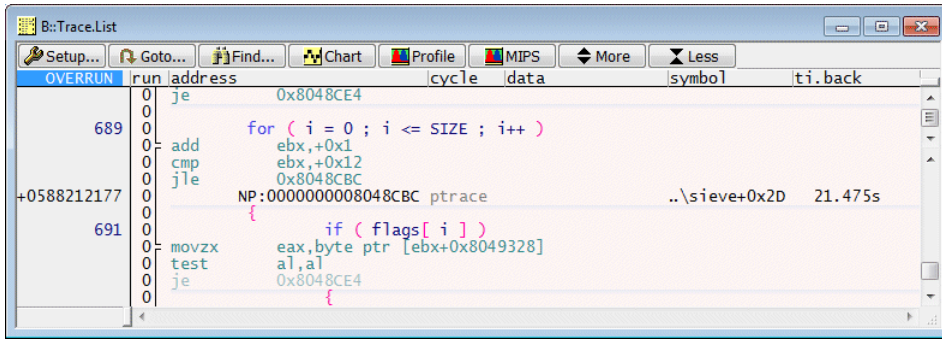
If the average data rate at the trace port exceeds the maximum transmission rate of the host interface in use, a **PowerTrace FIFO Overrun** occurs. TRACE32 stops streaming and empties the POWER TRACE II / POWER TRACE III FIFO. Streaming is re-started after the POWER TRACE II / POWER TRACE III FIFO is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



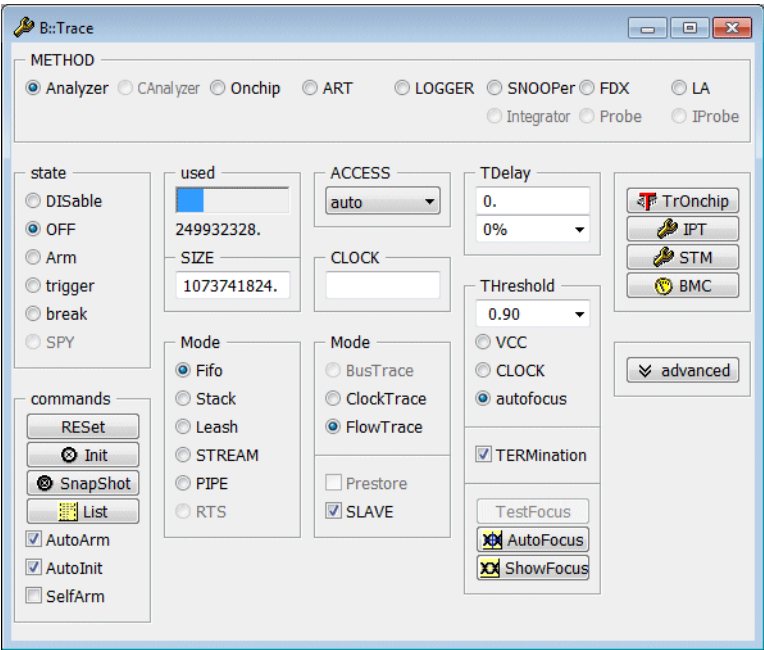
OVERRUNs are not visible at record level.

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

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

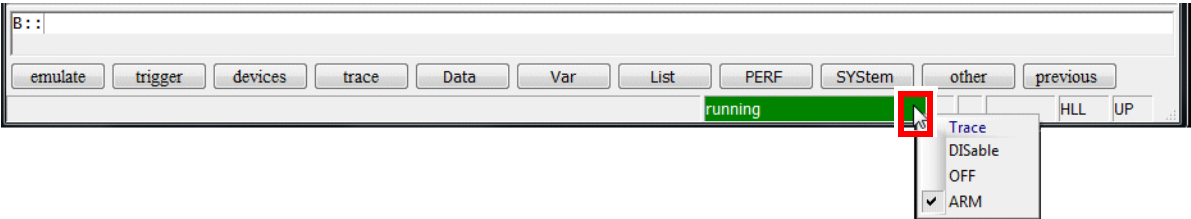
# States of the Trace

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



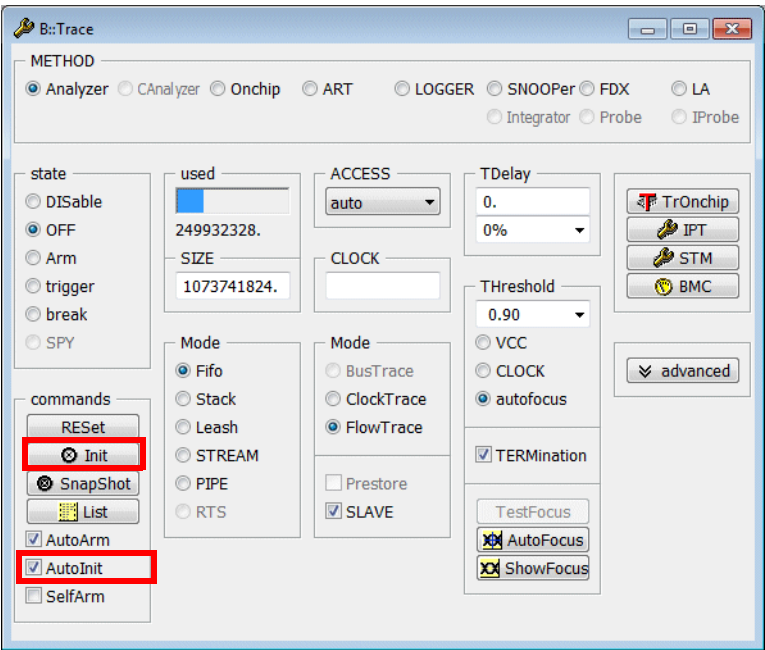
States of the Trace	
DISable	The trace is disabled.
OFF	The trace is not sampling. The trace contents can be analyzed and displayed.
Arm	The trace is sampling. There is no access to the trace contents.

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



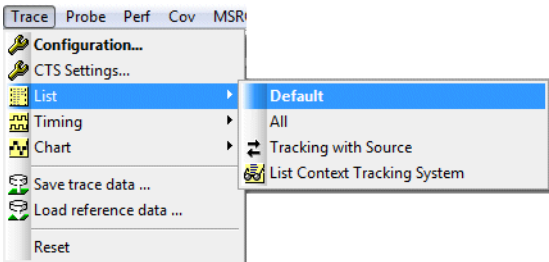
Since Intel® PT does not provide a mean to indicate a trigger, the Trace states **trigger** and **break** are never reached.

# The Autolnit Command



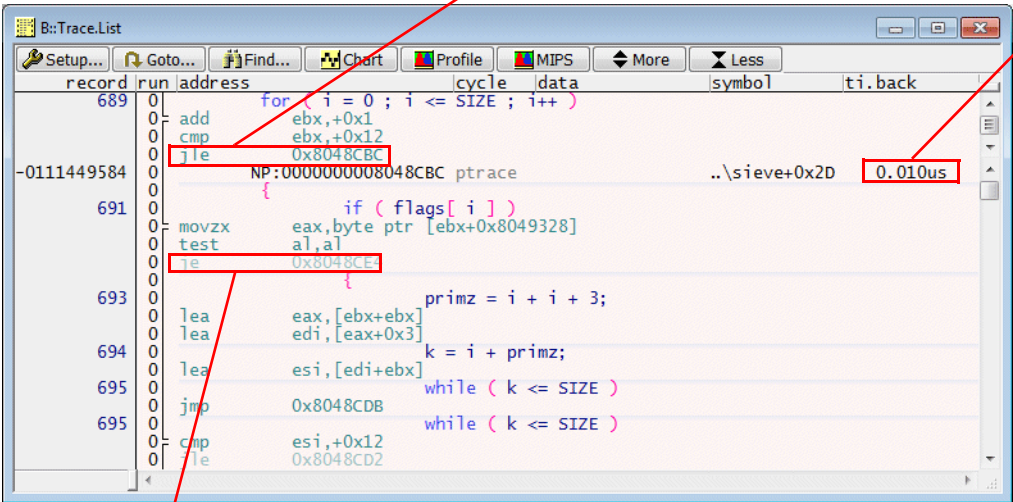
Init Button	Clear the trace memory. All other settings in the Trace configuration window remain valid.
Autolnit CheckBox	ON: The trace memory is cleared whenever the program execution is started (Go, Step).

Default Listing



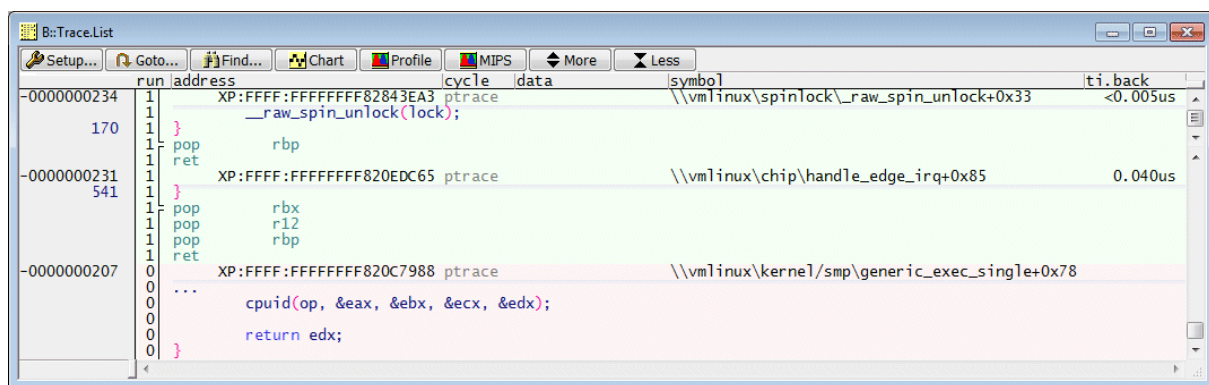
Conditional  
branch taken

Timing information



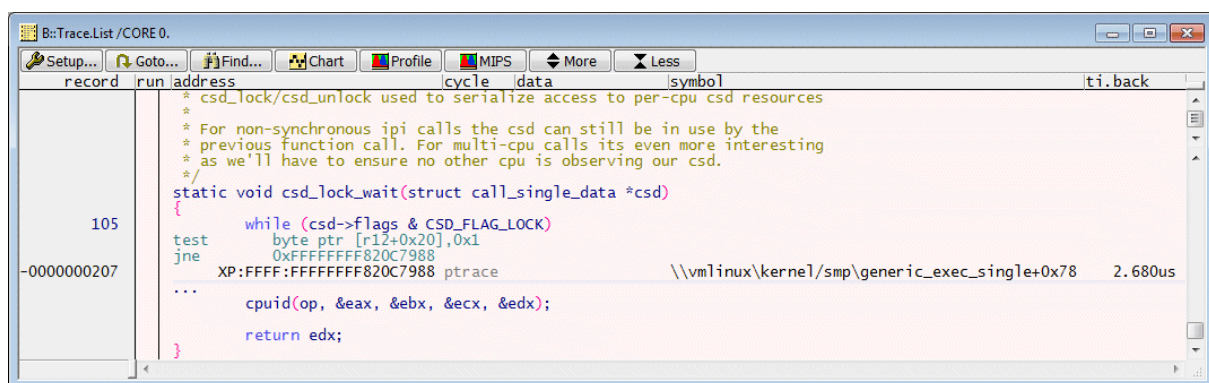
Conditional  
branch not taken  
(pastel printed)

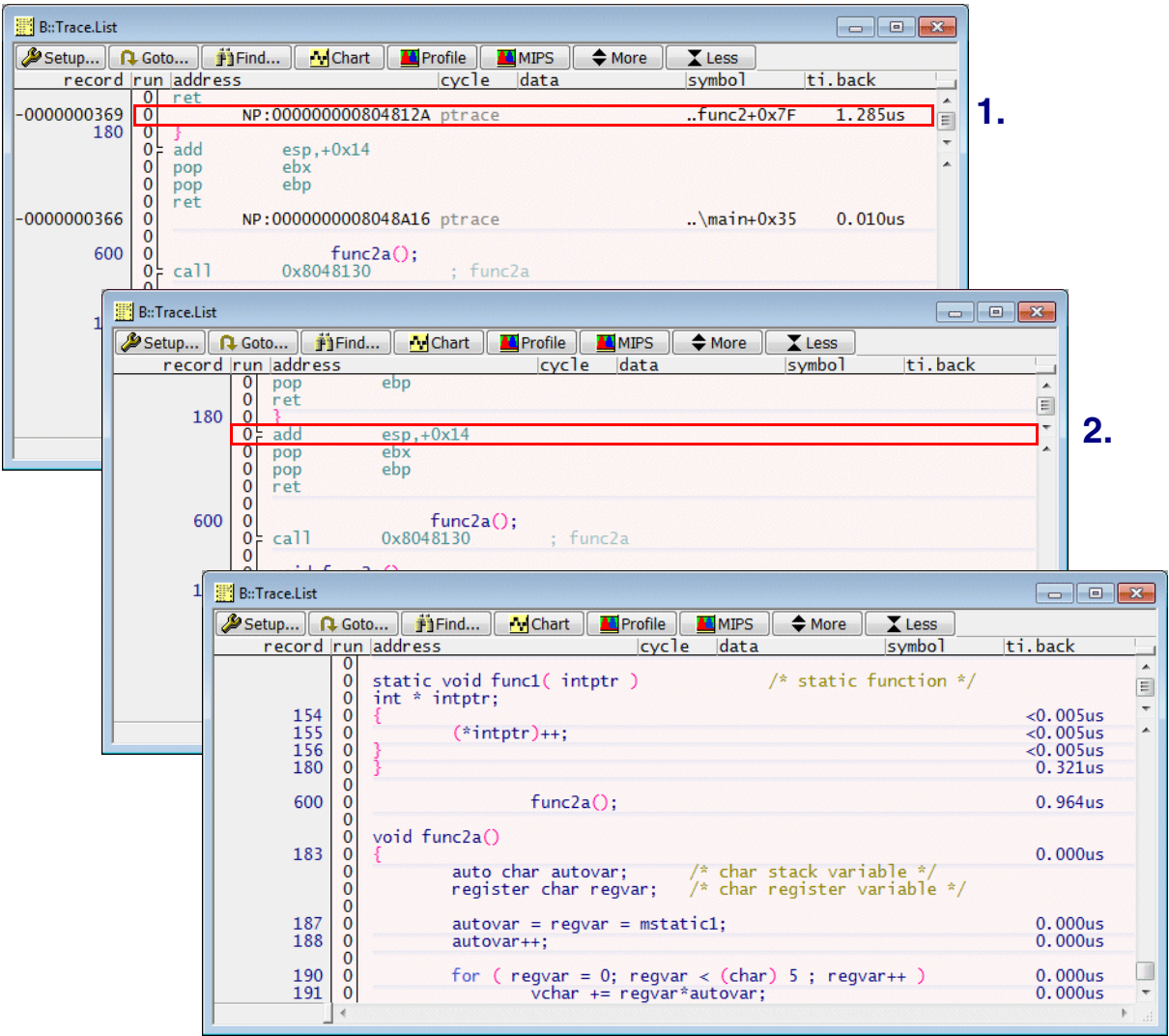
The trace information for all cores is displayed by default in the **Trace.List** window. The column run and the coloring of the trace information are used for core indication.



### Trace.List /CORE <n>

The option CORE allows a per core display of the trace information.



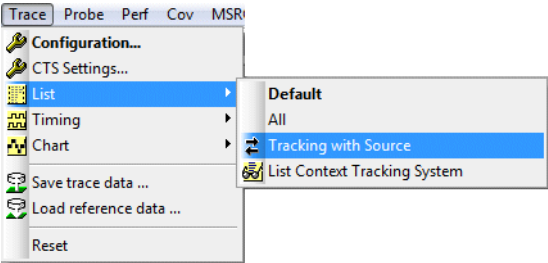


1. time Less	Suppress the display of the ptrace packets.
2. time Less	Suppress the display of the assembly code.

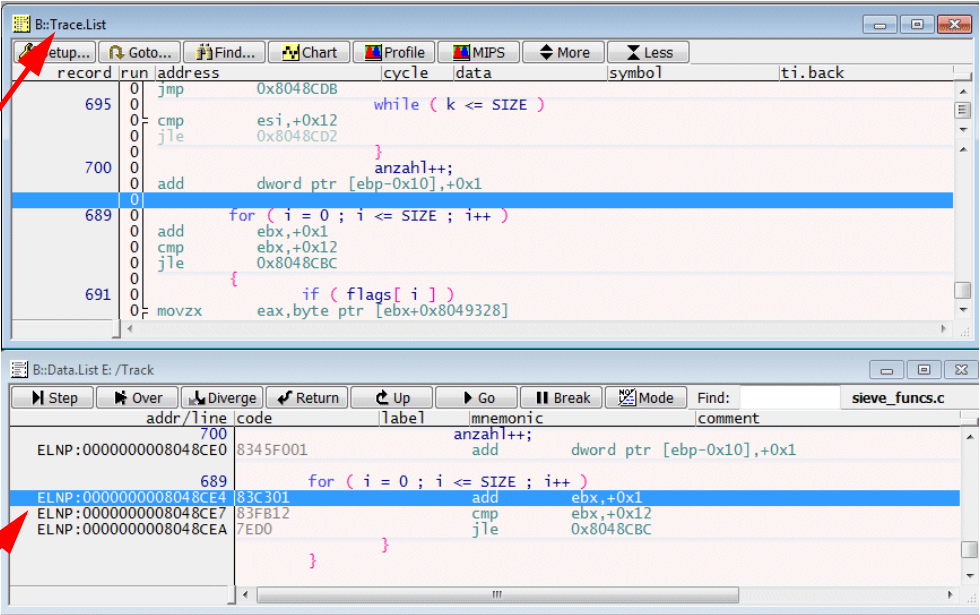
The **More** button works vice versa.



# Correlating the Trace Listing with the Source Listing



Active Window

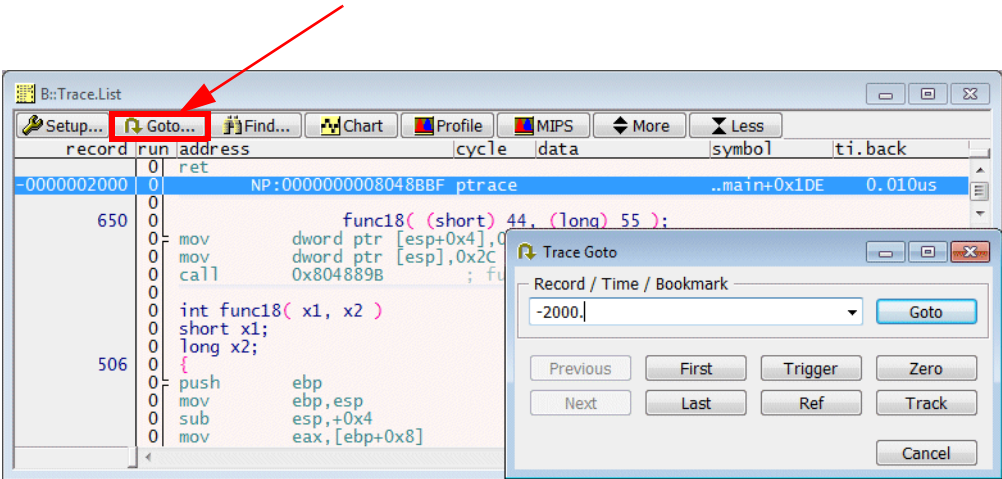


All windows opened with the **/Track** option follow the cursor movements in the active window

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



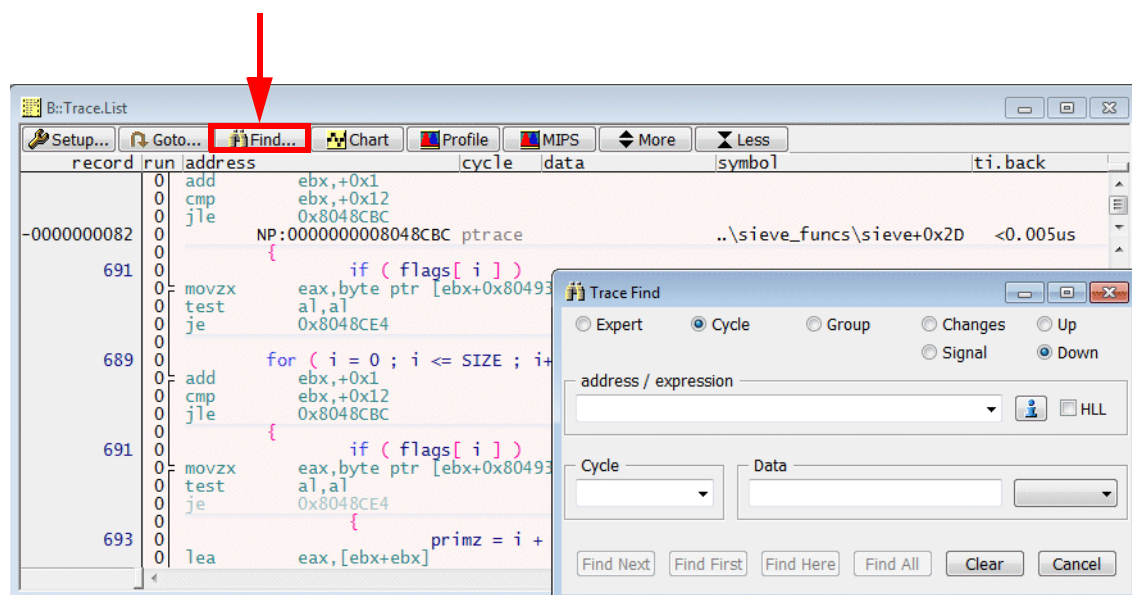
# Browsing through the Trace Buffer



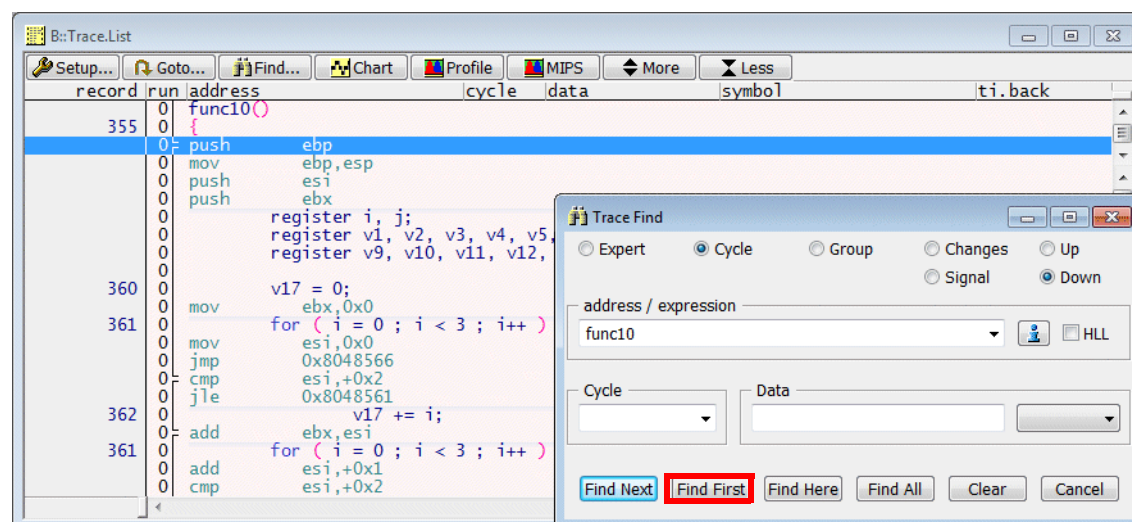
<b>Pg</b> ↑	Scroll page up.
<b>Pg</b> ↓	Scroll page down.
<b>Ctrl - Pg</b> ↑	Go to the first record sampled in the trace buffer.
<b>Ctrl - Pg</b> ↓	Go to the last record sampled in the trace buffer.

## 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.

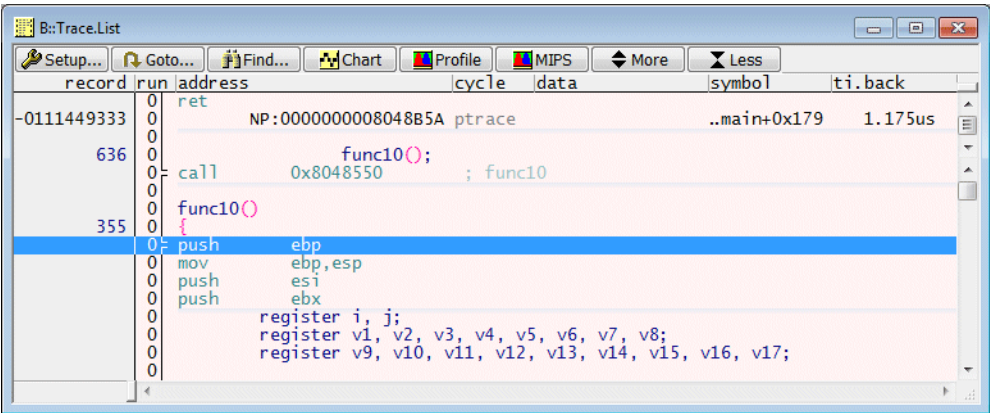


**Example:** Find the entry to the function `func10`.



A detailed description of the **Trace Find** dialog can be found in [“Application Note for Trace.Find”](#) (app\_trace\_find.pdf).

Default Display Items

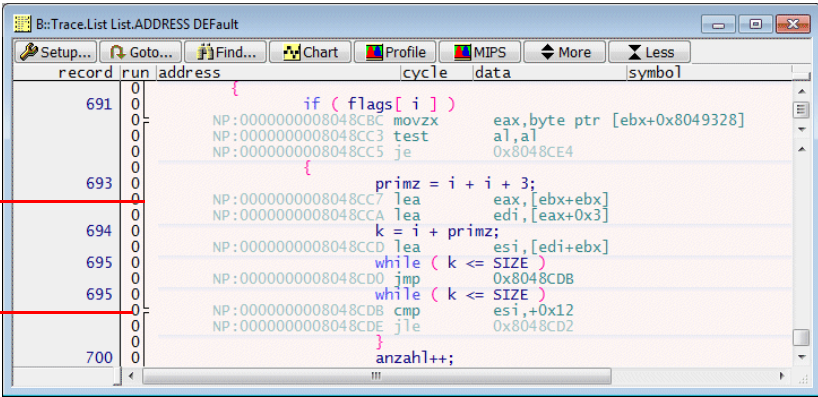


- **Column *record***  
Displays the record numbers
- **Column *run***  
The column run displays some graphic element to provide a quick overview on the instruction execution sequence.

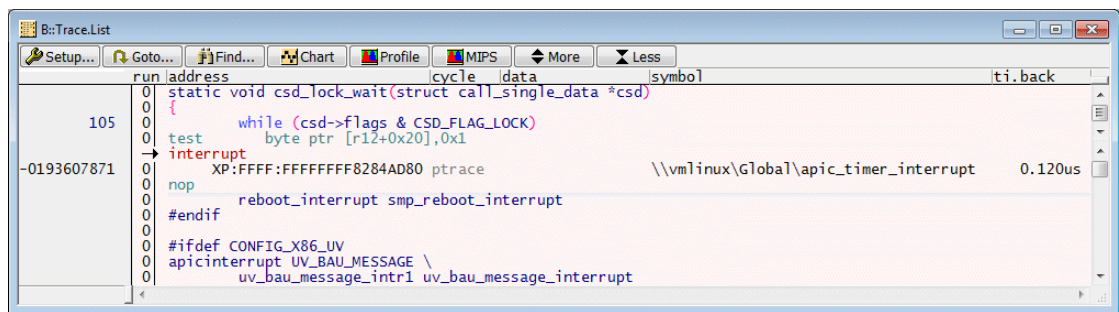
Trace.List List.ADDRESS DEFault

Sequential  
program execution  
(solid line)

Non-sequential  
program execution  
(broken line)



The column run also indicates Interrupts and TRAPs.

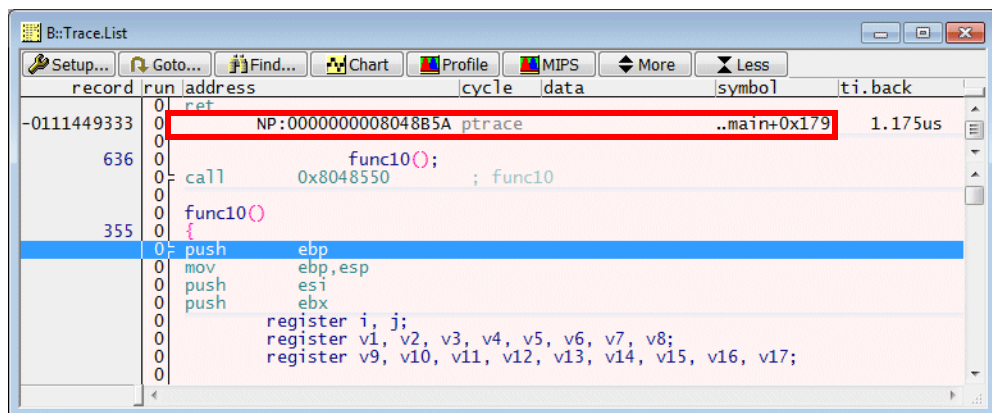


- **Column cycle**

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 screenshot shows the TRACE32 tool window titled "B::Trace.List Address Cycle sYmbol TIme.Back". It contains a table with the following columns: record, address, cycle, symbol, and ti.back. The table lists various records with their corresponding addresses, cycles, and symbols. The ti.back column shows time distances in microseconds (us). Red arrows highlight the ti.back column and specific values.

record	address	cycle	symbol	ti.back
-0026491571	NP:000000000804868E	ptrace	..\sieve_funcs\func10+0x13E	<0.005us
-0026491570	NP:00000000080486A6	ptrace	..\sieve_funcs\func10+0x156	<0.005us
-0026491569	NP:00000000080486C1	ptrace	..\sieve_funcs\func10+0x171	<0.005us
-0026491568	NP:00000000080486D9	ptrace	..\sieve_funcs\func10+0x189	0.010us
-0026491567	NP:00000000080486F4	ptrace	..\sieve_funcs\func10+0x1A4	<0.005us
-0026491566	NP:000000000804870C	ptrace	..\sieve_funcs\func10+0x1BC	<0.005us
-0026491565	NP:0000000008048727	ptrace	..\sieve_funcs\func10+0x1D7	<0.005us
-0026491564	NP:000000000804873F	ptrace	..\sieve_funcs\func10+0x1EF	<0.005us
-0026491562	NP:000000000804875A	ptrace	..\sieve_funcs\func10+0x20A	<0.005us
-0026491561	NP:0000000008048772	ptrace	..\sieve_funcs\func10+0x222	<0.005us
-0026491560	NP:000000000804878D	ptrace	..\sieve_funcs\func10+0x23D	0.010us
-0026491557	NP:000000000804885F	ptrace	..86\sieve_funcs\main+0x17E	<0.005us
-0026491556	NP:000000000804879C	ptrace	..86\sieve_funcs\func11+0x9	<0.005us
-0026491553	NP:00000000080487E3	ptrace	..6\sieve_funcs\func11+0x50	<0.005us
-0026491550	NP:0000000008048868	ptrace	..86\sieve_funcs\main+0x18A	0.010us
-0026491549	NP:0000000008048839	ptrace	..6\sieve_funcs\func13+0x50	<0.005us
-0026491525	NP:0000000008048836	ptrace	..6\sieve_funcs\func13+0x4D	0.740us
-0026491522	NP:0000000008048836	ptrace	..6\sieve_funcs\func13+0x4D	<0.005us
-0026491519	NP:0000000008048836	ptrace	..6\sieve_funcs\func13+0x4D	0.010us
-0026491516	NP:0000000008048887	ptrace	..86\sieve_funcs\main+0x1A6	<0.005us
-0026491513	NP:0000000008048893	ptrace	..86\sieve_funcs\main+0x1B2	<0.005us

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

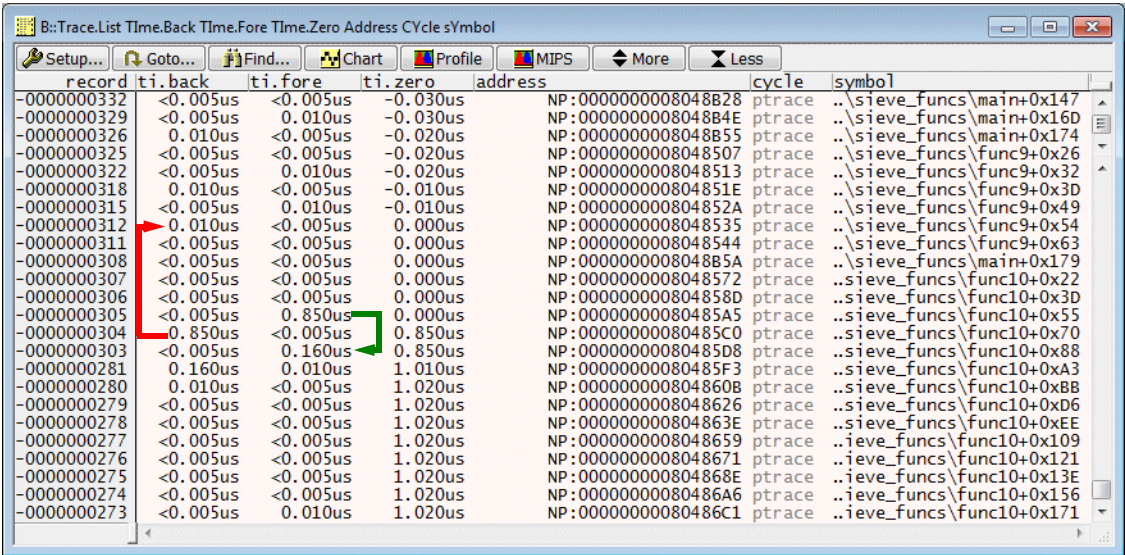
For details on the TRACE32 tool timestamp refer to [“Tool Timestamp \(POWER TRACE II / POWER TRACE III only\)”](#), page 8.



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



## Set the Global Zero Point (PowerTrace II only)

The screenshot shows the PowerTrace II application window with a table of trace records. The table has columns: record, ti.back, ti.fore, ti.zero, address, cycle, and symbol. Record -0000000312 is selected. A context menu is open, and the 'Set Zero' option is highlighted. A red arrow points from the text 'Establish the selected record as global zero point' to the 'Set Zero' option.

record	ti.back	ti.fore	ti.zero	address	cycle	symbol
-0000000318	0.010us	<0.005us	-0.010us	NP:000000000804851E	ptrace	..\sieve_funcs\func9+0x3D
-0000000315	<0.005us	0.010us	-0.010us	NP:000000000804852A	ptrace	..\sieve_funcs\func9+0x49
-0000000312	0.010us	<0.005us	0.000us	NP:0000000008048535	ptrace	..\sieve_funcs\func9+0x54
-0000000311	<0.005us	<0.005us		000000008048544	ptrace	..\sieve_funcs\func9+0x63
-0000000308	<0.005us	<0.005us		000000008048B5A	ptrace	..\sieve_funcs\main+0x179
-0000000307	<0.005us	<0.005us		000000008048572	ptrace	..\sieve_funcs\func10+0x22
-0000000306	<0.005us	<0.005us		00000000804858D	ptrace	..\sieve_funcs\func10+0x3D
-0000000305	<0.005us	0.850us		0000000080485A5	ptrace	..\sieve_funcs\func10+0x55
-0000000304	0.850us	<0.005us		0000000080485C0	ptrace	..\sieve_funcs\func10+0x70
-0000000303	<0.005us	0.160us		0000000080485D8	ptrace	..\sieve_funcs\func10+0x88
-0000000281	0.160us	0.010us		0000000080485F3	ptrace	..\sieve_funcs\func10+0xA3
-0000000280	0.010us	<0.005us		00000000804860B	ptrace	..\sieve_funcs\func10+0xBB
-0000000279	<0.005us	<0.005us		000000008048626	ptrace	..\sieve_funcs\func10+0xD6
-0000000278	<0.005us	<0.005us		00000000804863E	ptrace	..\sieve_funcs\func10+0xEE

**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).

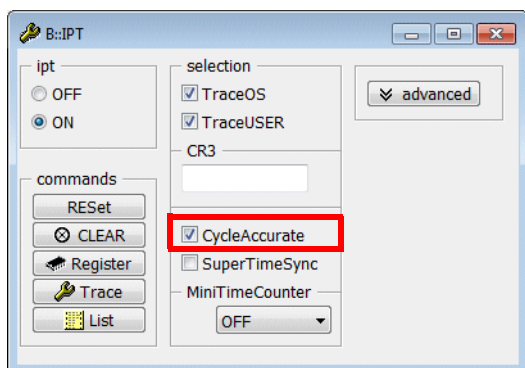
- **Cycle Accurate Tracing Cons.**

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.



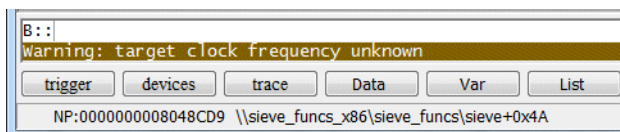
### IPT.CycleAccurate ON

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
```

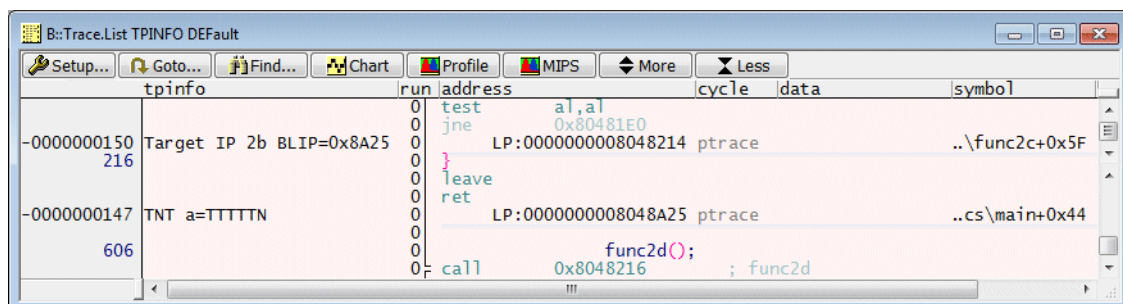


TRACE32 displays the warning above when the recorded trace information is analyzed and displayed the first time. This warning points out that all displayed time information (Time.Back, Time.Zero) might be inaccurate.

B::Trace.List CLOCKS.Back Default Time.Back.OFF						
	clocks.b	run	address	cycle	data	symbol
-0000000303	11.	0	add	ebx,+0x1		
		0	cmp	ebx,+0x12		
		0	jle	0x8048C8C		
		0	LP:0000000008048CEC ptrace			
		0	}			
		0	}			
		0	}			
		0	return anzahl;			
		0	mov	eax,[ebp-0x10]		
		0	}			
		0	add	esp,+0x10		
-0000000299	24.	0	pop	ebx		
		0	pop	esi		
		0	pop	edi		
		0	pop	ebp		
		0	ret			
		0	LP:0000000008048C80 ptrace			
		0	}			
		0	/** func40();**/			
		0	for (j = 0; j < 10; j++)			
		0	add	dword ptr [ebp-0x0C],+0x1		
		0	cmp	dword ptr [ebp-0x0C],+0x9		
		0	jle	0x8048C7B		

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
```

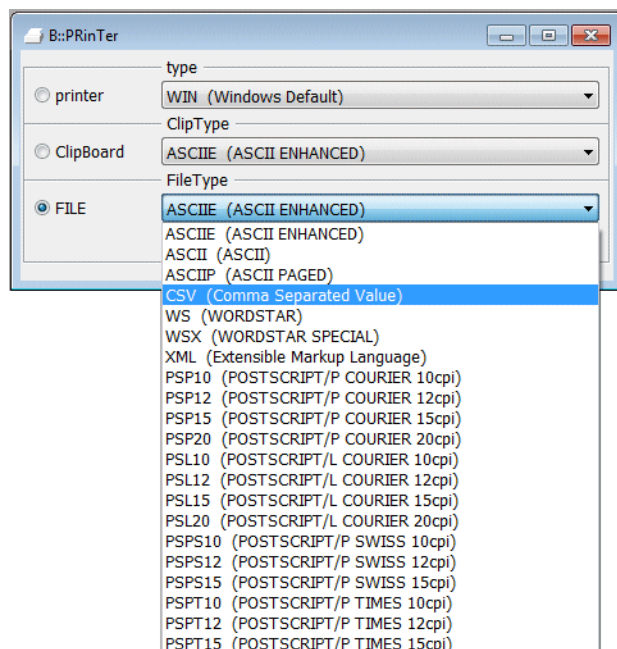


tpinfo	run	address	cycle	data	symbol
-0000000150	0	test			
216	0	jne			
	0	LP:0000000008048214		ptrace	..\func2c+0x5F
	0	leave			
-0000000147	0	ret			
606	0	LP:0000000008048A25		ptrace	..cs\main+0x44
	0	func2d();			
	0	call			
	0	0x8048216			
	0				

# 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.

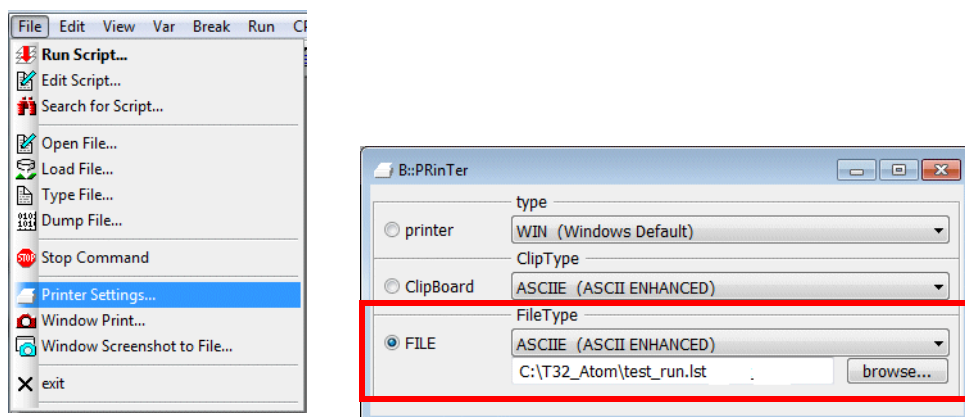


2. Save the trace contents in a compact format into a file. Load the trace contents at a subsequent date into a TRACE32 Instruction Set Simulator and analyze it there.
3. Export the STP byte stream to postprocess it with an external tool.

## 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 ASCIIIE           ; specify output format
                                   ; here enhanced ASCII

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

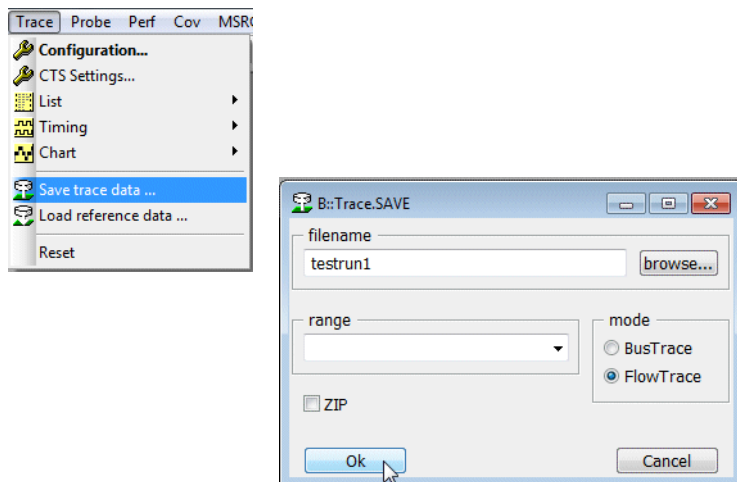
2. It might make sense to save only a part of the trace contents into the file. Use the record numbers to specify the trace part you are interested in.

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.

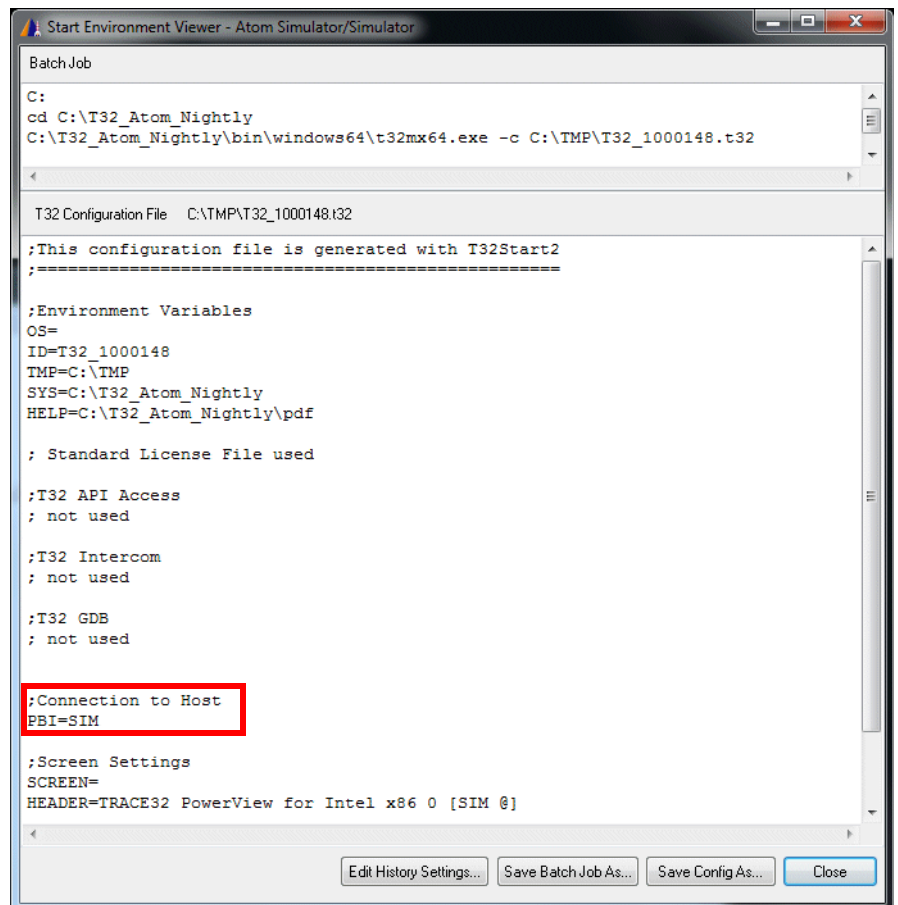
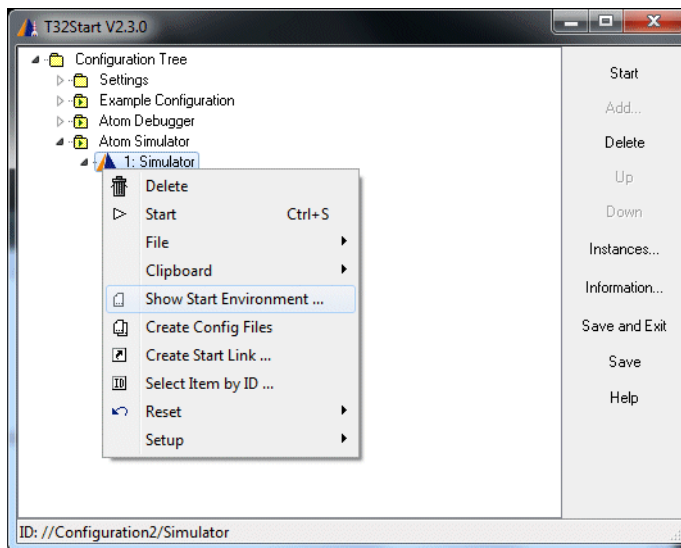
## 1. Save the contents of the trace memory into a file.



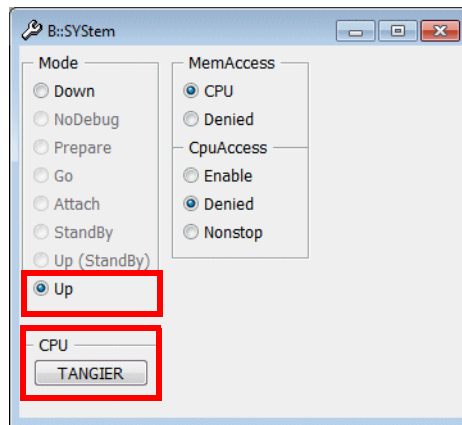
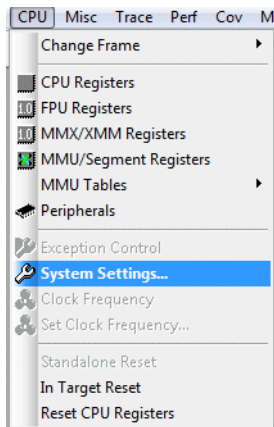
The default extension for the trace file is **.ad**.

```
Trace.SAVE testrun1
```

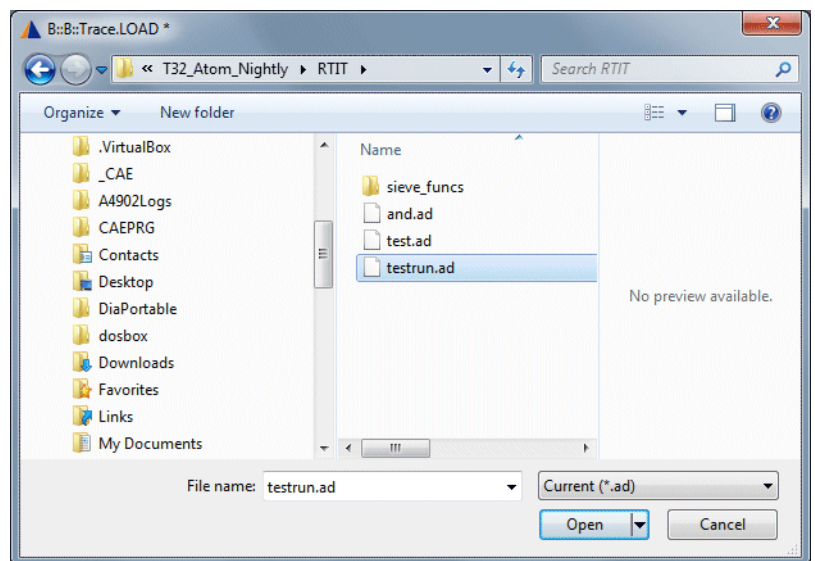
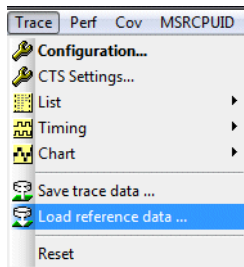
## 2. Start a TRACE32 Instruction Set Simulator (PBI=SIM).



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



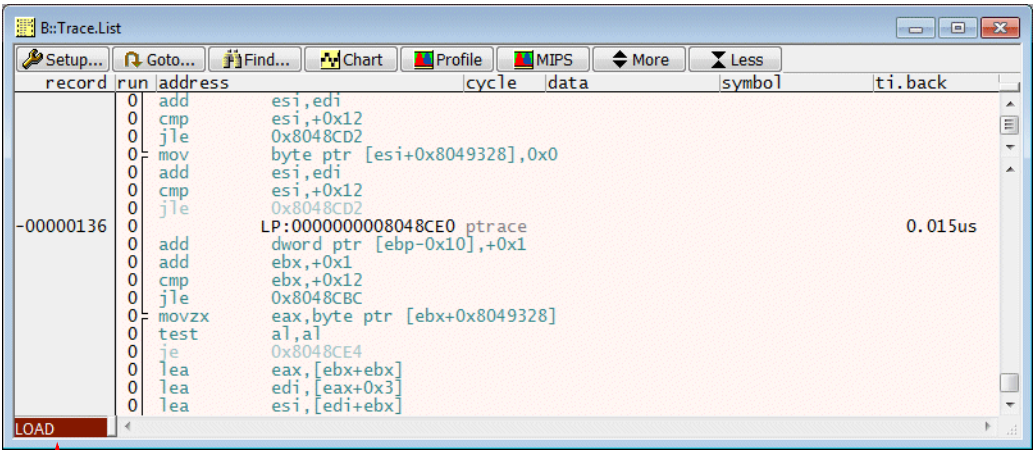
4. Load the trace file.



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.

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

## 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
```

## Export STP Byte Stream

---

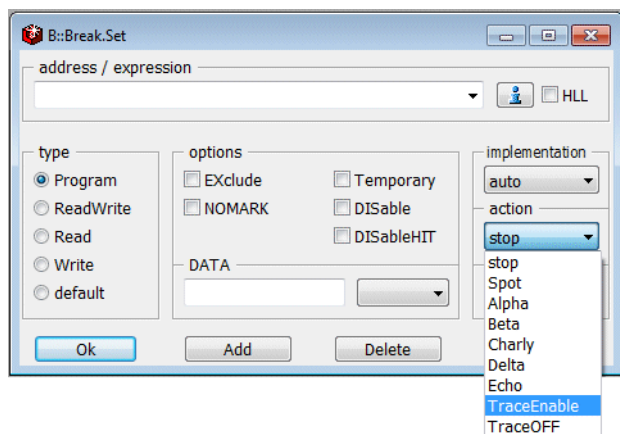
**Trace.EXPORT.TracePort** *<file>*      Export trace raw data (no timestamps).

```
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:

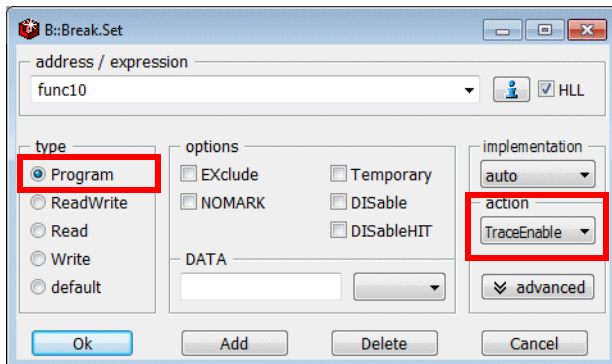
**TraceEnable** advises Intel® PT to generate program flow information for the specified address range only.

**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.

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

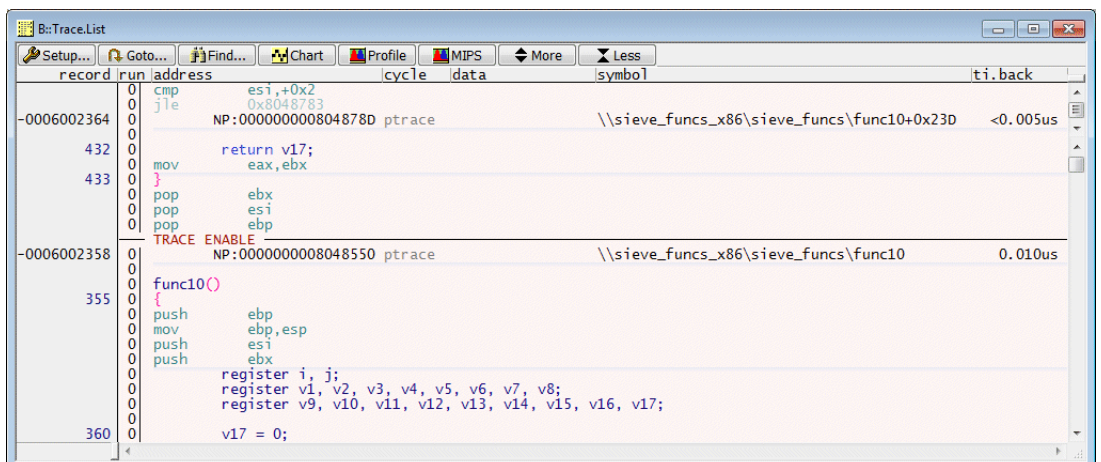
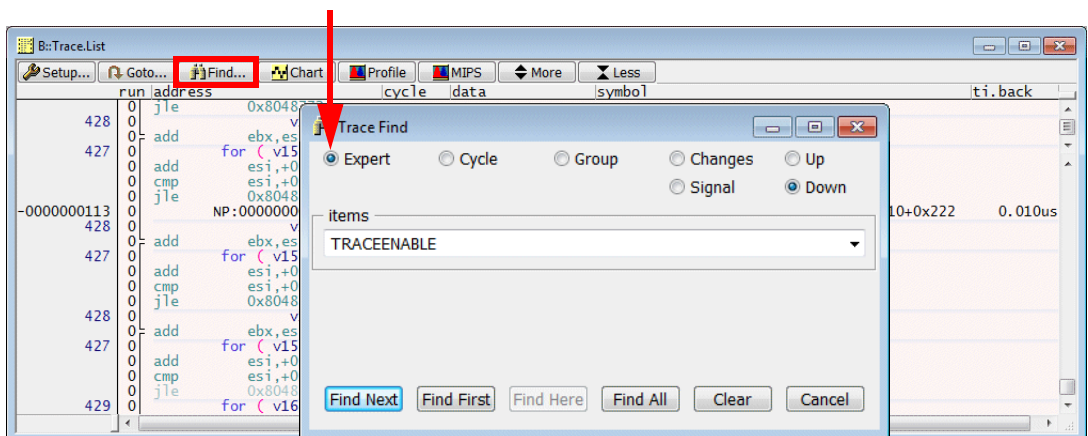
1. **Set a Program breakpoint to the address range of func10 and select the action TraceEnable.**



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
```

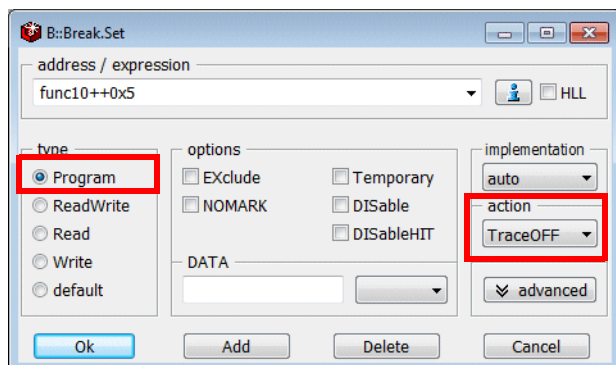
```
...
```

```
Break
```

```
Trace.List
```

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

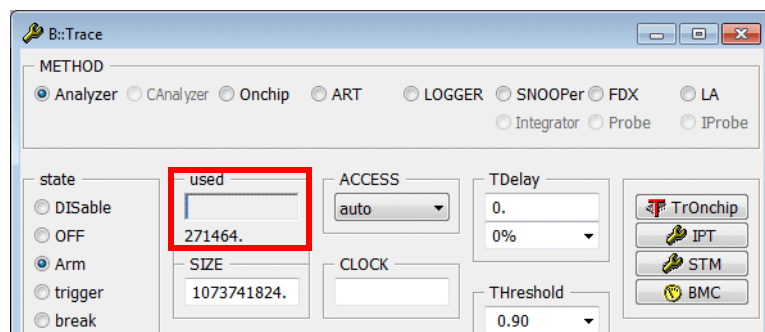
1. **Set a Program breakpoint to the start address of func10 plus 5 bytes and select the action TraceOFF.**



2. **Start the program execution.**

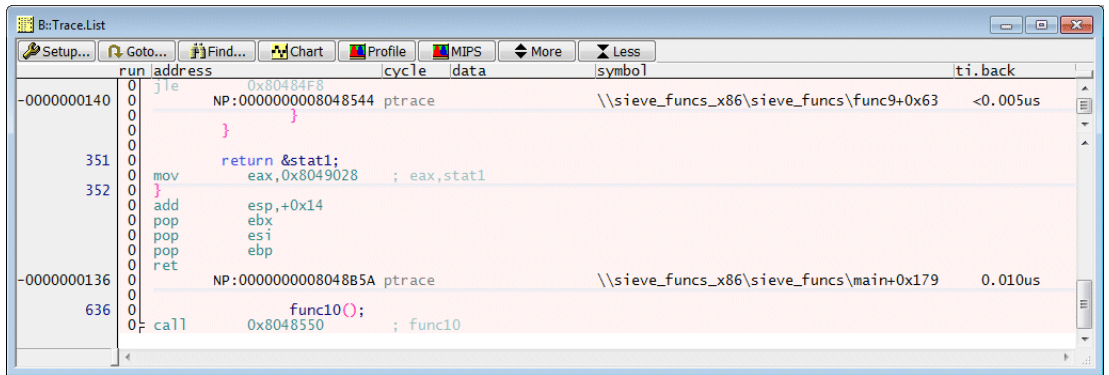
TRACE32 has, unfortunately, no way to detect that Intel® PT stopped the generation of trace information.

**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
```



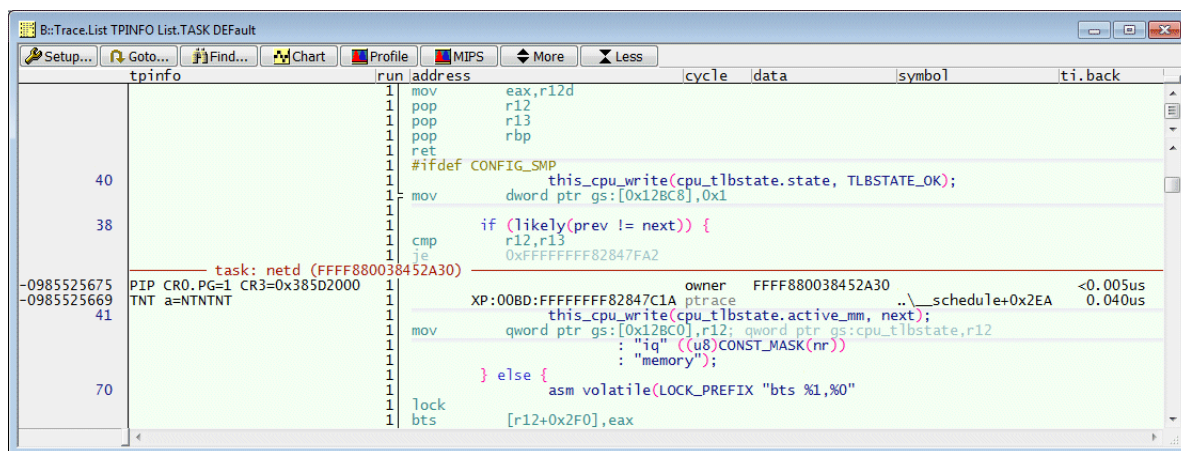
# OS-Aware Tracing

OS-aware tracing requires that OS-aware debugging is configured. For more information refer to “[OS-aware Debugging](#)” (trace32\_concepts.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 to check all assignments currently known to TRACE32. The **traceid** represents the PCID in this display.

magic	name	id	space	traceid
FFFF8800385C6D80	vold:223	223.	183.	0x00B7 385FF000
FFFF880039AA4380	vold:224	224.	183.	0x00B7 385FF000
FFFF8800384521C0	nvm_server	188.	188.	0x00BC 385EC000
FFFF880038452A30	netd	189.	189.	0x00BD 385D2000
FFFF88002FD74BF0	netd:517	517.	189.	0x00BD 385D2000
FFFF88002FD74380	netd:518	518.	189.	0x00BD 385D2000
FFFF88002FD71950	netd:521	521.	189.	0x00BD 385D2000
FFFF88002FD75460	netd:522	522.	189.	0x00BD 385D2000
FFFF88002FD76D80	netd:523	523.	189.	0x00BD 385D2000
FFFF88002FD70000	netd:524	524.	189.	0x00BD 385D2000
FFFF8800384532A0	debuggerd	190.	190.	0x00BE 38485000
FFFF880038453B10	surfaceflinger	191.	191.	0x00BF 38601000
FFFF880038450000	surfaceflinger:Binder_1	231.	191.	0x00BF 38601000

tpinfo	run	address	cycle	data	symbol	ti.back
	1	pop r13				
	1	pop rbp				
	1	ret				
38	1	if (likely(prev != next)) {				
	1	mov rcx,qword ptr [rbp-0x40]				
	1	cmp rbx,r14				
	1	je 0xFFFFFFF82174C19				
-0986647737	1	contextid: 00000000CD93000		context 00000000CD93000		<0.005us
-0986647730	1	PIP CR0.PG=1 CR3=0xCD93000		XP:FFFF:FFFFFFFF821747F3 ptrace	..\flush_oldd_exec+0x303	0.040us
40	1	TNT a=NNNNNT		#ifdef CONFIG_SMP		
41	1	this_cpu_write(cpu_tlbstate.state, TLBSTATE_OK);		mov dword ptr gs:[0x128C8],0x1		
	1	this_cpu_write(cpu_tlbstate.active_mm, next);		mov qword ptr gs:[0x0],rbx		
	1	: "iq" ((u8)CONST_MASK(nr))		: "memory";		
	1	} else {				

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.

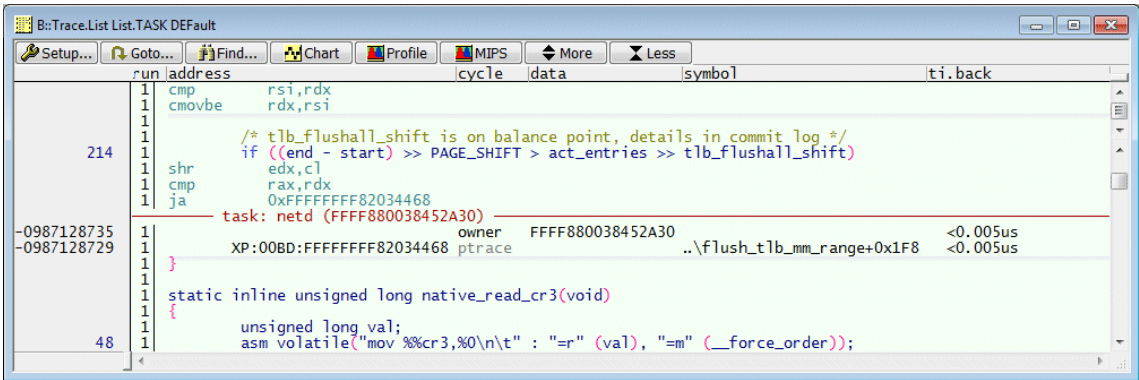
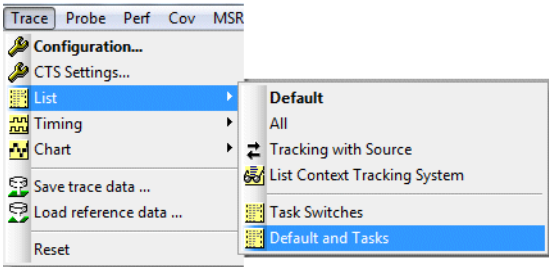
run	address	cycle	data	symbol	ti.back
-1016945789	1	XP:FFFF:00000000F770C9F2	unknown		<0.005us
-1016945788	1	XP:FFFF:00000000F770C9F2	unknown		<0.005us
-1016945786	1	XP:FFFF:00000000F770C9F2	unknown		<0.005us
-1016945781	1	XP:FFFF:00000000F76E7F00	unknown		0.040us
-1016945780	1	XP:FFFF:00000000F76E7F00	unknown		<0.005us
-1016945777	1	XP:FFFF:00000000F76E81C2	unknown		<0.005us
-1016945776	1	XP:FFFF:00000000F76E81C2	unknown		0.040us
-1016945775	1	XP:FFFF:00000000F76E81C2	unknown		<0.005us

#### 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.

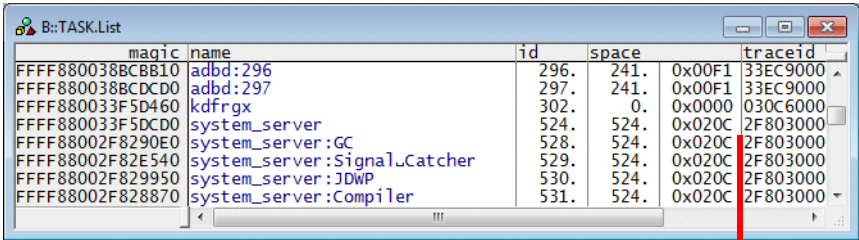
RTIT was implemented on very few devices, then it was extended to the Intel Processor Trace which supports the process switching trace. The RTIT trace is also covered by TRACE32 using the IPT command group.

# Program Flow and Process Switches



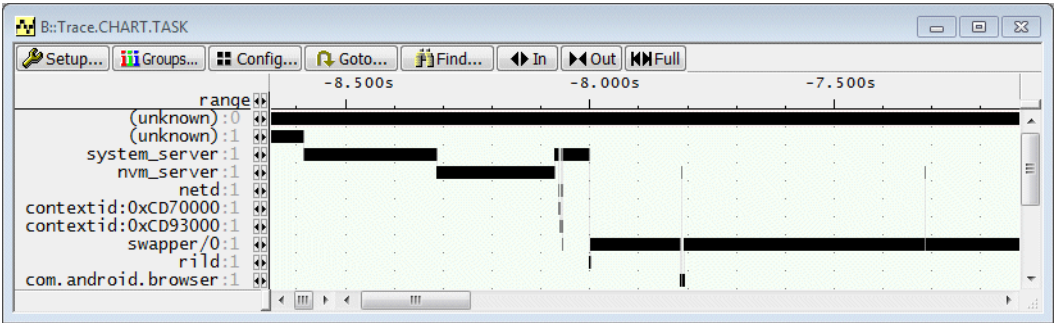
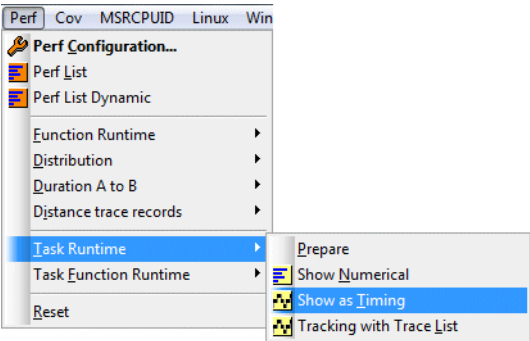
Trace.List List.TASK DEFault ; display trace listing with  
; decoded task switch information

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



mag1c	name	id	space	traceid
FFFF880038BCB810	adbd:296	296.	241.	0x00F1 33EC9000
FFFF880038BCDCD0	adbd:297	297.	241.	0x00F1 33EC9000
FFFF880033F5D460	kdfgrx	302.	0.	0x0000 030C6000
FFFF880033F5DCD0	system_server	524.	524.	0x020C 2F803000
FFFF88002F8290E0	system_server:GC	528.	524.	0x020C 2F803000
FFFF88002F82E540	system_server:Signal_Catcher	529.	524.	0x020C 2F803000
FFFF88002F829950	system_server:JDWP	530.	524.	0x020C 2F803000
FFFF88002F828870	system_server:Compiler	531.	524.	0x020C 2F803000

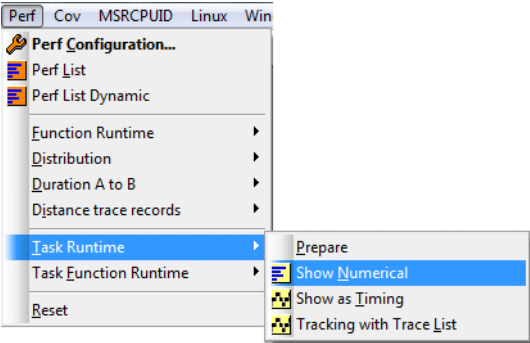
Threads do not have their own traceid



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

<b>Trace.Chart.TASK [/SplitCORE]</b>	Display process time chart - graphical display - split the results per core - sort the results per recording order
<b>Trace.Chart.TASK /MergeCORE</b>	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.



The image shows the 'B:\Trace.STATistic.TASK' window. It displays a table of process runtime statistics. The table has columns for 'range', 'total', 'min', 'max', 'avr', 'count', 'ratio%', and '1%'. The data is sorted by recording order.

range	total	min	max	avr	count	ratio%	1%
(unknown):0	8.669s	-	8.669s	8.669s	0.	-	←
(unknown):1	80.357ms	80.357ms	80.357ms	80.357ms	0.	-	←
system_server:1	5.725s	4.625us	5.383s	212.032ms	27.	66.036%	←
nvm_server:1	244.484ms	81.890us	242.902ms	40.747ms	6.	2.820%	←
netd:1	1.116ms	2.605us	267.280us	111.633us	10.	0.012%	←
contextid:0xCD70000:1	1.920ms	1.920ms	1.920ms	1.920ms	1.	0.022%	←
contextid:0xCD93000:1	2.195ms	4.330us	2.064ms	548.821us	4.	0.025%	←
swapper/0:1	2.579s	109.235us	492.556ms	184.240ms	14.	29.753%	←
rild:1	435.045us	117.705us	198.870us	145.015us	3.	0.005%	←
com.android.browser:1	16.972ms	47.455us	8.422ms	4.243ms	4.	0.195%	←
com.android.music:1	1.373ms	4.685us	351.215us	85.786us	16.	0.015%	←
zygote:1	7.790ms	307.480us	3.727ms	1.948ms	4.	0.089%	←

**Trace.STATistic.TASK [/SplitCORE]**

- Process runtime statistic
- numerical display
  - split the results per core
  - sort the results per recording order

**Trace.STATistic.TASK /MergeCORE**

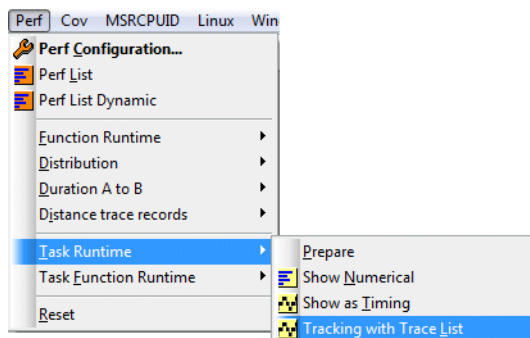
- Process runtime statistic
- numerical display
  - merge the results of all cores

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.

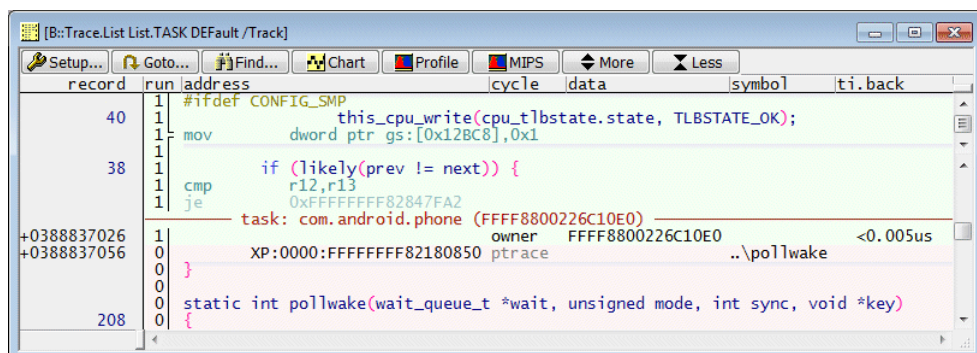
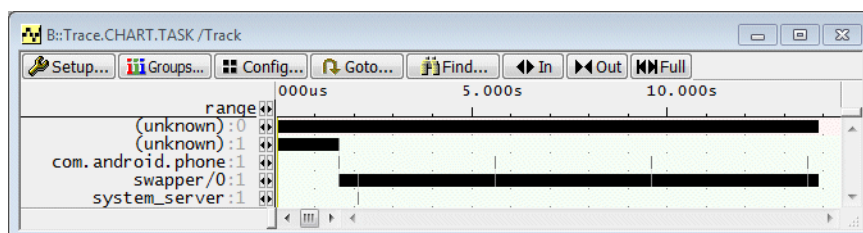


```
Trace.Chart.TASK /Track                ; open process time chart
                                         ; window

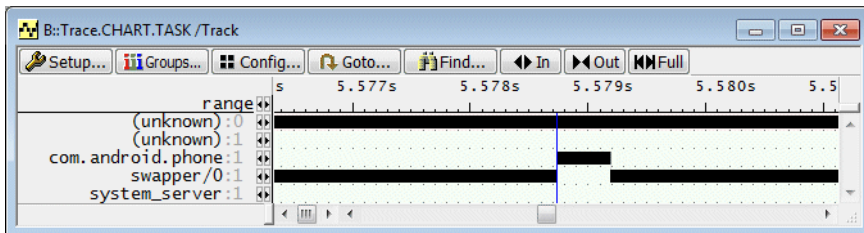
Trace.List List.TASK Default /Track     ; open a default trace
                                         ; listing that includes
                                         ; process information

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

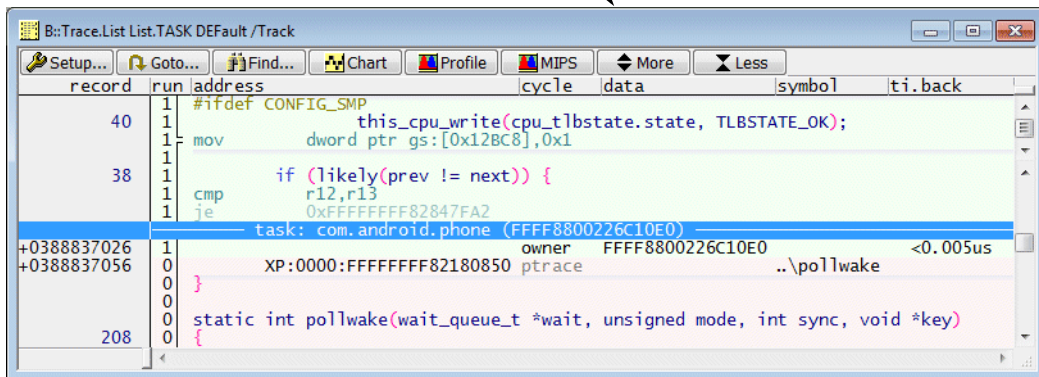
; tracking between trace windows is based on the timestamp
; information
```



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



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





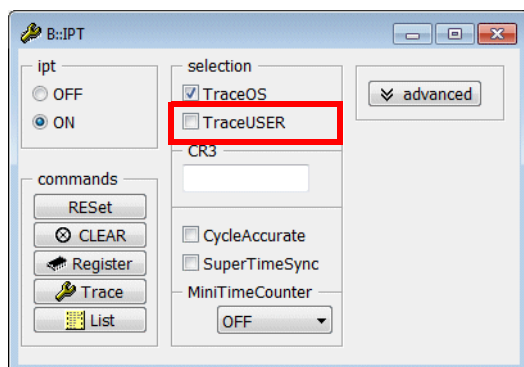
## Filtering by Privilege Level

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
```

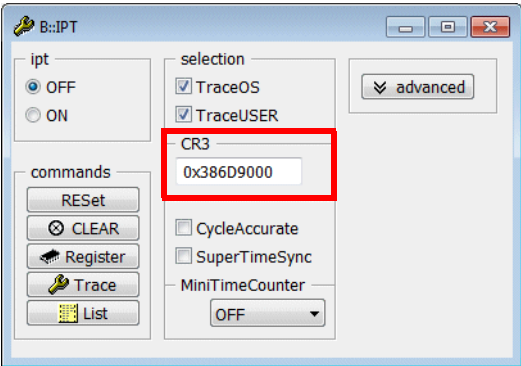
# Filtering by Process

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.

```

B:\Trace.List
run address cycle data symbol ti.back
1506 1 trace_save_cmdline(task);
1 1 }
1 1 pop rbx;
1 1 pop rbp;
1 1 xchg eax, eax;
1 1 ret;
+0000028219 1 XP:00D8:FFFFFFFF8210E6BF ptrace ..sched_switch+0x2F 13.194ms
66 1 if (!tracer_enabled || sched_stopped)
1 1 mov eax, dword ptr [rip+0x10DC35B]
1 1 test eax, eax
1 1 je 0xFFFFFFFF8210E729
+0000028276 0 TRACE ENABLE
0 0 XP:00D8:FFFFFFFF82847C4F ptrace ..\_schedule+0x31F
0 0 : "iq" ((u8)~CONST_MASK(nr));
0 0 } else {
0 0 asm volatile(LOCK_PREFIX "btr %1,%0"
0 0 lock btr [r13+0x2F0], eax
0 0 ...
0 0 /* Re-load page tables */
0 0 load_cr3(next->pgd);

```

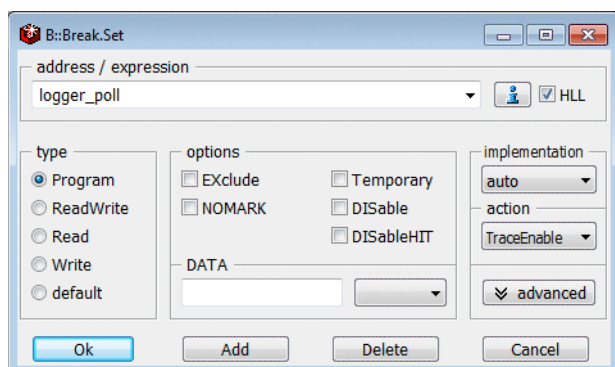
TRACE ENABLE indicates the re-start of the program flow trace generation.

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.

## Filter on Function executed by Process

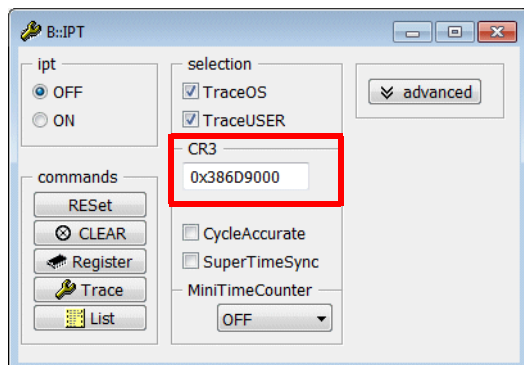
**Example:** Advise Intel® PT to generate only program flow information when the function “logger\_poll” is running in the context of the process “logcat”.

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.

record	run	address	cycle	data	symbol	ti.back
662	1	if (log->w_off != reader->r_off)				
	1	ret  = POLLIN   POLLRDNORM;				
	1	mutex_unlock(&log->mutex);				
	1	call 0xFFFFFFFF828464D0; mutex_unlock				
+0000738672	1	TRACE ENABLE				
	1	XP:0102:FFFFFFFF8262CA67 ptrace			\\vm\linux\logger\logger_poll+0x87	<0.005us
665	1	return ret;				
	1	mov eax, ebx				
	1	pop rbx				
	1	pop r12				
	1	pop r13				
	1	pop rbp				
	1	ret				
+0000738686	1	TRACE ENABLE				
	1	XP:0102:FFFFFFFF8262C9E0 ptrace			\\vm\linux\logger\logger_poll	0.080us

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

## Belated Analysis

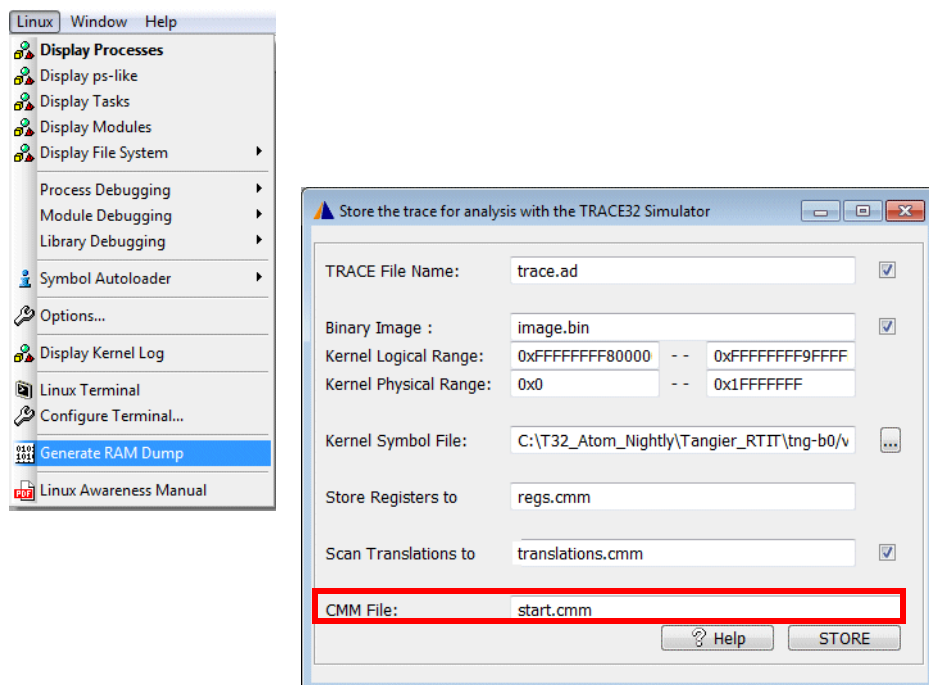
Postprocessing of recorded trace information with the TRACE32 Instruction Set Simulator requires complex preparations if an operating system that uses dynamic memory management to handle processes is used (e.g. Linux).

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.



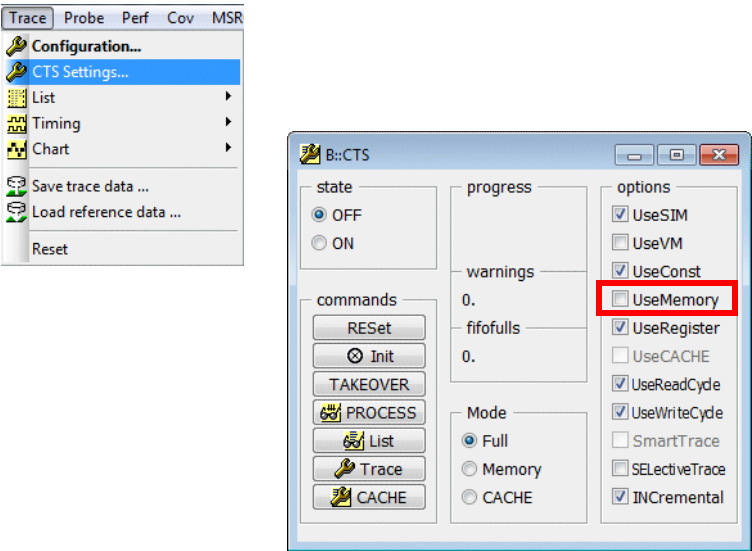


# Trace-based Debugging (CTS)

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

## Setup

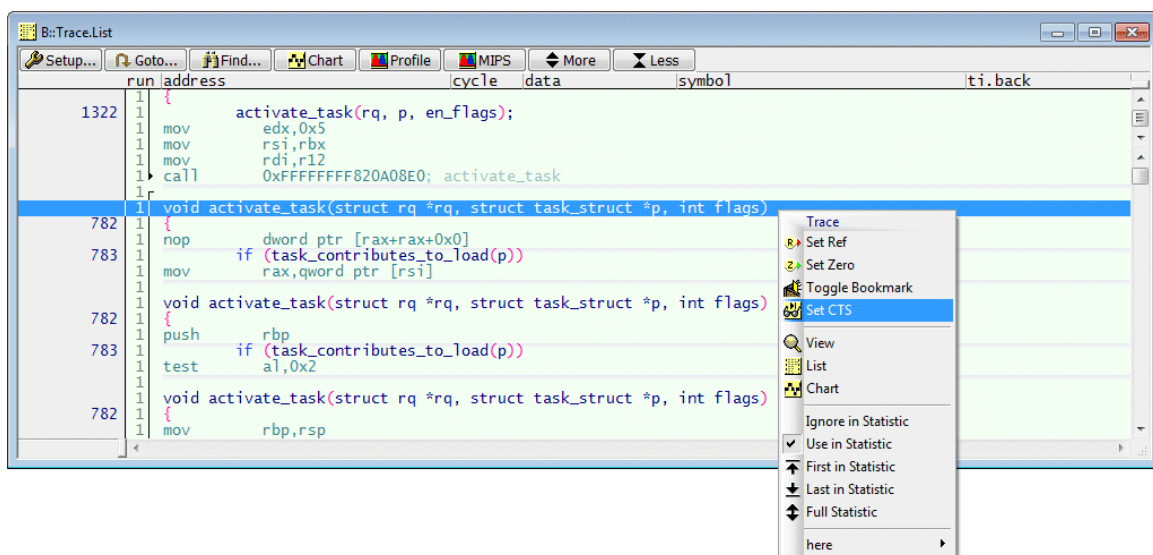
Since Intel® PT does not provide any information on read/write accesses, **UseMemory** has to be unchecked. A full explanation on this is given later in the chapter “**CTS Technique**”, page 93.



**CTS.UseFinalMemory OFF**

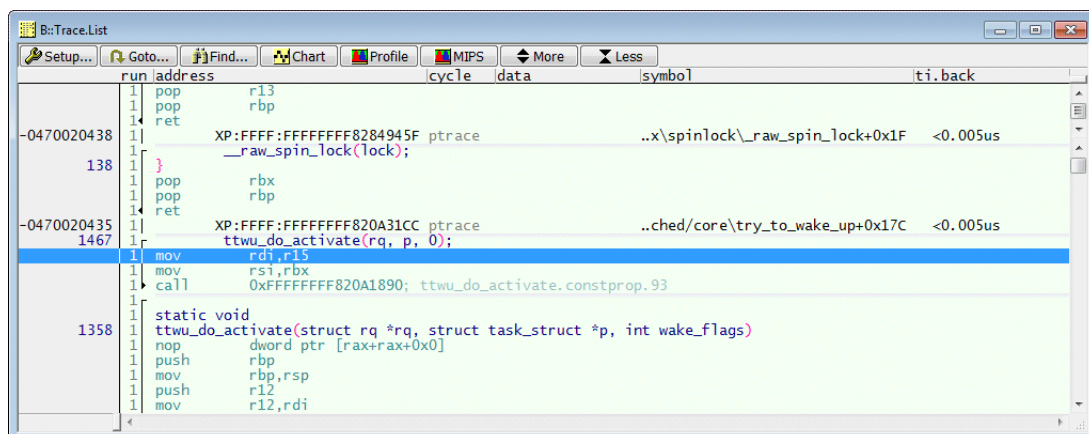


Specify the starting point for the trace re-run by selecting **Set CTS** from the Trace pull-down menu. The starting point in the example below is the entry to the function **activate\_task** executed by core 1.



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

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



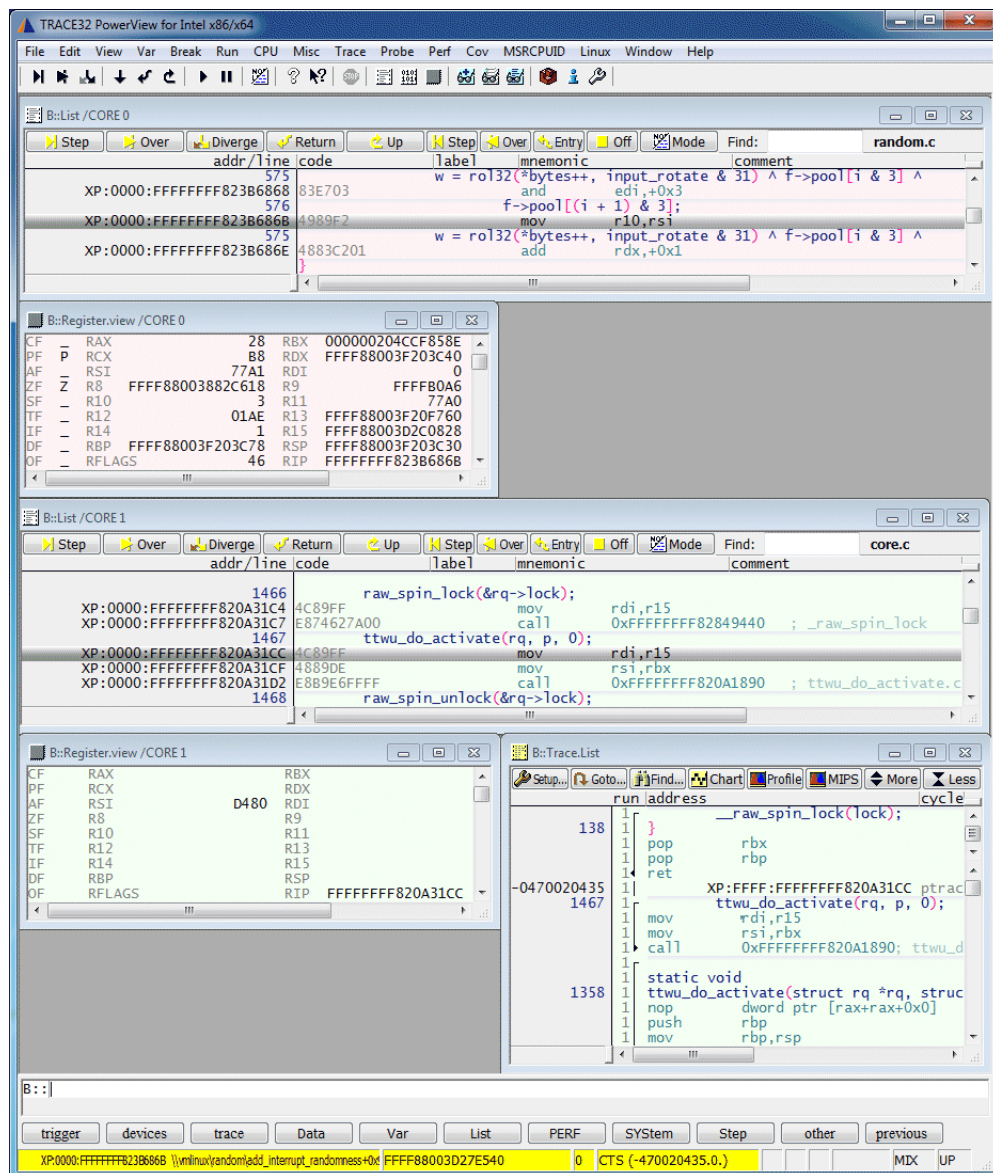
- The TRACE32 PowerView GUI does no longer show the current state of the target system, but it shows the target state as it was, when the starting point instruction was executed. This display mode is called CTS View.

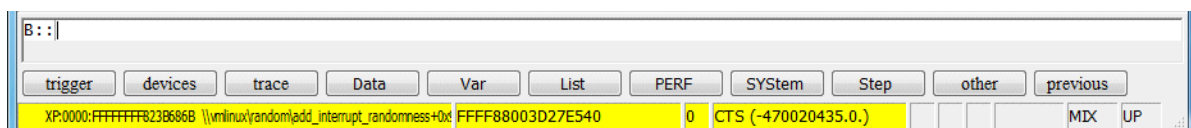
**CTS View** means:

- 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 commands

Backward debugging commands

addr/line	code	label	mnemonic	comment
1467		ttwu_do_activate	(rq, p, 0);	
XP:0000:FFFFFFFF820A31CC	4C89FF		mov rdi,r15	
XP:0000:FFFFFFFF820A31CF	4889DE		mov rsi,rbx	
XP:0000:FFFFFFFF820A31D2	E8B9E6FFFF		call 0xFFFFFFFF820A1890	; ttwu_do_activate.c
1468		raw_spin_unlock	(&rq->lock);	
XP:0000:FFFFFFFF820A31D7	4C89FF		mov rdi,r15	
XP:0000:FFFFFFFF820A31DA	E881617A00		call 0xFFFFFFFF82849360	; _raw_spin_unlock
			#endif /* CONFIG_SMP */	

## Forward Debugging

Step

Over

Diverge

Return

Up

Single step

Step over call

No function

Re-run until function exit

No function

## Backward Debugging

Step

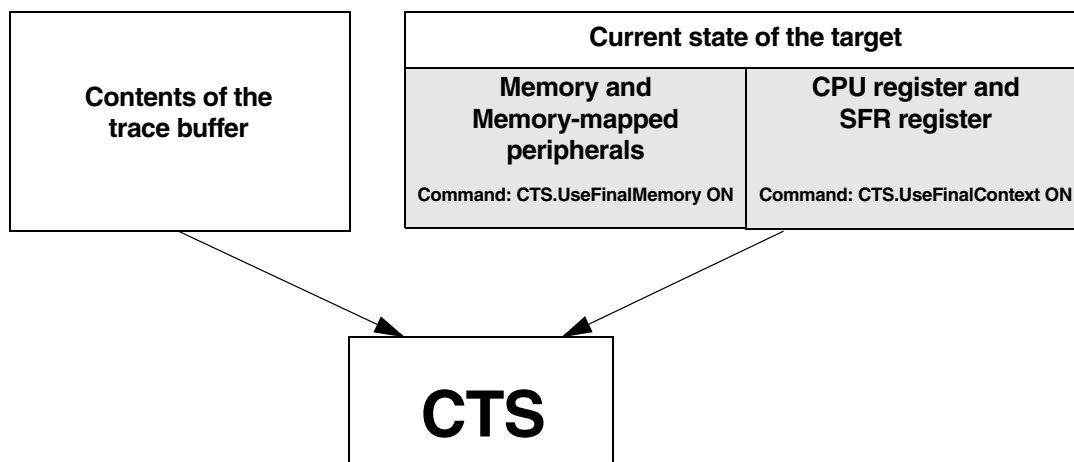
Over

Entry

Single step backward

Step backward over call

Re-run backward to function entry



**CTS.UseFinalMemory ON** Default setting within TRACE32

If **CTS.UseFinalMemory** 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.UseFinalMemory** has to be set to OFF.

If **CTS.UseFinalMemory** 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.UseFinalContext ON** Default setting within TRACE32

If **CTS.UseFinalContext** 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.UseFinalContext** 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

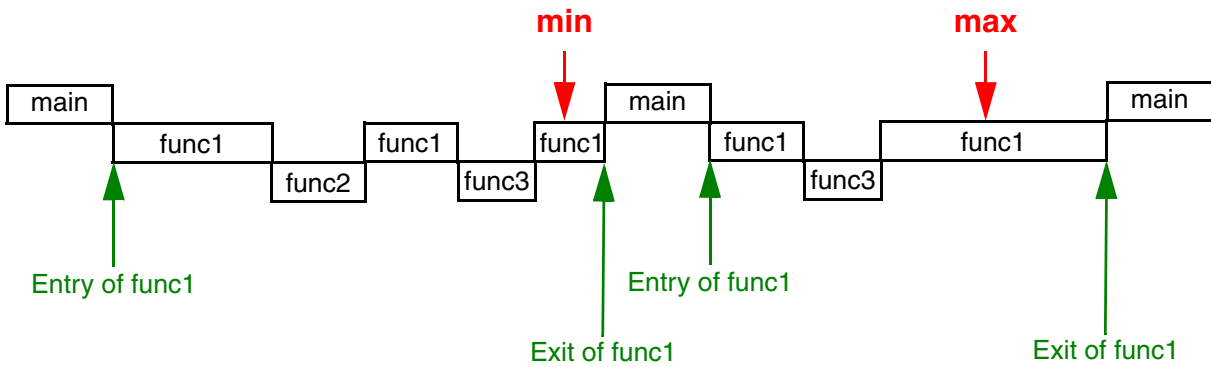
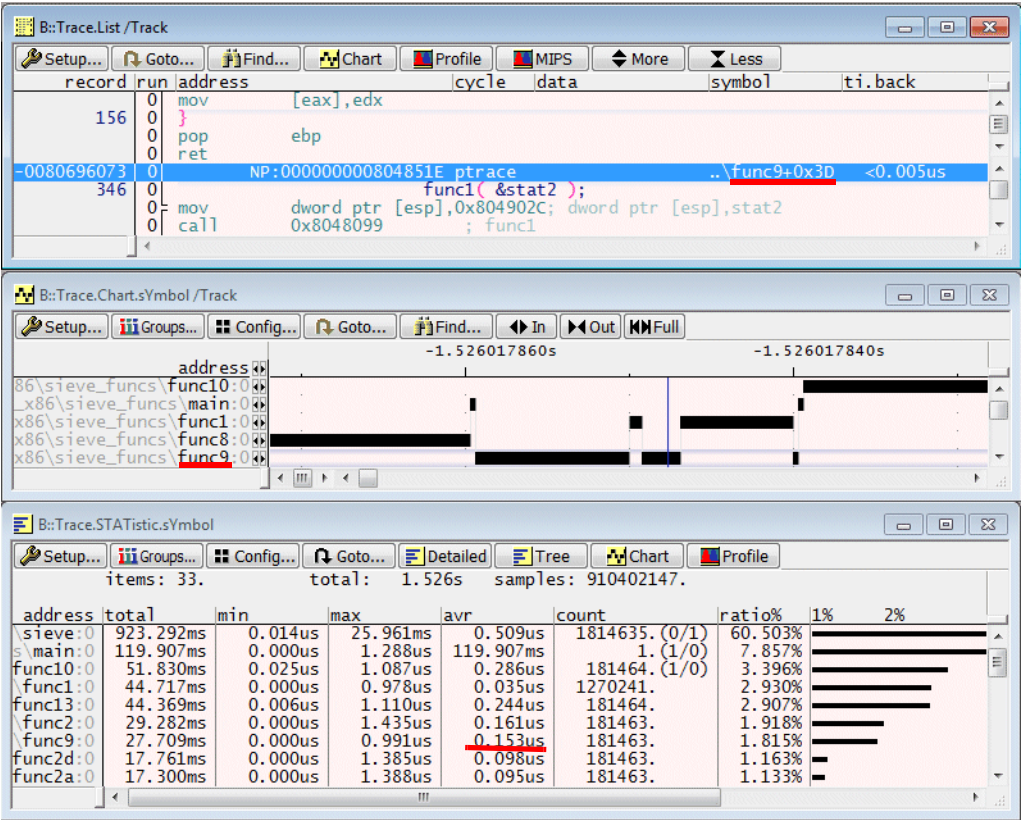
---

TRACE32 provides two methods to analyze function run-times:

- Flat analysis
- Nesting analysis

# Basic Knowledge about Flat 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.

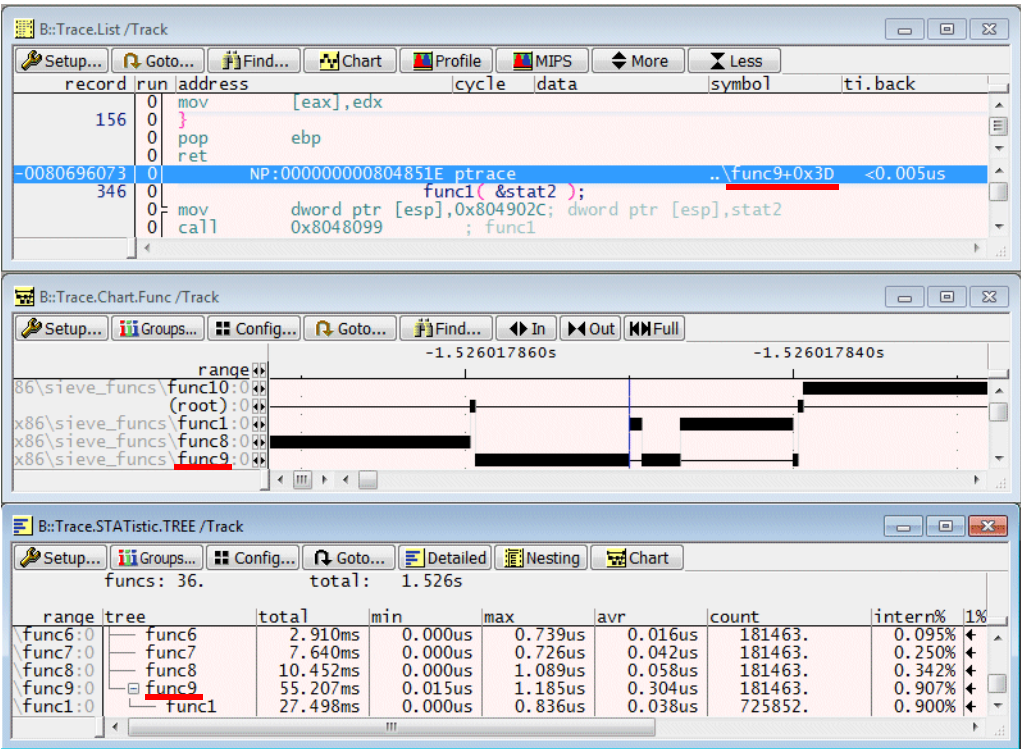


min	shortest time continuously in the address range of the function/symbol region
max	longest time continuously in the address range of the function/symbol region

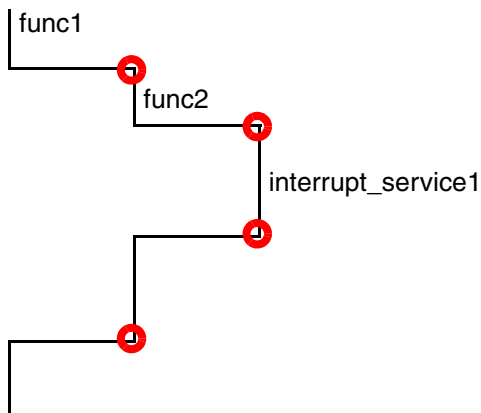


# Basic Knowledge about Nesting Analysis

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

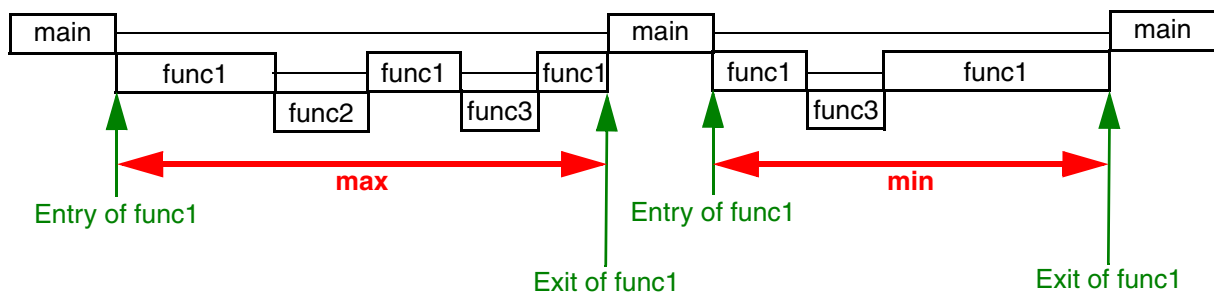






In order to display a nesting function run-time analysis TRACE32 analyzes the structure of the program execution by processing the trace information. The focus is put on the transition between functions (see picture above). The following events are of interest:

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**



<b>min</b>	shortest time within the function including all subfunctions and traps
<b>max</b>	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 than the flat analysis. Missing or tricky function entries/exits may require additional setups before nesting analysis can be used.

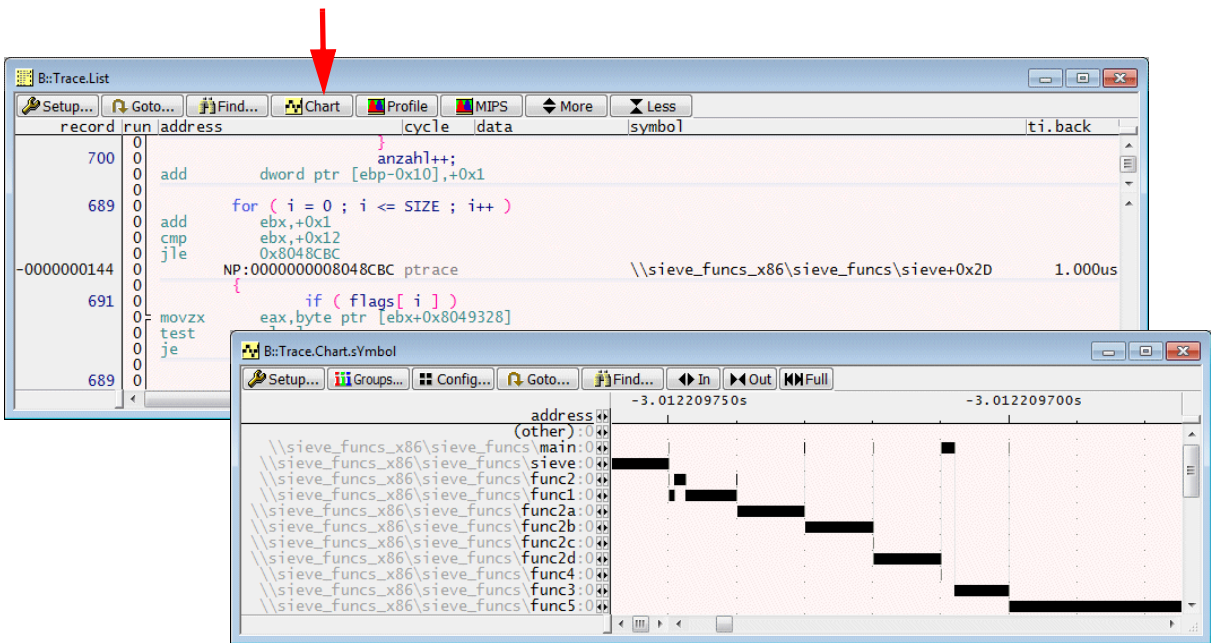
# Flat Function-Runtime Analysis

**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

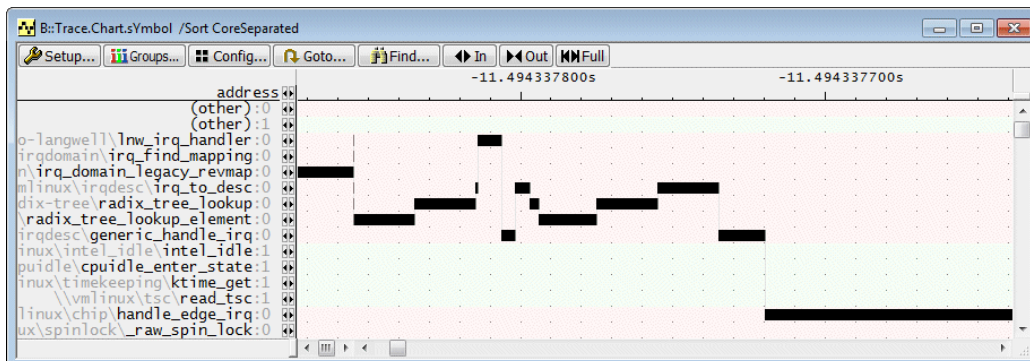
Pushing the **Chart** button in the **Trace.List** window opens a **Trace.Chart.sYmbol** window



**Trace.Chart.sYmbol** [/SplitCore /Sort CoreTogether]

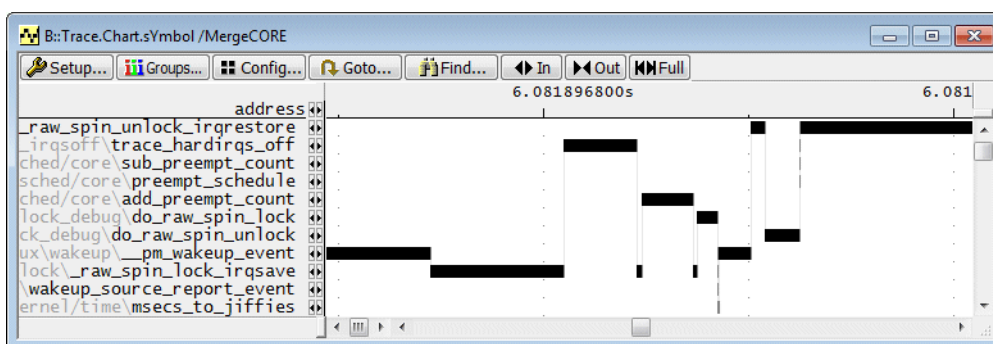
- Flat function run-time analysis
- graphical display
  - split the result per core
  - sort results per core and then per recording order

# Core Options



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

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



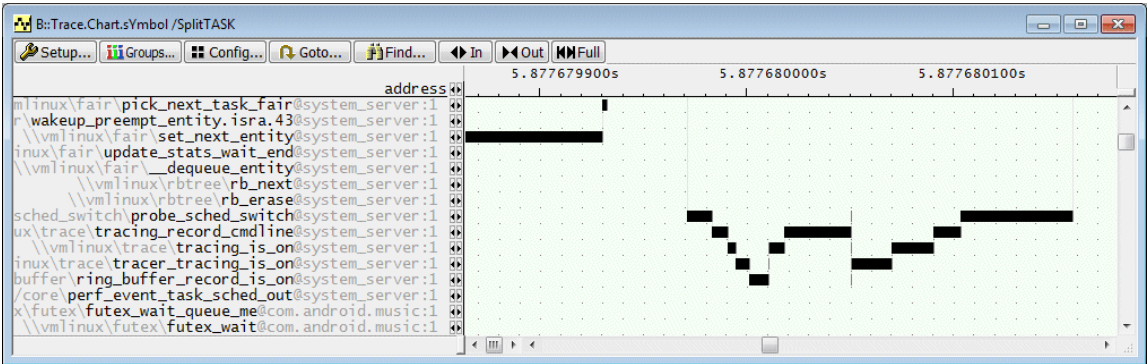
## Trace.Chart.sYmbol /MergeCORE

- 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.

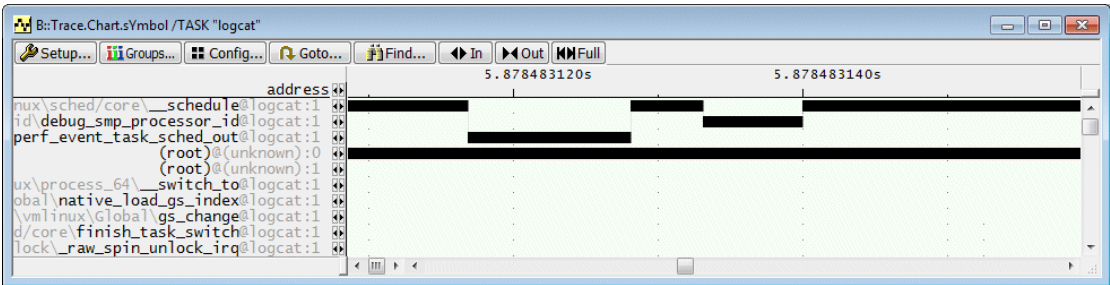
Trace.Chart.sYmbol /SplitTASK

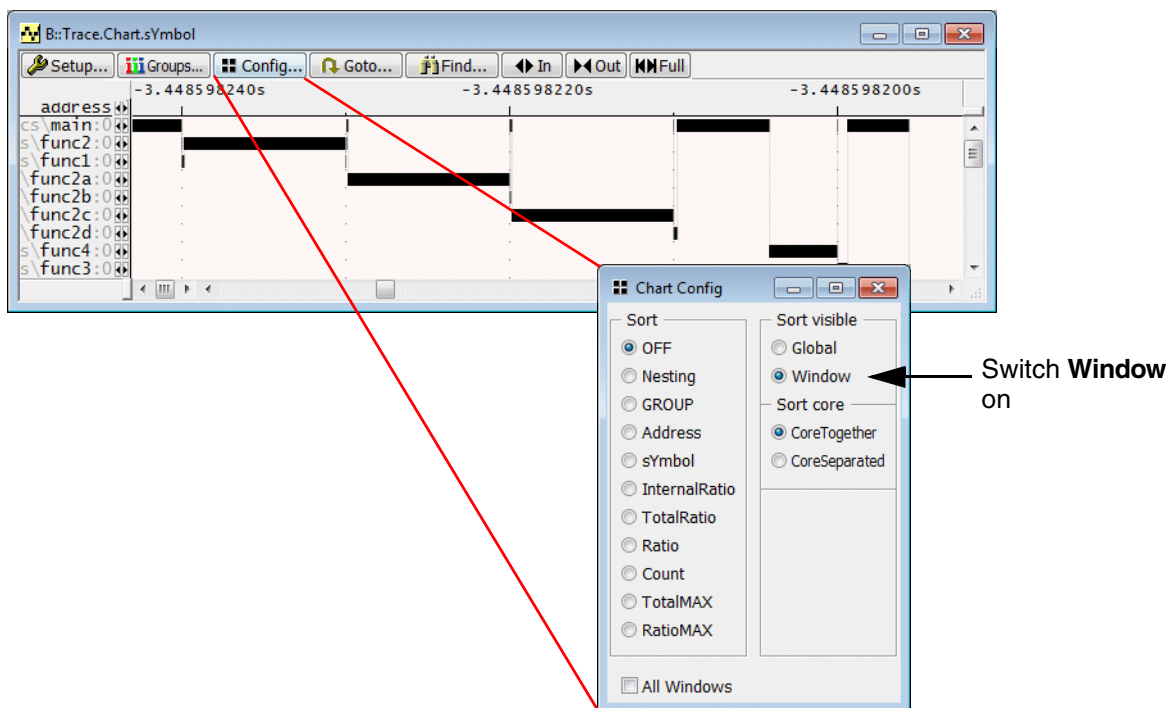
Display function time chart including process information (OS, OS+MMU only)



Trace.Chart.sYmbol /TASK <name>

Display function time chart for specified process (OS, OS+MMU only)

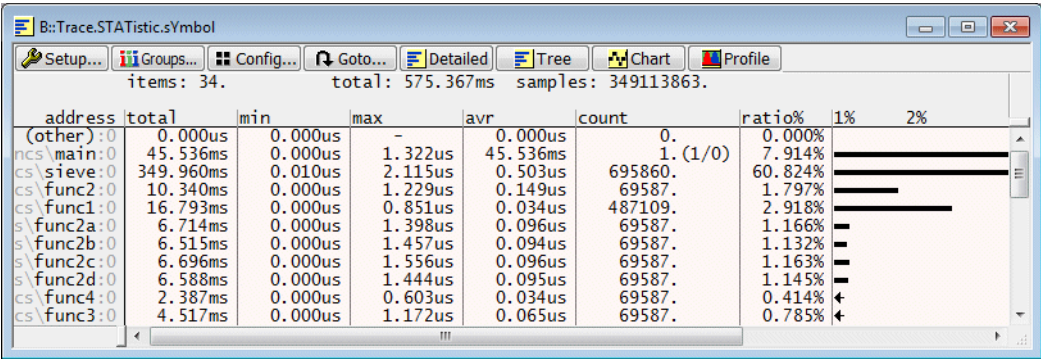




If **Window** in the **Sort visible** field is switched ON in the **Chart Config** window, the functions that are active at the selected point of time are visualized in the scope of the **Trace.Chart.sYmbol** window. This is helpful especially if you scroll horizontally.

# Function Run-time Statistic

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

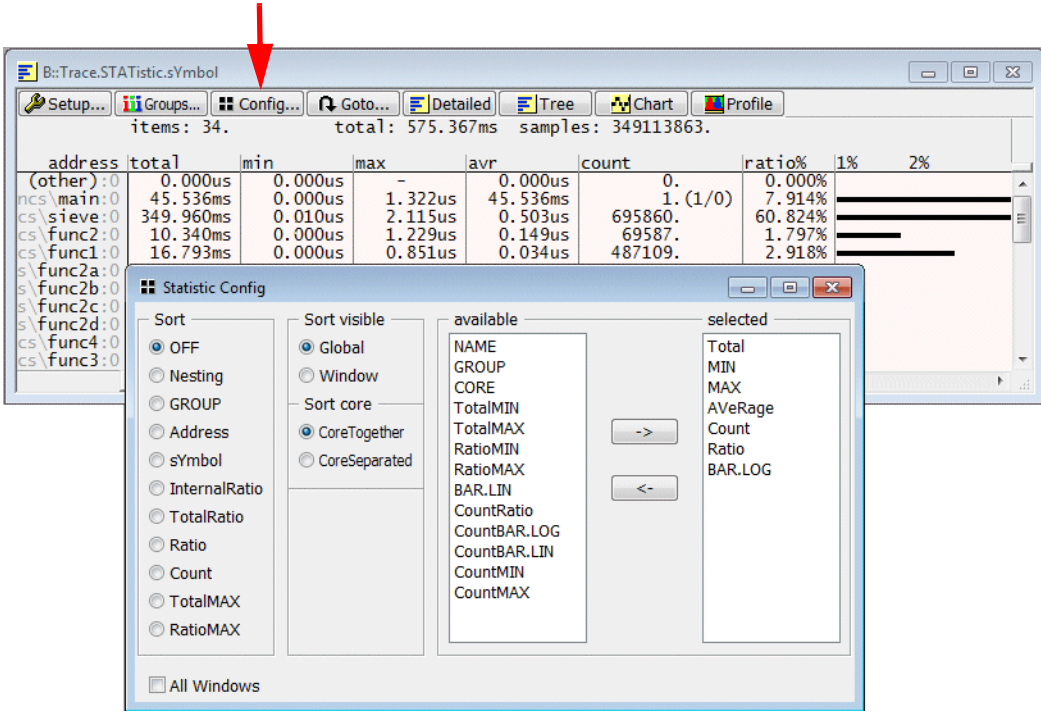


survey	
item	number of recorded functions/symbol regions
total	time period recorded by the trace
samples	total number of recorded changes of functions/symbol regions (program flow continuously in the address range of a function/symbol region)

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)

<b>count</b>	number of new entries (start address executed) into the address range of the function/symbol region
<b>ratio</b>	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

<b>Trace.PROfileChart.sYmbol</b>	Display dynamic program behavior graphically.
<b>MIPS.PROfileChart.sYmbol</b>	Display MIPS for all program symbols graphically.
<b>MIPS.STATistic.sYmbol</b>	Display MIPS for all program symbols numerically.



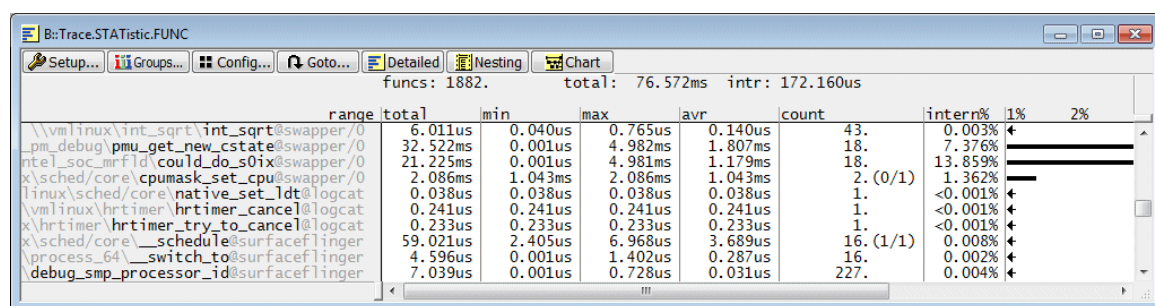
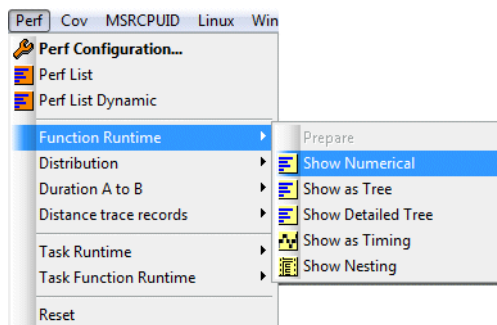
## Nesting Function Analysis OS

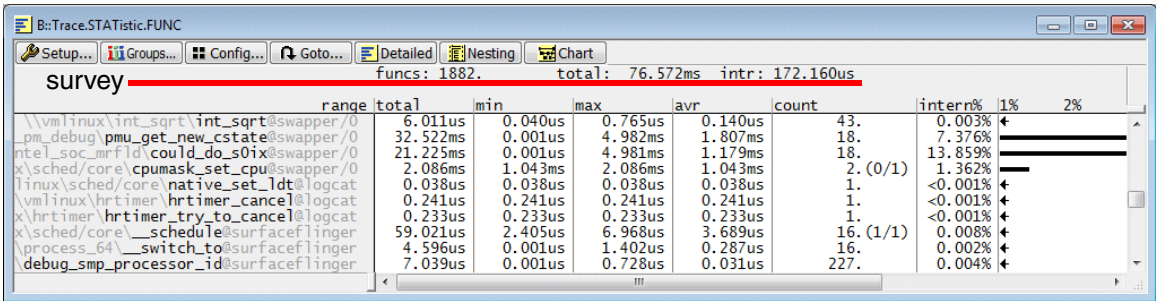
Function nesting analysis for OS requires that OS-aware debugging is configured. For more information refer to **“OS-aware Debugging”** (trace32\_concepts.pdf).

## Trace.STATistic.Func

## Nesting function run-time analysis

- numeric display
- core information is discarded exceptions are the @ (unknown) task and the @ (interrupt) task

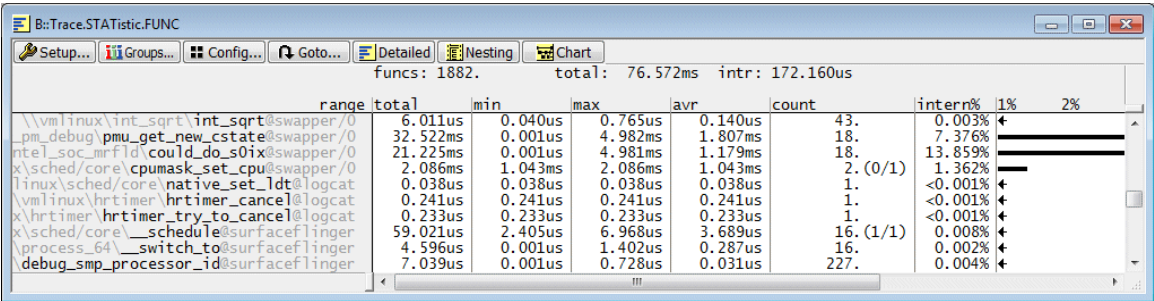




survey	
func	number of functions in the trace
total	total measurement time
intr	total time in interrupt service routines

OVERFLOW    funcs: 174.    total: 1.392s    intr: 902.302us    stack overflow at 186892464.

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 <a href="mailto:support@lauterbach.com">support@lauterbach.com</a> .
<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 <a href="#">Trace.STATistic.TREE</a> can help you to identify the function. If you need further help please contact <a href="mailto:support@lauterbach.com">support@lauterbach.com</a> .
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 <a href="#">Trace.STATistic.TREE</a> can help you to identify the function. If you need further help please contact <a href="mailto:support@lauterbach.com">support@lauterbach.com</a> .



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

\\vmlinux\hrtimer\hrtimer\_cancel@logcat

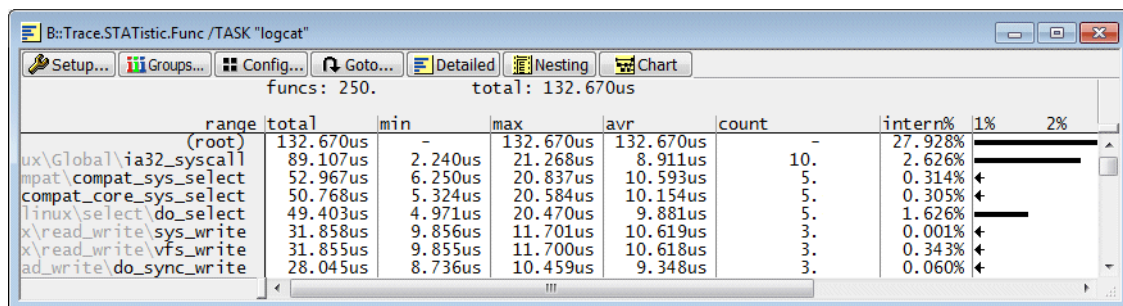
HLL function **hrtimer\_cancel** running in process **@logcat**.

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.

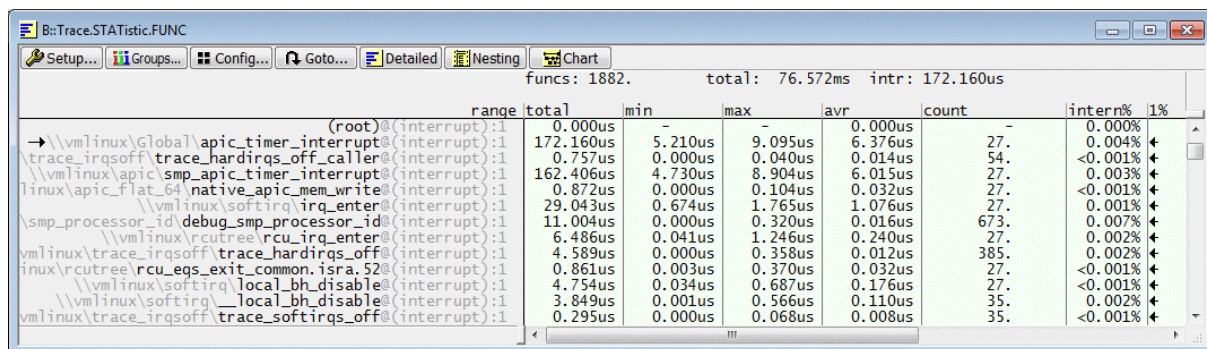
**Trace.STATistic.Func /TASK** <task\_magic> | <task\_name> | <task\_id>

Trace.STATistic.Func /TASK "logcat"



## Interrupt Functions

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



The screenshot shows the B::Trace.STATISTIC.FUNC window with the following data:

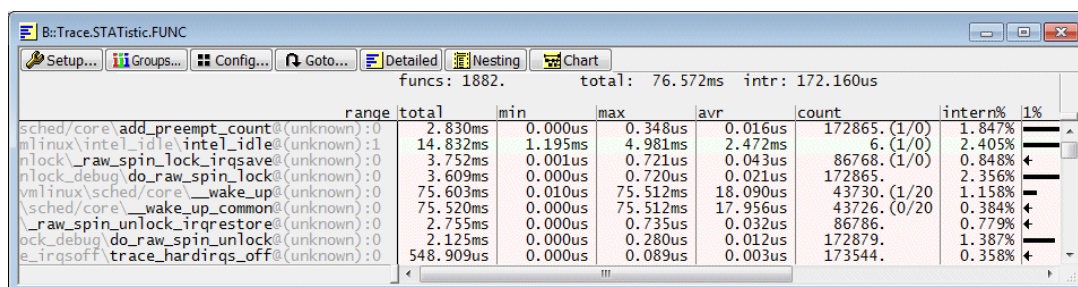
	range	total	min	max	avr	count	intern%	1%
(root)@(interrupt):1		0.000us	-	-	0.000us	-	0.000%	
→\\vmlinux\Global\apic_timer_interrupt@(interrupt):1		172.160us	5.210us	9.095us	6.376us	27.	0.004%	
\\vmlinux\apic\trace_hardirqs_off_caller@(interrupt):1		0.757us	0.000us	0.040us	0.014us	54.	<0.001%	
\\vmlinux\apic\apic_timer_interrupt@(interrupt):1		162.406us	4.730us	8.904us	6.015us	27.	0.003%	
\\vmlinux\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%	
\\vmlinux\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%	
\\vmlinux\rcutree\rcu_eqgs_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

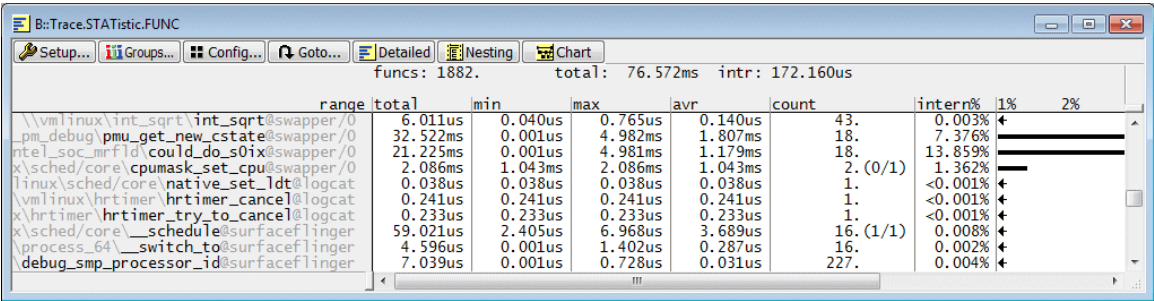
All function recorded before the first process switch is recorded are assigned to the @(unknown) task. Core information is provided for the @(unknown) task.



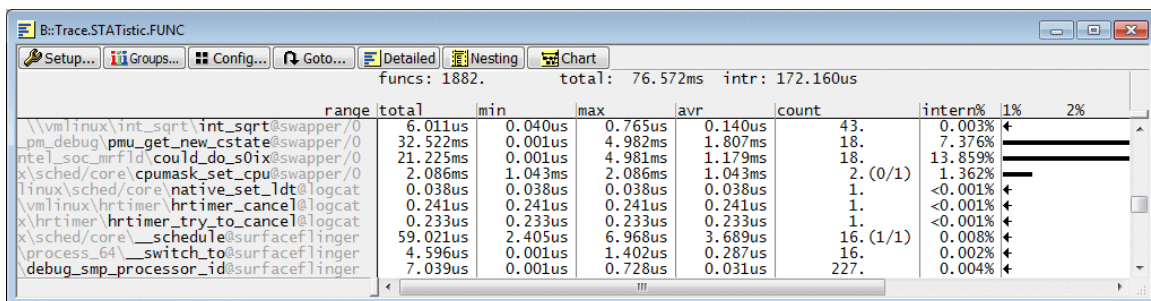
The screenshot shows the B::Trace.STATISTIC.FUNC window with the following data:

	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%	
\\vmlinux\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%	
\\vmlinux\sched\core\__wake_up\_common@(unknown):0		75.520ms	0.000us	75.512ms	17.956us	43726. (0/20)	0.384%	
\\vmlinux\_raw\_spin\_unlock\_irqrestore@(unknown):0		2.755ms	0.000us	0.735us	0.032us	86786.	0.779%	
lock\_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%	

# Default Results



columns (cont.)	
total	total time within the function
min	<p>shortest time between function entry and exit, time spent in interrupt service routines is excluded</p> <p>No <b>min</b> time is displayed if a function exit was never executed.</p>
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:

2. (2/0): 2 times within the function, 2 function entries missing
4. (0/3): 4 times within the function, 3 function exits missing
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)
--	--

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

Setup...Groups...Config...Goto...DetailedNestingChart

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

	range	total	min	max	avr	count	intern%	1%	2%
\\vm\linux\int_sqrt\int_sqrt@swapper/0		6.011us	0.040us	0.765us	0.140us	43.	0.003%		
_pm_debug\pmu_get_new_cstate@swapper/0		32.522ms	0.001us	4.982ms	1.807ms	18.	7.376%		
ntel_soc_mrfld\could_do_s0ix@swapper/0		21.225ms	0.001us	4.981ms	1.179ms	18.	13.859%		
x\sched\core\cpumask_set_cpu@swapper/0		2.086ms	1.043ms	2.086ms	1.043ms	2. (0/1)	1.362%		
linux\sched\core\native_set_ldt@logcat		0.038us	0.038us	0.038us	0.038us	1.	<0.001%		
\\vm\linux\hrtimer\hrtimer_cancel@logcat		0.241us	0.241us	0.241us	0.241us	1.	<0.001%		
x\hrtimer\hrtimer_try_to_cancel@logcat		0.233us	0.233us	0.233us	0.233us	1.	<0.001%		
x\sched\core\__schedule@surfaceflinger									
\process_64\__switch_to@surfaceflinger									
debug_smp_processor_id@surfaceflinger									

Statistic Config

Sort

☒ OFF

☐ Nesting

☐ GROUP

☐ Address

☐ sYmbol

☐ InternalRatio

☐ TotalRatio

☐ Ratio

☐ Count

☐ TotalMAX

☐ RatioMAX

Sort visible

☒ Global

☐ Window

Sort core

☒ CoreTogether

☐ CoreSeparated

available

NAME  
GROUP  
TASK  
TotalRatio  
TotalBAR.LOG  
TotalBAR.LIN  
Internal  
IAVeRage  
IMIN  
IMAX  
InternalBAR.LIN  
External  
EAVeRage  
EMIN  
EMAX

->

-<

selected

Total  
MIN  
MAX  
AVeRage  
Count  
InternalRatio  
InternalBAR.LOG

☐ All Windows

©1989-2024 Lauterbach

Training Intel® Processor Tracing | 112



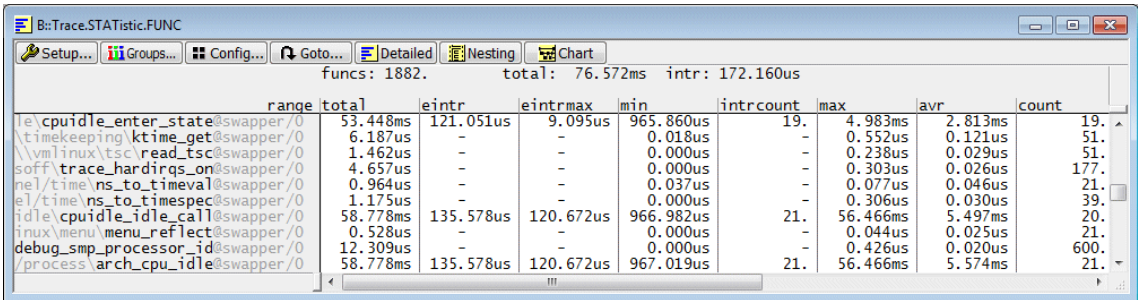
The screenshot shows a window titled "Bt::Trace.STATistic.FUNC" with a menu bar (Setup..., Groups..., Config..., Goto..., Detailed, Nesting, Chart) and a toolbar. The main display area shows a table of function execution statistics. The table has columns: range, total, internal, iavr, imin, imax, and avr. The data is sorted by total time in descending order. The first row is for the function range "\vm\linux\int\_sqrt\int\_sqrt@swapper/0" with a total time of 6.011us. The second row is for the function range "\vm\linux\pmu\_get\_new\_cstate@swapper/0" with a total time of 32.522ms. The third row is for the function range "\vm\linux\ntel\_soc\_mrflid\could\_do\_s0ix@swapper/0" with a total time of 21.225ms. The fourth row is for the function range "\vm\linux\x\sched\core\cpumask\_set\_cpu@swapper/0" with a total time of 2.086ms. The fifth row is for the function range "\vm\linux\linux\sched\core\native\_set\_ldt@logcat" with a total time of 0.038us. The sixth row is for the function range "\vm\linux\vm\linux\hrtimer\hrtimer\_cancel@logcat" with a total time of 0.241us. The seventh row is for the function range "\vm\linux\x\hrtimer\hrtimer\_try\_to\_cancel@logcat" with a total time of 0.233us. The eighth row is for the function range "\vm\linux\x\sched\core\\_\_schedule@surfaceflinger" with a total time of 59.021us. The ninth row is for the function range "\vm\linux\process\_64\\_\_switch\_to@surfaceflinger" with a total time of 4.596us. The tenth row is for the function range "\vm\linux\debug\_smp\_processor\_id@surfaceflinger" with a total time of 7.039us. The table also shows the total time for all functions (1882. total: 76.572ms) and the intr time (172.160us).

range	total	internal	iavr	imin	imax	avr
\vm\linux\int_sqrt\int_sqrt@swapper/0	6.011us	6.011us	0.140us	0.040us	0.765us	0.140us
\vm\linux\pmu_get_new_cstate@swapper/0	32.522ms	11.297ms	627.621us	0.000us	4.982ms	1.807ms
\vm\linux\ntel_soc_mrflid\could_do_s0ix@swapper/0	21.225ms	21.225ms	1.179ms	0.000us	4.981ms	1.179ms
\vm\linux\x\sched\core\cpumask_set_cpu@swapper/0	2.086ms	2.086ms	1.043ms	2.086ms	2.086ms	1.043ms
\vm\linux\linux\sched\core\native_set_ldt@logcat	0.038us	0.038us	0.038us	0.038us	0.038us	0.038us
\vm\linux\vm\linux\hrtimer\hrtimer_cancel@logcat	0.241us	0.008us	0.008us	0.008us	0.008us	0.241us
\vm\linux\x\hrtimer\hrtimer_try_to_cancel@logcat	0.233us	0.042us	0.042us	0.042us	0.042us	0.233us
\vm\linux\x\sched\core\__schedule@surfaceflinger	59.021us	12.326us	0.770us	0.398us	1.907us	3.689us
\vm\linux\process_64\__switch_to@surfaceflinger	4.596us	3.249us	0.203us	0.001us	0.870us	0.287us
\vm\linux\debug_smp_processor_id@surfaceflinger	7.039us	7.039us	0.031us	0.000us	0.728us	0.031us

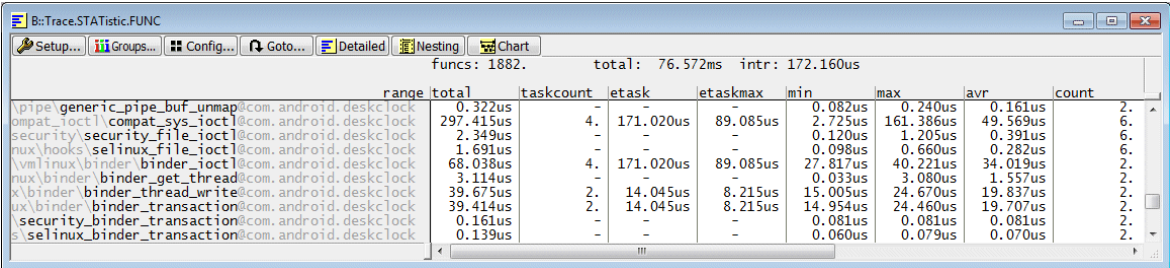
### columns (cont.) - times only in function

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



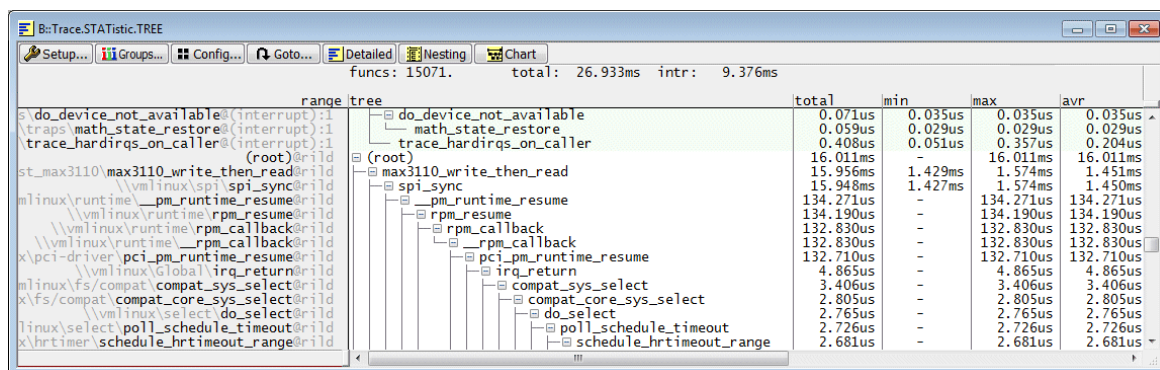
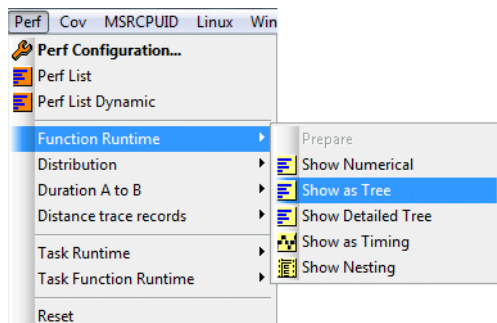
columns (cont.) - interrupt times	
ExternalINTR	total time the function was interrupted
ExternalINTRMAX	max. time one function pass was interrupted
INTRCount	number of interrupts that occurred during the function run-time



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.

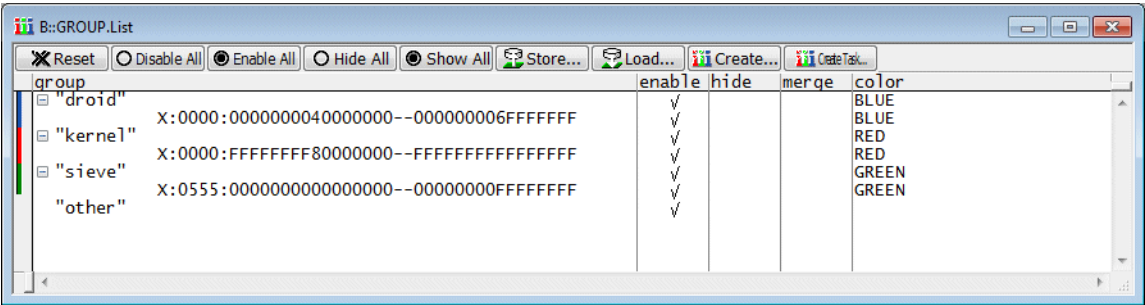
```
Trace.STATistic.TREE /TASK "rild"
```

## 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:

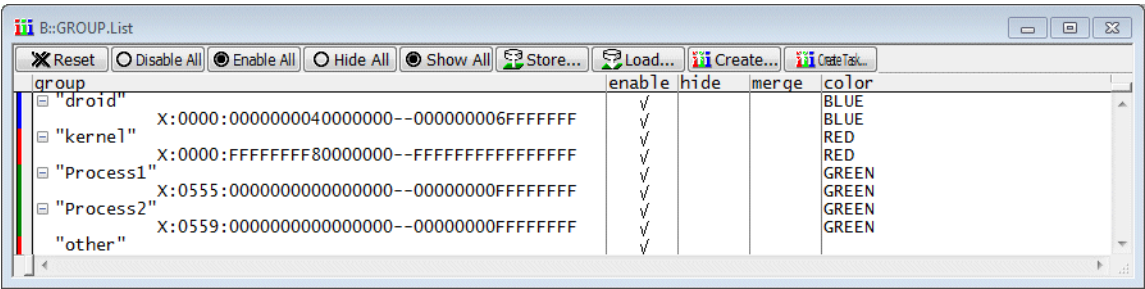
- A GROUP “kernel”, color RED, to mark the OS kernel.
- A GROUP “droid”, color BLUE, to mark virtual machine byte code e.g. Android/Dalvik.
- A GROUP <process\_name> per process, color GREEN.
- A GROUPs <module\_name> per kernel module, color YELLOW.



A group can have the following statuses:

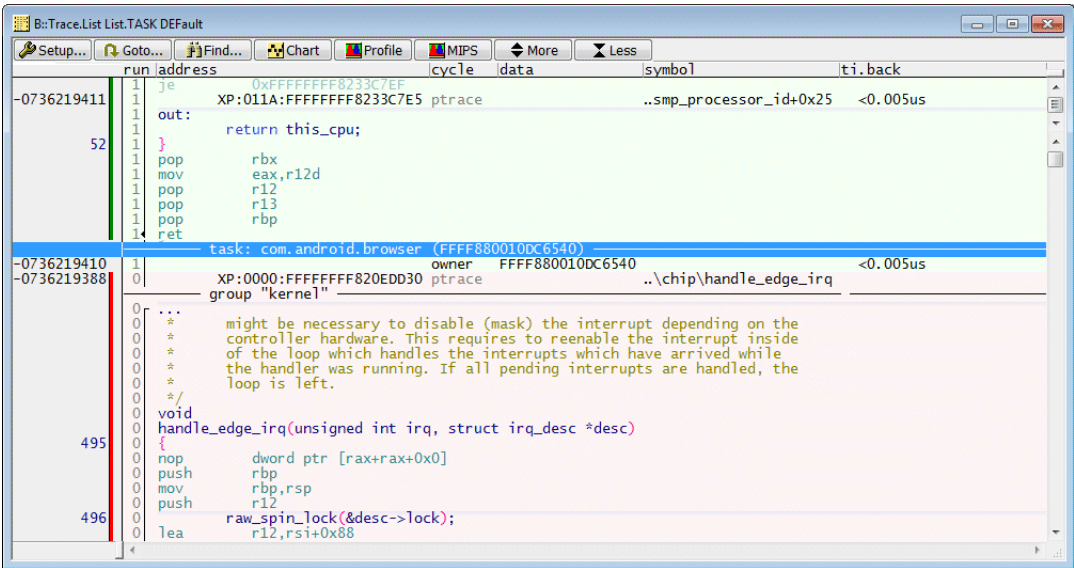
- enable
- enable + merge
- enable + hide

# GROUP Status ENable

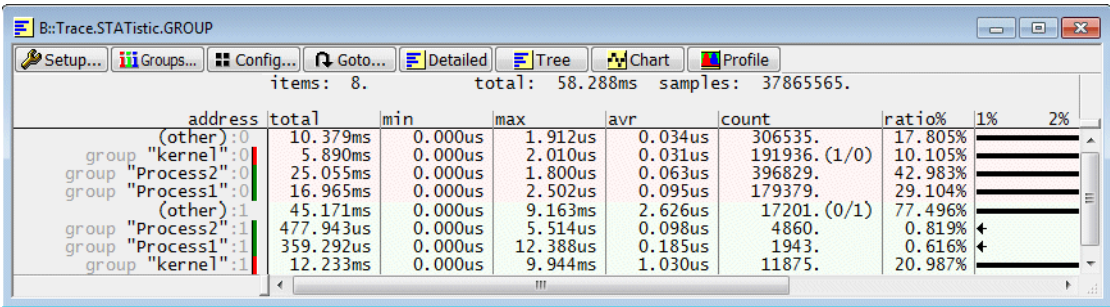


If a GROUP is enabled:

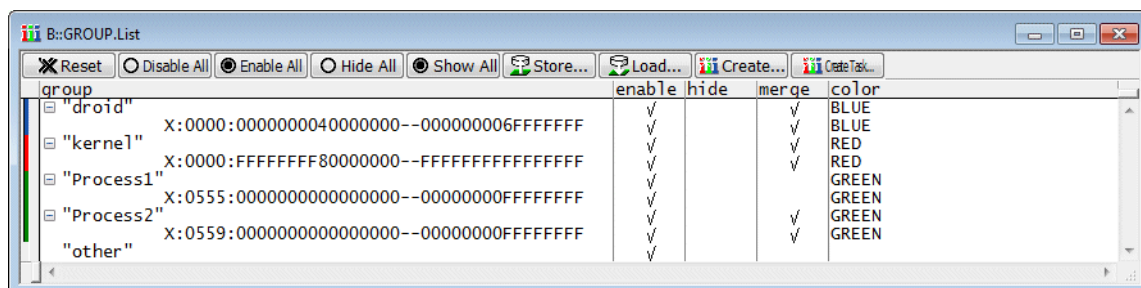
- The trace information recorded for the group members is marked with the color assigned to the group.



- Group-based trace analyses commands are provided e.g. [Trace.STATistic.GROUP](#).

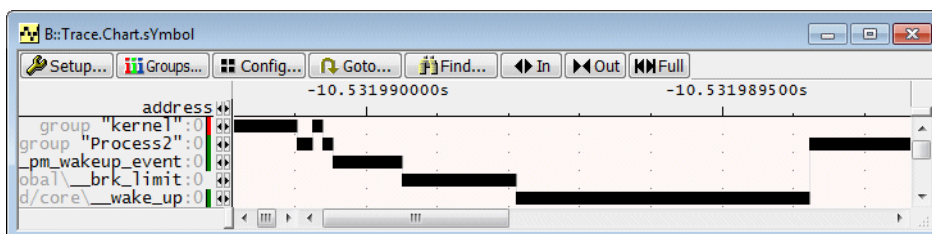


# GROUP Status ENable+Merge

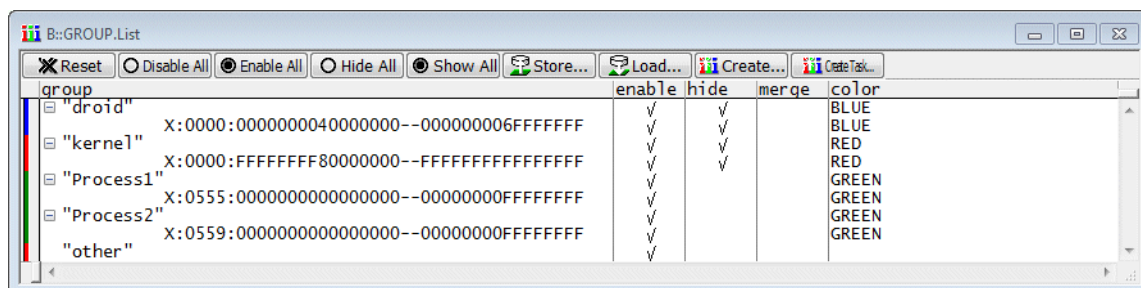


If a GROUP is enabled and merge is checked:

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

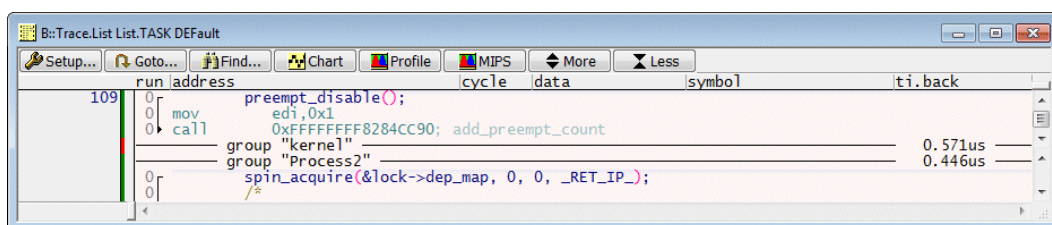


# GROUP Status Enable+HIDE



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.





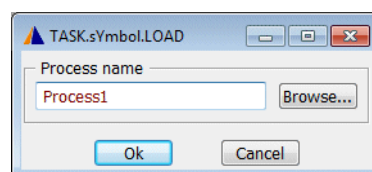
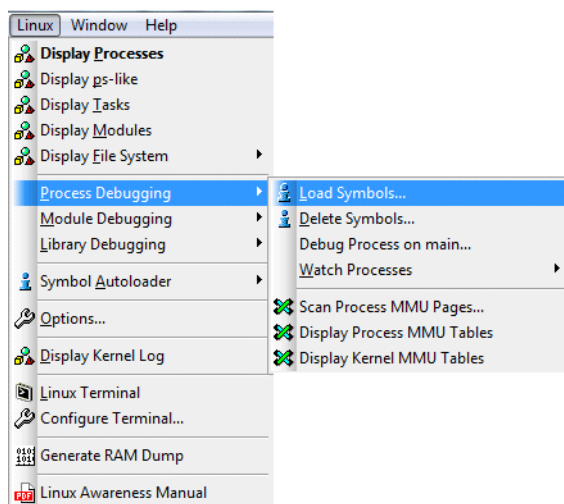
# GROUP Creation

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.