LAUTERBACH
DEVELOPMENT TOOLS

Training AURIX Tracing

MANUAL

Training AURIX Tracing

Training AURIX Tracing

Version 06-Jun-2024

History

31-Aug-22	The MCDS command group and the MCDS.state window have been fully redesigned. Parts of this training are therefore outdated. This applies to the TriCore AURIX TC2x, TC3x and TC4x.
15-Feb-13	Initial version.

Protocol Description

The MCDS (Multi Core Debug Solution) included in the ED device allows to generate trace information.

The following message types are generated:

- **Instruction Pointer Call Messages (ptrace)**

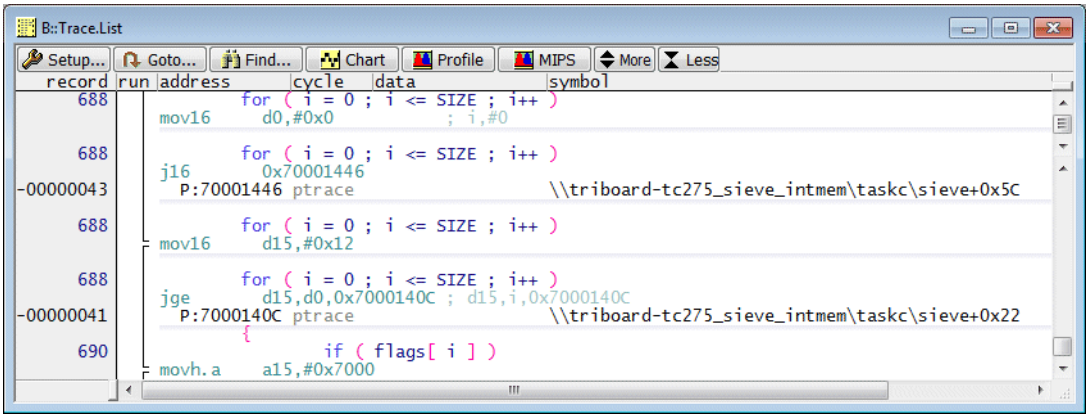
  Instruction Pointer Call Messages are generated for all taken direct and indirect branches, interrupts and traps. They include:

  - take-off address (branch source address)

  - landing address (branch destination address)

  - number of bytes executed since the last Instruction Pointer Call Message

  - source-ID information
  (the AURIX can generate Instruction Pointer Call Messages for up to 2 cores)



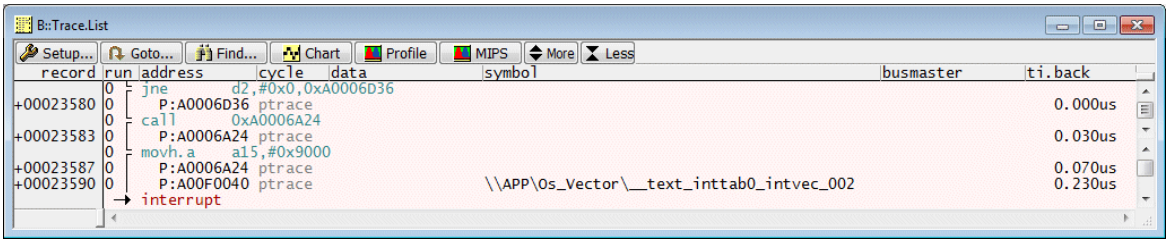
Support for the **Continuous Compact Function Trace** (CFT) is currently under construction.

For some trace analysis features (e.g. nesting function run-time analysis) it might be beneficial to differentiate between taken branches and interrupts/traps.

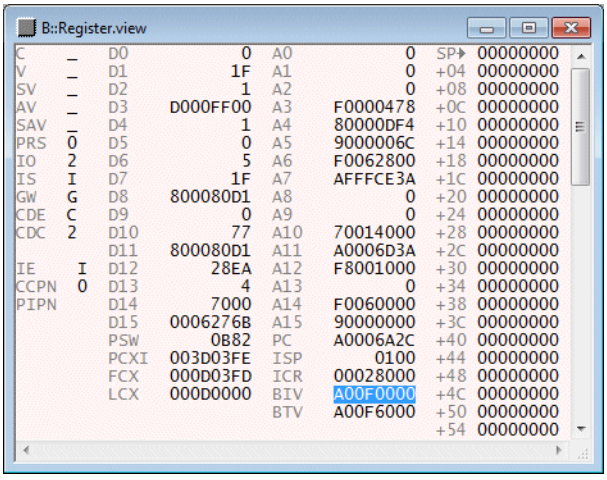
TRACE32 reads the contents of the BIV (Interrupt Vector Table Pointer) register and regards the following address space as interrupt vector table:

- BIV[0]==0: *<contents_of_biv>*++(255. * 32. byte)
- BIV[0]==1: *<contents_of_biv>*++(255. * 16. byte)

If the landing address (branch destination address) of an Instruction Pointer Call Messages is the address of an interrupt vector, TRACE32 indicates an interrupt in the trace recording.



Example



BIV == 0xa00f0000 results in an Interrupt Vector Table within the following address range: 0xa00f0000++0x1fdf

Further options for the exception decoding are described in **“Exception Decoding”** (mcds_user.pdf).

TRACE32 reads the contents of the BTV (Trap Vector Table Pointer) register and regards the following address space as trap vector table:

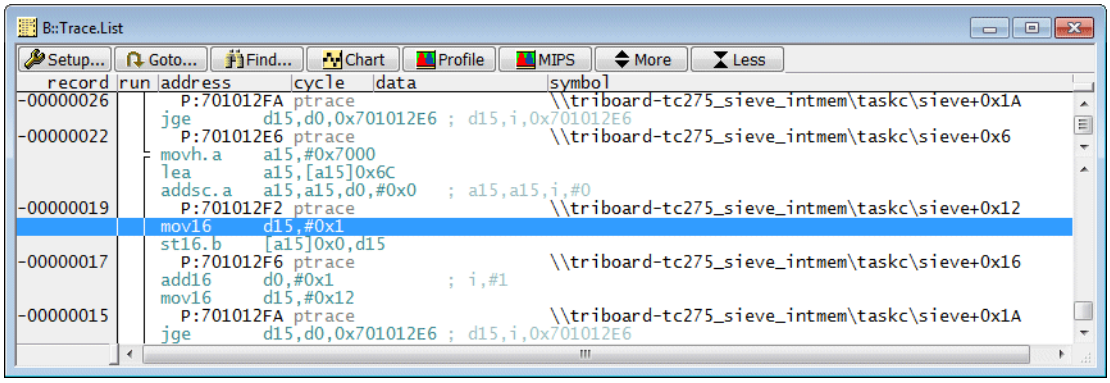
- *<contents_of_btv>*++(8. * 32. Byte)

If the landing address (branch destination address) of an Instruction Pointer Call Messages is a trap vector TRACE32 indicates a trap in the trace recording.

Further options for the trap decoding are described in **"Exception Decoding"** (mcds_user.pdf).

- **Instruction Pointer Messages (ptrace)**

  Instruction Pointer Messages can be generated between the branches if required.



- **Write Data Trace Messages**

  Write Data Trace Messages can be generated for

  - core write accesses

  - transactions on the bus system

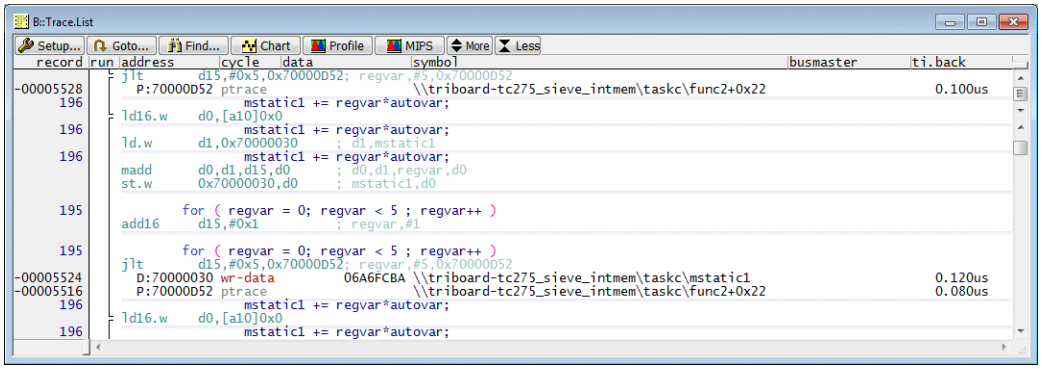
  They include:

  - data write address

  - data write value

  - source-ID information

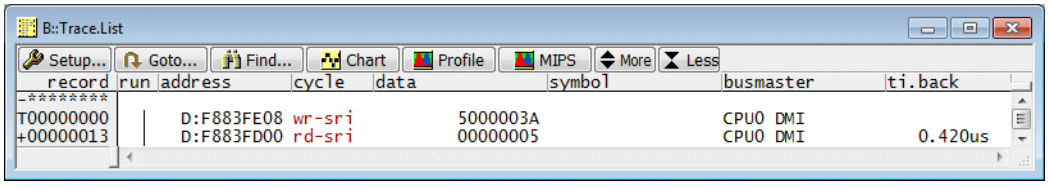
  Source-ID information: Write Data Trace Messages can be generated by up to 2 cores, by the SPB (System Peripheral Bus) and by the SRI (Shared Resource Interconnect)



Write Data Trace Message generated by a core write access

Currently write cycles can not be assigned to its executing instruction. This is why **wr-data** is printed in red.

```
; BusMaster indicates the source of the bus transaction
Trace.List DEFault BusMaster
```



Write Data Trace Message generated by SRI transfer

- **Read Data Trace Messages**

  Read Data Trace Messages can be generated for

  - core read accesses

  - transactions on the bus system

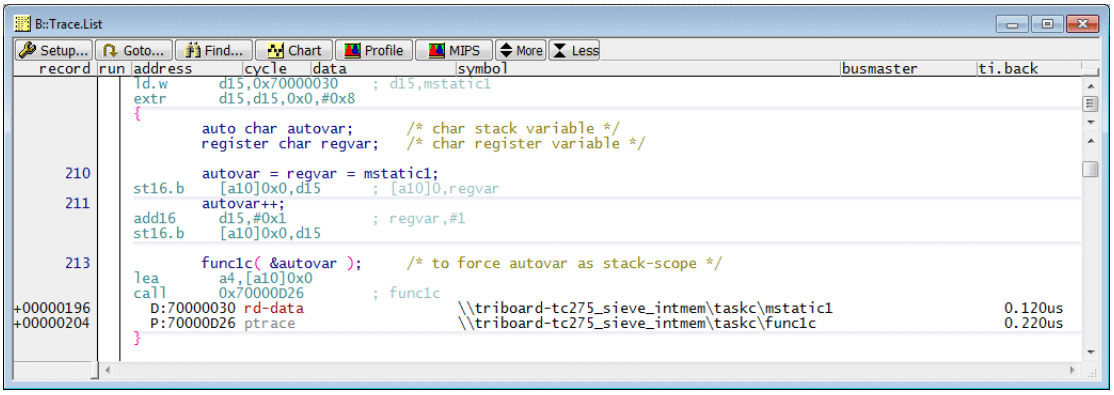
  They include:

  - data read address

  - data read value (transactions on the bus system only)

  - source-ID information

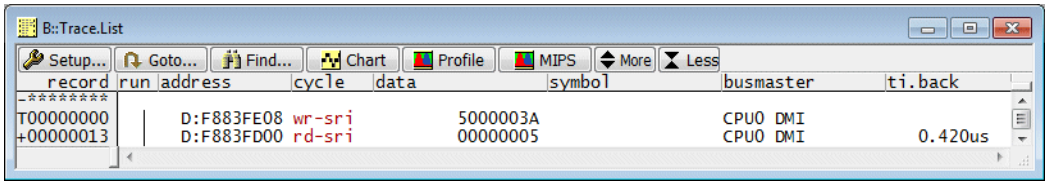
  Source-ID information: Read Data Trace Messages can be generated by up to 2 cores, by the SPB (System Peripheral Bus) and by the SRI (Shared Resource Interconnect)



Read Data Trace Message generated by a core read access (data read address only)

Currently read cycles can not be assigned to its executing instruction. This is why **rd-data** is printed in red.

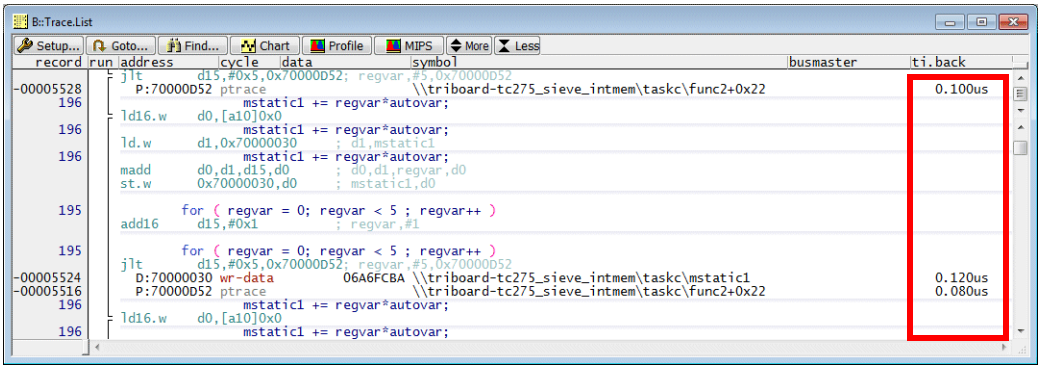
```
; BusMaster indicates the source of the bus transaction
Trace.List DEFault BusMaster
```



Read Data Trace Message generated by SRI transfer (data read address and data read value)

- **Timestamp Messages**

  The MCDS trace infrastructure generates trace messages that provide the number of MCDS clocks needed by a set of instructions. If the MCDS clock is known, TRACE32 calculates time out of this information.



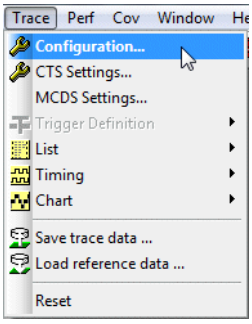
Timestamp calculated out of the ticks (number of MCDS clocks) needed by a set of instructions

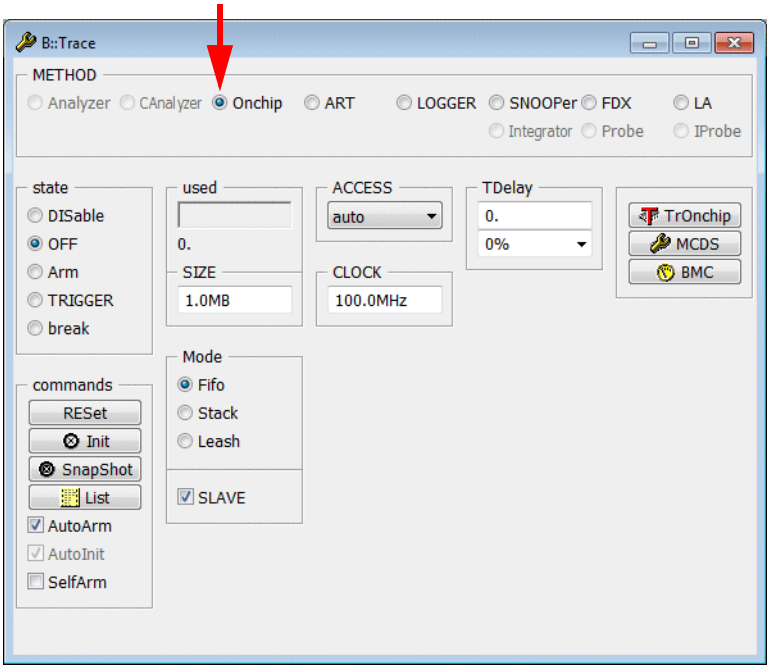
Onchip Trace Buffer (EMEM)

If TRACE32 is started

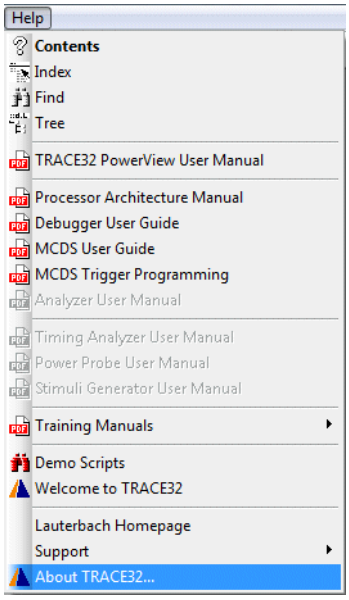
- when a TriCore debugger is connected
- and if the debug communication to an AURIX ED device is established

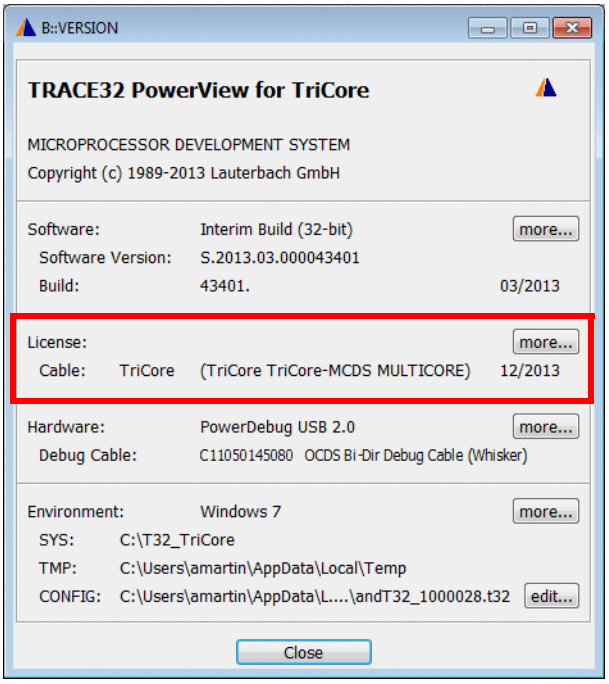
the source for the trace information is the so-called **Onchip** trace (**Trace.METHOD Onchip**).





Please be aware that tracing with the **Onchip** trace requires an **TriCore-MCDS** license in the debug cable. This can be checked as follows:



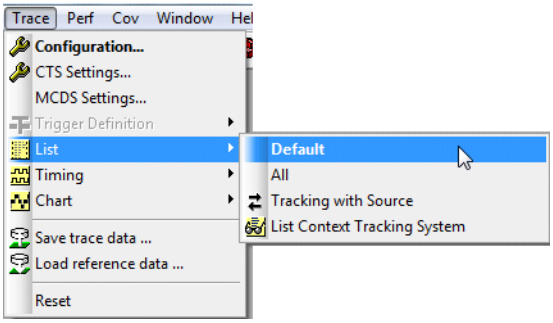


The setting **Trace.METHOD Onchip** has the following impacts:

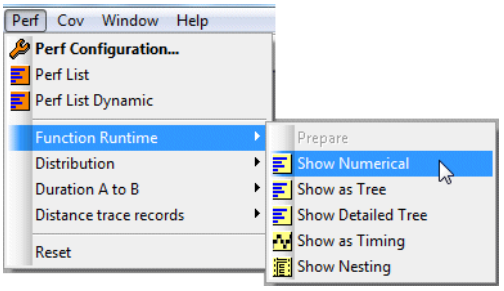
1. **Trace** is an alias for **Onchip**.

```
Trace.List                              ; Trace.List means
                                        ; Onchip.List

Trace.Mode Fifo                         ; Trace.Mode Fifo means
                                        ; Onchip.Mode Fifo
```

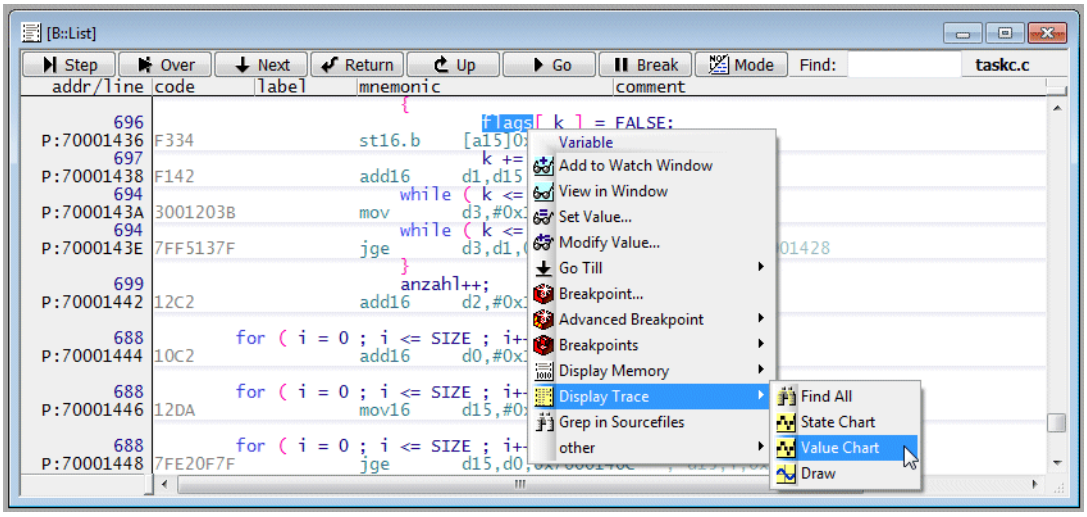
2. All commands from the **Trace** menu apply to the trace information stored in the Onchip trace buffer.



3. All Trace commands from the **Perf** menu apply to the trace information stored in the Onchip trace buffer.



4. **Display Trace** in the **Variable** pull-down applies to the trace information stored in the Onchip trace buffer.



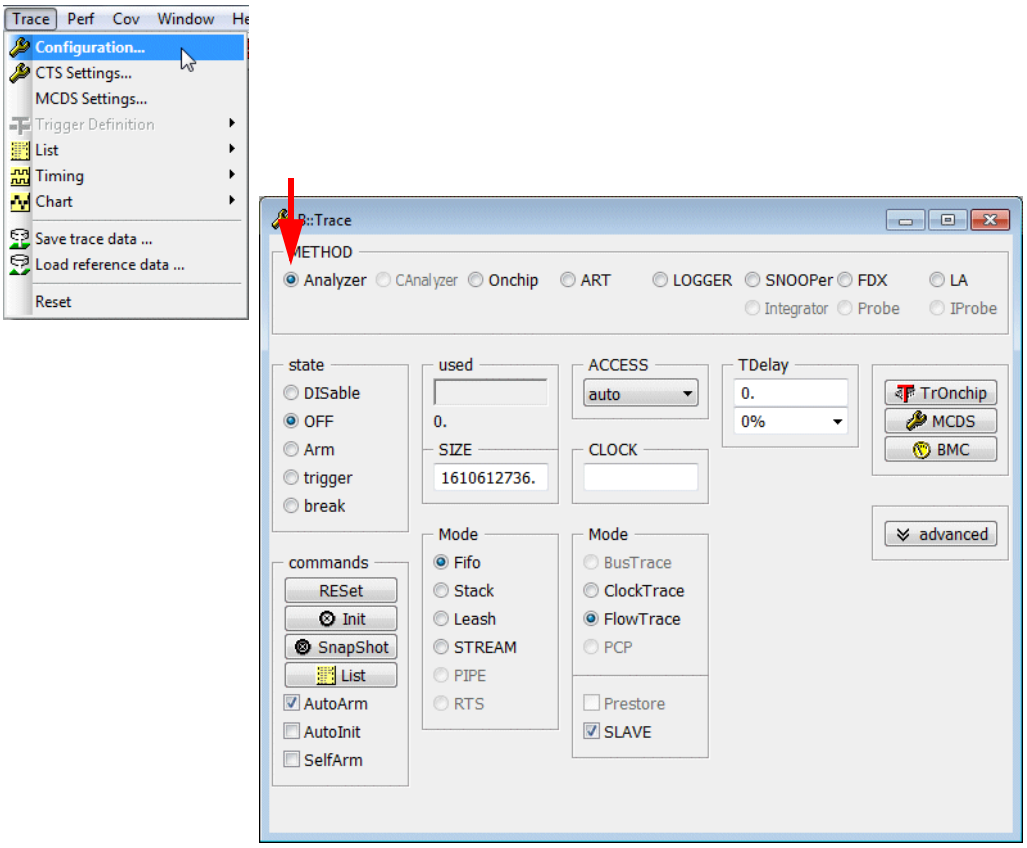
5. TRACE32 is advised to use the trace information stored to the Onchip trace buffer as source for the trace evaluations for the following command groups:

COVerage.<*sub_cmd*>	Trace-based code coverage
ISTAT.<*sub_cmd*>	Detailed instruction analysis
MIPS.<*sub_cmd*>	MIPS analysis
BMC.<*sub_cmd*>	Synthesize instruction flow with recorded benchmark counter information

If TRACE32 is started

- when a PowerTrace hardware and a PREPROCESSOR SERIAL is connected

the source for the trace information is the so-called **Analyzer** (Trace.METHOD Analyzer).

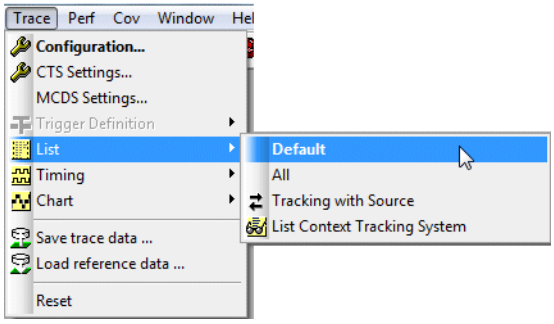


The setting **Trace.METHOD Analyzer** has the following impacts:

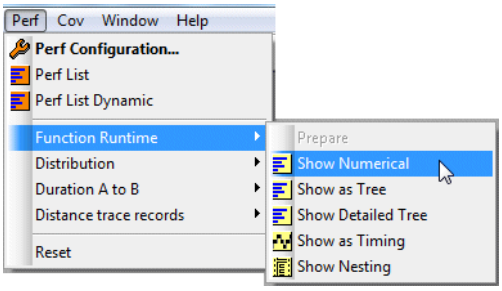
1. **Trace** is an alias for **Analyzer**.

```
Trace.List                              ; Trace.List means
                                        ; Analyzer.List

Trace.Mode Fifo                         ; Trace.Mode Fifo means
                                        ; Analyzer.Mode Fifo
```

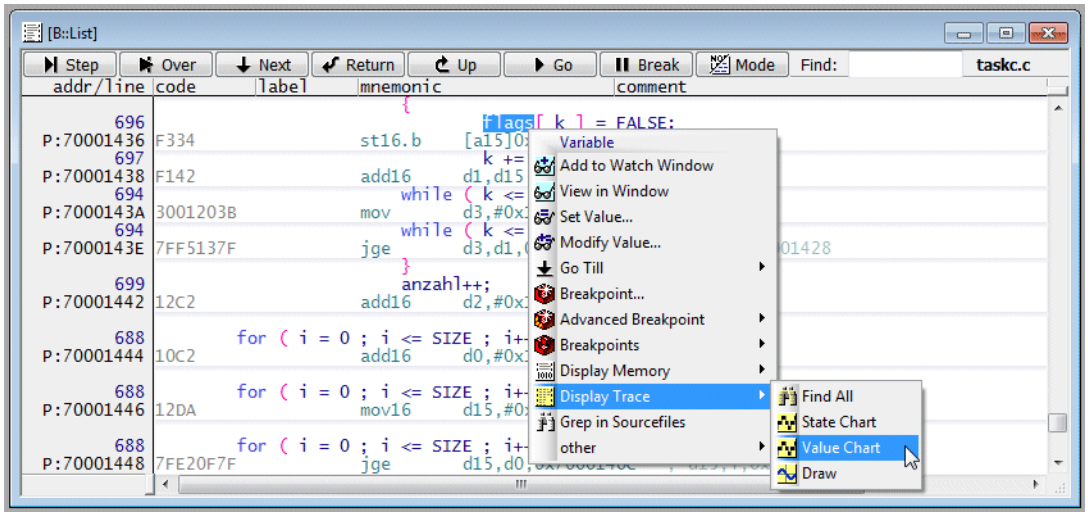
2. All commands from the **Trace** menu apply to the trace information stored in the trace memory of the PowerTrace hardware (Analyzer).



3. All **Trace** commands from the **Perf** menu apply to the trace information stored in the trace memory of the PowerTrace hardware (Analyzer).



4. **Display Trace** in the **Variable** pull-down applies to the trace information stored in the trace memory of the PowerTrace hardware (Analyzer).



5. TRACE32 is advised to use the trace information recorded to the trace memory of the PowerTrace hardware (Analyzer) as source for the trace evaluations for the following command groups:

COVerage.*<sub_cm>*	Trace-based code coverage
ISTAT.*<sub_cmd>*	Detailed instruction analysis
MIPS.*<sub_cmd>*	MIPS analysis
BMC.*<sub_cmd>*	Synthesize instruction flow with recorded benchmark counter information

Onchip Trace Configuration

Steps

Using the onchip trace might require the following setup:

1. Specify the size of the onchip trace buffer.
2. Enable Timestamp Messages (chip timestamp).

1. Specify the size of the onchip trace buffer

TRACE32 will use the complete available emulation memory (EMEM) provided by the ED device as onchip trace buffer if not specified otherwise.

If you want to use part of the emulation memory for other purposes e.g. calibration this has to be configured before the communication between the debugger and the core(s) is established by **SYStem.Up**.

The following commands are available to specify the size of the onchip trace buffer.

MCDS.TraceBuffer SIZE *<size>*	Specify the trace buffer *<size>*.
MCDS.TraceBuffer UpperGAP *<size>*	Specify *<size>* of upper gap in EMEM that can not be used as onchip trace buffer.
MCDS.TraceBuffer LowerGAP *<size>*	Specify *<size>* of lower gap in EMEM that can not be used as onchip trace buffer.

If a third-party tool or the application has already allocated its part of the EMEM, the following command can detect which part of the EMEM can be used as trace buffer. Please be aware that the result of **MCDS.TraceBuffer DETECT** always requires a sanity check.

MCDS.TraceBuffer DETECT	Auto-detect EMEM configuration

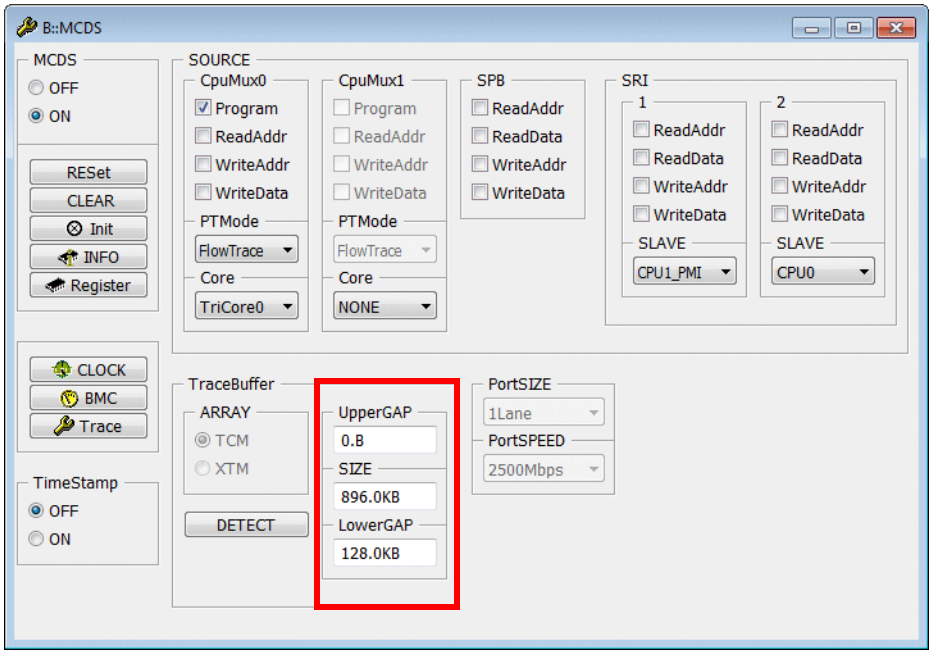
The following 5 uses cases will show you how the size of the onchip trace buffer is configured.

Use case 1: First calibration memory then trace buffer

E.g. tile 0 and 1 are used by the calibration tool, the rest of the EMEM can be used as onchip trace buffer.

1024 KByte EMEM
64 KByte per tile

```
MCDS.TraceBuffer SIZE 896.KB          ; size of trace buffer
                                      ; 1024 KByte - (2 * 64 KByte)

                                      ; the lower gap is automatically
                                      ; calculated by TRACE32
```

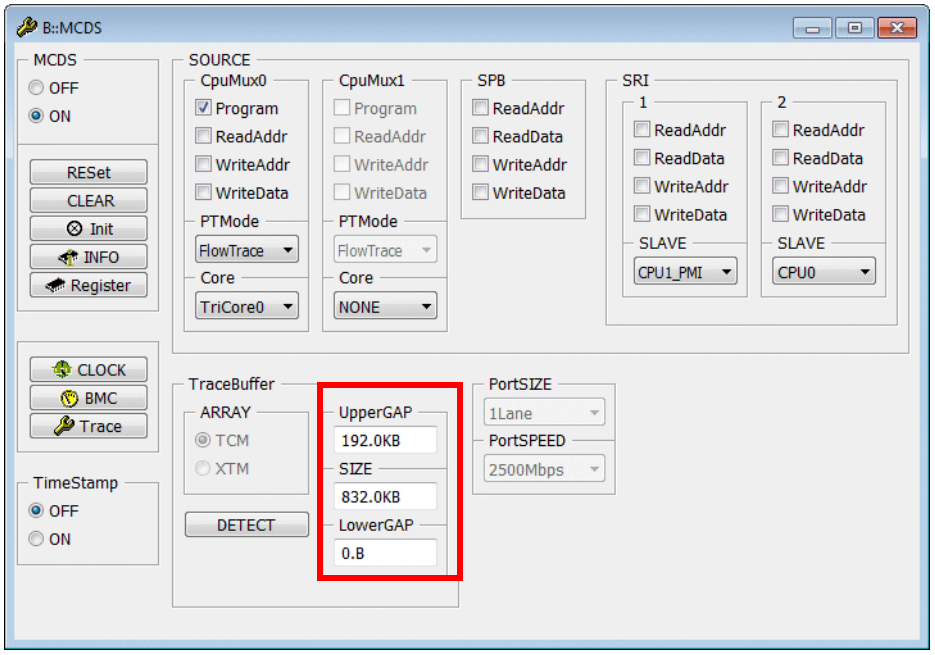


Use case 2: First trace buffer then calibration memory

E.g. tile 13 to 15 are used by the calibration tool, the rest of the EMEM can be used as onchip trace buffer.

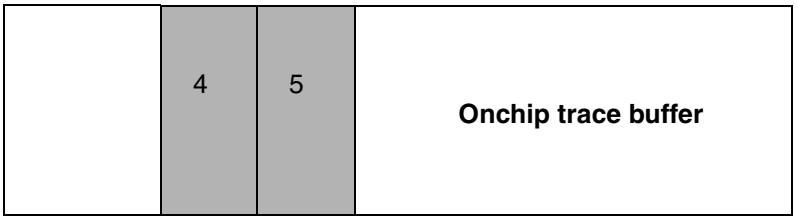
1024 KByte EMEM
64 KByte per tile

```
MCDS.TraceBuffer UpperGAP 192.KB      ; size of upper gap
                                      ; 3 * 64 KByte

                                      ; the size of the onchip trace
                                      ; buffer is automatically
                                      ; calculated by TRACE32
```



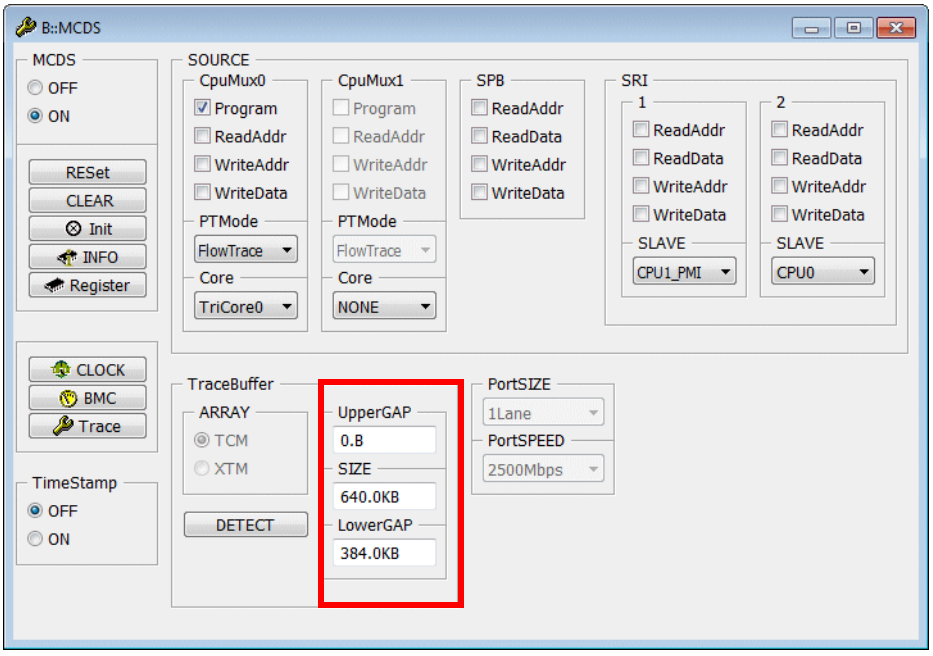
Use case 3: Calibration memory in the middle of EMEM, trace buffer after the calibration memory

E.g. tile 4 and 5 are used by the calibration tool, the rest of the EMEM can be used as onchip trace buffer. Since the trace buffer has to be mapped continuously tile 0-3 can not be used.



1024 KByte EMEM
64 KByte per tile

```
MCDS.TraceBuffer SIZE 640.KB          ; size of trace buffer
                                      ; 1024 KByte - (6 * 64 KByte)

                                      ; the lower gap is automatically
                                      ; calculated by TRACE32
```

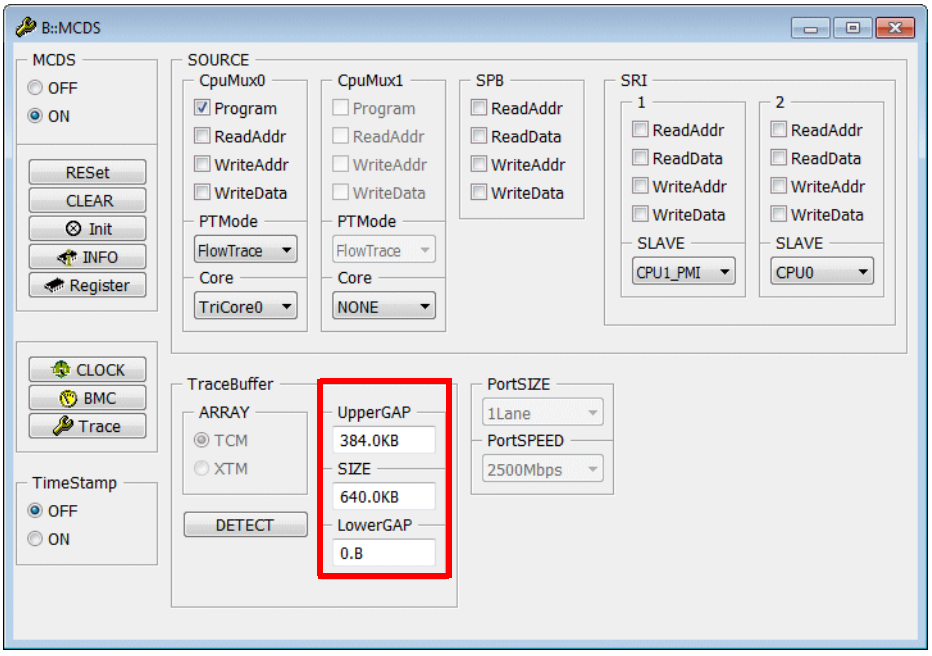


Use case 4: Calibration memory in the middle of EMEM, trace buffer before the calibration memory

E.g. tile 10 and 11 are used by the calibration tool, the rest of the EMEM can be used as onchip trace buffer. Since the trace buffer has to be mapped continuously tile 12 to 15 can not be used.

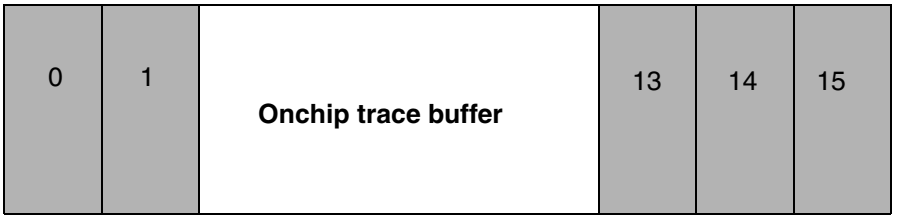
1024 KByte EMEM
64 KByte per tile

```
MCDS.TraceBuffer UpperGAP 384.KB      ; size of upper gap
                                      ; 6 * 64 KByte

                                      ; the size of the onchip trace
                                      ; buffer is automatically
                                      ; calculated by TRACE32
```



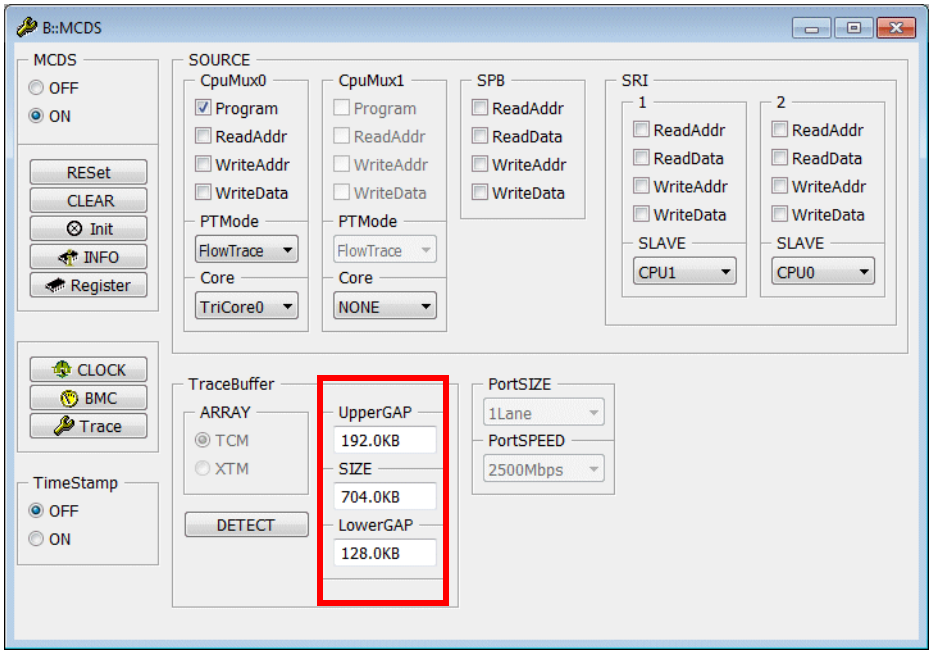
Use case 5: First calibration memory then trace buffer then application memory

E.g. tile 0 and 1 are used by the calibration tool, tile 13 to 15 are used by the application. The rest of the EMEM can be used as onchip trace buffer.



1024 KByte EMEM
64 KByte per tile

```
MCDS.TraceBuffer SIZE 704.KB          ; size of trace buffer
                                      ; 1024 KByte - (5 * 64 KByte)

MCDS.TraceBuffer LowerGAP 128.KB      ; size of the lower gap
                                      ; 2 * 64 KByte
```



The Timestamp Messages are disabled by default.

Enabling the time information requires the following steps:

2.1. Inform TRACE32 about the system clocks (mainly oscillator clock frequency) and enable all derived calculations. One of the derived calculations is **f(mcds)**. Since Timestamp Messages provide the number of MCDS clocks needed by a set of executed instructions 1/f(mcds) can be used to calculate time information.

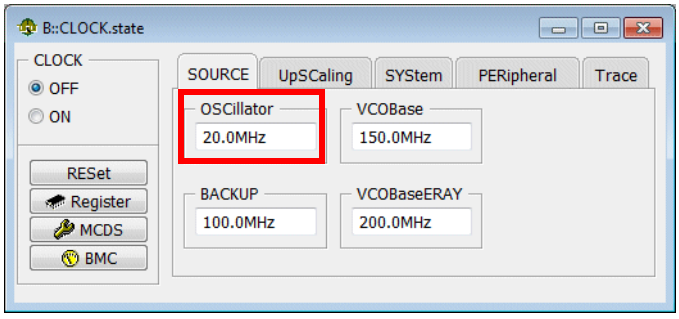
2.2. Enable Timestamp Messages.

NOTE:	Please be aware that there a constraints for the CPU clock, the Backbone Bus clock and the MCDS clock. For details refer to your AURIX manual.

A more complex configuration is required if you use a free-running clock.

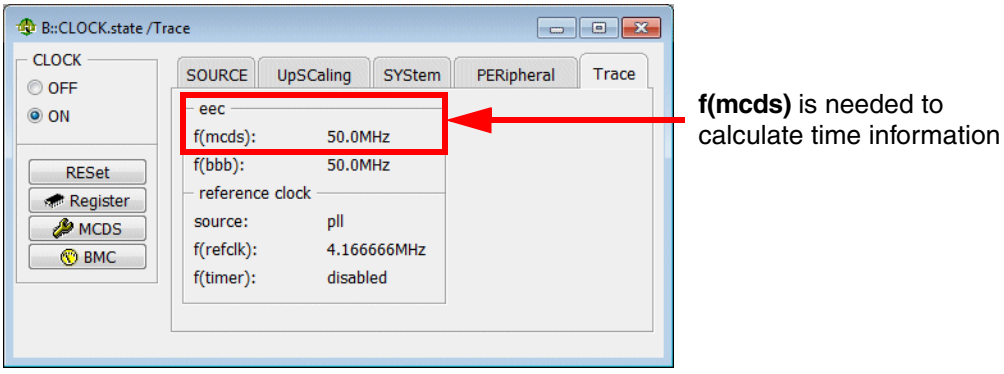
2.1. Inform TRACE32 about the system clocks (mainly oscillator clock frequency) and enable all derived calculations.

CLOCK.state	Display the system clocks configuration/calculation window.



CLOCK.OSCillator *<frequency>*	Specify your board oscillator frequency, if it differs from the default setting.

CLOCK.ON	Declare system clocks as valid.



f(mcds) is needed to calculate time information

NOTE:	The system clock and its derived calculations are valid for all cores in a chip. The TRACE32 Resource Management ensures that the settings are consistent between the different TRACE32 instances of an AMP system (joint settings).

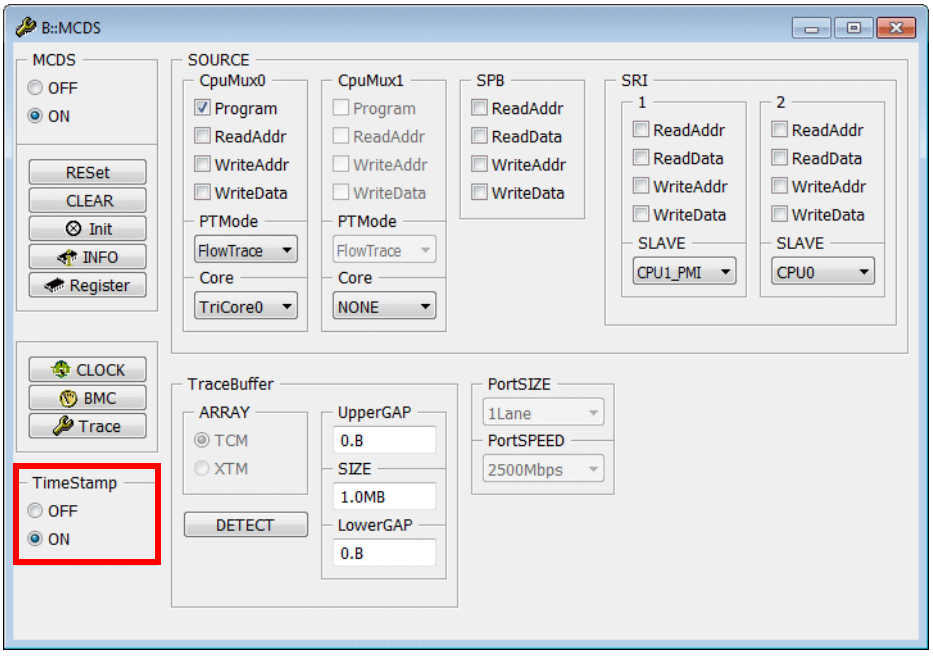
Please be aware that calculating the time out of the MCDS clock fails, if the pll change during the trace recording.

Please be aware that f(mcds) and f(cpux) can be set via TRACE32 commands, e.g. for the post mortem trace analysis (PowerTrace only).

MCDS.CLOCK.Frequency.McdsClock *<frequency>*	Specify f(mcds).
Trace.CLOCK *<freq>*	Specify f(cpu).
Trace.CLOCK *<freq0> <freq1> …* (SMP tracing only)	Specify f(cpu0), f(cpu1), f(cpu2).

2.1. Enable Timestamp Messages.

MCDS.TimeStamp ON	Ticks (number of MCDS clock cycles) are enabled.



Examples

Example 1

```
                                        ; use complete emulation memory
                                        ; (EMEM) provided by ED device as
                                        ; onchip trace buffer

SYStem.Up

;…

CLOCK.ON                                ; declare settings of system clocks
                                        ; as valid
                                        ; your board oscillator frequency
                                        ; is 20.MHz

                                        ; this setting provides f(mcds) to
                                        ; TRACE32

MCDS.TimeStamp ON                       ; enable Timestamp Messages
                                        ; (ticks)
```

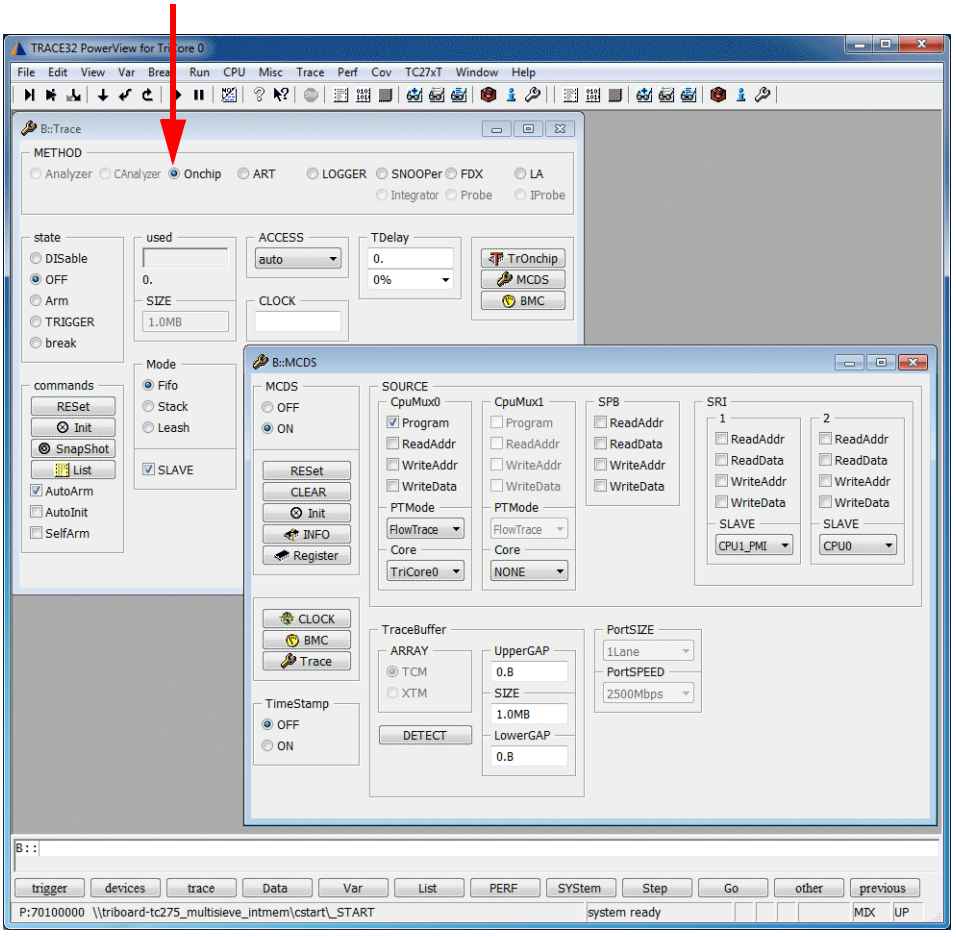
Example 2

```
                                        ; use complete emulation memory
                                        ; (EMEM) provided by ED device as
                                        ; onchip trace buffer

SYStem.Up

;…

CLOCK.state                             ; display system clocks
                                        ; configuration and calculation
                                        ; window

CLOCK.OSCillator 30.MHz                 ; your board oscillator frequency
                                        ; is 30.MHz

CLOCK.ON                                ; declare settings of system clocks
                                        ; as valid

                                        ; this setting provides f(mcds) to
                                        ; TRACE32

MCDS.TimeStamp ON                       ; enable Timestamp Messages
                                        ; (ticks)
```

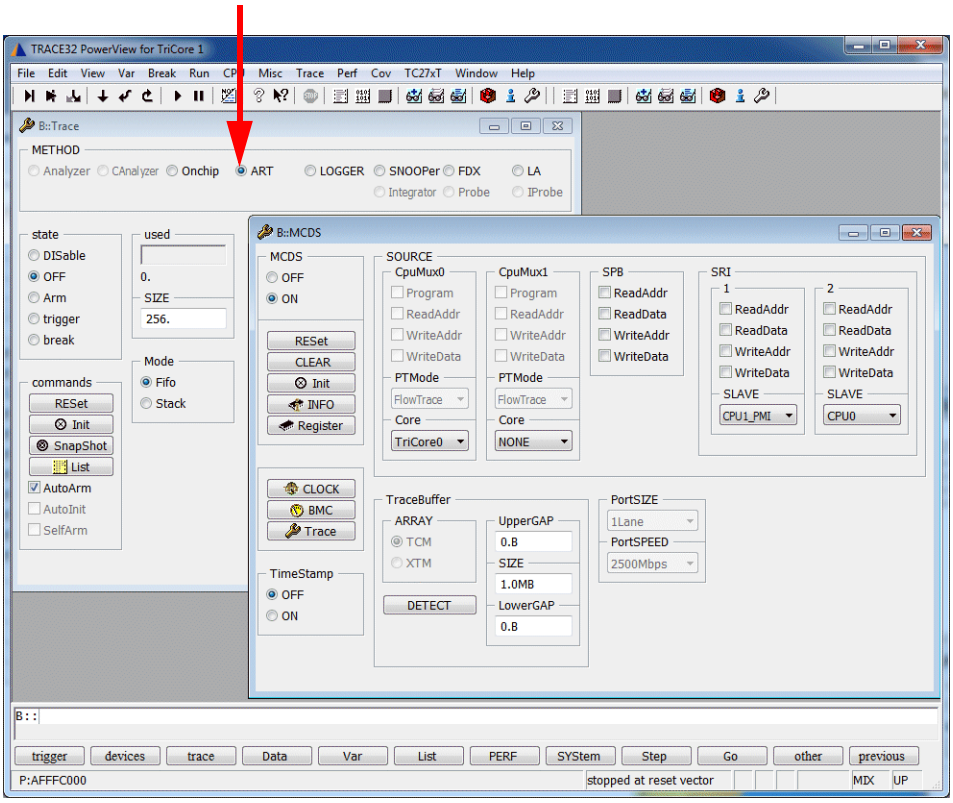
Example 3

```
MCDS.TraceBuffer SIZE 640.KB          ; size of trace buffer
                                      ; 1024 KByte - (6 * 64 KByte)

;…

SYStem.Up

;…

CLOCK.ON                              ; declare settings of system clocks
                                      ; as valid
                                      ; your board oscillator frequency
                                      ; is 20.MHz

                                      ; this setting provides f(mcds) to
                                      ; TRACE32

MCDS.TimeStamp ON                     ; enable Timestamp Messages
                                      ; (ticks)
```

Please be aware that **Trace.METHOD Onchip** is only set for the first TRACE32 instance.



For all other TRACE32 instances **Trace METHOD ART** (Advanced Register Trace) is selected.

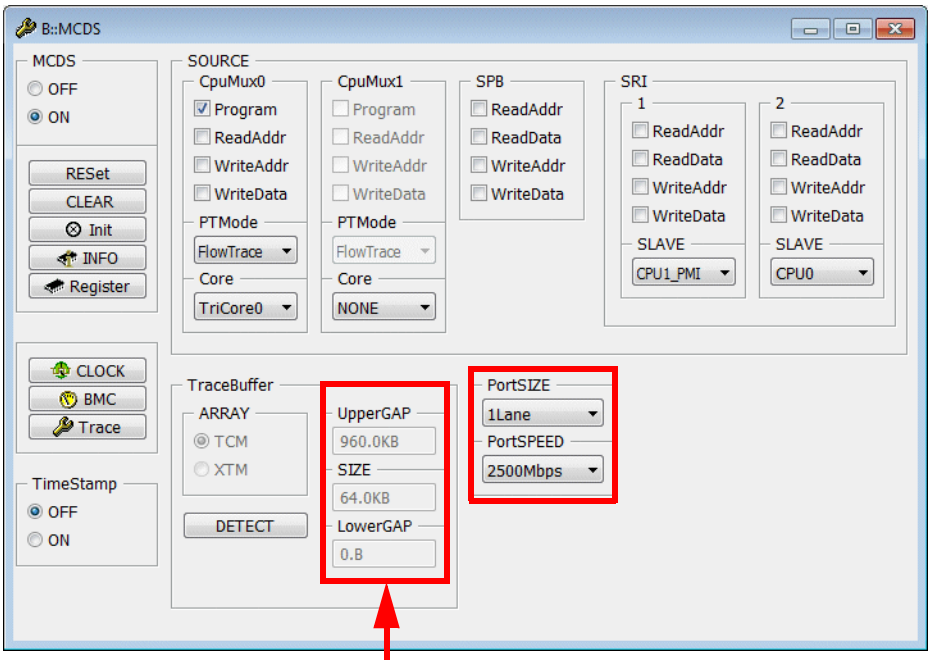


It is recommended to set **Trace.METHOD Onchip** for all other TRACE32 instances:

```
Trace.METHOD Onchip                          ; select Trace.METHOD Onchip
```

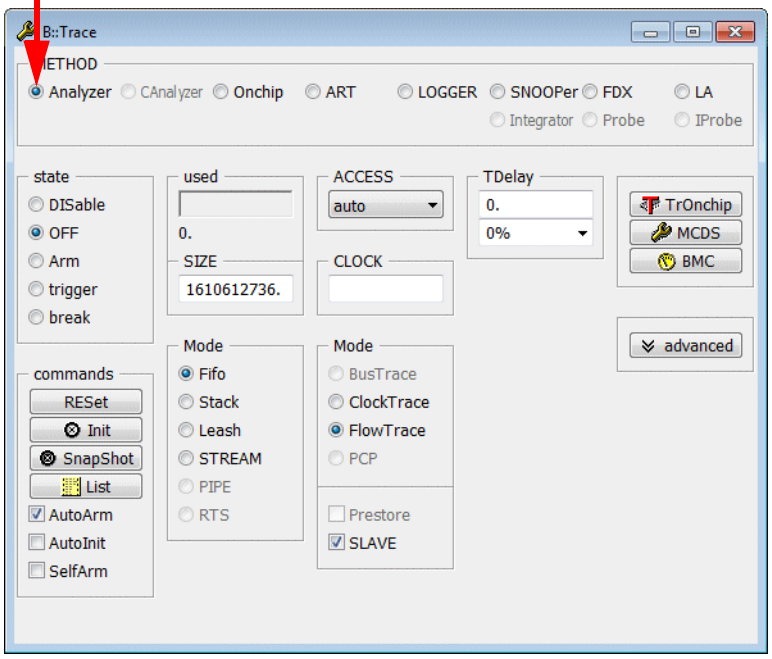
Auto-Configuration

When the communication between the debugger and the core(s) is established by **SYStem.Up** channel training is performed for the Aurora GigaBit Trace.

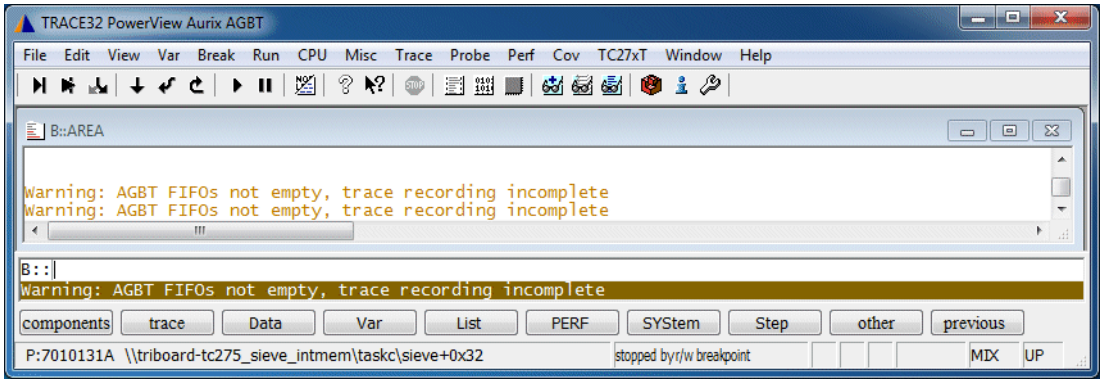


Tile 0 of the EMEM is used as internal buffer (FIFO) for the Aurora GigaBit Trace

Analyzer (off-chip tracing) is automatically selected by TRACE32



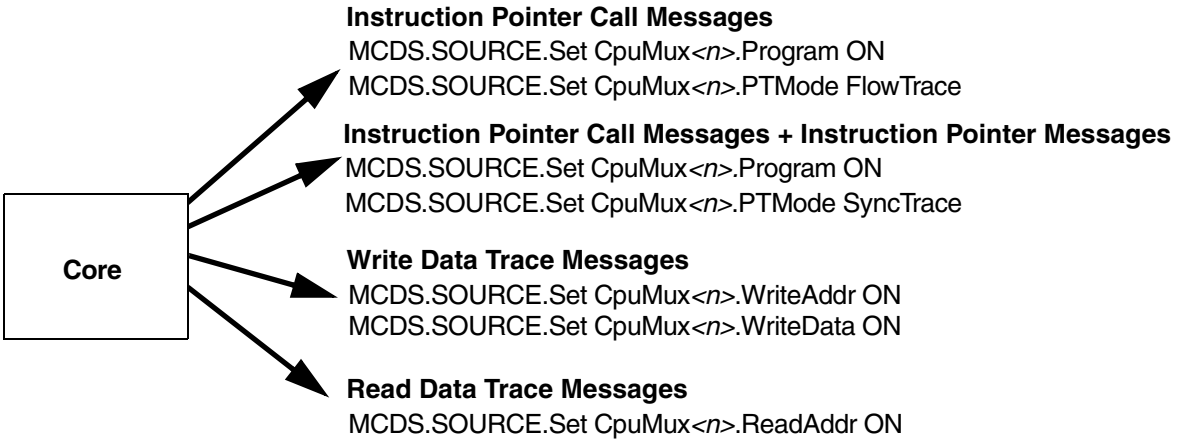
When the program execution is stopped, it may happen that the trace information is not completely flushed. TRACE32 can detect that, but it can not trigger flushing. This is a known AGBT bug.



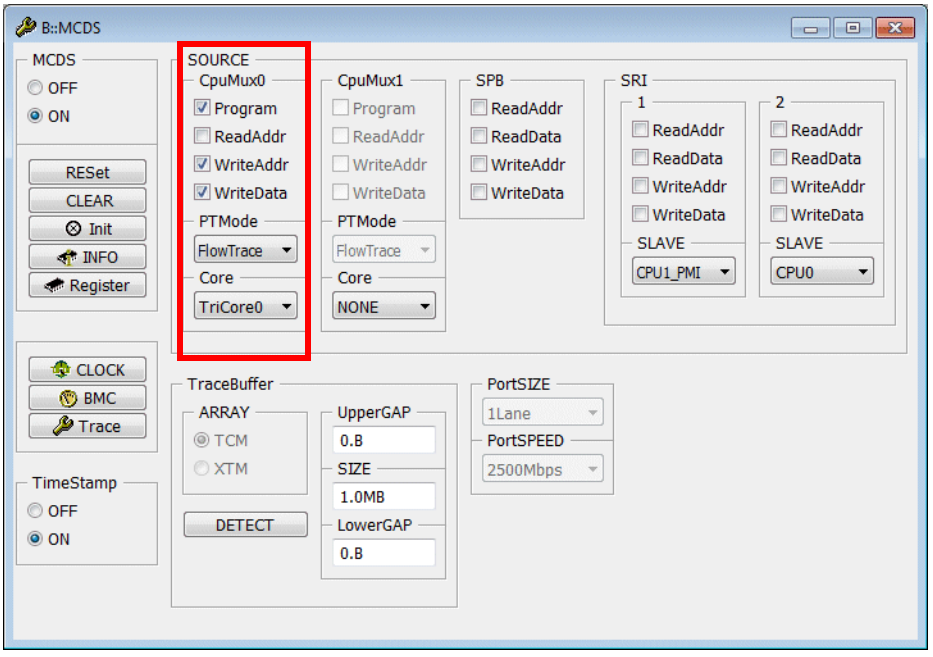
Trace information on the last executed instructions might be lost.

Cores as Trace Source

A core can generate the following trace messages

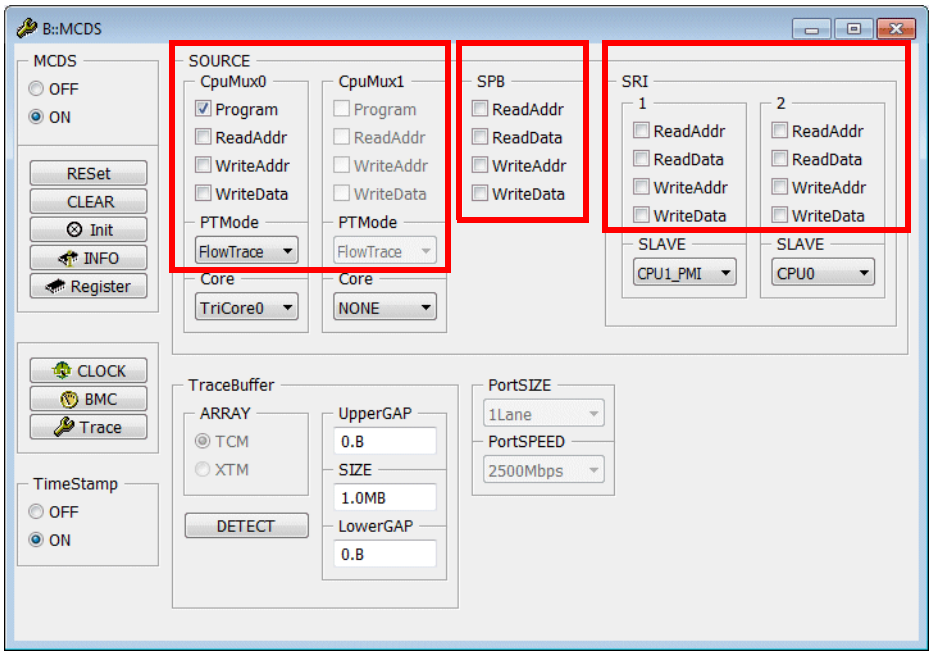



```
MCDS.state                                ; display MCDS configuration
                                          ; window
```

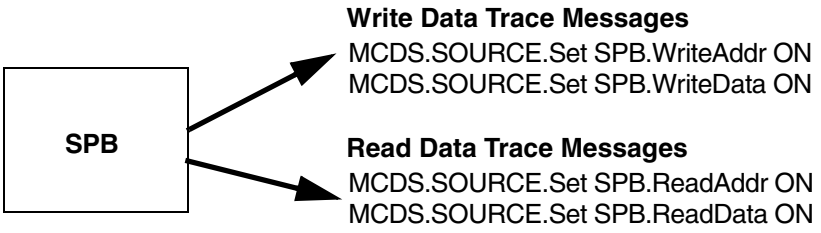


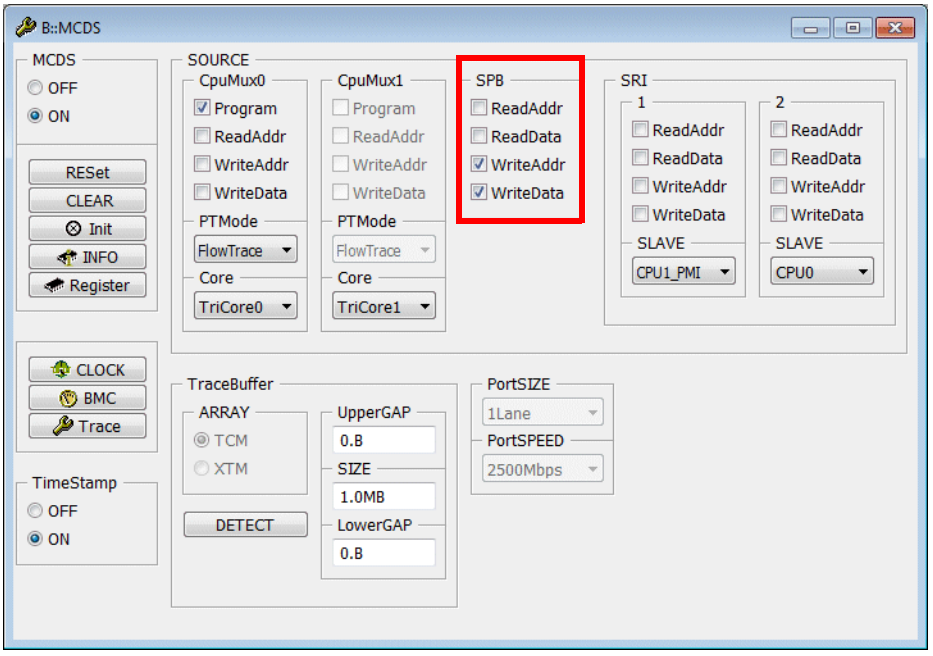
```
MCDS.SOURCE.DEFault                    ; reset the message configuration
                                       ; to its default setting
```

The command **MCDS.SOURCE.DEFault** applies to the settings marked in the picture below:



The following trace messages are generated for the System Peripheral Bus

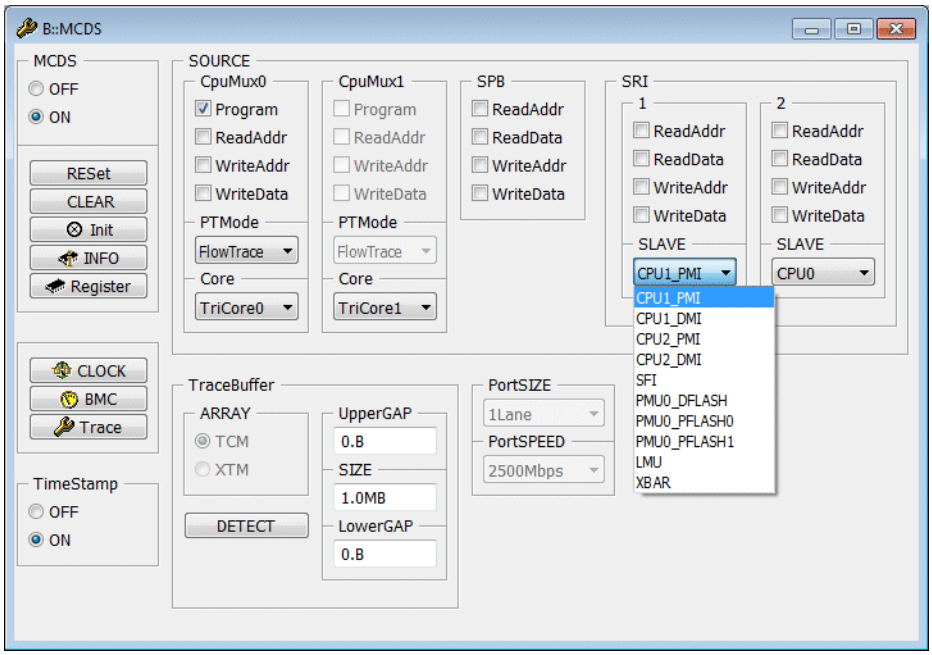




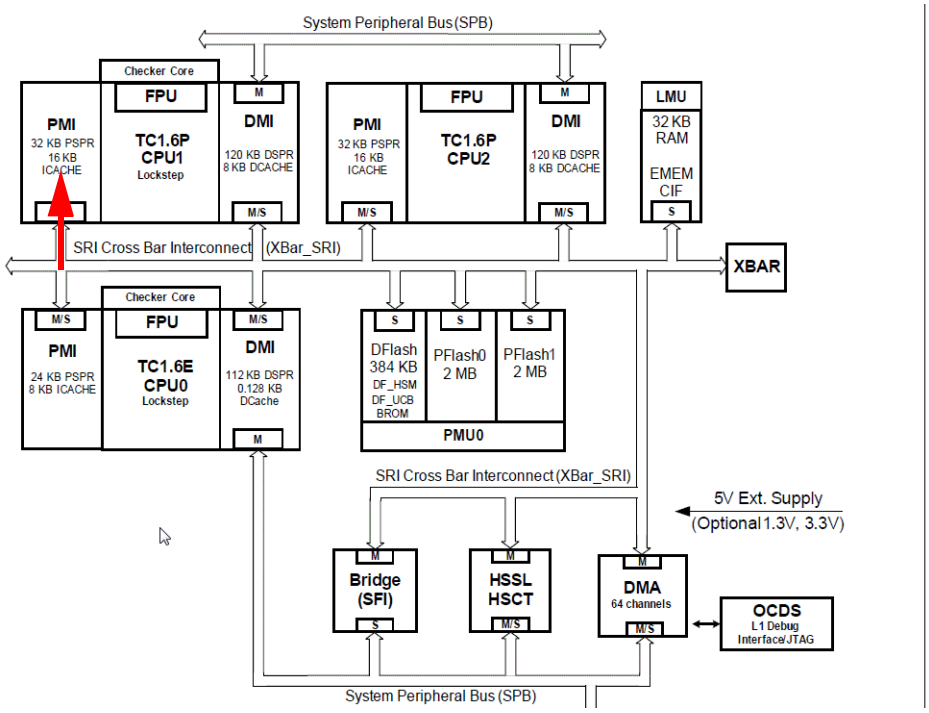
Shared Resource Interconnect as Trace Source

Due to bandwidth issues it is not possible to generated trace messages for all transfers on the SRI.

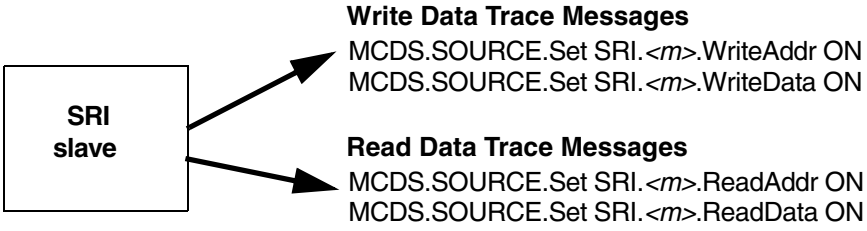
Trace messages can be generated for transfers to two selected slaves.

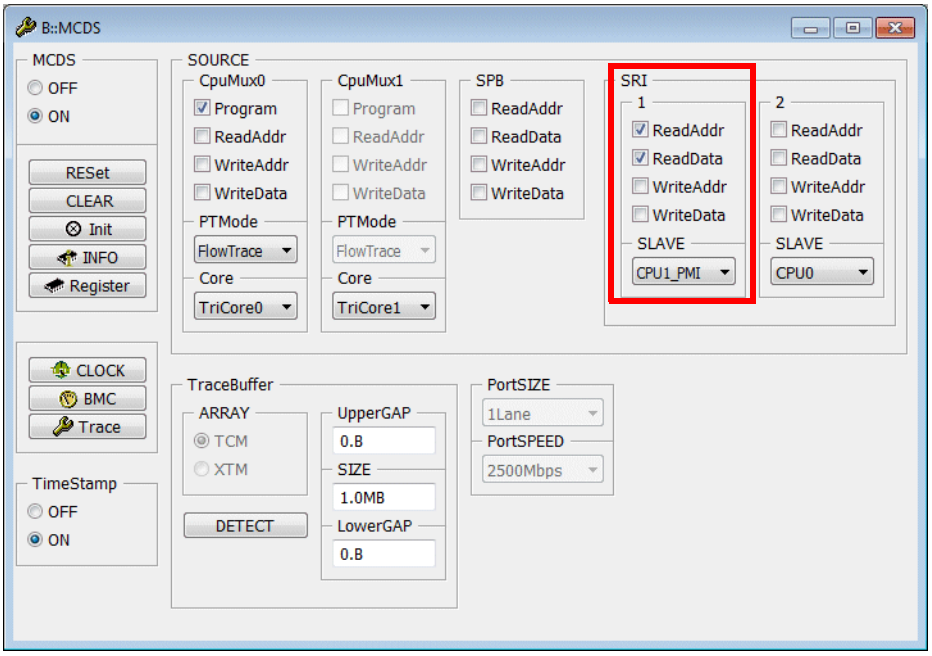


```
; Select PMI of CPU1 as first SRIslave

; This selection enables MCDS to generate trace messages for all
; transfers to the PMI of core 1
MCDS.SOURCE.Set SRI.1.SLAVE CPU1_PMI
```



For each SRI slave the following messages can be generated:





Message Display in TRACE32

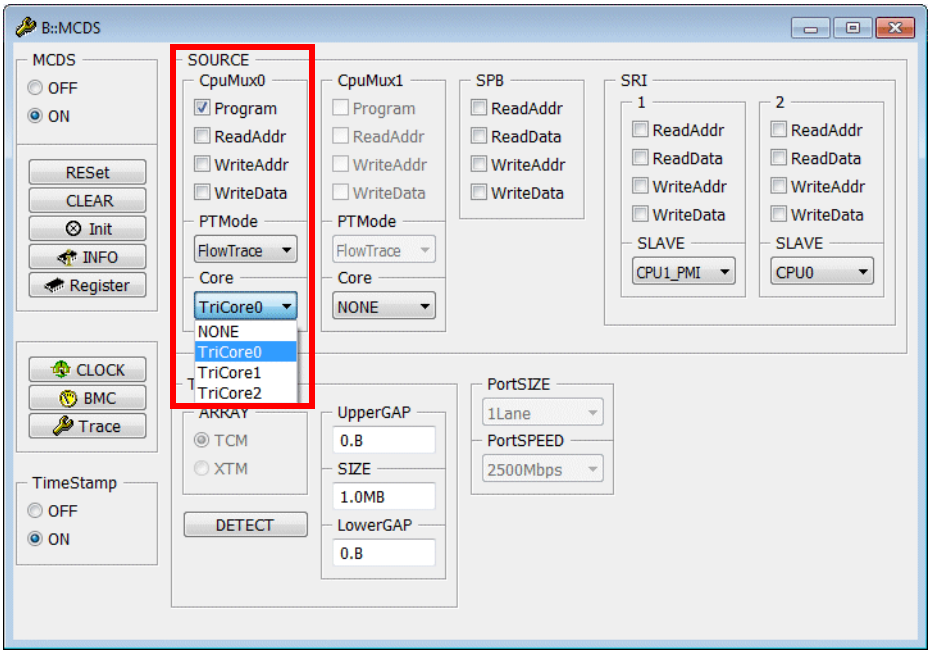
In general, the following applies:

- A TRACE32 instance displays trace information for all the cores it controls.
- Trace information generated by the SPB/SRI is always displayed in all TRACE32 instances that are started to debug the TriCore cores of the AURIX chip.

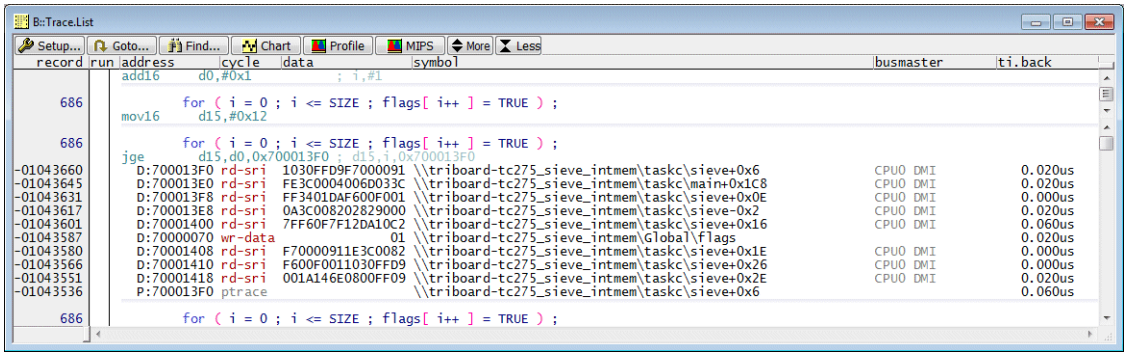
A TRACE32 instance controls one TriCore core (TC 1.6.1 CPU0 here).

- Please make sure that you have selected the core that is controlled by the TRACE32 instance in the MCDS window. For example:

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```



All trace messages enabled for the selected core and SPB/SRI are displayed in the TRACE32 Trace Listing.



Three TRACE32 instances are started, each TRACE32 instance controls one TC 1.6.1 core.

Since MCDS can generated trace information only for 2 cores, you have to configure the trace multiplexers.

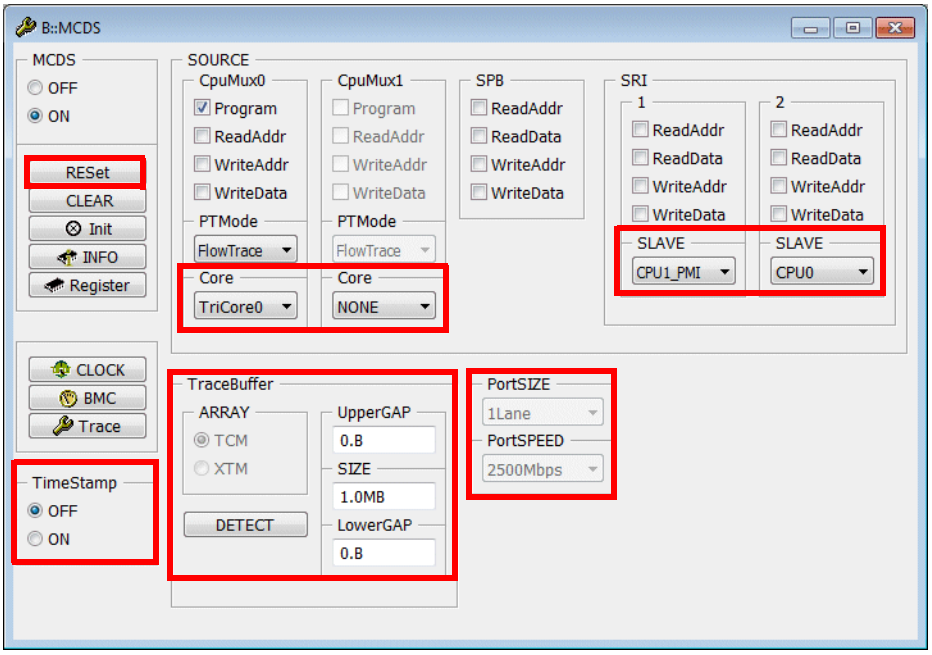
```
; configuration command for trace multiplexer CpuMux0
MCDS.SOURCE.Set CpuMux0.Core TriCore0 | TriCore1 | TriCore2

; configuration command for trace multiplexer CpuMux1
MCDS.SOURCE.Set CpuMux1.Core TriCore1 | TriCore2 | OTGM
```

Each TRACE32 instance has its own MCDS window.

Since more than one TRACE32 instance can configure MCDS (single source) the following rules apply:

1. Joint settings

The TRACE32 Resource Management maintains consistency between the TRACE32 instances.



MCDS reset

> **MCDS.RESet**

On-chip/off-chip trace configuration

> **MCDS.TraceBuffer.SIZE** *<size>*
>
> **MCDS.TraceBuffer.LowerGAP** *<size>*
>
> **MCDS.TraceBuffer.UpperGAP** *<size>*
>
> **MCDS.PortSIZE** *<lanes>*
>
> **MCDS.PortSPEED** *<speed>*

Timestamp configuration

> **MCDS.TimeStamp OFF | ON**

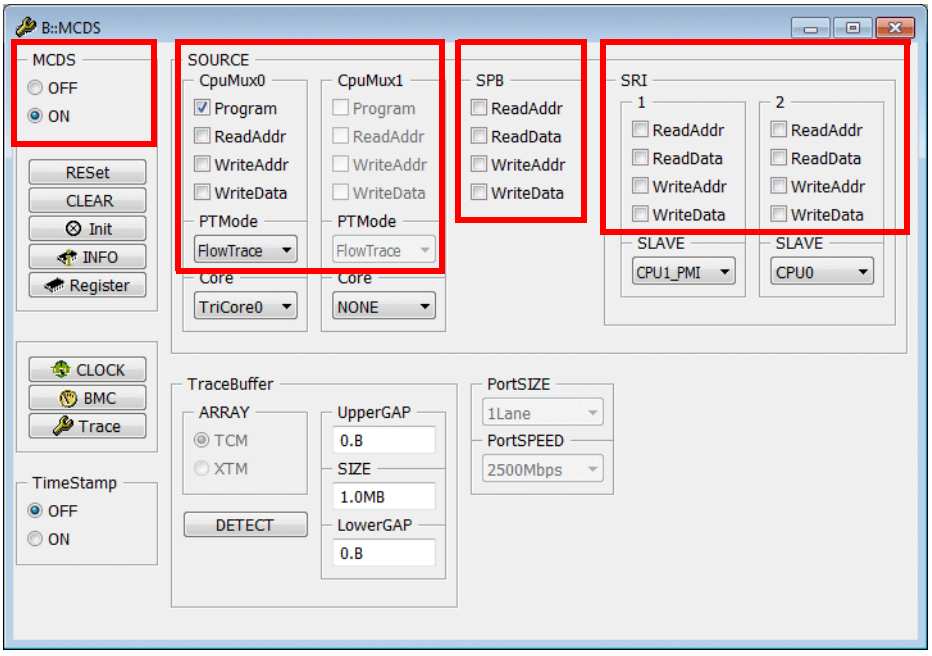
MCDS.SOURCE.Set CpuMux0.Core TriCore0 | TriCore1 | TriCore2

MCDS.SOURCE.Set CpuMux1.Core TriCore1 | TriCore2 | OTGM

MCDS.SOURCE.Set SRI.1|2.SLAVE *<slave>*

2. Exclusive settings

These settings can be done by each TRACE32 instance individually.



Connect/disconnect core from MCDS

MCDS.ON

MCDS.OFF

Enabling/disabling of trace messages

MCDS.SOURCE.Set CpuMux0.<*source*> **ON | OFF**

MCDS.SOURCE.Set CpuMux1.<*source*> **ON | OFF**

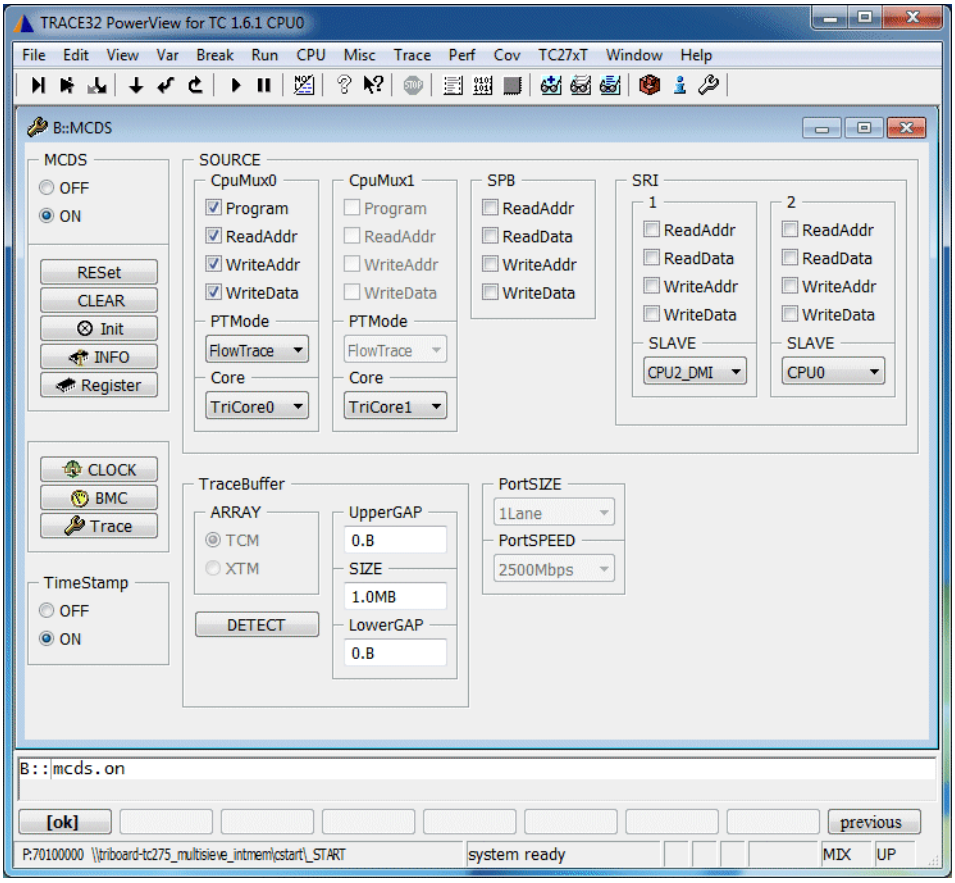
MCDS.SOURCE.Set SPB.<*source*> **ON | OFF**

MCDS.SOURCE.Set SRI 1.<*source*> **ON | OFF**

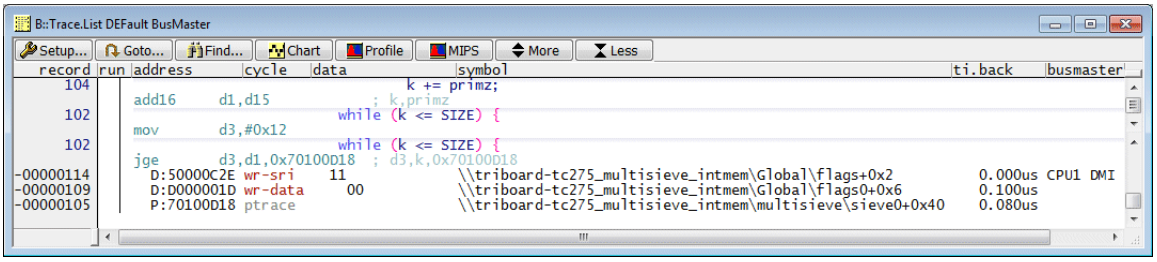
MCDS.SOURCE.Set SRI 2.<*source*> **ON | OFF**

Example

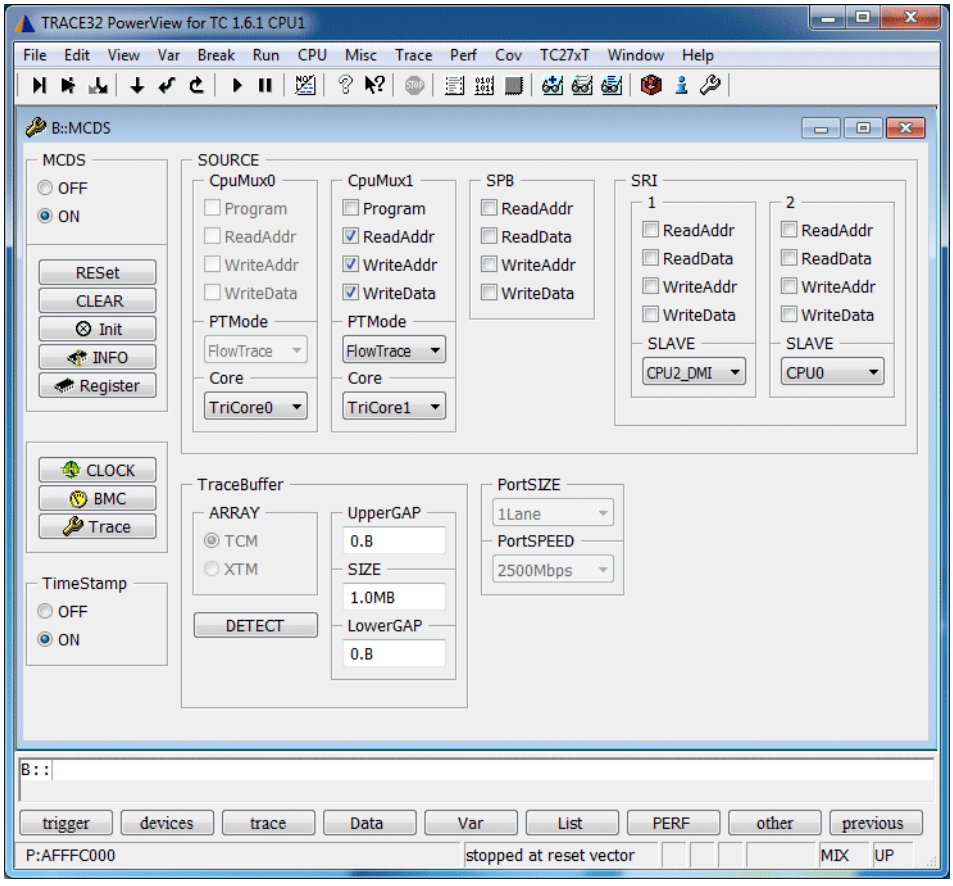
MCDS settings for TC 1.6.1 CPU0



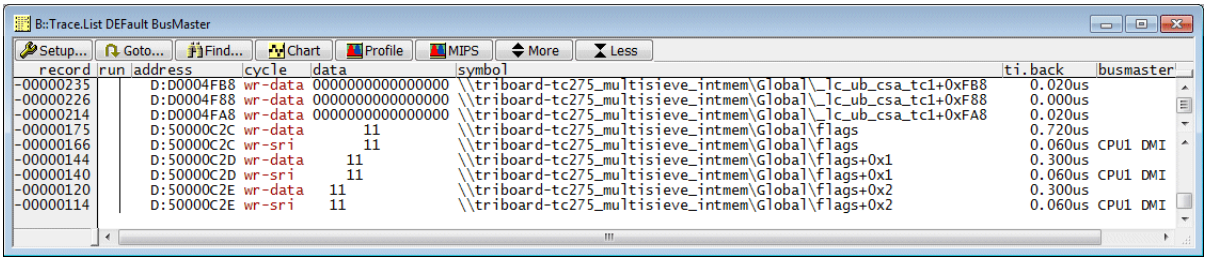
All trace messages enabled for TriCore0 and all SPB/SRI trace messages enabled in any TRACE32 instance are displayed in the TRACE32 Trace Listing.



MCDS setting for TC 1.6.1 CPU1

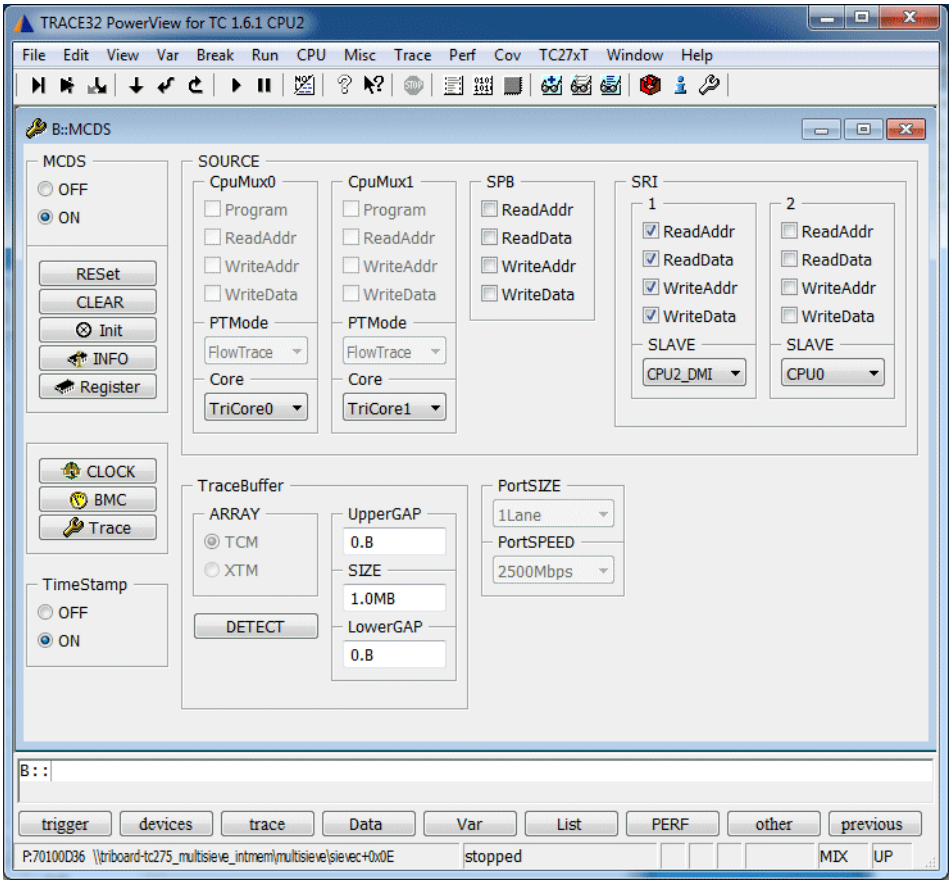


All trace messages enabled for TriCore1 and all SPB/SRI trace messages enabled in any TRACE32 instance are displayed in the TRACE32 Trace Listing.

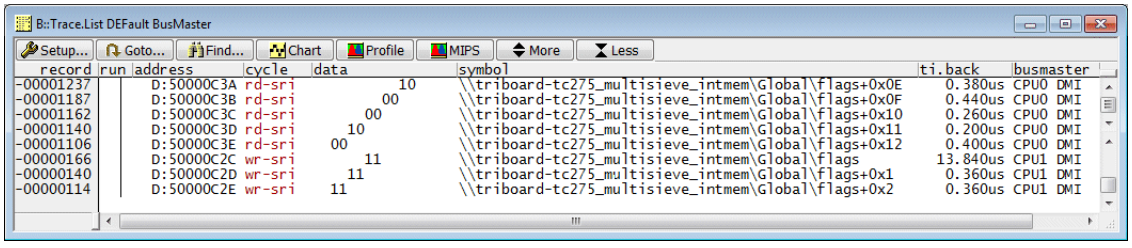


record	run	address	cycle	data	symbol	ti.back	busmaster
-00000235		D:D0004FB8	wr-data	0000000000000000	\\triboard-tc275_multisieve_intmem\Global_lc_ub_csa_tc1+0xFB8	0.020us	
-00000226		D:D0004F88	wr-data	0000000000000000	\\triboard-tc275_multisieve_intmem\Global_lc_ub_csa_tc1+0xF88	0.000us	
-00000214		D:D0004FA8	wr-data	0000000000000000	\\triboard-tc275_multisieve_intmem\Global_lc_ub_csa_tc1+0xFA8	0.020us	
-00000175		D:50000C2C	wr-data	11	\\triboard-tc275_multisieve_intmem\Global\flags	0.720us	
-00000166		D:50000C2C	wr-sri	11	\\triboard-tc275_multisieve_intmem\Global\flags	0.060us	CPU1 DMI
-00000144		D:50000C2D	wr-data	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x1	0.300us	
-00000140		D:50000C2D	wr-sri	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x1	0.060us	CPU1 DMI
-00000120		D:50000C2E	wr-data	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x2	0.300us	
-00000114		D:50000C2E	wr-sri	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x2	0.060us	CPU1 DMI

MCDS setting for TC 1.6.1 CPU2



All SPB/SRI trace messages enabled in any TRACE32 instance are displayed in the TRACE32 Trace Listing.

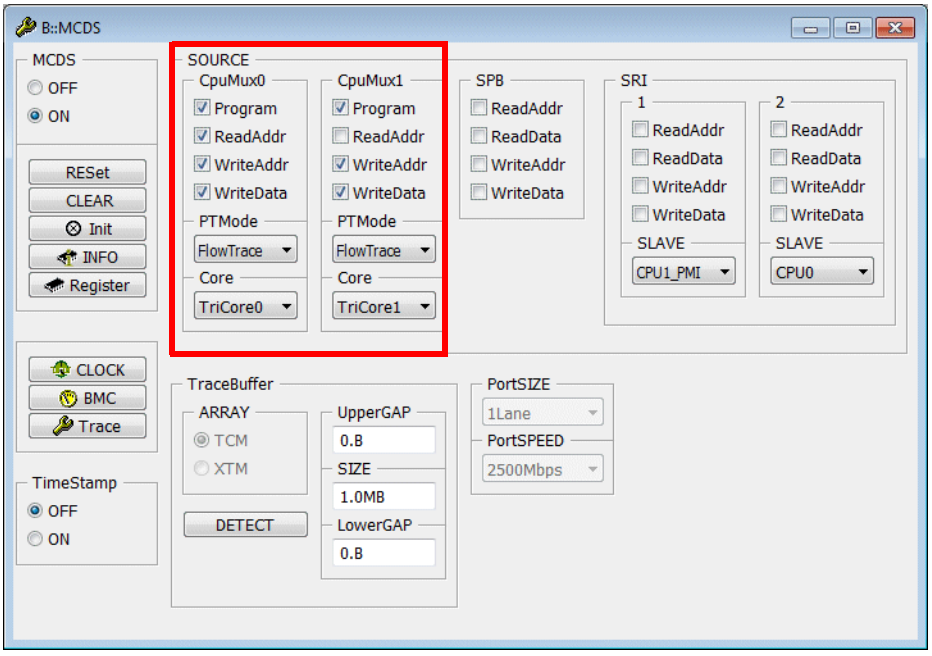


A TRACE32 instance controls all TriCore cores.

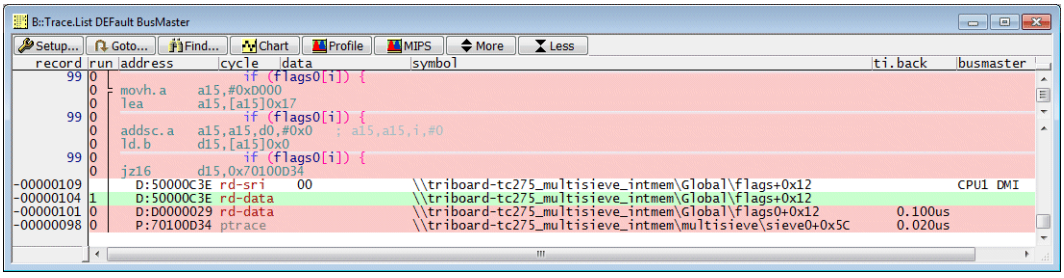
Since MCDS can generated trace information only for 2 cores, you have to configure the trace multiplexers.

```
; configuration command for trace multiplexer CpuMux0
MCDS.SOURCE.Set CpuMux0.Core TriCore0 | TriCore1 | TriCore2

; configuration command for trace multiplexer CpuMux1
MCDS.SOURCE.Set CpuMux1.Core TriCore1 | TriCore2 | OTGM
```

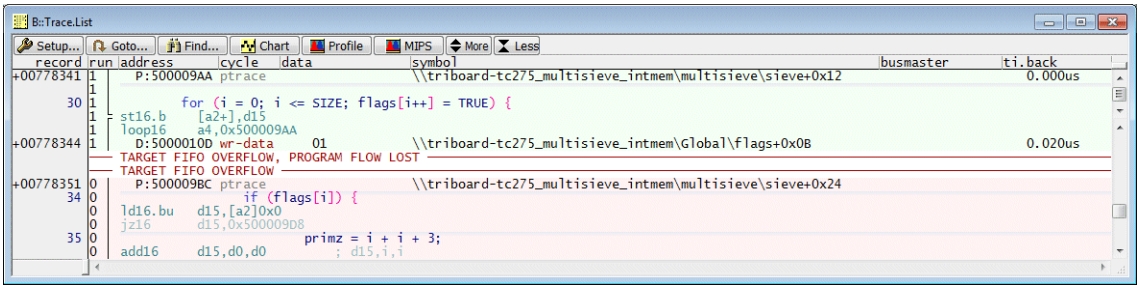
One MCDS window is provided to control the message generation for all cores.



All trace messages enabled for the selected cores and SPB/SRI trace messages are displayed in the TRACE32 Trace Listing.



FIFOFULL



TARGET FIFO OVERFLOW, PROGRAM FLOW LOST occurs, when more trace information is generated than can be transferred into the EMEM.

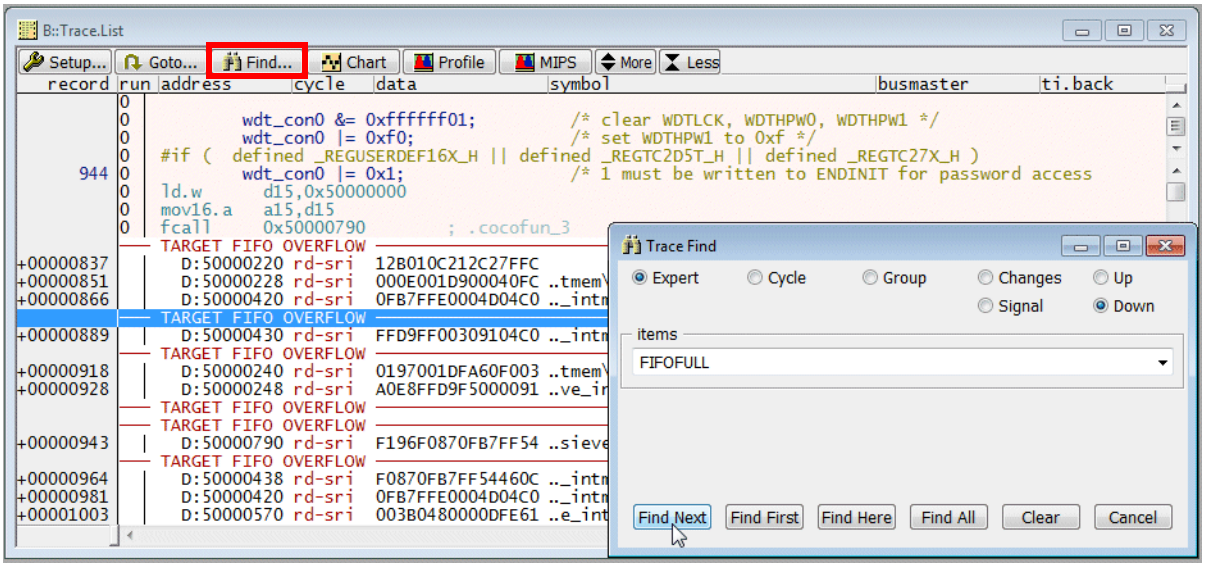
In order to get an immediate display of the trace contents TRACE32 uploads only the currently displayed section from the trace buffer to the host.

To check the number of FIFOFULLs it is recommended to upload the complete trace contents to the host by the command: **`Trace.FLOWPROCESS`**.

The complete number of FIFOFULL can be displayed by:

```
PRINT %Decimal Trace.FLOW.FIFOFULL()
```

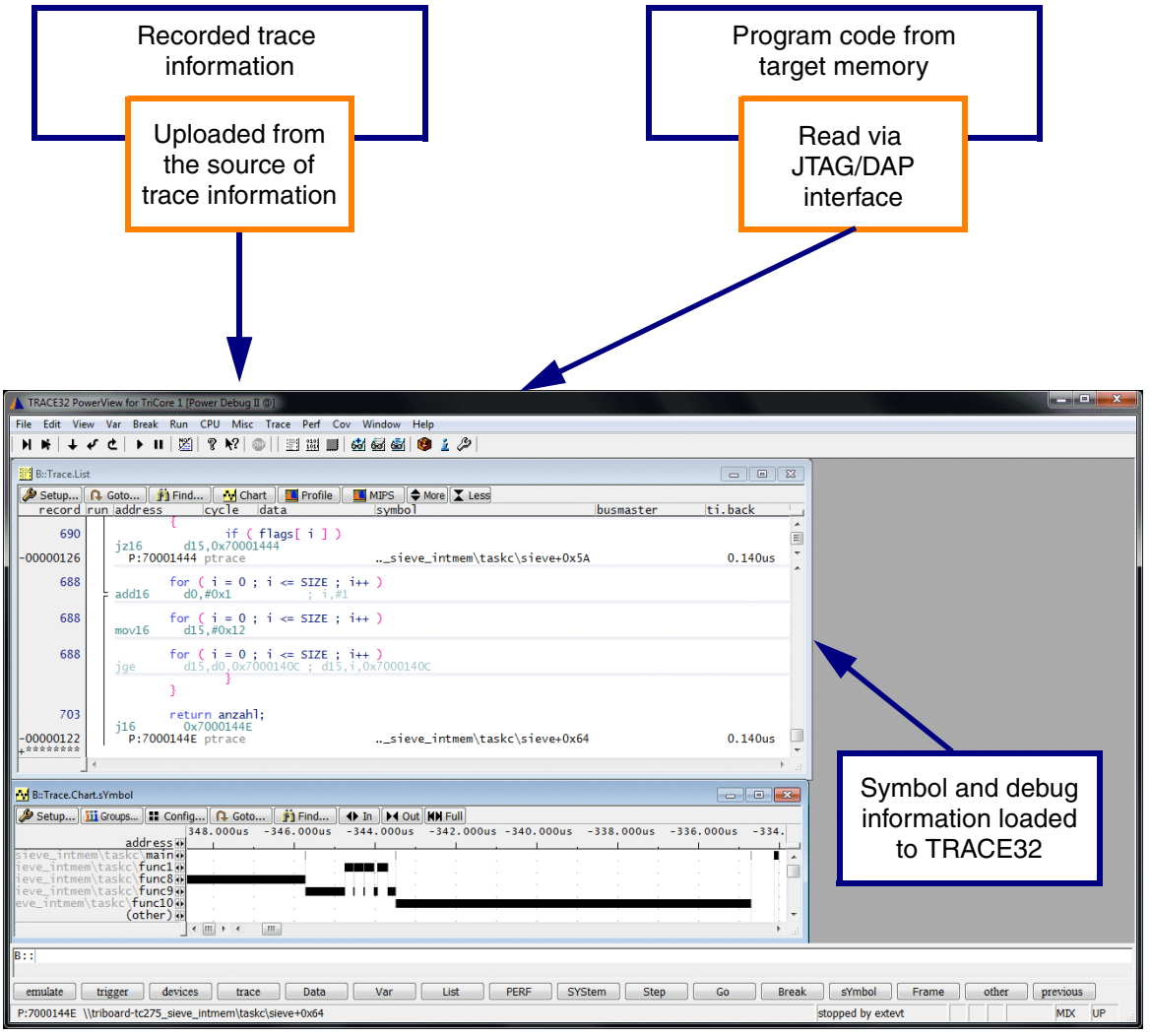
The single FIFOFULLs can be found in the trace:



Sources of Information for the Trace Display

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/DAP interface.
- The symbol and debug information already loaded to TRACE32.



Influencing Factors on the Trace Information

The main influencing factor on the trace information is the MCDS. It specifies what type of trace messages is generated for the user.

Basic settings for trace messages were introduced in **"Trace Sources and Their Messages"**, page 37.

More advanced settings are described later in **"Trace Control by Filter and Trigger - Overview"**, page 93.

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

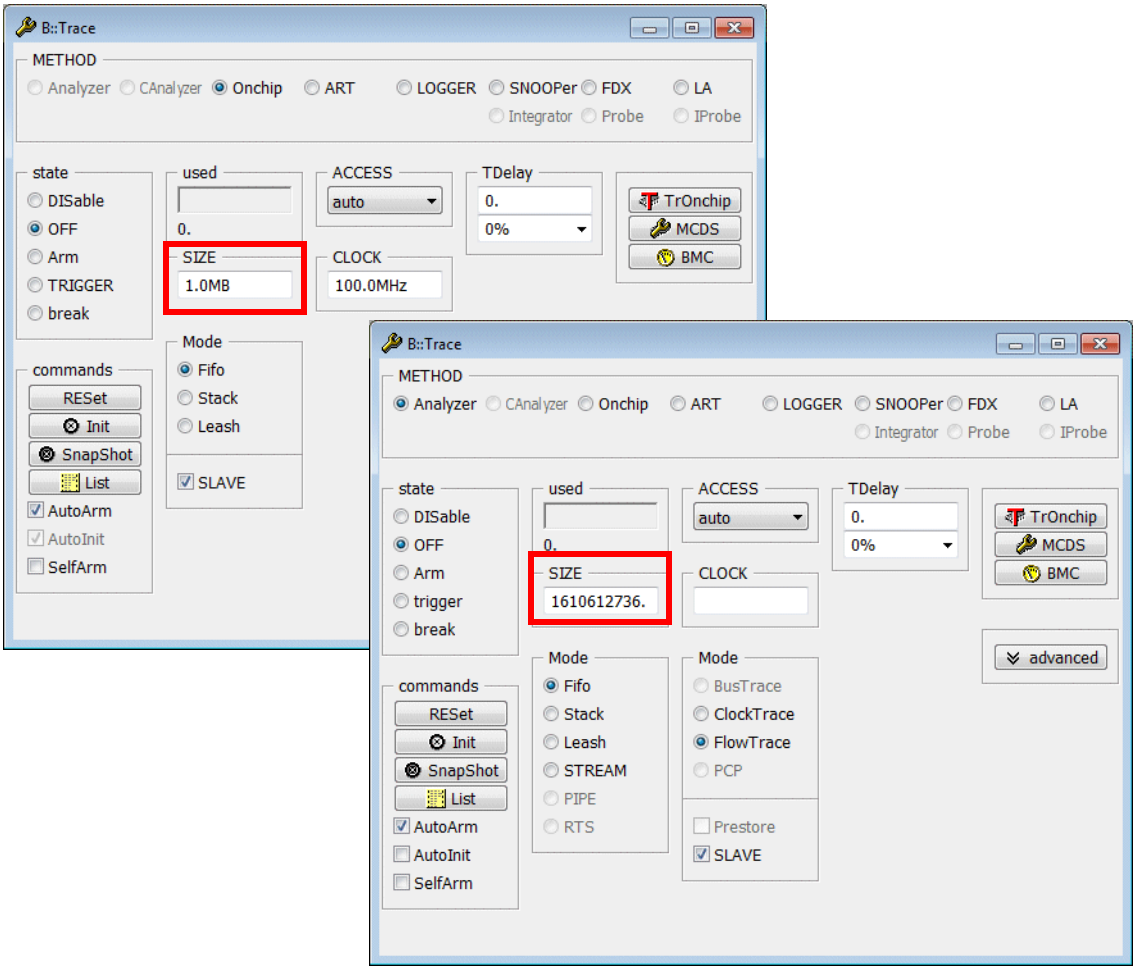
Mode Setting

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:

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

  Onchip trace buffer (EMEM)



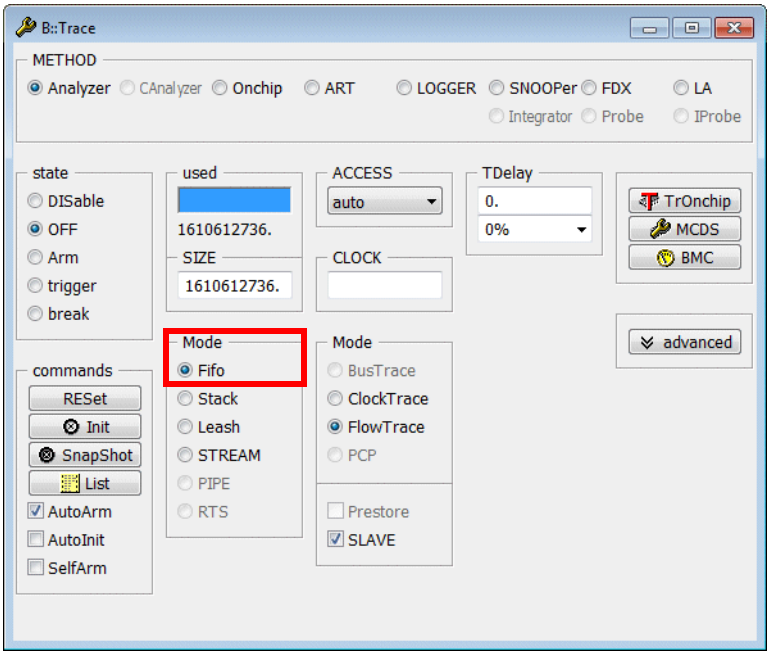
Trace memory of PowerTrace hardware

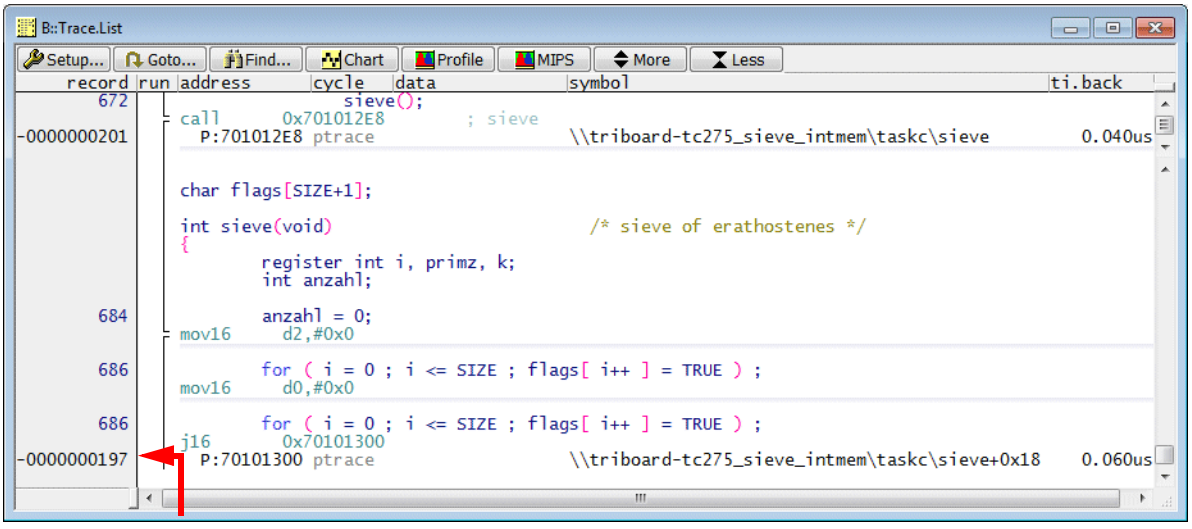
- **STREAM Mode (PowerTrace hardware only):** STREAM mode specifies that the trace information is immediately streamed to a file on the host computer. Peak loads at the trace port are intercepted by the trace memory of the PowerTrace, which can be considered as a large FIFO.

  STREAM mode allows a trace memory of several Tera Frames.

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

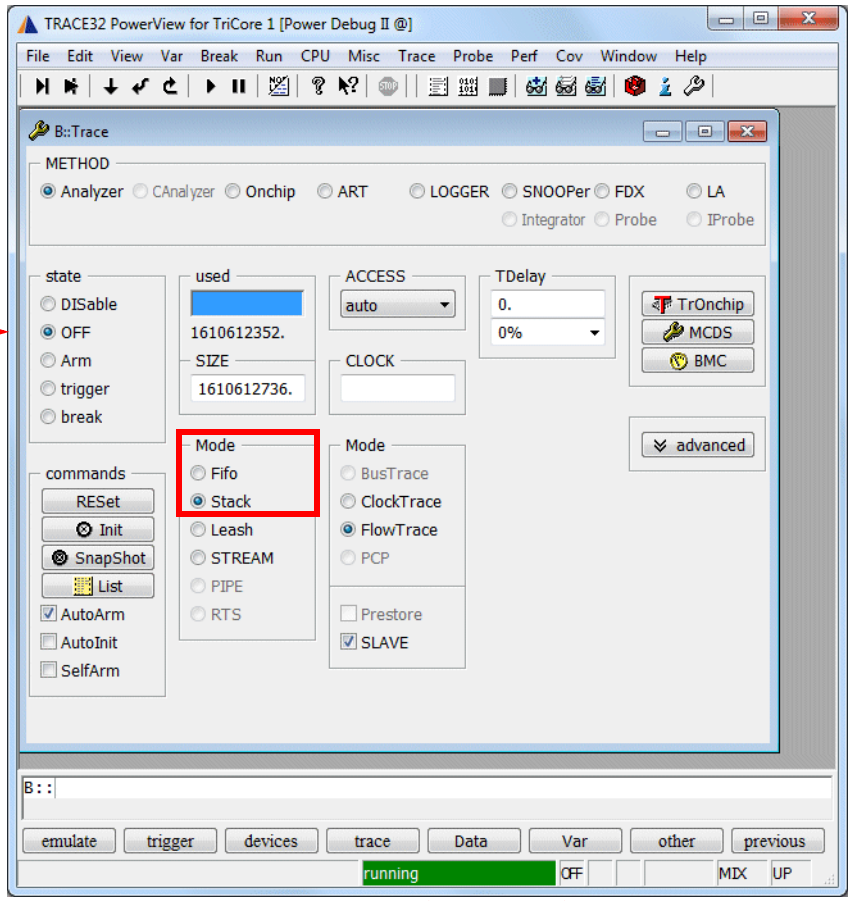
```
Trace.Mode Fifo                         ; default mode

                                        ; 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
```

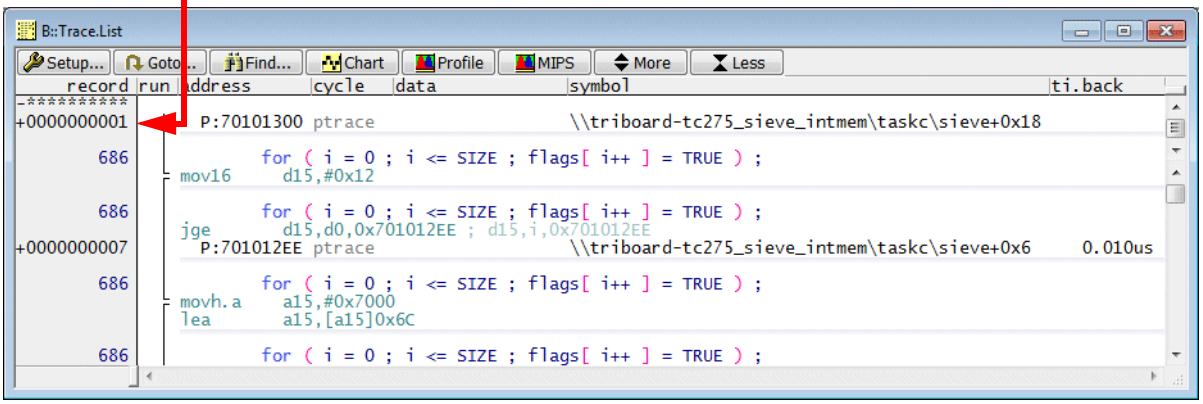


The trace recording is stopped as soon as the trace memory is full (OFF state)

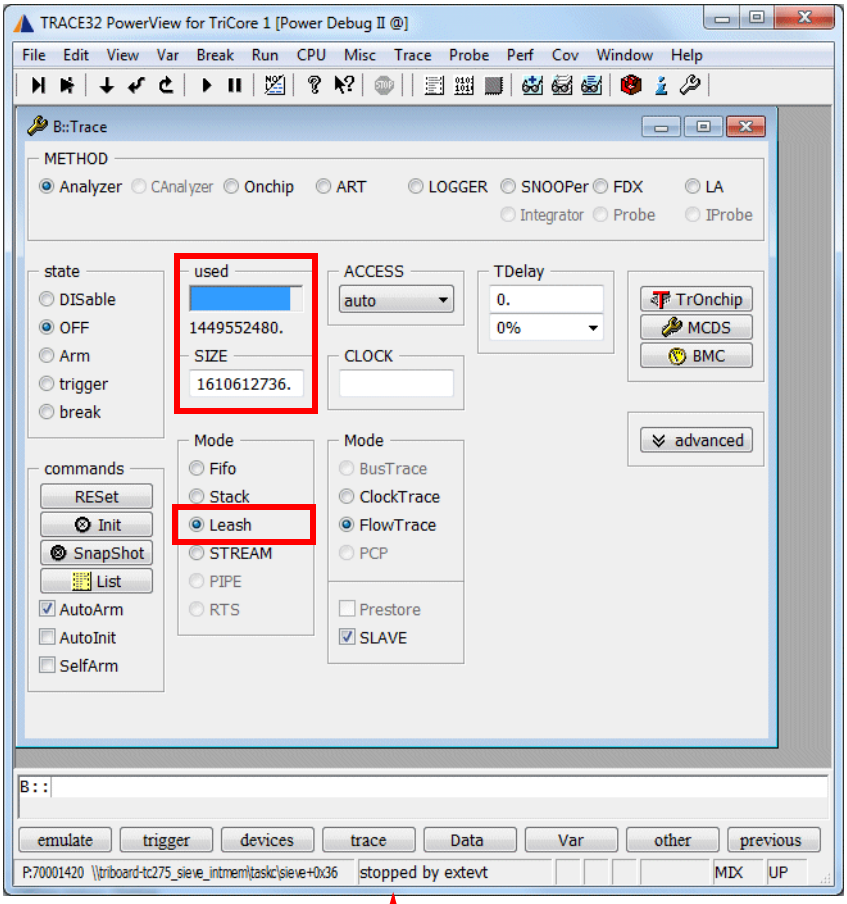
Green **running** in the Debug field indicates that program execution is running

OFF in the Trace field indicates that the trace recording is switched off

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



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

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



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

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

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 up to several T Frames.

STREAM mode required 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 specific a different name and location for the streaming file.

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

TRACE32 stops the streaming when less then the 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.

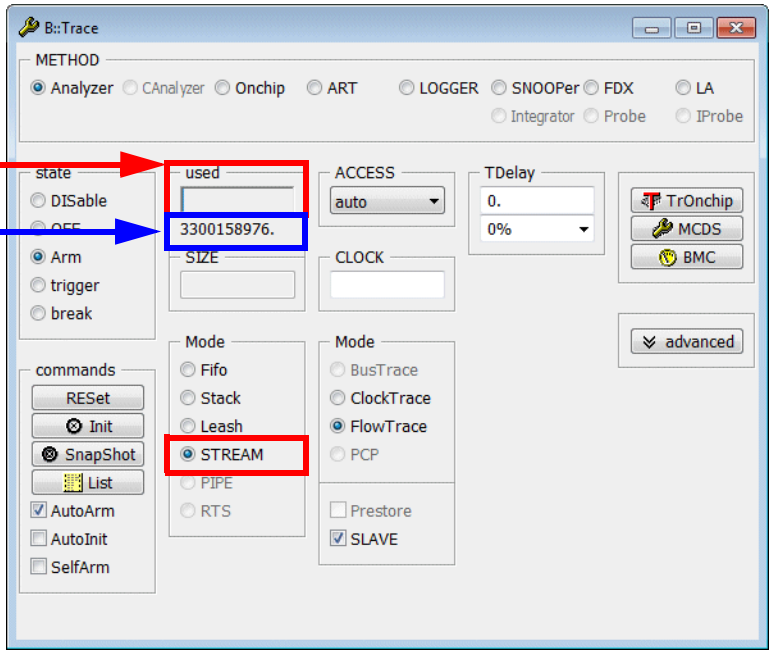
```
Trace.STREAMFileLimit 5000000000.     ; streaming file is limited to
                                      ; 5 GByte

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.

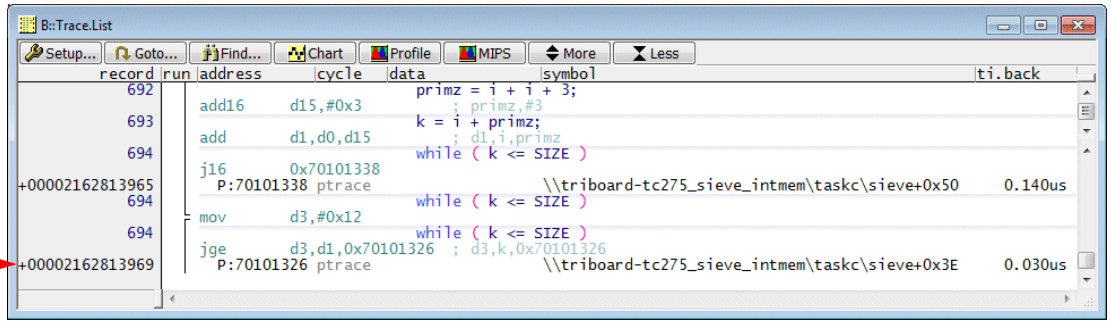
STREAM mode can only be used if the average data rate at the trace port does not exceed the maximum transmission rate of the host interface in use. Peak loads at the trace port are intercepted by the trace memory within the PowerTrace, which can be considered to be operating as a large FIFO.

```
Trace.Mode STREAM                       ; trace information is immediately
                                        ; streamed to a file on the host
                                        ; computer

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

used graphically: number of records buffered by the trace memory in the PowerTrace

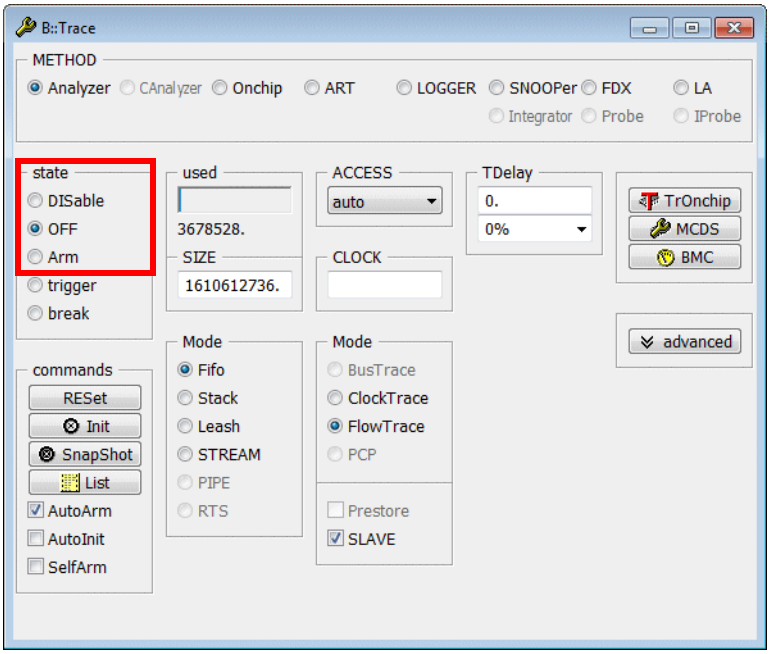


used numerically: Number of records saved to streaming file

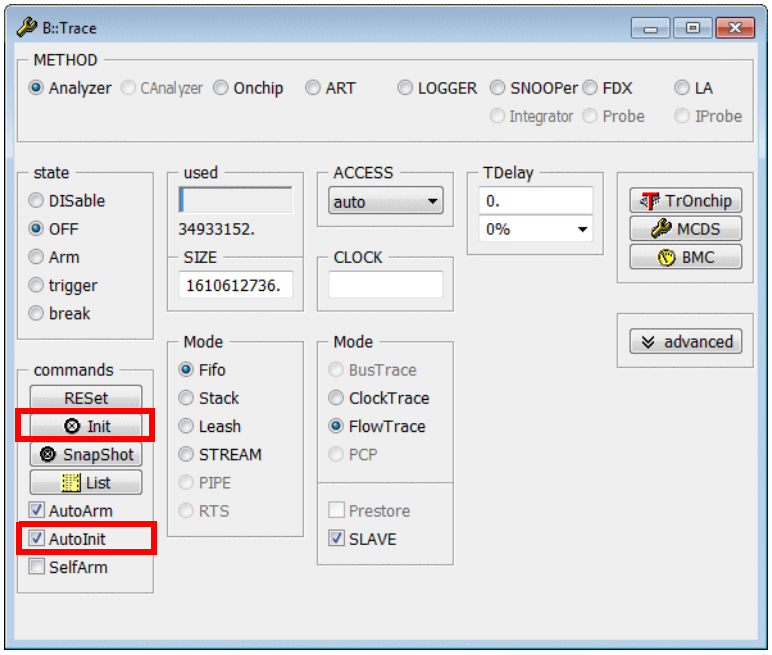


STREAM mode can generate very large record numbers

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



States of the Trace	
DISable	The trace is disabled.
OFF	The trace is not recording. The trace contents can be displayed.
Arm	The trace is recording. The trace contents can not be displayed.



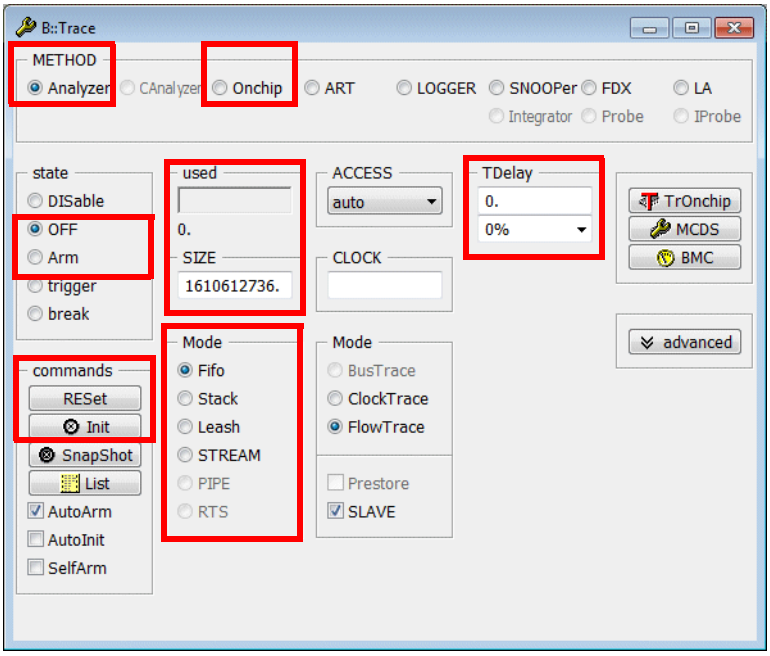
Init Button	Clear the trace memory. All other settings in the Trace Configuration window remain valid.
AutoInit CheckBox	ON: The trace memory is cleared whenever the program execution is started (Go, Step). Please be aware that the onchip trace memory always start recording at the lowest address. As a result the **AutoInit** option is not required.

Each TRACE32 instance has its own Trace Configuration window.

Since more the on TRACE32 instance can configure the trace (single source) the following rules apply:

1. Joint settings

The TRACE32 Resource Management maintains consistency between the TRACE32 instances.



Selection of the trace sink

> **Trace.METHOD Analyzer**
>
> **Trace.METHOD Onchip**

For the AURIX all cores can either use the on-chip trace or off-chip trace. Therefore, either the trace method Onchip or the trace method Analyzer has to be selected in all TRACE32 instances. As soon as an inconsistent selection is done in a TRACE32 instance, the TRACE32 Resource Management disables the traces in the other TRACE32 instances.

Trace state

```
Trace.OFF
Trace.Arm
```

Trace reset/init

```
Trace.RESet
Trace.Init
```

Trace size

```
Trace.SIZE <size>
```

Trace mode

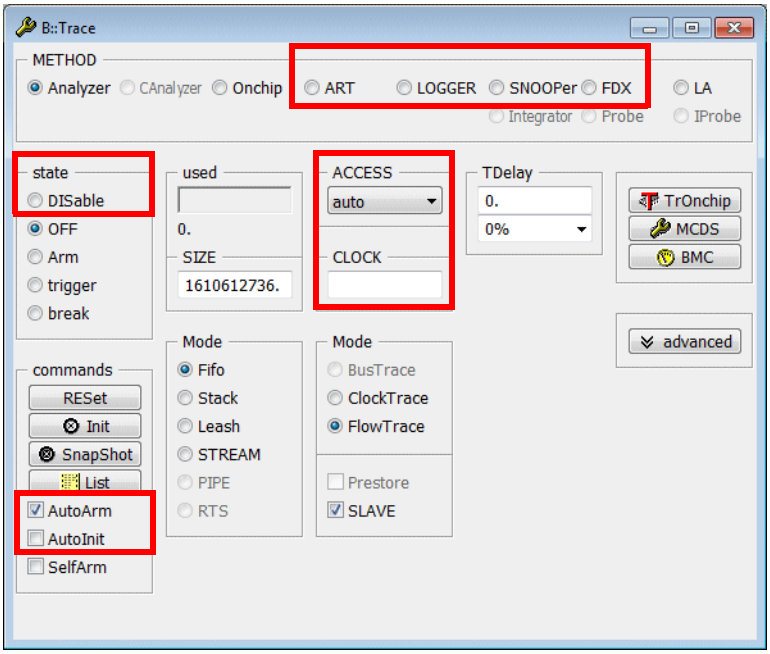
```
Trace.Mode Fifo | Stack | Leash | STREAM
```

Trigger delay

```
Trace.TDelay <delay>
```

2. Exclusive settings

These settings can be done by each TRACE32 instance individually.



Trace method

All TRACE32 software trace methods have their own resources in their TRACE32 instance.

> **Trace.Mode ART | LOGGER | SNOOPer | FDX**

Trace disable

> **Trace.DISable**

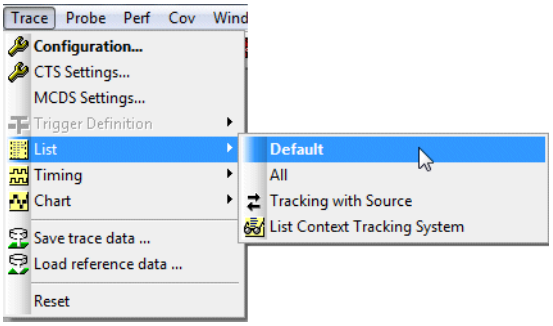
Trace AutoArm and AutoInit

> **Trace.AutoArm** [**ON** | **OFF**]
>
> **Trace.AutoInit** [**ON** | **OFF**]

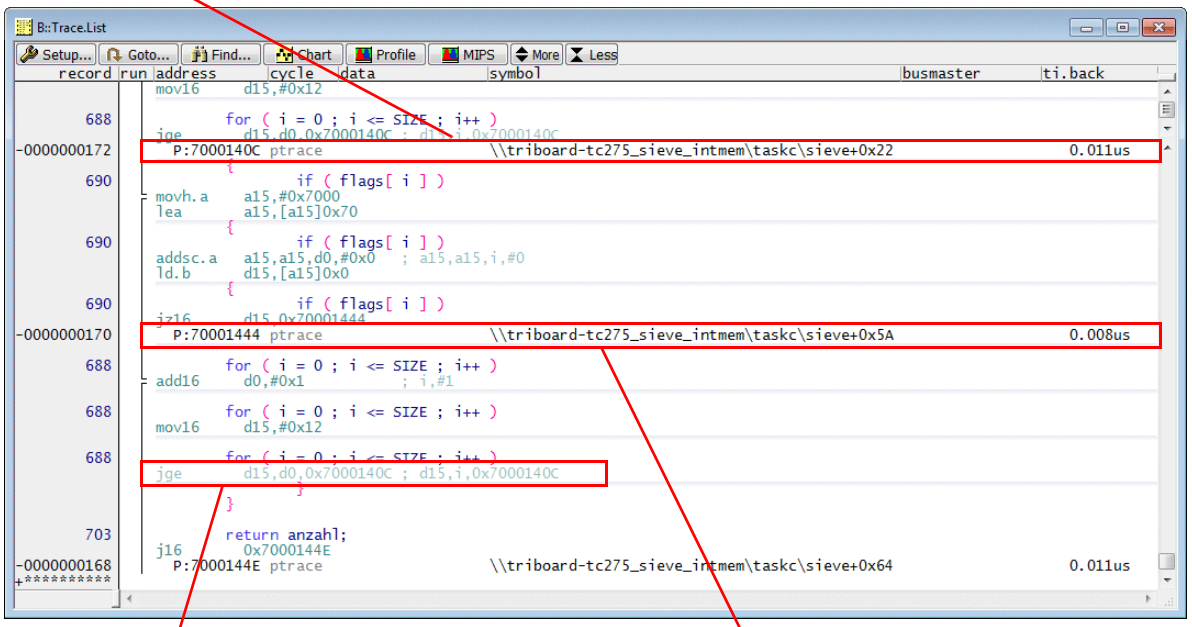
Access to source code/definition of the core clock

> **Trace.ACCESS** *<access>*
>
> **Trace.CLOCK** *<core_clock>*

Default Listing



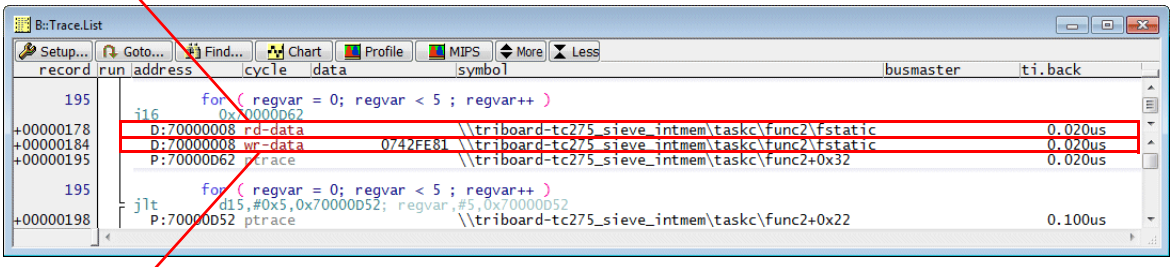
ptrace: Instruction Pointer Call Message/Instruction Pointer Message with timestamp



Conditional branch not taken (pastel)

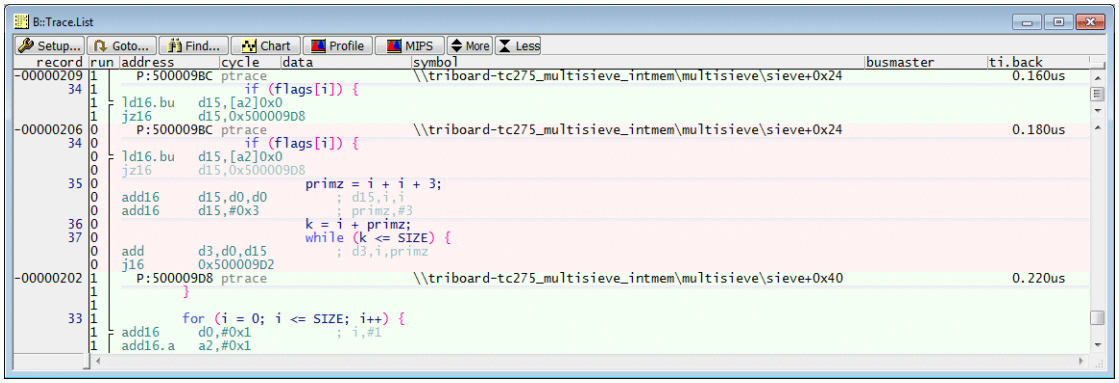
Conditional branch taken

rd-data: Read Data Trace Message with timestamp (read-address only)

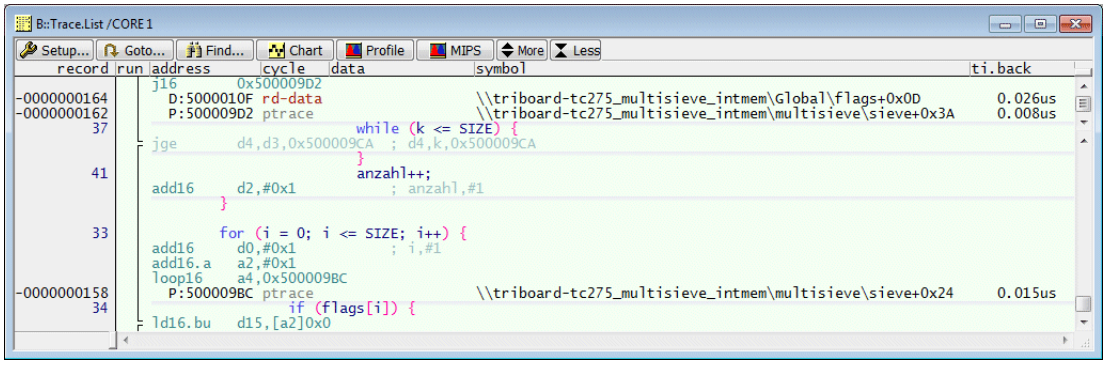


wr-data: Write Data Trace Message with timestamp

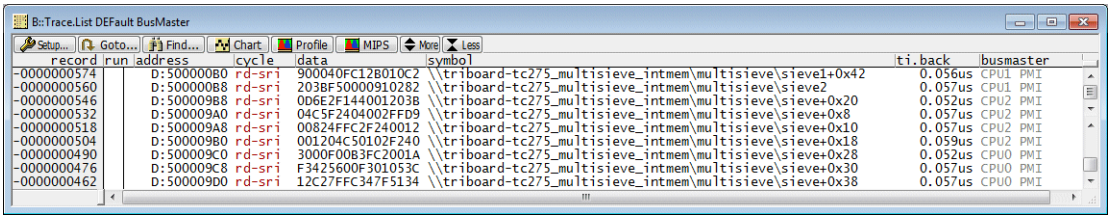
The trace information for all cores is displayed by default in the Trace.List window if you are working with an **SMP system**. The column run and the coloring of the trace information are used for core indication.



Trace.List /CORE<*n*> The commands allows a per core display

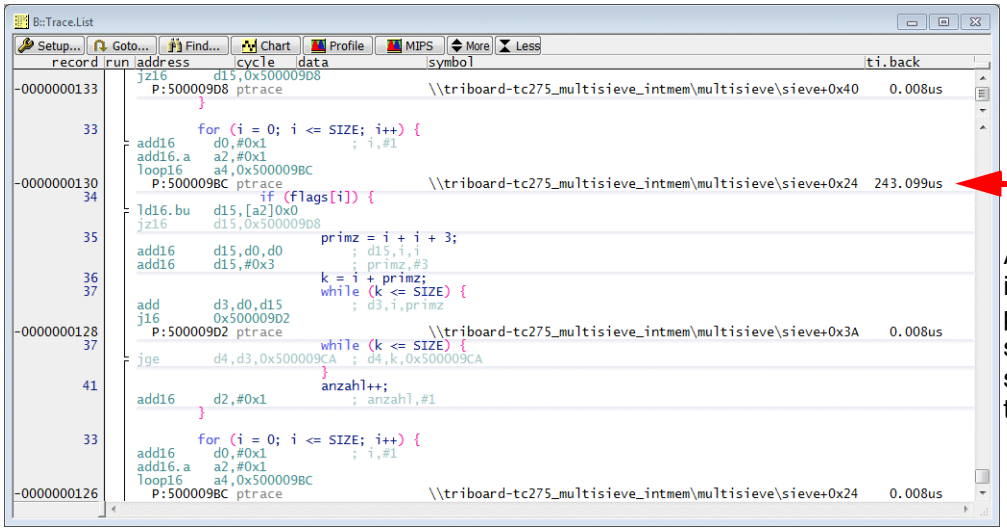


```
Trace.List DEFault BusMaster
```



record	run	address	cycle	data	symbol	ti.back	busmaster
-0000000574		D:500000B0	rd-sri	900040FC12B010C2	\\triboard-tc275_multisieve_intmem\multisieve\sieve1+0x42	0.056us	CPU1 PMI
-0000000560		D:500000B8	rd-sri	203BF50000910282	\\triboard-tc275_multisieve_intmem\multisieve\sieve2	0.057us	CPU1 PMI
-0000000546		D:500009B8	rd-sri	0D6E2F144001203B	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x20	0.052us	CPU2 PMI
-0000000532		D:500009A0	rd-sri	04C5F2404002FFD9	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x8	0.057us	CPU2 PMI
-0000000518		D:500009A8	rd-sri	00824FFC2F240012	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x10	0.057us	CPU2 PMI
-0000000504		D:500009B0	rd-sri	001204C50102F240	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x18	0.059us	CPU2 PMI
-0000000490		D:500009C0	rd-sri	3000F00B3FC2001A	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x28	0.052us	CPU0 PMI
-0000000476		D:500009C8	rd-sri	F3425600F301053C	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x30	0.057us	CPU0 PMI
-0000000462		D:500009D0	rd-sri	12C27FFC347F5134	\\triboard-tc275_multisieve_intmem\multisieve\sieve+0x38	0.057us	CPU0 PMI

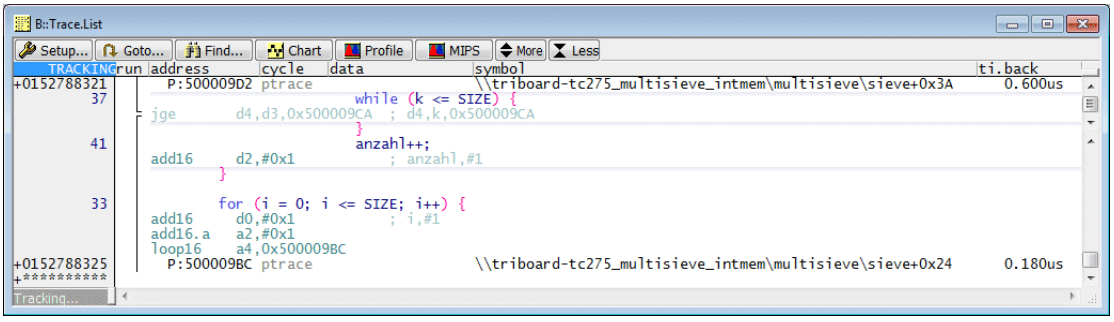
Off-chip trace: Please be aware that the AURIX chip buffers a bigger number of trace messages before they are sent out together via the serial interface. Since the TRACE32 timestamps the trace information when it is saved into the trace memory of the PowerTrace, the timestamps are imprecise and not suitable especially to measure the runtime of short program sections.



A big timestamp indicates the point when the serial trace started to stream trace information

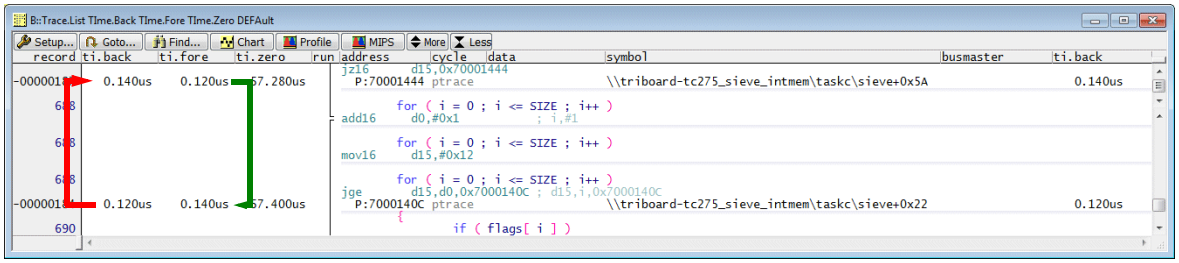
If exact timestamps are important for your trace analysis, you have to enable Timestamp Messages. Please refer to **"2. Enable MCDS timestamp messages"**, page 27 for details.

Enabling the Timestamp Messages has the caveat that the display of the trace information might need some time because TRACE32 has to process the trace information always from the start of the trace recording. As long as TRACE32 processes the trace information "Tracking" is displayed.

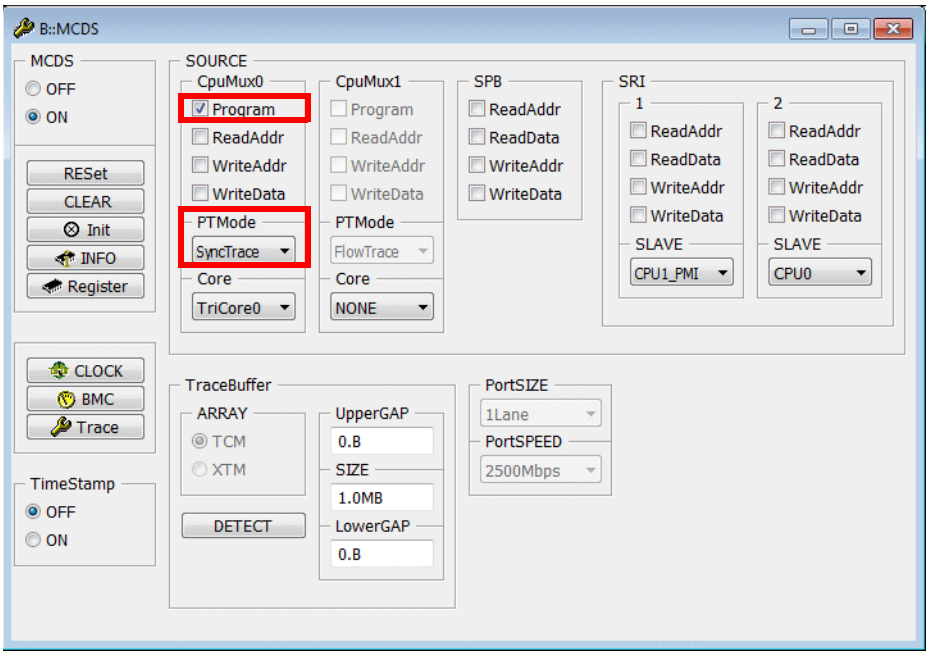


TIme.BACK	Time relative to the previous record (left arrow in red)
TIme.FORE	Time relative to the next record (right arrow in green).
TIme.ZERO	Timestamp Messages enabled: Time relative to the first record in the trace (zero point) PowerTrace/PREPROCESSOR SERIAL with Timestamp Messages disabled: Time relative to the TRACE32 global zero point The TRACE32 global zero point is established when the communication between the debugger and the master core is established by **SYStem.Up**.

```
Trace.List TIme.BACK TIme.FORE TIme.ZERO DEFault
```

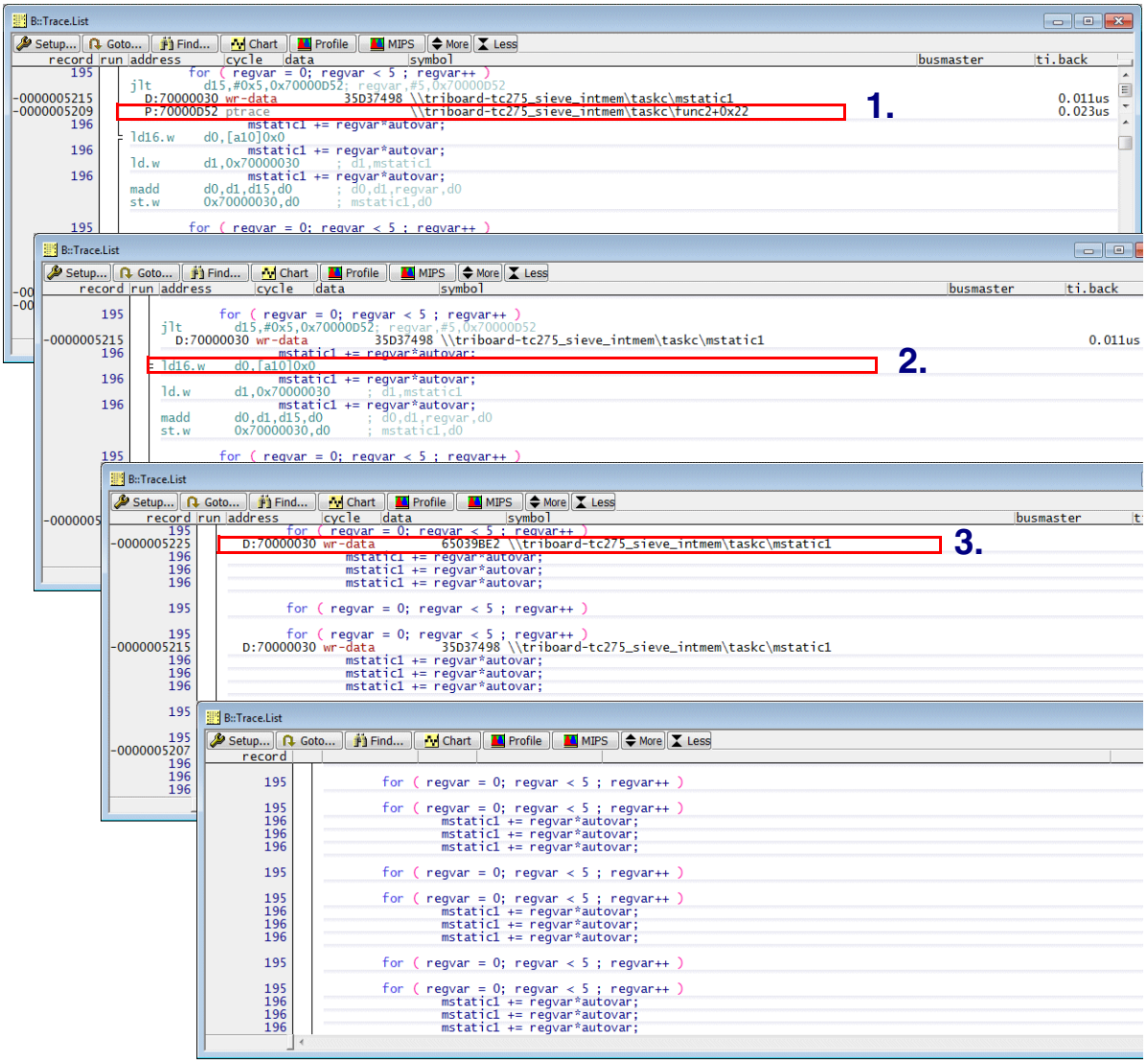


Information on the executed instructions is only generated on branches by default. The timestamps per instruction become more detailed, if Instruction Pointer Messages are enabled.



MCDS.SOURCE.Set CpuMux<*n*>.PTMode SyncTrace

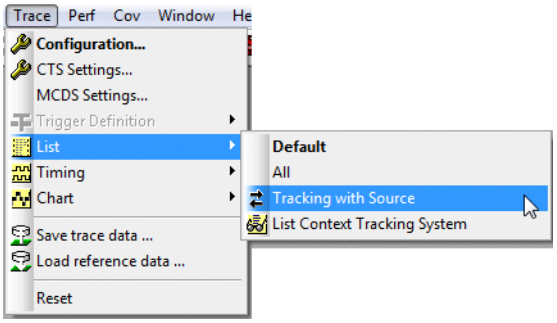
But this reduces tracing time, since more trace packets are generated (about four times less).



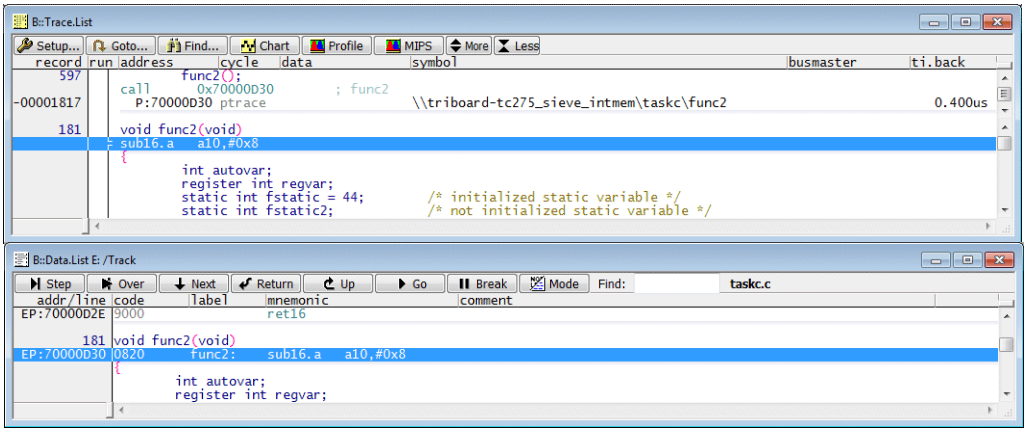
1. time Less	Suppress the display of the ptrace information (ptrace).
2. time Less	Suppress the display of the assembly code.
3. time Less	Suppress the data access information (e.g. wr-data cycles).

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

Correlating the Trace Listing with the Source Listing



Active Window

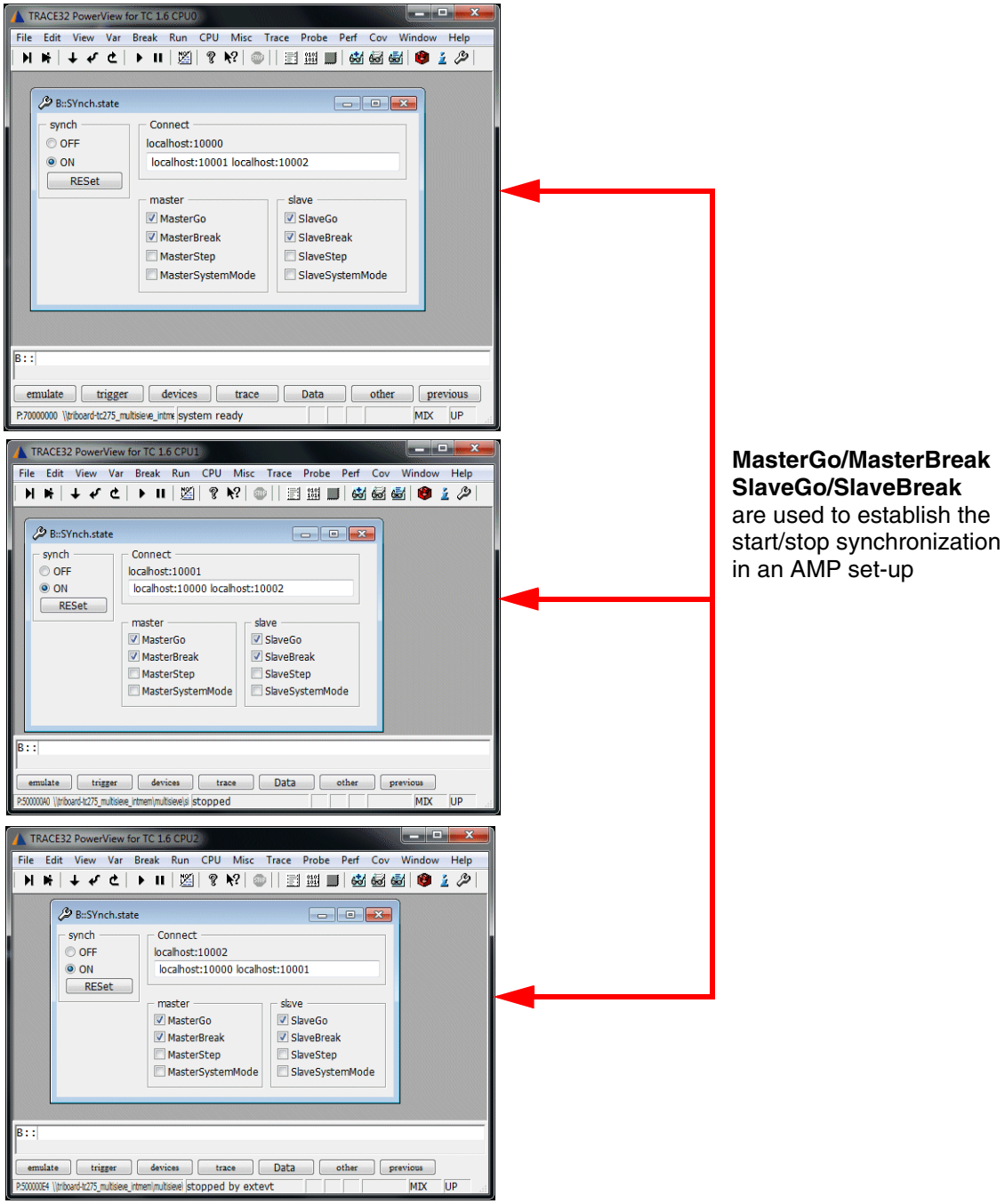


All windows opened with the ***/Track*** option follow the cursor movements in the active window

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

AMP - Correlate to a Trace Listing in another TRACE32 Instance

In an AMP configuration each TRACE32 instance displays the trace information for the core it controls. In order to analyze the interaction of the cores it is possible to establish a **Time Tracking** between the trace information in the different TRACE32 instances. Time Tracking between TRACE32 instances is established via the **SYnch.XTrack** command. If the start/stop synchronization for the cores is already established, establishing the **Time Tracking** is very simple.



The following additionally settings are required to establish a **Time Tracking** between two or more TRACE32 instances:

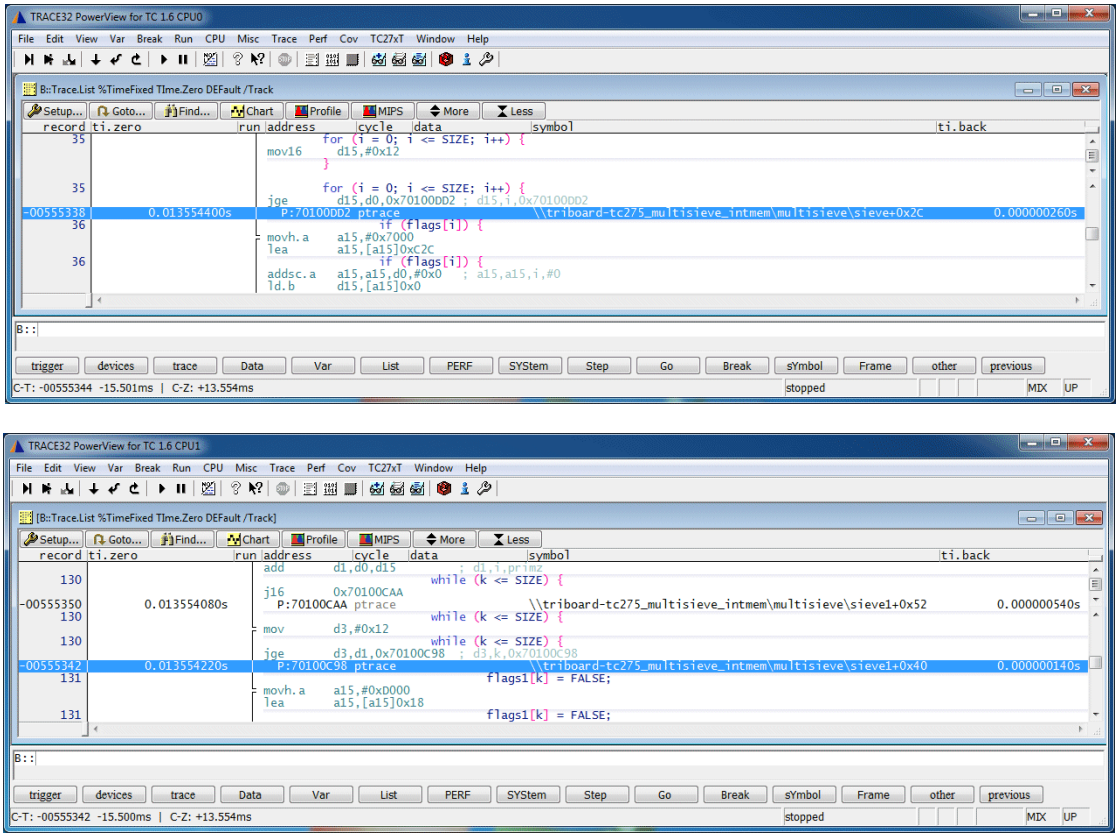
SYnch.XTrack {*<intercom_name>*}	Establish time synchronization to another TRACE32 instance

```
; in TRACE32 instance for TC 1.6 CPU0
SYnch.XTrack localhost:10001 localhost:10002

; in TRACE32 instance for TC 1.6 CPU1
SYnch.XTrack localhost:10000 localhost:10002

; in TRACE32 instance for TC 1.6 CPU2
SYnch.XTrack localhost:10000 localhost:10001
```

Tracking points are:

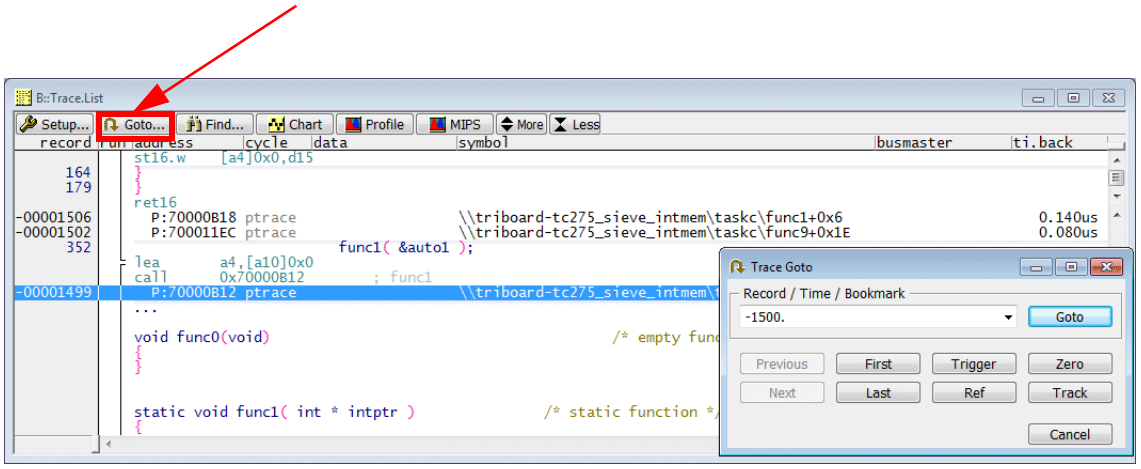
Off-chip trace **Timestamp Messages disabled**	TRACE32 global zero time
On-chip trace **Timestamp Messages disabled**	Record number
Timestamp Messages enabled	AURIX zero time: Time relative to the first record in the trace

```
Trace.List %TimeFixed TIme.ZERO DEFault /Track
```



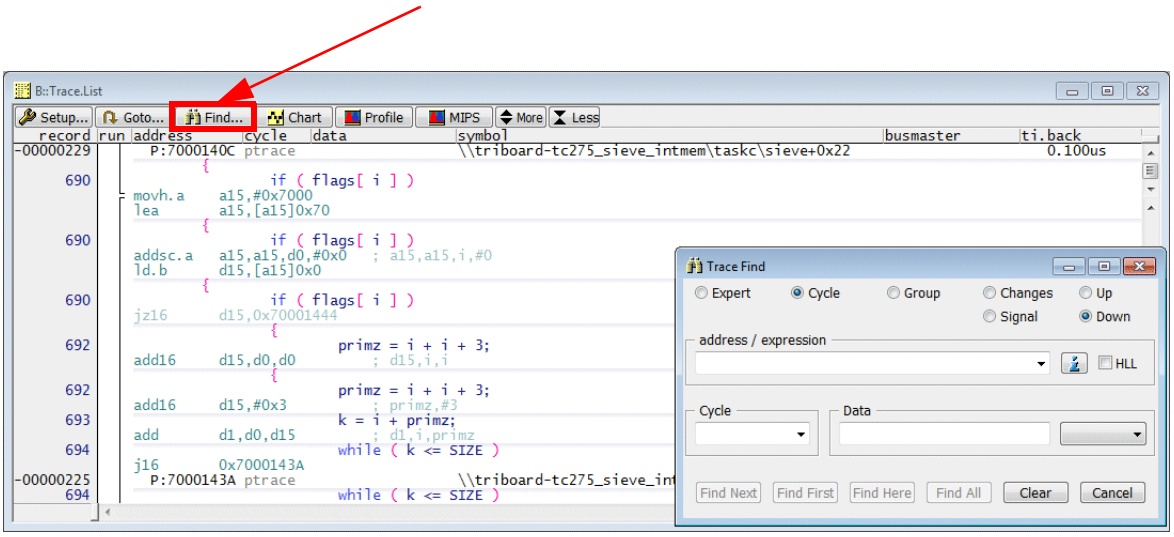
Time tracking between the Trace Listings of two TRACE32 instances

Browsing through the Trace Buffer

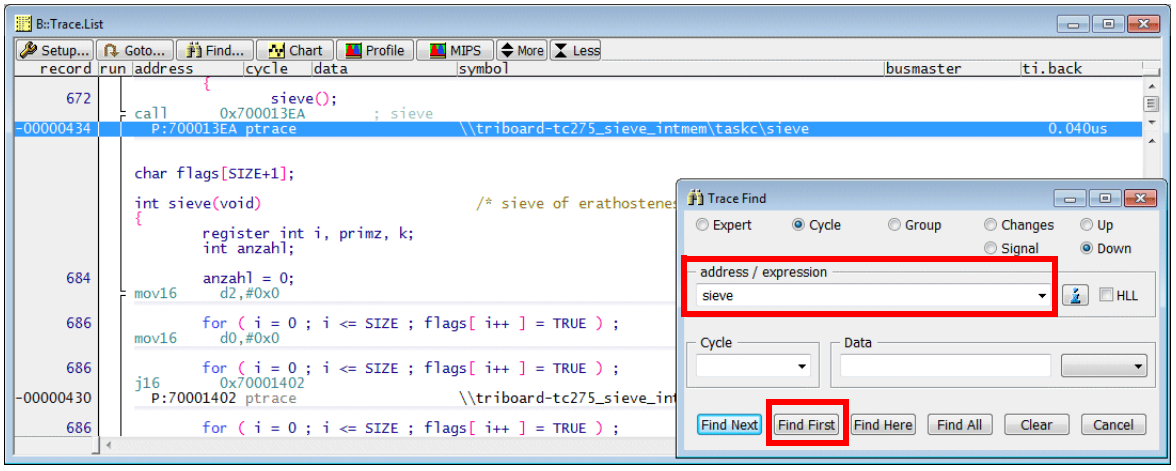


Pg ↑	Scroll page up.
Pg ↓	Scroll page down.
Ctrl - Pg ↑	Go to the first record sampled in the trace buffer.
Ctrl - Pg ↓	Go to the last record sampled in the trace buffer.

Find a Specific Event



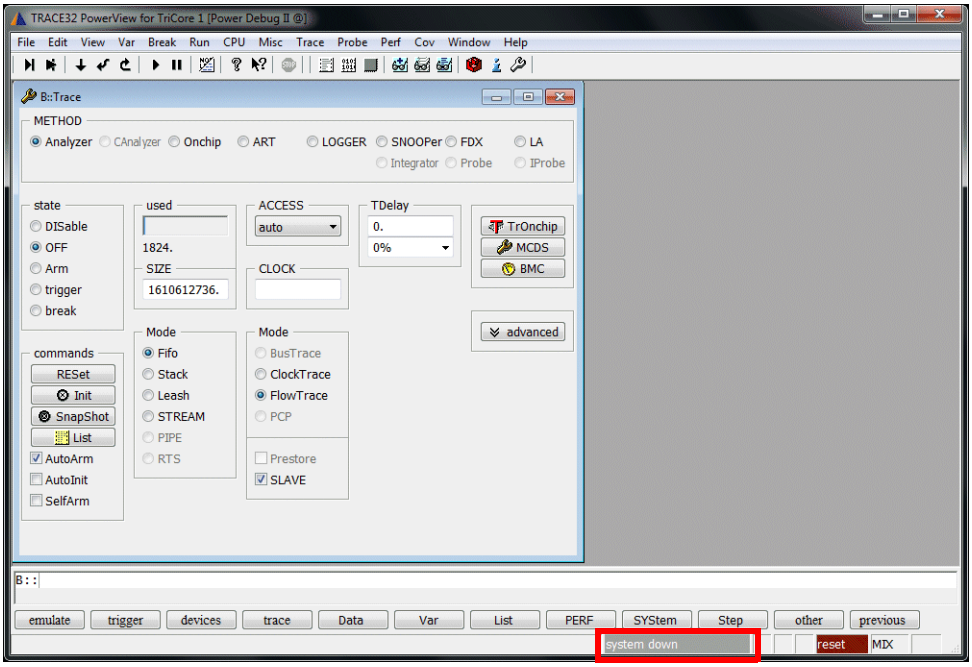
Example: Find a specific symbol address.



A more detail description on how to find specific events in the trace is given in **"Application Note for Trace.Find"** (app_trace_find.pdf).

Post Mortem Trace Analysis (PowerTrace only)

Trace decompression and display requires by default, that the program code is read from the target memory via JTAG/DAP. If the communication to the target is lost (system down) an alternative way to read the program code can be provided.



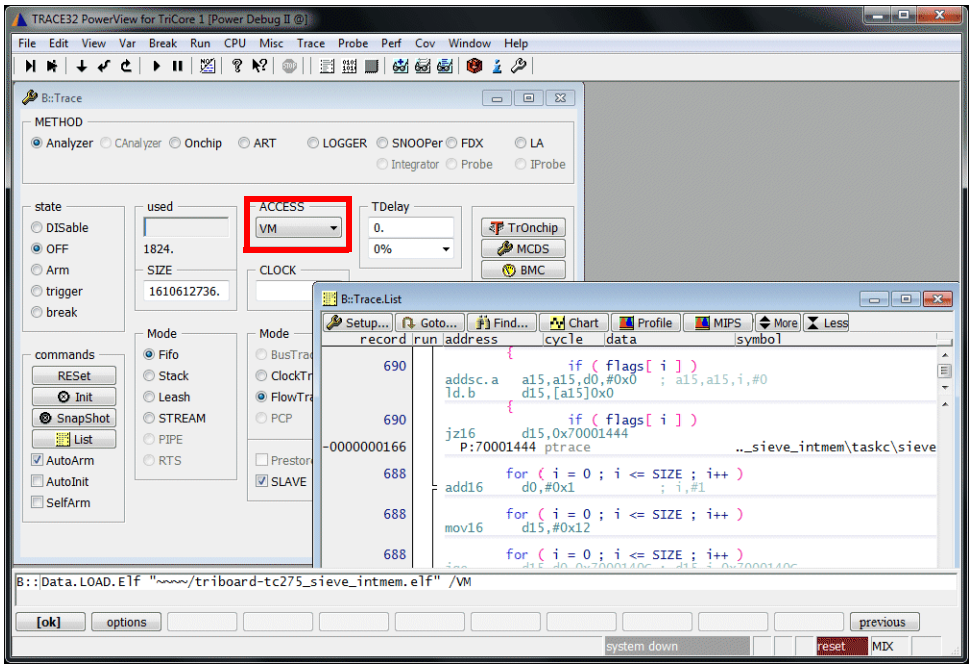
In order to decompress and display trace information after the communication to the target is lost proceed as follows:

1. Load the program code to the **TRACE32 Virtual Memory**.

```
Data.LOAD.Elf triboard-tc275_sieve_intmem.elf /VM
```

2. Advise TRACE32 to read program code from TRACE32 Virtual Memory.

```
Trace.ACCESS VM
```



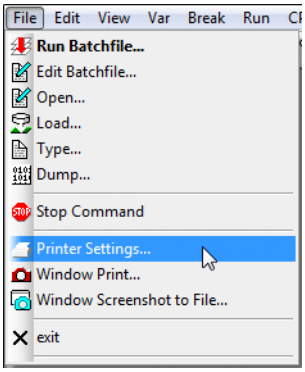
Belated Trace Analysis

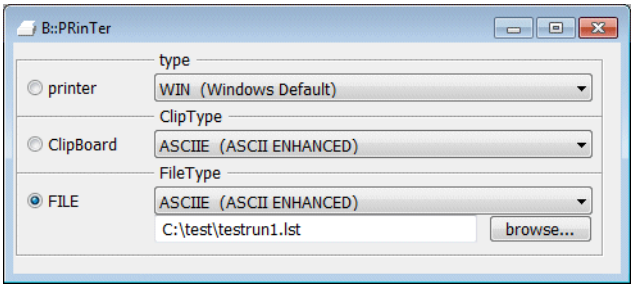
There are several ways for a belated trace analysis:

1. Save a part of the trace contents into an ASCII file and analyze this trace contents by reading.
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.

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

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






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

PRinTer.FILE testrun1.lst               ; specify the file name
```

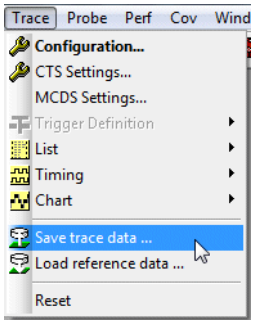
2. It only makes sense to save a part of the trace contents into an ASCII-file. Use the record numbers to specify the trace part you are interested in.

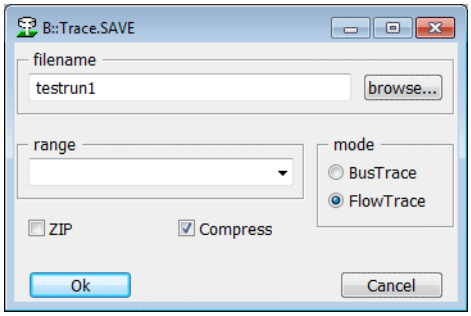
   TRACE32 provides the command prefix **WinPrint.** to redirect the result of a display command into a file.

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

3. Use an ASCII editor to display the result.

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

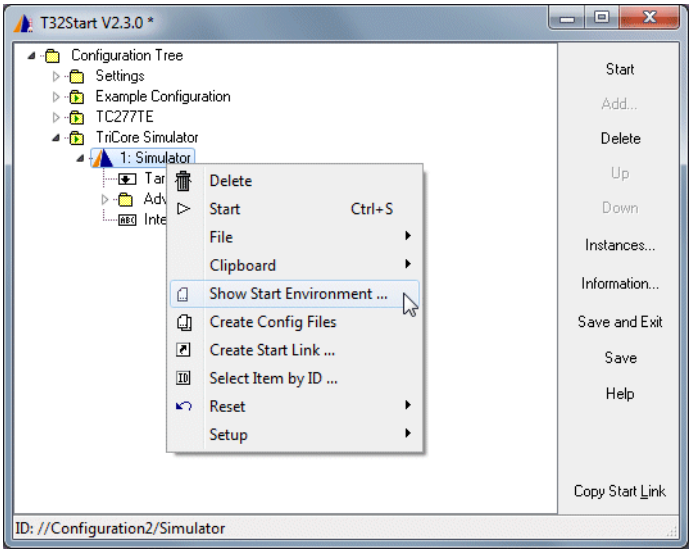


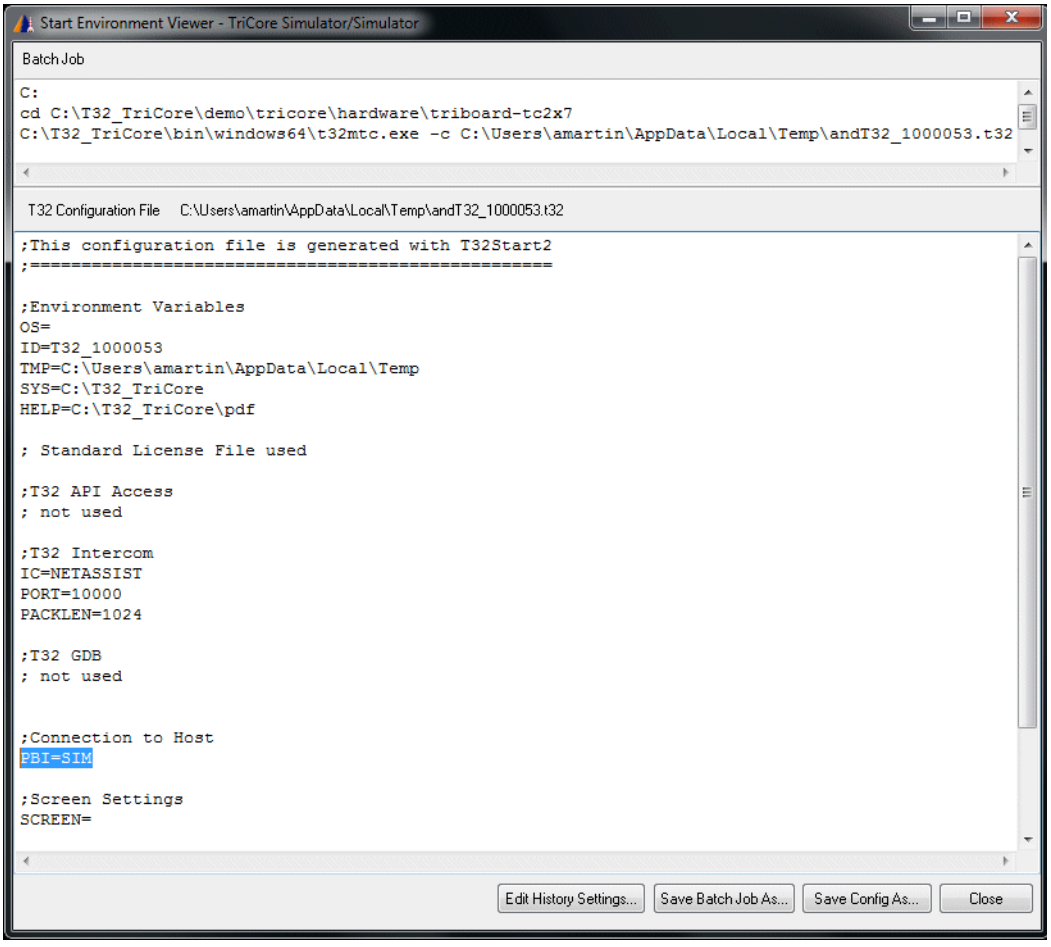


The default extension for the trace file is **.ad**.

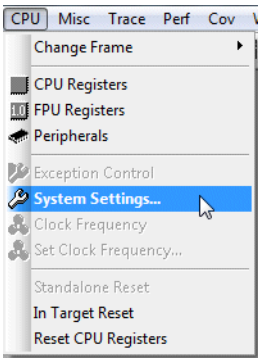
```
Trace.SAVE testrun1.ad
```

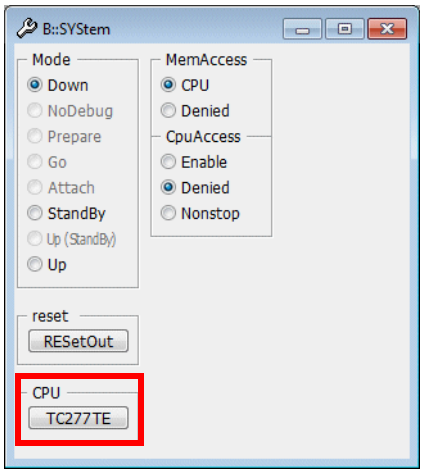
2. Start a TRACE32 Instruction Set Simulator (PBI=SIM).





3. **Select your target CPU within the simulator.**

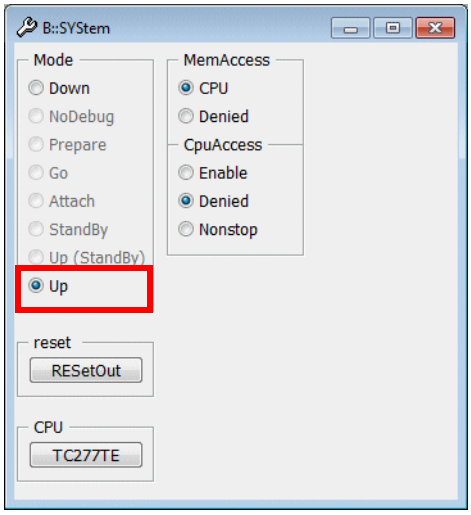




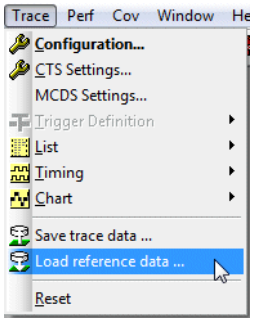
4. **If you are debugging an SMP system, inform the simulator which cores form the SMP system.**

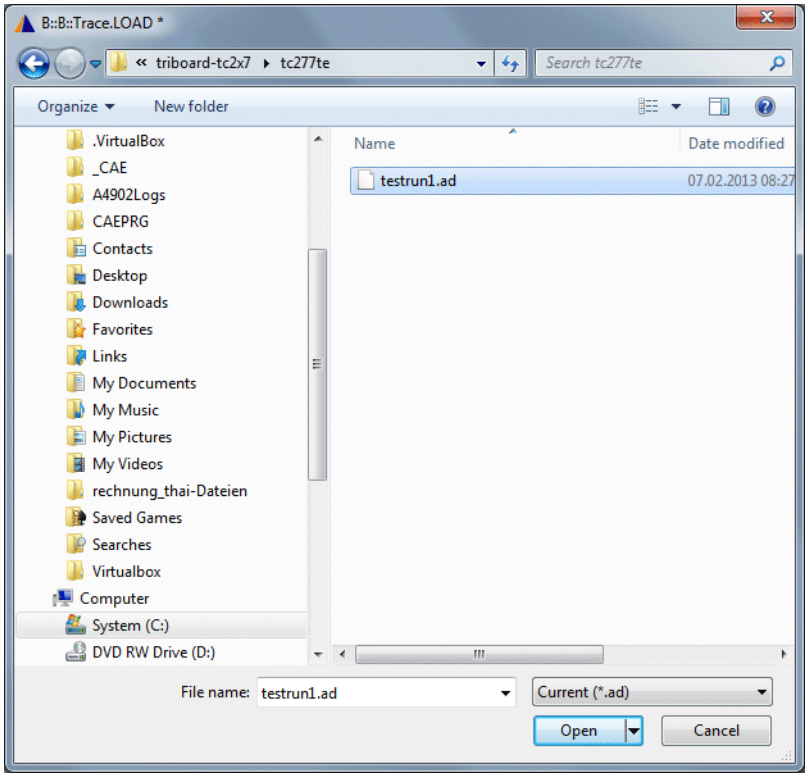
```
CORE.ASSIGN 1. 2. 3.
```

5. **Then establish the communication between TRACE32 and the simulator.**



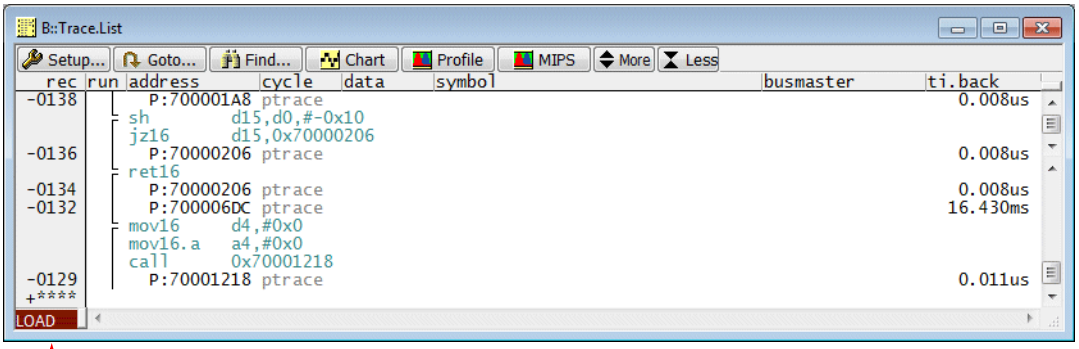
6. Load the trace file.






```
Trace.LOAD testrun1
```

7. Display the trace contents.



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

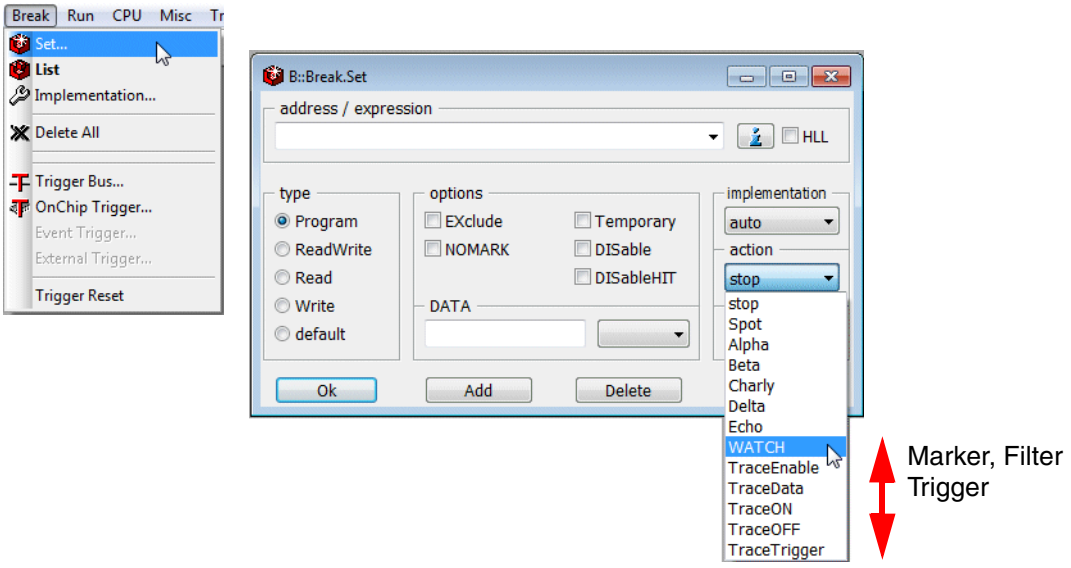
8. Load symbol and debug information if you need it.

```
Data.LOAD.Elf triboard-tc275_sieve_intmem.elf /NoCODE
```

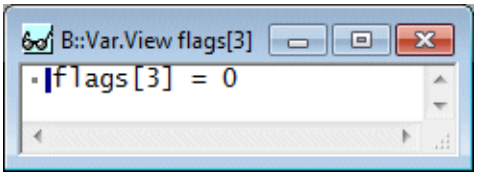
The TRACE32 Instruction Set Simulator provides the same trace display and analysis commands as the TRACE32 debugger.

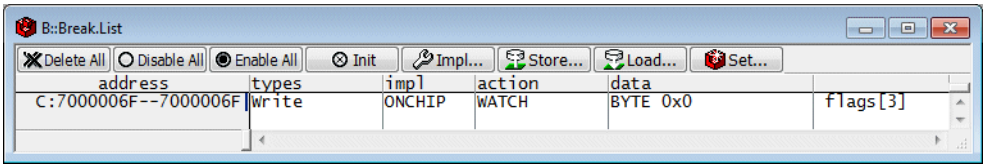
Trace Control by Filter and Trigger - Overview

The **action** field in the Break.Set dialog provides Marker, Filter and Trigger.



Marker, filter and trigger get a **blue breakpoint indicator** within TRACE32.





Marker

WATCH is a so-called marker. It is used to indicate the occurrence of an event in the trace display.

Filter

A Processor Observation Block is the hardware within MCDS that generates trace messages out of the activities of a core. Filters are used to advise the Processor Observation Block to reduce the generation of trace messages to the information of interest. Please be aware, that Filters have no effect on the Bus Observation Blocks (SPB/SRI).

Filters are **TraceEnable, TraceData, TraceOn** and **TraceOFF**.

Trigger

TraceTrigger is a so-called trigger. Triggers are used to advise MCDS to stop the generation of trace messages.

Available Resources

The MCDS provides complex qualification- and trigger mechanism. TRACE32 uses these mechanisms as effectively as possible. Due to the complexity of the qualification and trigger mechanisms it is not possible to provide detailed numbers for the available resources.

Fundamental behavior for AMP systems:

- Filters and Triggers are programmed for the core that is controlled by the TRACE32 instance.
- Filter advise the Processor Observation Block of the core controlled by the TRACE32 instance to generated the trace information of interest.
- Marker/Trigger advise the Processor Observation Block of the core controlled by the TRACE32 instance to indicate the occurrence of an event.

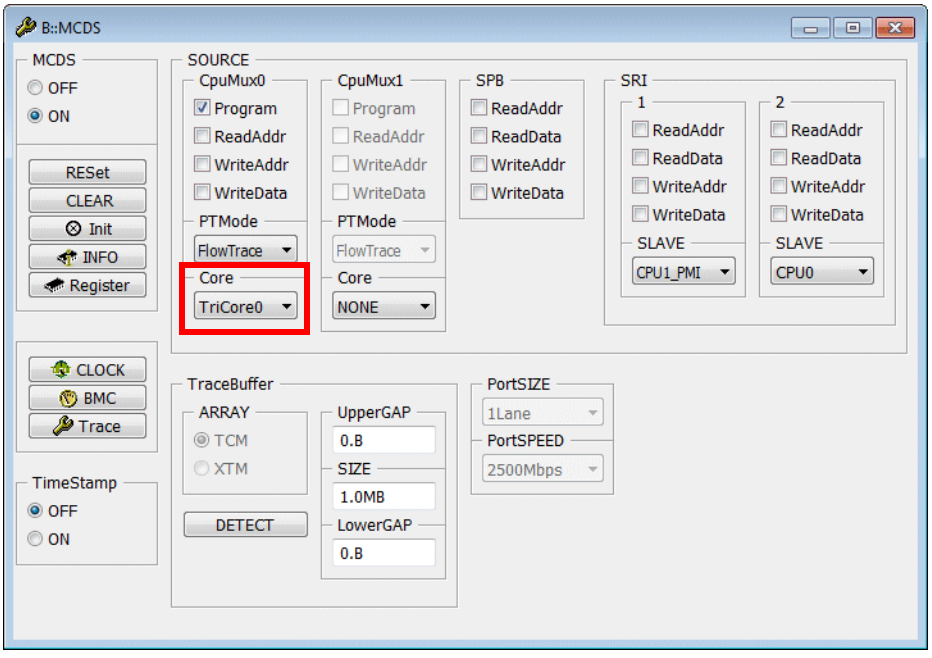
WATCH Marker

Advise Processor Observation Block to indicate the occurrence of an event.

Example: Indicate that 0x0 was written to the variable flags[3].

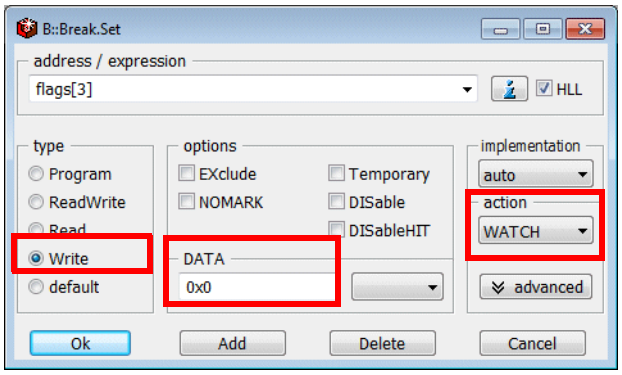
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write of 0x0 to variable flags[3].
- **Requested messages:** Instruction Pointer Call Messages, Timestamp Messages.

1. Configure the trace multiplexer.



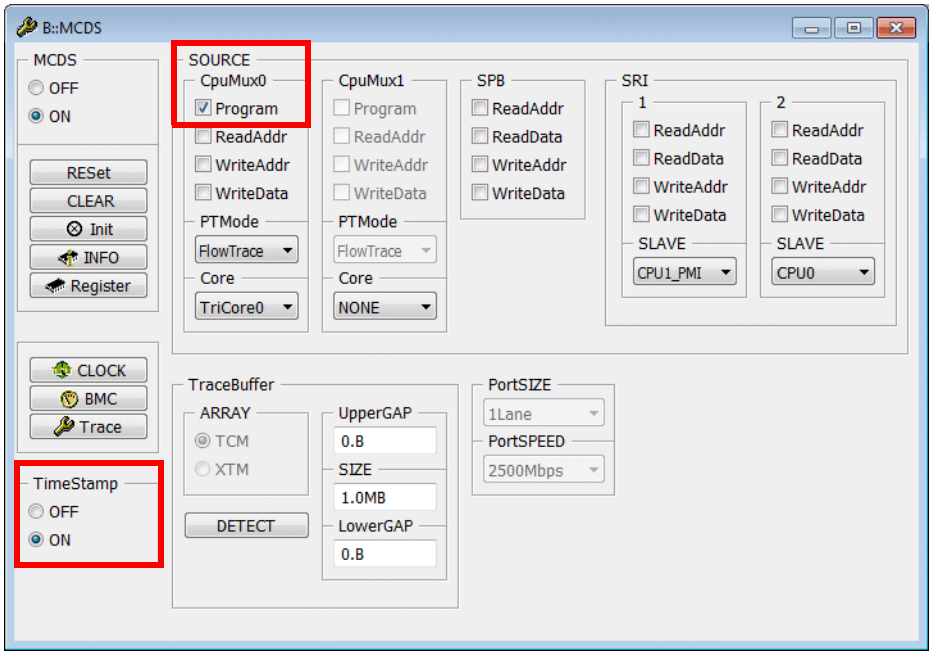

```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. Specify the event.




```
Var.Break.Set flags[3] /Write /DATA.Byte 0x0 /WATCH
```

3. Configure which trace messages are generated.

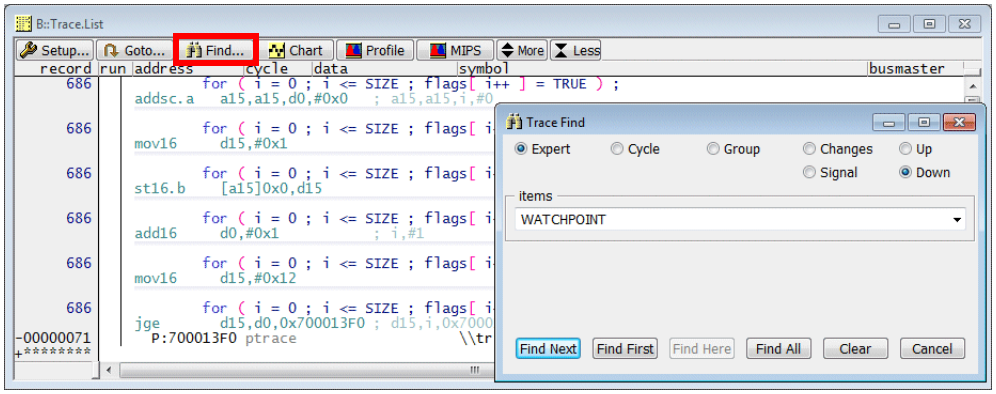


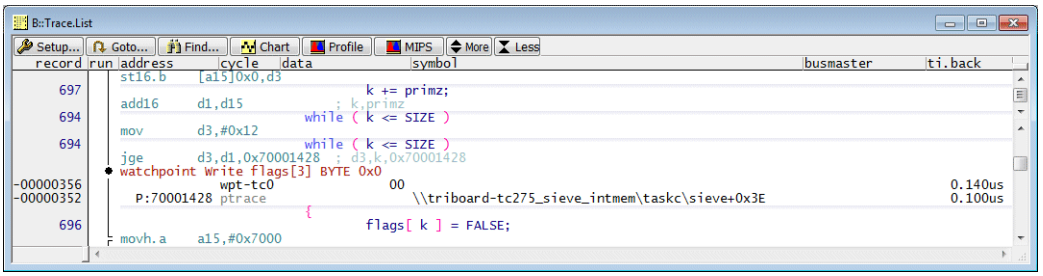

```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0
```

4. Start and stop the program execution.

5. Display the result.

It might be necessary to search for the result.



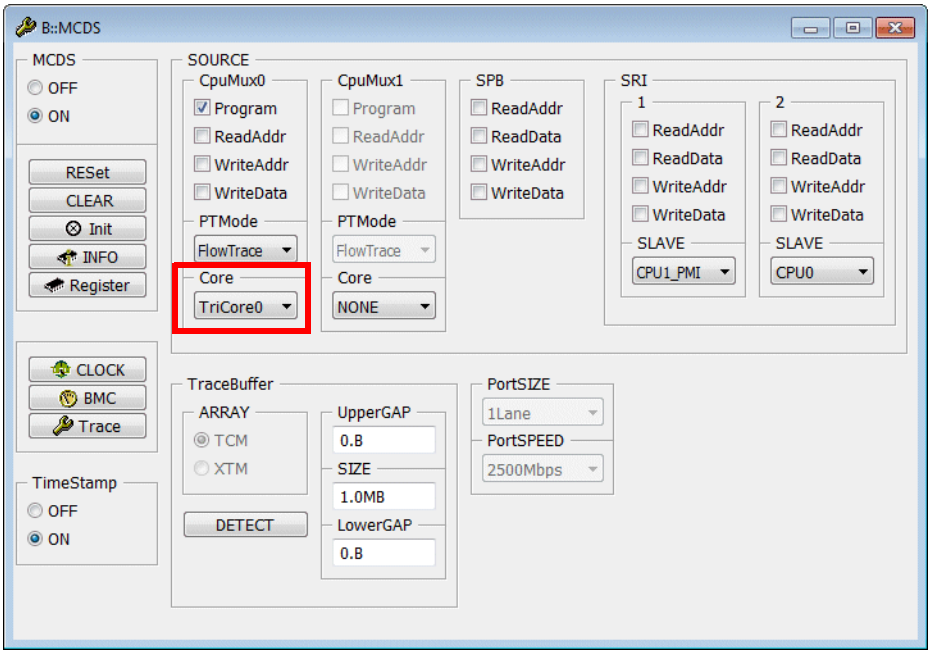


Advise the Processor Observation Block to generate trace messages for the enabled SOURCEs when the specified event is true.

Example 1: Restrict the generated trace information to the entries to the function sieve.

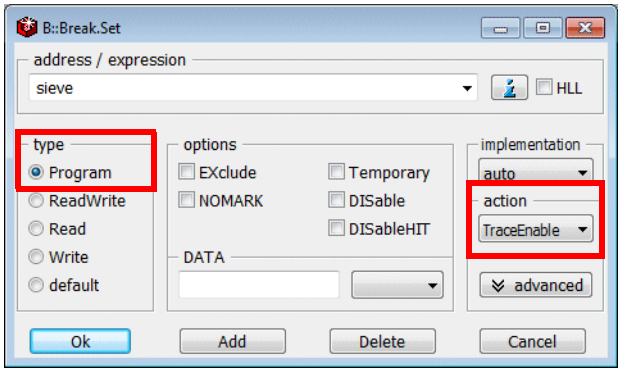
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Entry to function sieve.
- **Requested messages:** Instruction Pointer Call Messages, Timestamp Messages.

1. Configure the trace multiplexer.



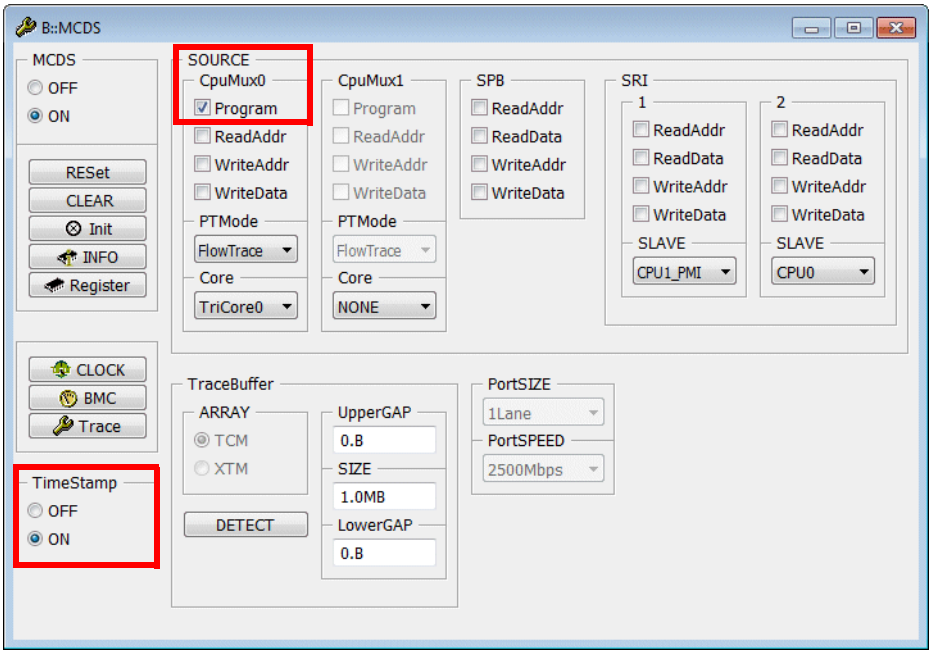

```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. Specify the event.




```
Break.Set   sieve /Program /TraceEnable
```

3. Configure which trace messages are generated while the event is true.

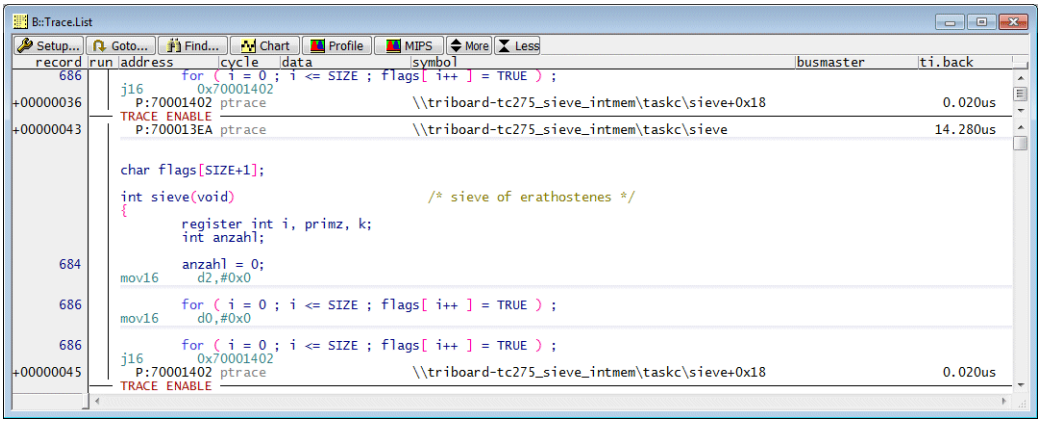



```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0
```

4. Start the program execution and stop it.

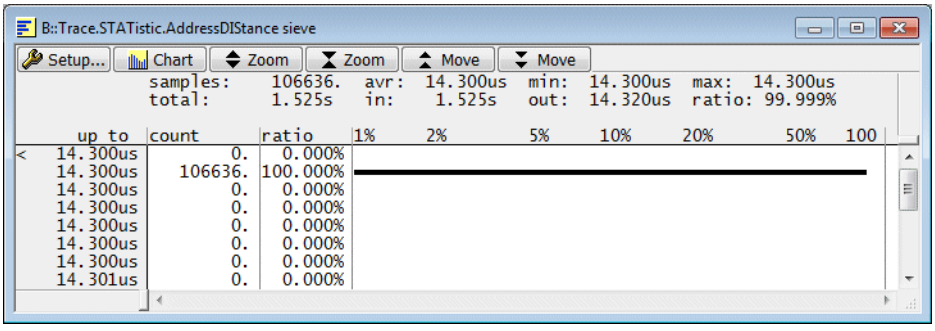
5. Display the result.

The trace contains only small code sections generated for the entries to the function sieve (TRACE ENABLE).



The following **Trace.STATistic** command calculates the time intervals for a program address event. The program address event is here the entry to the function sieve:

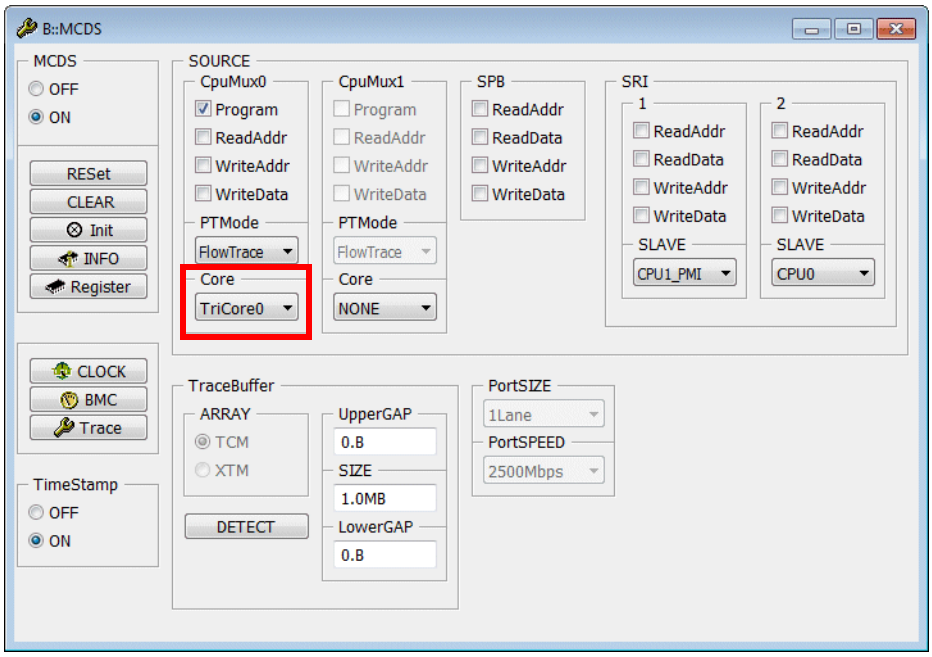
```
Trace.STATistic.AddressDIStance sieve
```



Example 2: Restrict the generated trace information to the entries to the function sieve and the exits from the function sieve.

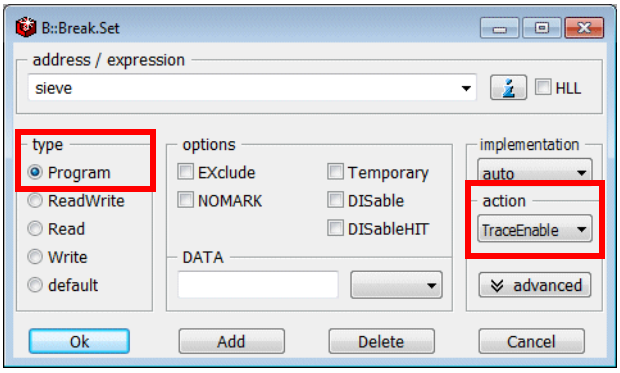
- **Core under debug:** TC 1.6.1 CPU0.
- **Events of interest:** Entry to function sieve and exit of function sieve
- **Requested messages:** Instruction Pointer Call Messages, Timestamp Messages.

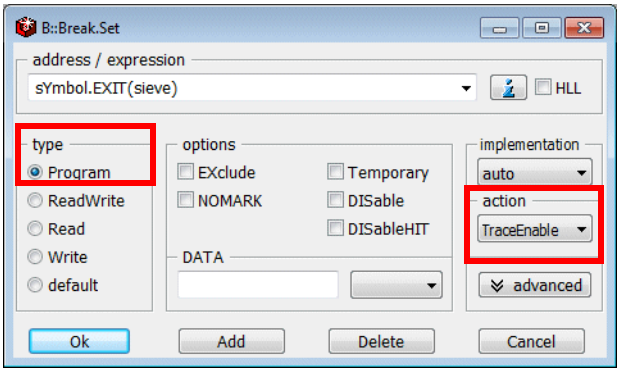
1. Configure the trace multiplexer.




```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. Specify the events.

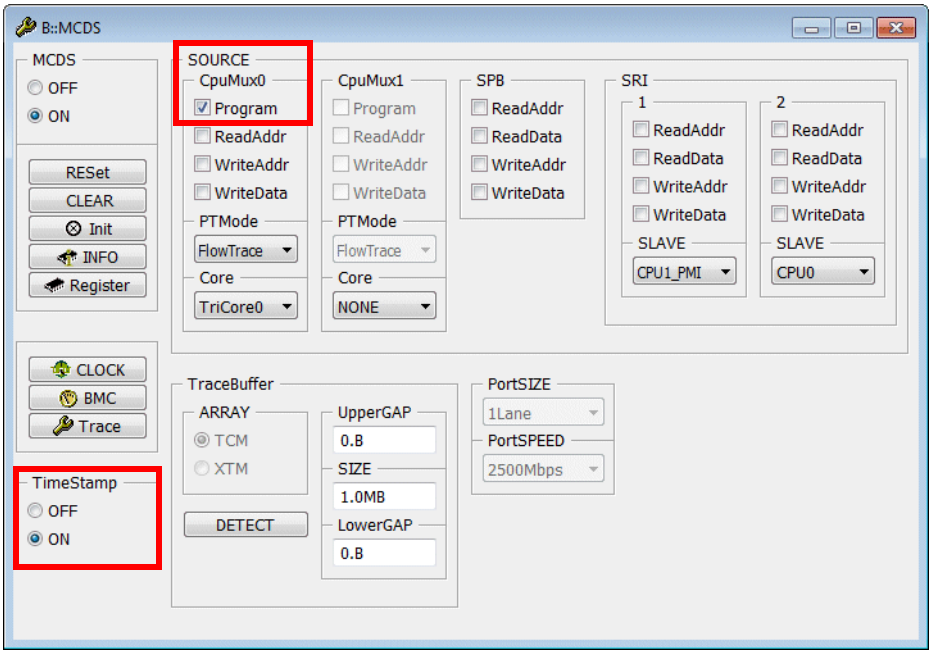




sYmbol.EXIT(<*symbol*>**)** Returns the exit address of the specified function

```
Break.Set   sieve /Program /TraceEnable

Break.Set   sYmbol.EXIT(sieve) /Program /TraceEnable
```

3. **Configure which trace messages are generated while the events are true.**

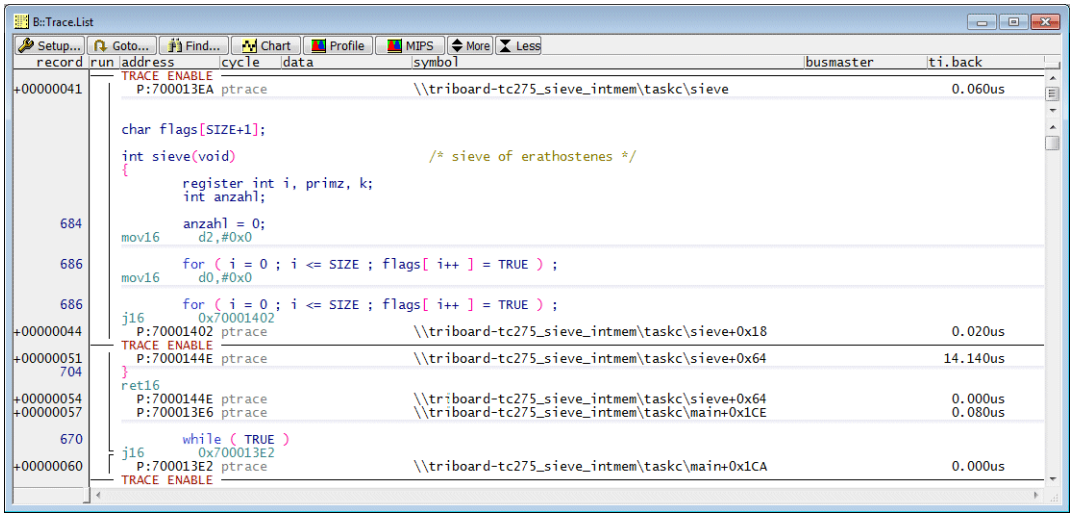



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON         ; enable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU0
```

4. **Start the program execution and stop it.**

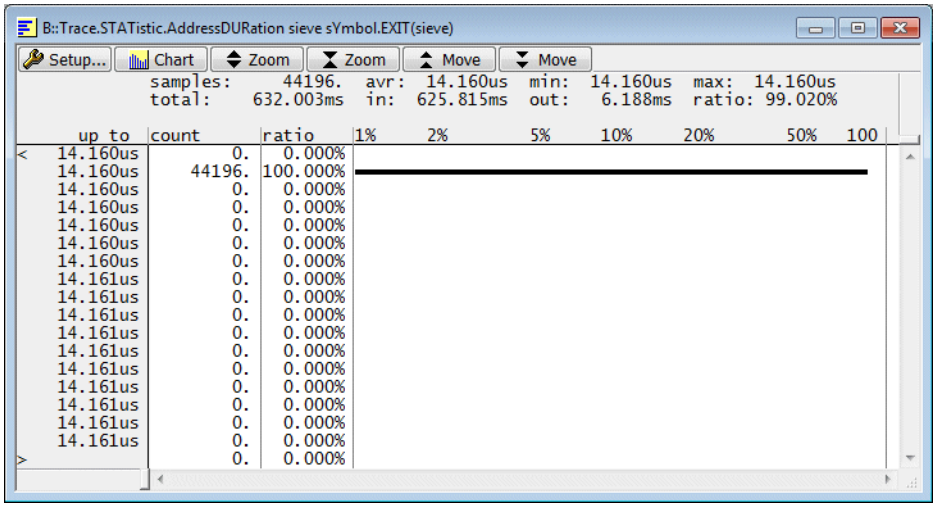
5. Display the result.

The trace contains only small code sections generated for the entries to the function sieve (TRACE ENABLE) and for the exits of the function sieve (TRACE ENABLE).



The following **Trace.STATistic** command calculates the time intervals between two program address events A and B. The entry to the function sieve is A in this example, the exit from the function is B.

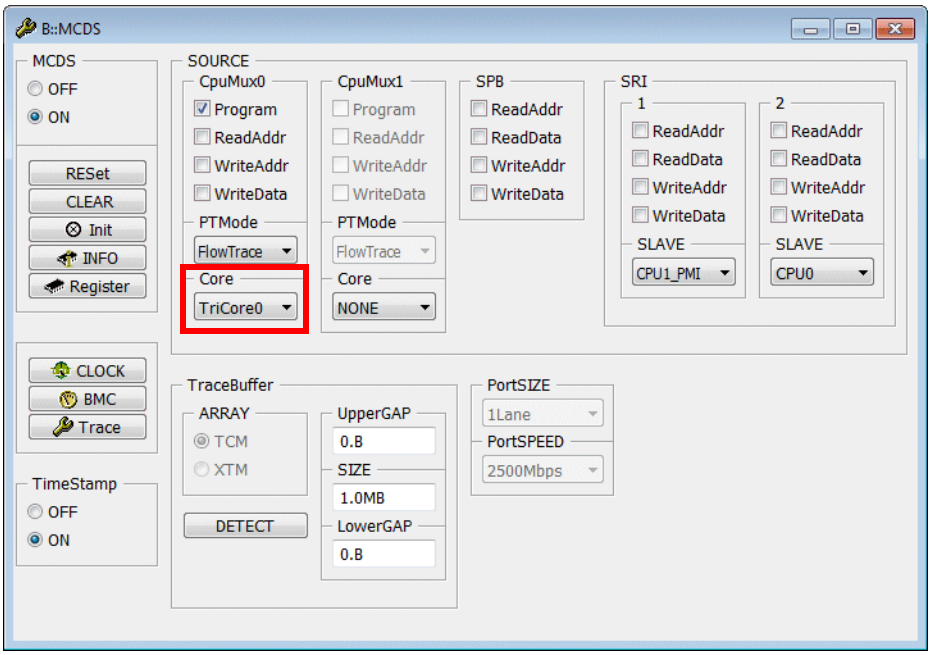
```
Trace.STATistic.AddressDURation sieve sYmbol.EXIT(sieve)
```



Example 3: Restrict the generated trace information to write accesses to the variable flags[3].

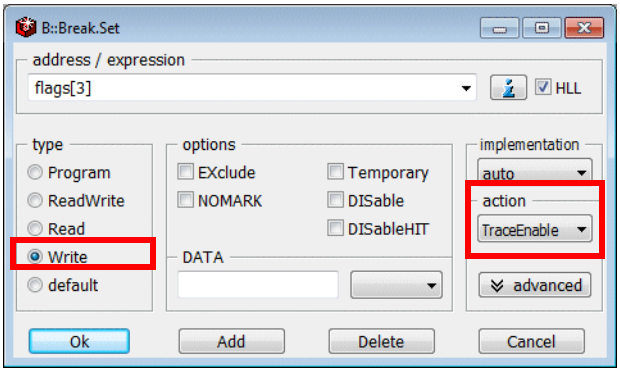
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write access to variable flags[3].
- **Requested messages:** Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



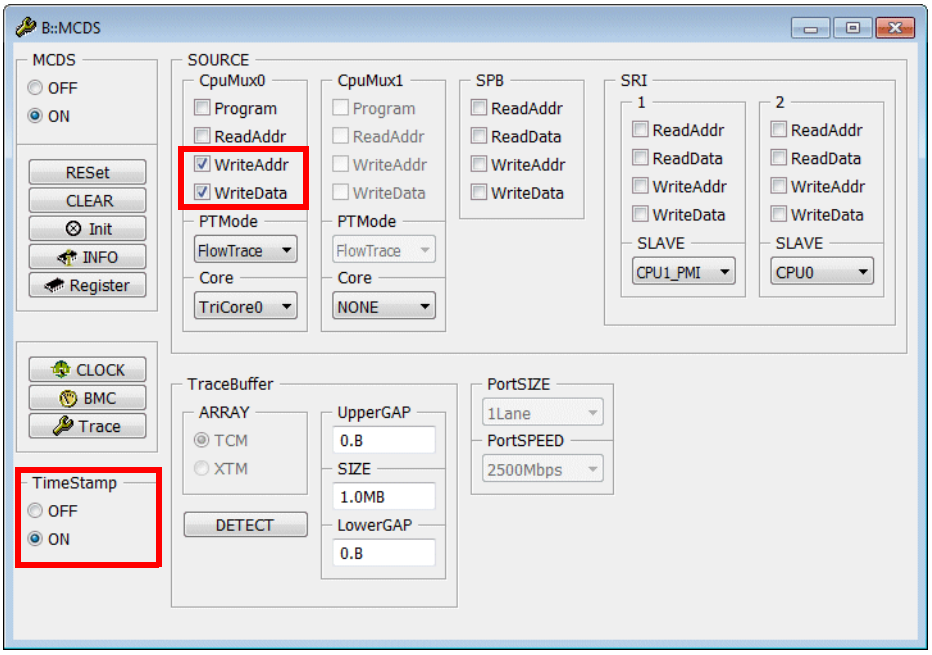

```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. Specify the events.




```
Var.Break.Set flags[3] /Write /TraceEnable
```

3. Configure which trace messages are generated while the event is true.

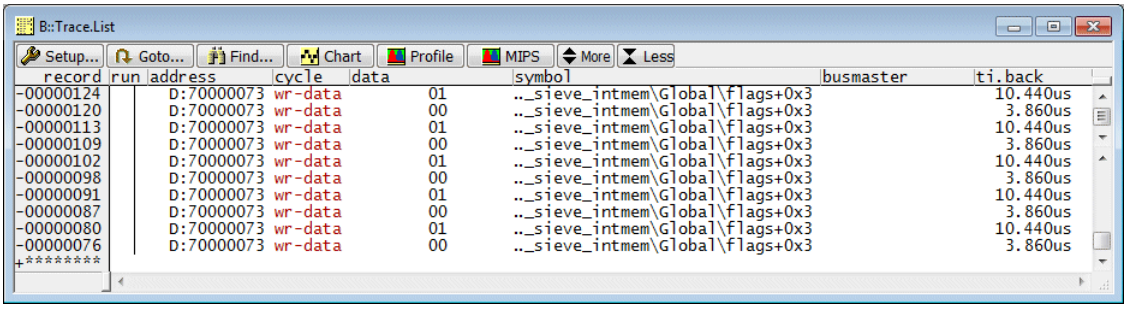



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON
```

4. Start the program execution and stop it.

5. Display the result.

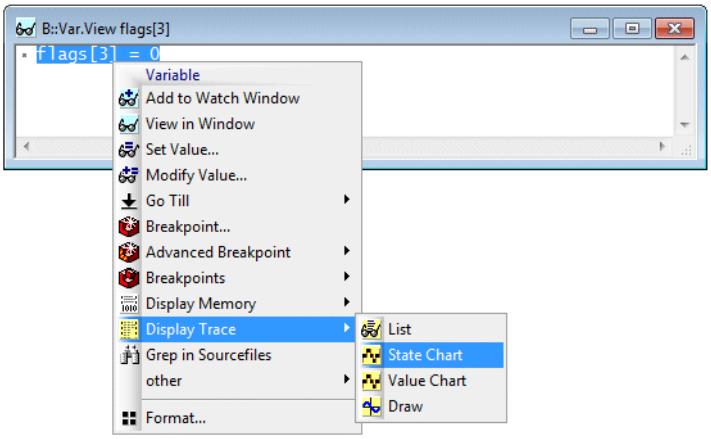
The trace contains only information on the write accesses to the variable flags[3].

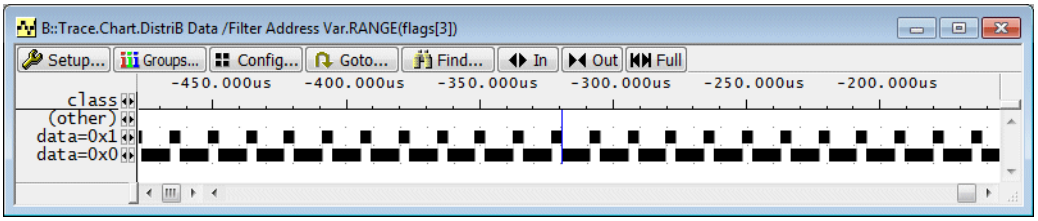


record	run	address	cycle	data	symbol	busmaster	ti.back
-00000124		D:70000073	wr-data	01	..._sieve_intmem\Global\flags+0x3		10.440us
-00000120		D:70000073	wr-data	00	..._sieve_intmem\Global\flags+0x3		3.860us
-00000113		D:70000073	wr-data	01	..._sieve_intmem\Global\flags+0x3		10.440us
-00000109		D:70000073	wr-data	00	..._sieve_intmem\Global\flags+0x3		3.860us
-00000102		D:70000073	wr-data	01	..._sieve_intmem\Global\flags+0x3		10.440us
-00000098		D:70000073	wr-data	00	..._sieve_intmem\Global\flags+0x3		3.860us
-00000091		D:70000073	wr-data	01	..._sieve_intmem\Global\flags+0x3		10.440us
-00000087		D:70000073	wr-data	00	..._sieve_intmem\Global\flags+0x3		3.860us
-00000080		D:70000073	wr-data	01	..._sieve_intmem\Global\flags+0x3		10.440us
-00000076		D:70000073	wr-data	00	..._sieve_intmem\Global\flags+0x3		3.860us
+********							

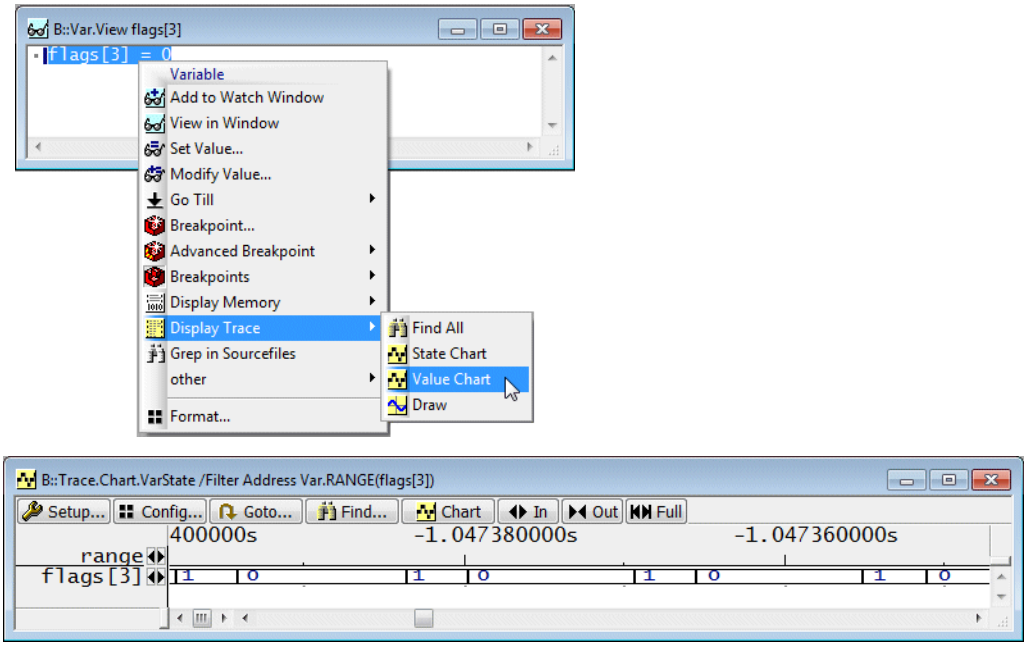
The Variable pull-down provides various way to analyze the variable contents over the time.

```
; open a window to display the variable
Var.View flags[3]
```



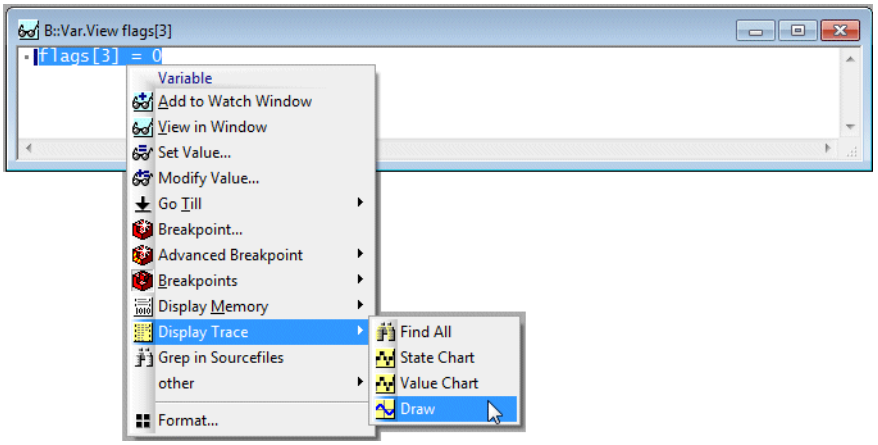


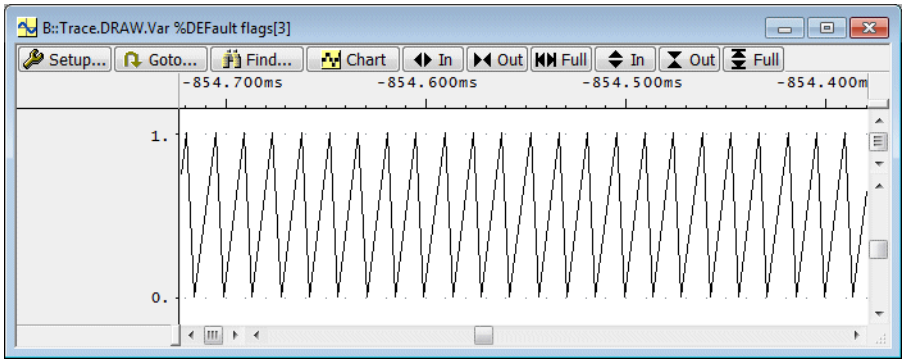
Display the value changes of a variable graphically
Trace.Chart.DistriB Data /Filter Address Var.RANGE(<*var*>**)**



Display variable contents over the time numerically
Trace.Chart.VarState /Filter Address Var.RANGE(<*var*>**)**

Var.RANGE(<*var*>**)**	Returns the address range in which the content of a variable is stored.





Display variable contents over the time graphically
Trace.DRAW.Var **%DEFault** *<var>*

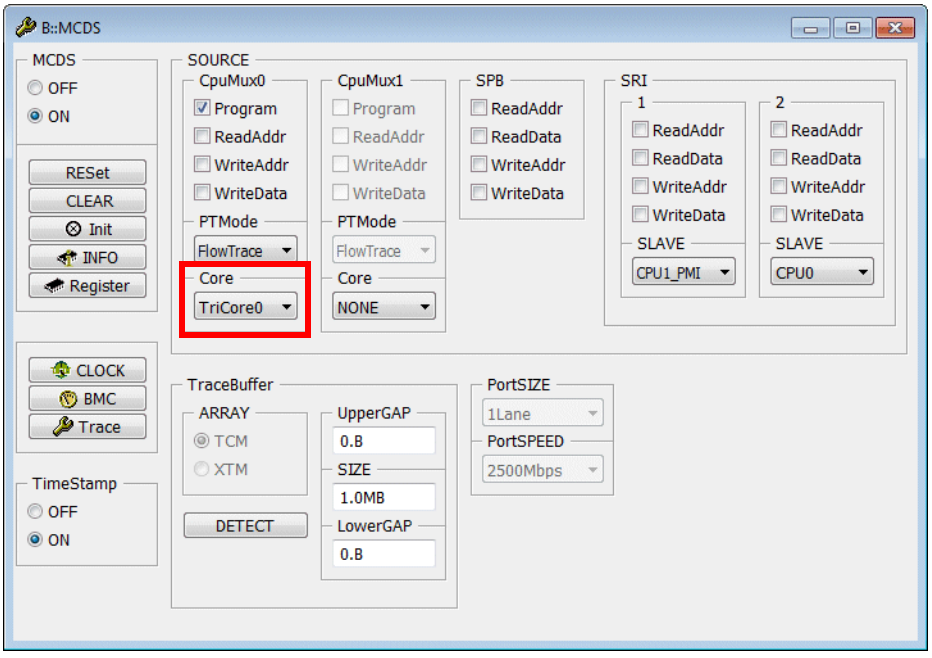
Advise the Processor Observation Block to generate trace messages for the instruction flow and for the specified events.

Motivation:	The TraceData filter is of great importance for the nesting function run-time analysis if an operating system is used.

Example: Generate trace information for the complete instruction flow and for all write accesses to flags[12].

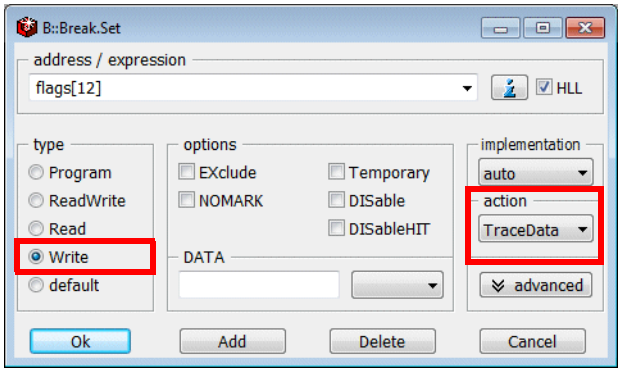
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write access to flags[12].
- **Requested Messages:** Timestamp Messages.

1. Configure the trace multiplexer.




```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. **Specify the event.**



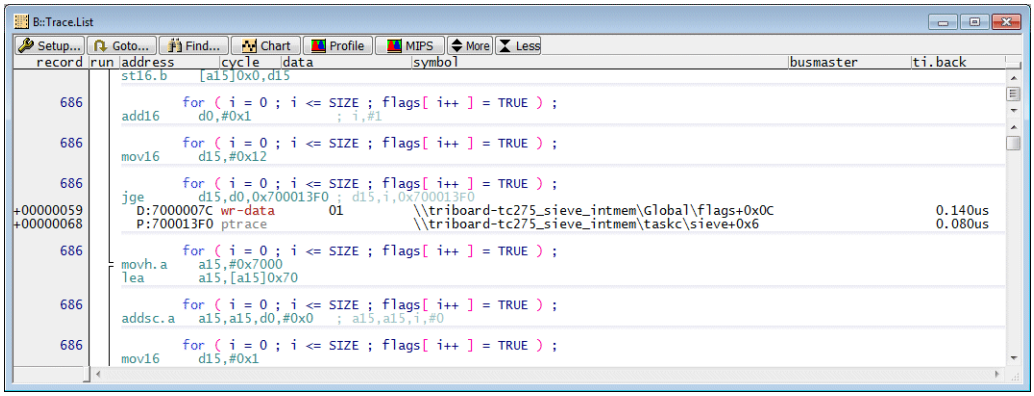

```
Var.Break.Set flags[12] /Write /TraceData
```

3. **TRACE32 PowerView takes care of the trace message generation.**

4. **Start the program execution and stop it.**

5. **Display the result.**

The trace contains the complete program flow and all write accesses to the variable flags[12].



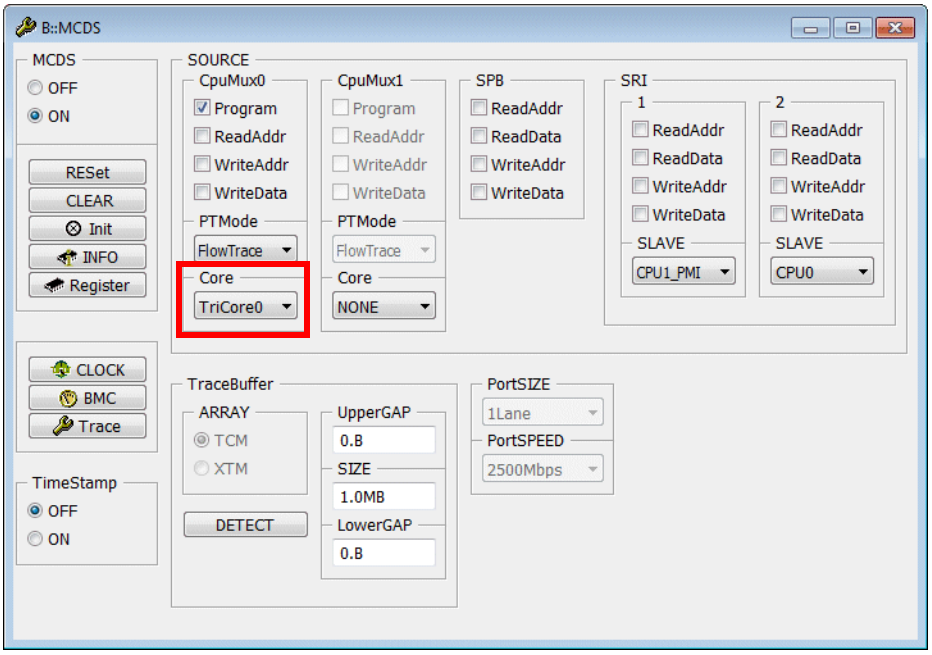
TraceON: Advise the Processor Observation Block to start the generation of trace messages for the enabled SOURCEs.

TraceOFF: Advise the Processor Observation Block to stop the generation of trace messages.

Example: Restrict the generation of trace messages to the function func2.

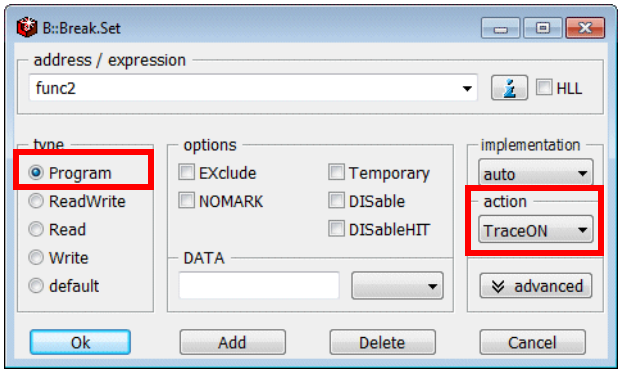
- **Core under debug:** TC 1.6.1 CPU0.
- **Events of interest:** Entry to function func2, exit of function func2
- **Requested Messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Read Data Trace Messages, Timestamp Messages.

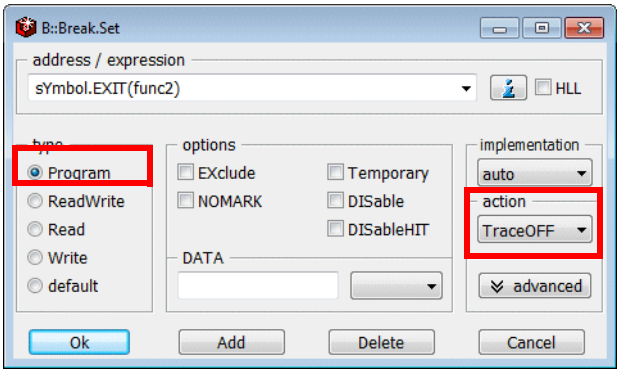
1. Configure the trace multiplexer.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source
```

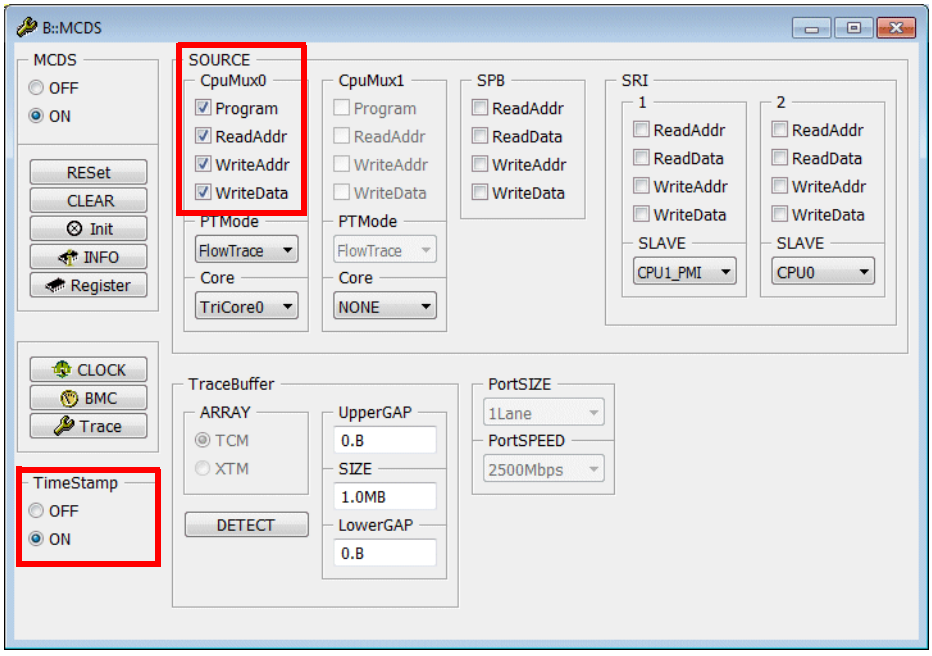
2. Specify the events.






```
Break.Set   func2 /Program /TraceON
Break.Set   sYmbol.EXIT(func2) /Program /TraceOFF
```

3. **Configure which trace messages are generated when the message generation is active (after TraceON event).**

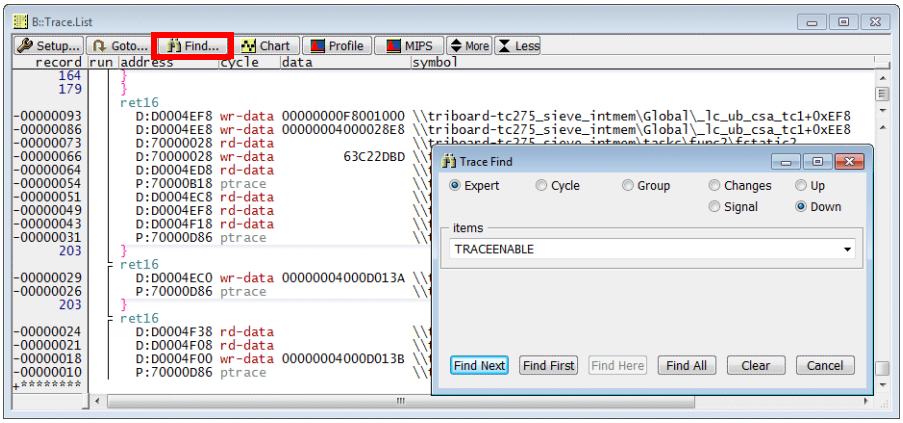


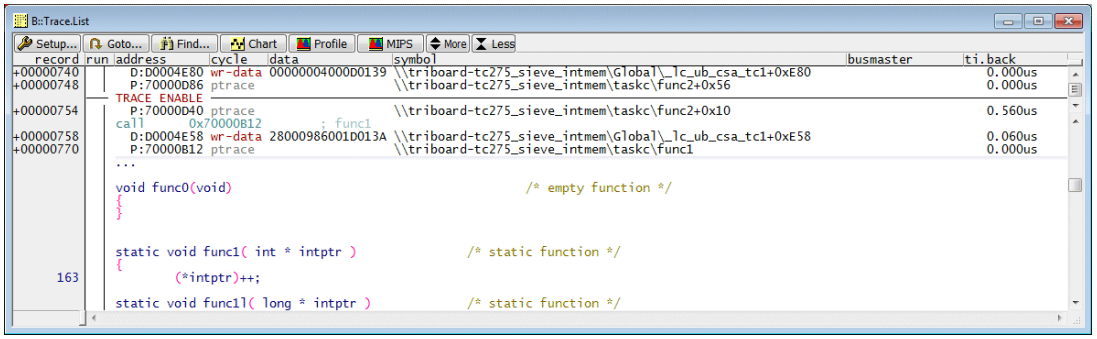

```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON         ; enable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.ReadAddr ON        ; enable Read Data Trace
                                           ; Messages for TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
MCDS.SOURCE.Set CpuMux0.WriteData ON       ; Messages for TC 1.6.1 CPU0
```

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

5. Display the result.

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





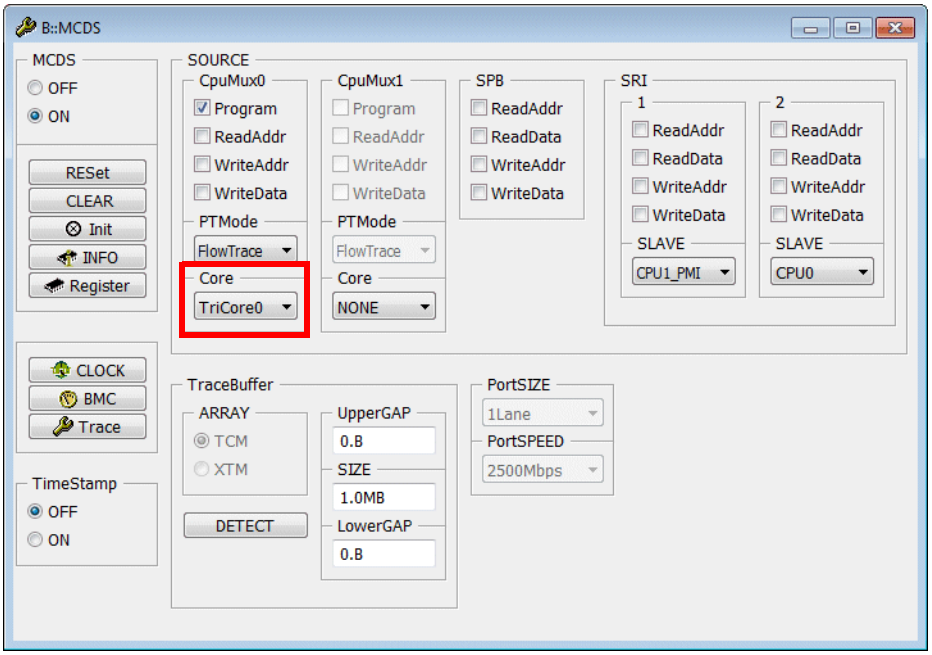
Trace message generation is started after the TraceON event occurred. As a result the event itself is not visible in the trace.

Advise the Processor Observation Block to end the message generation at the specified event.

Example 1: Stop the trace recording when 0x0 was written to the variable flags[3].

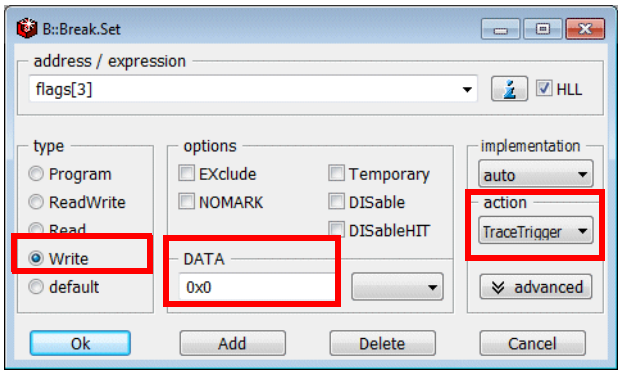
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write of 0x0 to variable flags[3].
- **Requested Messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



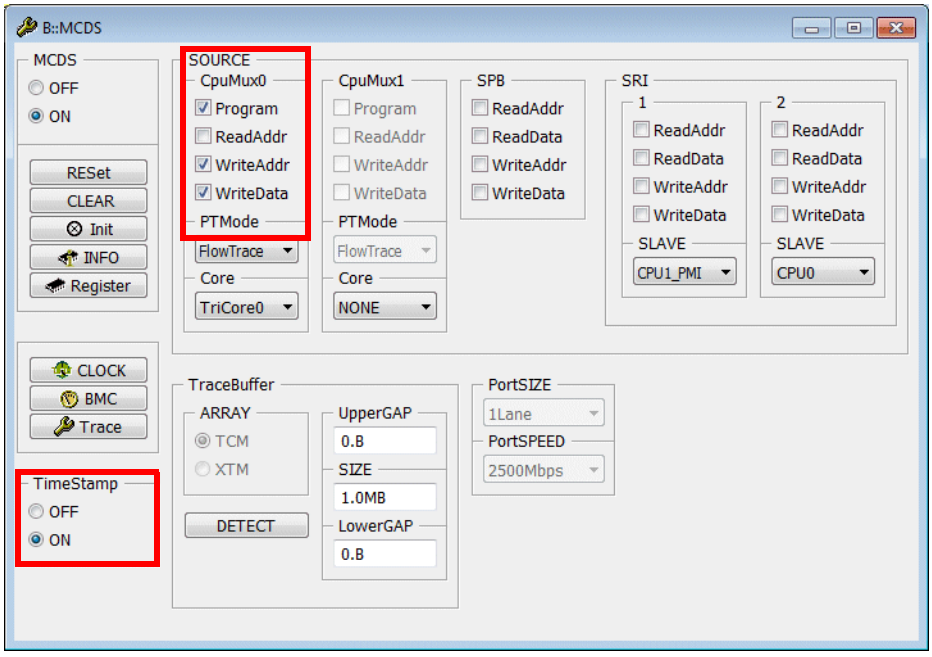

```
; enable TC 1.6.1 CPU0 as trace source
MCDS.SOURCE.Set CpuMux0.Core TriCore0
```

2. Specify the event.



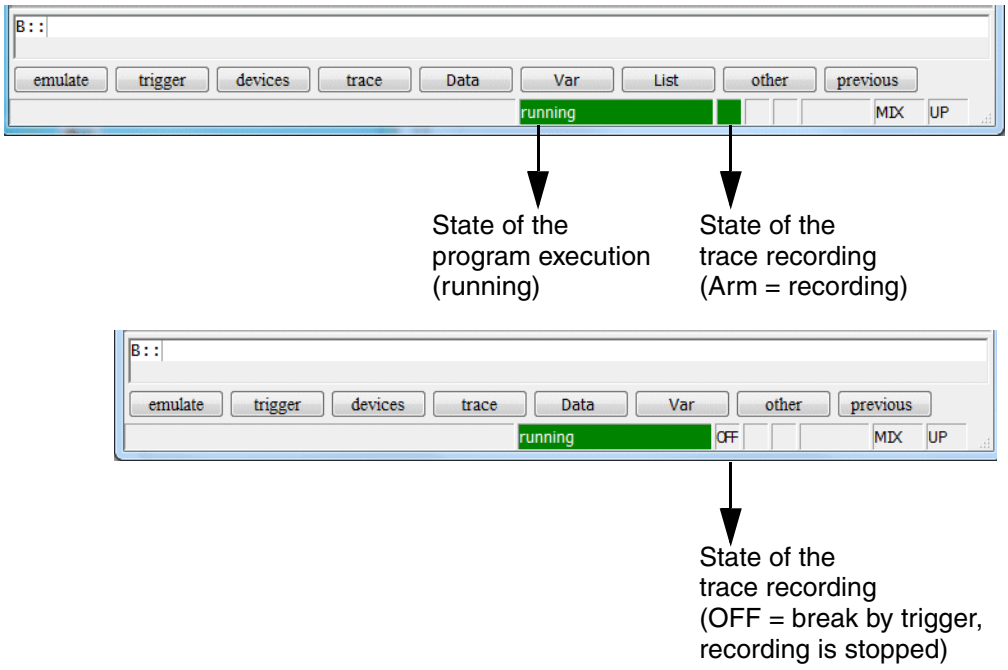

```
Var.Break.Set flags[3] /Write /DATA.Byte 0x0 /TraceTrigger
```

3. Configure which trace messages are generated until the trigger event occurs.



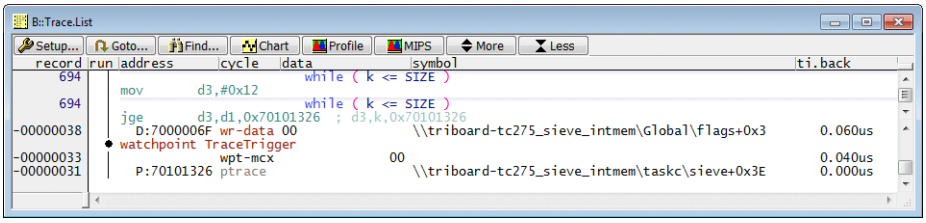

```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON         ; enable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
MCDS.SOURCE.Set CpuMux0.WriteData ON       ; Messages for TC 1.6.1 CPU0
```

4. Start the program execution.



5. Display the result.

MCDS ends the generation of trace messages and flushes all internal buffer when the specified event occurs. TRACE32 automatically generates a **watchpoint TraceTrigger** message when the trigger event occurs. This helps you to find the actual trigger event in the trace.

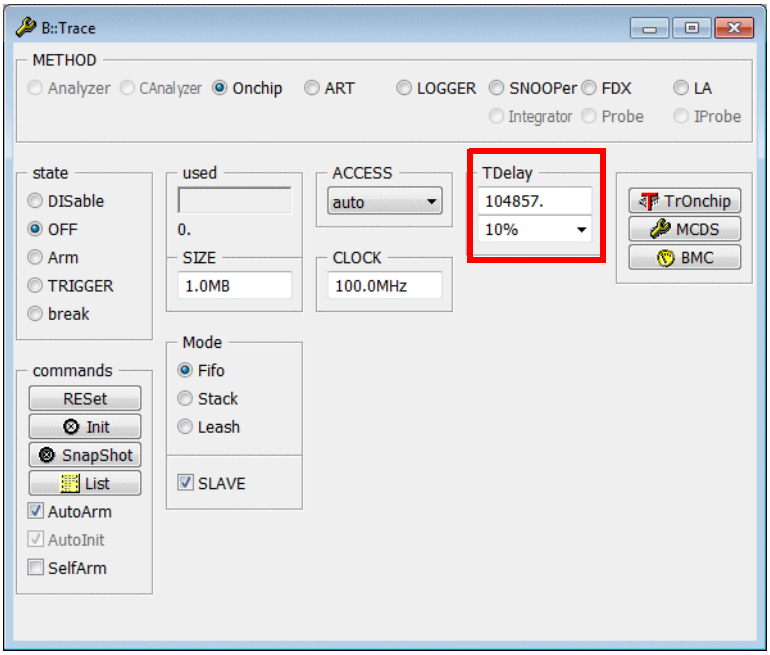


Example 2: Stop the trace recording after another 10% of the trace memory was filled when 0x0 was written to the variable flags[3].

- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write of 0x0 to variable flags[3].
- **Requested Messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Timestamp Messages.

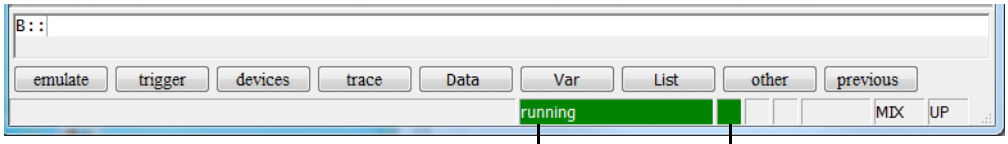
1. to 3. as in example 1.

4. Specific the delay.



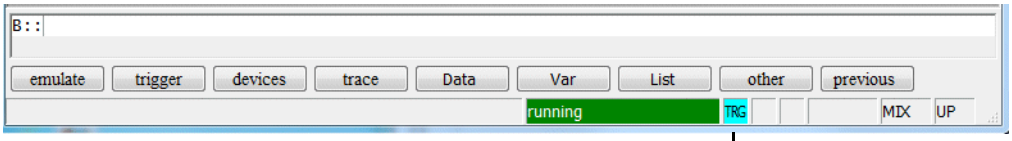

```
Trace.TDelay 10%
```

5. Start the program execution.

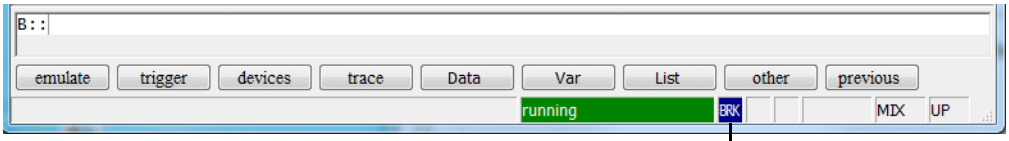


State of the program execution (running)

State of the trace recording (Arm = recording)

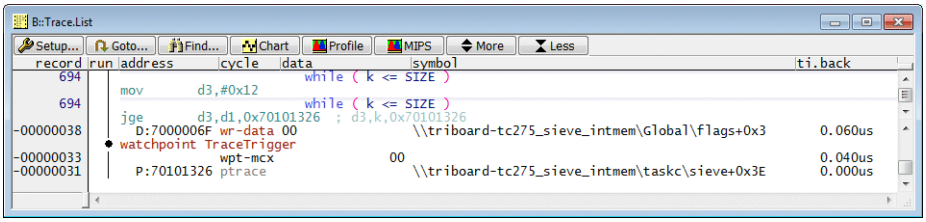


State of the trace recording (TRG = trigger occurred, delay counter started)



State of the trace recording (BRK = delay counter elapsed, recording is stopped)

6. Display the result.



Filter and Trigger - SMP Systems

Fundamental behavior for SMP systems:

- Filters and Triggers are programmed for all cores that are connected to the trace multiplexer.
- Filters advise the Processor Observation Blocks of the connected cores to generated the trace information of interest.
- Marker/Trigger advise the Processor Observation Blocks of the connected cores to indicate the occurrence of an event.

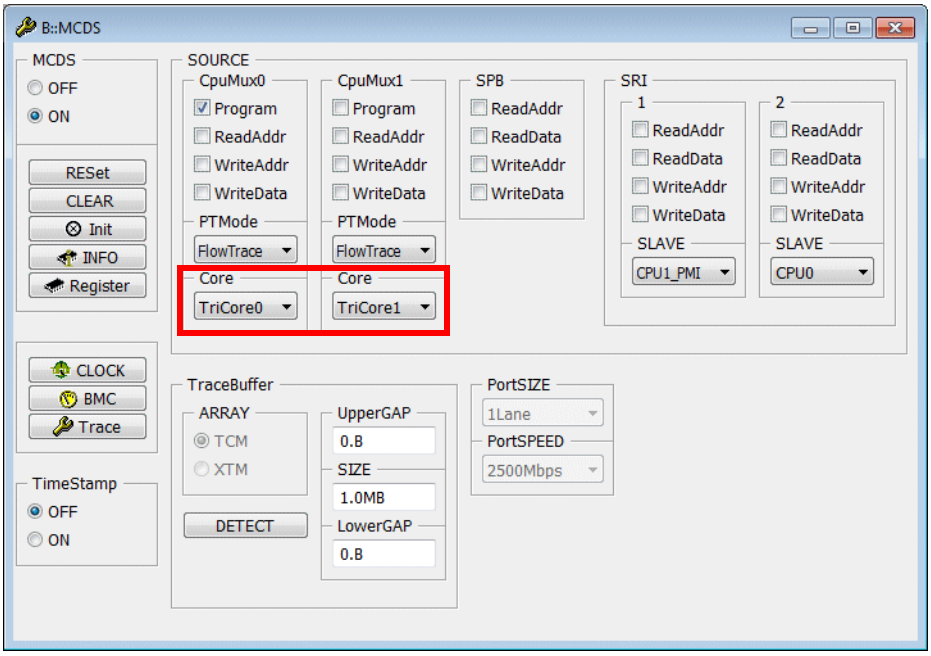
WATCH Marker

Advise Processor Observation Blocks of all cores to indicate the occurrence of an event.

Example: Indicate that 0x0 was written to the variable flags[3].

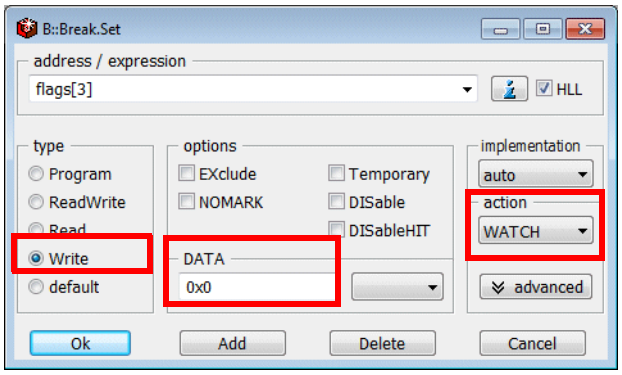
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write of 0x0 to variable flags[3]
- **Requested trace messages:** Instruction Pointer Call Messages for both cores, Timestamp Messages.

1. **Configure the trace multiplexer.**



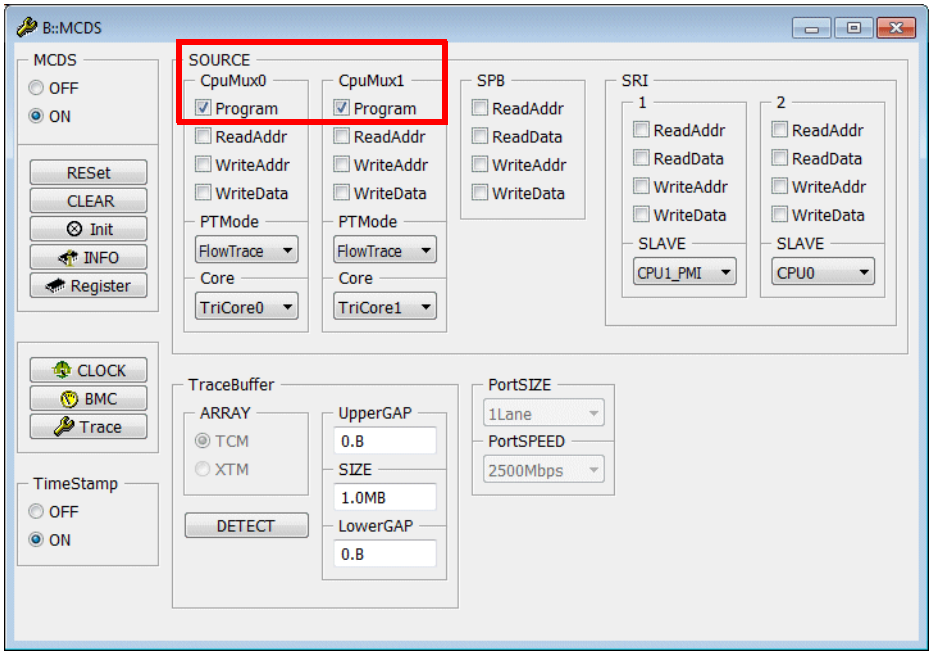

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the event.




```
Var.Break.Set flags[3] /Write /DATA.Byte 0x0 /WATCH
```

3. Configure which trace messages are generated.

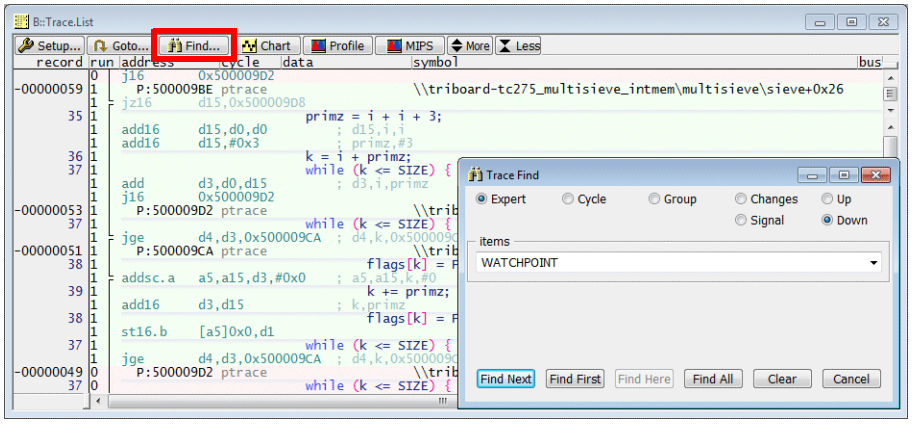


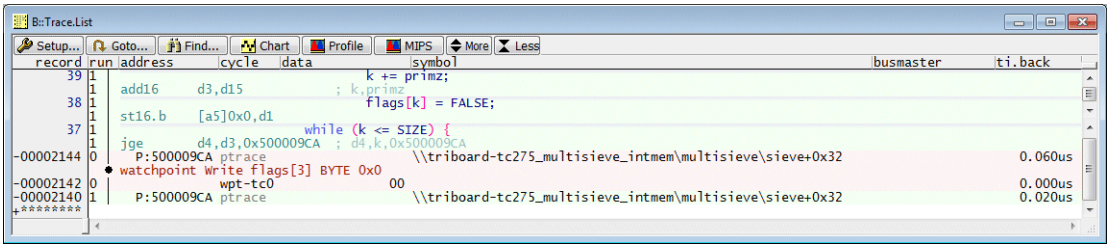

```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU1
```

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

5. **Display the result.**

It might be necessary to search for the result.

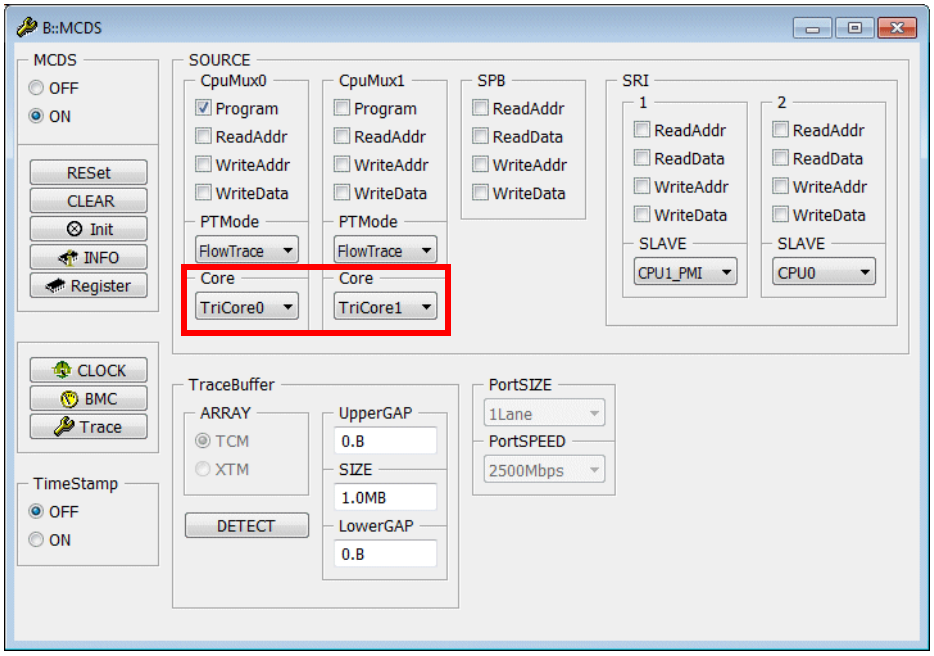




Advise the Processor Observation Block to generate trace messages for the enabled SOURCEs when the specified event is true.

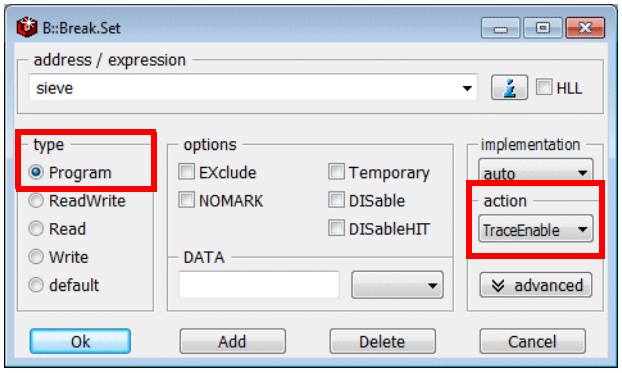
Example 1: Restrict the generated trace information to the entries to the function sieve.

- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Entry to function sieve.
- **Requested trace messages:** Instruction Pointer Call Messages for both cores, Timestamp Messages.



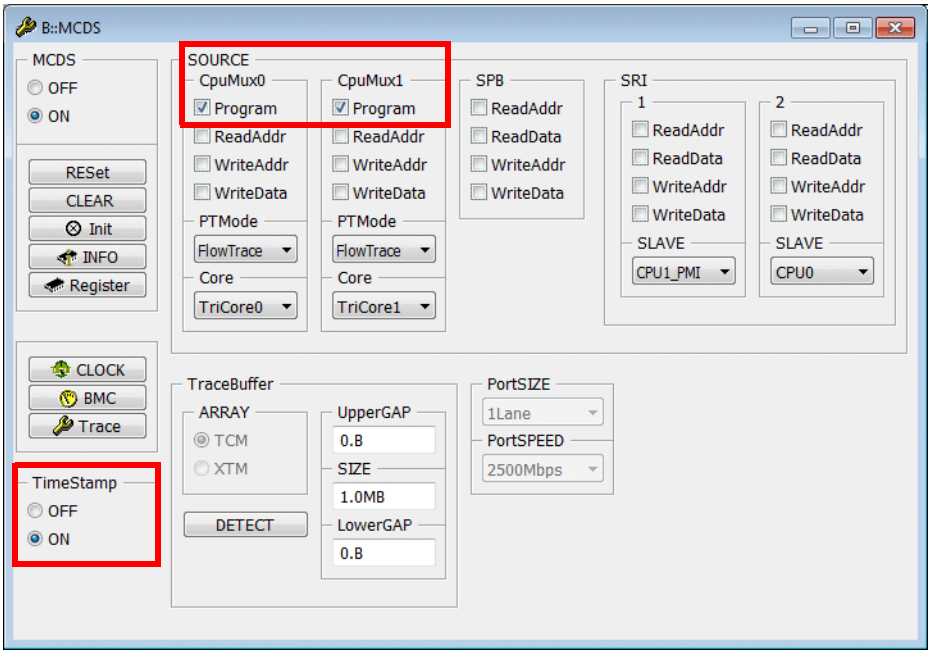

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the event.




```
Break.Set   sieve /Program /TraceEnable
```

3. Configure which trace messages are generated while the event is true.

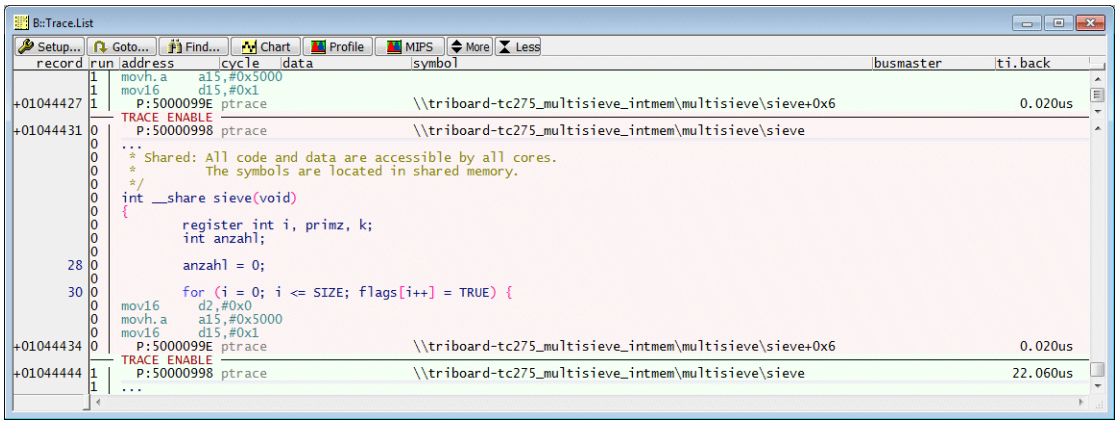



```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU1
```

4. Start the program execution and stop it.

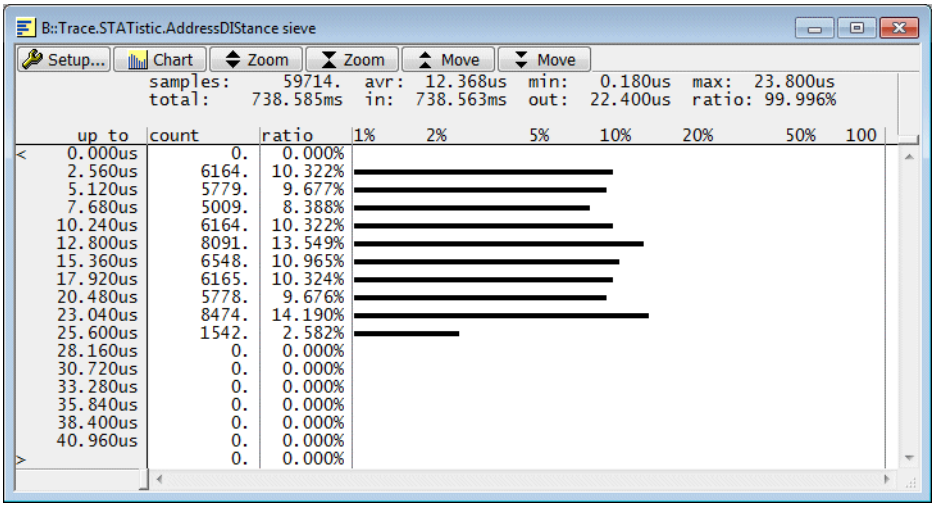
5. Display the result.

The trace contains only small code sections generated for the entries to the function sieve (TRACE ENABLE).



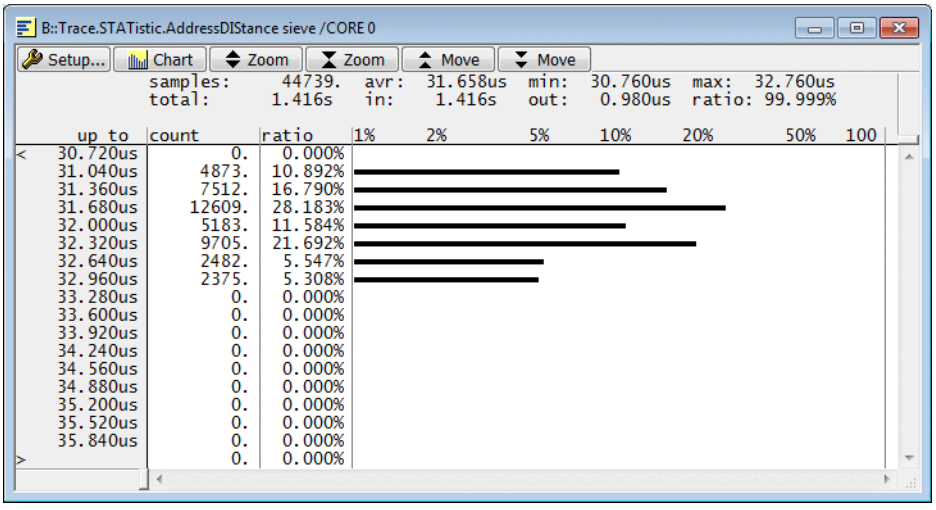
The following **Trace.STATistic** command calculates the time intervals for a program address event. The program address event is here the entry to the function sieve. The core information is discarded for this calculation.

```
Trace.STATistic.AddressDIStance sieve [/JoinCORE]
```



If you need the result per core, use the following command:

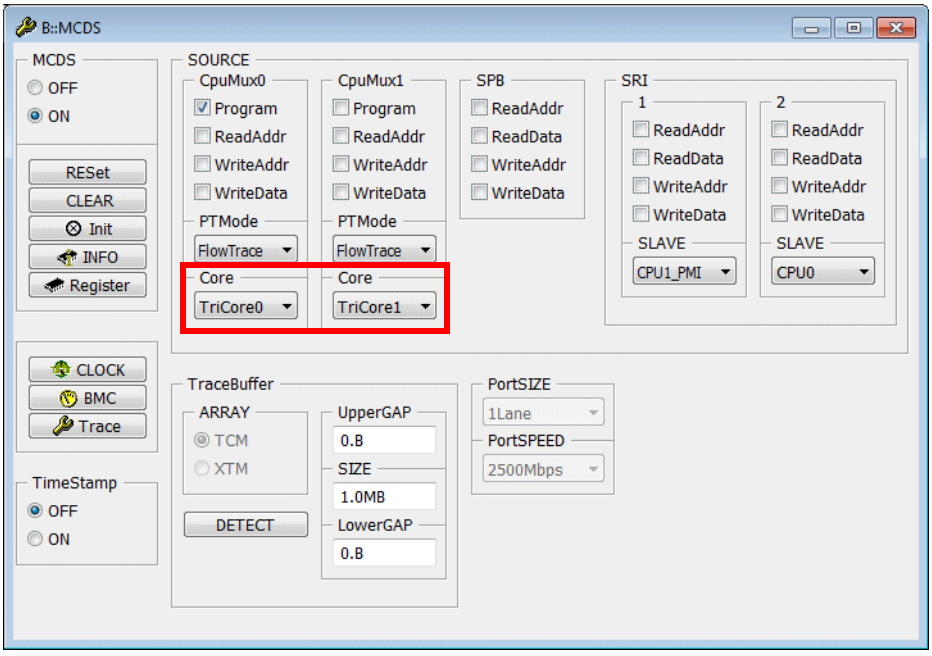
```
Trace.STATistic.AddressDIStance sieve /CORE 0
```



Example 2: Restrict the generated trace information to the entries to the function sieve and the exits from the function sieve.

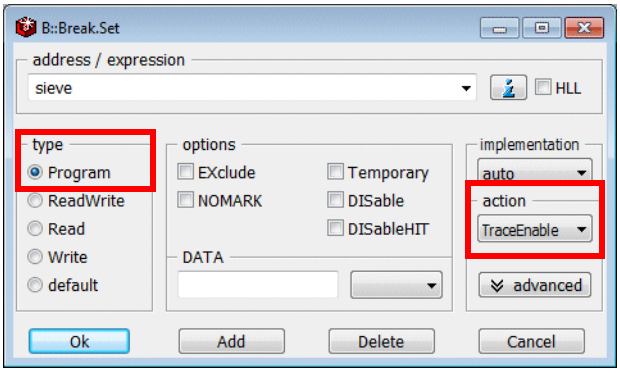
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Entry to function sieve and exit of function sieve.
- **Requested trace messages:** Instruction Pointer Call Messages for both cores, Timestamp Messages.

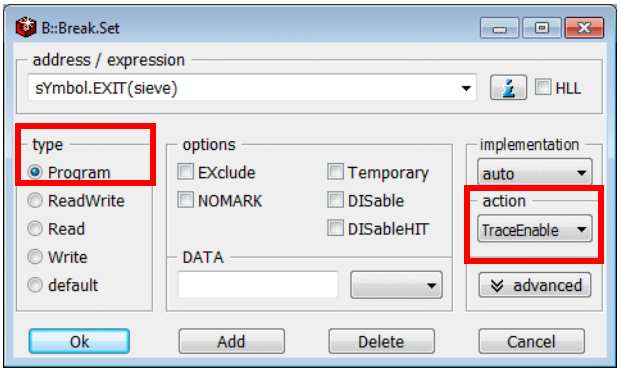
1. Configure the trace multiplexer.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the events.

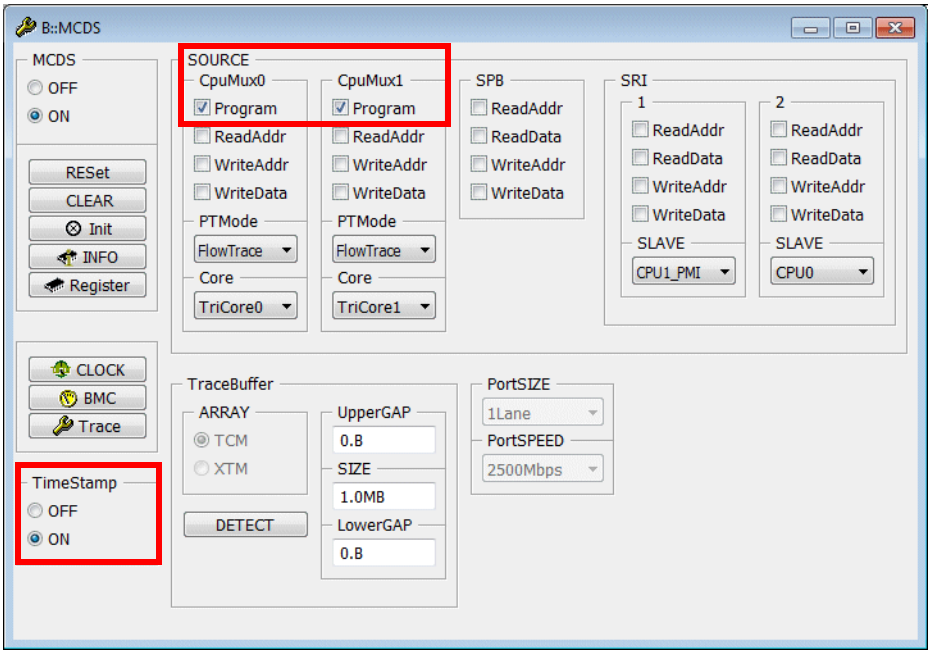




sYmbol.EXIT(<*symbol*>**)** Returns the exit address of the specified function

```
Break.Set   sieve /Program /TraceEnable

Break.Set   sYmbol.EXIT(sieve) /Program /TraceEnable
```

3. **Configure which trace messages are generated while the events are true.**

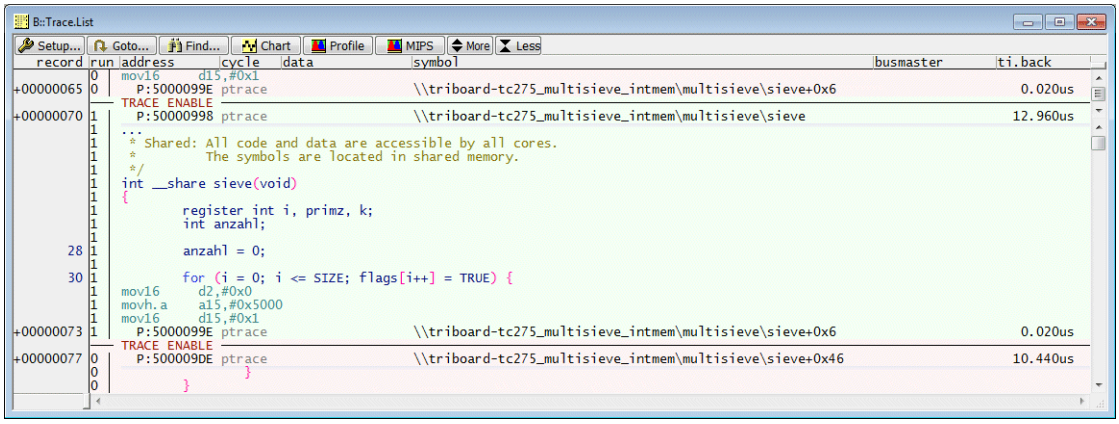



```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program ON      ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU1
```

4. **Start the program execution and stop it.**

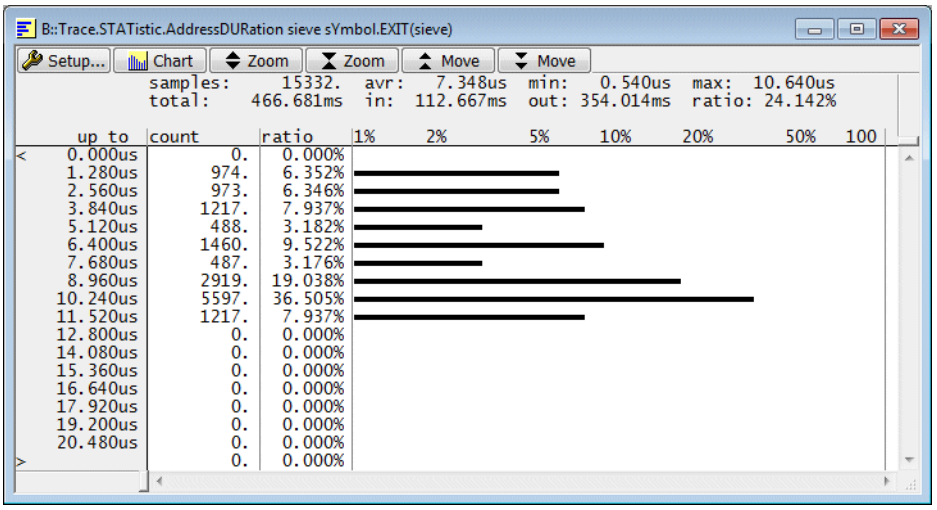
5. Display the result.

The trace contains only small code sections generated for the entries to the function sieve (TRACE ENABLE) and for the exits of the function sieve (TRACE ENABLE).



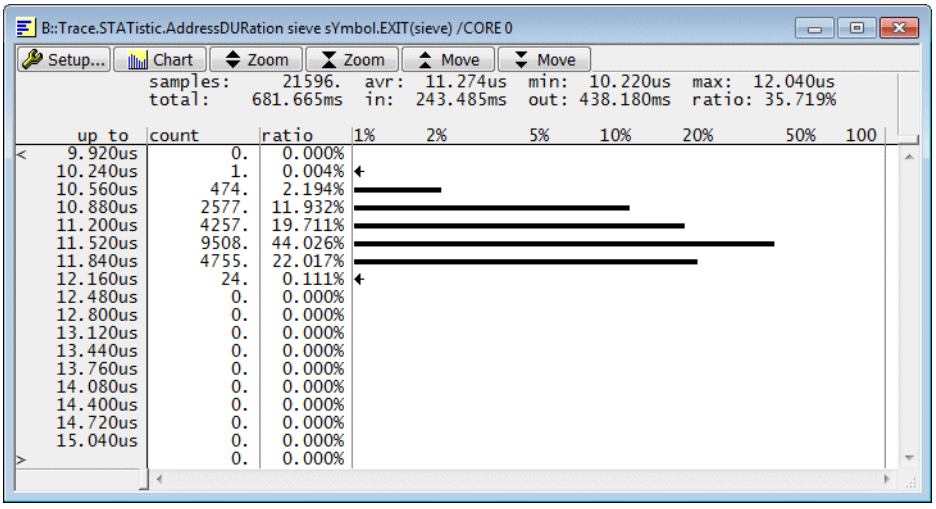
The following **Trace.STATistic** command calculates the time intervals between two program address events A and B. The entry to the function sieve is A in this example, the exit from the function is B. The core information is discarded for this calculation.

```
Trace.STATistic.AddressDURation sieve sYmbol.EXIT(sieve) [/JoinCORE])
```



If you need the result per core, use the following command:

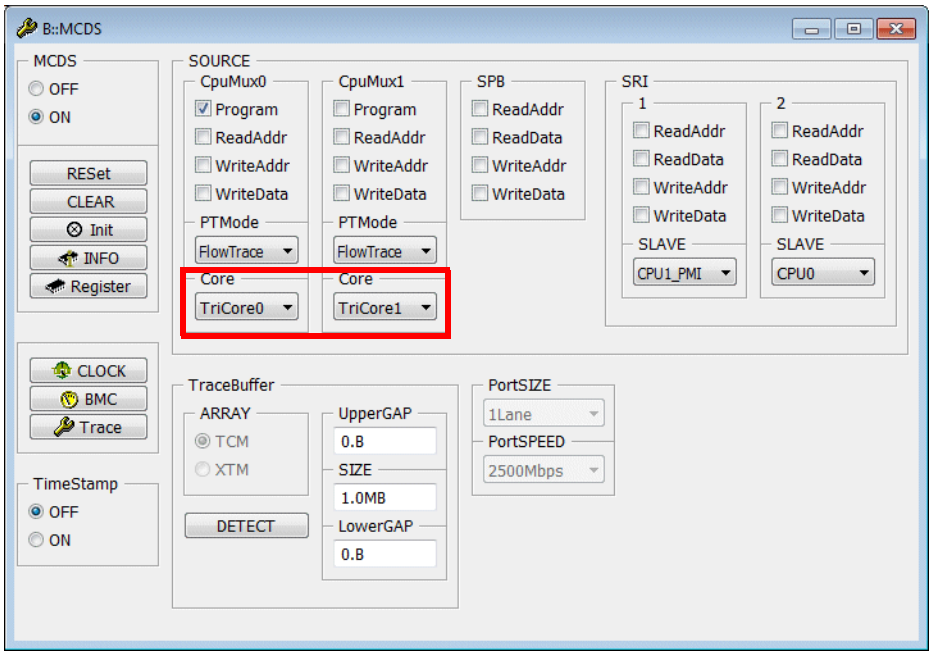
```
Trace.STATistic.AddressDURation sieve sYmbol.EXIT(sieve) /CORE 0
```



Example 3: Restrict the generated trace information to write accesses to the variable flags[3].

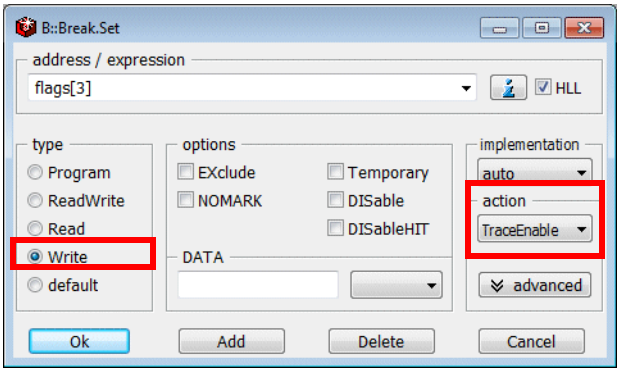
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write access to variable flags[3].
- **Requested trace messages:** Write Data Trace Messages for both cores, Timestamp Messages.

1. Configure the trace multiplexer.



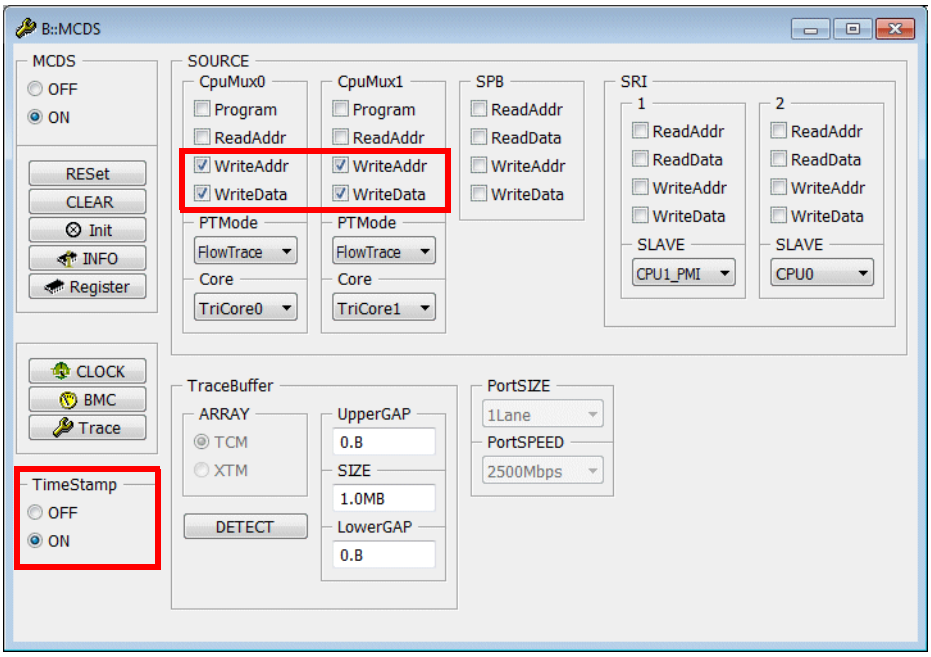

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the events.




```
Var.Break.Set flags[3] /Write /TraceEnable
```

3. Configure which trace messages are generated while the event is true.

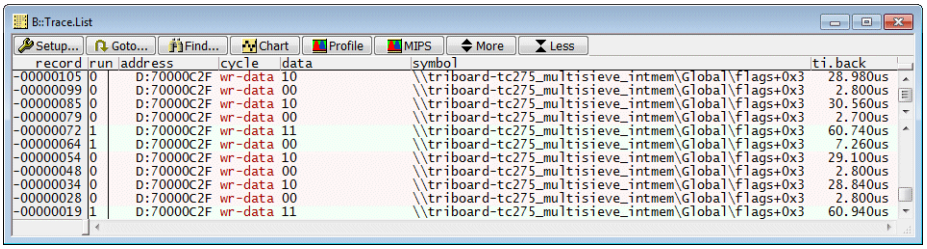



```
MCDS.TimeStamp ON                          ; enable Timestamps
                                           Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON

MCDS.SOURCE.Set CpuMux1.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU1
MCDS.SOURCE.Set CpuMux1.WriteData ON
```

4. Start the program execution and stop it.

5. Display the result.

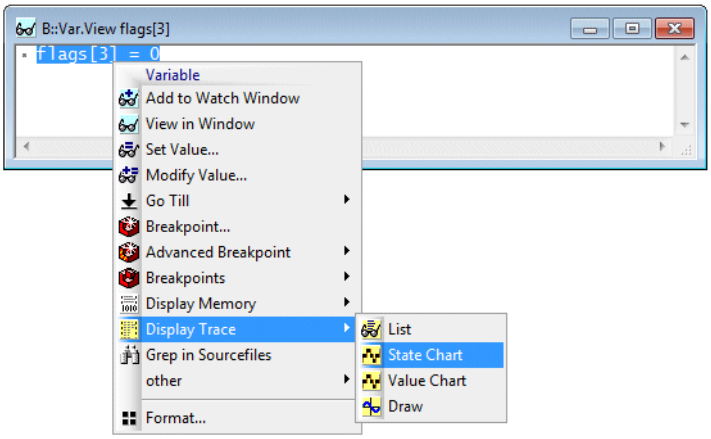
The trace contains only write accesses to the variable flags[3].

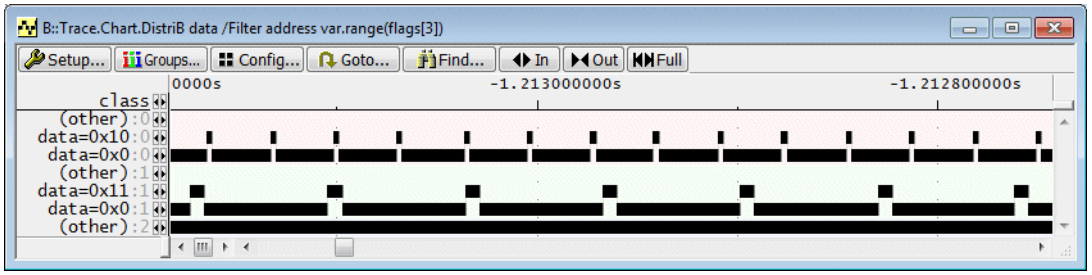


record	run	address	cycle	data	symbol	ti.back
-00000105	0	D:70000C2F	wr-data	10	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	28.980us
-00000099	0	D:70000C2F	wr-data	00	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	2.800us
-00000085	0	D:70000C2F	wr-data	10	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	30.560us
-00000079	0	D:70000C2F	wr-data	00	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	2.700us
-00000072	1	D:70000C2F	wr-data	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	60.740us
-00000064	1	D:70000C2F	wr-data	00	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	7.260us
-00000054	0	D:70000C2F	wr-data	10	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	29.100us
-00000048	0	D:70000C2F	wr-data	00	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	2.800us
-00000034	0	D:70000C2F	wr-data	10	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	28.840us
-00000028	0	D:70000C2F	wr-data	00	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	2.800us
-00000019	1	D:70000C2F	wr-data	11	\\triboard-tc275_multisieve_intmem\Global\flags+0x3	60.940us

The Variable pull-down provides various way to analyze the variable contents over the time.

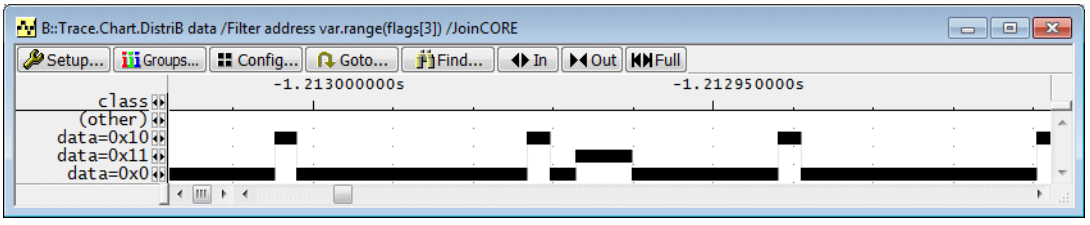
```
; open a window to display the variable
Var.View flags[3]
```



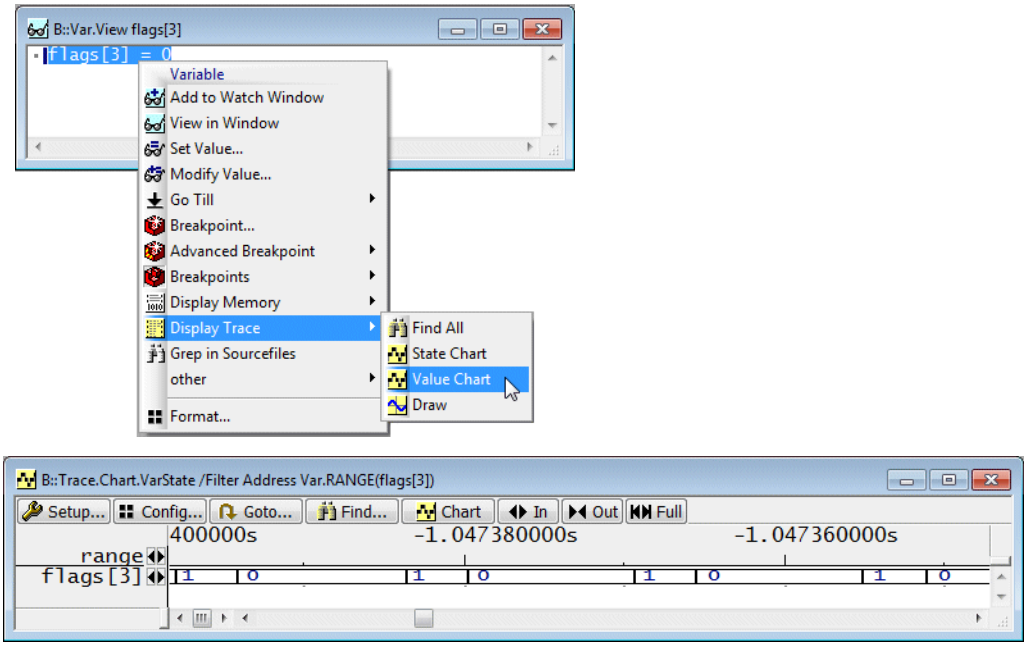


Display the value changes of a variable graphically - split the result per core
Trace.Chart.DistriB Data /Filter Address Var.RANGE(<*var*>**) [/SplitCORE]**

Var.RANGE(<*var*>**)** Returns the address range in which the content of a variable is stored.

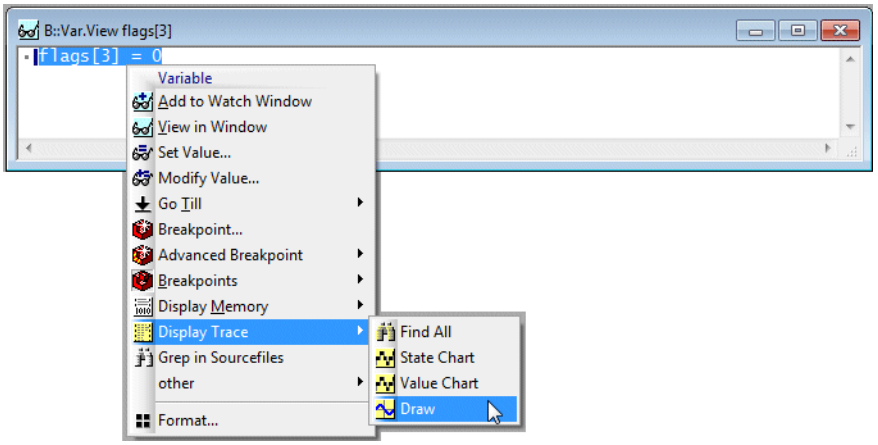


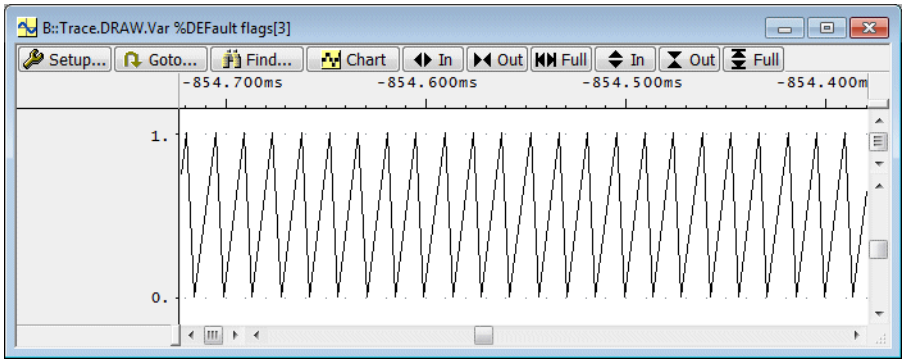
Display the value changes of a variable graphically - ignore core information
Trace.Chart.DistriB Data /Filter Address Var.RANGE(<*var*>**) /JoinCORE**



Display variable contents over the time numerically - the core information is discarded
Trace.Chart.VarState /Filter Address Var.RANGE(<*var*>**)** [**/JoinCORE**]

Display variable contents over the time numerically - the core information is discarded
Trace.Chart.VarState /Filter Address Var.RANGE(<*var*>**) /CORE** <*n*>





Display variable contents over the time graphically - the core information is discarded
Trace.DRAW.Var **%DEFault** *<var>* [**/JoinCORE**]

Display variable contents over the time graphically - the core information is discarded
Trace.DRAW.Var **%DEFault** *<var>* **/CORE** *<n>*

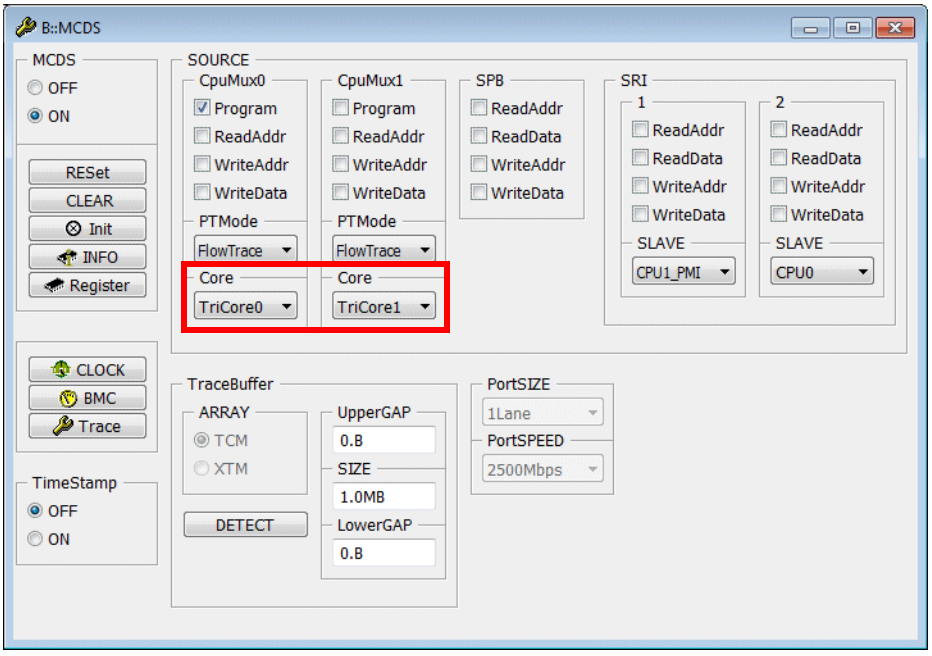
Advise the Processor Observation Block to generate trace messages for the instruction flow and for the specified events.

NOTE:	The TraceData filter is of great importance for the nesting function run-time analysis if an operating system is used.

Example: Generate trace information for the complete program flow and for all write accesses to flags[12].

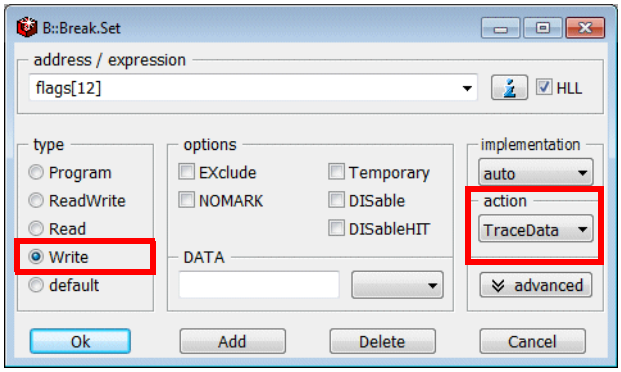
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write access to variable flags[12].
- **Requested trace messages:** Timestamp Messages.

1. Configure the trace multiplexer.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. **Specify the event.**



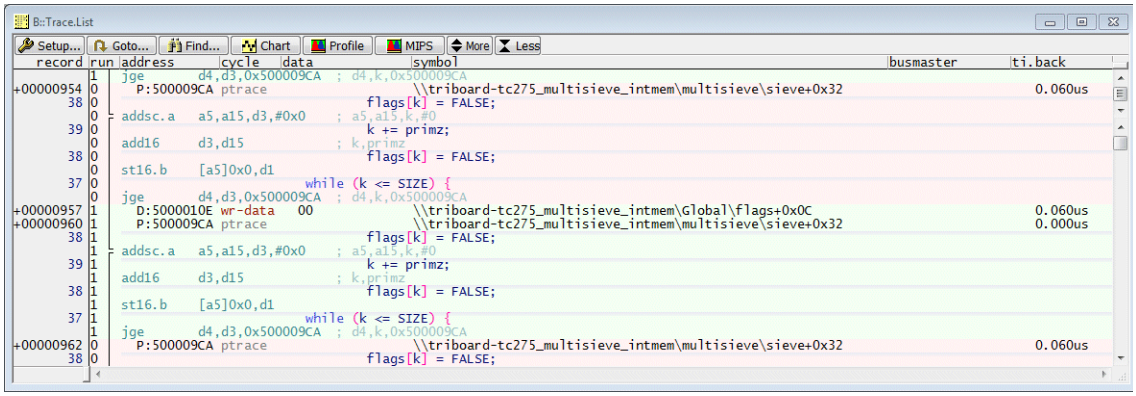

```
Var.Break.Set flags[12] /Write /TraceData
```

3. **TRACE32 takes care of the message generation.**

4. **Start the program execution and stop it.**

5. **Display the result.**

The trace contains the complete program flow and all write accesses to the variable flags[12].



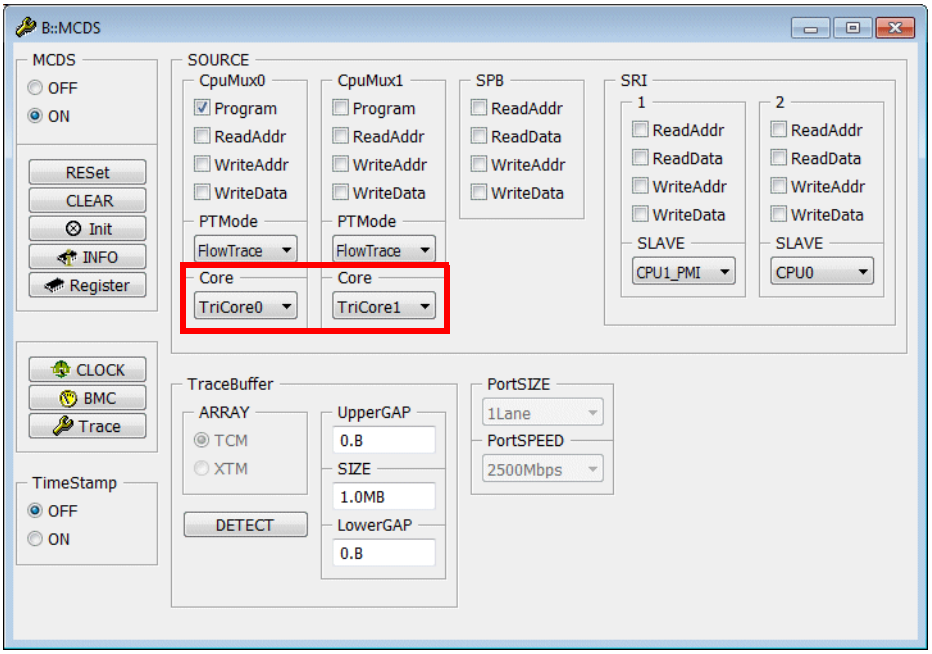
TraceON: Advise the Processor Observation Blocks to start the generation of trace messages for the enabled SOURCEs at the specified event.

TraceOFF: Advise the Processor Observation Blocks to stop the generation of trace messages at the specified event.

Example: Restrict the generation of trace messages to the function sieve1.

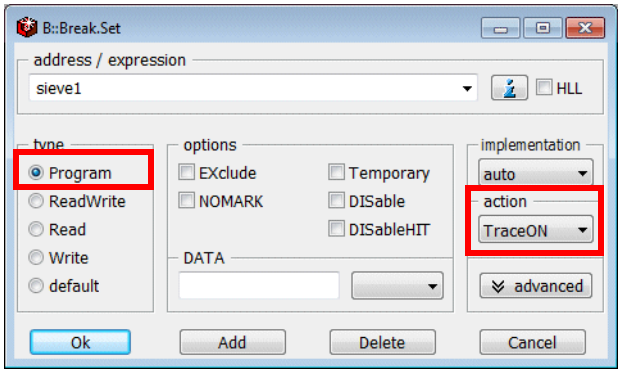
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Entry to function sieve1 and exit of function sieve1.
- **Requested trace messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Read Data Trace Messages, Timestamp Messages.

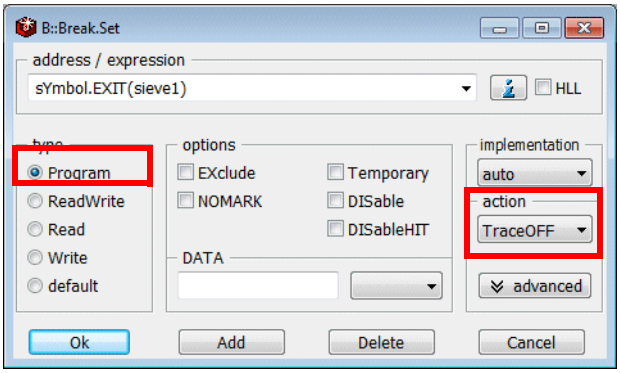
1. Configure the trace multiplexer.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

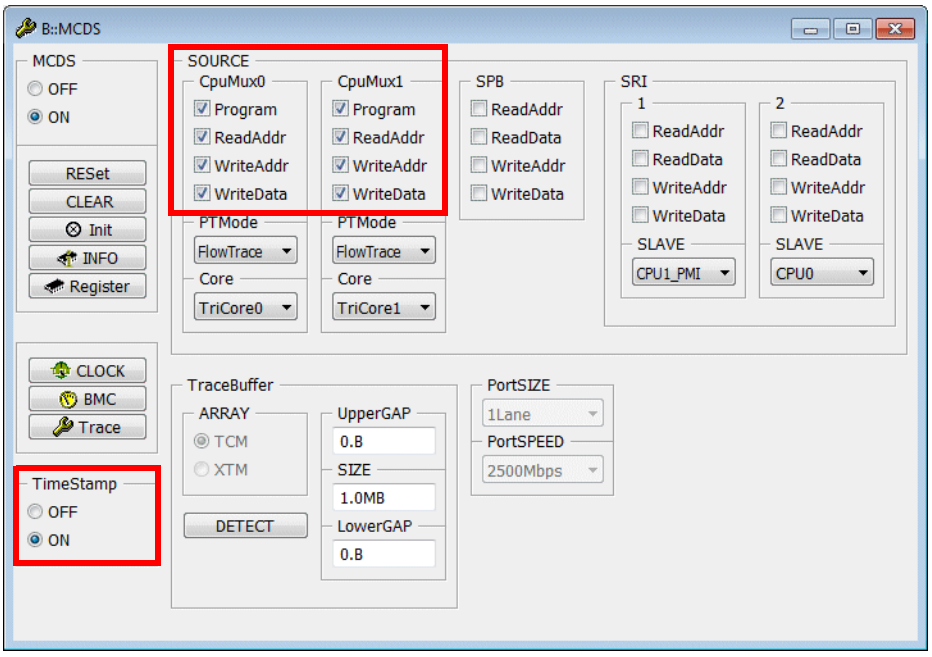
2. Specify the events.






```
Break.Set   sieve1 /Program /TraceON
Break.Set   sYmbol.EXIT(sieve1) /Program /TraceOFF
```

3. **Configure which messages are generated when the message generation is active (after TraceON event).**

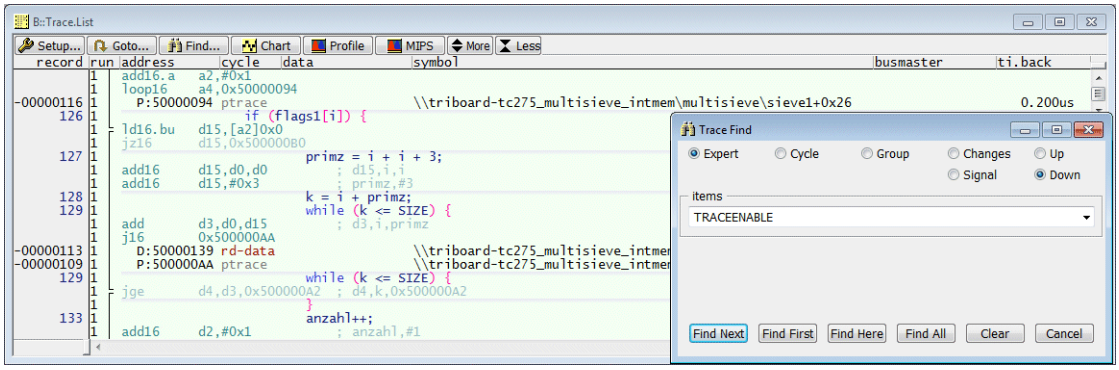


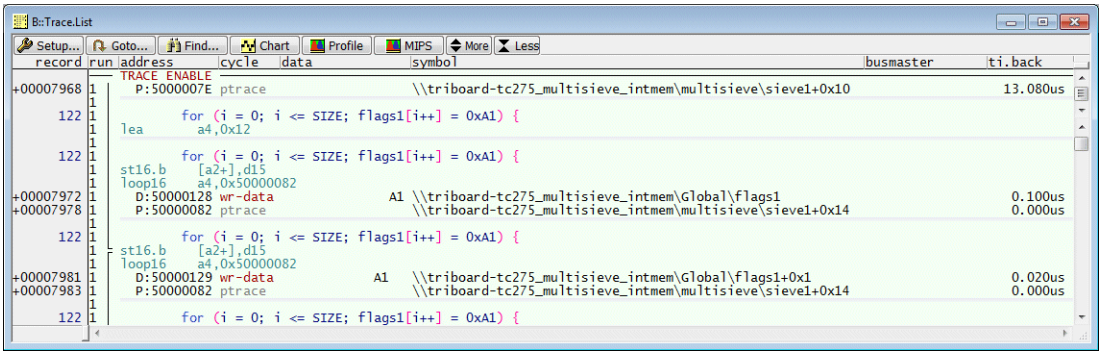

```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON         ; enable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.ReadAddr ON        ; enable Read Data Trace
                                           ; Messages for TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
MCDS.SOURCE.Set CpuMux0.WriteData ON       ; Messages for TC 1.6.1 CPU0


MCDS.SOURCE.Set CpuMux1.Program ON         ; enable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux1.ReadAddr ON        ; enable Read Data Trace
                                           ; Messages for TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux1.WriteAddr ON       ; enable Write Data Trace
MCDS.SOURCE.Set CpuMux1.WriteData ON       ; Messages for TC 1.6.1 CPU1
```

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

5. Display the result.

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





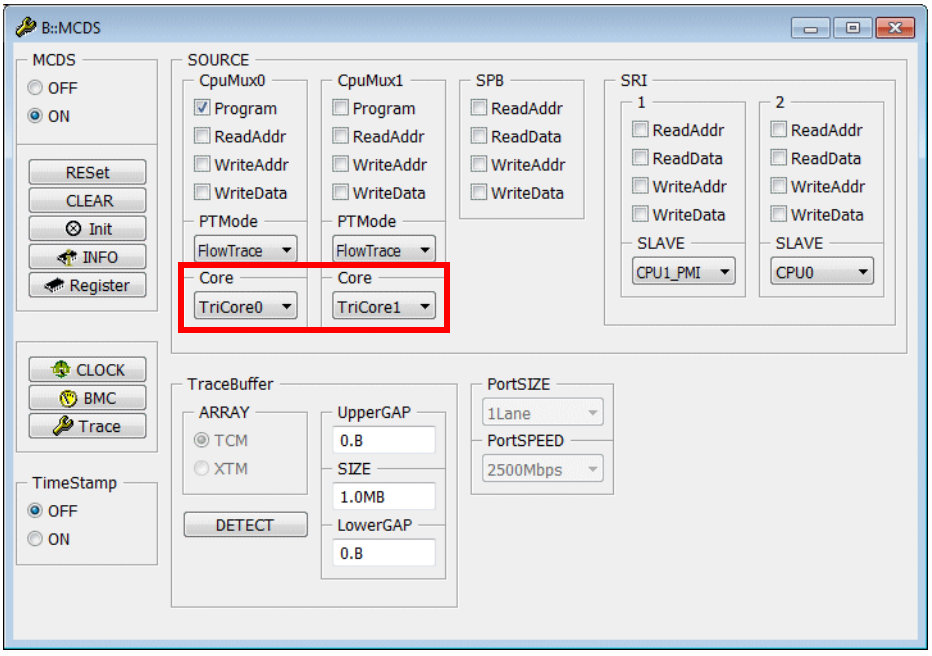
Trace message generation is started after the TraceON event occurred. As a result this event is not visible in the trace.

Advise the Processor Observation Blocks to end the message generation at the specified event.

Example 1: Stop the trace recording when 161. was written to the variable flagsc[8].

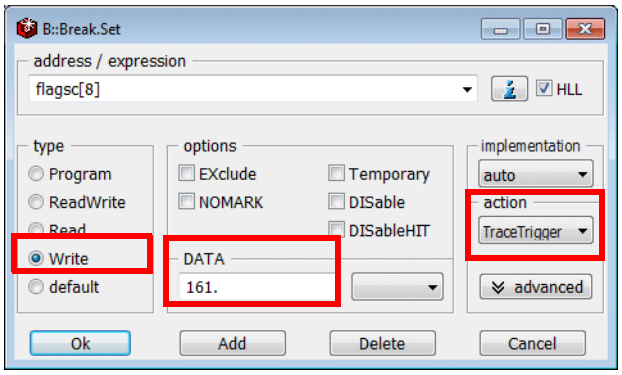
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write of 161. to variable flagsc[8].
- **Requested trace messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



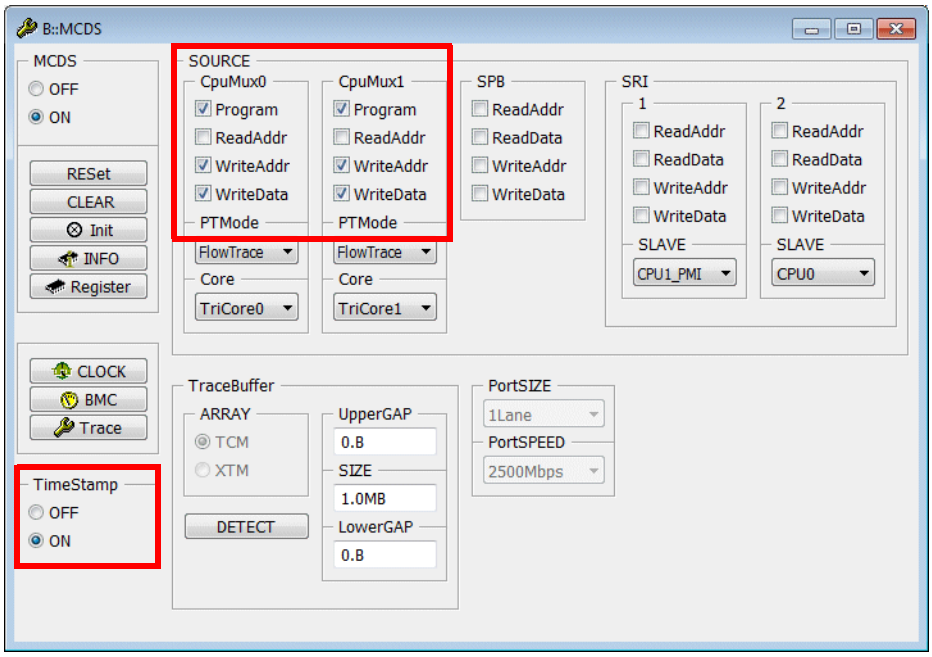

```
MCDS.SOURCE CpuMux0 Core TriCore0     ; enable TC 1.6.1 CPU0 as
                                      ; trace source

MCDS.SOURCE CpuMux1 Core TriCore1     ; enable TC 1.6.1 CPU1 as
                                      ; trace source
```

2. Specify the event.



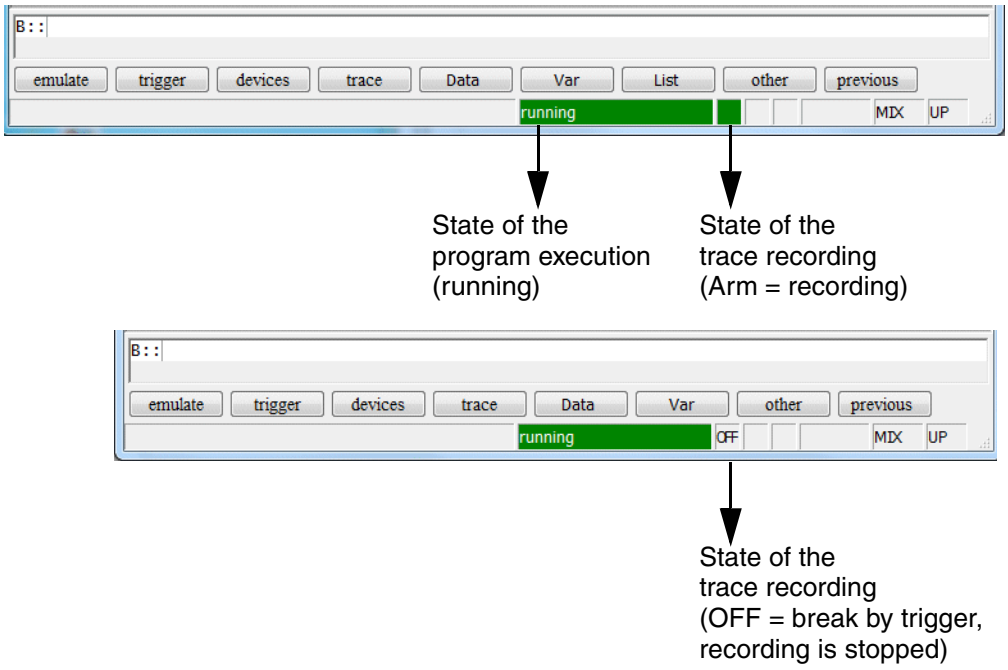

```
Var.Break.Set flagsc[8] /Write /DATA.Byte 0xA1 /TraceTrigger
```

3. Configure which trace messages are generated until the trigger event occurs.



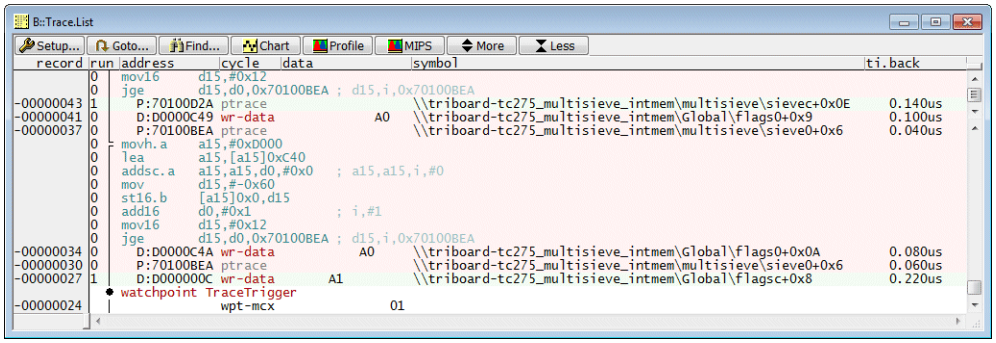

```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE CpuMux0 Program ON          ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE CpuMux0 WriteAddr ON        ; enable Write Data Trace
MCDS.SOURCE CpuMux0 WriteData ON        ; Messages for TC 1.6.1 CPU0


MCDS.SOURCE CpuMux1 Program ON          ; enable Instruction Pointer
                                        ; Call Messages for
                                        ; TC 1.6.1 CPU1

MCDS.SOURCE CpuMux1 WriteAddr ON        ; enable Write Data Trace
MCDS.SOURCE CpuMux1 WriteData ON        ; Messages for TC 1.6.1 CPU1
```

4. Start the program execution.



5. Display the result.

MCDS ends the generation of trace messages and flushes all internal buffer when the specified event occurs. TRACE32 automatically generates a **watchpoint TraceTrigger** message when the trigger event occurs. This helps you to find the actual trigger event in the trace.

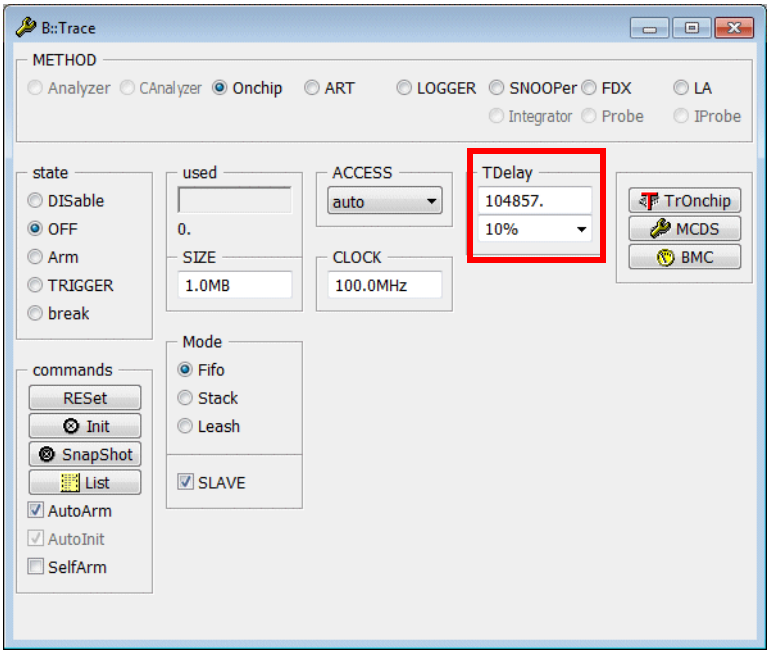


Example 2: Stop the trace recording after another 10% of the trace memory was filled when 161. was written to the variable flagsc[8].

- **System under debug:** SMP system with 3 TriCore cores.
- **Cores connected to the trace multiplexer:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write of 161. to variable flagsc[8].
- **Requested trace messages:** Instruction Pointer Call Messages, Write Data Trace Messages, Timestamp Messages.

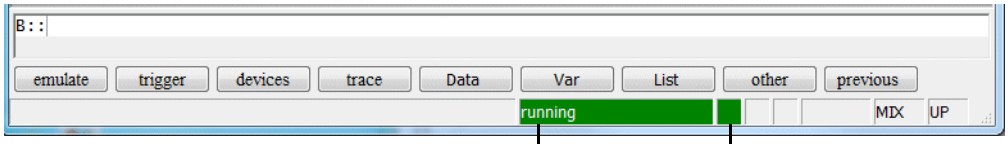
1. to 3. as in example 1.

4. Specific the delay.



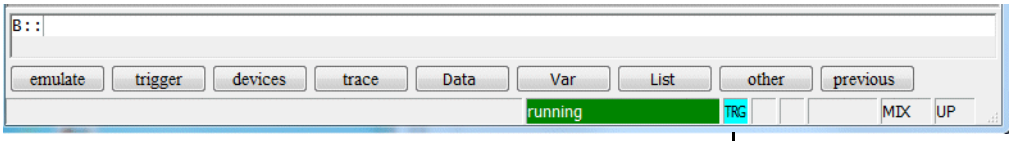

```
Trace.TDelay 10%
```

5. Start the program execution.

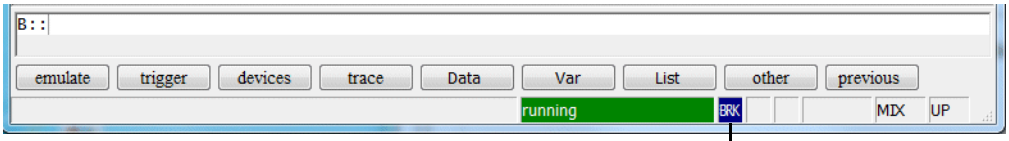


State of the program execution (running)

State of the trace recording (Arm = recording)

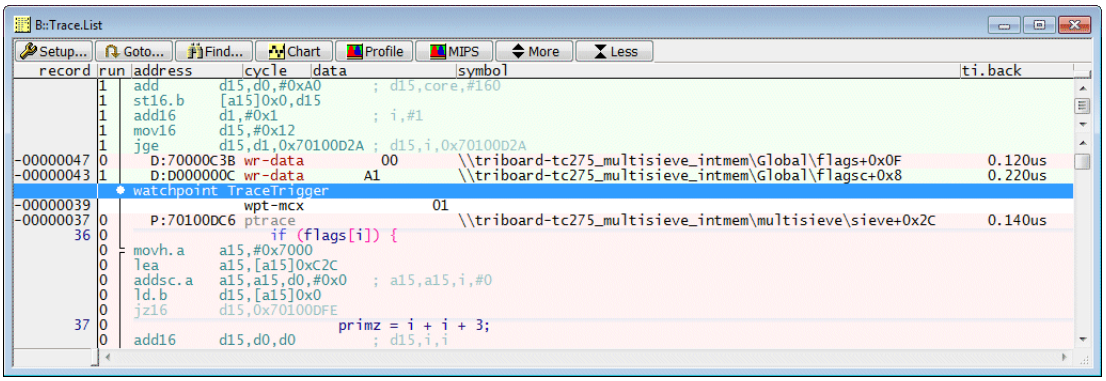


State of the trace recording (TRG = trigger occurred, delay counter started)



State of the trace recording (BRK = delay counter elapsed, recording is stopped)

6. Display the result.



A TraceTrigger watchpoint indicates the occurrence of the TraceTrigger event.

Activate the TRACE32 OS Awareness (Supported OS)

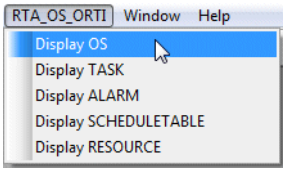
AMP Systems: Since each core is controlled by a separate operating system, the OS Awareness has to be activated separately for each TRACE32 instance.

Since most users use an OSEK operating system this is taken as an example here. Setup command:

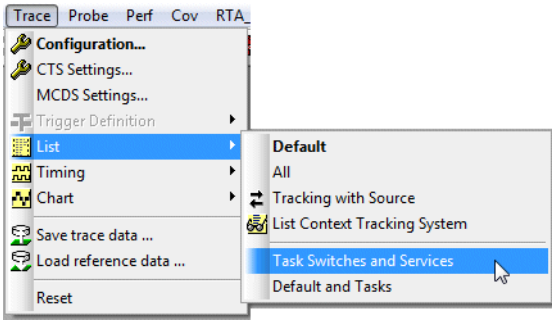
TASK.ORTI *<orti_file>*	Load the ORTI file

Loading the ORTI file results in the following:

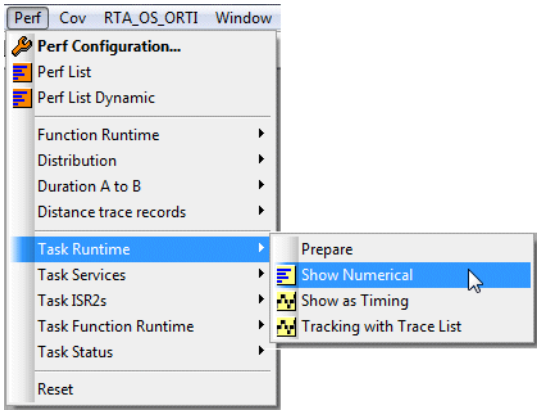
- Symbolic debugging of the OSEK OS is possible. Debug commands are provided via an ORTI menu.



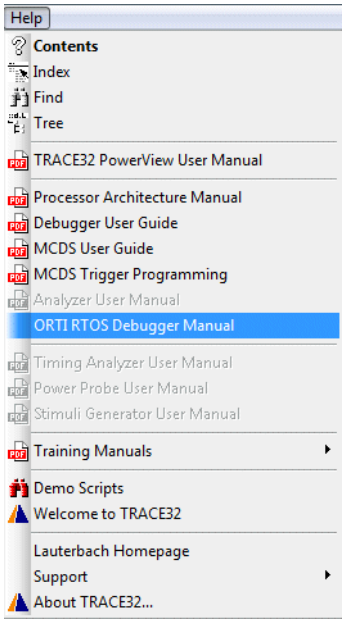
- The **Trace** menu is extended for OS-aware trace display.



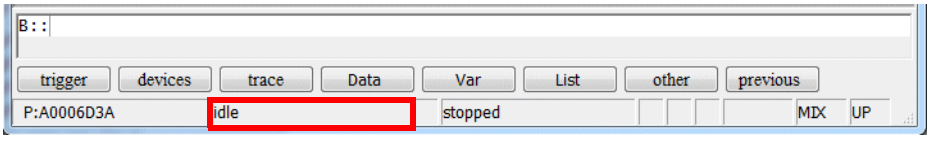
- The **Perf** menu is extended for OS-aware profiling.



- The manual of the OS Awareness for OSEK/**ORTI** is added to the **Help** menu.



- The name of the current task is displayed in the **Task** field of the TRACE32 state line.



Exporting the Task Switches

Each operating system has a variable that contains the information which task is currently running. This variable can hold a task ID, a pointer to the task control block or something else that is unique for each task.

MCDS can be configured to generate a Write Data Trace Message when a write access to this variable occurs.

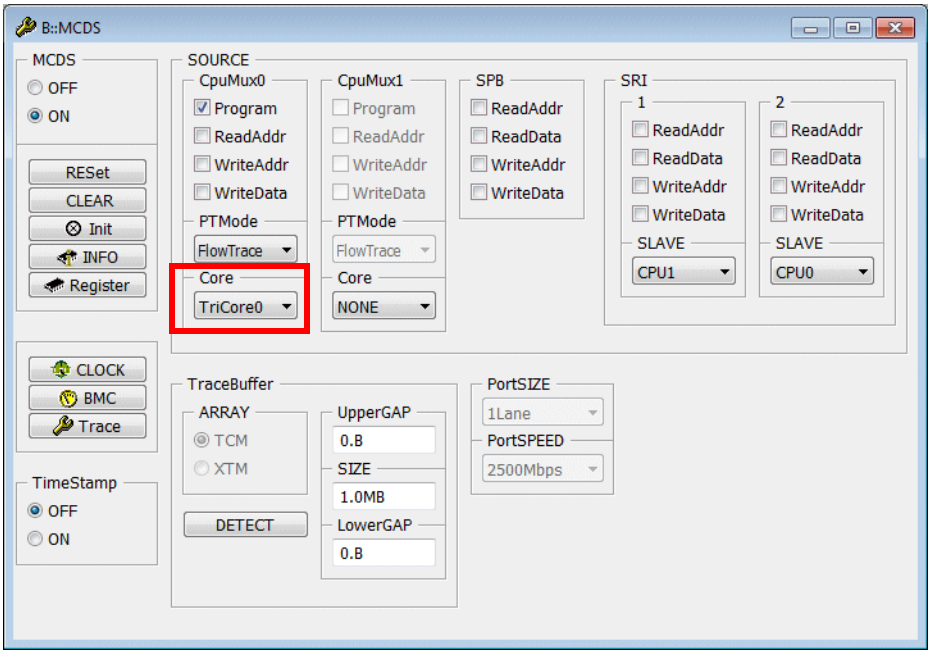
The address of this variable is provided by the TRACE32 function **TASK.CONFIG(magic)**.

```
PRINT TASK.CONFIG(magic)                ; print the address that holds
                                        ; the task identifier
```

Example: Advise the Processor Observation Block to generate trace messages only on task switches.

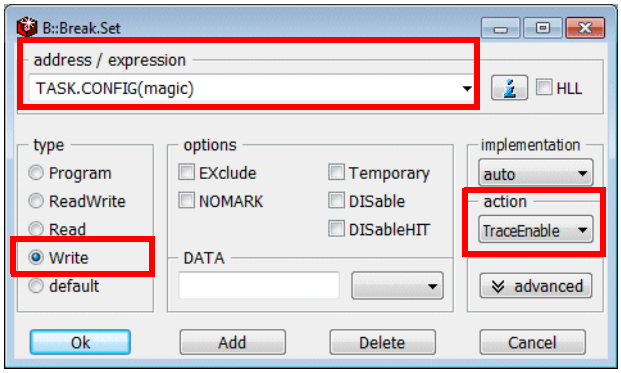
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write access to TASK.CONFIG(magic)
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



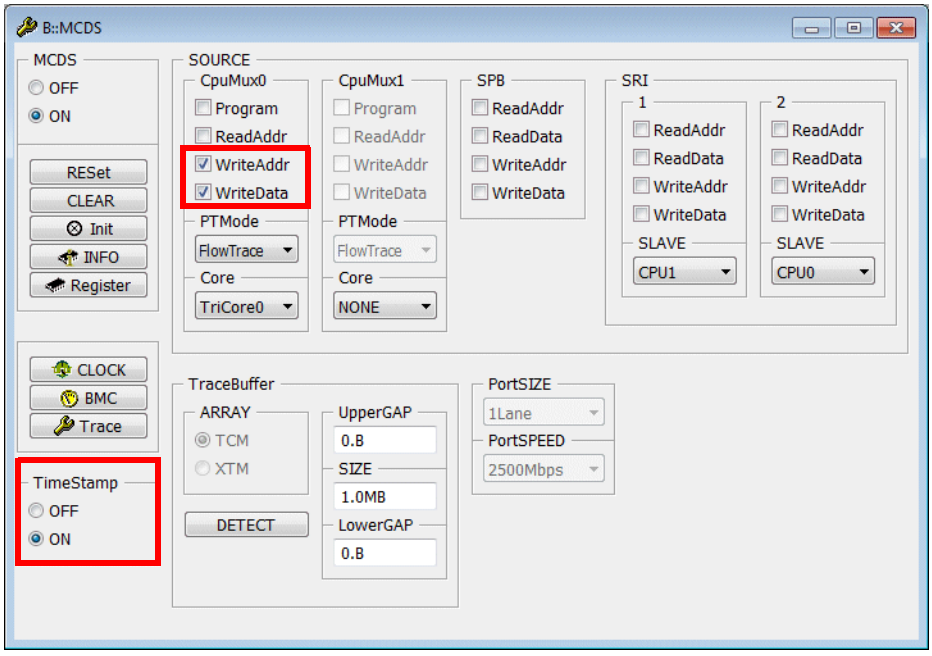

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source
```

2. Specify the event.




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

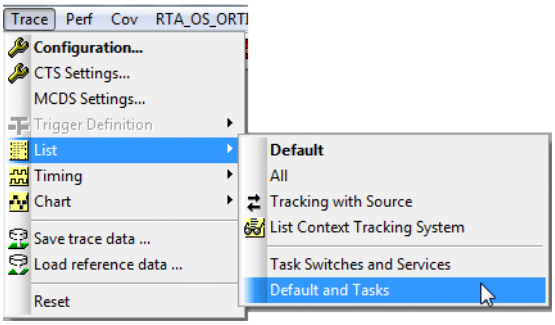
3. Configure which trace messages are generated.

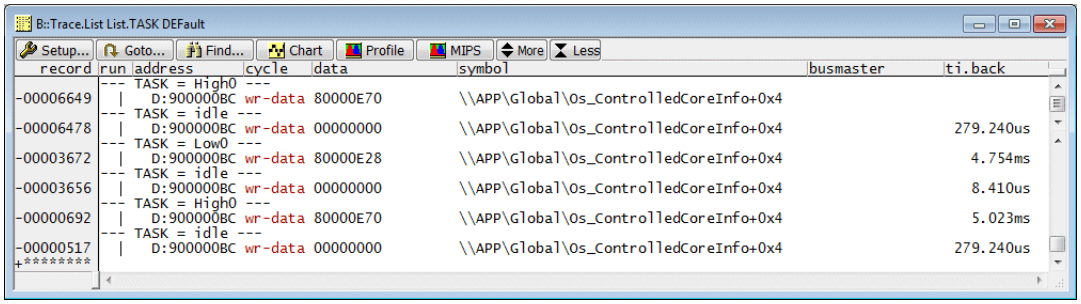



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON
```

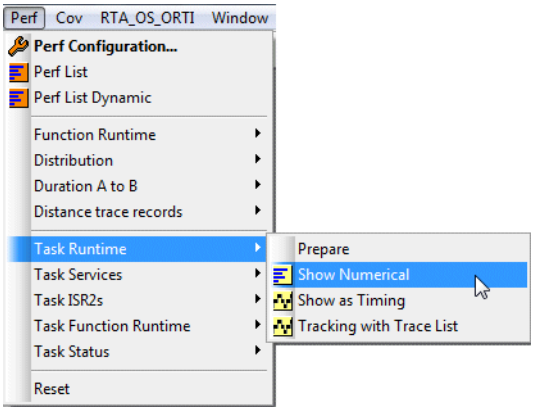
4. **Start and stop the program execution to fill the trace buffer.**

5. **Display the result.**

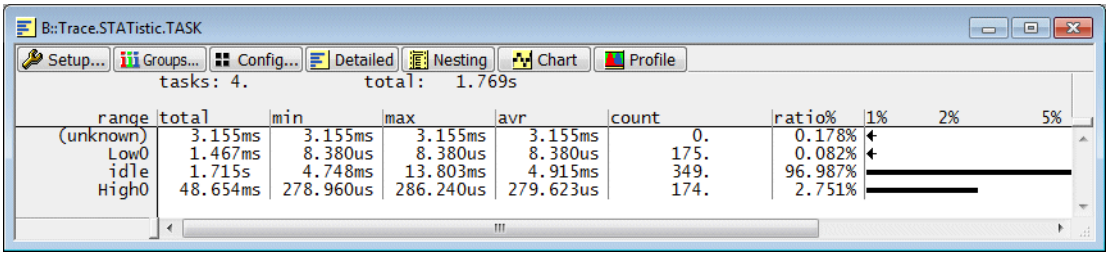




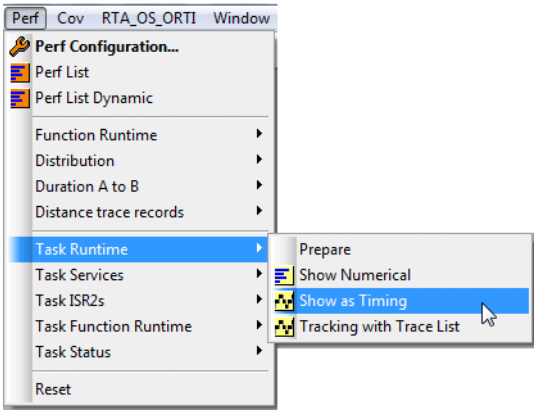
The following two commands perform a statistical analysis of the task switches:

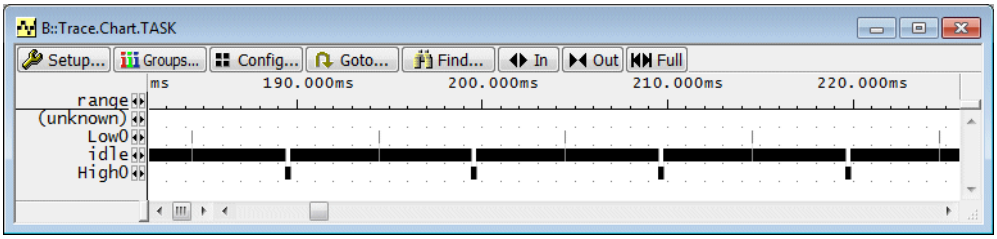


Trace information recorded before the first task switch is assigned to (unknown).



Trace.STATistic.TASK	Numeric analysis of task run-times.





Trace.Chart.TASK Time-chart of tasks.

The ORTI file may also provide means to analyze the time in task service routines.

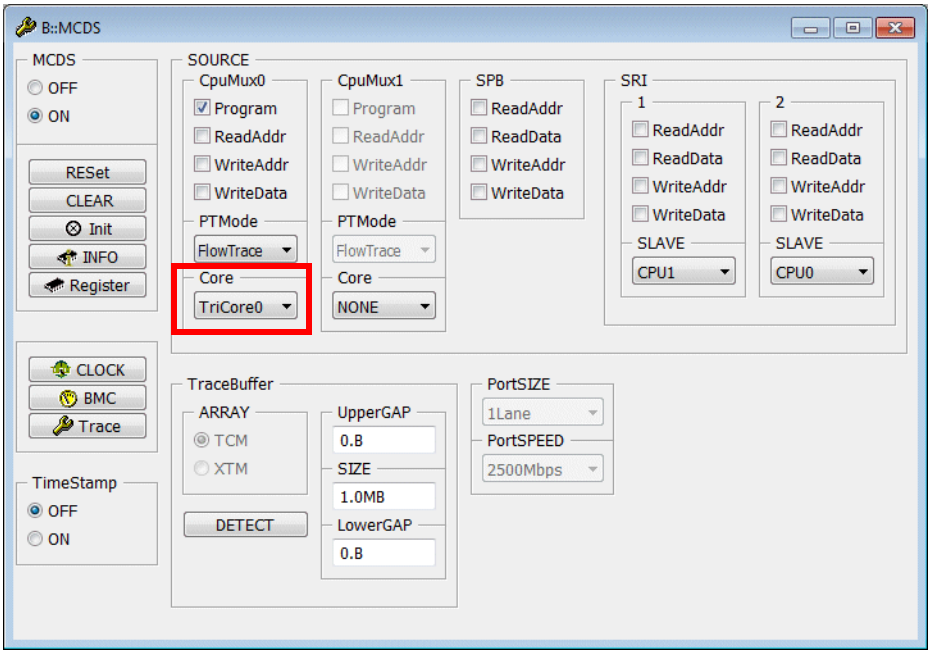
TASK.CONFIG(magic_service) is the name of the TRACE32 function that is used for this purpose.

```
PRINT TASK.CONFIG(magic_service)      ; print the address that holds
                                      ; the service table
```

Example: Advise the Processor Observation Block to generate trace messages only on write accesses to the service table.

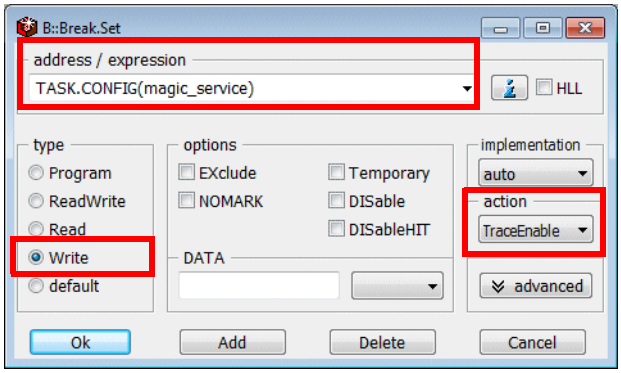
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write access to TASK.CONFIG(magic_service)
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



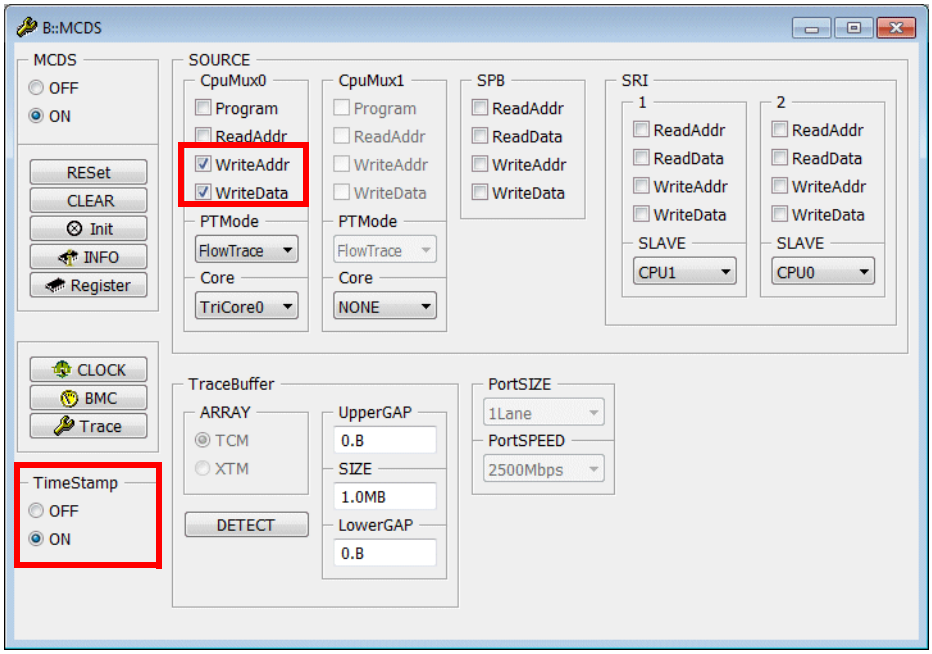

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source
```

2. Specify the event.




```
Break.Set TASK.CONFIG(magic_service) /Write /TraceEnable
```

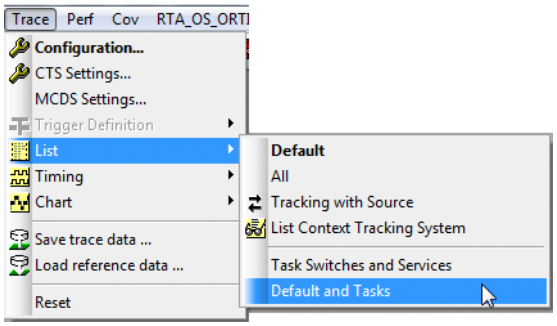
3. Configure which trace messages are generated.

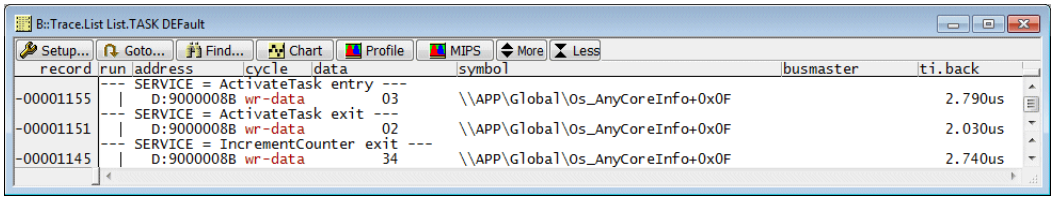



```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF     ; disable Instruction
                                        ; Pointer Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON    ; enable Write Data Trace
                                        ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON
```

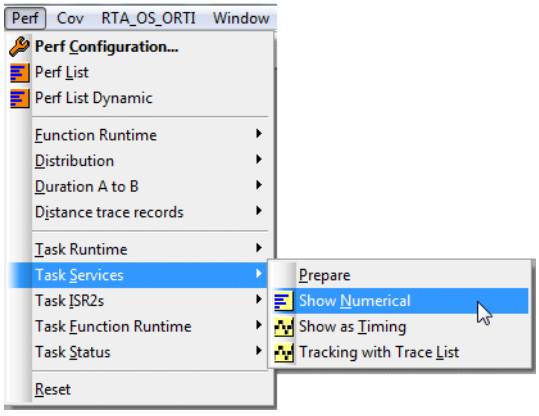
4. Start and stop the program execution to fill the trace buffer.

5. Display the result.

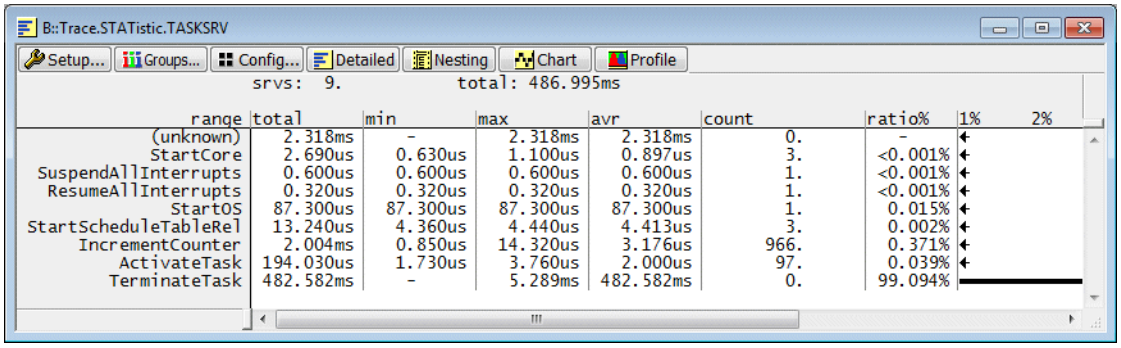




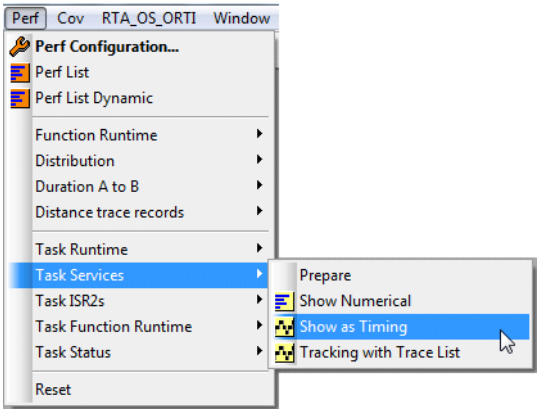
The following two commands perform a statistical analysis of the time in task service routines:

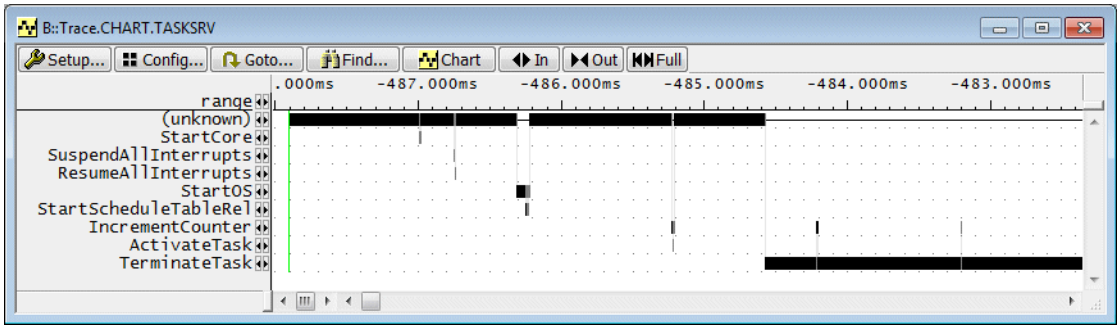


(unknown) represents the time in which the core is not in an OSEK service routine



Trace.STATistic.TASKSRV	Numeric analysis of task services.





Trace.Chart.TASKSRV Time-chart of task services.

Exporting ISR2 (OSEK Interrupt Service Routines)

The ORTI file may also provide means to analyze the time in interrupt service routines.

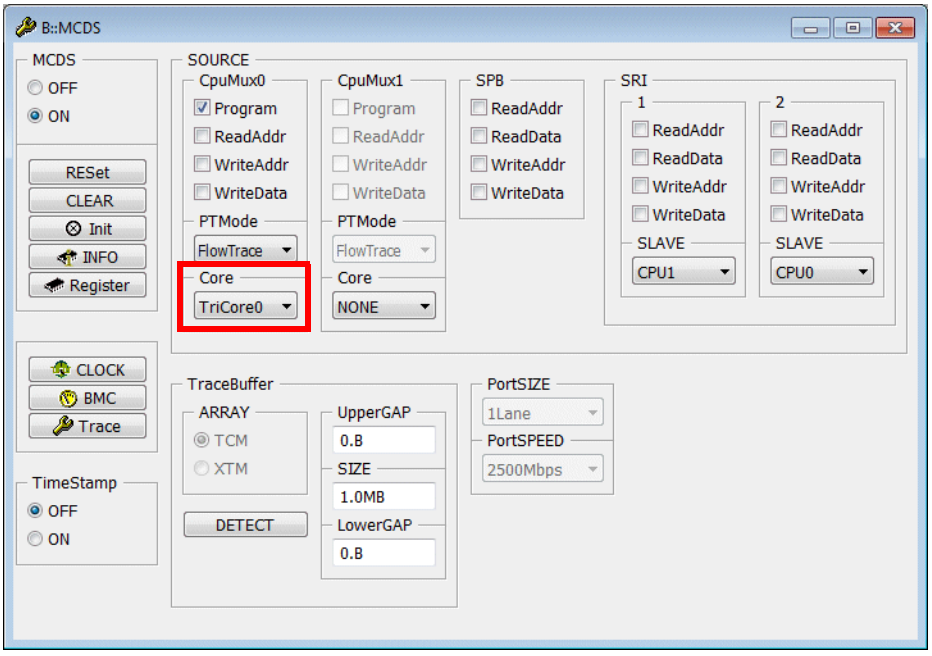
TASK.CONFIG(magic_isr2) is the name of the TRACE32 function that is used for this purpose.

```
PRINT TASK.CONFIG(magic_isr2)          ; print the address that holds
                                       ; the interrupt service table
```

Example: Advise the Processor Observation Block to generate trace messages only on write accesses to the interrupt service table.

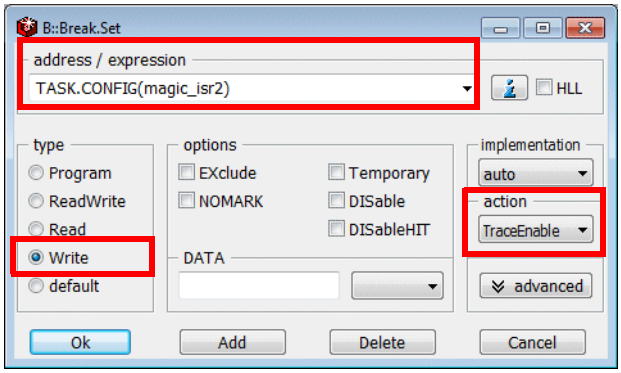
- **Core under debug:** TC 1.6.1 CPU0.
- **Event of interest:** Write access to TASK.CONFIG(magic_isr2)
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. Configure the trace multiplexer.



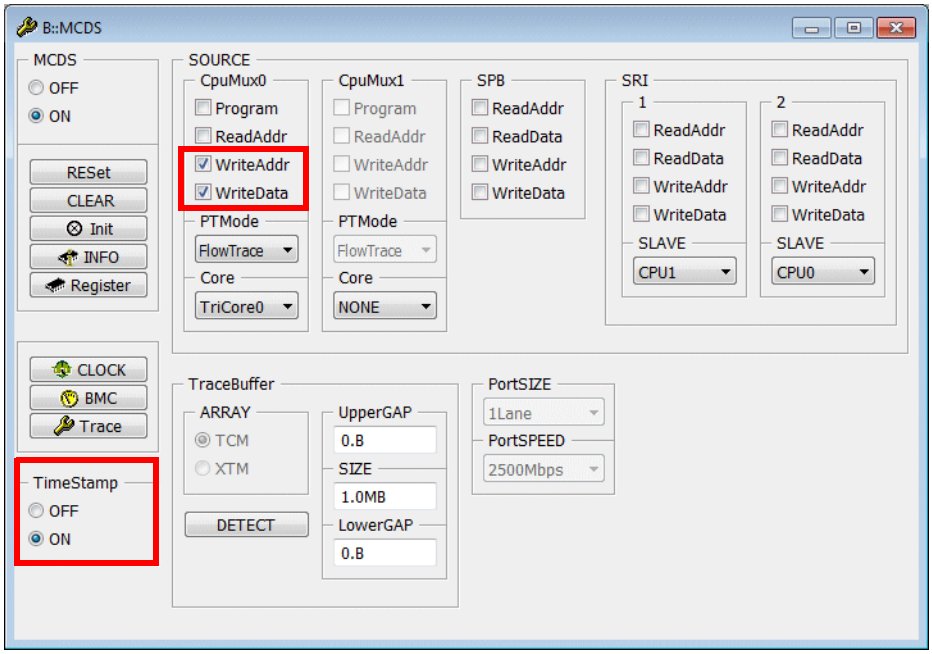

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source
```

2. Specify the event.




```
Break.Set TASK.CONFIG(magic_isr2) /Write /TraceEnable
```

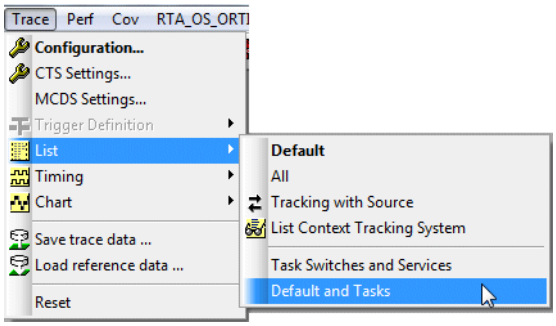
3. Configure which trace messages are generated.

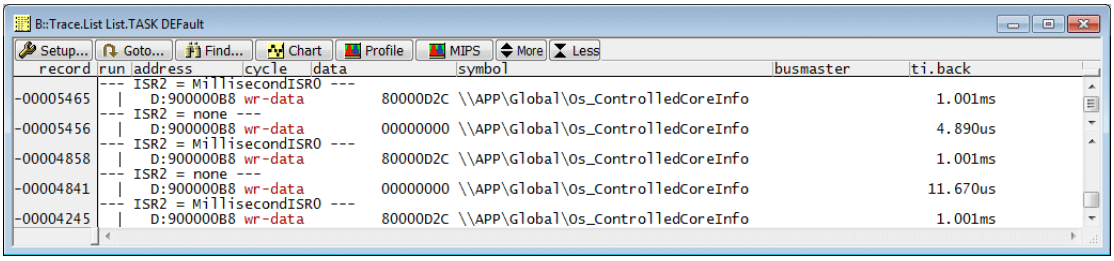



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON
```

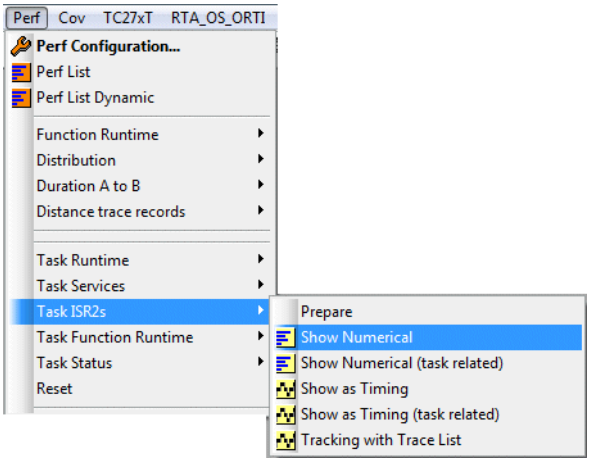
4. **Start and stop the program execution to fill the trace buffer.**

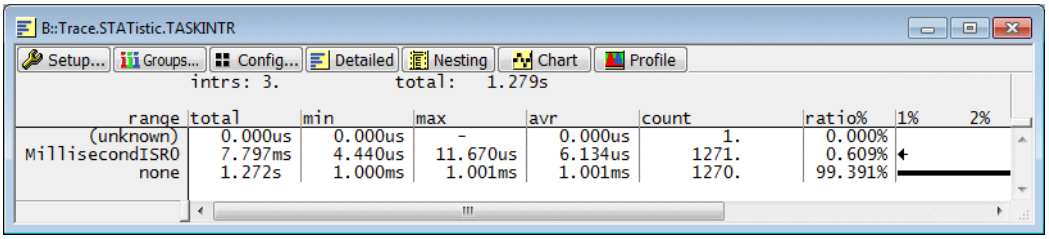
5. **Display the result.**



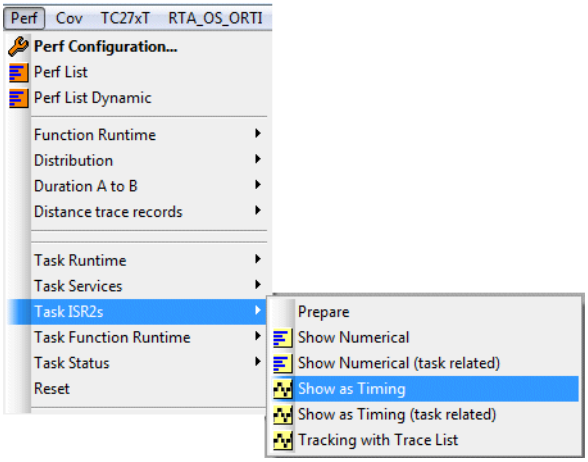


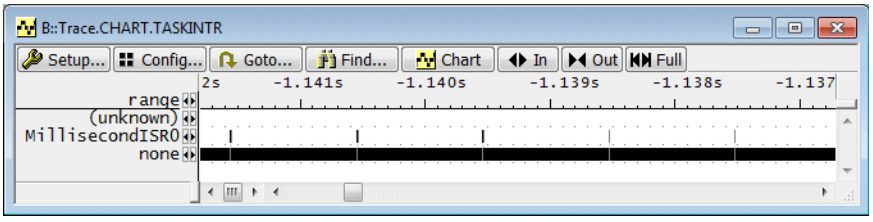
The following two commands perform a statistical analysis of the time in interrupt service routines:





Trace.STATistic.TASKINTR Numeric analysis of ISR2s.

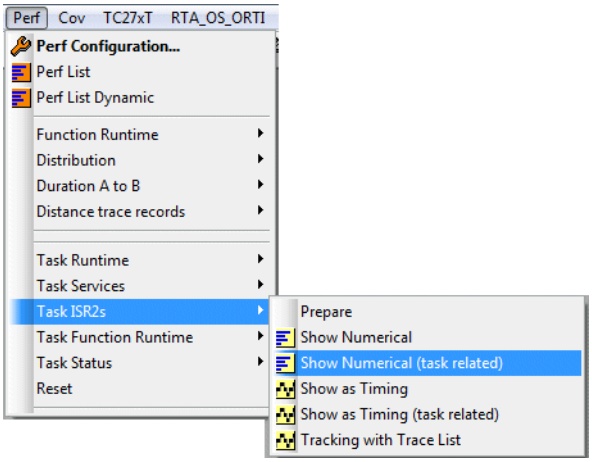




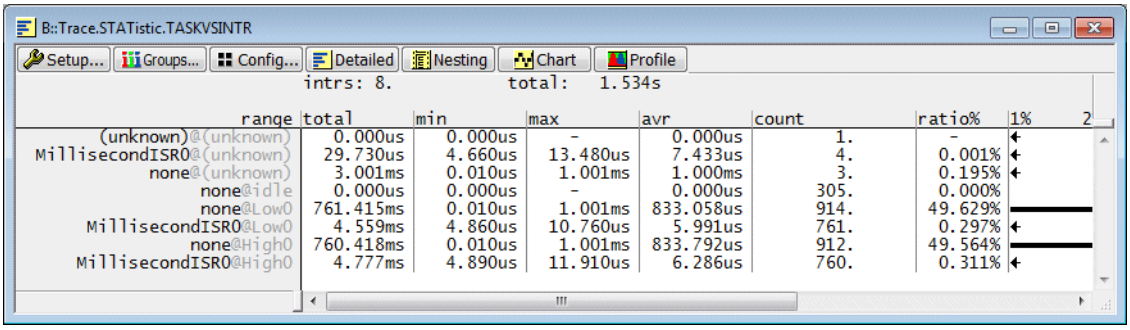
Trace.Chart.TASKINTR Time-chart of ISR2s.

Exporting Task Switches and ISR2

The following commands allow to perform a statistical analysis of the OSEK interrupt service routines related to the active tasks, if you export task switch and ISR2 information.

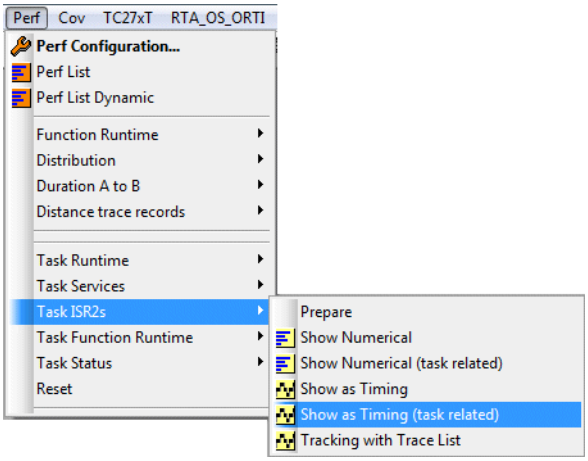


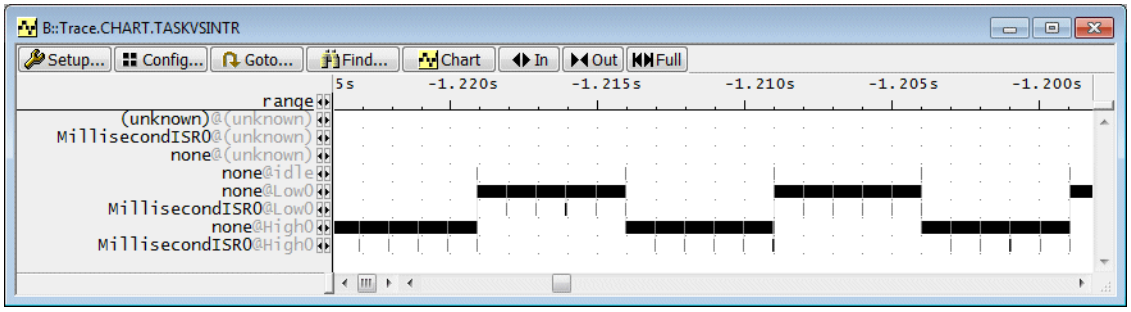
ISR2 information that was generated before the first task information is assigned to the @(unknown) task



range	total	min	max	avr	count	ratio%	1%
(unknown)@(unknown)	0.000us	0.000us	-	0.000us	1.	-	
MillisecondISR0@(unknown)	29.730us	4.660us	13.480us	7.433us	4.	0.001%	
none@(unknown)	3.001ms	0.010us	1.001ms	1.000ms	3.	0.195%	
none@idle	0.000us	0.000us	-	0.000us	305.	0.000%	
none@Low0	761.415ms	0.010us	1.001ms	833.058us	914.	49.629%	
MillisecondISR0@Low0	4.559ms	4.860us	10.760us	5.991us	761.	0.297%	
none@High0	760.418ms	0.010us	1.001ms	833.792us	912.	49.564%	
MillisecondISR0@High0	4.777ms	4.890us	11.910us	6.286us	760.	0.311%	

Trace.STATistic.TASKVSINTR Task-related statistic on interrupt service routines





Trace.Chart.TASKVSINTR	Time-chart on task related interrupt service routines

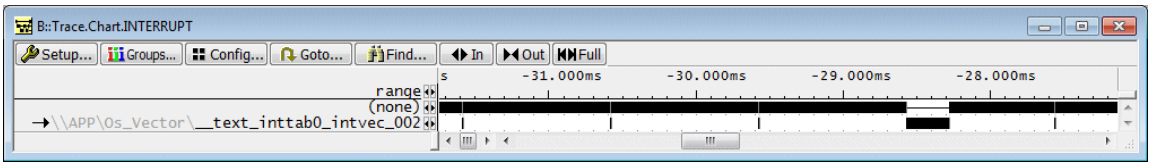
Exporting Task Switches and all Instructions

General setup:

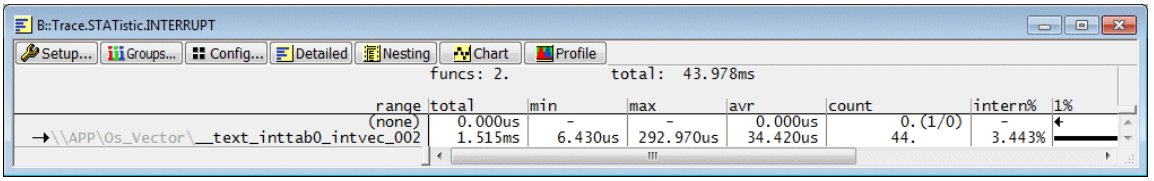
```
Break.Set TASK.CONFIG(magic) /Write /TraceData

; advise TRACE32 to regard the time between interrupt entry
; and exit as function
Trace.STATistic.InterruptIsFunction ON
```

Statistic Analysis of Interrupts

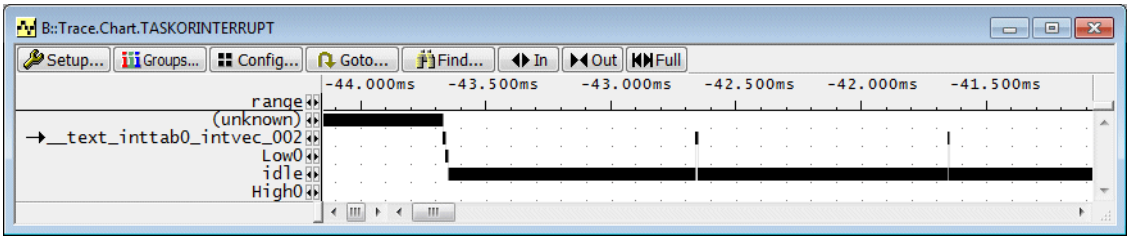
Trace.Chart.INTERRUPT Interrupt time chart



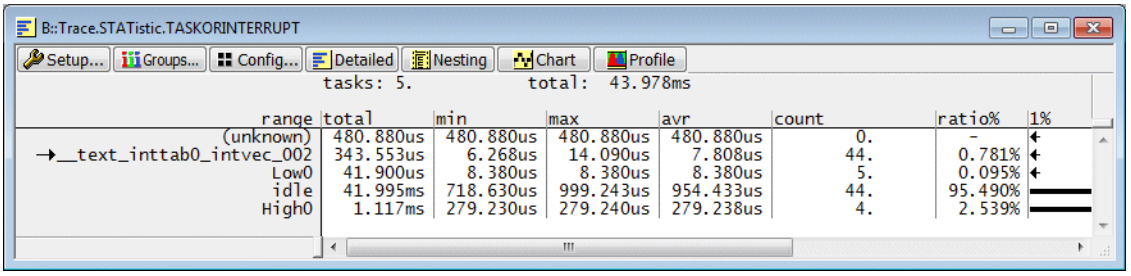
Trace.STATistic.INTERRUPT Interrupt statistic



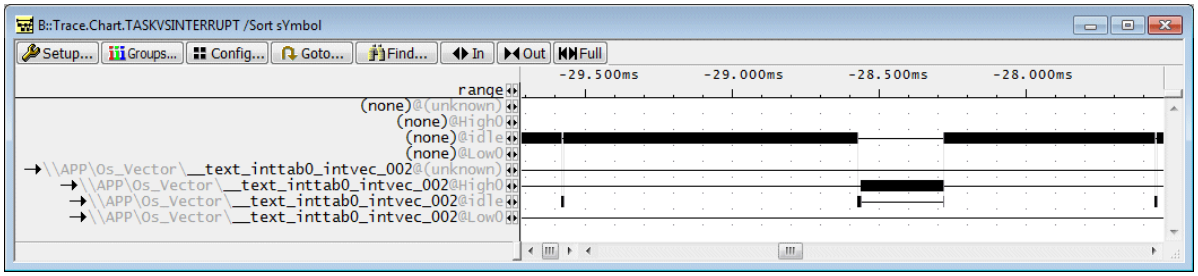
Trace.Chart.TASKORINTERRUPT Time chart of interrupts and tasks



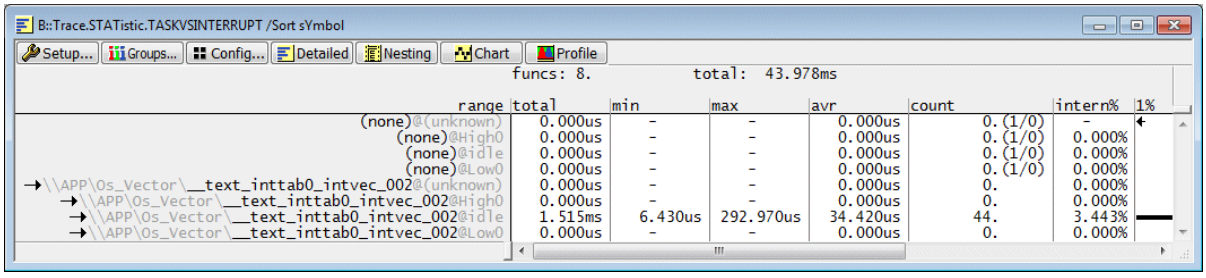
Trace.STATistic.TASKORINTERRUPT Statistic of interrupts and tasks



Trace.Chart.TASKVSINTERRUPT Time chart interrupts, task-related



Trace.STATistic.TASKVSINTERRUPT Statistic of interrupts, task-related



Belated Trace Analysis (OS)

The TRACE32 Instruction Set Simulator can be used for a belated OS-aware trace evaluation. To set up the TRACE32 Instruction Set Simulator for belated OS-aware trace evaluation proceed as follows:

1. Save the trace information for the belated evaluation to a file.

```
Trace.SAVE belated__orti.ad
```

2. Set up the TRACE32 Instruction Set Simulator for a belated OS-aware trace evaluation (here OSEK on a TC277TE):

```
SYStem.CPU TC277TE                          ; select the target CPU

SYStem.Up                                   ; establish the
                                            ; communication between
                                            ; TRACE32 and the TRACE32
                                            ; Instruction Set
                                            ; Simulator

Trace.LOAD belated_orti.ad                  ; load the trace file

Data.Load.Elf my_app.out                    ; load the symbol and
                                            ; debug information

TASK.ORTI my_orti.ort                       ; load the ORTI file

Trace.List List.TASK DEFault                ; display the trace
                                            ; listing
```

Enable an OS-aware Tracing (Not-Supported OS)

If you use an OS that is not supported by Lauterbach you can use the “simple” awareness to configure your debugger for OS-aware tracing.

Current information on the “simple” awareness can be found in ~~/demo/kernel/simple/readme.txt.

Each operating system has a variable that contains the information which task is currently running. This variable can hold a task ID, a pointer to the task control block or something else that is unique for each task.

Use the following command to inform TRACE32 about this variable:

```
TASK.CONFIG ~~/demo/kernel/simple/simple.t32 <var> Var.SIZEOF(<var>)
```

If current_thread is the name of your variable the command would be as follows:

```
TASK.CONFIG ~~/demo/kernel/simple/simple current_thread \
Var.SIZEOF(current_thread)
```

The OS-aware debugging is easier to perform, if you assign names to your tasks.

TASK.NAME.Set *<task_id> <name>*	Specify a name for your task
TASK.NAME.view	Display all specified names

```
TASK.NAME.Set 0x58D68 "My_Task 1"
```

All cores are controlled by an SMP operating system.

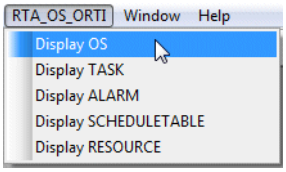
Activate the TRACE32 OS Awareness (Supported OS)

Since most users use an OSEK operating system this is taken as an example here. Setup command:

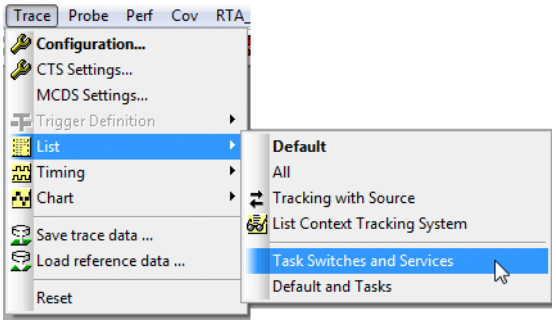
TASK.ORTI *<orti_file>*	Load the ORTI file

Loading the ORTI file results in the following:

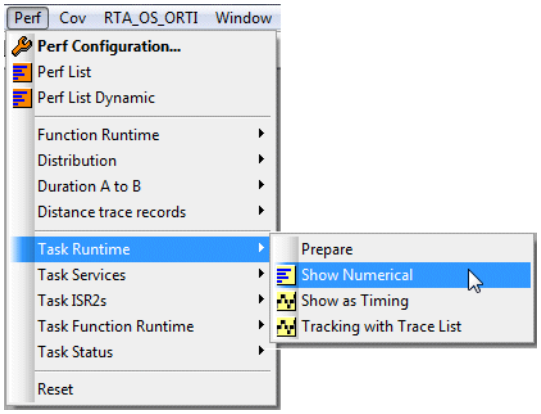
- Symbolic debugging of the OSEK OS is possible. Debug commands are provided via an ORTI menu.



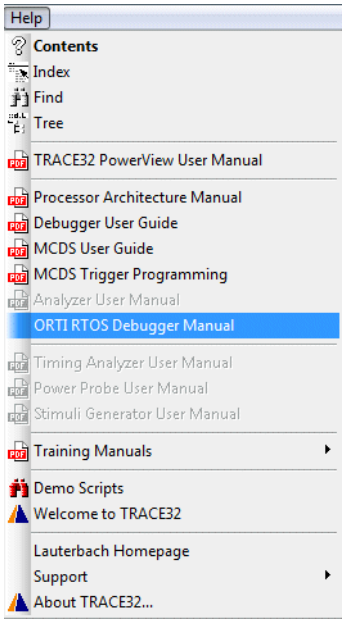
- The **Trace** menu is extended for OS-aware trace display.



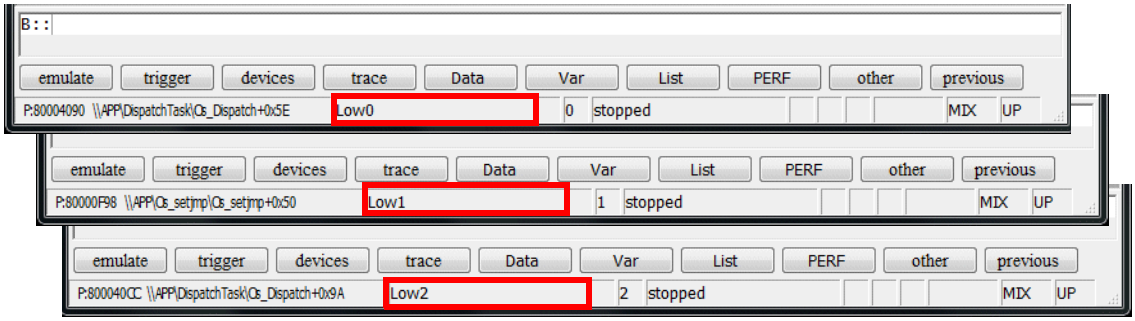
- The **Perf** menu is extended for OS-aware profiling.



- The manual of the OS Awareness for OSEK/**ORTI** is added to the **Help** menu.



- The name of the current task is displayed in the **Task** field of the TRACE32 state line.



Exporting the Task Switches

An SMP operating system has one variable per core that contains the information which task is currently running. This variable can hold a task ID, a pointer to the task control block or something else that is unique for each task.

MCDS can be configured to generate Write Data Trace Messages when a write accesses to these variables occur.

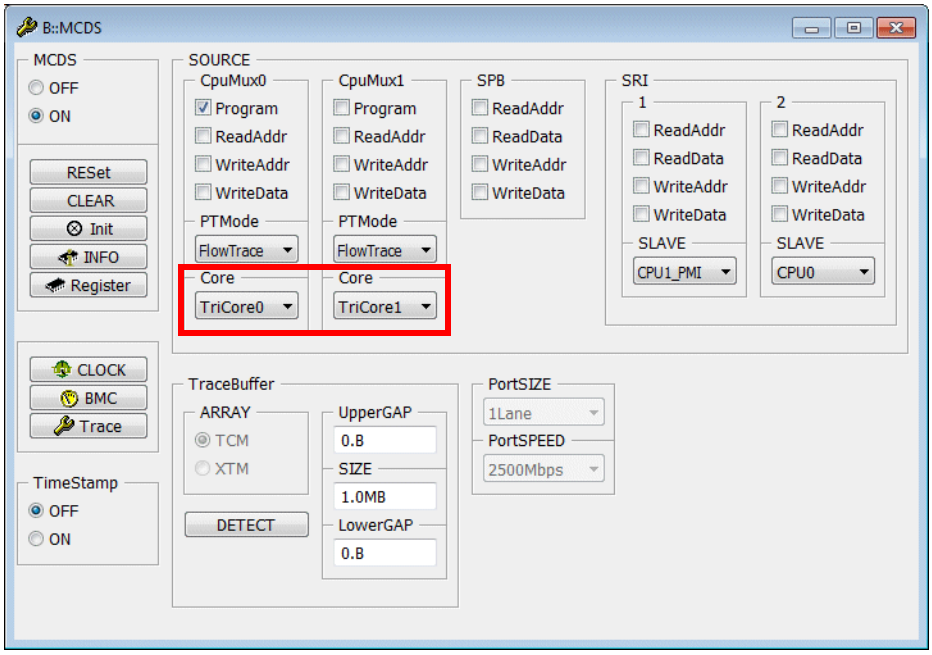
The addresses of these variables are provided by the TRACE32 functions **TASK.CONFIG(magic[***<core>***])**.

```
PRINT TASK.CONFIG(magic[0])           ; print the address that holds
                                      ; the task identifier for
                                      ; TC 1.6.1 CPU0

PRINT TASK.CONFIG(magic[1])           ; print the address that holds
                                      ; the task identifier for
                                      ; TC 1.6.1 CPU1

PRINT TASK.CONFIG(magic[2])           ; print the address that holds
                                      ; the task identifier for
                                      ; TC 1.6.1 CPU2
```

Example: Advise the Processor Observation Blocks to generate trace messages only on task switches.

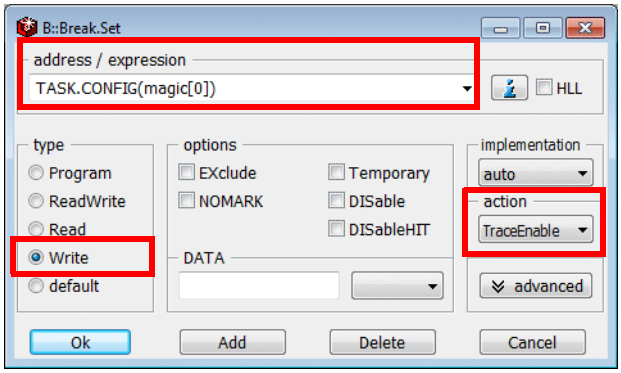
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores under debug:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write accesses to TASK.CONFIG(magic[0]) and TASK.CONFIG(magic[1]).
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. **Configure the trace multiplexer.**

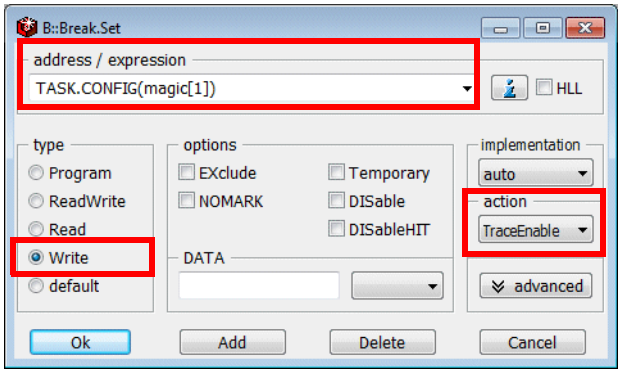



```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the events.

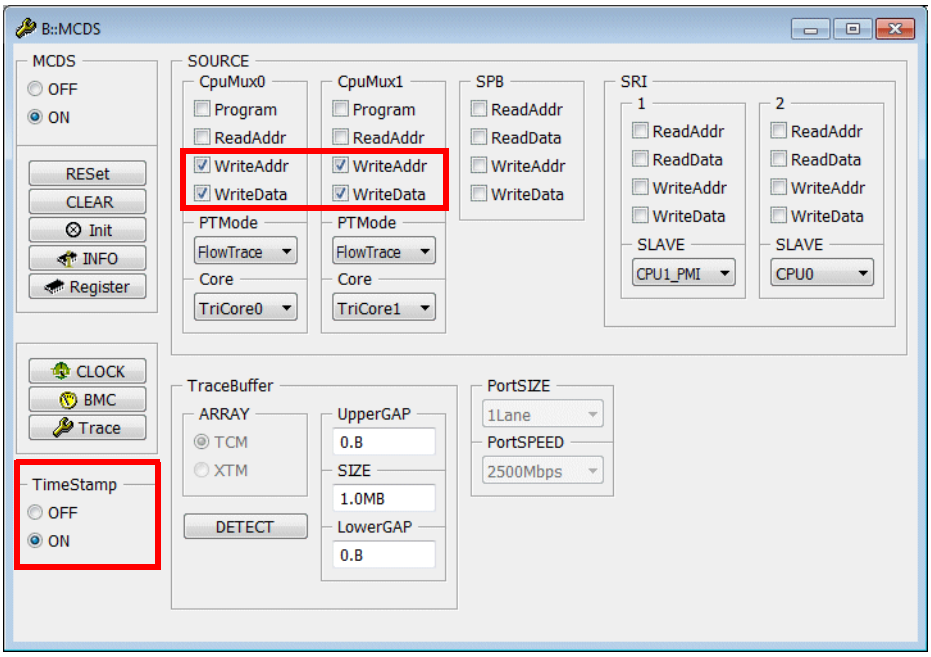



```
Break.Set TASK.CONFIG(magic[0]) /Write /TraceEnable
```




```
Break.Set TASK.CONFIG(magic[1]) /Write /TraceEnable
```

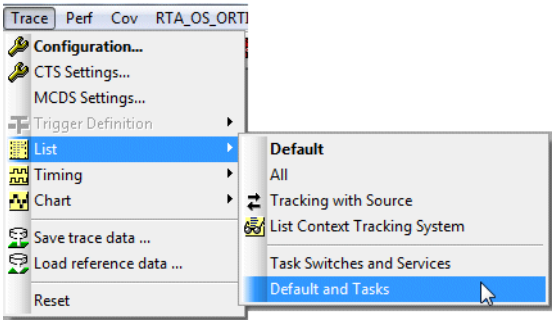
3. **Configure which trace messages are generated.**

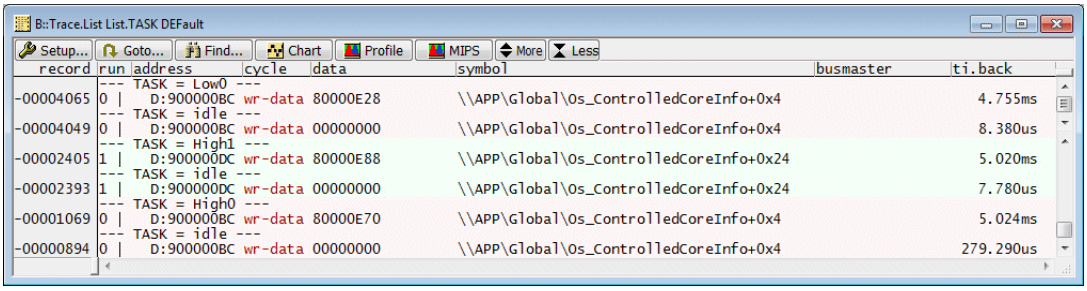



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON

MCDS.SOURCE.Set CpuMux1.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU1
MCDS.SOURCE.Set CpuMux1.WriteData ON
```

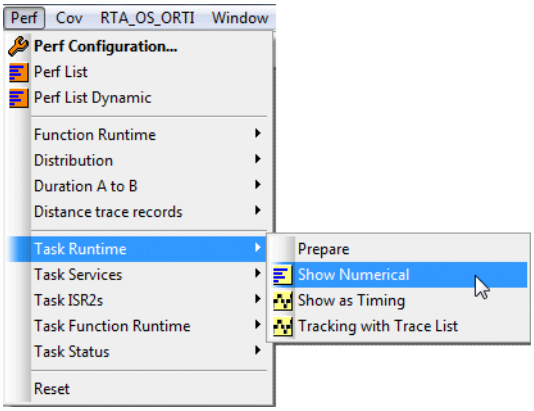
4. **Start the program execution and stop it.**

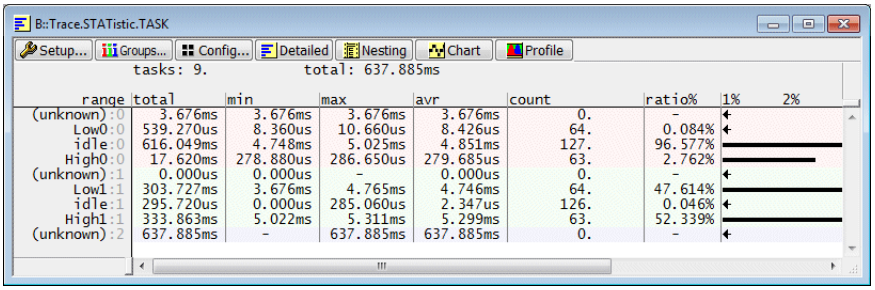
5. Display the result.

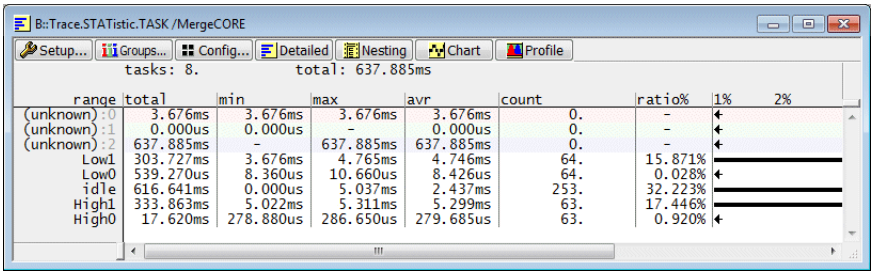




The following two commands perform a statistical analysis of the task switches:



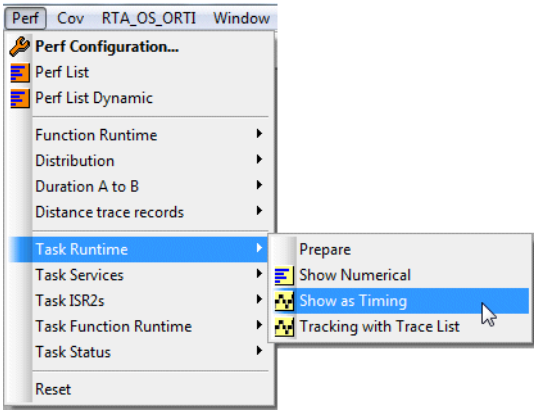


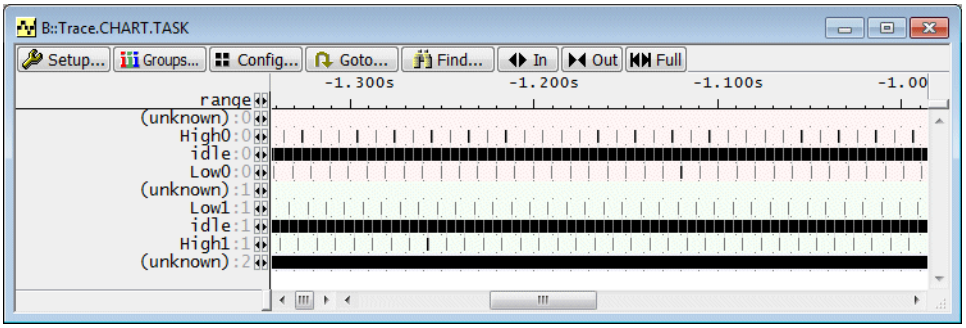


Trace information recorded before the first task switch is assigned to (unknown).

Since no trace information is recorded for TC 1.6.1 CPU2, it stays (unknown) for the total recording time.

Trace.STATistic.TASK [**/SplitCORE**]	Numeric task run-time analysis - split the result per core
Trace.STATistic.TASK [**/MergeCORE**]	Numeric task run-time analysis - merge the results of all cores





Trace.Chart.TASK [**/SplitCORE**] Time-chart of tasks - split the result per core

The ORTI file may also provide means to analyze the time in task service routines.

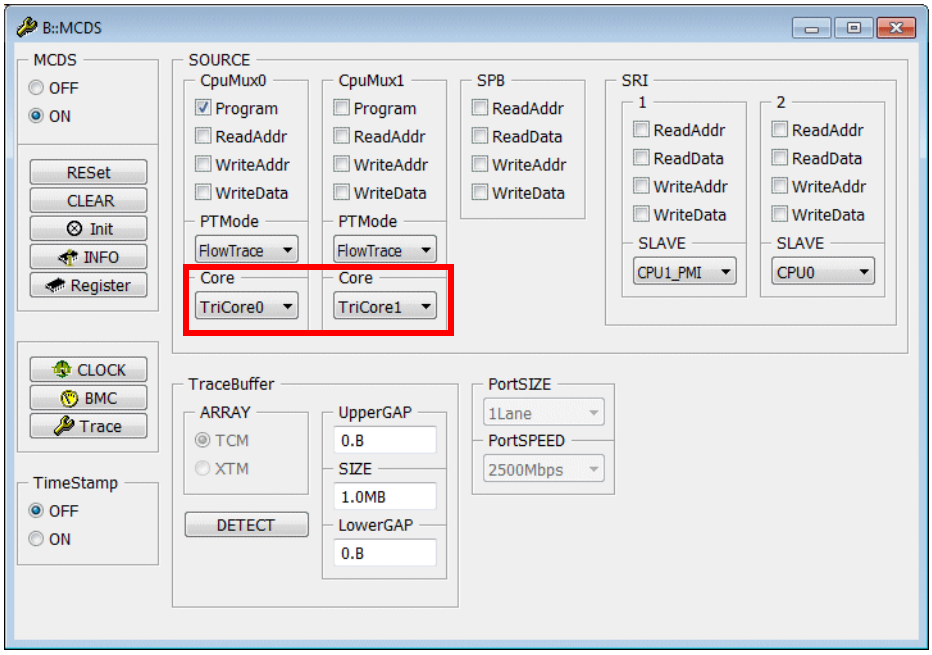
TASK.CONFIG(magic_service[<*core*>**])** is the name of the TRACE32 function that is used for this purpose.

```
PRINT TASK.CONFIG(magic_service[0])        ; print the address that holds
                                           ; the task service table for
                                           ; TC 1.6.1 CPU0

PRINT TASK.CONFIG(magic_service[1])        ; print the address that holds
                                           ; the task service table for
                                           ; TC 1.6.1 CPU1

PRINT TASK.CONFIG(magic_service[2])        ; print the address that holds
                                           ; the task service table for
                                           ; TC 1.6.1 CPU2
```

Example: Advise the Processor Observation Blocks to generate trace messages only on write accesses to the service tables.

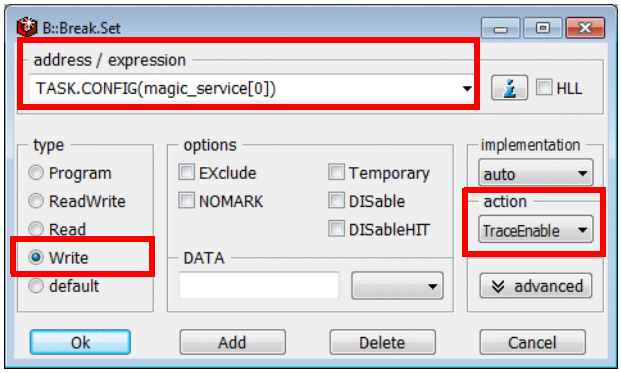
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores under debug:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write accesses to TASK.CONFIG(magic_service[0]) and TASK.CONFIG(magic_service[1]).
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. **Configure the trace multiplexer.**

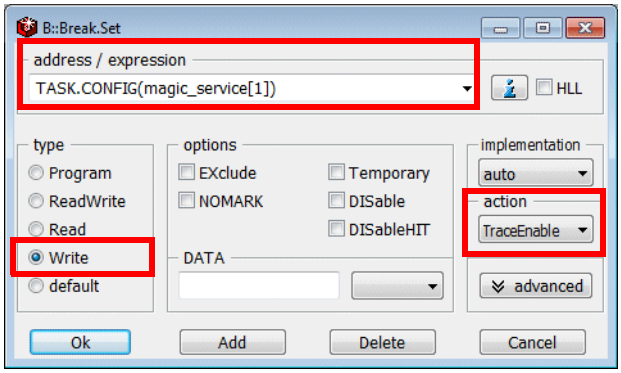



```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the events.

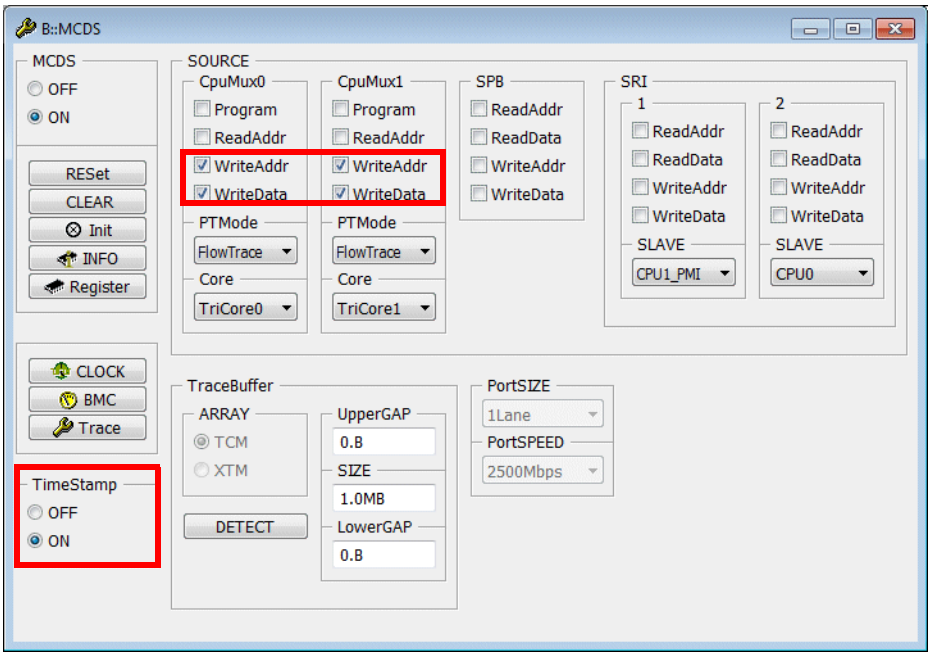



```
Break.Set TASK.CONFIG(magic_service[0]) /Write /TraceEnable
```




```
Break.Set TASK.CONFIG(magic_service[1]) /Write /TraceEnable
```

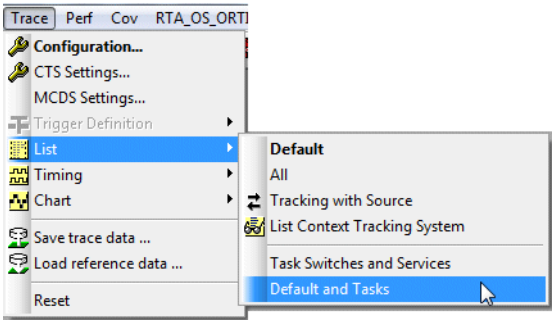
3. Configure which trace messages are generated.

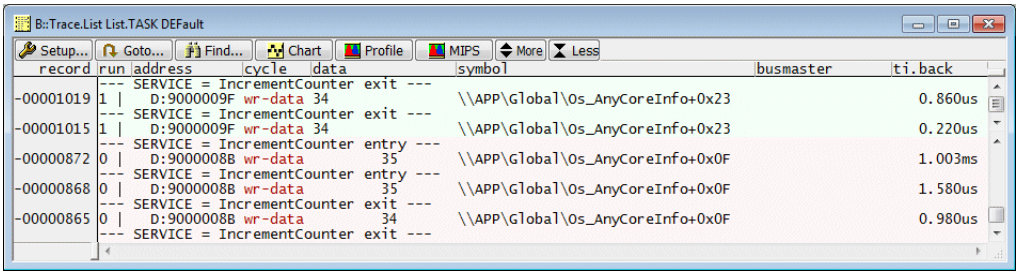



```
MCDS.TimeStamp ON                       ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program         ; disable Instruction Pointer
OFF                                     ; Call Messages for
                                        ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program         ; disable Instruction Pointer
OFF                                     ; Call Messages for
                                        ; TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux0.WriteAddr       ; enable Write Data Trace
ON                                      ; Messages for TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux0.WriteData
ON

MCDS.SOURCE.Set CpuMux1.WriteAddr       ; enable Write Data Trace
ON                                      ; Messages for TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux1.WriteData
ON
```

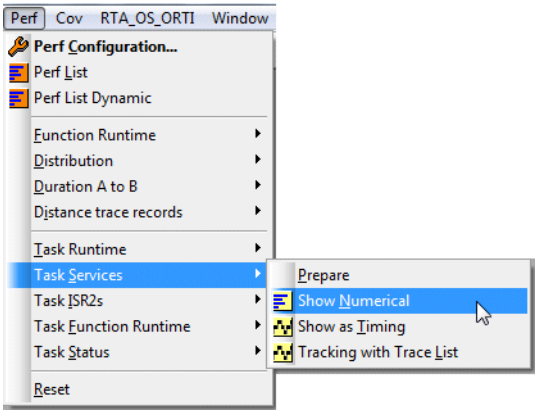
4. Start the program execution and stop it.

5. Display the result.

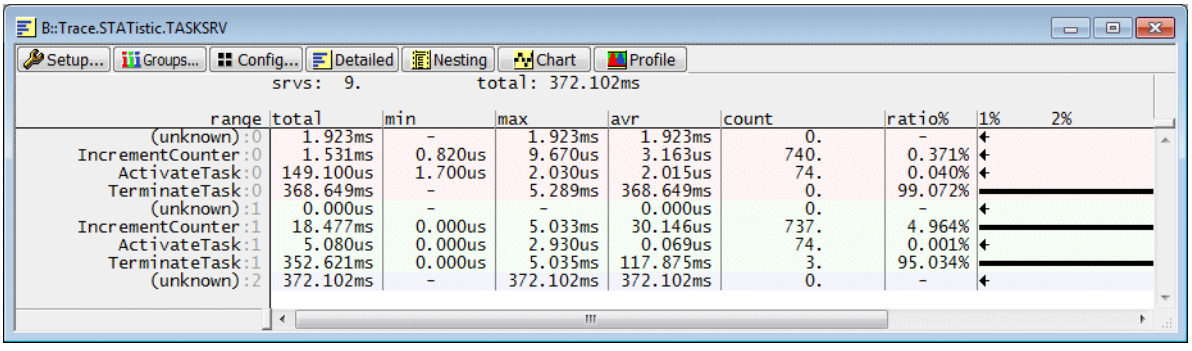




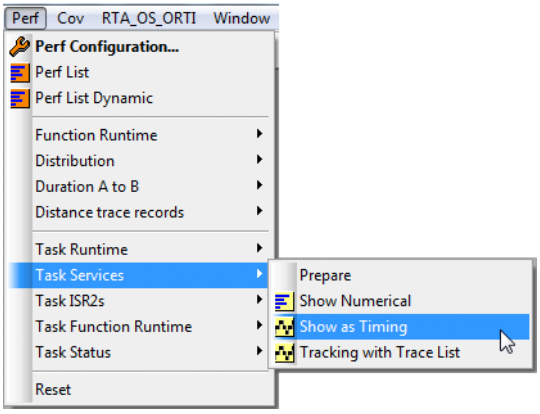
The following two commands perform a statistical analysis of the time in task service routines:

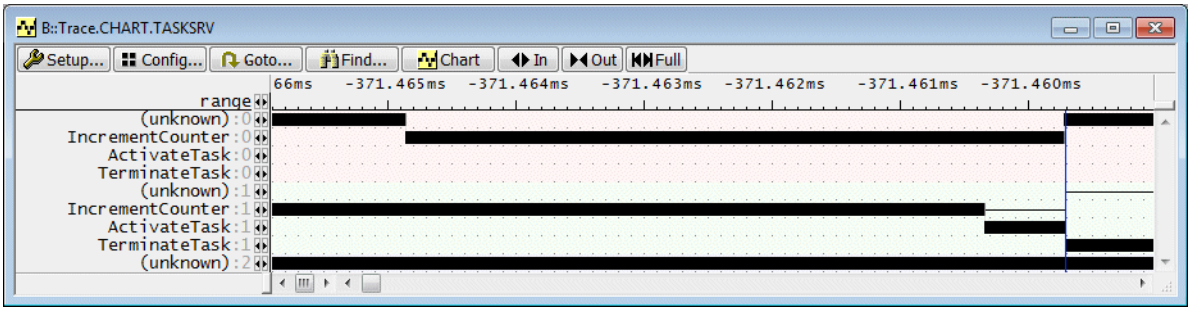


(unknown) represents the time in which the core is not in an OSEK service routine



Trace.STATistic.TASKSRV [**/SplitCORE**]	Numeric analysis of task services - split the result per core.





Trace.Chart.TASKSRV [**/SplitCORE**]	Time-chart of task services - split the result per core.

Exporting ISR2 (OSEK Interrupt Service Routines)

The ORTI file may also provide means to analyze the time in interrupt service routines.

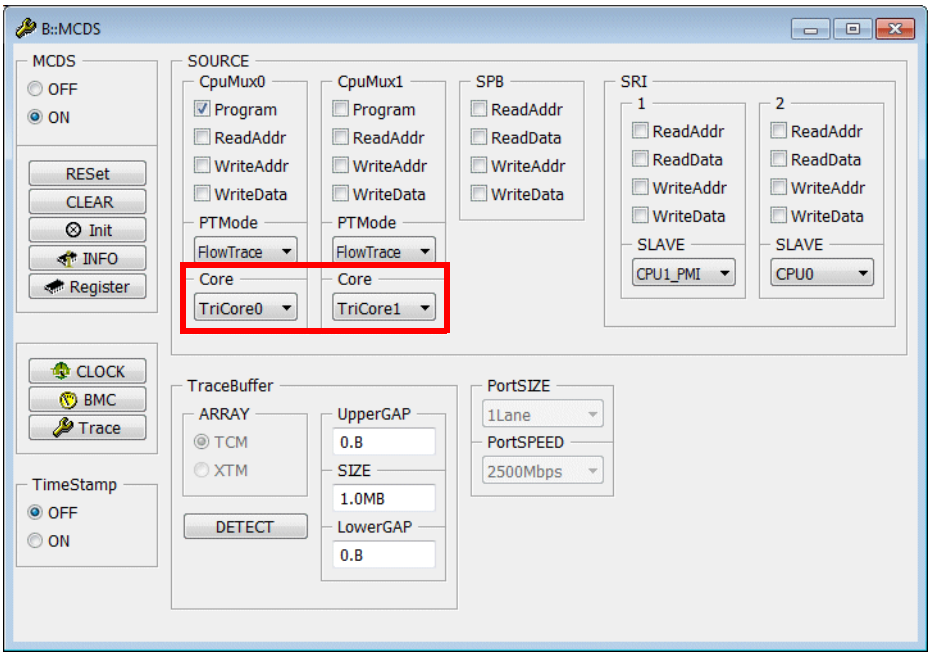
TASK.CONFIG(magic_isr2[<*core*>**])** is the name of the TRACE32 function that is used for this purpose.

```
PRINT TASK.CONFIG(magic_isr2[0])          ; print the address that holds
                                          ; the interrupt service table
                                          ; for TC 1.6.1 CPU0

PRINT TASK.CONFIG(magic_isr2[1])          ; print the address that holds
                                          ; the interrupt service table
                                          ; for TC 1.6.1 CPU1

PRINT TASK.CONFIG(magic_isr2[2])          ; print the address that holds
                                          ; the interrupt service table
                                          ; for TC 1.6.1 CPU2
```

Example: Advise the Processor Observation Blocks to generate trace messages only on write accesses to the interrupt service tables.

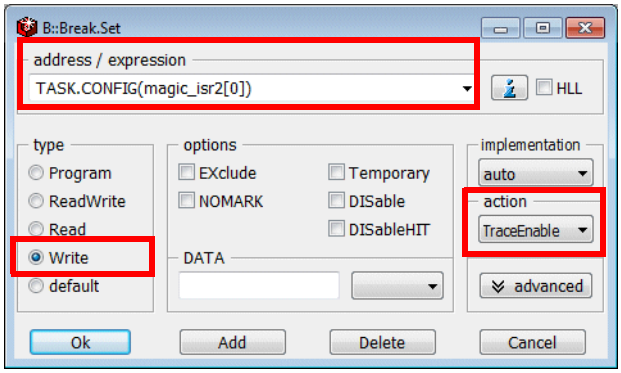
- **System under debug:** SMP system with 3 TriCore cores.
- **Cores under debug:** TC 1.6.1 CPU0 and TC 1.6.1 CPU1.
- **Event of interest:** Write accesses to TASK.CONFIG(magic_isr2[0]) and TASK.CONFIG(magic_isr2[1]).
- **Requested Messages:** Write Data Trace Messages, Timestamp Messages.

1. **Configure the trace multiplexer.**

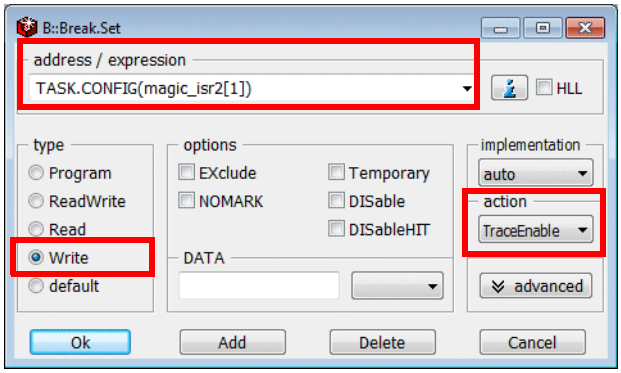



```
MCDS.SOURCE.Set CpuMux0 Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1 Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source
```

2. Specify the events.

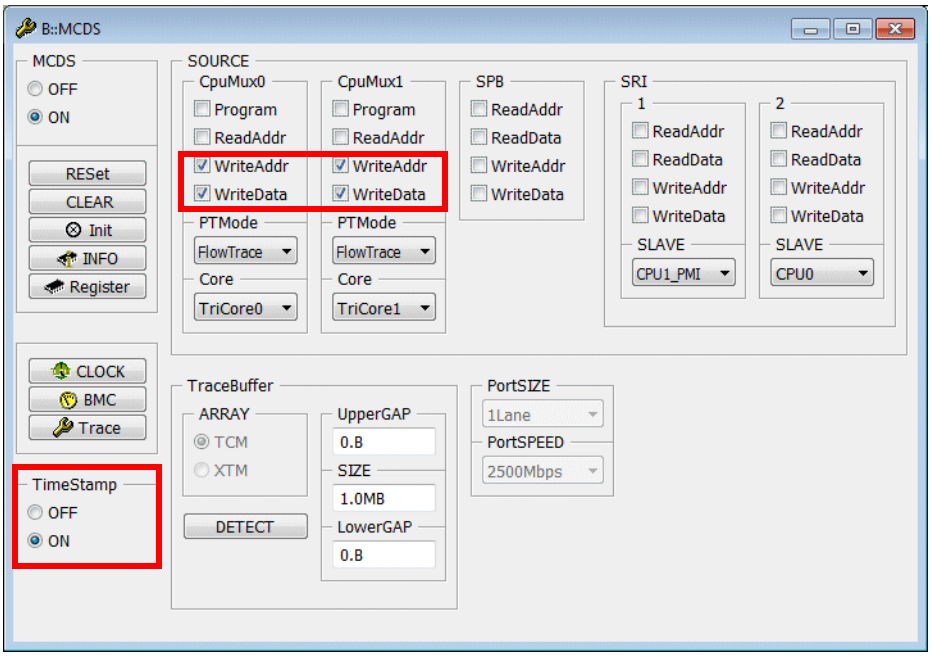



```
Break.Set TASK.CONFIG(magic_service[0]) /Write /TraceEnable
```




```
Break.Set TASK.CONFIG(magic_service[1]) /Write /TraceEnable
```

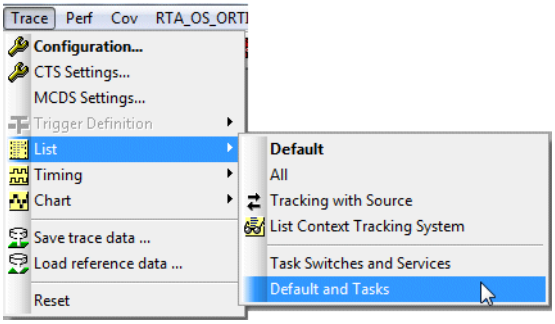
3. Configure which trace messages are generated.

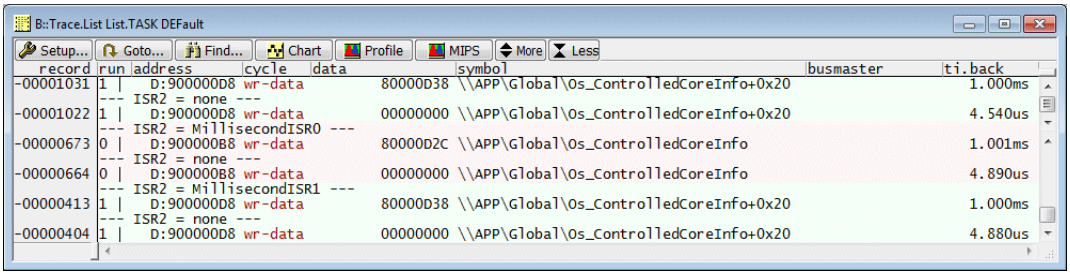



```
MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program OFF        ; disable Instruction
                                           ; Pointer Call Messages for
                                           ; TC 1.6.1 CPU1

MCDS.SOURCE.Set CpuMux0.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU0
MCDS.SOURCE.Set CpuMux0.WriteData ON

MCDS.SOURCE.Set CpuMux1.WriteAddr ON       ; enable Write Data Trace
                                           ; Messages for TC 1.6.1 CPU1
MCDS.SOURCE.Set CpuMux1.WriteData ON
```

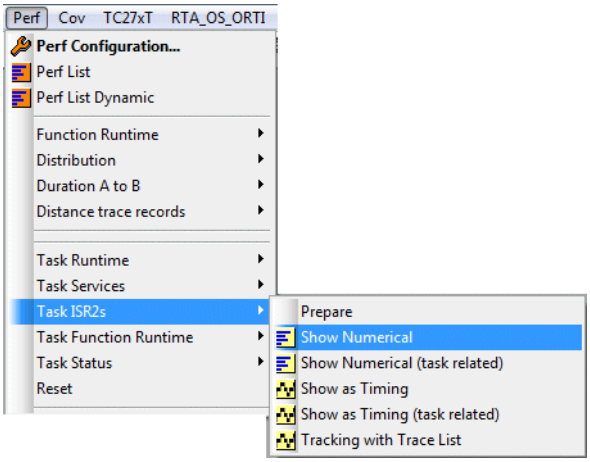
4. Start the program execution and stop it.

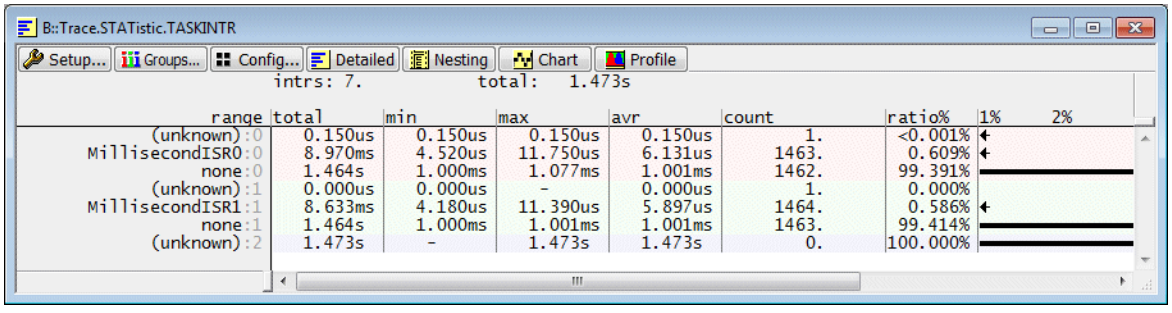
5. Display the result.



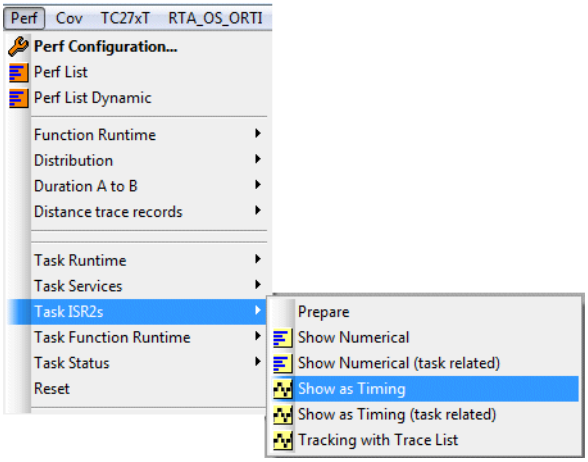


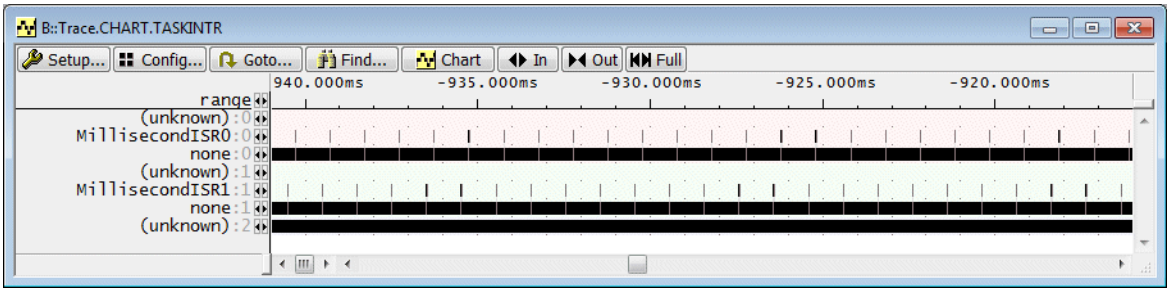
The following two commands perform a statistical analysis of the time in interrupt service routines:





Trace.STATistic.TASKINTR [**/SplitCORE**]	Numeric analysis of ISR2s - split the result per core.

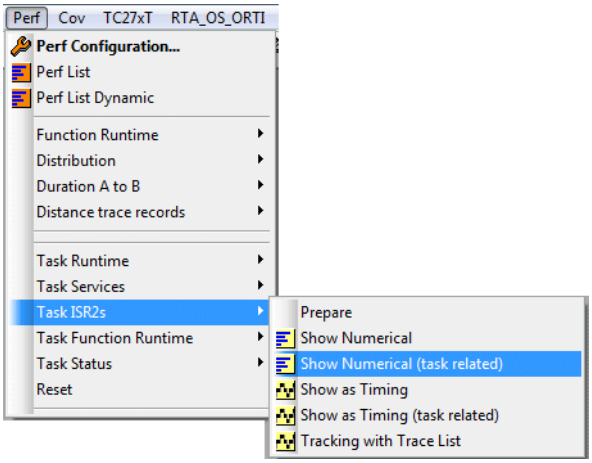




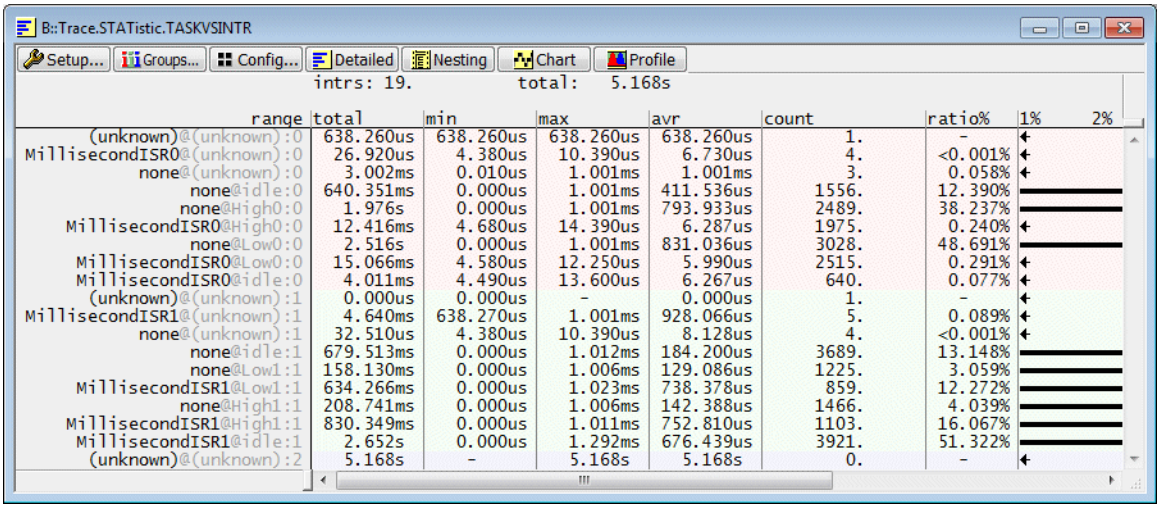
Trace.Chart.TASKINTR [**/SplitCORE**] Time-chart of ISR2s - split the result per core.

Exporting Task Switches and ISR2

The following commands allow to perform a statistical analysis of the OSEK interrupt service routines related to the active tasks, if you export task switch and ISR2 information.

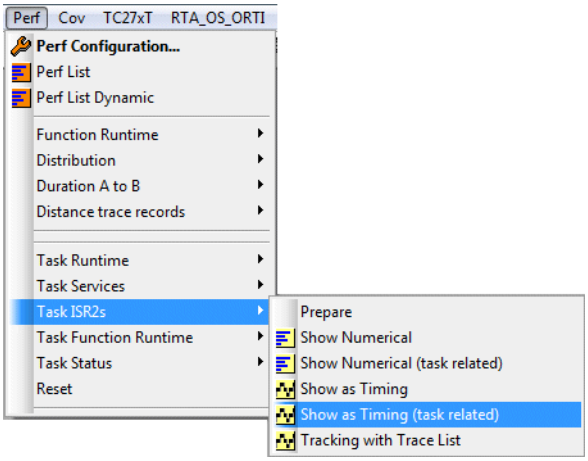


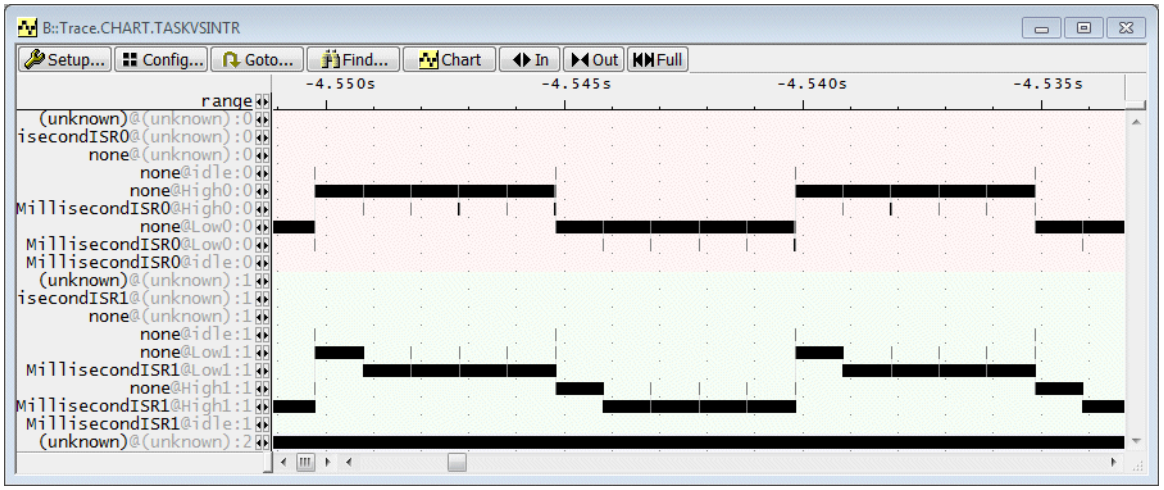
ISR2 information that was generated before the first task information is assigned to the @(unknown) task



range	total	min	max	avr	count	ratio%	1% 2%
(unknown)@(unknown):0	638.260us	638.260us	638.260us	638.260us	1.	-	
MillisecondISR0@(unknown):0	26.920us	4.380us	10.390us	6.730us	4.	<0.001%	
none@(unknown):0	3.002ms	0.010us	1.001ms	1.001ms	3.	0.058%	
none@idle:0	640.351ms	0.000us	1.001ms	411.536us	1556.	12.390%	
none@High0:0	1.976s	0.000us	1.001ms	793.933us	2489.	38.237%	
MillisecondISR0@High0:0	12.416ms	4.680us	14.390us	6.287us	1975.	0.240%	
none@Low0:0	2.516s	0.000us	1.001ms	831.036us	3028.	48.691%	
MillisecondISR0@Low0:0	15.066ms	4.580us	12.250us	5.990us	2515.	0.291%	
MillisecondISR0@idle:0	4.011ms	4.490us	13.600us	6.267us	640.	0.077%	
(unknown)@(unknown):1	0.000us	0.000us	-	0.000us	1.	-	
MillisecondISR1@(unknown):1	4.640ms	638.270us	1.001ms	928.066us	5.	0.089%	
none@(unknown):1	32.510us	4.380us	10.390us	8.128us	4.	<0.001%	
none@idle:1	679.513ms	0.000us	1.012ms	184.200us	3689.	13.148%	
none@Low1:1	158.130ms	0.000us	1.006ms	129.086us	1225.	3.059%	
MillisecondISR1@Low1:1	634.266ms	0.000us	1.023ms	738.378us	859.	12.272%	
none@High1:1	208.741ms	0.000us	1.006ms	142.388us	1466.	4.039%	
MillisecondISR1@High1:1	830.349ms	0.000us	1.011ms	752.810us	1103.	16.067%	
MillisecondISR1@idle:1	2.652s	0.000us	1.292ms	676.439us	3921.	51.322%	
(unknown)@(unknown):2	5.168s	-	5.168s	5.168s	0.	-	

Trace.STATistic.TASKVSINTR [**/SplitCORE**]	Task-related statistic on interrupt service routines - split the result per core





Trace.Chart.TASKVSINTR [**/SplitCORE**]	Time-chart on task related interrupt service routines - split the result per core

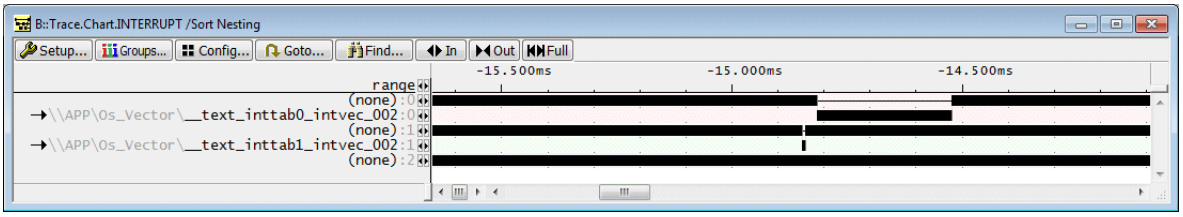
Exporting Task Switches and all Instructions

General setup:

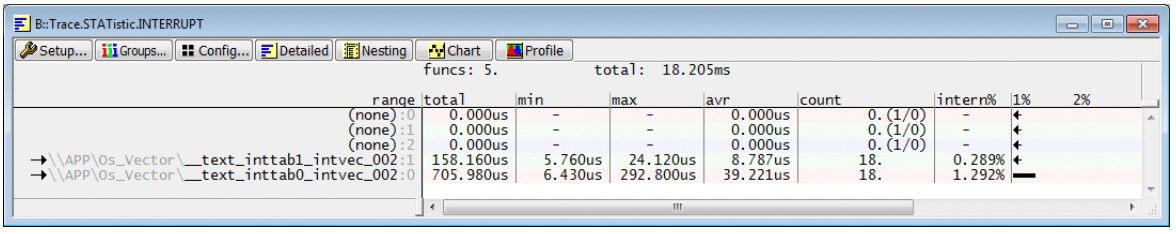
```
Break.Set TASK.CONFIG(magic[0]) /Write /TraceData
Break.Set TASK.CONFIG(magic[1]) /Write /TraceData
; Break.Set TASK.CONFIG(magic[2]) /Write /TraceData

; advise TRACE32 to regard the time between interrupt entry
; and exit as function
Trace.STATistic.InterruptIsFunction ON
```

Statistic Analysis of Interrupts

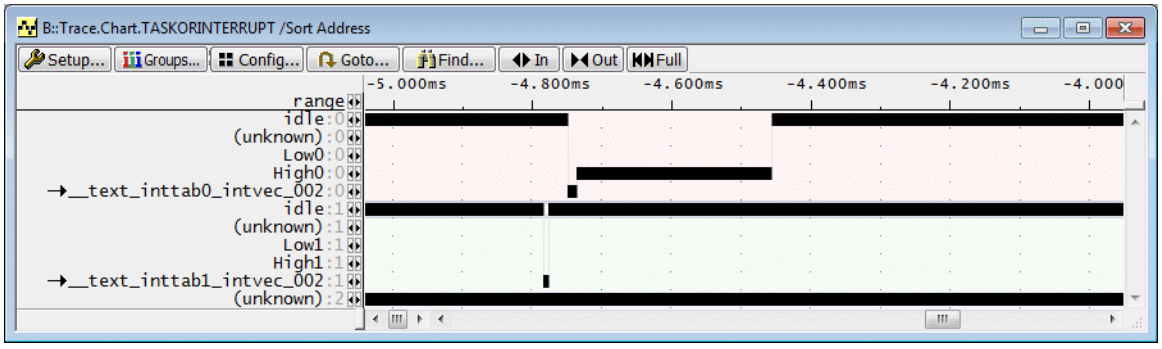
Trace.Chart.INTERRUPT [**/SplitCORE**] Interrupt time chart - split the result per core



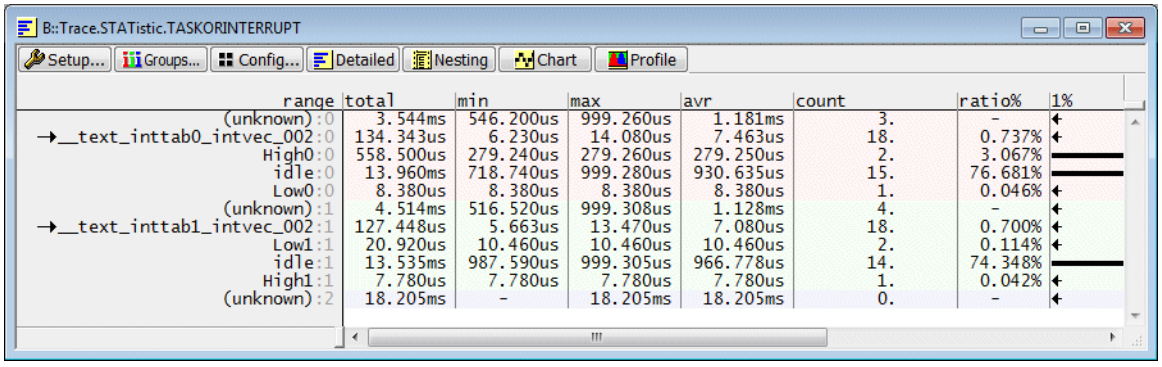
Trace.STATistic.INTERRUPT [**/SplitCORE**] Interrupt statistic - split the result per core



Trace.Chart.TASKORINTERRUPT [**/SplitCORE**] Time chart of interrupts and tasks - split the result per core



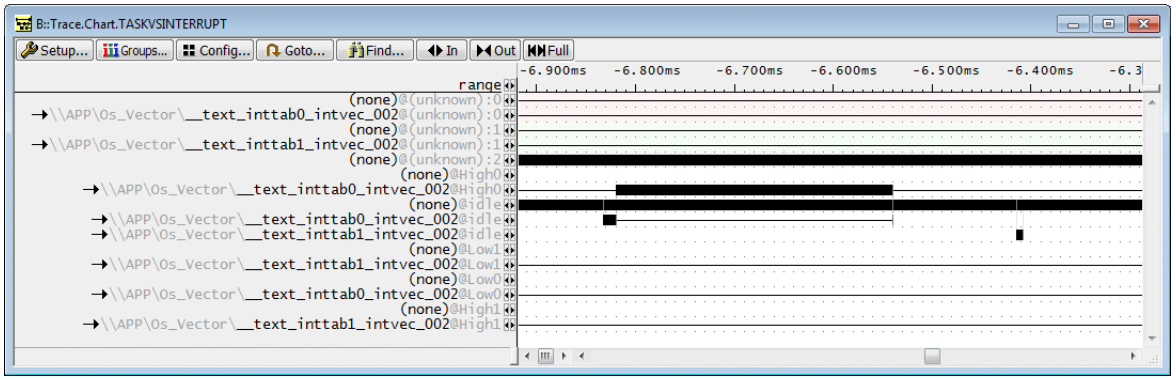
Trace.STATistic.TASKORINTERRUPT [**/SplitCORE**] Statistic of interrupts and tasks - split the result per core



range	total	min	max	avr	count	ratio%	1%
(unknown):0	3.544ms	546.200us	999.260us	1.181ms	3.	-	
→__text_inttab0_intvec_002:0	134.343us	6.230us	14.080us	7.463us	18.	0.737%	
High0:0	558.500us	279.240us	279.260us	279.250us	2.	3.067%	
idle:0	13.960ms	718.740us	999.280us	930.635us	15.	76.681%	
Low0:0	8.380us	8.380us	8.380us	8.380us	1.	0.046%	
(unknown):1	4.514ms	516.520us	999.308us	1.128ms	4.	-	
→__text_inttab1_intvec_002:1	127.448us	5.663us	13.470us	7.080us	18.	0.700%	
Low1:1	20.920us	10.460us	10.460us	10.460us	2.	0.114%	
idle:1	13.535ms	987.590us	999.305us	966.778us	14.	74.348%	
High1:1	7.780us	7.780us	7.780us	7.780us	1.	0.042%	
(unknown):2	18.205ms	-	18.205ms	18.205ms	0.	-	

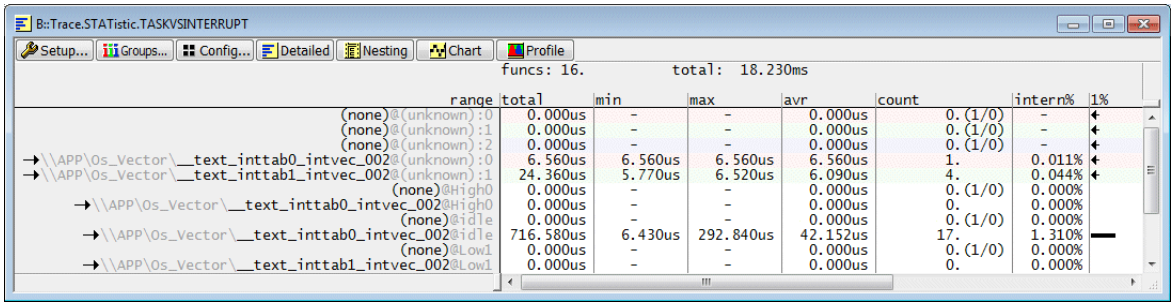
Trace.Chart.TASKVSINTERRUPT

Time chart interrupts, task-related



Trace.STATistic.TASKVSINTERRUPT

Statistic of interrupts, task-related



Belated Trace Analysis (OS)

The TRACE32 Instruction Set Simulator can be used for a belated OS-aware trace evaluation. To set up the TRACE32 Instruction Set Simulator for belated OS-aware trace evaluation proceed as follows:

1. Save the trace information for the belated evaluation to a file.

```
Trace.SAVE belated__orti.ad
```

2. Set up the TRACE32 Instruction Set Simulator for a belated OS-aware trace evaluation (here OSEK on a TC277TE):

```
SYStem.CPU TC277TE                          ; select the target CPU

CORE.ASSIGN 1. 2. 3.                        ; configure the SMP
                                            ; system

SYStem.Up                                   ; establish the
                                            ; communication between
                                            ; TRACE32 and the TRACE32
                                            ; Instruction Set
                                            ; Simulator

Trace.LOAD belated_orti.ad                  ; load the trace file

Data.Load.Elf my_app.out                    ; load the symbol and
                                            ; debug information

TASK.ORTI my_orti.ort                       ; load the ORTI file

Trace.List List.TASK DEFault                ; display the trace
                                            ; listing
```

Function Run-Time Analysis - Basic Concept

Software under Analysis (no OS or OS)

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

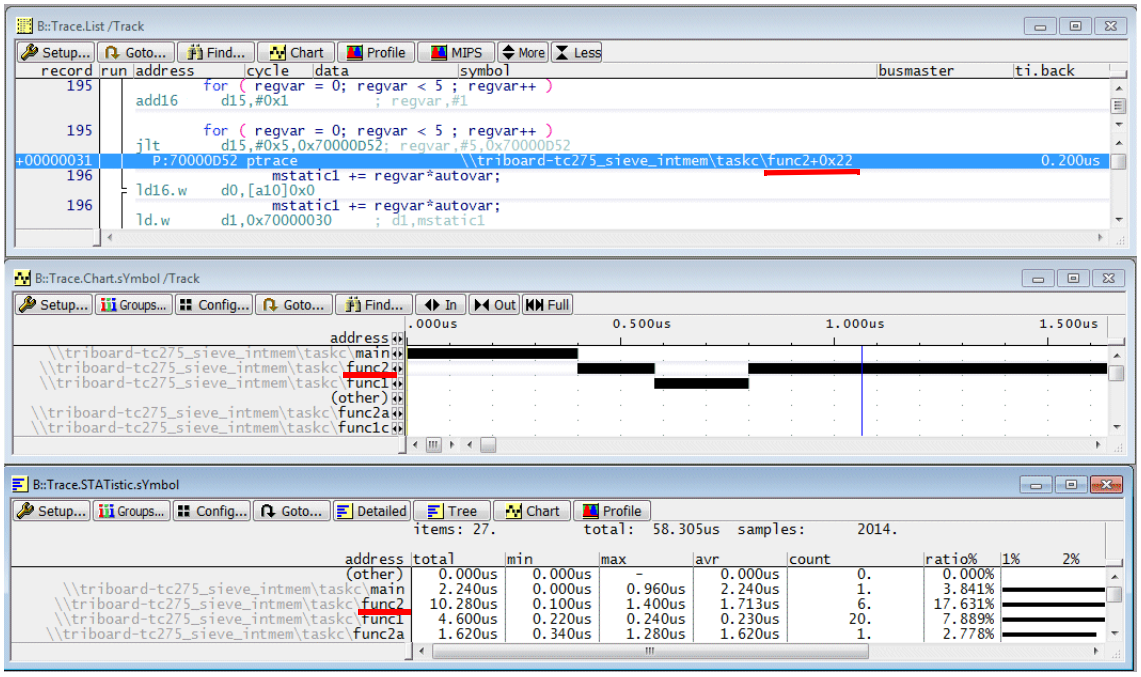
1. Software without operating system (abbreviation: **no OS**)
2. Software with an operating system (abbreviation: **OS**)

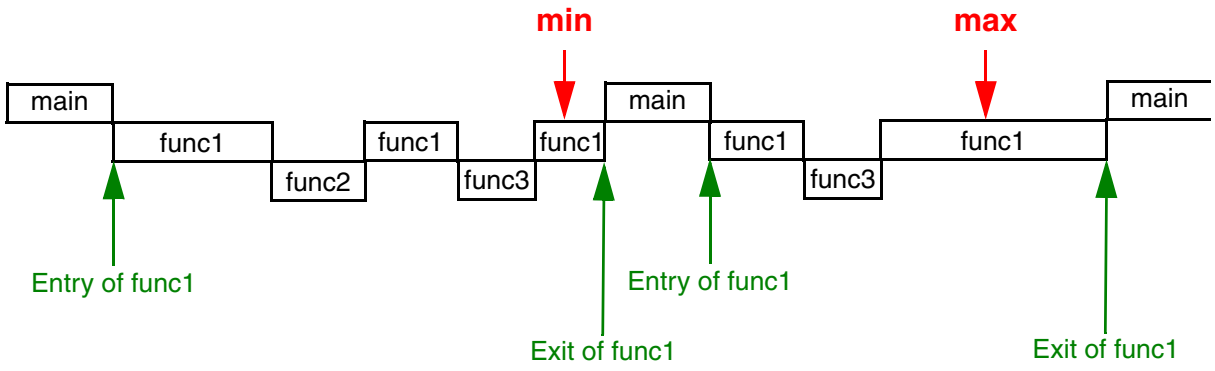
Flat vs. Nesting Analysis

TRACE32 provides two methods to analyze function run-times:

- Flat analysis
- Nesting analysis

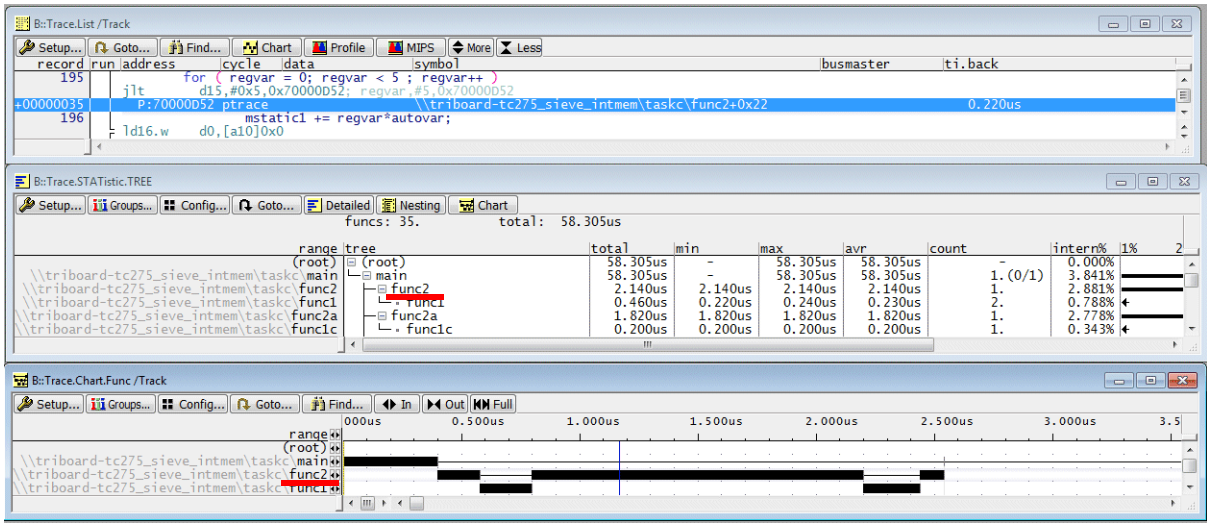
The flat function run-time analysis bases on the symbolic instruction addresses of the trace entries. The time spent by an instruction is assigned to the corresponding function/symbol region.

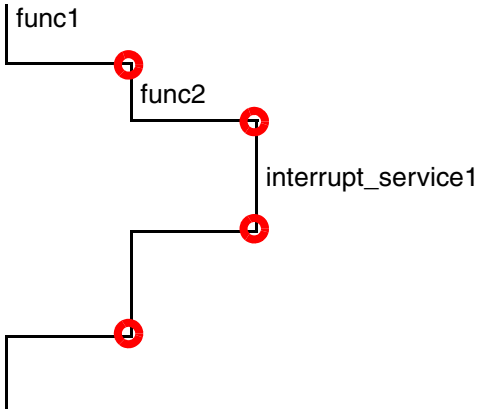




min	shortest time continuously in the address range of the function/symbol region
max	longest time continuously in the address range of the function/symbol region

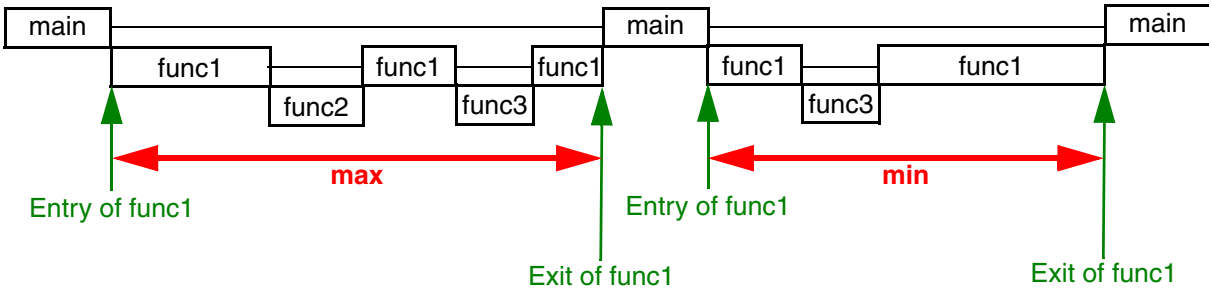
The function nesting analysis analyses only high-level language functions.





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

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



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

Summary

The nesting analysis provides more details on the structure and the timing of the program run, but it is much more sensitive than the flat analysis.

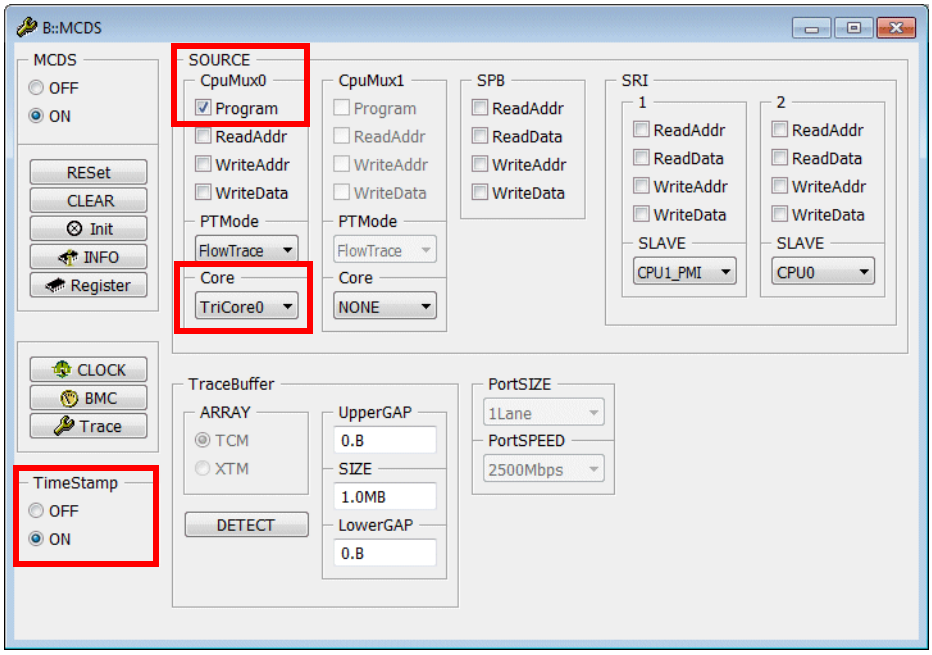
It is recommended to reduce the trace information generated by MCDS to the required minimum.

- To make best use of the available trace memory.

Optimum MCDS Configuration (No OS)

Flat function run-time analysis does **not** require any **data information** if no OS is used. That's why it is recommended to disable the generation of Write and Read Data Trace Messages.

Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.

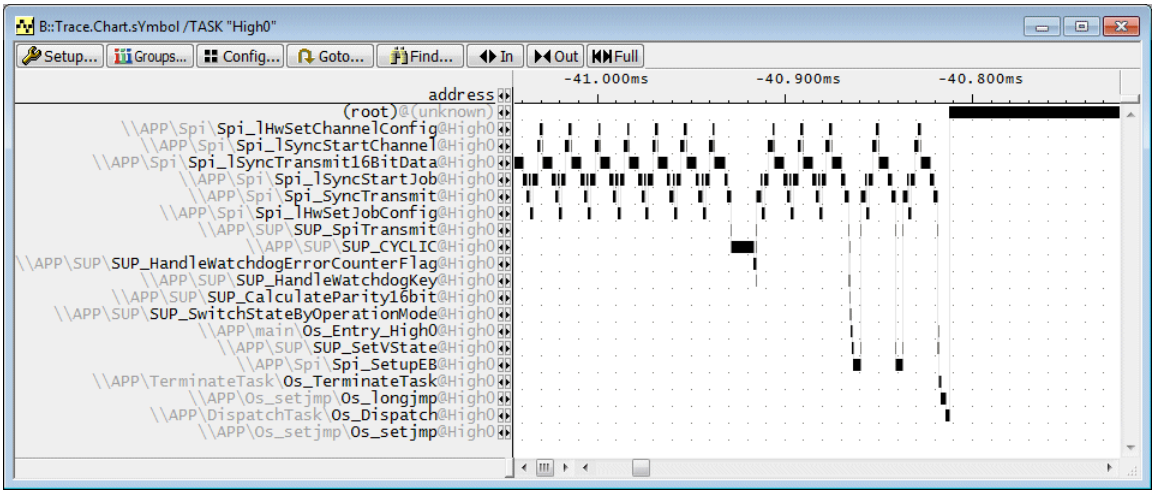



```
MCDS.SOURCE CpuMux0 Core TriCore0          ; enable TC 1.6.1 CPU0 as
                                           ; trace source

MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE CpuMux0 Program ON             ; enable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU0
```

Your function time chart **can** include task information if you advise MCDS to export the instruction flow and task switches. For details refer to the chapter **"OS-Aware Tracing - Single-Core and AMP"**, page 157 of this training.

Function time chart with task information:

```
Trace.Chart.sYmbol /TASK "High0"
```



Command to advise MCDS to export the instruction flow and task switches.

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

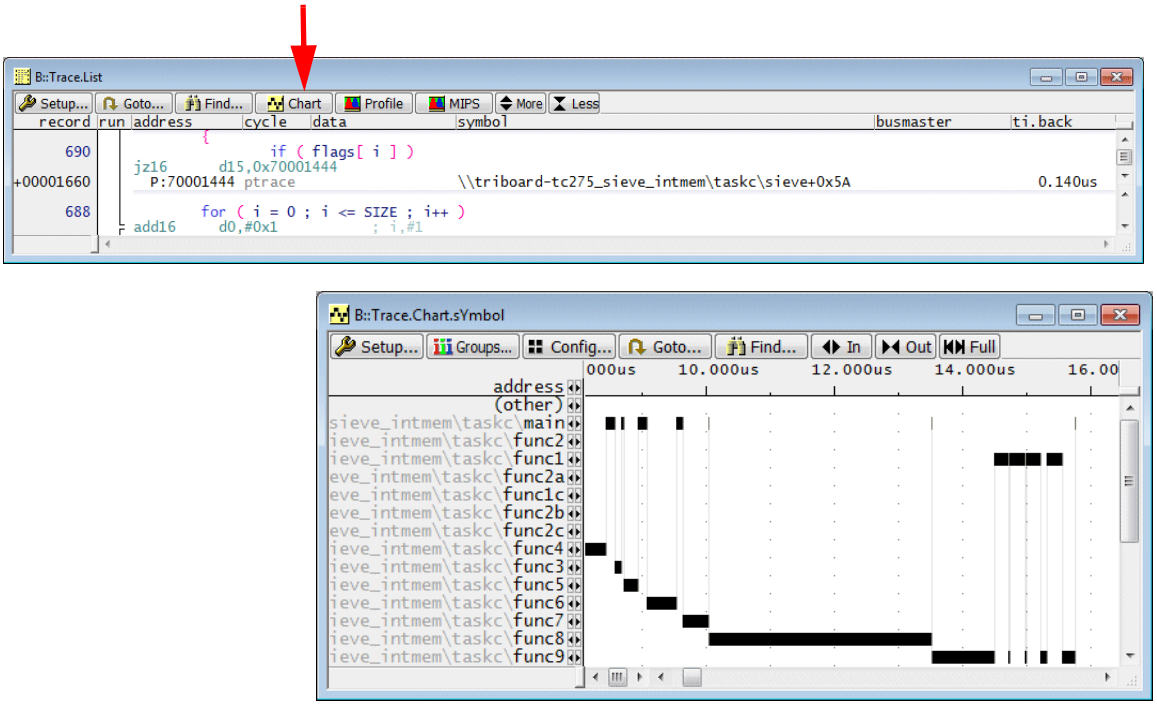
Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.

```
CLOCK.ON

MCDS.TimeStamp ON
```

Function Timing Diagram (no TASK Information)

TRACE32 PowerView provides a timing diagram which shows when the program counter was in which function/symbol range.

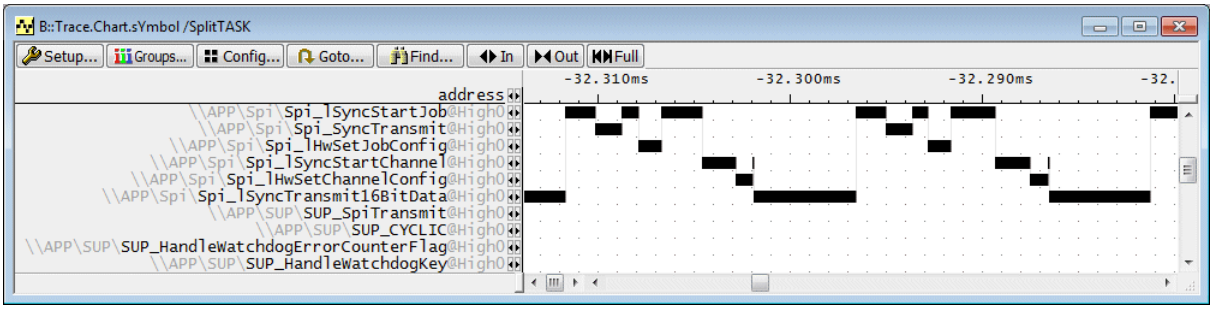
Pushing the **Chart** button in the **Trace.List** window opens a **Trace.Chart.sYmbol** window

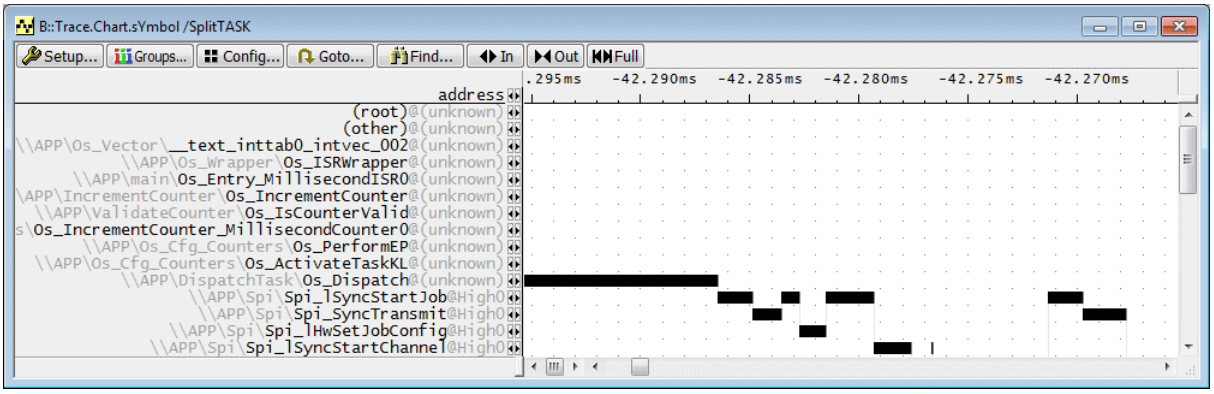


Trace.Chart.sYmbol	Display function time chart (no OS)
Trace.Chart.sYmbol [**/MergeTASK**]	Display function time chart (OS but task information is not of interest)

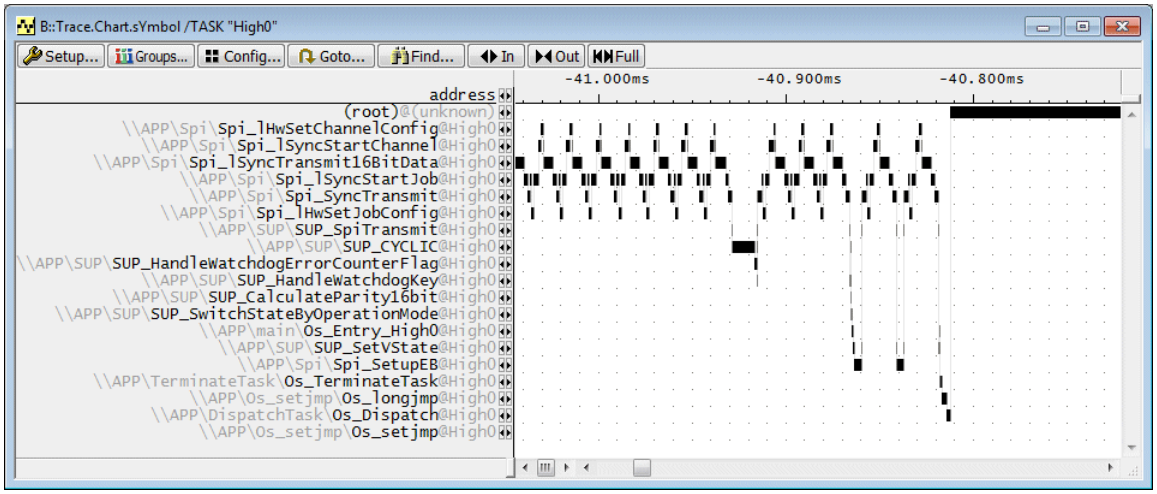
Function Timing Diagram (TASK information)

Trace.Chart.sYmbol /SplitTASK	Display function time chart including task information (OS only)

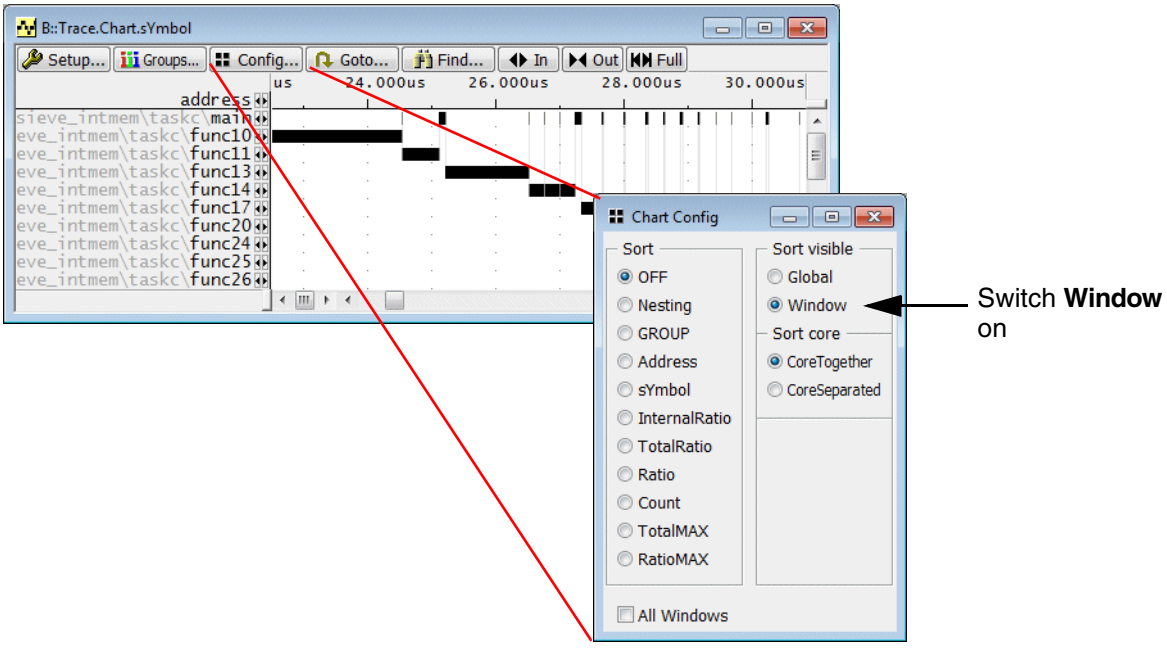




@*<task_name>*	Task name information
@(unknown)	• Function was running before the OS was started • Function was recorded before first task switch information was recorded
(root)@(unknown)	No trace information available.



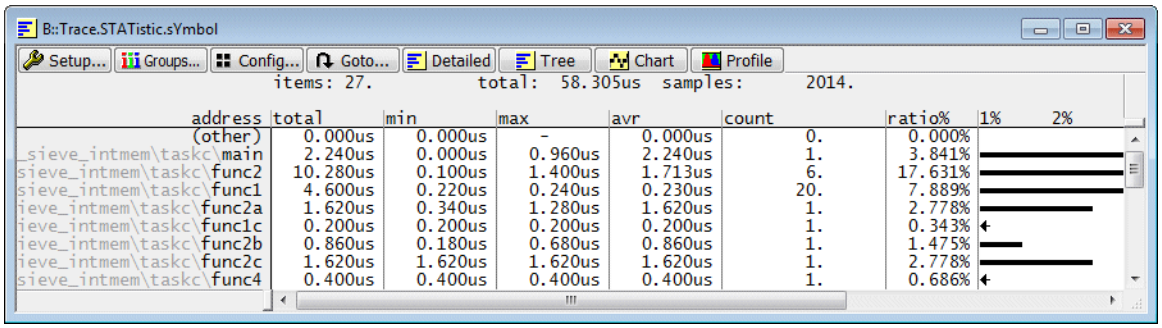
Trace.Chart.sYmbol /TASK *<name>*	Display function time chart for specified task (OS only)



If **Window** in the **Sort visible** field is switched ON in the **Chart Config** dialog, 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.

Numeric Analysis

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

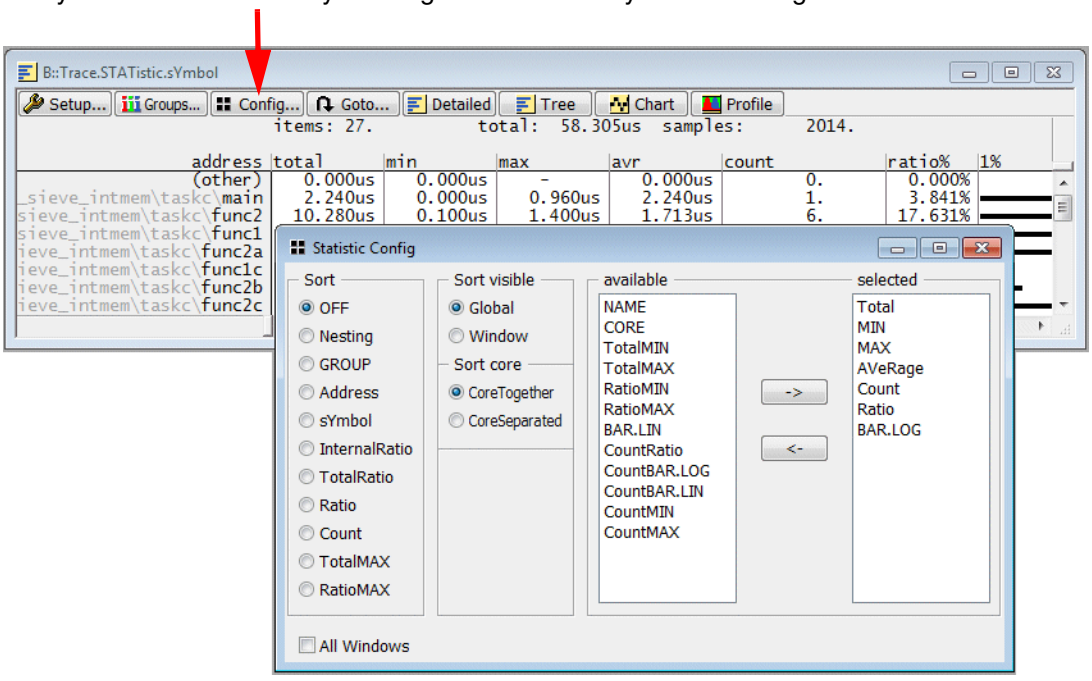


survey	
item	number of recorded functions/symbol regions
total	time period recorded by the trace
samples	total number of recorded changes of functions/symbol regions (instruction flow continuously in the address range of a function/symbol region)

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

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

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



Trace.STATistic.sYmbol	Flat function run-time analysis - numerical display
Trace.STATistic.sYmbol [/MergeTASK]	Flat function run-time analysis (OS but task information is not of interest) - numerical display
Trace.STATistic.sYmbol /SplitTASK	Flat function run-time analysis including task information (OS only) - numerical display
Trace.STATistic.sYmbol /TASK *<name>*	Flat function run-time analysis for specified task (OS only) - numerical display

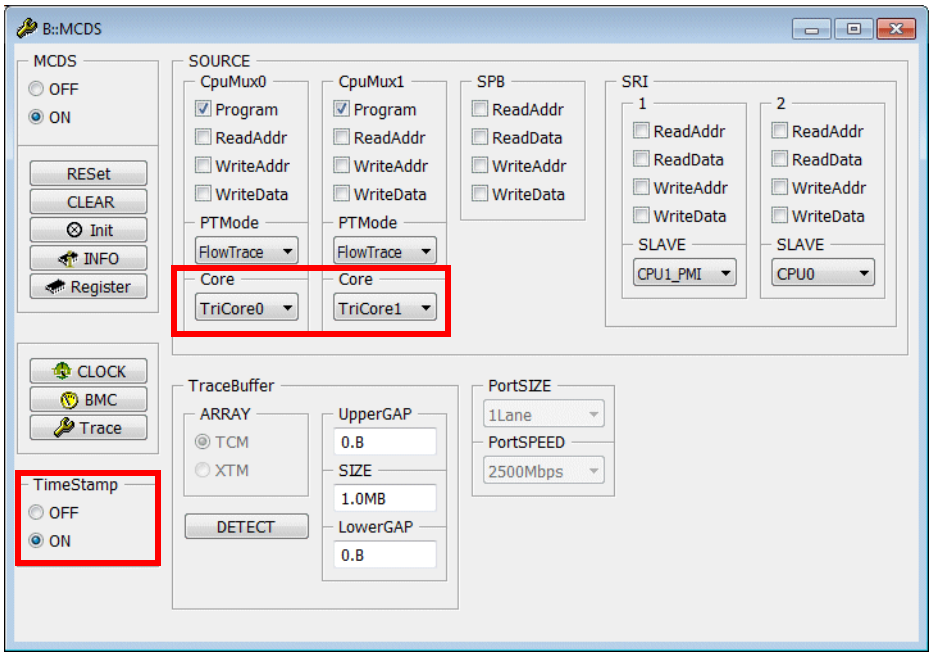
It is recommended to reduce the trace information generated by MCDS to the required minimum.

- To make best use of the available trace memory.

Optimum MCDS Configuration (OS)

Your function time chart **can** include task information if you advise MCDS to export the instruction flow and task switches. For details refer to the chapter **"OS-Aware Tracing - SMP Systems"**, page 178 of this training.

Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0      ; enable TC 1.6.1 CPU0 as
                                           ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1      ; enable TC 1.6.1 CPU1 as
                                           ; trace source

MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON
```

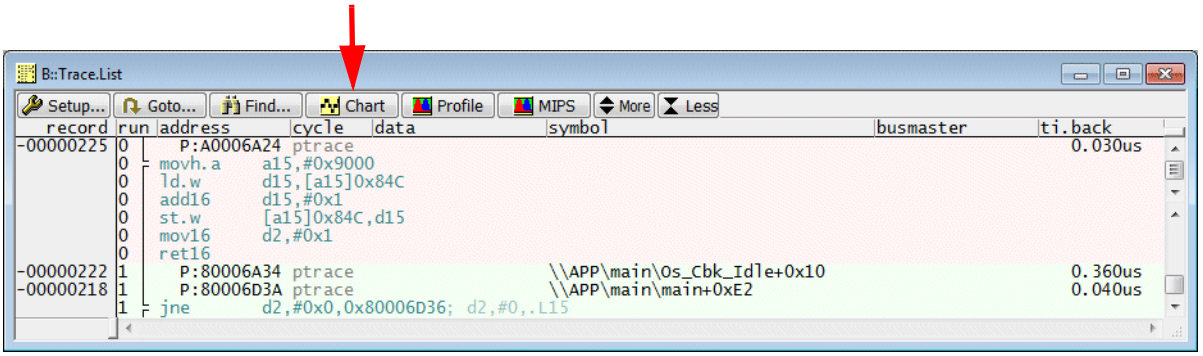
Commands to advise MCDS to export the instruction flow and task switches:

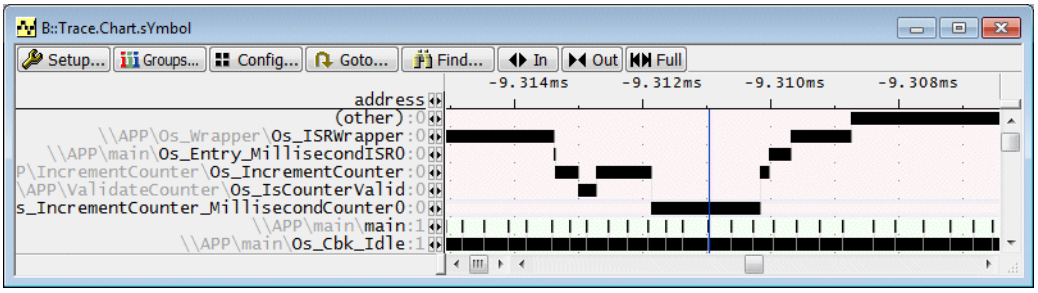
```
Break.Set TASK.CONFIG(magic[0]) /Write /TraceData

Break.Set TASK.CONFIG(magic[1]) /Write /TraceData

; Break.Set TASK.CONFIG(magic[2]) /Write /TraceData
```

Function Timing Diagram (no TASK Information)

TRACE32 PowerView provides a timing diagram which shows when the program counter was in which function/symbol range.

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

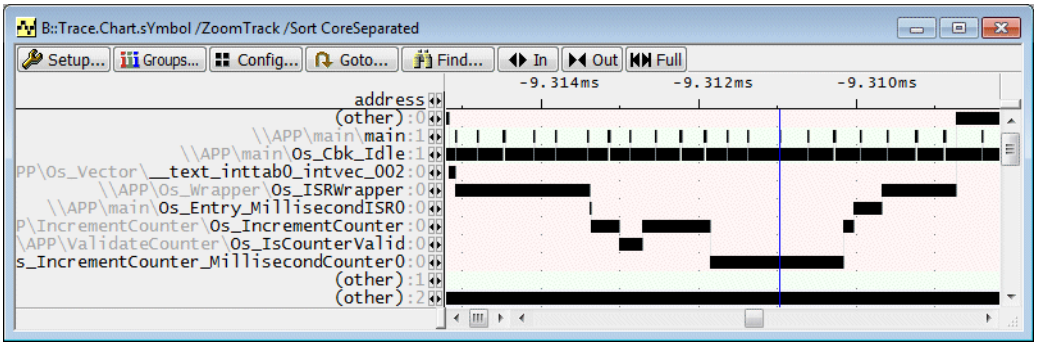




Flat function run-time analysis

- graphical display
- split the result per core
- sort results per core and the per recording order
- no task information

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

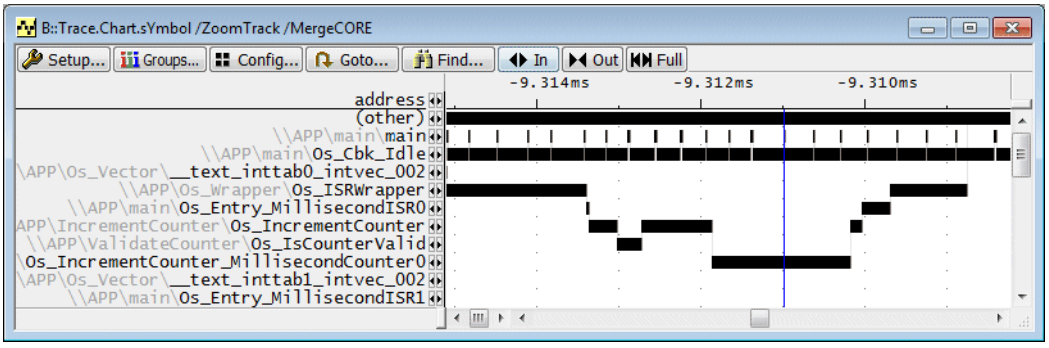


Since no trace information is recorded for TC 1.6.1 CPU2 (other):2 is active for the complete recording time.

Flat function run-time analysis

- graphical display
- split the result per core
- sort results by recording order
- no task information

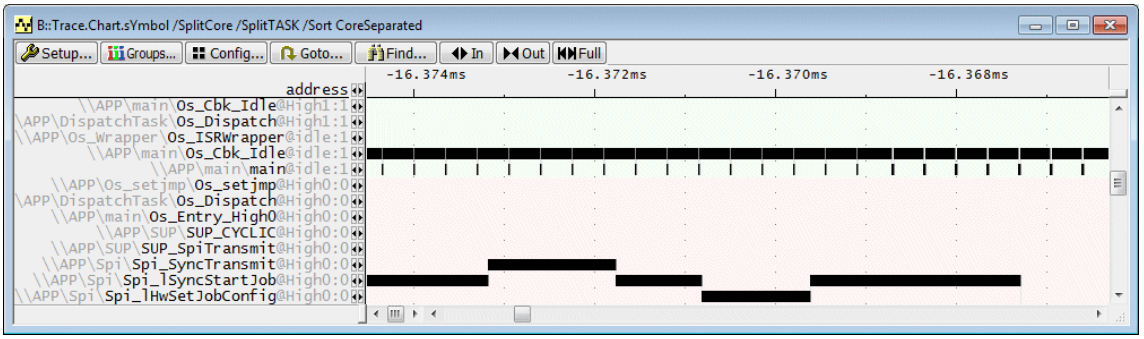
Trace.Chart.sYmbol [**/SplitCore /MergeTASK**] **/Sort CoreSeparated**



Trace.Chart.sYmbol [**/MergeTASK**] **/MergeCore**

Flat function run-time analysis
- graphical display
- merge the results of all cores
- no task information

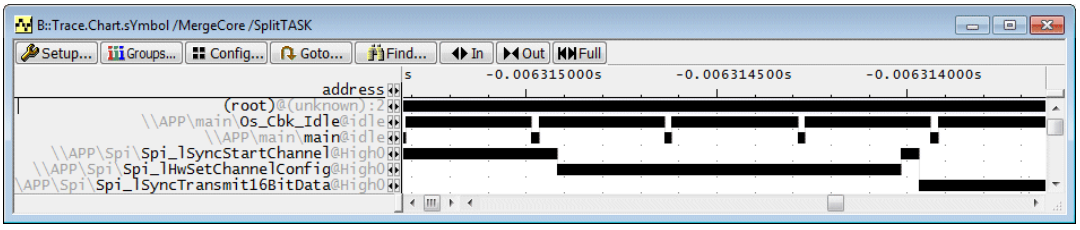
Function Timing Diagram (TASK Information)



Flat function run-time analysis

- graphical display
- split the result per core
- task information
- sort results by recording order

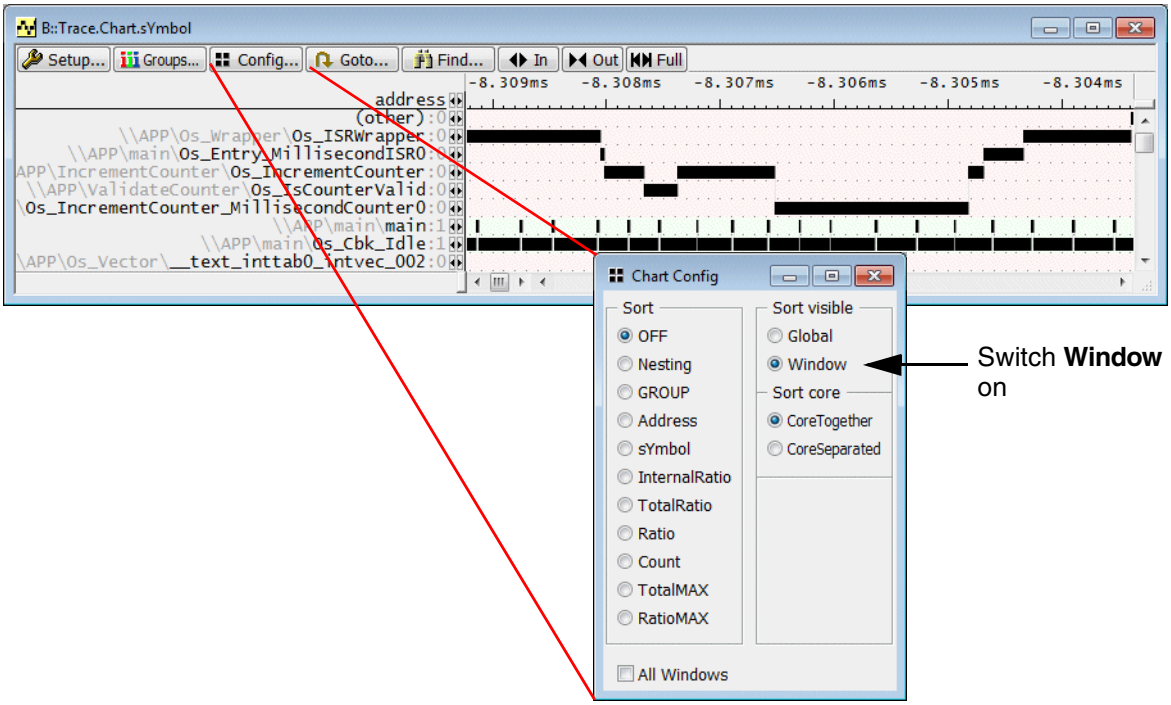
Trace.Chart.sYmbol [/SplitCore] /SplitTASK /Sort CoreSeparated



Flat function run-time analysis

- graphical display
- merge the results of all cores
- task information

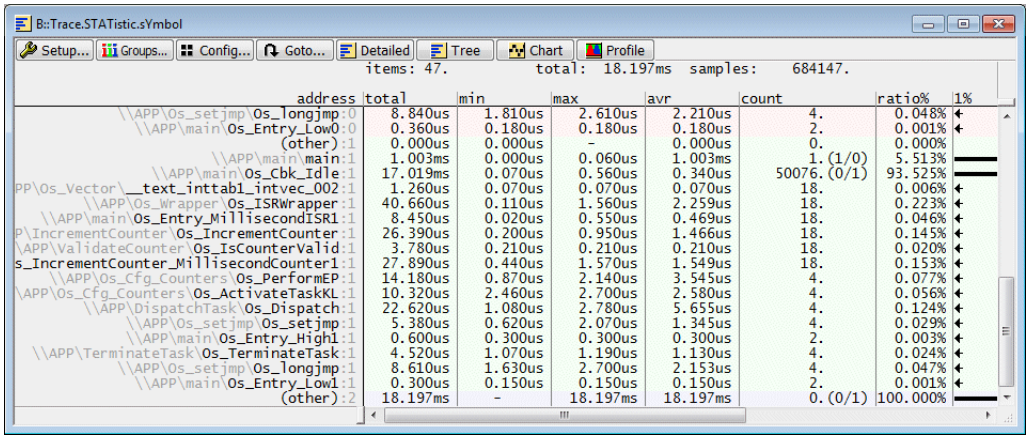
Trace.Chart.sYmbol /MergeCore /SplitTASK



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.

Numeric Analysis

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



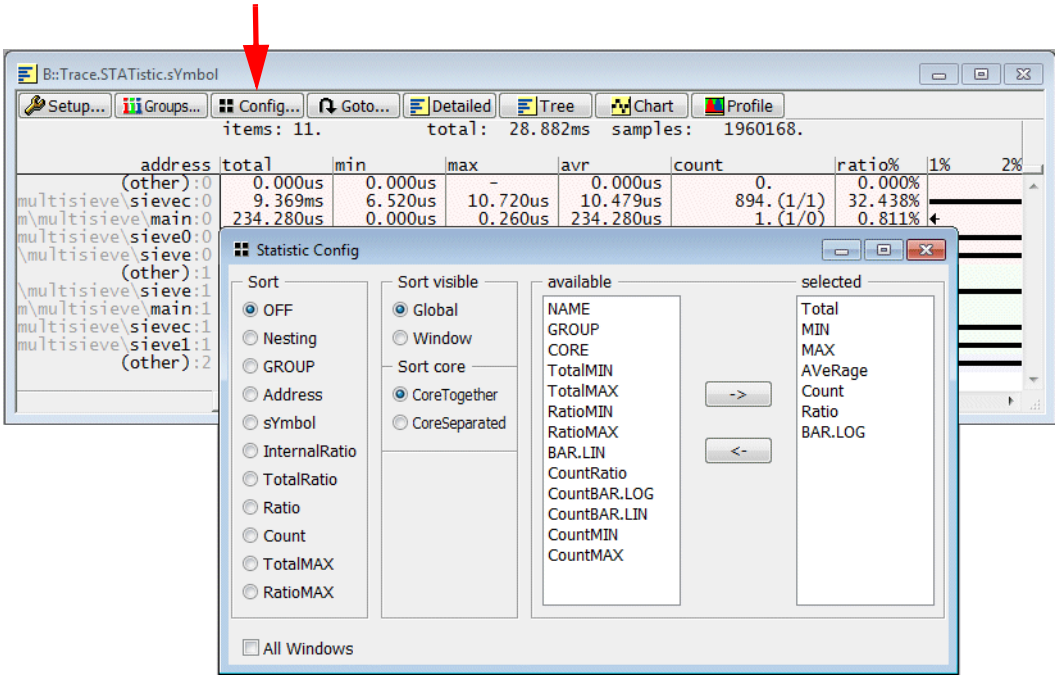
address	total	min	max	avr	count	ratio%
\\APP\Os_setjmp\Os_longjmp:0	8.840us	1.810us	2.610us	2.210us	4.	0.048%
\\APP\main\Os_Entry_Low0:0	0.360us	0.180us	0.180us	0.180us	2.	0.001%
(other):1	0.000us	0.000us	-	0.000us	0.	0.000%
\\APP\main\main:1	1.003ms	0.000us	0.060us	1.003ms	1.(1/0)	5.513%
\\APP\main\Os_Cbk_Idle:1	17.019ms	0.070us	0.560us	0.340us	50076.(0/1)	93.525%
PP\Os_Vector__text_inttab1_intvec_002:1	1.260us	0.070us	0.070us	0.070us	18.	0.006%
\\APP\Os_Wrapper\Os_ISRWrapper:1	40.660us	0.110us	1.560us	2.259us	18.	0.223%
\\APP\main\Os_Entry_MillisecondISR1:1	8.450us	0.020us	0.550us	0.469us	18.	0.046%
P\IncrementCounter\Os_IncrementCounter:1	26.390us	0.200us	0.950us	1.466us	18.	0.145%
\APP\ValidateCounter\Os_IsCounterValid:1	3.780us	0.210us	0.210us	0.210us	18.	0.020%
s_IncrementCounter_MillisecondCounter1:1	27.890us	0.440us	1.570us	1.549us	18.	0.153%
\\APP\Os_Cfg_Counters\Os_PerformEP:1	14.180us	0.870us	2.140us	3.545us	4.	0.077%
\APP\Os_Cfg_Counters\Os_ActivateTaskKL:1	10.320us	2.460us	2.700us	2.580us	4.	0.056%
\\APP\DispatchTask\Os_Dispatch:1	22.620us	1.080us	2.780us	5.655us	4.	0.124%
\\APP\Os_setjmp\Os_setjmp:1	5.380us	0.620us	2.070us	1.345us	4.	0.029%
\\APP\main\Os_Entry_High1:1	0.600us	0.300us	0.300us	0.300us	2.	0.003%
\\APP\TerminateTask\Os_TerminateTask:1	4.520us	1.070us	1.190us	1.130us	4.	0.024%
\\APP\Os_setjmp\Os_longjmp:1	8.610us	1.630us	2.700us	2.153us	4.	0.047%
\\APP\main\Os_Entry_Low1:1	0.300us	0.150us	0.150us	0.150us	2.	0.001%
(other):2	18.197ms	-	18.197ms	18.197ms	0.(0/1)	100.000%

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 (instruction flow continuously in the address range of a function/symbol region)

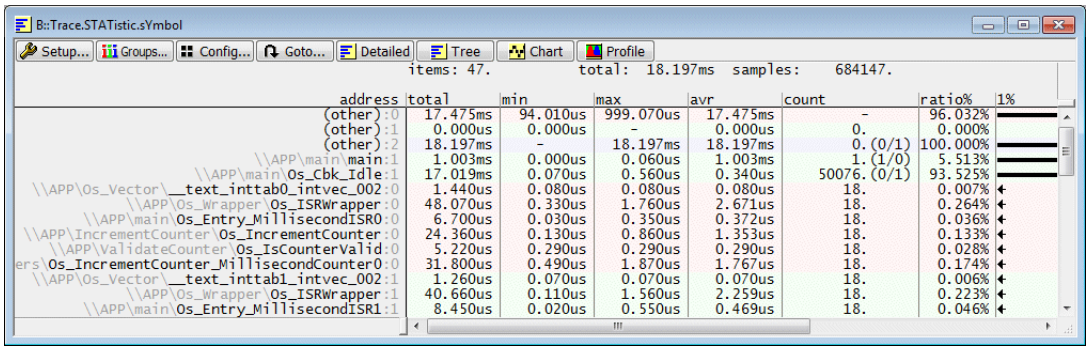
function details	
address	function/symbol region name (here per core) **(other)** program sections that can not be assigned to a function/symbol region
total	time period in the function/symbol region during the recorded time period
min	shortest time continuously in the address range of the function/symbol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

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

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

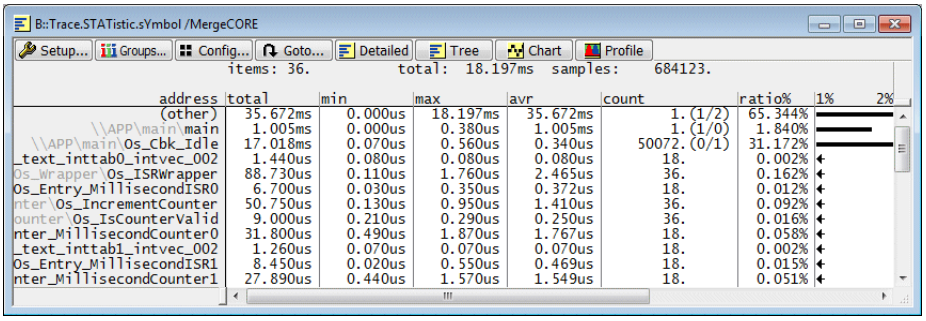


Trace.STATistic.sYmbol [**/SplitCORE /Sort CoreTogether**]	Flat function run-time analysis - numeric display - split the result per core - sort results per core and then per recording order - no task information



Trace.STATistic.sYmbol [**/SplitCORE**] **/Sort CoreSeparated**

Flat function run-time analysis
- numeric display
- split the result per core
- sort results per recording order
- no task information



Trace.STATistic.sYmbol /MergeCORE

Flat function run-time analysis
- numeric display
- merge the results of all cores
- no task information

Nesting Function Run-Time Analysis - Single

The following applies to single core and AMP applications.

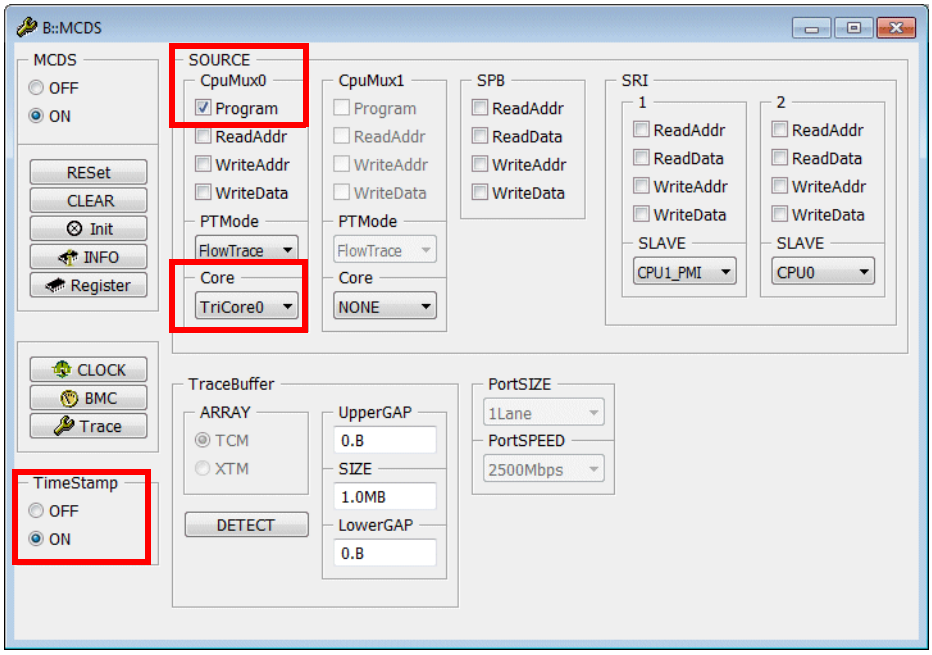
Restrictions

1. **The nesting analysis analyses only high-level language functions.**
2. **The nesting function run-time analysis expects common ways to enter/exit functions.**
3. **The nesting analysis is sensitive with regards to FIFOFULLs.**

Optimum MCDS Configuration (No OS)

Nesting function run-time analysis does **not** require any **data information** if no OS is used. That's why it is recommended to disable the generation of Write Data and Read Data Trace Messages.

Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.

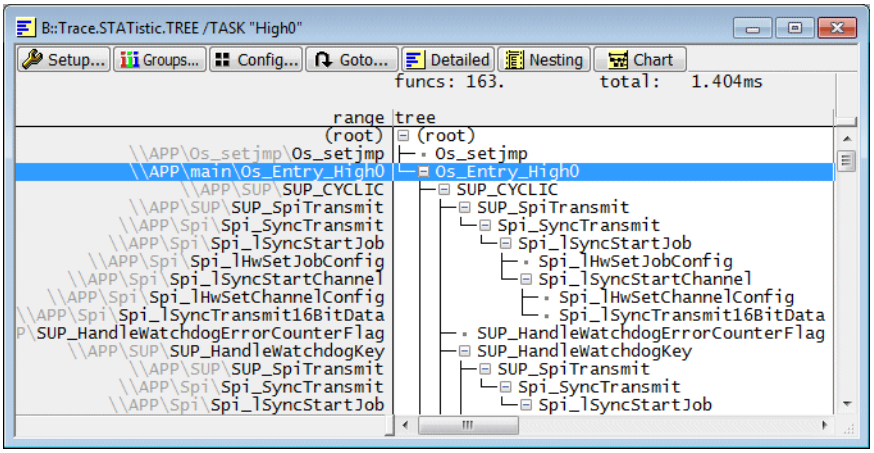



```
MCDS.SOURCE.Set CpuMux0.Core TriCore0      ; enable TC 1.6.1 CPU0 as
                                           ; trace source

MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON

MCDS.SOURCE.Set CpuMux0.Program ON         ; enable Instruction Pointer
                                           ; Call Messages for
                                           ; TC 1.6.1 CPU0
```

```
; default setting since 2015-01
Trace.STATistic.INTERRUPTISFUNCTION
```

TRACE32 PowerView builds up a separate call tree for each task/process.

```
Trace.STATistic.TREE /TASK "High0"
```

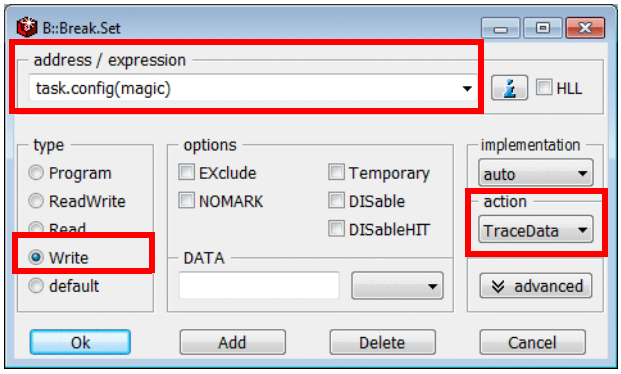


In order to hook a function entry/exit into the correct call tree, TRACE32 PowerView needs to know which task was running when the entry/exit occurred.

The standard way to get information on the current task is to advise the MCDS to generate trace messages for the instruction flow and the task switches. For details refer to the **"OS-Aware Tracing - Single-Core and AMP"** in Training AURIX Tracing, page 157 (training_aurix_trace.pdf).

Advise the Processor Observation Block to generate trace messages for the complete instruction flow and for the task switches.

Filter settings



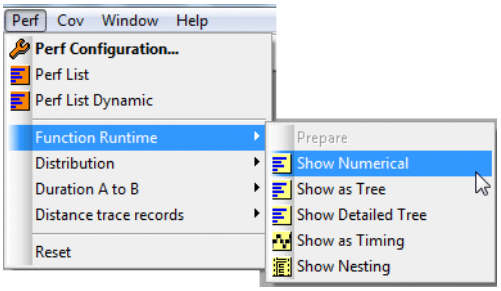
Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.

```
MCDS.SOURCE.Set CpuMux0.Core TriCore0      ; enable TC 1.6.1 CPU0 as
                                           ; trace source

MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON
```

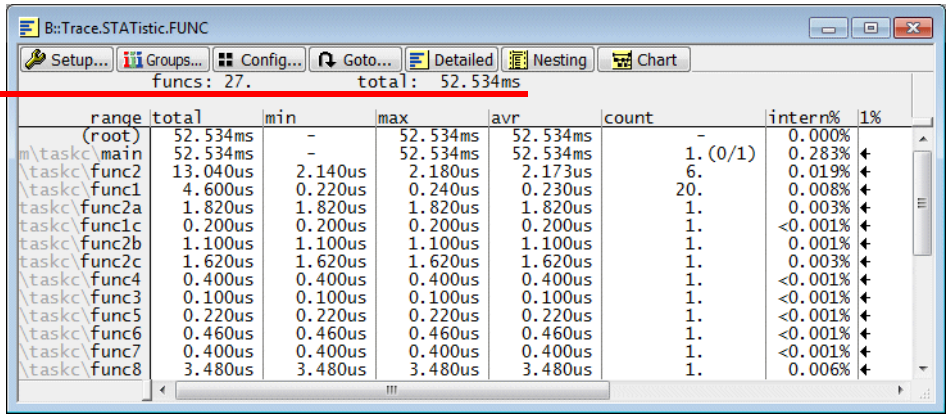
```
; default setting since 2015-01
Trace.STATistic.InterruptIsFunction
```

Statistics Items

Trace.STATistic.Func	Nesting function run-time analysis - numeric display



survey

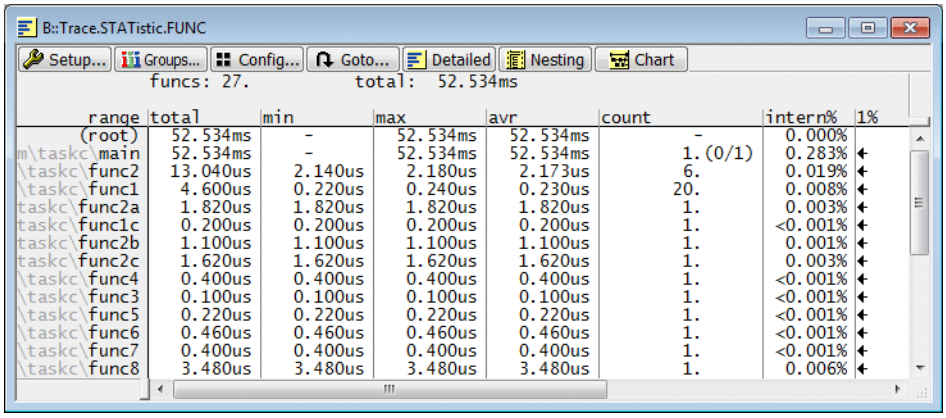



```
funcs: 92.          total:   4.203ms  intr:  20.665ms
```

```
funcs: 45.          total:  44.918ms  intr: 651.830us  31 problems   18 workarounds
```

survey	
funcs: *<number>*	number of functions in the trace
total: *<time>*	total measurement time
intr: *<time>*	total time in interrupt service routines

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



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

- **HLL function**

  \\triboard-tc275_sieve_intmem\taskc\func6

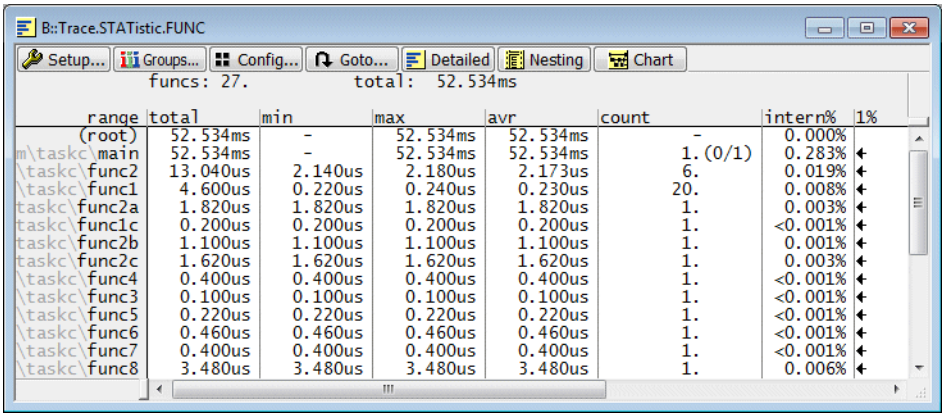
- **(root)**

  (root)

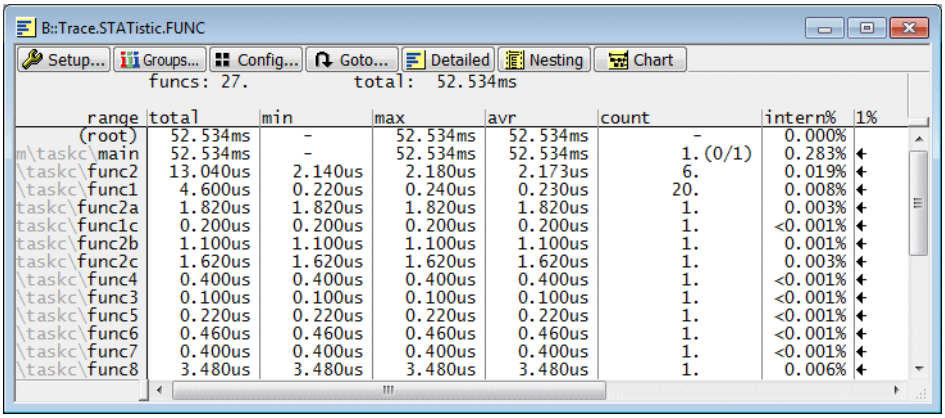
  The function nesting is regarded as tree, root is the root of the function nesting.

- **Interrupt** (branch to an address of an interrupt vector)

  →\\APP__text_inttab0_intvec_002



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



range	total	min	max	avr	count	intern%	1%
(root)	52.534ms	-	52.534ms	52.534ms	-	0.000%	
m\taskc\main	52.534ms	-	52.534ms	52.534ms	1.(0/1)	0.283%	
\taskc\func2	13.040us	2.140us	2.180us	2.173us	6.	0.019%	
\taskc\func1	4.600us	0.220us	0.240us	0.230us	20.	0.008%	
taskc\func2a	1.820us	1.820us	1.820us	1.820us	1.	0.003%	
taskc\func1c	0.200us	0.200us	0.200us	0.200us	1.	<0.001%	
taskc\func2b	1.100us	1.100us	1.100us	1.100us	1.	0.001%	
taskc\func2c	1.620us	1.620us	1.620us	1.620us	1.	0.003%	
\taskc\func4	0.400us	0.400us	0.400us	0.400us	1.	<0.001%	
\taskc\func3	0.100us	0.100us	0.100us	0.100us	1.	<0.001%	
\taskc\func5	0.220us	0.220us	0.220us	0.220us	1.	<0.001%	
\taskc\func6	0.460us	0.460us	0.460us	0.460us	1.	<0.001%	
\taskc\func7	0.400us	0.400us	0.400us	0.400us	1.	<0.001%	
\taskc\func8	3.480us	3.480us	3.480us	3.480us	1.	0.006%	

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

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

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

```
3671.(0/1)
```

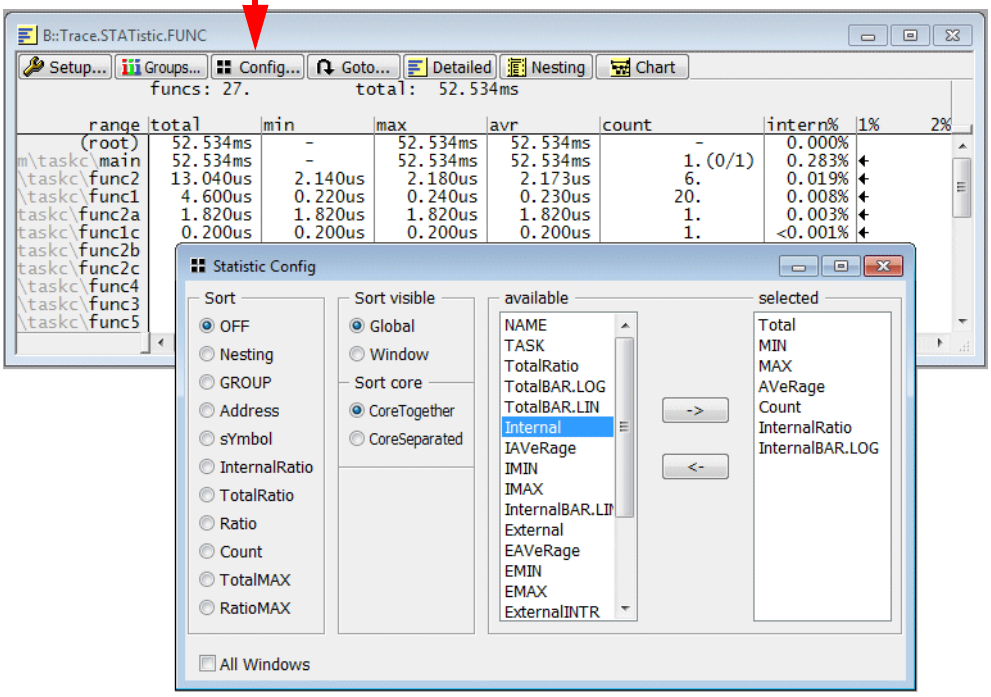
Interpretation examples:

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

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

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

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

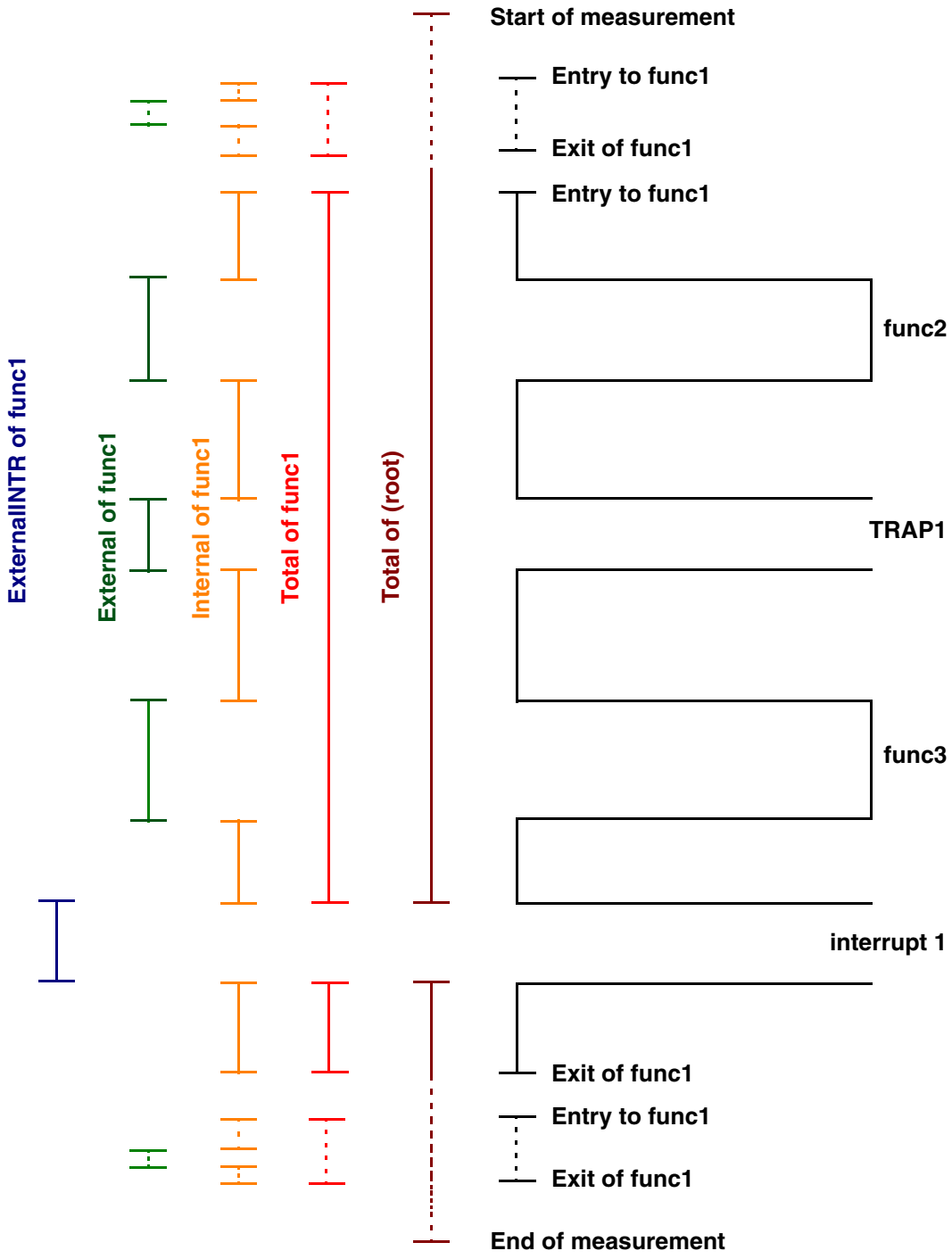


columns (cont.) - times only in function	
Internal	total time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IAVeRage	average time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IMIN	shortest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

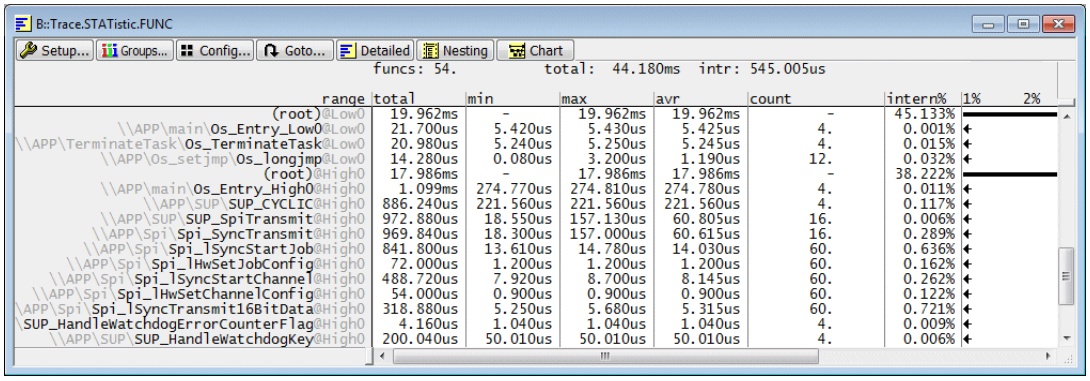
columns (cont.) - times in sub-functions and TRAP handlers	
External	total time spent within called sub-functions/TRAP handlers
EAVeRage	average time spent within called sub-functions/TRAP handlers
EMIN	shortest time spent within called sub-functions/TRAP handlers
EMAX	longest time spent within called sub-functions/TRAP handlers

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

The following graphic give an overview how times are calculated:



Function per Task



range	total	min	max	avr	count	intern%	1% 2%
(root)@Low0	19.962ms	-	19.962ms	19.962ms	-	45.133%	
\\APP\main\Os_Entry_Low0@Low0	21.700us	5.420us	5.430us	5.425us	4.	0.001%	
\\APP\TerminateTask\Os_TerminateTask@Low0	20.980us	5.240us	5.250us	5.245us	4.	0.015%	
\\APP\Os_setjmp\Os_longjmp@Low0	14.280us	0.080us	3.200us	1.190us	12.	0.032%	
(root)@High0	17.986ms	-	17.986ms	17.986ms	-	38.222%	
\\APP\main\Os_Entry_High0@High0	1.099ms	274.770us	274.810us	274.780us	4.	0.011%	
\\APP\SUP\SUP_CYCLIC@High0	886.240us	221.560us	221.560us	221.560us	4.	0.117%	
\\APP\SUP\SUP_SpiTransmit@High0	972.880us	18.550us	157.130us	60.805us	16.	0.006%	
\\APP\Spi\Spi_SyncTransmit@High0	969.840us	18.300us	157.000us	60.615us	16.	0.289%	
\\APP\Spi\Spi_lSyncStartJob@High0	841.800us	13.610us	14.780us	14.030us	60.	0.636%	
\\APP\Spi\Spi_lHwSetJobConfig@High0	72.000us	1.200us	1.200us	1.200us	60.	0.162%	
\\APP\Spi\Spi_lSyncStartChannel@High0	488.720us	7.920us	8.700us	8.145us	60.	0.262%	
\\APP\Spi\Spi_lHwSetChannelConfig@High0	54.000us	0.900us	0.900us	0.900us	60.	0.122%	
\APP\Spi\Spi_lSyncTransmit16BitData@High0	318.880us	5.250us	5.680us	5.315us	60.	0.721%	
\SUP_HandleWatchdogErrorCounterFlag@High0	4.160us	1.040us	1.040us	1.040us	4.	0.009%	
\\APP\SUP\SUP_HandleWatchdogKey@High0	200.040us	50.010us	50.010us	50.010us	4.	0.006%	

- **HLL function**

  \\APP\SUP\SUP_SpiTransmit@High0

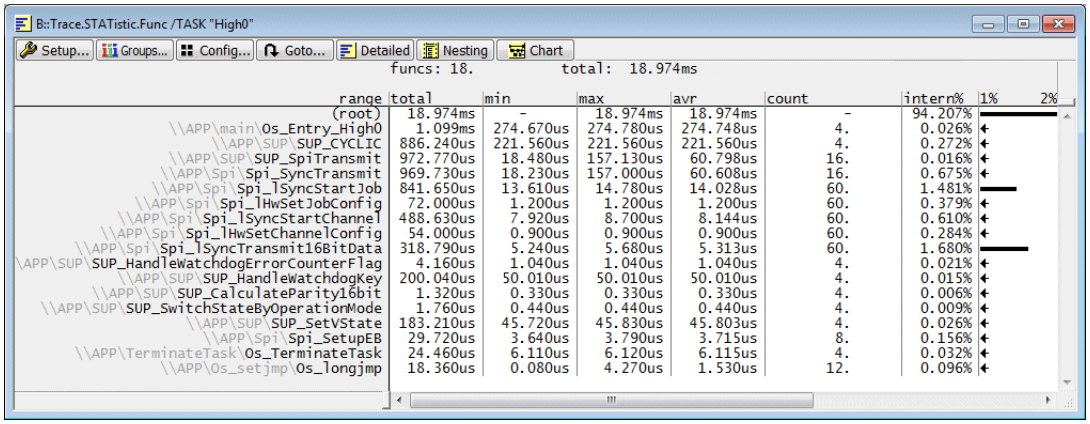
  HLL function “SUP_SpiTransmit” running in task “High0”

- **Root of call tree for task “High0”**

  (root)@High0

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

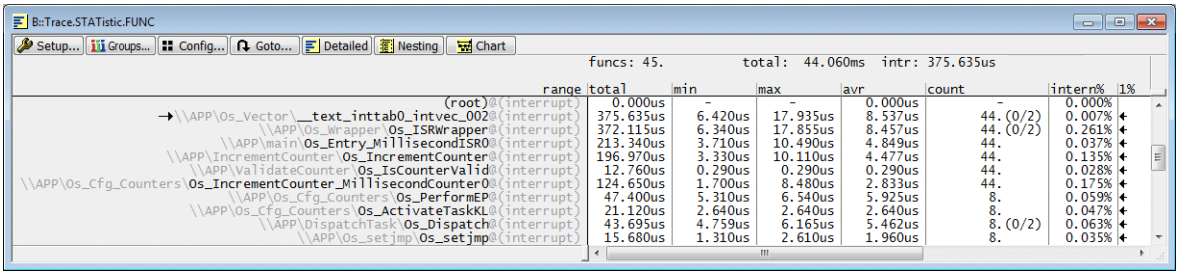
```
Trace.STATistic.Func /TASK "High0"
```



range	total	min	max	avr	count	intern%	1% 2%
(root)	18.974ms	-	18.974ms	18.974ms	-	94.207%	
\\APP\main\Os_Entry_High0	1.099ms	274.670us	274.780us	274.748us	4.	0.026%	
\\APP\SUP\SUP_CYCLIC	886.240us	221.560us	221.560us	221.560us	4.	0.272%	
\\APP\SUP\SUP_SpiTransmit	972.770us	18.480us	157.130us	60.798us	16.	0.016%	
\\APP\Spi\Spi_SyncTransmit	969.730us	18.230us	157.000us	60.608us	16.	0.675%	
\\APP\Spi\Spi_lSyncStartJob	841.650us	13.610us	14.780us	14.028us	60.	1.481%	
\\APP\Spi\Spi_lHwSetJobConfig	72.000us	1.200us	1.200us	1.200us	60.	0.379%	
\\APP\Spi\Spi_lSyncStartChannel	488.630us	7.920us	8.700us	8.144us	60.	0.610%	
\\APP\Spi\Spi_lHwSetChannelConfig	54.000us	0.900us	0.900us	0.900us	60.	0.284%	
\\APP\Spi\Spi_lSyncTransmit16BitData	318.790us	5.240us	5.680us	5.313us	60.	1.680%	
\APP\SUP\SUP_HandleWatchdogErrorCounterFlag	4.160us	1.040us	1.040us	1.040us	4.	0.021%	
\\APP\SUP\SUP_HandleWatchdogKey	200.040us	50.010us	50.010us	50.010us	4.	0.015%	
\\APP\SUP\SUP_CalculateParity16bit	1.320us	0.330us	0.330us	0.330us	4.	0.006%	
\\APP\SUP\SUP_SwitchStateByOperationMode	1.760us	0.440us	0.440us	0.440us	4.	0.009%	
\\APP\SUP\SUP_SetVState	183.210us	45.720us	45.830us	45.803us	4.	0.026%	
\\APP\Spi\Spi_SetupEB	29.720us	3.640us	3.790us	3.715us	8.	0.156%	
\\APP\TerminateTask\Os_TerminateTask	24.460us	6.110us	6.120us	6.115us	4.	0.032%	
\\APP\Os_setjmp\Os_longjmp	18.360us	0.080us	4.270us	1.530us	12.	0.096%	

Interrupt Functions

Interrupt are assigned to the @interrupt task.



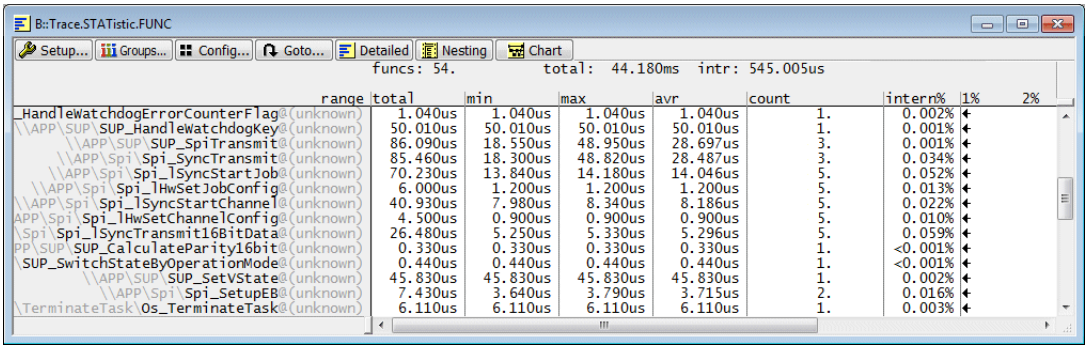
B::Trace.STATistic.FUNC

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

funcs: 45. total: 44.060ms intr: 375.635us

range	total	min	max	avr	count	intern%	1%
(root)@(interrupt)	0.000us	-	-	0.000us	-	0.000%	
→\\APP\Os_Vector__text_inttab0_intvec_002@(interrupt)	375.635us	6.420us	17.935us	8.537us	44.(0/2)	0.007%	
\\APP\Os_Wrapper\Os_ISRWrapper@(interrupt)	372.115us	6.340us	17.855us	8.457us	44.(0/2)	0.261%	
\\APP\main\Os_Entry_MillisecondISR0@(interrupt)	213.340us	3.710us	10.490us	4.849us	44.	0.037%	
\\APP\IncrementCounter\Os_IncrementCounter@(interrupt)	196.970us	3.330us	10.110us	4.477us	44.	0.135%	
\\APP\ValidateCounter\Os_IsCounterValid@(interrupt)	12.760us	0.290us	0.290us	0.290us	44.	0.028%	
\\APP\Os_Cfg_Counters\Os_IncrementCounter_MillisecondCounter0@(interrupt)	124.650us	1.700us	8.480us	2.833us	44.	0.175%	
\\APP\Os_Cfg_Counters\Os_PerformEP@(interrupt)	47.400us	5.310us	6.540us	5.925us	8.	0.059%	
\\APP\Os_Cfg_Counters\Os_ActivateTaskKL@(interrupt)	21.120us	2.640us	2.640us	2.640us	8.	0.047%	
\\APP\DispatchTask\Os_Dispatch@(interrupt)	43.695us	4.759us	6.165us	5.462us	8.(0/2)	0.063%	
\\APP\Os_setjmp\Os_setjmp@(interrupt)	15.680us	1.310us	2.610us	1.960us	8.	0.035%	

The unknown Task



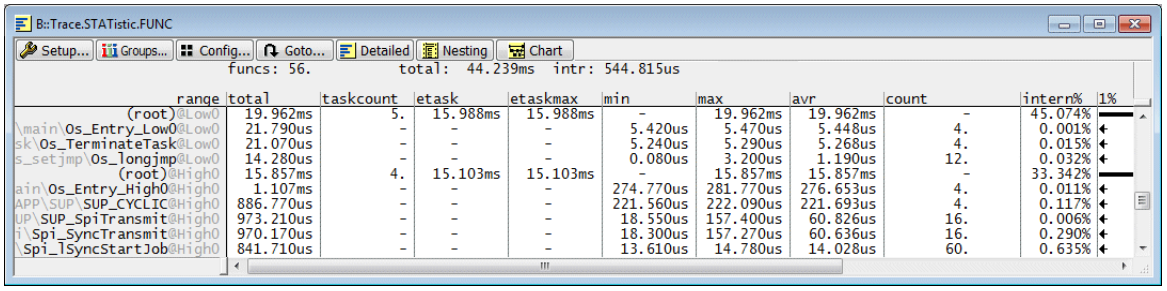
B::Trace.STATistic.FUNC

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

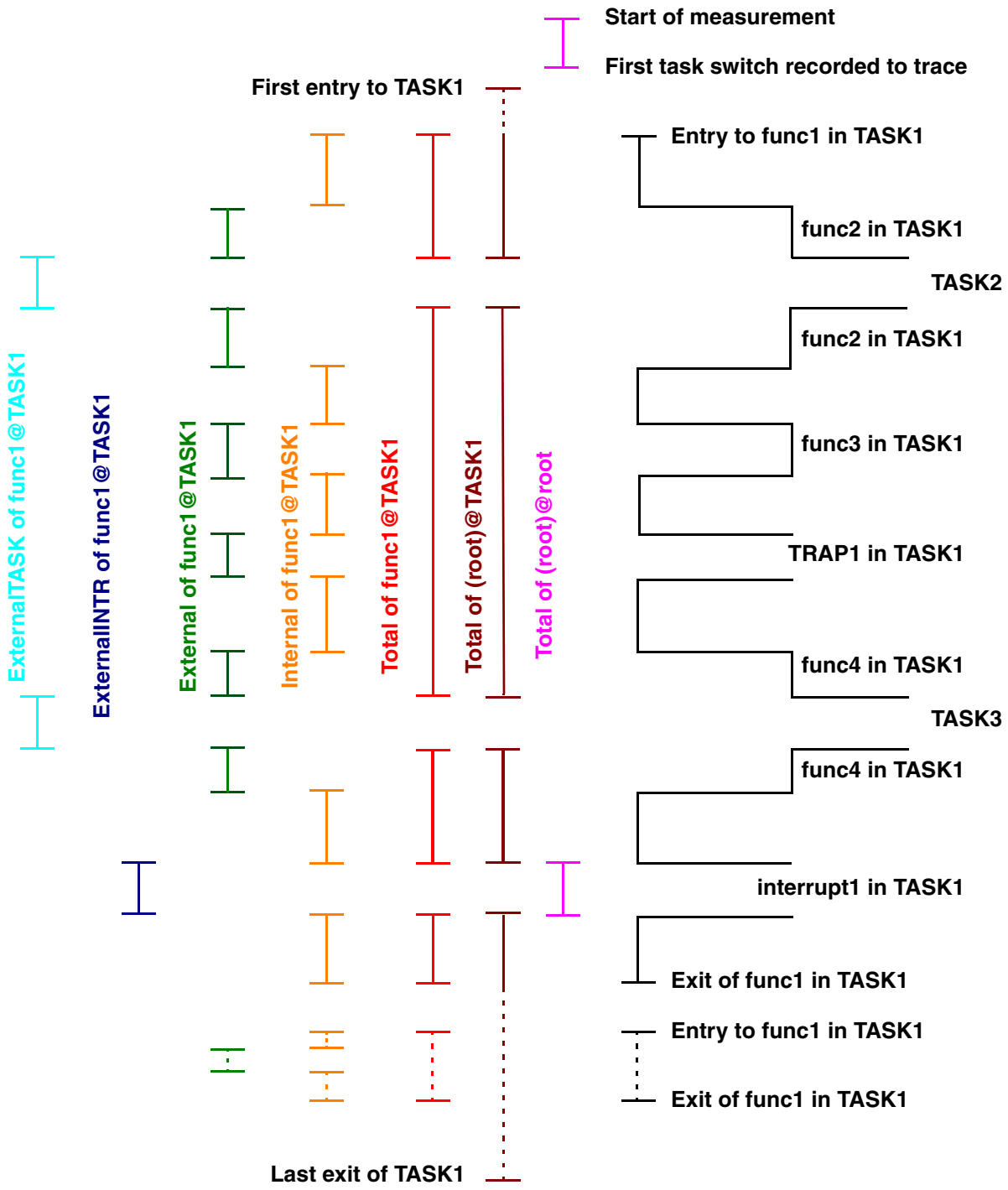
funcs: 54. total: 44.180ms intr: 545.005us

range	total	min	max	avr	count	intern%	1%	2%
_HandleWatchdogErrorCounterFlag@(unknown)	1.040us	1.040us	1.040us	1.040us	1.	0.002%		
\\APP\SUP\SUP_HandleWatchdogKey@(unknown)	50.010us	50.010us	50.010us	50.010us	1.	0.001%		
\\APP\SUP\SUP_SpiTransmit@(unknown)	86.090us	18.550us	48.950us	28.697us	3.	0.001%		
\\APP\Spi\Spi_SyncTransmit@(unknown)	85.460us	18.300us	48.820us	28.487us	3.	0.034%		
\\APP\Spi\Spi_lSyncStartJob@(unknown)	70.230us	13.840us	14.180us	14.046us	5.	0.052%		
\\APP\Spi\Spi_lHwSetJobConfig@(unknown)	6.000us	1.200us	1.200us	1.200us	5.	0.013%		
\\APP\Spi\Spi_lSyncStartChannel@(unknown)	40.930us	7.980us	8.340us	8.186us	5.	0.022%		
APP\Spi\Spi_lHwSetChannelConfig@(unknown)	4.500us	0.900us	0.900us	0.900us	5.	0.010%		
\Spi\Spi_lSyncTransmit16BitData@(unknown)	26.480us	5.250us	5.330us	5.296us	5.	0.059%		
PP\SUP\SUP_CalculateParity16bit@(unknown)	0.330us	0.330us	0.330us	0.330us	1.	<0.001%		
\SUP_SwitchStateByOperationMode@(unknown)	0.440us	0.440us	0.440us	0.440us	1.	<0.001%		
\\APP\SUP\SUP_SetVState@(unknown)	45.830us	45.830us	45.830us	45.830us	1.	0.002%		
\\APP\Spi\Spi_SetupEB@(unknown)	7.430us	3.640us	3.790us	3.715us	2.	0.016%		
\TerminateTask\Os_TerminateTask@(unknown)	6.110us	6.110us	6.110us	6.110us	1.	0.003%		

All function recorded before the first task switch record are assigned to the unknown task.

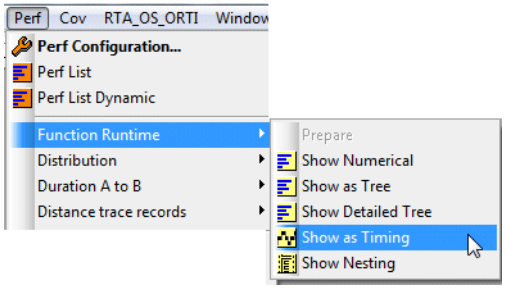


columns - task/thread related information	
TASKCount	number of tasks that interrupt the function
ExternalTASK	total time in other tasks
ExternalTASKMAX	max. time 1 function pass was interrupted by a task

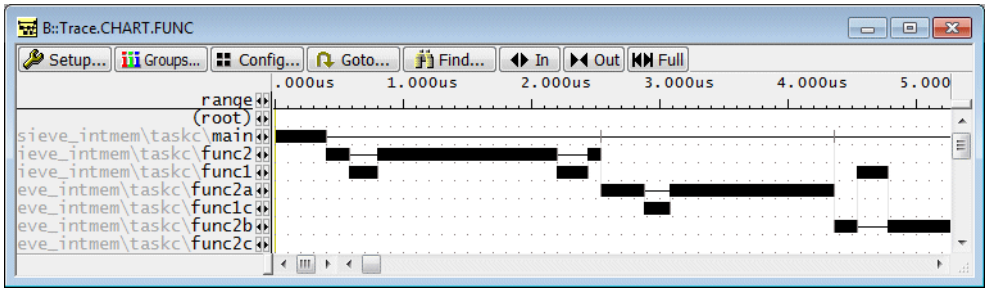

Start of measurement
First task switch recorded to trace
First entry to TASK1
Entry to func1 in TASK1
func2 in TASK1
TASK2
func2 in TASK1
func3 in TASK1
TRAP1 in TASK1
func4 in TASK1
TASK3
func4 in TASK1
interrupt1 in TASK1
Exit of func1 in TASK1
Entry to func1 in TASK1
Exit of func1 in TASK1
Last exit of TASK1
ExternalTASK of func1@TASK1
ExternalINTR of func1@TASK1
External of func1@TASK1
Internal of func1@TASK1
Total of func1@TASK1
Total of (root)@TASK1
Total of (root)@root

More Nesting Analysis Commands

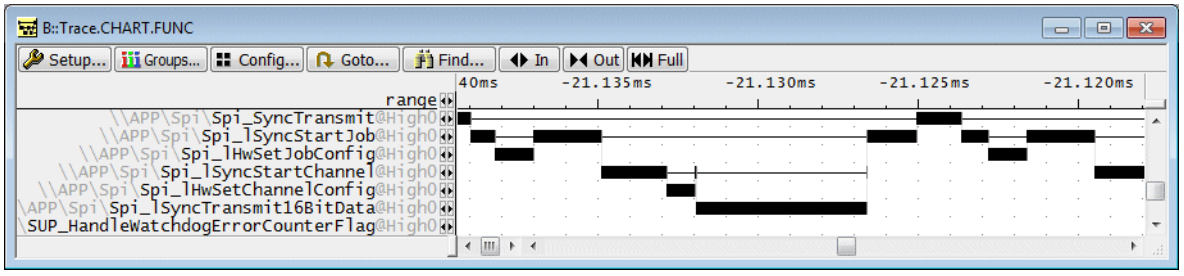
Trace.Chart.Func	Nesting function run-time analysis - graphical display



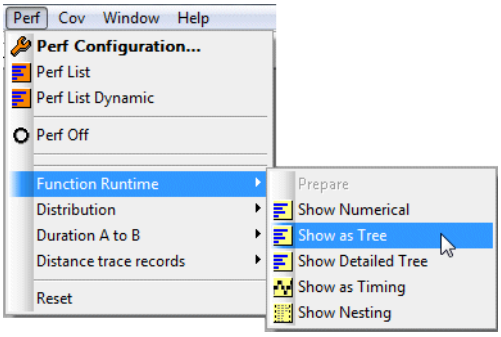
Look and Feel (No OS)



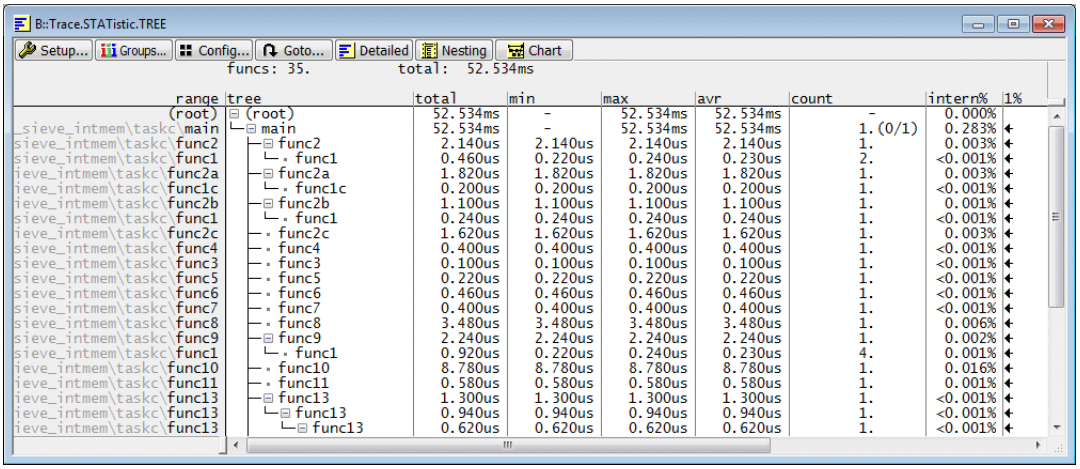
Look and Feel (OS)



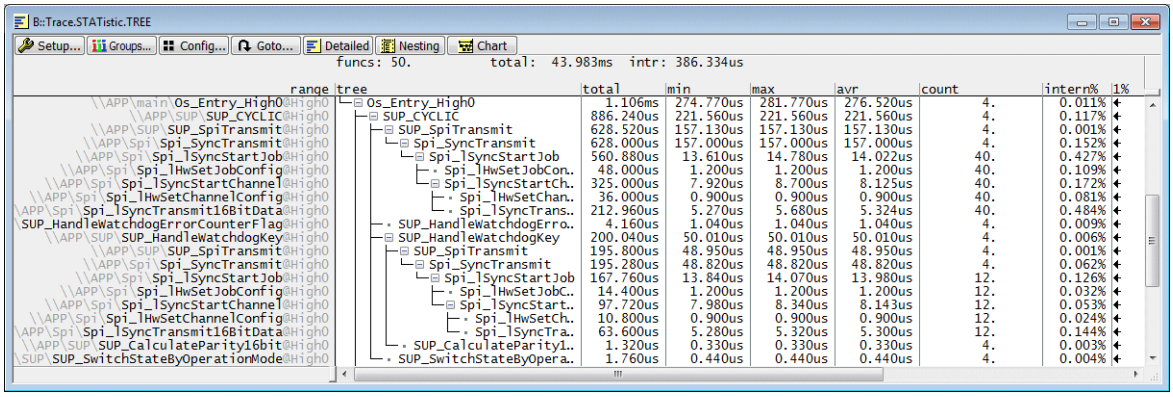
Trace.STATistic.TREE Nesting function run-time analysis
- tree display



Look and Feel (No OS)



range	tree	total	min	max	avr	count	intern%	1%
(root)	(root)	52.534ms	-	52.534ms	52.534ms	-	0.000%	
_sieve_intmem\taskc\main	main	52.534ms	-	52.534ms	52.534ms	1.(0/1)	0.283%	
sieve_intmem\taskc\func2	func2	2.140us	2.140us	2.140us	2.140us	1.	0.003%	
sieve_intmem\taskc\func1	func1	0.460us	0.220us	0.240us	0.230us	2.	<0.001%	
ieve_intmem\taskc\func2a	func2a	1.820us	1.820us	1.820us	1.820us	1.	0.003%	
ieve_intmem\taskc\func1c	func1c	0.200us	0.200us	0.200us	0.200us	1.	<0.001%	
ieve_intmem\taskc\func2b	func2b	1.100us	1.100us	1.100us	1.100us	1.	0.001%	
sieve_intmem\taskc\func1	func1	0.240us	0.240us	0.240us	0.240us	1.	<0.001%	
ieve_intmem\taskc\func2c	func2c	1.620us	1.620us	1.620us	1.620us	1.	0.003%	
sieve_intmem\taskc\func4	func4	0.400us	0.400us	0.400us	0.400us	1.	<0.001%	
sieve_intmem\taskc\func3	func3	0.100us	0.100us	0.100us	0.100us	1.	<0.001%	
sieve_intmem\taskc\func5	func5	0.220us	0.220us	0.220us	0.220us	1.	<0.001%	
sieve_intmem\taskc\func6	func6	0.460us	0.460us	0.460us	0.460us	1.	<0.001%	
sieve_intmem\taskc\func7	func7	0.400us	0.400us	0.400us	0.400us	1.	<0.001%	
sieve_intmem\taskc\func8	func8	3.480us	3.480us	3.480us	3.480us	1.	0.006%	
sieve_intmem\taskc\func9	func9	2.240us	2.240us	2.240us	2.240us	1.	0.002%	
sieve_intmem\taskc\func1	func1	0.920us	0.220us	0.240us	0.230us	4.	0.001%	
ieve_intmem\taskc\func10	func10	8.780us	8.780us	8.780us	8.780us	1.	0.016%	
ieve_intmem\taskc\func11	func11	0.580us	0.580us	0.580us	0.580us	1.	0.001%	
ieve_intmem\taskc\func13	func13	1.300us	1.300us	1.300us	1.300us	1.	<0.001%	
ieve_intmem\taskc\func13	func13	0.940us	0.940us	0.940us	0.940us	1.	<0.001%	
ieve_intmem\taskc\func13	func13	0.620us	0.620us	0.620us	0.620us	1.	<0.001%	



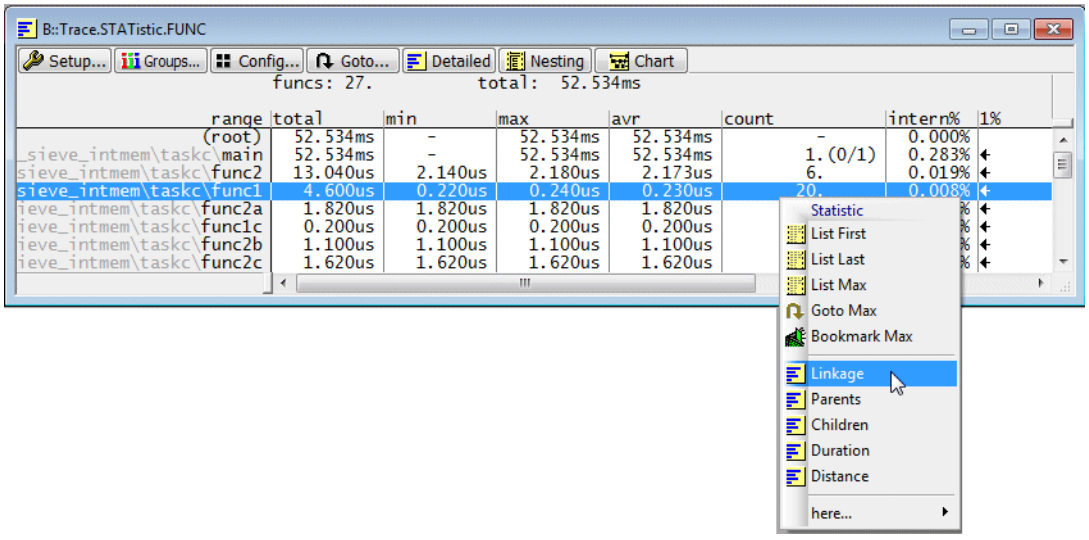
range	tree	total	min	max	avr	count	intern%	1%
\\APP\main\Os_Entry_High0@High0	Os_Entry_High0	1.106ms	274.770us	281.770us	276.520us	4.	0.011%	
\\APP\SUP\SUP_CYCLIC@High0	SUP_CYCLIC	886.240us	221.560us	221.560us	221.560us	4.	0.117%	
\\APP\SUP\SUP_SpiTransmit@High0	SUP_SpiTransmit	628.520us	157.130us	157.130us	157.130us	4.	0.001%	
\\APP\Spi\Spi_SyncTransmit@High0	Spi_SyncTransmit	628.000us	157.000us	157.000us	157.000us	4.	0.152%	
\\APP\Spi\Spi_lSyncStartJob@High0	Spi_lSyncStartJob	560.880us	13.610us	14.780us	14.022us	40.	0.427%	
\\APP\Spi\Spi_lHwSetJobConfig@High0	Spi_lHwSetJobCon..	48.000us	1.200us	1.200us	1.200us	40.	0.109%	
\\APP\Spi\Spi_lSyncStartChannel@High0	Spi_lSyncStartCh..	325.000us	7.920us	8.700us	8.125us	40.	0.172%	
\\APP\Spi\Spi_lHwSetChannelConfig@High0	Spi_lHwSetChan..	36.000us	0.900us	0.900us	0.900us	40.	0.081%	
\APP\Spi\Spi_lSyncTransmit16BitData@High0	Spi_lSyncTrans..	212.960us	5.270us	5.680us	5.324us	40.	0.484%	
\SUP_HandleWatchdogErrorCounterFlag@High0	SUP_HandleWatchdogErro..	4.160us	1.040us	1.040us	1.040us	4.	0.009%	
\\APP\SUP\SUP_HandleWatchdogKey@High0	SUP_HandleWatchdogKey	200.040us	50.010us	50.010us	50.010us	4.	0.006%	
\\APP\SUP\SUP_SpiTransmit@High0	SUP_SpiTransmit	195.800us	48.950us	48.950us	48.950us	4.	0.001%	
\\APP\Spi\Spi_SyncTransmit@High0	Spi_SyncTransmit	195.280us	48.820us	48.820us	48.820us	4.	0.062%	
\\APP\Spi\Spi_lSyncStartJob@High0	Spi_lSyncStartJob	167.760us	13.840us	14.070us	13.980us	12.	0.126%	
\\APP\Spi\Spi_lHwSetJobConfig@High0	Spi_lHwSetJobC..	14.400us	1.200us	1.200us	1.200us	12.	0.032%	
\\APP\Spi\Spi_lSyncStartChannel@High0	Spi_lSyncStart..	97.720us	7.980us	8.340us	8.143us	12.	0.053%	
\\APP\Spi\Spi_lHwSetChannelConfig@High0	Spi_lHwSetCh..	10.800us	0.900us	0.900us	0.900us	12.	0.024%	
\APP\Spi\Spi_lSyncTransmit16BitData@High0	Spi_lSyncTra..	63.600us	5.280us	5.320us	5.300us	12.	0.144%	
\\APP\SUP\SUP_CalculateParity16bit@High0	SUP_CalculateParity1..	1.320us	0.330us	0.330us	0.330us	4.	0.003%	
\SUP\SUP_SwitchStateByOperationMode@High0	SUP_SwitchStateByOpera..	1.760us	0.440us	0.440us	0.440us	4.	0.004%	

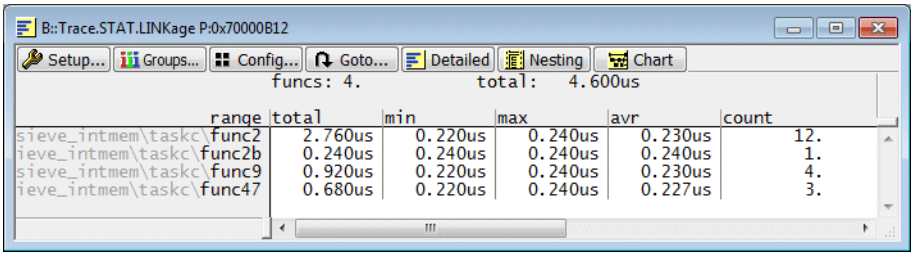
It is also possible to get a task/process-specific tree.

```
Trace.STATistic.TREE /TASK "High0"
```

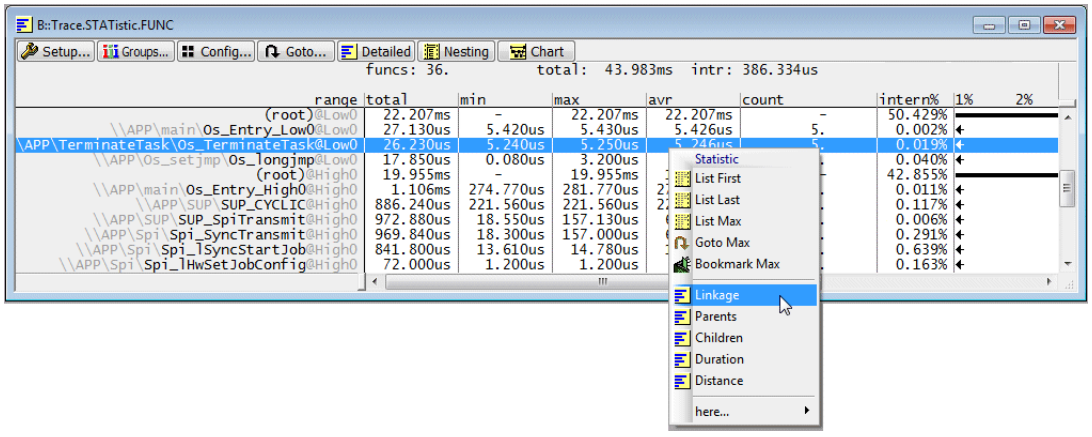
Trace.STATistic.LINKage *<address>* Nesting function run-time analysis
- linkage analysis

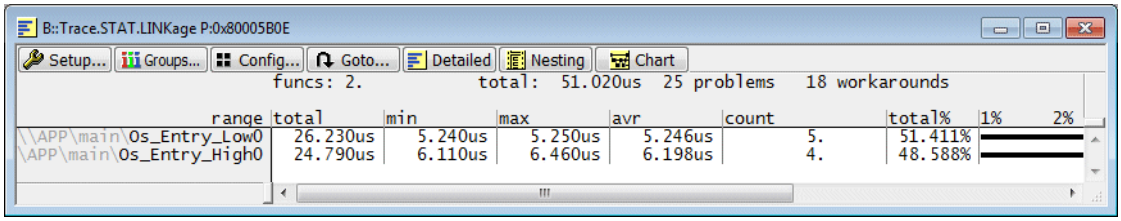
Look and Feel (No OS)





Look and Feel (OS)





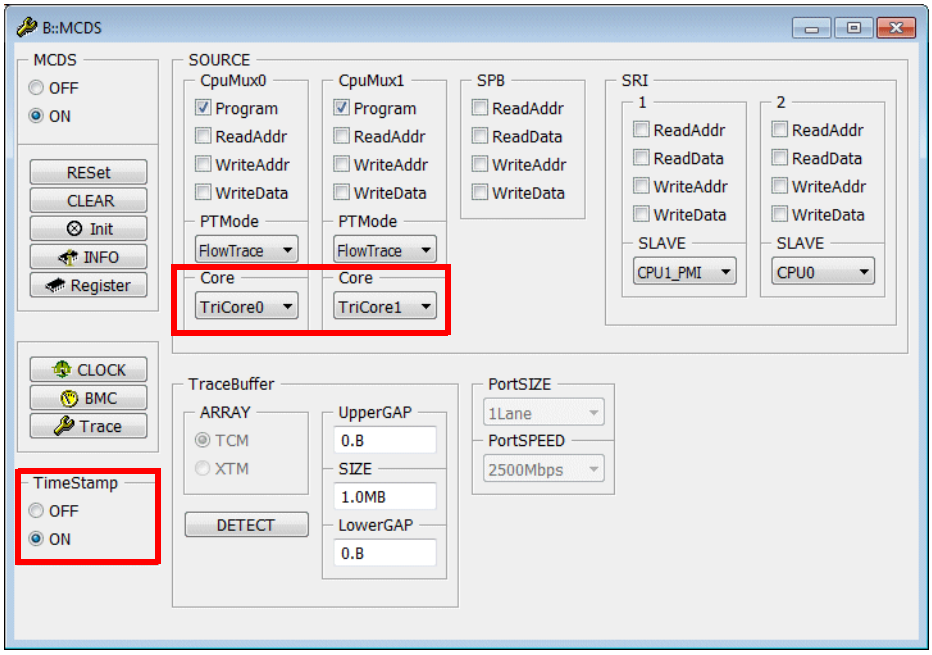
It is recommended to reduce the trace information generated by MCDS to the required minimum.

- To make best use of the available trace memory.

Optimum MCDS Configuration (OS)

Connect the cores of interest to the trace multiplexer

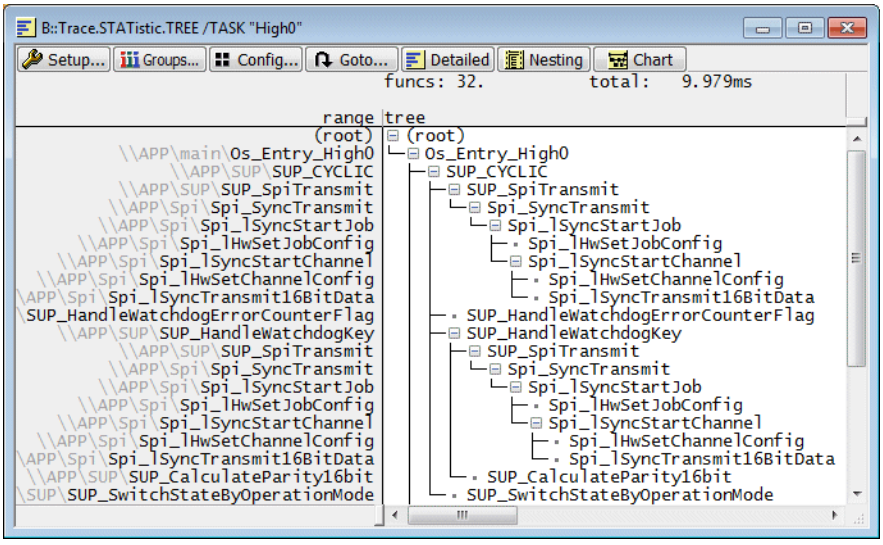
Since the serial off-chip trace provides imprecise timestamps Timestamp Messages have to be enabled for any run-time measurement.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0      ; enable TC 1.6.1 CPU0 as
                                           ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1      ; enable TC 1.6.1 CPU1 as
                                           ; trace source

MCDS.TimeStamp ON                          ; enable Timestamp Messages

CLOCK.ON
```

TRACE32 PowerView builds up a separate call tree for each task/process.

```
Trace.STATistic.TREE /TASK "High0"
```



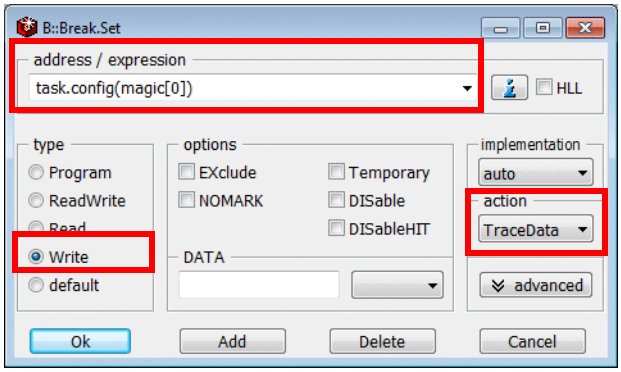
In order to hook a function entry/exit into the correct call tree, TRACE32 PowerView needs to know which task was running when the entry/exit occurred.

The standard way to get information on the current task is to advise the MCDS to generate trace messages for the instruction flow and the task switches. For details refer to the **“OS-Aware Tracing - SMP Systems”** in Training AURIX Tracing, page 178 (training_aurix_trace.pdf).

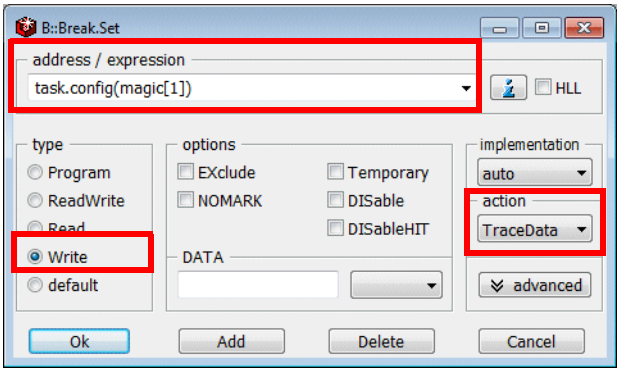
In order to do the optimum setting for the nesting analysis advise the Processor Observation Blocks to generate trace messages for the complete instruction flow and for the task switches.

Filter settings

Set filter for TC1.6.1 CPU0:



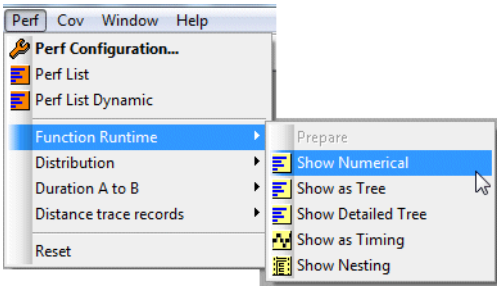
Set filter for TC1.6.1 CPU1:



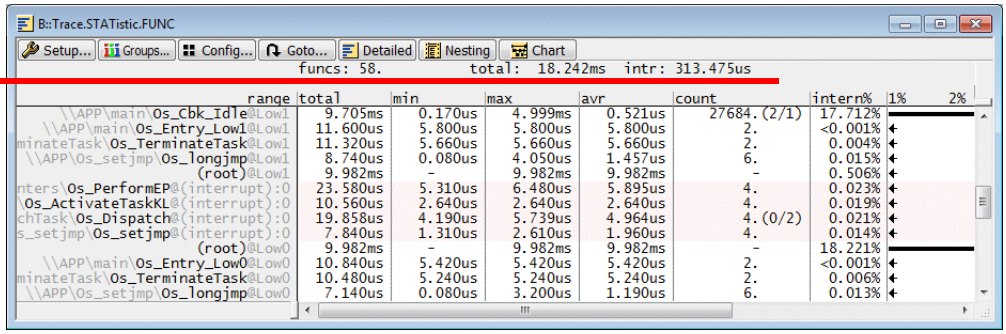

```
; default setting since 2015-01
Trace.STATistic.InterruptIsFunction
```

Statistics Items

Trace.STATistic.Func	Nesting function run-time analysis - numeric display - core information is discarded except for @(unknown) and @(interrupt)



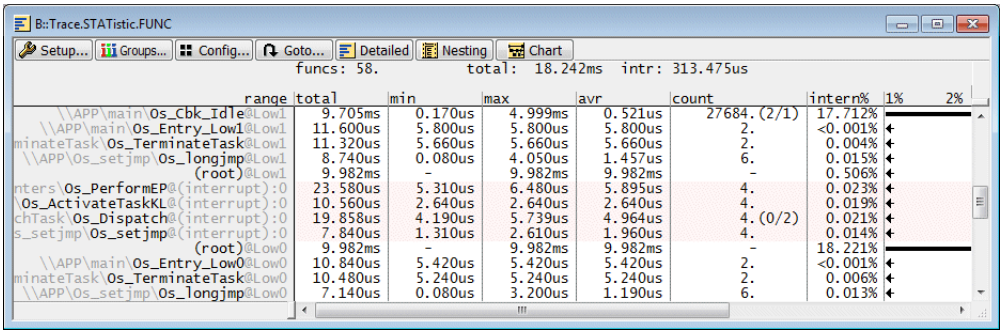
survey




```
funcs: 90.          total:  18.706ms  intr:  11.929ms  21 problems   131 workarounds
```

survey	
funcs: *<number>*	number of functions in the trace
total: *<time>*	total measurement time
intr: *<time>*	total time in interrupt service routines

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



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

- **HLL function**

  \\APP\Os_setjmp\Os_longjmp@Low1

  HLL function “Os_longjmp” running in task “Low1”

- **Root of call tree for task “High0”**

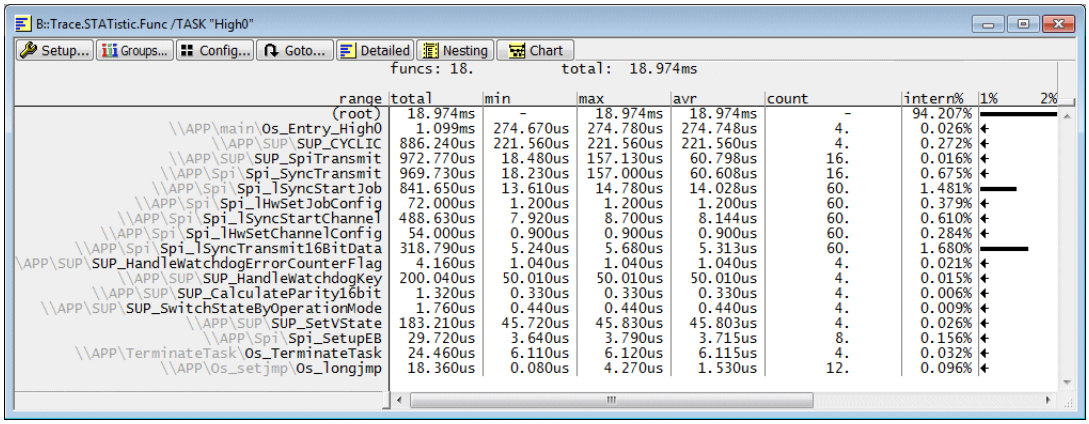
  (root)@High0

  The function nesting is regarded as tree, “root@High0” is the root of the call tree for the task “High0”.

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

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

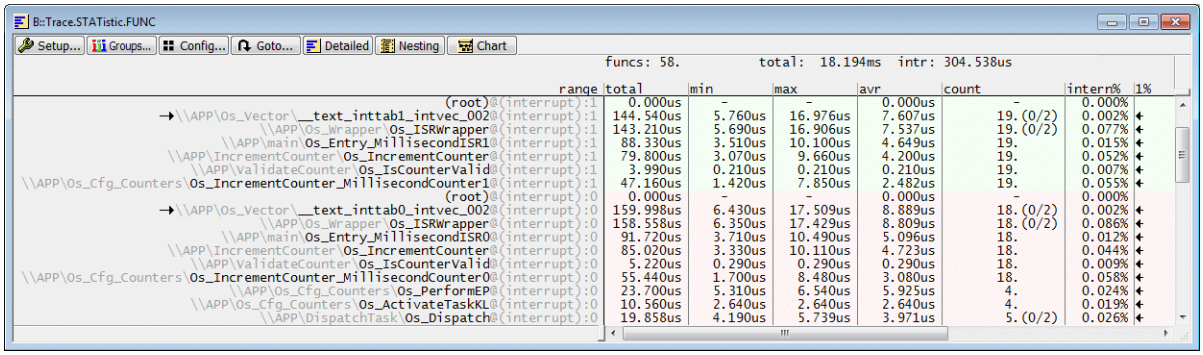
```
Trace.STATistic.Func /TASK "High0"
```



range	total	min	max	avr	count	intern%	1% 2%
(root)	18.974ms	-	18.974ms	18.974ms	-	94.207%	
\\APP\main\Os_Entry_High0	1.099ms	274.670us	274.780us	274.748us	4.	0.026%	
\\APP\SUP\SUP_CYCLIC	886.240us	221.560us	221.560us	221.560us	4.	0.272%	
\\APP\SUP\SUP_SpiTransmit	972.770us	18.480us	157.130us	60.798us	16.	0.016%	
\\APP\Spi\Spi_SyncTransmit	969.730us	18.230us	157.000us	60.608us	16.	0.675%	
\\APP\Spi\Spi_lSyncStartJob	841.650us	13.610us	14.780us	14.028us	60.	1.481%	
\\APP\Spi\Spi_lHwSetJobConfig	72.000us	1.200us	1.200us	1.200us	60.	0.379%	
\\APP\Spi\Spi_lSyncStartChannel	488.630us	7.920us	8.700us	8.144us	60.	0.610%	
\\APP\Spi\Spi_lHwSetChannelConfig	54.000us	0.900us	0.900us	0.900us	60.	0.284%	
\\APP\Spi\Spi_lSyncTransmit16BitData	318.790us	5.240us	5.680us	5.313us	60.	1.680%	
\\APP\SUP\SUP_HandleWatchdogErrorCounterFlag	4.160us	1.040us	1.040us	1.040us	4.	0.021%	
\\APP\SUP\SUP_HandleWatchdogKey	200.040us	50.010us	50.010us	50.010us	4.	0.015%	
\\APP\SUP\SUP_CalculateParity16bit	1.320us	0.330us	0.330us	0.330us	4.	0.006%	
\\APP\SUP\SUP_SwitchStateByOperationMode	1.760us	0.440us	0.440us	0.440us	4.	0.009%	
\\APP\SUP\SUP_SetVState	183.210us	45.720us	45.830us	45.803us	4.	0.026%	
\\APP\Spi\Spi_SetupEB	29.720us	3.640us	3.790us	3.715us	8.	0.156%	
\\APP\TerminateTask\Os_TerminateTask	24.460us	6.110us	6.120us	6.115us	4.	0.032%	
\\APP\Os_setjmp\Os_longjmp	18.360us	0.080us	4.270us	1.530us	12.	0.096%	

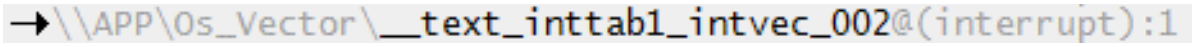
Interrupt Functions

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



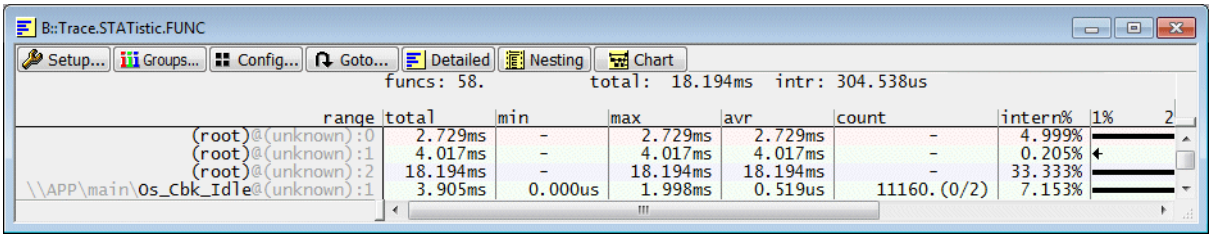
range	total	min	max	avr	count	intern%	1%
(root)@(interrupt):1	0.000us	-	-	0.000us	-	0.000%	
→\\APP\Os_Vector__text_inttab1_intvec_002@(interrupt):1	144.540us	5.760us	16.976us	7.607us	19.(0/2)	0.002%	
\\APP\Os_Wrapper\Os_ISRWrapper@(interrupt):1	143.210us	5.690us	16.906us	7.537us	19.(0/2)	0.077%	
\\APP\main\Os_Entry_MillisecondISR1@(interrupt):1	88.330us	3.510us	10.100us	4.649us	19.	0.015%	
\\APP\IncrementCounter\Os_IncrementCounter@(interrupt):1	79.800us	3.070us	9.660us	4.200us	19.	0.052%	
\\APP\ValidateCounter\Os_IsCounterValid@(interrupt):1	3.990us	0.210us	0.210us	0.210us	19.	0.007%	
\\APP\Os_Cfg_Counters\Os_IncrementCounter_MillisecondCounter1@(interrupt):1	47.160us	1.420us	7.850us	2.482us	19.	0.055%	
(root)@(interrupt):0	0.000us	-	-	0.000us	-	0.000%	
→\\APP\Os_Vector__text_inttab0_intvec_002@(interrupt):0	159.998us	6.430us	17.509us	8.889us	18.(0/2)	0.002%	
\\APP\Os_Wrapper\Os_ISRWrapper@(interrupt):0	158.558us	6.350us	17.429us	8.809us	18.(0/2)	0.086%	
\\APP\main\Os_Entry_MillisecondISR0@(interrupt):0	91.720us	3.710us	10.490us	5.096us	18.	0.012%	
\\APP\IncrementCounter\Os_IncrementCounter@(interrupt):0	85.020us	3.330us	10.110us	4.723us	18.	0.044%	
\\APP\ValidateCounter\Os_IsCounterValid@(interrupt):0	5.220us	0.290us	0.290us	0.290us	18.	0.009%	
\\APP\Os_Cfg_Counters\Os_IncrementCounter_MillisecondCounter0@(interrupt):0	55.440us	1.700us	8.480us	3.080us	18.	0.058%	
\\APP\Os_Cfg_Counters\Os_PerformEP@(interrupt):0	23.700us	5.310us	6.540us	5.925us	4.	0.024%	
\\APP\Os_Cfg_Counters\Os_ActivateTaskKL@(interrupt):0	10.560us	2.640us	2.640us	2.640us	4.	0.019%	
\\APP\DispatchTask\Os_Dispatch@(interrupt):0	19.858us	4.190us	5.739us	3.971us	5.(0/2)	0.026%	

Interrupt (branch to an address of an interrupt vector).



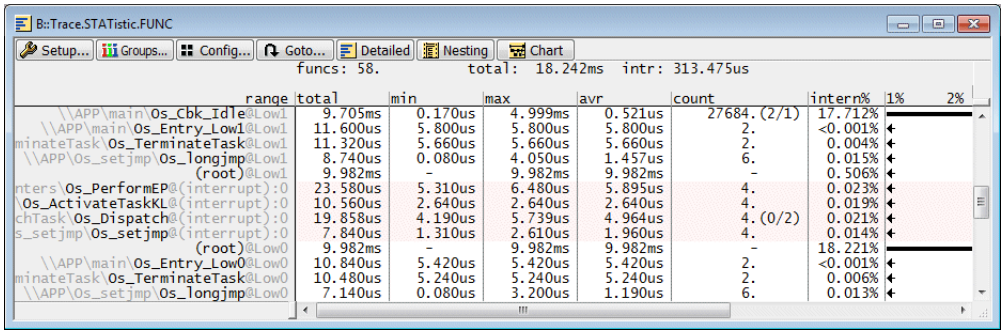
The unknown Task

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

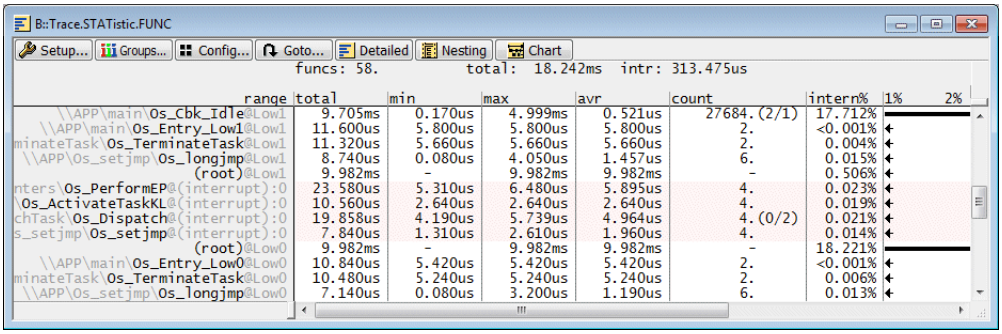


range	total	min	max	avr	count	intern%	1% 2
(root)@(unknown):0	2.729ms	-	2.729ms	2.729ms	-	4.999%	
(root)@(unknown):1	4.017ms	-	4.017ms	4.017ms	-	0.205%	
(root)@(unknown):2	18.194ms	-	18.194ms	18.194ms	-	33.333%	
\\APP\main\Os_Cbk_Idle@(unknown):1	3.905ms	0.000us	1.998ms	0.519us	11160.(0/2)	7.153%	

Since no trace information is recorded for TC 1.6.1 CPU2 total of (root)@(unknown):2 is equal to the complete recording time.



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



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

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

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

```
3671.(0/1)
```

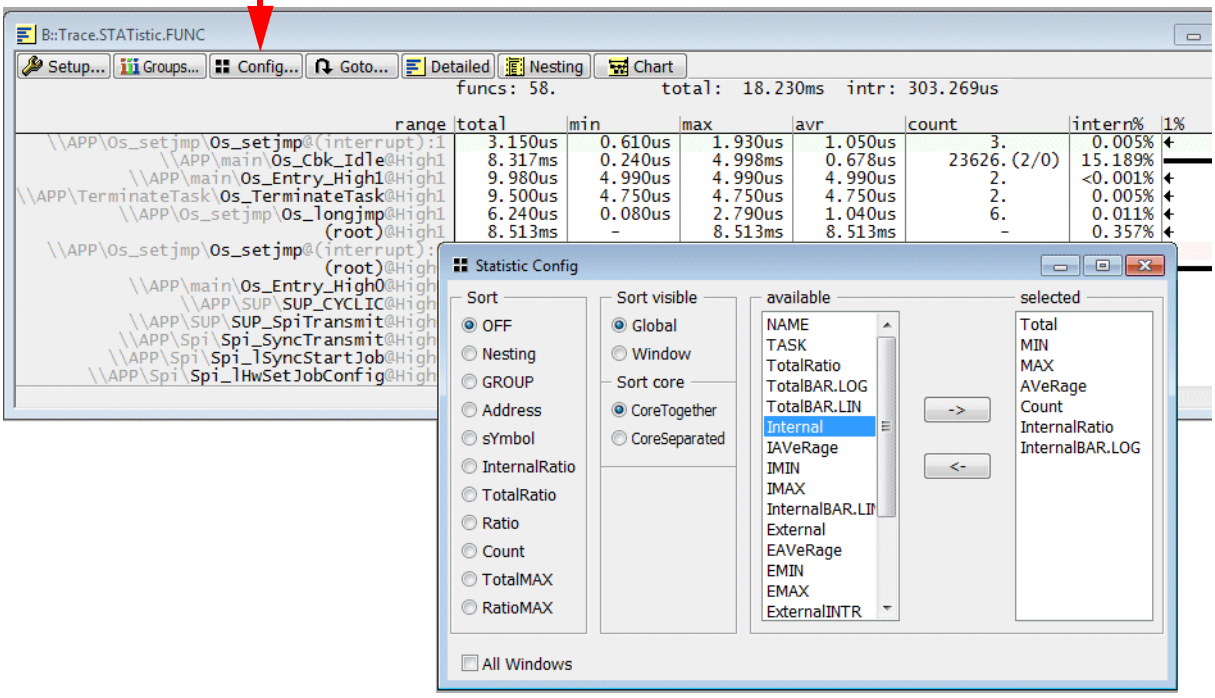
Interpretation examples:

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

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

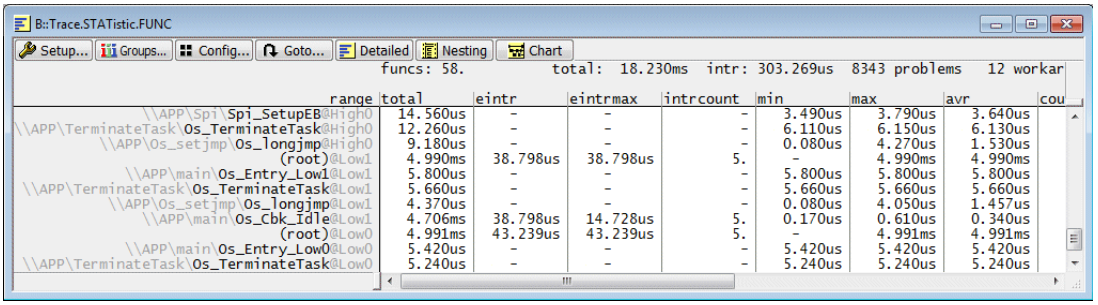
columns (cont.)	
intern% **(InternalRatio, InternalBAR.LOG)**	ratio of time within the function without subfunctions, TRAP handlers, interrupts

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

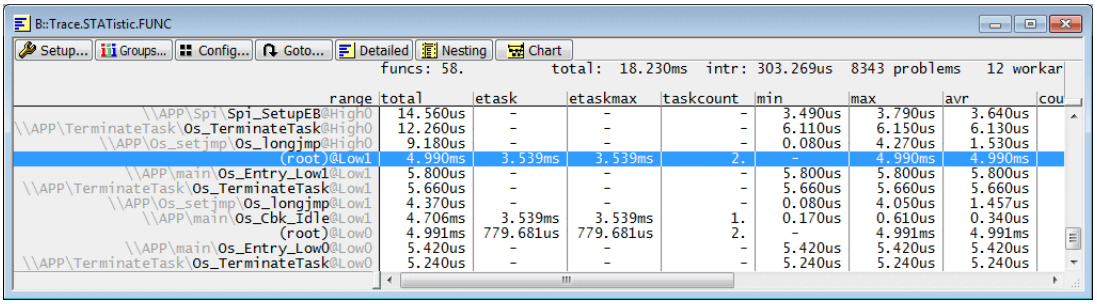


columns (cont.) - times only in function	
Internal	total time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IAVeRage	average time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines
IMIN	shortest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

columns (cont.) - times in sub-functions and TRAP handlers	
External	total time spent within called sub-functions/TRAP handlers
EAVeRage	average time spent within called sub-functions/TRAP handlers
EMIN	shortest time spent within called sub-functions/TRAP handlers
EMAX	longest time spent within called sub-functions/TRAP handlers

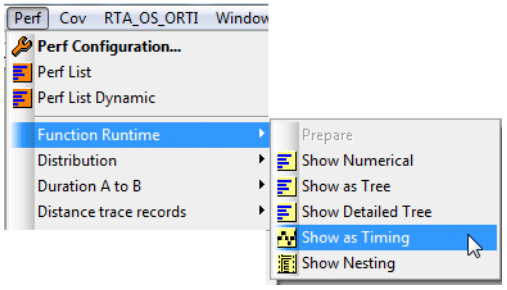


columns (cont.) - interrupt times	
ExternalINTR	total time the function was interrupted
ExternalINTRMAX	max. time one function pass was interrupted
INTRCount	number of interrupts that occurred during the function run-time

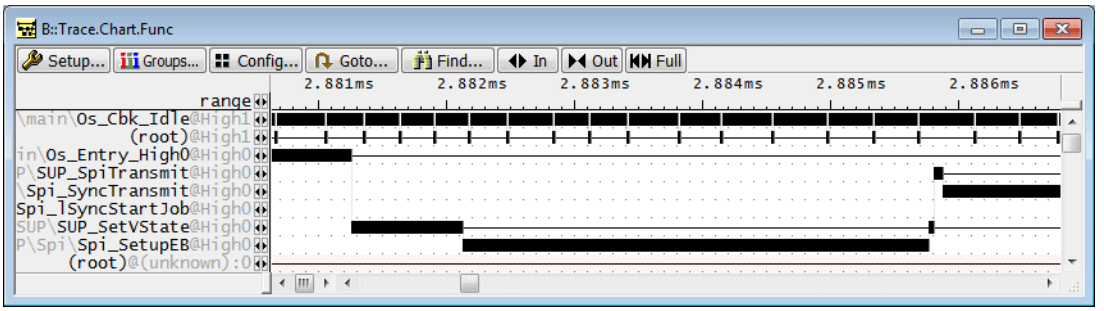


columns - task/thread related information	
TASKCount	number of tasks that interrupt the function/task
ExternalTASK	total time in other tasks
ExternalTASKMAX	max. time 1 function/task pass was interrupted by a task

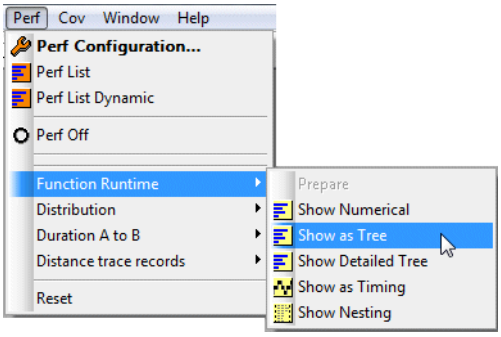
Trace.Chart.Func	Nesting function run-time analysis - graphical display



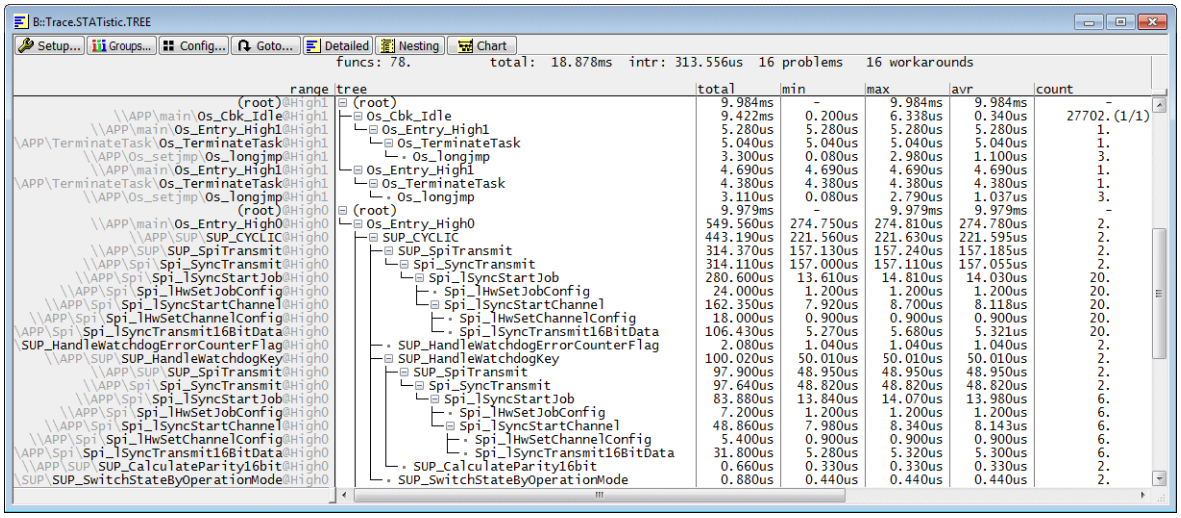
Look and Feel (OS)



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



Look and Feel (OS)

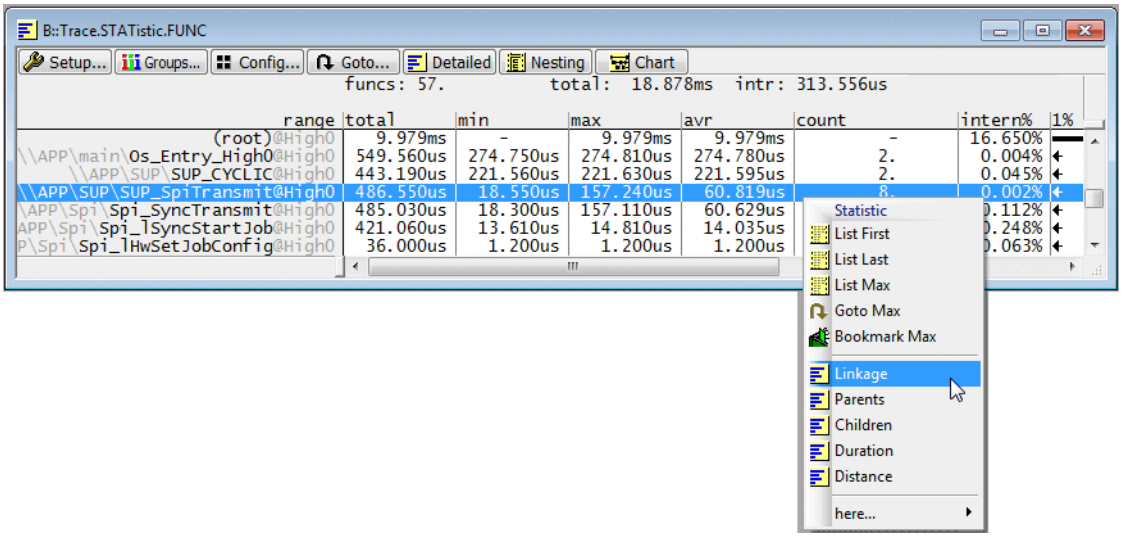


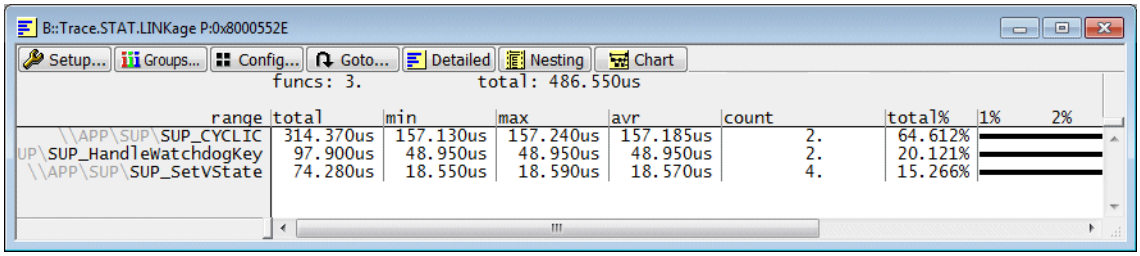
It is also possible to get a task/process-specific tree.

```
Trace.STATistic.TREE /TASK "High0"
```

Trace.STATistic.LINKage *<address>* Nesting function run-time analysis
- linkage analysis

Look and Feel (OS)





Trace-based Code Coverage

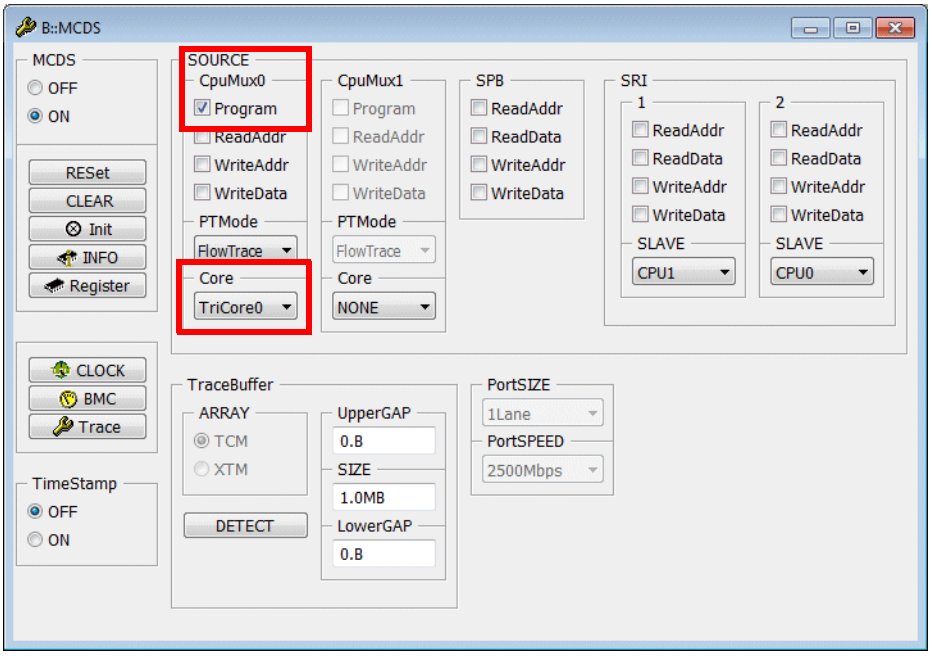
The manual **"Application Note for Trace-Based Code Coverage"** (app_code_coverage.pdf) gives a detailed introduction to the trace-based code coverage. However, the manual does not contain details about the architecture-specific setups. Here is an overview of the setups for TriCore™ AURIX™.

General SetUp

Single-Core and AMP Systems

The core under debug has to be configured for the trace multiplexer.

Instruction Pointer Call Messages are sufficient for Trace-based Code Coverage. Time information is not required.

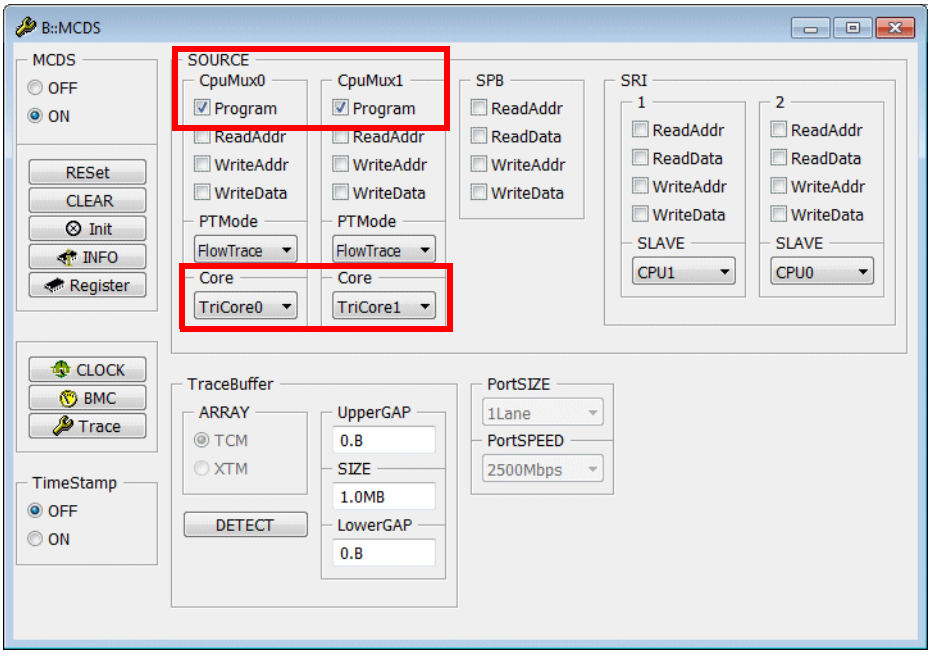



```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux0.Program ON       ; enable Instruction Pointer
                                         ; Call Messages for
                                         ; TC 1.6.1 CPU0
```

Since each core in a AMP system executes an independent task, Trace-based Code Coverage has to be performed per core.

Up to two cores can be configured for the trace multiplexer.

Instruction Pointer Call Messages are sufficient for Trace-based Code Coverage. Time information is not required.




```
MCDS.SOURCE.Set CpuMux0.Core TriCore0    ; enable TC 1.6.1 CPU0 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux1.Core TriCore1    ; enable TC 1.6.1 CPU1 as
                                         ; trace source

MCDS.SOURCE.Set CpuMux0.Program ON       ; enable Instruction Pointer
                                         ; Call Messages for
                                         ; TC 1.6.1 CPU0

MCDS.SOURCE.Set CpuMux1.Program ON       ; enable Instruction Pointer
                                         ; Call Messages for
                                         ; TC 1.6.1 CPU1
```

Since the core information is discarded for the Trace-based Code Coverage, the same procedure can be used as for single-core/AMP systems.