Trace Export for Third-Party Timing Tools

Release 09.2023

Trace Export for Third-Party Timing Tools

Trace Export for Third-Party Timing Tools

Version 09-Oct-2023

Introduction

There are timing tools on the market that are specialized in trace-based timing analysis and visualization. Examples of such tools are:

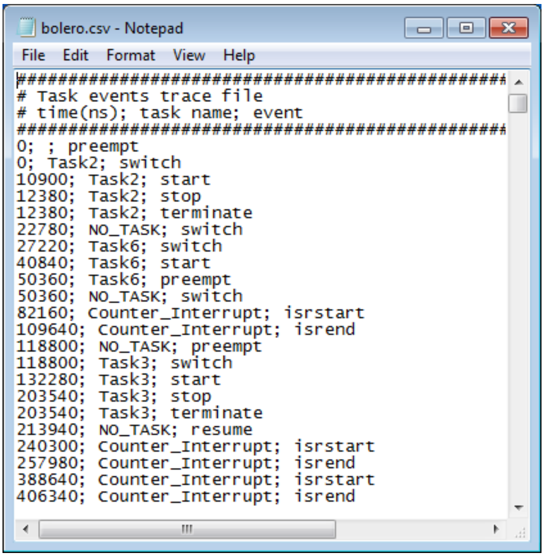
- **Symtavision:** TraceAnalyzer
- **INCHRON:** chronVIEW
- **Timing Architects:** Inspector

TRACE32 provides the following command to export trace information, recorded with TRACE32, for analysis with a third-party timing tool:

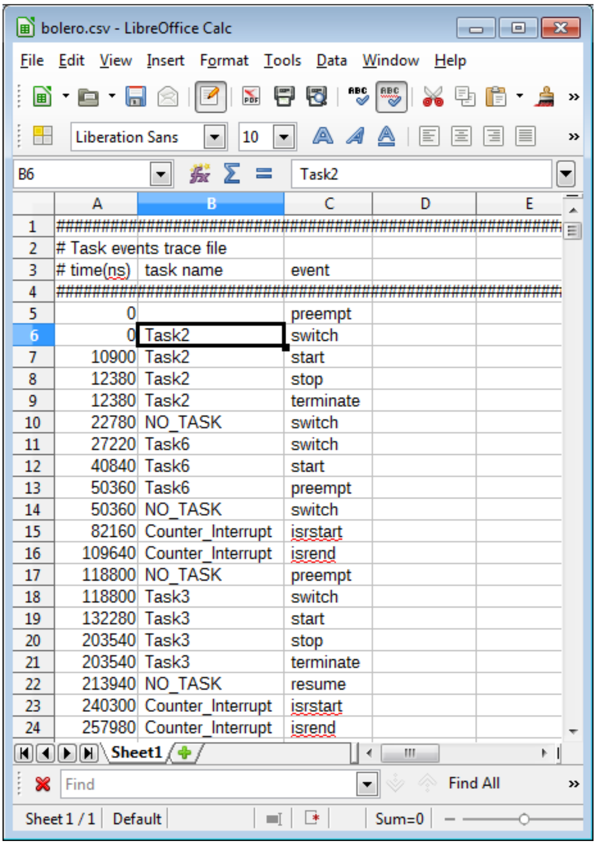
```
Trace.EXPORT.TASKEVENTS <file>
```

All timing tools listed above are used in the automotive industry, so we limit ourselves in this document to AUTOSAR/OSEK operating systems. But the topic can be, of course, applied to other operating systems too.

The command **Trace.EXPORT.TASKEVENTS** generates a CSV (Comma-Separated Values) file that includes task events and their timing. See screenshots below. The generated format is intentionally generic so that it is suitable for any tool or any proprietary analysis.



```
#################################################
# Task events trace file
# time(ns); task name; event
#################################################
0; ; preempt
0; Task2; switch
10900; Task2; start
12380; Task2; stop
12380; Task2; terminate
22780; NO_TASK; switch
27220; Task6; switch
40840; Task6; start
50360; Task6; preempt
50360; NO_TASK; switch
82160; Counter_Interrupt; isrstart
109640; Counter_Interrupt; isrend
118800; NO_TASK; preempt
118800; Task3; switch
132280; Task3; start
203540; Task3; stop
203540; Task3; terminate
213940; NO_TASK; resume
240300; Counter_Interrupt; isrstart
257980; Counter_Interrupt; isrend
388640; Counter_Interrupt; isrstart
406340; Counter_Interrupt; isrend
```


bolero.csv - LibreOffice Calc
File Edit View Insert Format Tools Data Window Help
Liberation Sans
10
B6
Task2
Sheet1
Find
Find All
Sheet 1 / 1
Default
Sum=0

	A	B	C	D	E
1	##				
2	# Task events trace file				
3	# time(ns)	task name	event		
4	##				
5	0		preempt		
6	0	Task2	switch		
7	10900	Task2	start		
8	12380	Task2	stop		
9	12380	Task2	terminate		
10	22780	NO_TASK	switch		
11	27220	Task6	switch		
12	40840	Task6	start		
13	50360	Task6	preempt		
14	50360	NO_TASK	switch		
15	82160	Counter_Interrupt	isrstart		
16	109640	Counter_Interrupt	isrend		
17	118800	NO_TASK	preempt		
18	118800	Task3	switch		
19	132280	Task3	start		
20	203540	Task3	stop		
21	203540	Task3	terminate		
22	213940	NO_TASK	resume		
23	240300	Counter_Interrupt	isrstart		
24	257980	Counter_Interrupt	isrend		

Requirements

Recorded trace information has to fulfil the following requirements before it can be exported by the **Trace.EXPORT.TASKEVENTS** command:

- The recorded trace has to include the complete instruction execution sequence plus all task switches.
- All functions that start a task have to be marked with a TASKSTART marker.

```
sYmbol.MARKER.Create TASKSTART <address>
```

- All functions that terminate a task have to be marked with a TASKTERMINATE marker.

```
sYmbol.MARKER.Create TASKTERMINATE <address>
```

- All functions that start an interrupt service routine have to be marked with an ISRSTART marker (AUTOSAR/OSEK specific).

```
sYmbol.MARKER.Create ISRSTART <address>
```

- All functions that terminate an interrupt service routine have to be marked with an ISREND marker (AUTOSAR/OSEK specific).

```
sYmbol.MARKER.Create ISREND <address>
```

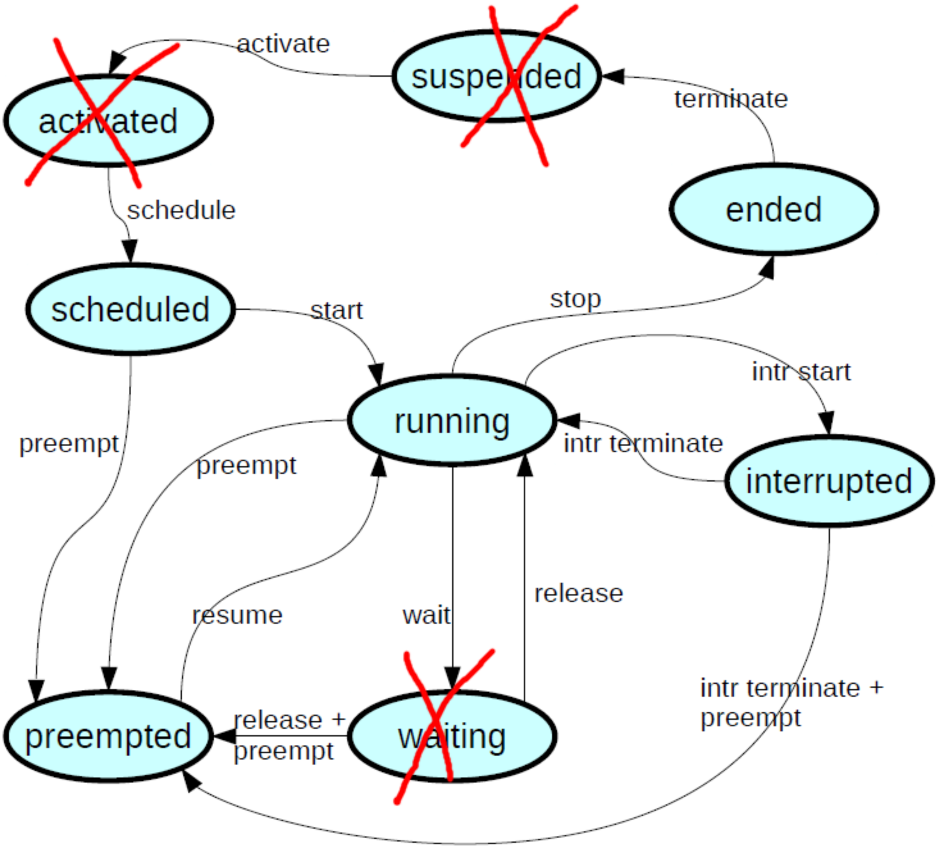
- All functions that start an AUTOSAR “Runnable” have to be marked with a RUNNABLESTARTPLUSSTOP marker (AUTOSAR specific). The end of the function is automatically used as end of this runnable.

```
sYmbol.MARKER.Create RUNNABLESTARTPLUSSTOP <address>
```

Processing

The task events are identified by processing the instruction execution sequence and the task switches recorded in the trace. The picture shows the state machine used.

However, not all states can be identified. States that cannot be identified are crossed out. E.g. the state "waiting" cannot be identified – instead the state "preempted" is reached.



The **events** (state transitions) in the CSV file have the following meanings:

activate	a suspended task is activated and goes into “ready” state
schedule	an activated task is scheduled for running in the OS
start	the function body of a task is called
stop	the task ends by itself (by ending the function or terminating)
terminate	the task is terminated
preempt	the task is preempted by a higher prio task
resume	the task is resumed from preemption and scheduled
wait	the task goes into waiting state

release	the task is released from waiting state and scheduled
switch	the task is scheduled for running, but the previous state is unknown; could be schedule, resume or release.
runnablestart	the function body of a “runnable” is called.
runnablestop	the function of a “runnable” exited.

Example

To better understand how the trace recording has to be prepared so that the task events and their timing can be exported with the command **Trace.EXPORT.TASKEVENTS**, we present a complete example. Important are especially steps 3-6.

Related Documents

The following documents can help you to better understand the demonstrated example:

- **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf).
- **"Training Nexus Tracing"** (training_nexus.pdf).

Environment

For the example we are using an OSEK/VDX application based on ERIKA Enterprise.

The whole workspace, including a ready-compiled ELF file, is available in the TRACE32 demo directory: `~~/demo/powerpc/kernel/erika` (example for the TRACE32 Instruction Set Simulator).

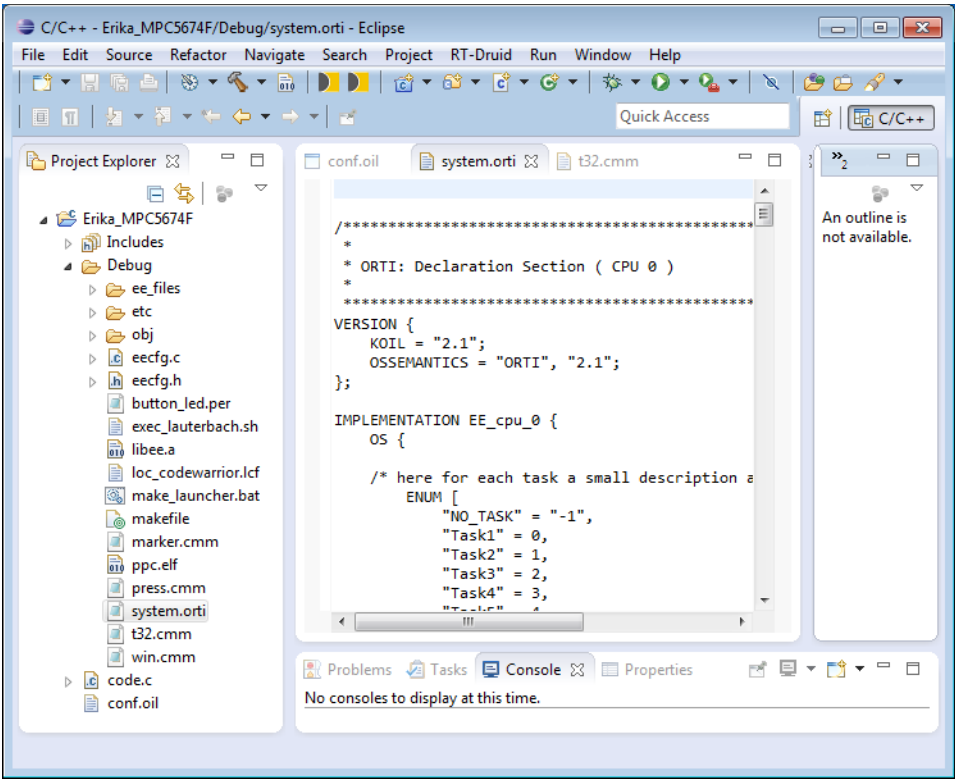
The development environment is available free of charge at **http://erika.tuxfamily.org**.

The binary build with ERIKA Enterprise will be executed on:

- A TRACE32 PowerPC Instruction Set Simulator.
- A Lauterbach Bolero MPC5646C evaluation board using TRACE32 hardware-based debug and trace tools.

Step 1: Create an AUTOSAR/OSEK Application

Create your AUTOSAR/OSEK application as usual. Instruct the builder to create an ORTI file – this is essential for the following process. If necessary, insert a PreTaskHook to export task switches to the trace. See also **“OS Awareness Manual OSEK/ORTI”** (rtos_orti.pdf).



```
/**********************************************
 *
 * ORTI: Declaration Section ( CPU 0 )
 *
 **********************************************
VERSION {
    KOIL = "2.1";
    OSSEMANTICS = "ORTI", "2.1";
};

IMPLEMENTATION EE_cpu_0 {
    OS {

    /* here for each task a small description a
        ENUM [
            "NO_TASK" = "-1",
            "Task1" = 0,
            "Task2" = 1,
            "Task3" = 2,
            "Task4" = 3,
```

An outline is not available.

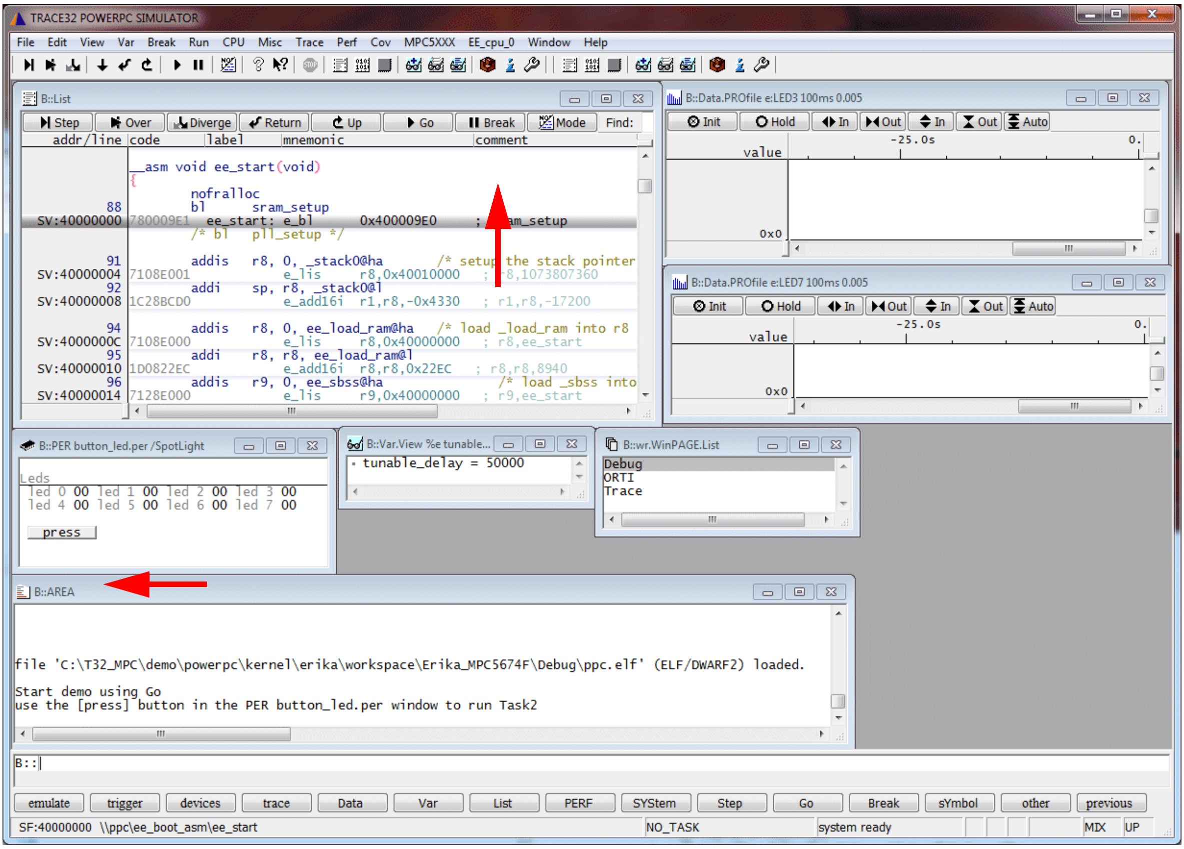
No consoles to display at this time.

Step 2: Set up TRACE32 and Run the Application

After the application (ppc.elf) and ORTI file (system.orti) is built, set up your debug environment and load the files.

Data.LOAD.Elf *<file>*	Load the ELF file
TASK.ORTI *<file>*	Load the ORTI file

The script work-settings.cmm in the "Debug" directory of the ERIKA project can be used as an example.


TRACE32 POWERPC SIMULATOR
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC5XXX EE_cpu_0 Window Help
B::List
Step Over Diverge Return Up Go Break Mode Find:
B::Data.PROfile e:LED3 100ms 0.005
Init Hold In Out In Out Auto
-25.0s
0.
value
0x0
B::Data.PROfile e:LED7 100ms 0.005
Init Hold In Out In Out Auto
-25.0s
0.
value
0x0
B::PER button_led.per /SpotLight
B::Var.View %e tunable...
B::wr.WinPAGE.List
B::AREA
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol other previous
MIX UP

```
   addr/line code         label      mnemonic                   comment
                 __asm void ee_start(void)
                 {
                         nofralloc
          88             bl       sram_setup
 SV:40000000 780009E1  ee_start: e_bl       0x400009E0        ; sram_setup
                         /* bl    pll_setup */

          91             addis    r8, 0, _stack0@ha        /* setup the stack pointer
 SV:40000004 7108E001             e_lis      r8,0x40010000    ; r8,1073807360
          92             addi     sp, r8, _stack0@l
 SV:40000008 1C28BCD0             e_add16i   r1,r8,-0x4330    ; r1,r8,-17200

          94             addis    r8, 0, ee_load_ram@ha   /* load _load_ram into r8
 SV:4000000C 7108E000             e_lis      r8,0x40000000    ; r8,ee_start
          95             addi     r8, r8, ee_load_ram@l
 SV:40000010 1D0822EC             e_add16i   r8,r8,0x22EC    ; r8,r8,8940
          96             addis    r9, 0, ee_sbss@ha             /* load _sbss into
 SV:40000014 7128E000             e_lis      r9,0x40000000    ; r9,ee_start
```

```
Leds
 led 0 00 led 1 00 led 2 00 led 3 00
 led 4 00 led 5 00 led 6 00 led 7 00

 press
```

```
tunable_delay = 50000
```

```
Debug
ORTI
Trace
```

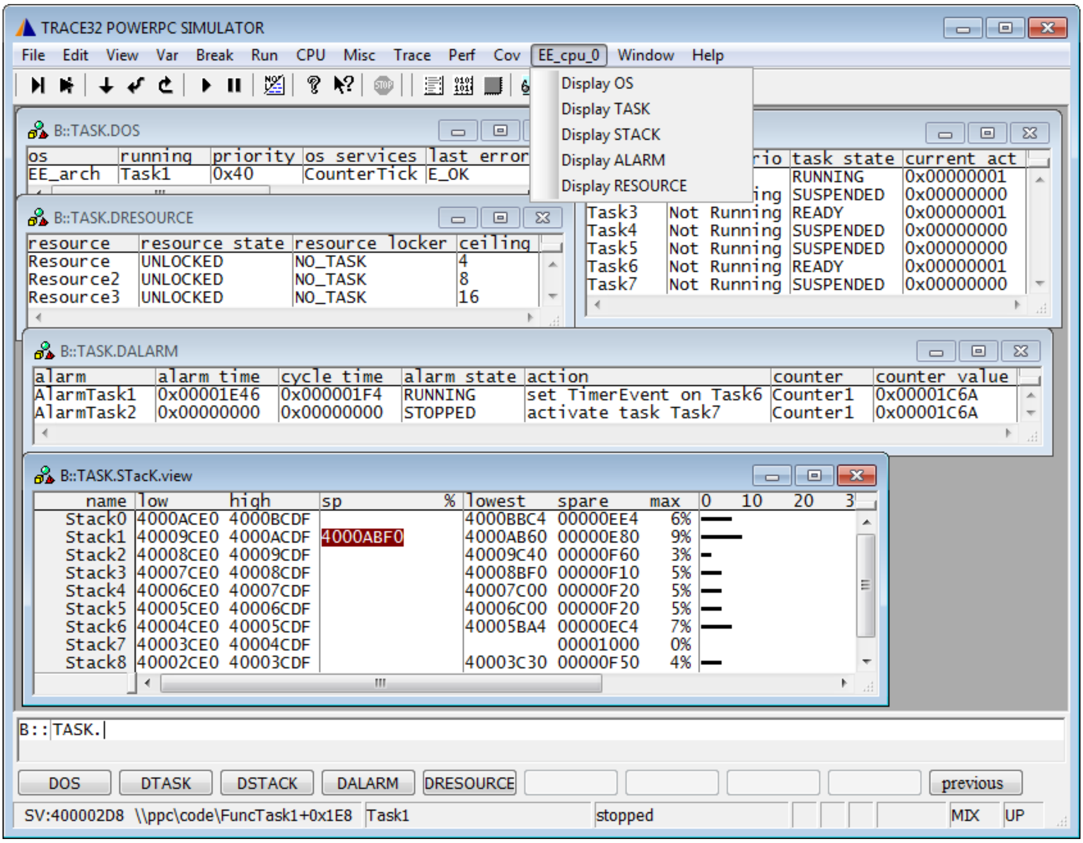
```
file 'C:\T32_MPC\demo\powerpc\kernel\erika\workspace\Erika_MPC5674F\Debug\ppc.elf' (ELF/DWARF2) loaded.

Start demo using Go
use the [press] button in the PER button_led.per window to run Task2
```

```
B::
SF:40000000  \\ppc\ee_boot_asm\ee_start                NO_TASK       system ready
```

Use the **press** button in the **PER button_led.per** window to start the demo and use the **break** button to stop it.

TRACE32 PowerView shows the OS resources after the demo stopped.



os	running	priority	os services	last error
EE_arch	Task1	0x40	CounterTick	E_OK

resource	resource state	resource locker	ceiling
Resource	UNLOCKED	NO_TASK	4
Resource2	UNLOCKED	NO_TASK	8
Resource3	UNLOCKED	NO_TASK	16

	rio	task state	current act
		RUNNING	0x00000001
	ing	SUSPENDED	0x00000000
Task3	Not Running	READY	0x00000001
Task4	Not Running	SUSPENDED	0x00000000
Task5	Not Running	SUSPENDED	0x00000000
Task6	Not Running	READY	0x00000001
Task7	Not Running	SUSPENDED	0x00000000

alarm	alarm time	cycle time	alarm state	action	counter	counter value
AlarmTask1	0x00001E46	0x000001F4	RUNNING	set TimerEvent on Task6	Counter1	0x00001C6A
AlarmTask2	0x00000000	0x00000000	STOPPED	activate task Task7	Counter1	0x00001C6A

name	low	high	sp	%	lowest	spare	max	0 10 20 3
Stack0	4000ACE0	4000BCDF			4000BBC4	00000EE4	6%	▬
Stack1	40009CE0	4000ACDF	4000ABF0		4000AB60	00000E80	9%	▬
Stack2	40008CE0	40009CDF			40009C40	00000F60	3%	▬
Stack3	40007CE0	40008CDF			40008BF0	00000F10	5%	▬
Stack4	40006CE0	40007CDF			40007C00	00000F20	5%	▬
Stack5	40005CE0	40006CDF			40006C00	00000F20	5%	▬
Stack6	40004CE0	40005CDF			40005BA4	00000EC4	7%	▬
Stack7	40003CE0	40004CDF				00001000	0%	
Stack8	40002CE0	40003CDF			40003C30	00000F50	4%	▬

```
B::TASK.
```

SV:400002D8 \\ppc\code\FuncTask1+0x1E8 Task1 stopped MIX UP

Step 3: Set up Real-time Trace within TRACE32

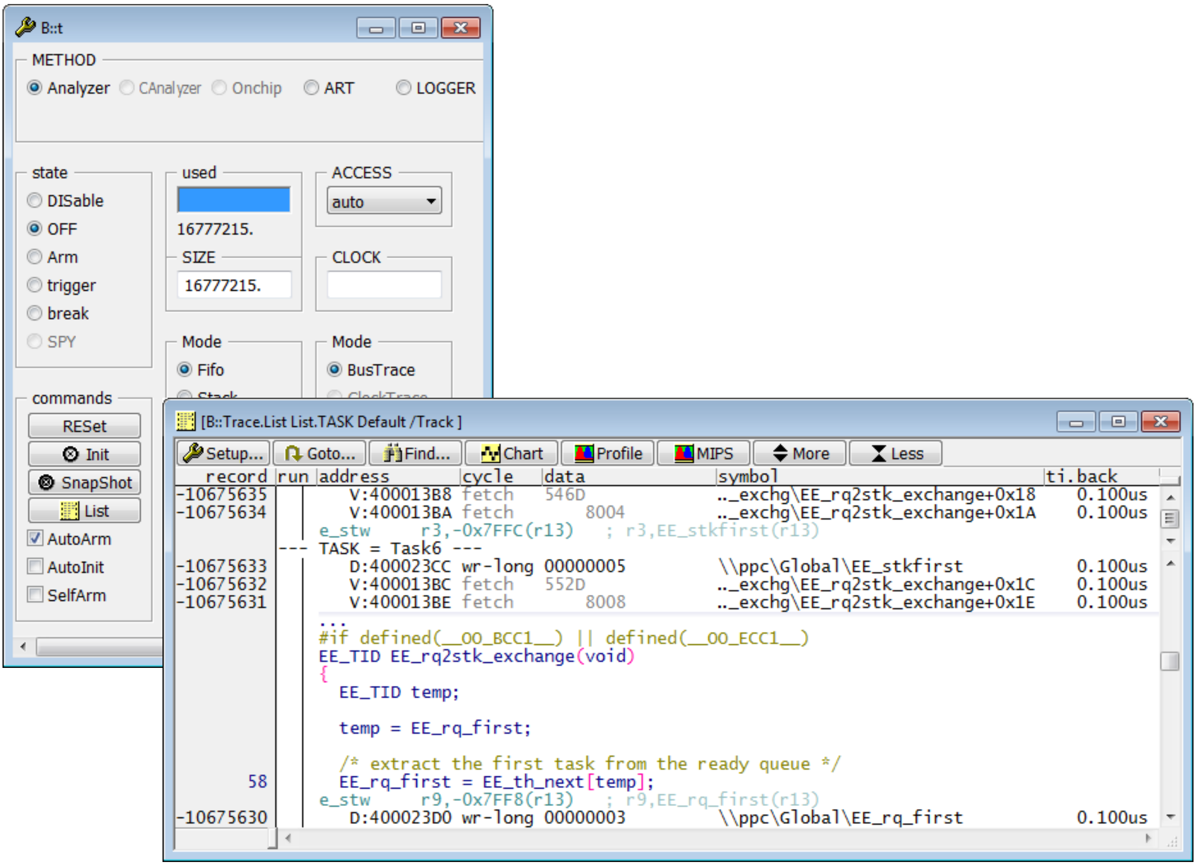
In order to provide all information for a detailed task analysis, the trace logic on the target has to be configured to provide the complete instruction execution sequence plus all task switches.

TRACE32 Instruction Set Simulator

No special configuration is required for the TRACE32 Instruction Set Simulator.

It is recommended to increase the size of the simulated trace memory (as done is our example script).

```
Trace.SIZE 16777215.
```



B::t

METHOD: ● Analyzer ○ CAnalyzer ○ Onchip ○ ART ○ LOGGER

state: ○ DISable ● OFF ○ Arm ○ trigger ○ break ○ SPY

used: 16777215.

SIZE: 16777215.

ACCESS: auto

CLOCK:

Mode: ● Fifo ○ Stack

Mode: ● BusTrace ○ ClockTrace

commands: ☑ AutoArm □ AutoInit □ SelfArm

[B::Trace.List List.TASK Default /Track]

```
   record run address    cycle   data              symbol                             ti.back
-10675635      V:400013B8 fetch   546D              .._exchg\EE_rq2stk_exchange+0x18     0.100us
-10675634      V:400013BA fetch         8004        .._exchg\EE_rq2stk_exchange+0x1A     0.100us
           e_stw     r3,-0x7FFC(r13)   ; r3,EE_stkfirst(r13)
          --- TASK = Task6 ---
-10675633      D:400023CC wr-long 00000005          \\ppc\Global\EE_stkfirst             0.100us
-10675632      V:400013BC fetch   552D              .._exchg\EE_rq2stk_exchange+0x1C     0.100us
-10675631      V:400013BE fetch         8008        .._exchg\EE_rq2stk_exchange+0x1E     0.100us
           ...
           #if defined(__OO_BCC1__) || defined(__OO_ECC1__)
           EE_TID EE_rq2stk_exchange(void)
           {
             EE_TID temp;

             temp = EE_rq_first;

             /* extract the first task from the ready queue */
       58    EE_rq_first = EE_th_next[temp];
           e_stw     r9,-0x7FF8(r13)   ; r9,EE_rq_first(r13)
-10675630      D:400023D0 wr-long 00000003          \\ppc\Global\EE_rq_first             0.100us
```

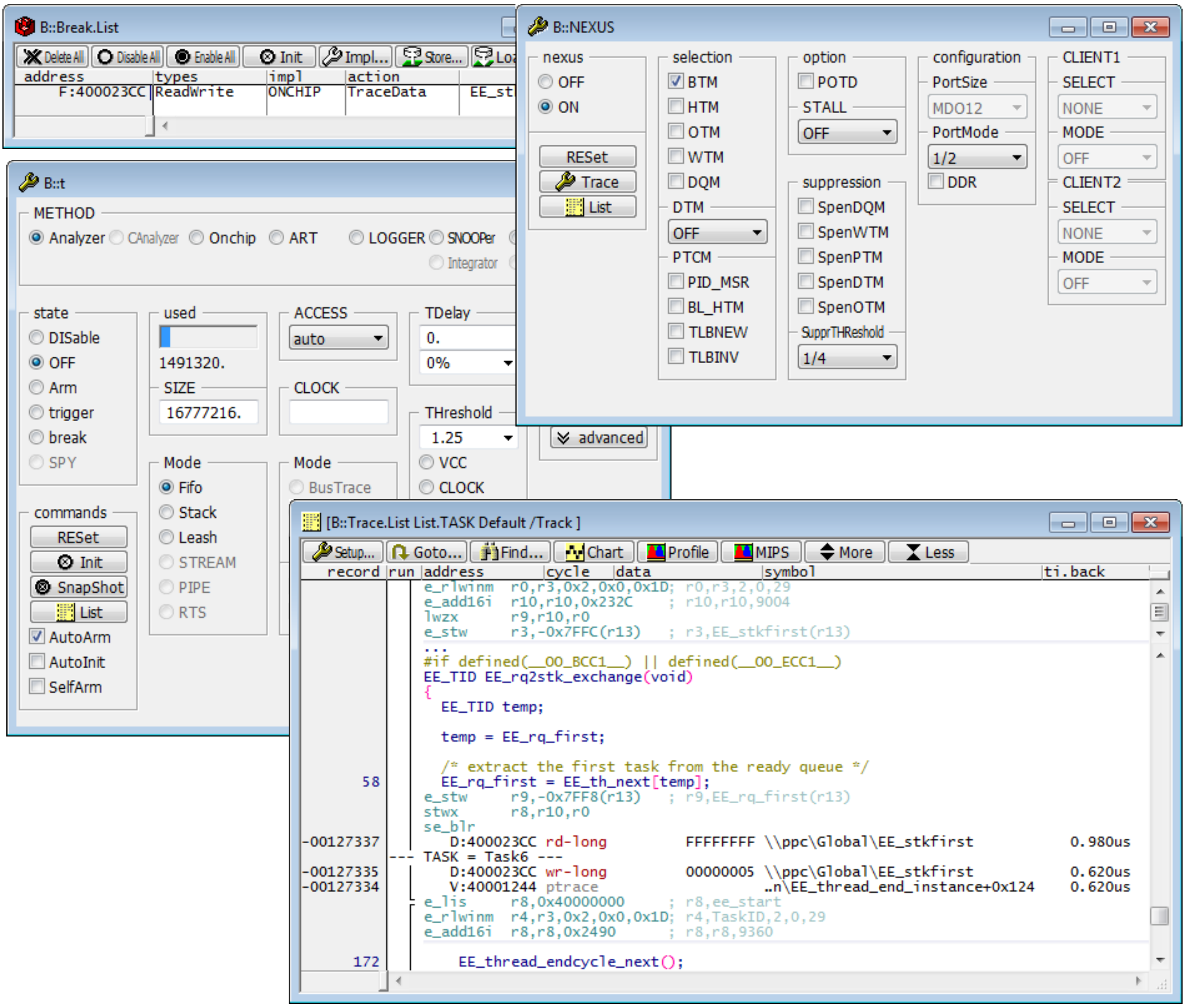
NEXUS Class 3

If your chip provides a NEXUS Class 3+ module, this NEXUS module has to be configured to generated trace information for the instruction execution sequence and the task switches.

For details refer to **"OS-Aware Tracing (ORTI File)"** in Nexus Training, page 187 (training_nexus.pdf).

```
NEXUS.BTM ON

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



address	types	impl	action	
F:400023CC	ReadWrite	ONCHIP	TraceData	EE_st

METHOD: ● Analyzer ○ CAnalyzer ○ Onchip ○ ART ○ LOGGER ○ SNOOPer ○ Integrator

state: ○ DISable ● OFF ○ Arm ○ trigger ○ break ○ SPY

used: 1491320.

SIZE: 16777216.

ACCESS: auto

CLOCK:

TDelay: 0. 0%

THreshold: 1.25 ○ VCC ○ CLOCK

Mode: ● Fifo ○ Stack ○ Leash ○ STREAM ○ PIPE ○ RTS

Mode: ○ BusTrace

☑ AutoArm
□ AutoInit
□ SelfArm

nexus: ○ OFF ● ON

selection: ☑ BTM □ HTM □ OTM □ WTM □ DQM

DTM: OFF

PTCM: □ PID_MSR □ BL_HTM □ TLBNEW □ TLBINV

option: □ POTD

STALL: OFF

suppression: □ SpenDQM □ SpenWTM □ SpenPTM □ SpenDTM □ SpenOTM

SupprTHReshold: 1/4

configuration: PortSize: MDO12

PortMode: 1/2 □ DDR

CLIENT1: SELECT: NONE MODE: OFF

CLIENT2: SELECT: NONE MODE: OFF

```
   record |run |address          |cycle   |data               |symbol                              |ti.back
              e_rlwinm  r0,r3,0x2,0x0,0x1D; r0,r3,2,0,29
              e_add16i  r10,r10,0x232C    ; r10,r10,9004
              lwzx      r9,r10,r0
              e_stw     r3,-0x7FFC(r13)   ; r3,EE_stkfirst(r13)
              ...
              #if defined(__OO_BCC1__) || defined(__OO_ECC1__)
              EE_TID EE_rq2stk_exchange(void)
              {
                EE_TID temp;

                temp = EE_rq_first;

                /* extract the first task from the ready queue */
       58       EE_rq_first = EE_th_next[temp];
              e_stw     r9,-0x7FF8(r13)   ; r9,EE_rq_first(r13)
              stwx      r8,r10,r0
              se_blr
-00127337        D:400023CC rd-long          FFFFFFFF \\ppc\Global\EE_stkfirst             0.980us
          --- TASK = Task6 ---
-00127335        D:400023CC wr-long          00000005 \\ppc\Global\EE_stkfirst             0.620us
-00127334        V:40001244 ptrace                    ..n\EE_thread_end_instance+0x124     0.620us
              e_lis     r8,0x40000000     ; r8,ee_start
              e_rlwinm  r4,r3,0x2,0x0,0x1D; r4,TaskID,2,0,29
              e_add16i  r8,r8,0x2490      ; r8,r8,9360

      172         EE_thread_endcycle_next();
```

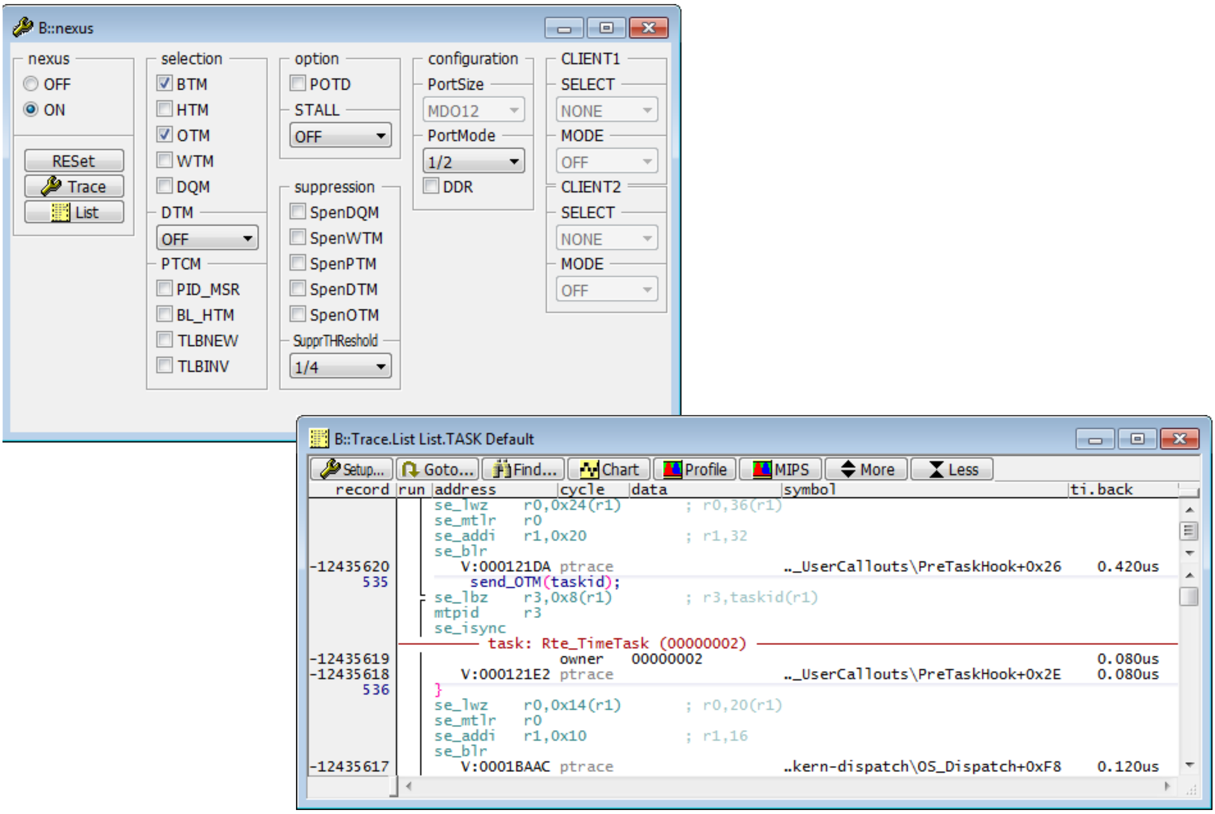
If your chip provides a NEXUS Class 2+ module, this NEXUS module has to be configured to generated trace information for the instruction execution sequence and the task switches.

```
NEXUS.BTM ON

NEXUS.OTM ON
```

For details refer to **"OS-Aware Tracing (ORTI File)"** in Nexus Training, page 187 (training_nexus.pdf).

You may need to write a PreTaskHook for this, if your OS version does not support ownership trace messages on task switches.



B::nexus

nexus: ○ OFF ● ON

selection: ☑ BTM □ HTM ☑ OTM □ WTM □ DQM

DTM: OFF

PTCM: □ PID_MSR □ BL_HTM □ TLBNEW □ TLBINV

option: □ POTD

STALL: OFF

suppression: □ SpenDQM □ SpenWTM □ SpenPTM □ SpenDTM □ SpenOTM

SupprTHReshold: 1/4

configuration: PortSize: MDO12; PortMode: 1/2; □ DDR

CLIENT1: SELECT: NONE; MODE: OFF

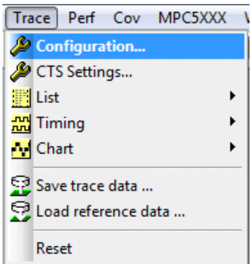
CLIENT2: SELECT: NONE; MODE: OFF

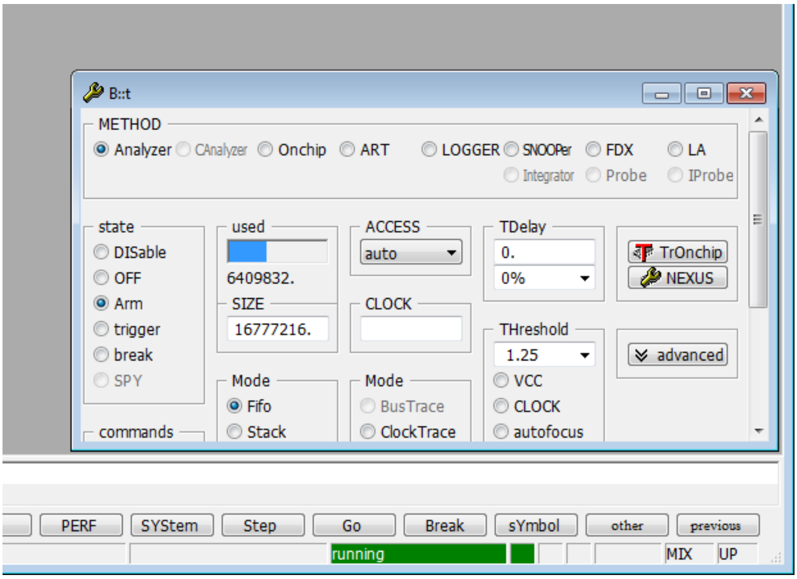
B::Trace.List List.TASK Default

```
    record|run |address       |cycle  |data              |symbol                           |ti.back
                  se_lwz     r0,0x24(r1)        ; r0,36(r1)
                  se_mtlr    r0
                  se_addi    r1,0x20            ; r1,32
                  se_blr
 -12435620          V:000121DA ptrace                     .._UserCallouts\PreTaskHook+0x26     0.420us
       535            send_OTM(taskid);
                  se_lbz     r3,0x8(r1)         ; r3,taskid(r1)
                  mtpid      r3
                  se_isync
           ---------- task: Rte_TimeTask (00000002) ------------------------------------------------
 -12435619                     owner    00000002                                                0.080us
 -12435618          V:000121E2 ptrace                     .._UserCallouts\PreTaskHook+0x2E     0.080us
       536        }
                  se_lwz     r0,0x14(r1)        ; r0,20(r1)
                  se_mtlr    r0
                  se_addi    r1,0x10            ; r1,16
                  se_blr
 -12435617          V:0001BAAC ptrace                     ..kern-dispatch\OS_Dispatch+0xF8     0.120us
```

Step 4: Run the Program Execution to Fill the Trace

Display a Trace Configuration window (**Trace.state)** and start the program execution.





METHOD
● Analyzer ○ CAnalyzer ○ Onchip ○ ART ○ LOGGER ○ SNOOPer ○ FDX ○ LA
○ Integrator ○ Probe ○ IProbe

state
○ DISable
○ OFF
● Arm
○ trigger
○ break
○ SPY

used
6409832.

SIZE
16777216.

ACCESS
auto

CLOCK

TDelay
0.
0%

THreshold
1.25
○ VCC
○ CLOCK
○ autofocus

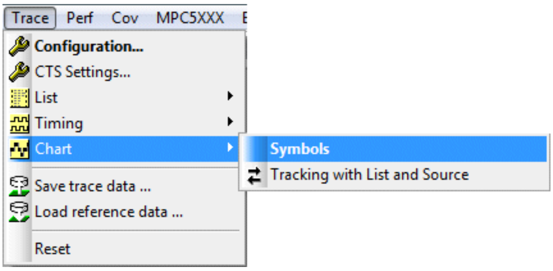
Mode
● Fifo
○ Stack

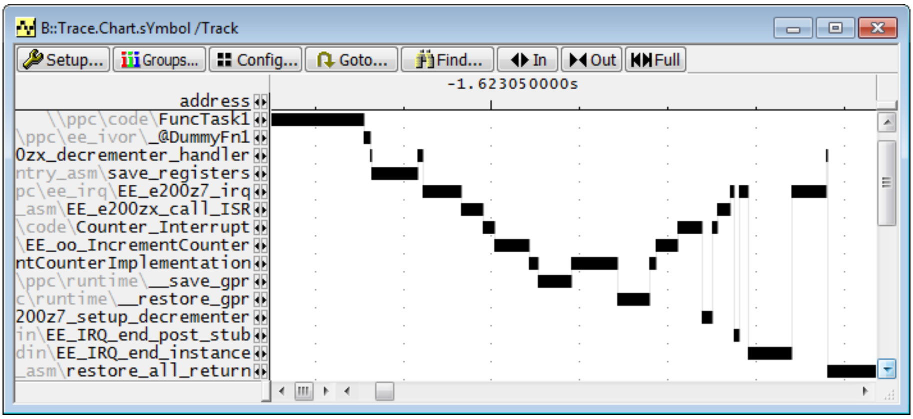
Mode
○ BusTrace
○ ClockTrace

running MIX UP

Stop the program execution by pushing the [Break] button.

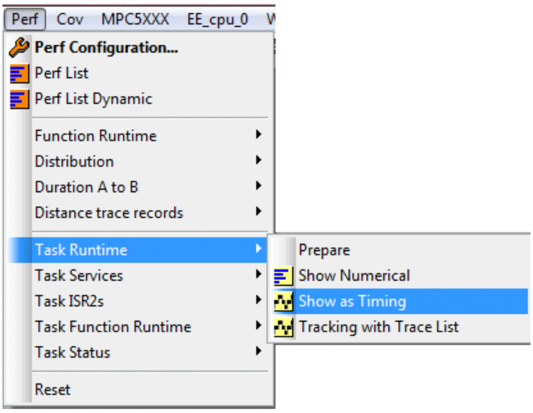
Use the **Trace.Chart.sYmbol** command to check if the trace information was recorded without errors.

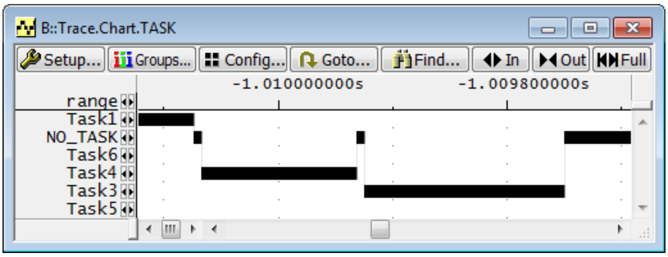




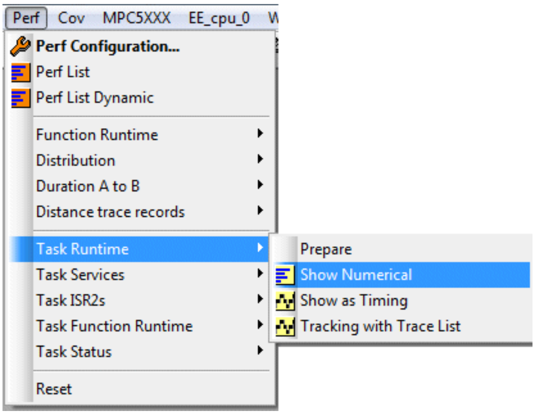
Details on possible errors and their causes can be found in **"FlowErrors"** in Nexus Training, page 48 (training_nexus.pdf) and **"FIFOFULL"** in Nexus Training, page 44 (training_nexus.pdf). Please be aware that an error-free trace is required in order to export task event information.

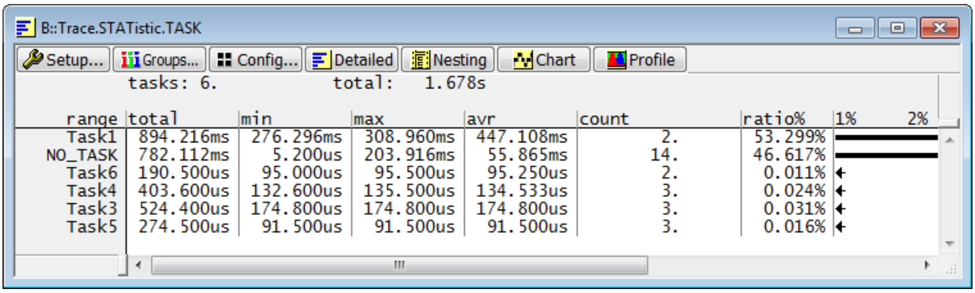
Use the **Trace.Chart.TASK** command to inspect the task switches.





The command **Trace.STATistic.TASK** provides the same result in a numeric display.





```
            tasks: 6.          total:    1.678s

   range |total     |min       |max       |avr       |count         |ratio%  |1%      2%
   Task1  894.216ms  276.296ms  308.960ms  447.108ms            2.    53.299%
 NO_TASK  782.112ms    5.200us  203.916ms   55.865ms           14.    46.617%
   Task6  190.500us   95.000us   95.500us   95.250us            2.     0.011%
   Task4  403.600us  132.600us  135.500us  134.533us            3.     0.024%
   Task3  524.400us  174.800us  174.800us  174.800us            3.     0.031%
   Task5  274.500us   91.500us   91.500us   91.500us            3.     0.016%
```

Step 5: Set up Markers for Trace Export

In order to identify the task events exported by the command **Trace.EXPORT.TASKEVENTS** the following program events have to be marked in the trace recording:

- Start addresses of tasks.
- Termination calls (if any).
- ISR routines.

In the example here, we declared:

- All task function entries as TASKSTART.
- The OS_TerminateTask call as TASKTERMINATE (another may be OS_ChainTask, which is not used here).
- The entry to the Counter_Interrupt routine as ISRSTART.
- The exit of the Counter_Interrupt routine as ISREND.

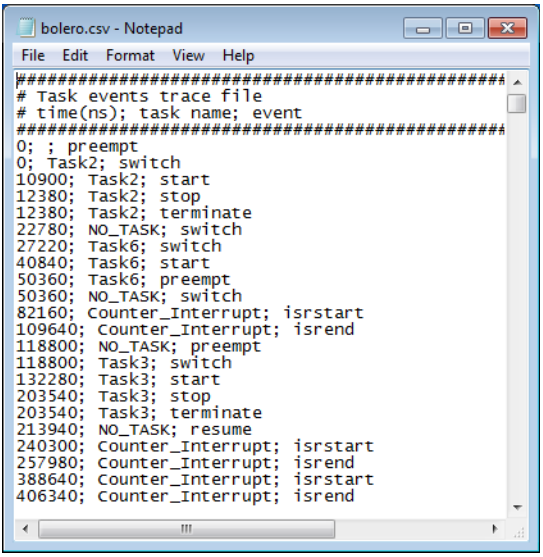
```
sYmbol.MARKER.Create TASKSTART FuncTask1
sYmbol.MARKER.Create TASKSTART FuncTask2
sYmbol.MARKER.Create TASKSTART FuncTask3
sYmbol.MARKER.Create TASKSTART FuncTask4
sYmbol.MARKER.Create TASKSTART FuncTask5
sYmbol.MARKER.Create TASKSTART FuncTask6
sYmbol.MARKER.Create TASKSTART FuncTask7
sYmbol.MARKER.Create TASKTERMINATE EE_oo_TerminateTask
sYmbol.MARKER.Create ISRSTART Counter_Interrupt
sYmbol.MARKER.Create ISREND sYmbol.EXIT(Counter_Interrupt)
```

Step 6: Export Task Events

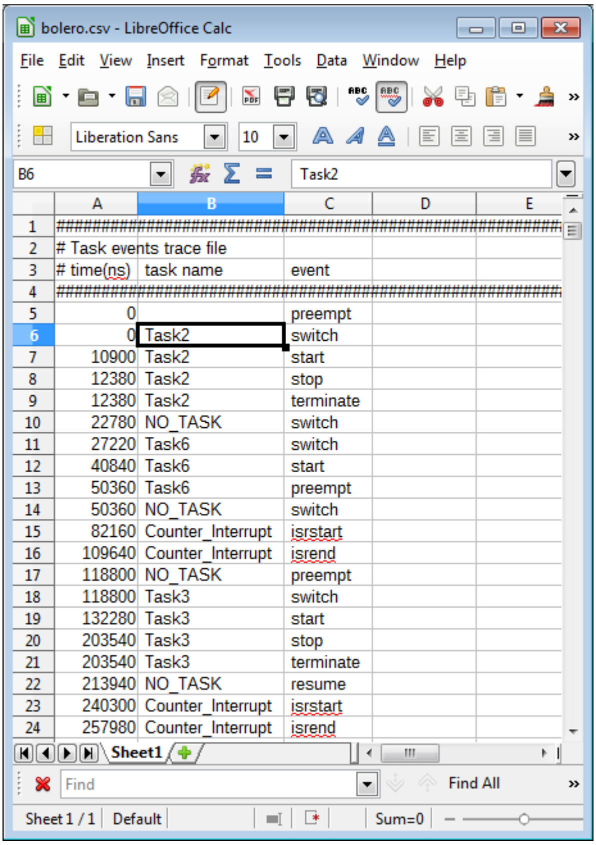
Now we're ready to export the task events. Simply use the command **Trace.EXPORT.TASKEVENTS** with the output file as parameter.

```
Trace.EXPORT.TASKEVENTS bolero.csv
```

As a result, you get a file in the CSV format (comma-separated value). This file contains state transitions of all tasks and ISRs found in the trace. You can edit the file with any application that understands this format, e.g. Notepad or any spreadsheet program:



```
#################################################
# Task events trace file
# time(ns); task name; event
#################################################
0; ; preempt
0; Task2; switch
10900; Task2; start
12380; Task2; stop
12380; Task2; terminate
22780; NO_TASK; switch
27220; Task6; switch
40840; Task6; start
50360; Task6; preempt
50360; NO_TASK; switch
82160; Counter_Interrupt; isrstart
109640; Counter_Interrupt; isrend
118800; NO_TASK; preempt
118800; Task3; switch
132280; Task3; start
203540; Task3; stop
203540; Task3; terminate
213940; NO_TASK; resume
240300; Counter_Interrupt; isrstart
257980; Counter_Interrupt; isrend
388640; Counter_Interrupt; isrstart
406340; Counter_Interrupt; isrend
```

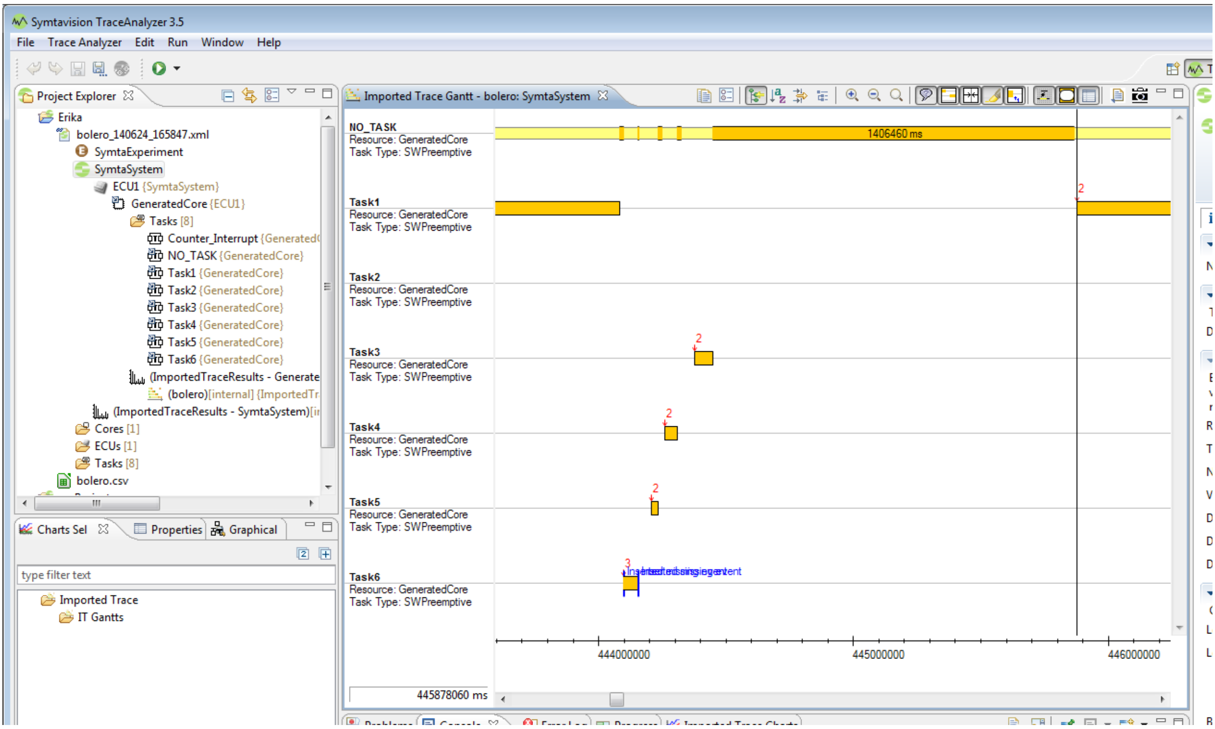

bolero.csv - LibreOffice Calc
File Edit View Insert Format Tools Data Window Help
Liberation Sans
10
B6
Task2
Sheet1
Find
Find All
Sheet 1 / 1
Default
Sum=0

	A	B	C	D	E
1	##				
2	# Task events trace file				
3	# time(ns)	task name	event		
4	##				
5	0		preempt		
6	0	Task2	switch		
7	10900	Task2	start		
8	12380	Task2	stop		
9	12380	Task2	terminate		
10	22780	NO_TASK	switch		
11	27220	Task6	switch		
12	40840	Task6	start		
13	50360	Task6	preempt		
14	50360	NO_TASK	switch		
15	82160	Counter_Interrupt	isrstart		
16	109640	Counter_Interrupt	isrend		
17	118800	NO_TASK	preempt		
18	118800	Task3	switch		
19	132280	Task3	start		
20	203540	Task3	stop		
21	203540	Task3	terminate		
22	213940	NO_TASK	resume		
23	240300	Counter_Interrupt	isrstart		
24	257980	Counter_Interrupt	isrend		

Symtavision TraceAnalyzer

In order to analyze your trace recording with Symtavision TraceAnalyzer proceed as follows:

1. Start Symtavision TraceAnalyzer (tested with 3.5.0).
2. Create new project folder (**File** --> **New** --> **Symtavision Project**).
3. Copy the CSV file exported with TRACE32 and the Symtavision Trace Converter python script into the project (drag and drop the files into the project).
4. Mark both files, right click and select **Import** from the context menu.
5. Select **Trace Import** -> **CSV Trace with Python preprocessing**"
   After processing, a new XML file is available.
6. Unfold the XML file.
7. Select **SymtaSystem**.
   Gantt View should now update automatically showing an analysis of the imported information.

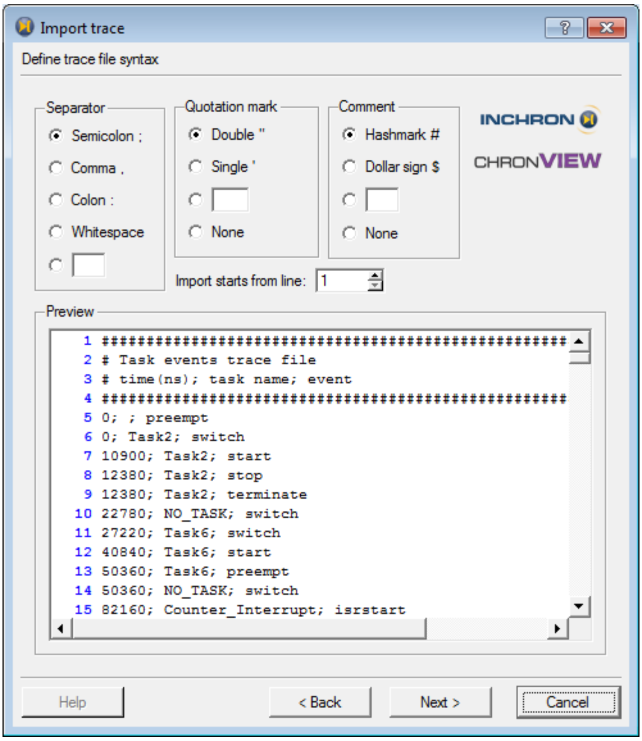


In order to analyze your trace recording with INCHRON chronVIEW proceed as follows:

1. Open INCHRON chronVIEW.

2. Import the CSV file into chronVIEW (**File --> Import CSV Trace --> bolero.csv**).

3. Define trace file syntax.



Import trace

Define trace file syntax

Separator: ● Semicolon ; ○ Comma , ○ Colon : ○ Whitespace ○

Quotation mark: ● Double " ○ Single ' ○ ○ None

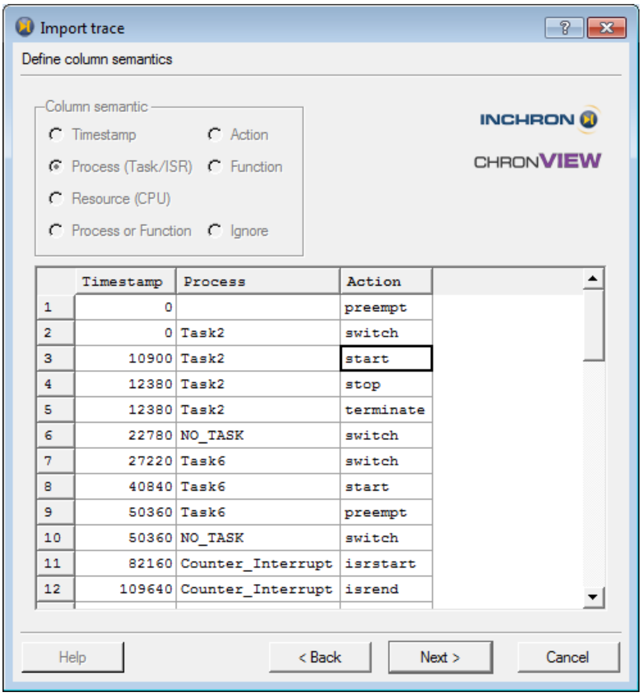
Comment: ● Hashmark # ○ Dollar sign $ ○ ○ None

Import starts from line: 1

Preview

```
########################################################
# Task events trace file
# time(ns); task name; event
########################################################
0; ; preempt
0; Task2; switch
10900; Task2; start
12380; Task2; stop
12380; Task2; terminate
22780; NO_TASK; switch
27220; Task6; switch
40840; Task6; start
50360; Task6; preempt
50360; NO_TASK; switch
82160; Counter_Interrupt; isrstart
```

4. Define column semantics.



Import trace

Define column semantics

Column semantic

○ Timestamp ○ Action

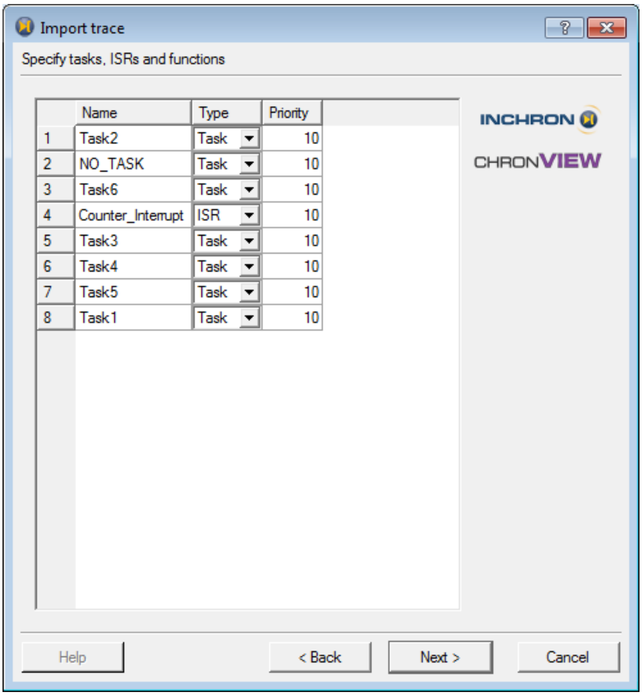
● Process (Task/ISR) ○ Function

○ Resource (CPU)

○ Process or Function ○ Ignore

	Timestamp	Process	Action
1	0		preempt
2	0	Task2	switch
3	10900	Task2	start
4	12380	Task2	stop
5	12380	Task2	terminate
6	22780	NO_TASK	switch
7	27220	Task6	switch
8	40840	Task6	start
9	50360	Task6	preempt
10	50360	NO_TASK	switch
11	82160	Counter_Interrupt	isrstart
12	109640	Counter_Interrupt	isrend

5. Specify tasks, ISRs and functions.

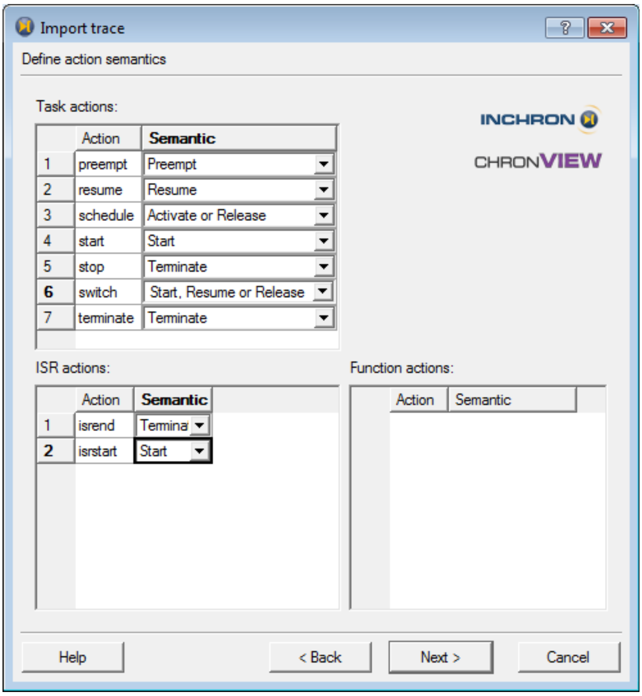


Import trace

Specify tasks, ISRs and functions

	Name	Type	Priority
1	Task2	Task	10
2	NO_TASK	Task	10
3	Task6	Task	10
4	Counter_Interrupt	ISR	10
5	Task3	Task	10
6	Task4	Task	10
7	Task5	Task	10
8	Task1	Task	10

6. Define action semantics.



Task actions:

	Action	Semantic
1	preempt	Preempt
2	resume	Resume
3	schedule	Activate or Release
4	start	Start
5	stop	Terminate
6	switch	Start, Resume or Release
7	terminate	Terminate

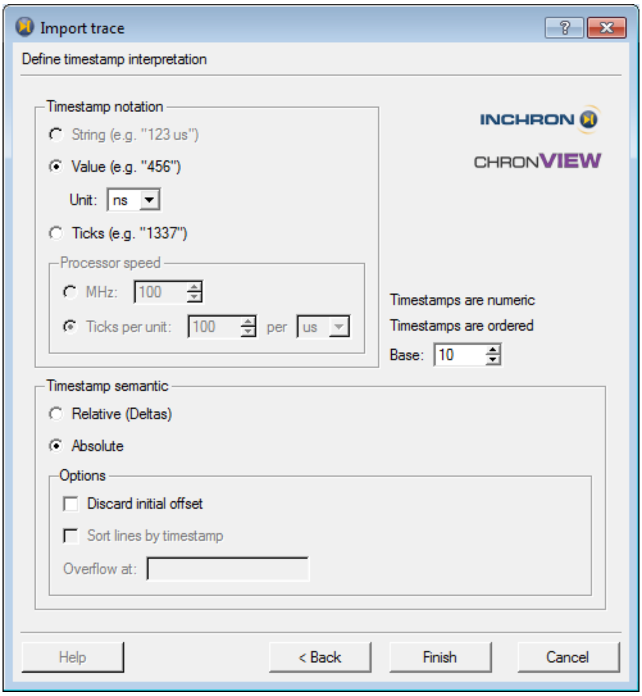
ISR actions:

	Action	Semantic
1	isrend	Termina
2	isrstart	Start

Function actions:

Action	Semantic

7. Define timestamp interpretation.



Import trace

Define timestamp interpretation

Timestamp notation

○ String (e.g. "123 us")

● Value (e.g. "456")

Unit: ns

○ Ticks (e.g. "1337")

Processor speed

○ MHz: 100

● Ticks per unit: 100 per us

Timestamps are numeric

Timestamps are ordered

Base: 10

Timestamp semantic

○ Relative (Deltas)

● Absolute

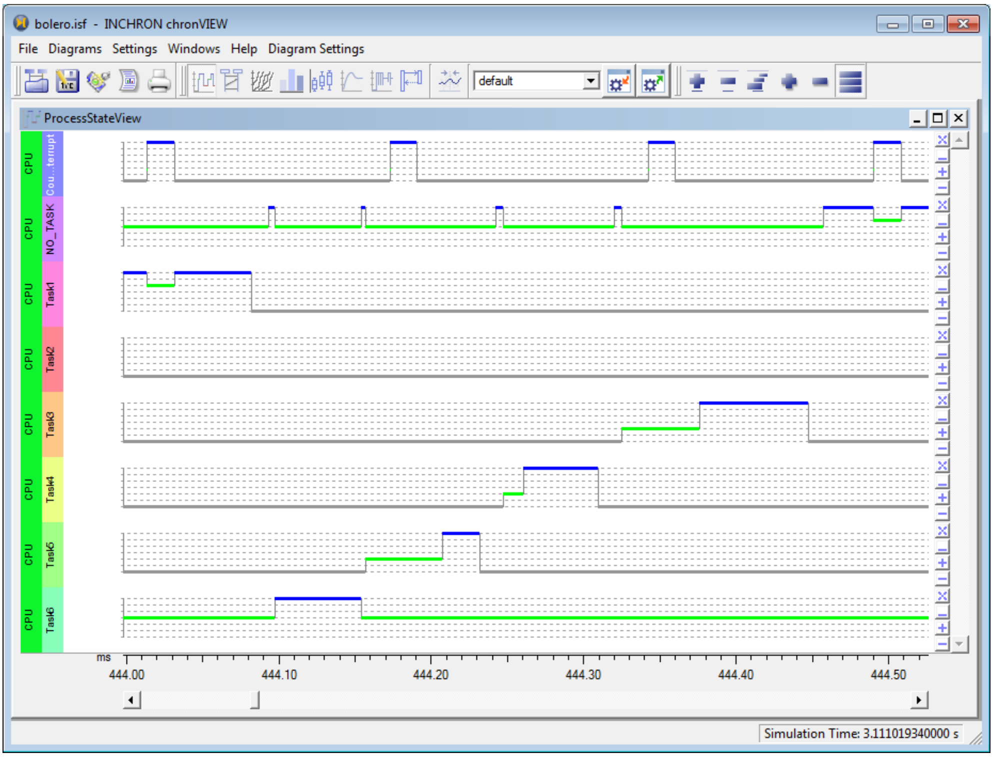
Options

□ Discard initial offset

□ Sort lines by timestamp

Overflow at:

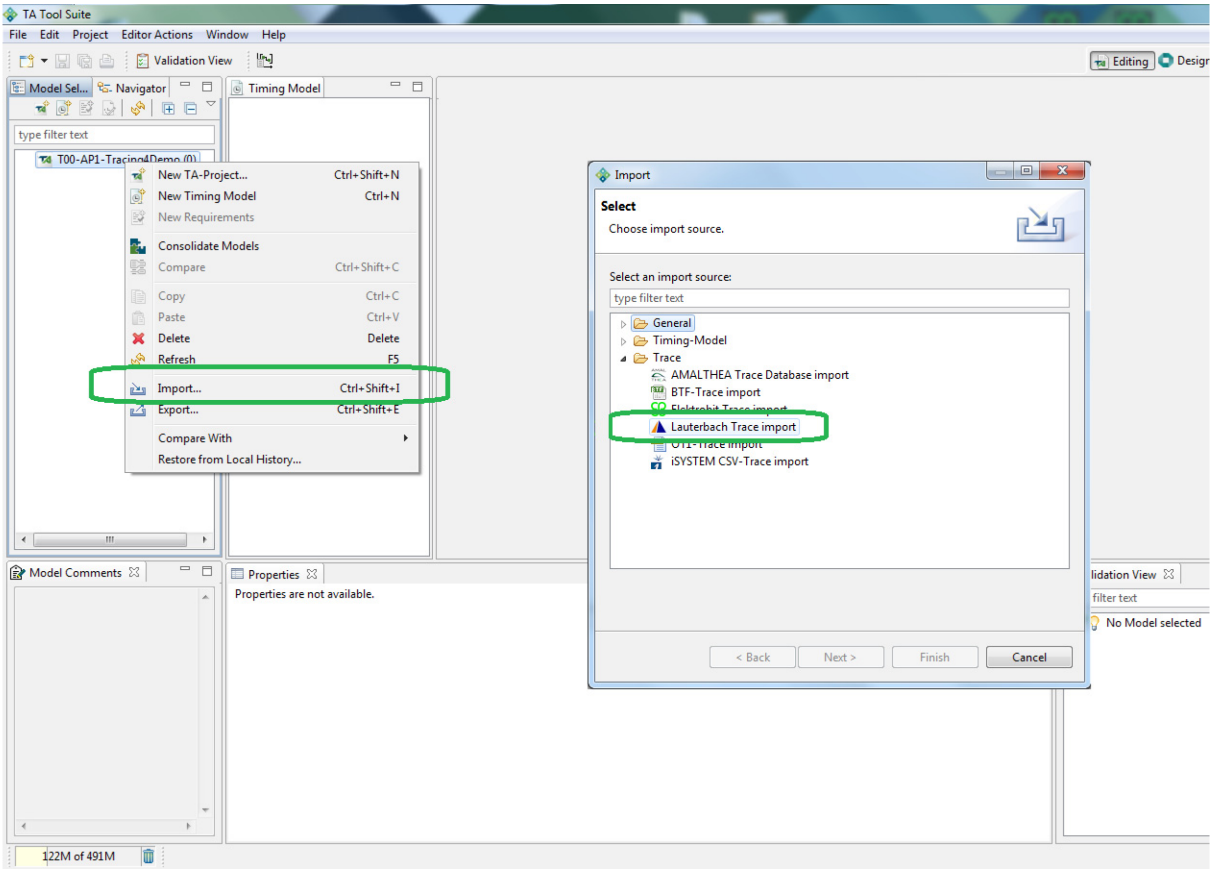
8. Press the **Finish** button to get the result.



In order to analyze your trace recording with the TA inspector proceed as follows:

1. Start the TA Tool Suite and make sure that a TA project is present inside the workspace.

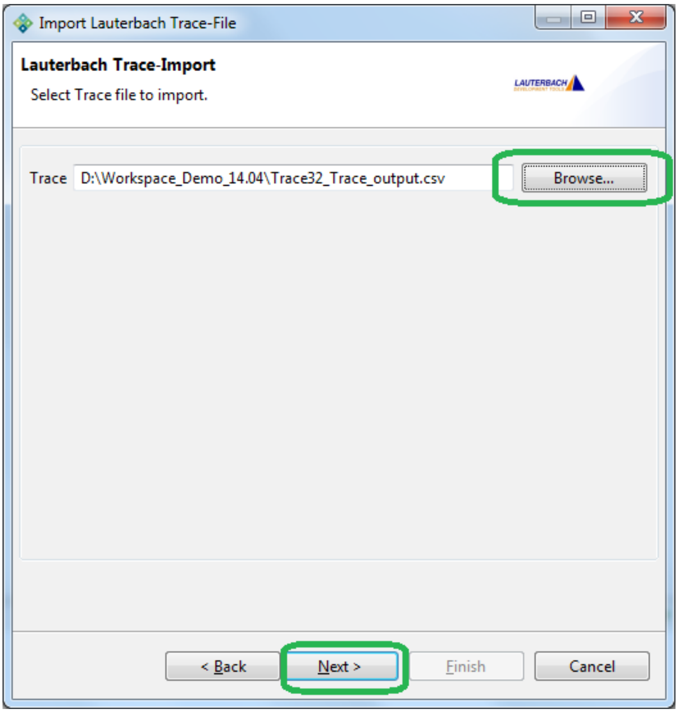
2. Right-click on the project in the **Model Selector** window.



3. Select **Import…** from the appearing context menu.

4. Inform TA that you will import a trace file exported by a Lauterbach TRACE32 tool by choosing **Lauterbach Trace import** from the Trace folder.

5. In the next step select the trace file you want to import.

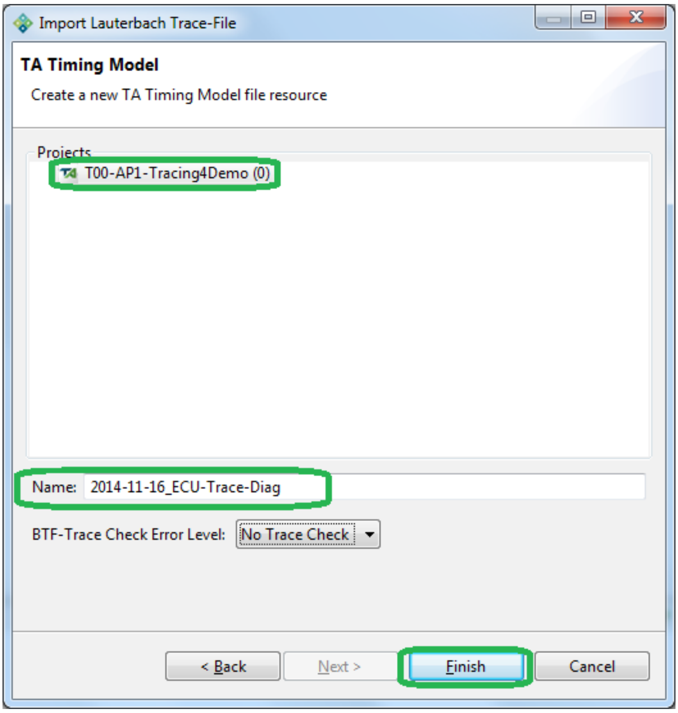

Import Lauterbach Trace-File

Lauterbach Trace-Import

Select Trace file to import.

Trace: D:\Workspace_Demo_14.04\Trace32_Trace_output.csv

6. Then select the project and specify the name for the timing model.


Import Lauterbach Trace-File

TA Timing Model

Create a new TA Timing Model file resource

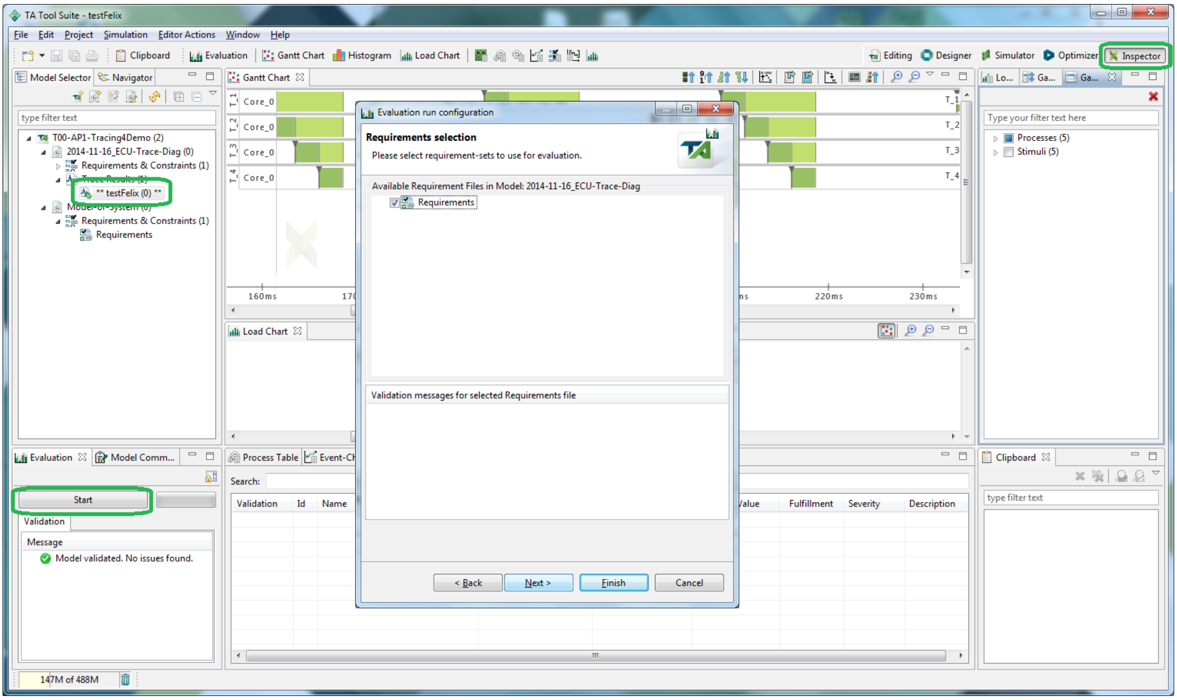
Projects: T00-AP1-Tracing4Demo (0)

Name: 2014-11-16_ECU-Trace-Diag

BTF-Trace Check Error Level: No Trace Check

7. Click finish to start the import process.

8. After the import is completed left-click the **Inspector** button (top right corner).



TA Tool Suite - testFelix
File Edit Project Simulation Editor Actions Window Help
Clipboard Evaluation Gantt Chart Histogram Load Chart
Editing Designer Simulator Optimizer Inspector
Model Selector Navigator
type filter text
T00-AP1-Tracing4Demo (2)
2014-11-16_ECU-Trace-Diag (0)
Requirements & Constraints (1)
Trace Results (1)
** testFelix (0) **
Model-of-System (0)
Requirements & Constraints (1)
Requirements
Gantt Chart
Core_0
T_1
T_2
T_3
T_4
160ms
170
220ms
230ms
Load Chart
Lo... Ga... Ga...
Type your filter text here
Processes (5)
Stimuli (5)
Evaluation Model Comm...
Start
Validation
Message
Process Table Event-Ch
Search:
Validation Id Name Value Fulfillment Severity Description
Clipboard
type filter text
147M of 488M

Evaluation run configuration

Requirements selection

Please select requirement-sets to use for evaluation.

Available Requirement Files in Model: 2014-11-16_ECU-Trace-Diag

☑ Requirements

Validation messages for selected Requirements file

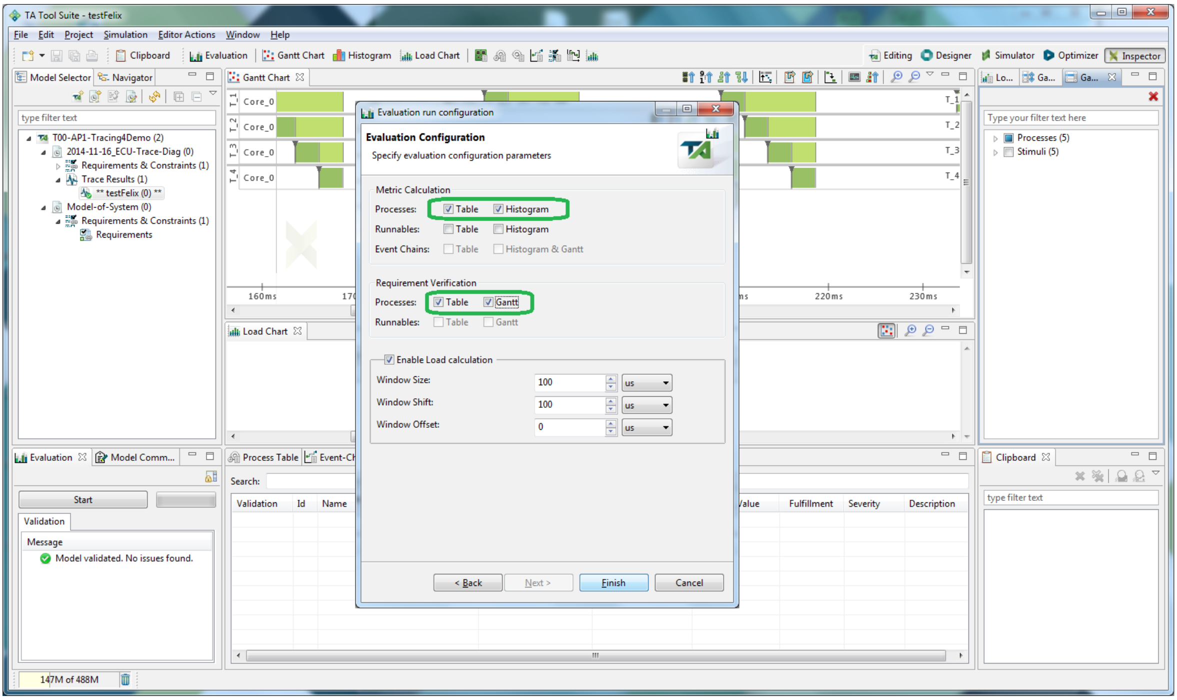
< Back | Next > | Finish | Cancel

Model validated. No issues found.

9. Select a trace file and start the calculation.

10. The calculation needs a **requirement-set** for your timing model.

11. Additionally the **evaluation configuration parameters** need to be specified.



Evaluation run configuration

Evaluation Configuration

Specify evaluation configuration parameters

Metric Calculation

- Processes: ☑ Table ☑ Histogram
- Runnables: □ Table □ Histogram
- Event Chains: □ Table □ Histogram & Gantt

Requirement Verification

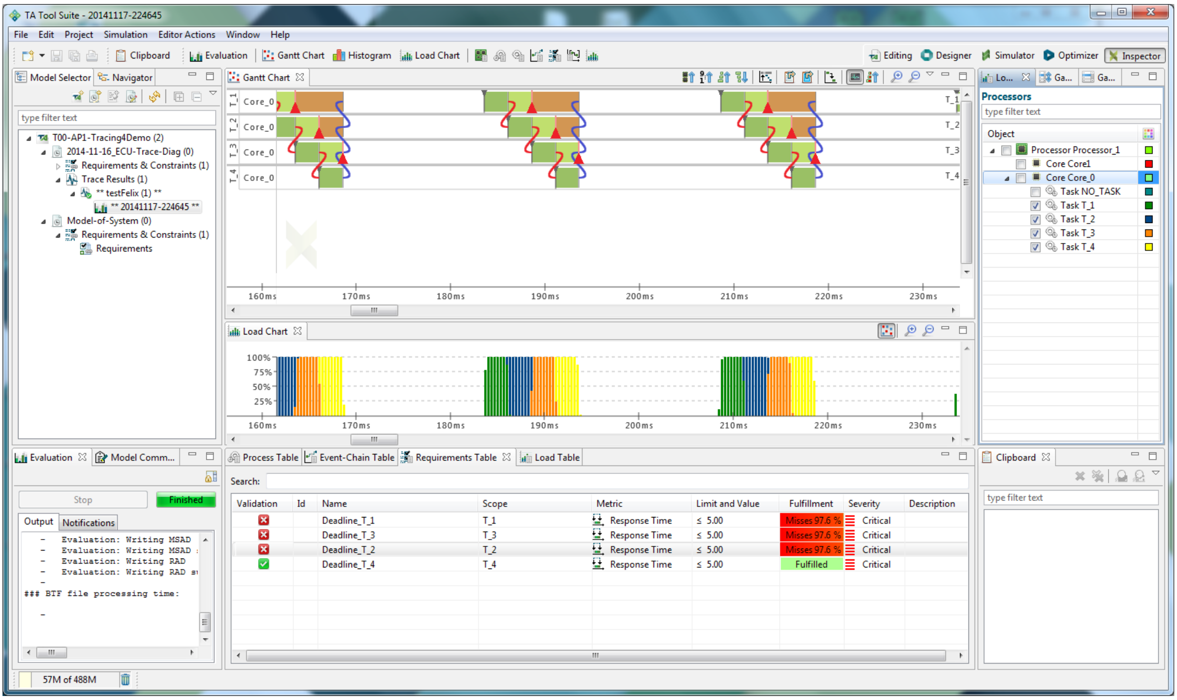
- Processes: ☑ Table ☑ Gantt
- Runnables: □ Table □ Gantt

☑ Enable Load calculation

Window Size:	100	us
Window Shift:	100	us
Window Offset:	0	us

Validation: Model validated. No issues found.

12. When the calculation is done, the results are displayed.



```
   -   Evaluation: Writing MSAD
   -   Evaluation: Writing MSAD :
   -   Evaluation: Writing RAD
   -   Evaluation: Writing RAD s
   -
### BTF file processing time:

   -
```

Requirements Table

Validation	Id	Name	Scope	Metric	Limit and Value	Fulfillment	Severity	Description
☒		Deadline_T_1	T_1	Response Time	≤ 5.00	Misses 97.6 %	Critical	
☒		Deadline_T_3	T_3	Response Time	≤ 5.00	Misses 97.6 %	Critical	
☒		Deadline_T_2	T_2	Response Time	≤ 5.00	Misses 97.6 %	Critical	
✓		Deadline_T_4	T_4	Response Time	≤ 5.00	Fulfilled	Critical	