

# Trace Export for Third-Party Timing Tools

MANUAL

# Trace Export for Third-Party Timing Tools

---

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

[TRACE32 Documents](#) .....



[OS Awareness Manuals](#) .....



[OS Awareness for OSEK/ORTI](#) .....



[Application Note for OSEK/ORTI](#) .....



**Trace Export for Third-Party Timing Tools** .....

1

**Introduction** .....

3

**Requirements** .....

5

**Processing** .....

6

**Example** .....

8

    Related Documents

8

    Environment

8

    Step 1: Create an AUTOSAR/OSEK Application

9

    Step 2: Set up TRACE32 and Run the Application

10

    Step 3: Set up Real-time Trace within TRACE32

12

        TRACE32 Instruction Set Simulator

12

        NEXUS Class 3

13

        NEXUS Class 2

14

    Step 4: Run the Program Execution to Fill the Trace

15

    Step 5: Set up Markers for Trace Export

18

    Step 6: Export Task Events

19

**Timing Tools** .....

20

    Syntavision TraceAnalyzer

20

    INCHRON chronVIEW

21

    Timing Architects - Inspector

25

## Introduction

---

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

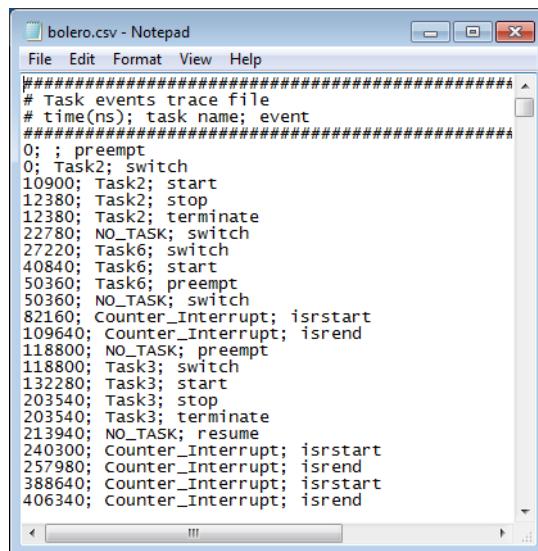
- **Symtavision:** TraceAnalyzer
- **INCHRON:** chronVIEW
- **Timing Architects:** Inspector

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

**Trace.EXPORT.TASKEVENTS <file>**

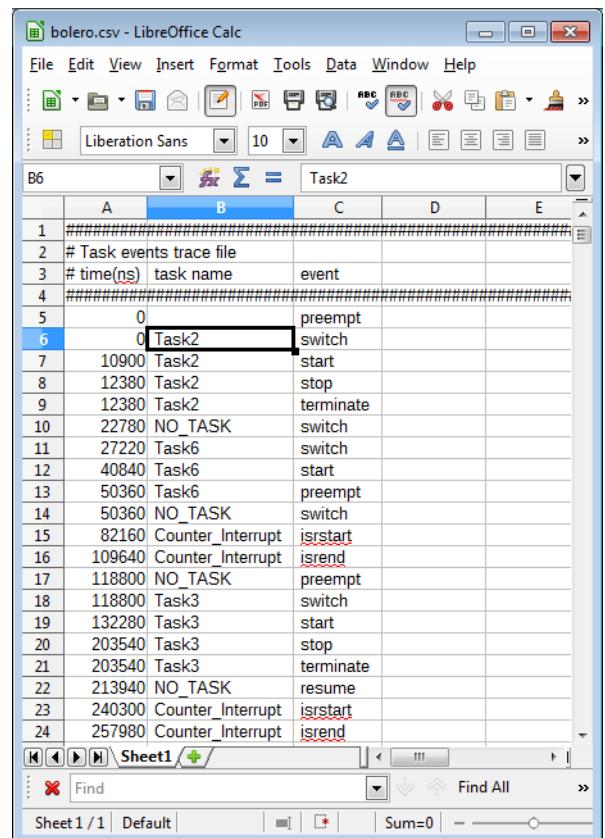
All timing tools listed above are used in the automotive industry, so we limit ourselves in this document to AUTOSAR/OSEK operating systems. But the topic can be, of course, applied to other operating systems too.

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.



bolero.csv - Notepad

```
File Edit Format View Help
#####
# 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

	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	213940	NO_TASK	terminate		
22	240300	Counter_Interrupt	isrstart		
23	257980	Counter_Interrupt	isrend		
24					

# 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).

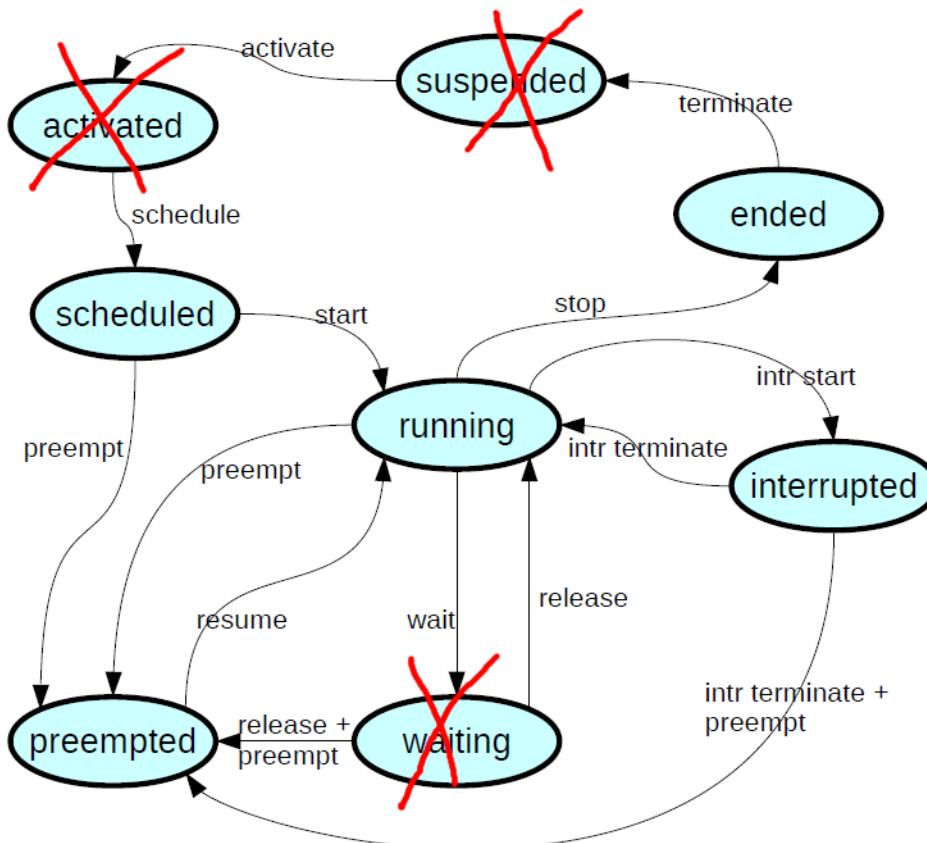
## **sYmbol.MARKER.Create ISREND <address>**

- 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>**

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:

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

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

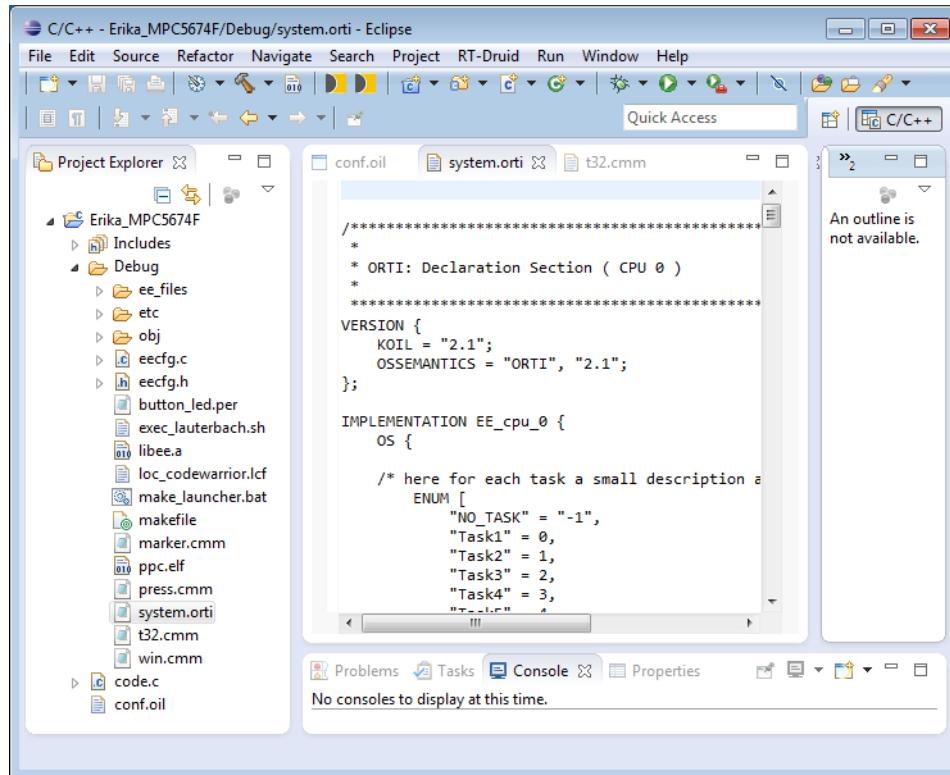
The development environment is available free of charge at <http://erika.tuxfamily.org>.

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).



## 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.

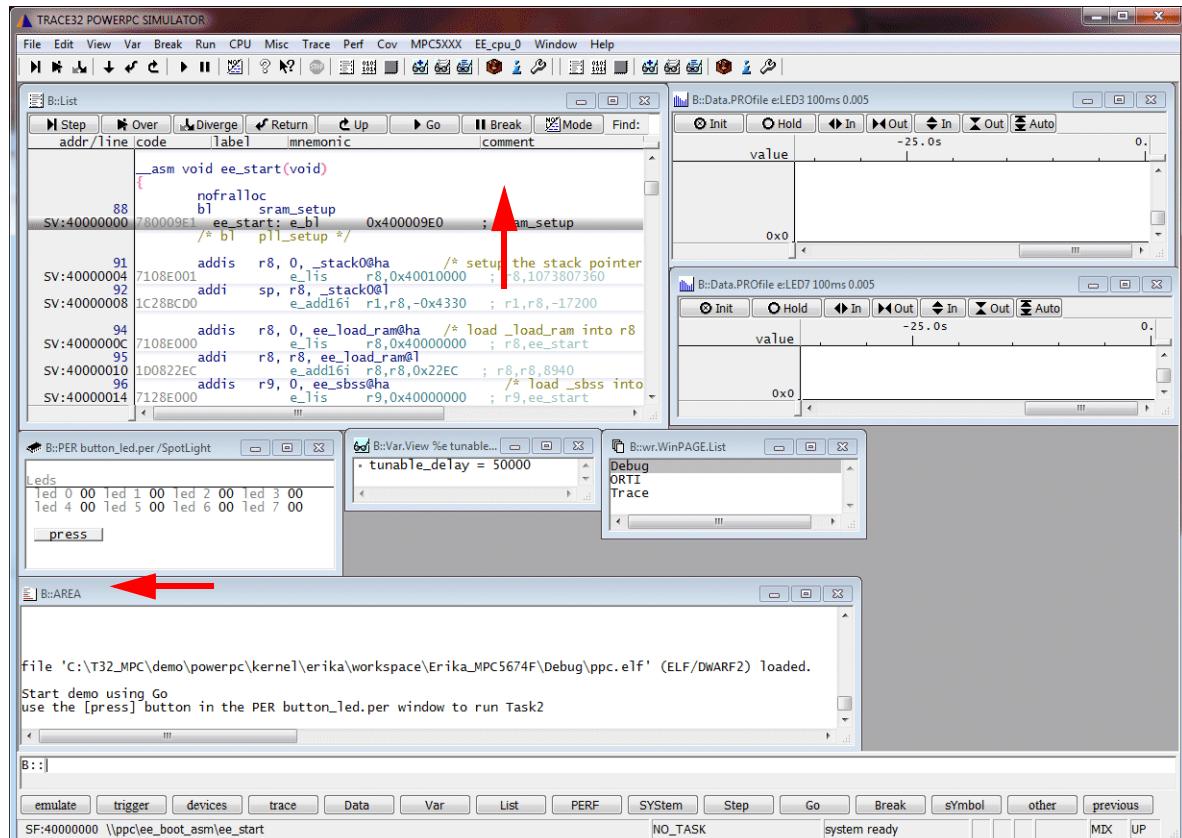
**Data.LOAD.ELF <file>**

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.



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.

TRACE32 POWERPC SIMULATOR

File Edit View Var Break Run CPU Misc Trace Perf Cov EE\_cpu\_0 Window Help

B:TASK.DOS

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

B:TASK.DRESOURCE

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

B:TASK.DALARM

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

B:TASK.STackK.view

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

B:TASK. |

DOS DTASK DSTACK DALARM DRESOURCE previous

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

## Step 3: Set up Real-time Trace within TRACE32

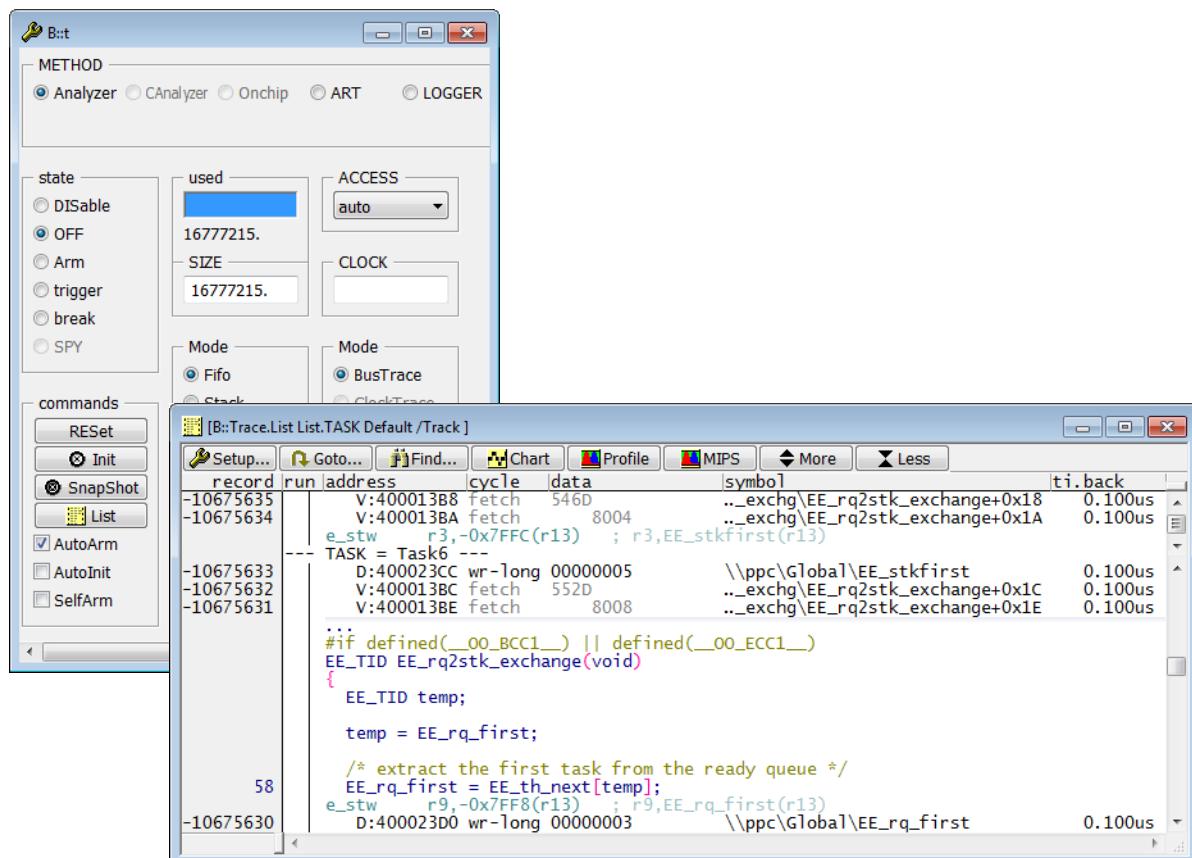
In order to provide all information for a detailed task analysis, the trace logic on the target has to be configured to provide the complete instruction execution sequence plus all task switches.

### 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 in our example script).

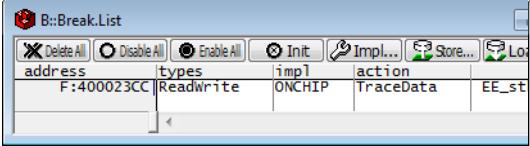
```
Trace.SIZE 16777215.
```

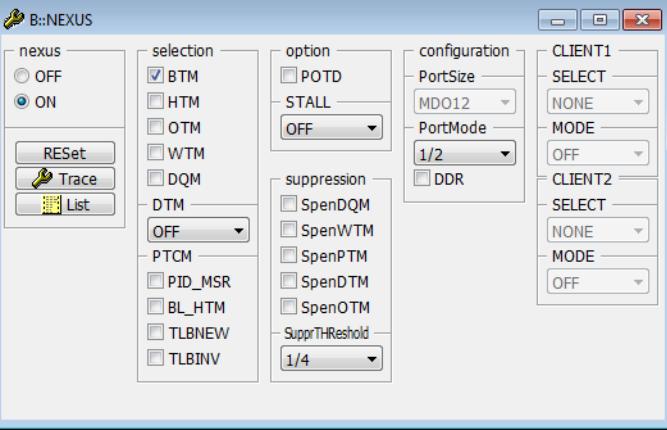


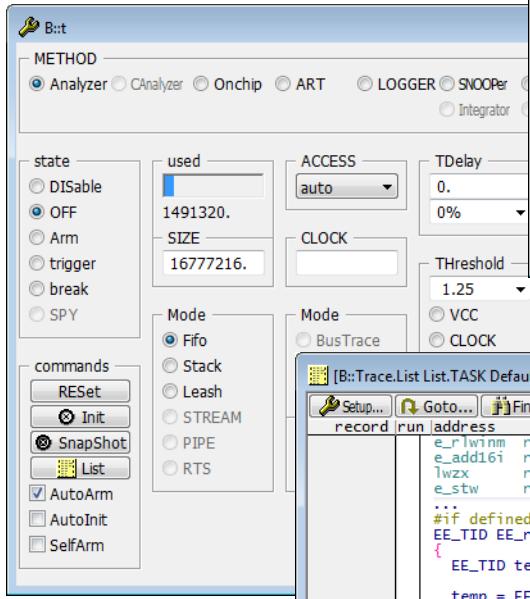
If your chip provides a NEXUS Class 3+ module, this NEXUS module has to be configured to generate trace information for the instruction execution sequence and the task switches.

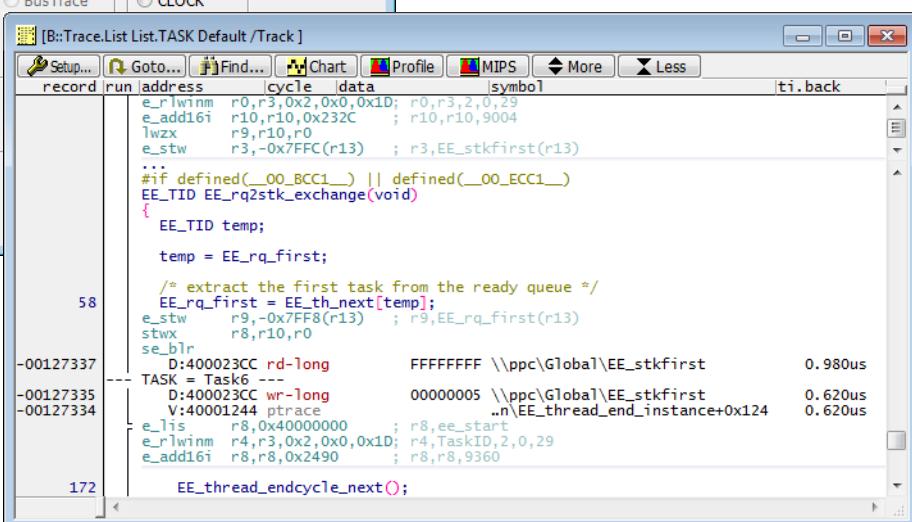
For details refer to “[OS-Aware Tracing \(ORTI File\)](#)” in Nexus Training, page 187 (training\_nexus.pdf).

NEXUS.BTM ON  
 Break.Set TASK.CONFIG(magic) /TraceData









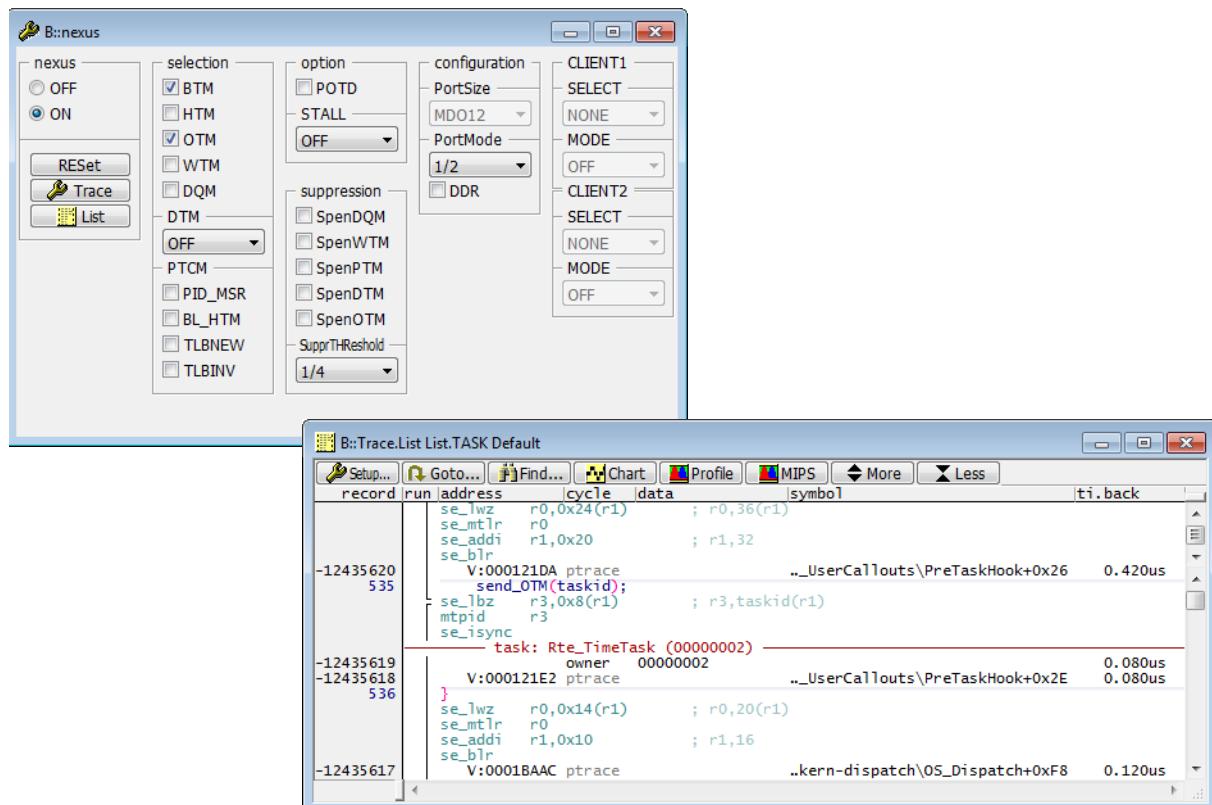
If your chip provides a NEXUS Class 2+ module, this NEXUS module has to be configured to generate trace information for the instruction execution sequence and the task switches.

```
NEXUS.BTM ON
```

```
NEXUS.OTM ON
```

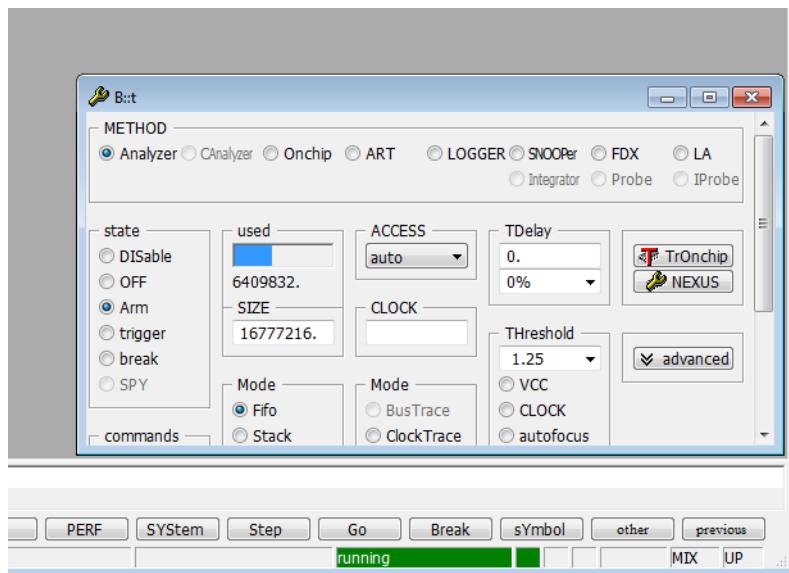
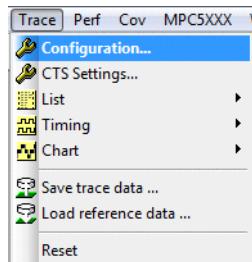
For details refer to “[OS-Aware Tracing \(ORTI File\)](#)” in Nexus Training, page 187 (training\_nexus.pdf).

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



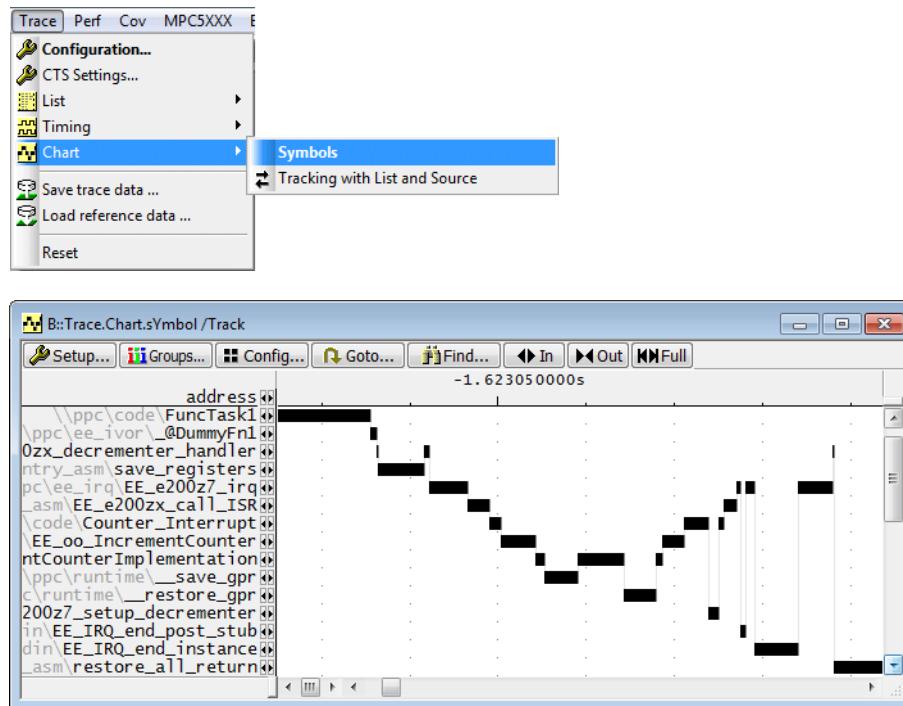
## Step 4: Run the Program Execution to Fill the Trace

Display a Trace Configuration window ([Trace.state](#)) and start the program execution.



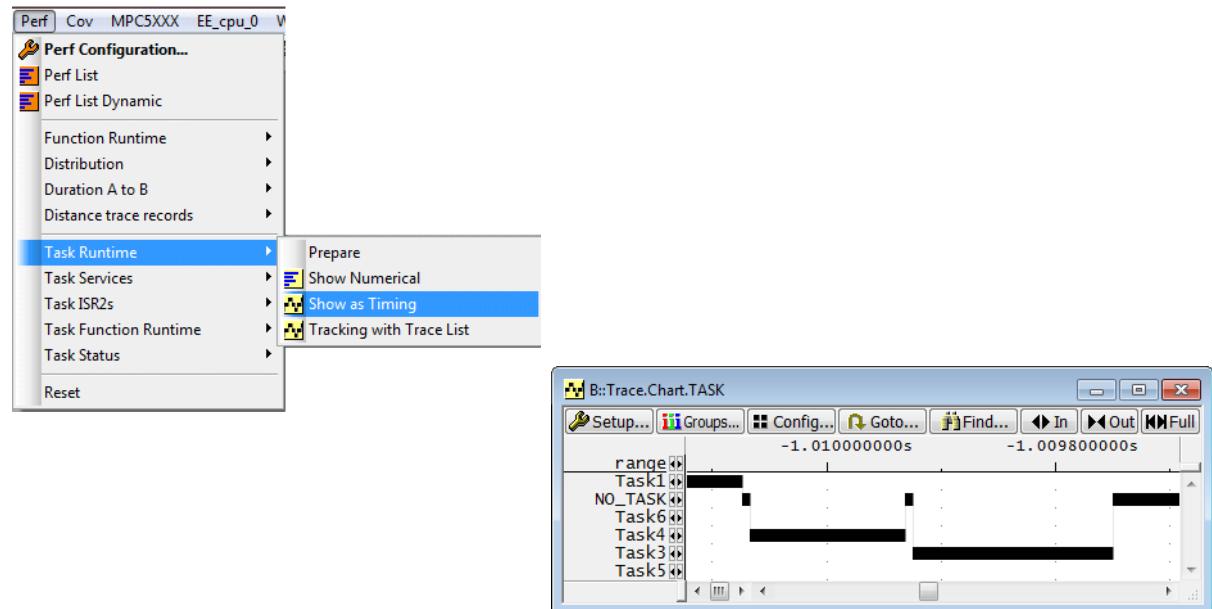
Stop the program execution by pushing the [Break] button.

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

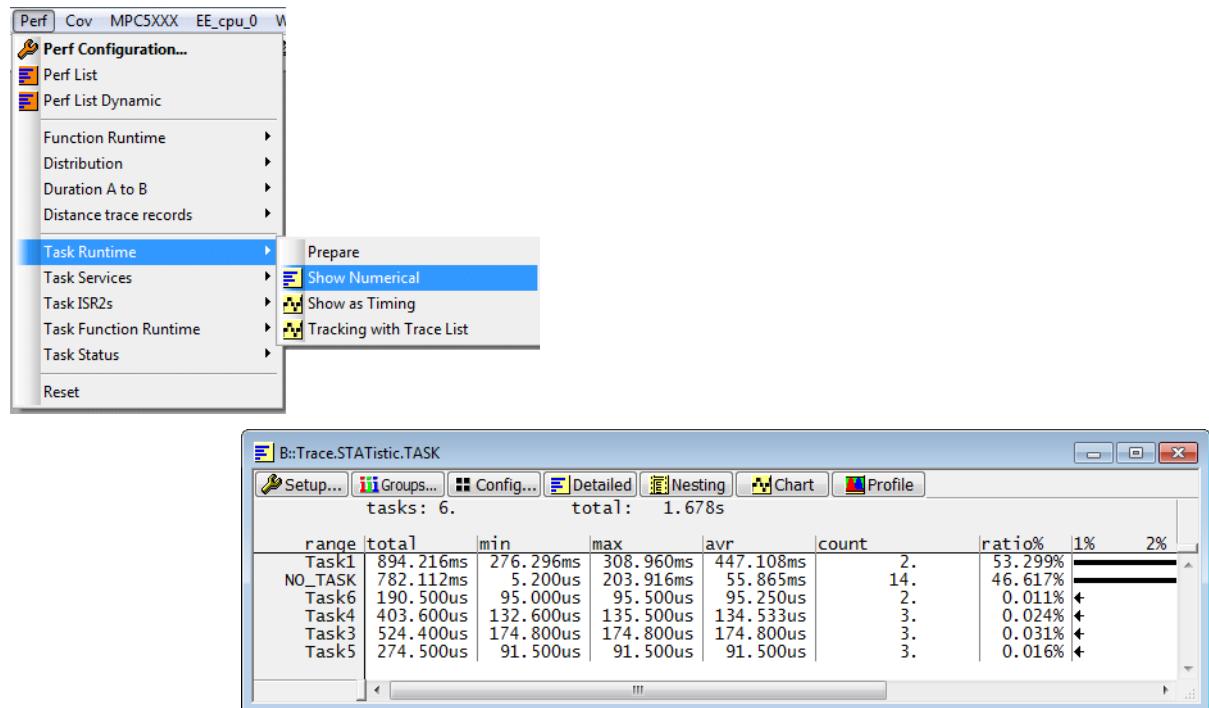


Details on possible errors and their causes can be found in “**FlowErrors**” in Nexus Training, page 48 (training\_nexus.pdf) and “**FIFOFULL**” in Nexus Training, page 44 (training\_nexus.pdf). Please be aware that an error-free trace is required in order to export task event information.

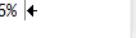
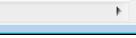
Use the **Trace.Chart.TASK** command to inspect the task switches.



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



The screenshot shows the Lauterbach Trace Export interface. On the left, a menu tree is open under the 'Perf' tab, with 'Task Runtime' selected. A sub-menu 'Prepare' is open, showing three options: 'Show Numerical' (selected), 'Show as Timing', and 'Tracking with Trace List'. The main window displays a table of task execution times. The table has the following data:

	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.

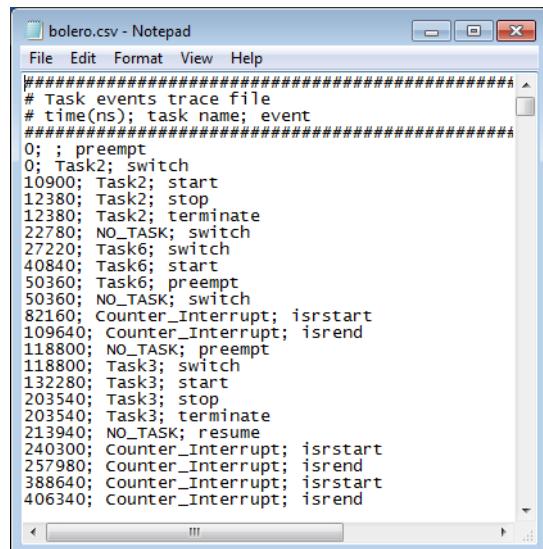
```
sYmbol.MARKER.Create TASKSTART FuncTask1
sYmbol.MARKER.Create TASKSTART FuncTask2
sYmbol.MARKER.Create TASKSTART FuncTask3
sYmbol.MARKER.Create TASKSTART FuncTask4
sYmbol.MARKER.Create TASKSTART FuncTask5
sYmbol.MARKER.Create TASKSTART FuncTask6
sYmbol.MARKER.Create TASKSTART FuncTask7
sYmbol.MARKER.Create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

## Step 6: Export Task Events

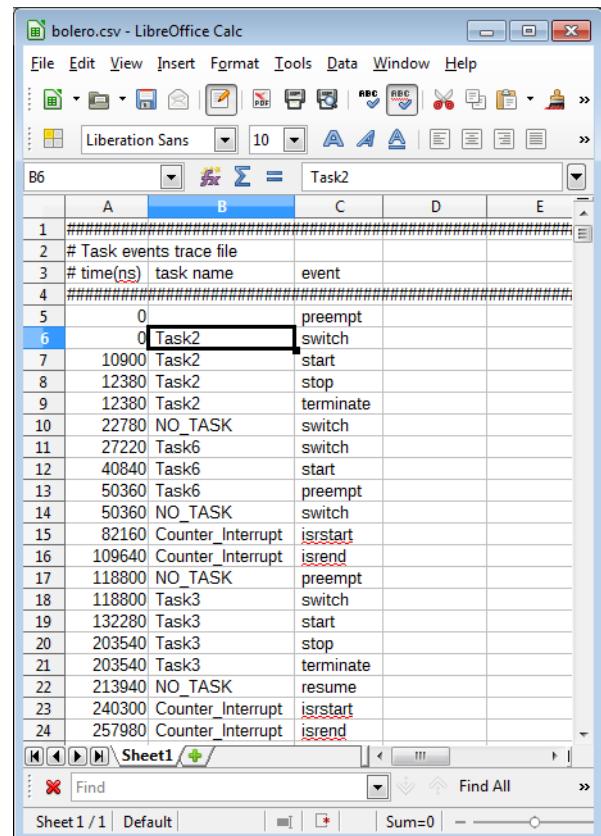
Now we're ready to export the task events. Simply use the command **Trace.EXPORT.TASKEVENTS** with the output file as parameter.

```
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:



```
bolero.csv - Notepad
File Edit Format View Help
#####
# 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
```



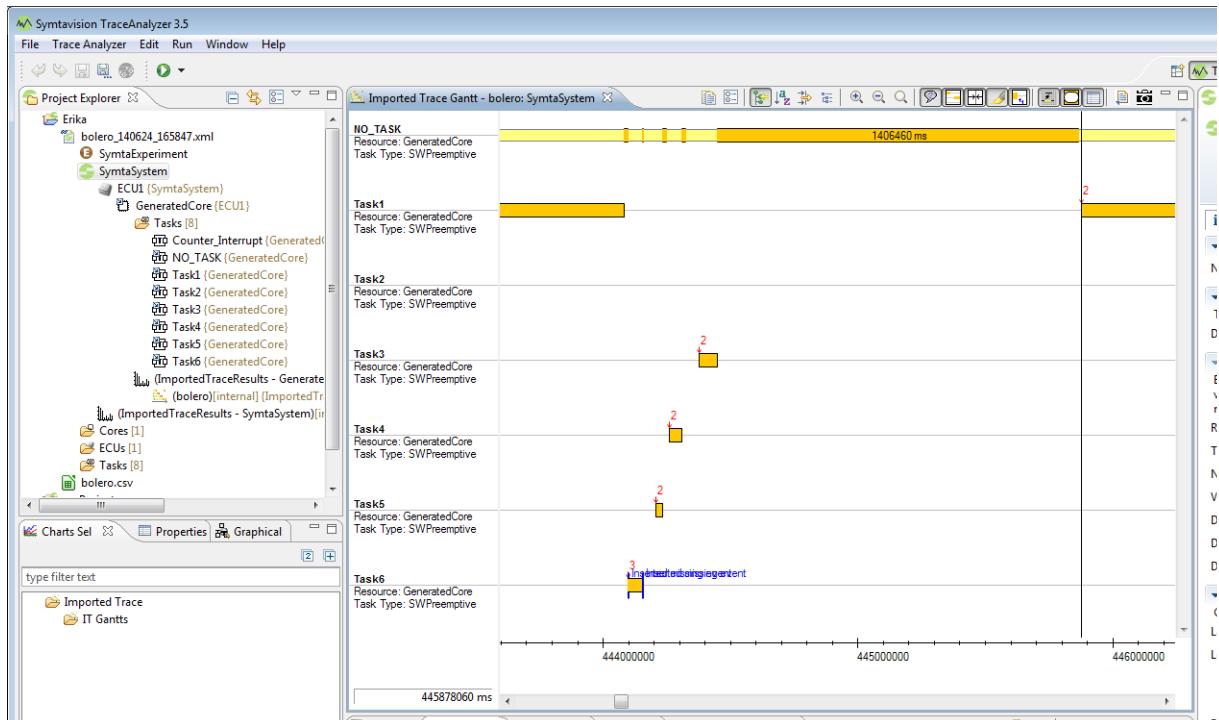
	A	B	C	D	E
1	#####				
2	# Task events trace file				
3	# time(ns)	task name	event		
4	#####				
5	0	preempt			
6	0	Task2	switch		
7	10900	Task2	start		
8	12380	Task2	stop		
9	12380	Task2	terminate		
10	22780	NO_TASK	switch		
11	27220	Task6	switch		
12	40840	Task6	start		
13	50360	Task6	preempt		
14	50360	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:

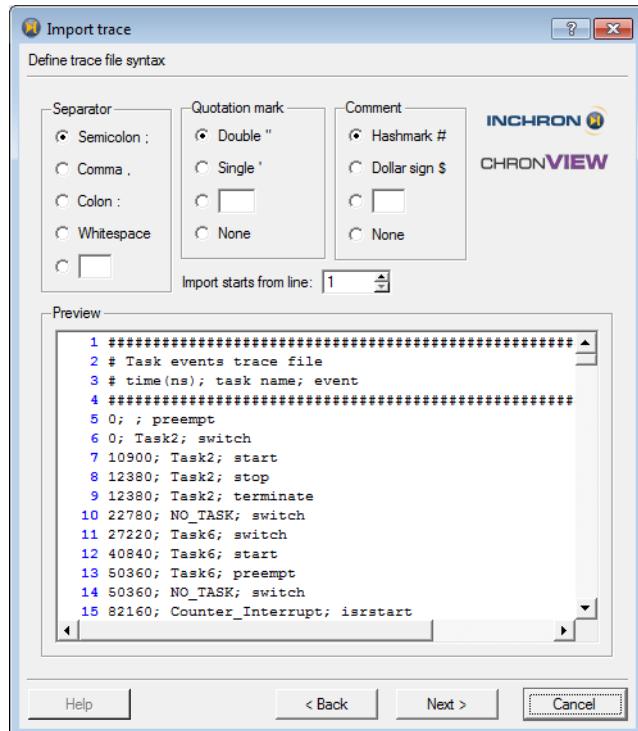
1. Start Symtavision TraceAnalyzer (tested with 3.5.0).
2. Create new project folder (**File --> New --> Symtavision Project**).
3. Copy the CSV file exported with TRACE32 and the Symtavision Trace Converter python script into the project (drag and drop the files into the project).
4. Mark both files, 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.

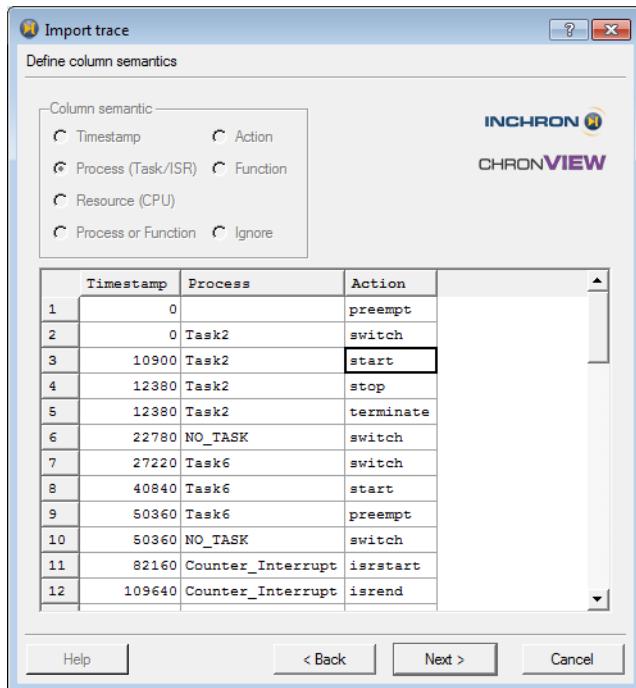


In order to analyze your trace recording with INCHRON chronVIEW proceed as follows:

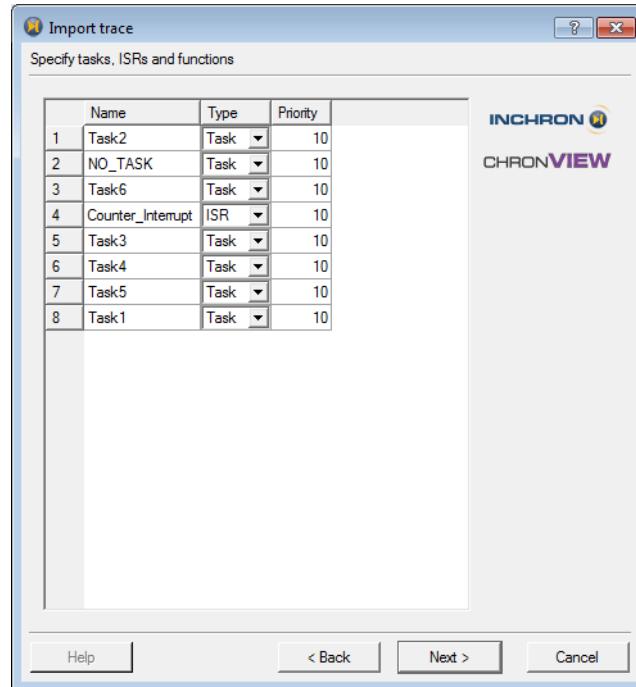
1. Open INCHRON chronVIEW.
2. Import the CSV file into chronVIEW (**File** → **Import CSV Trace** → **bolero.csv**).
3. Define trace file syntax.



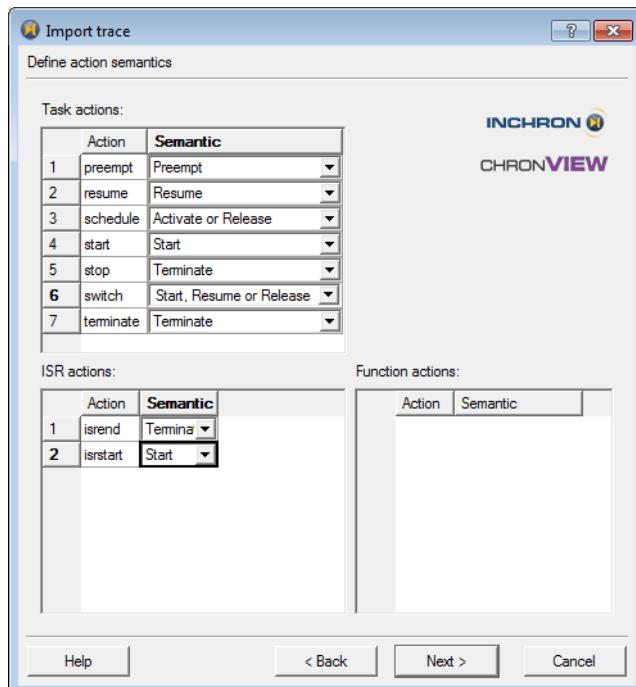
#### 4. Define column semantics.



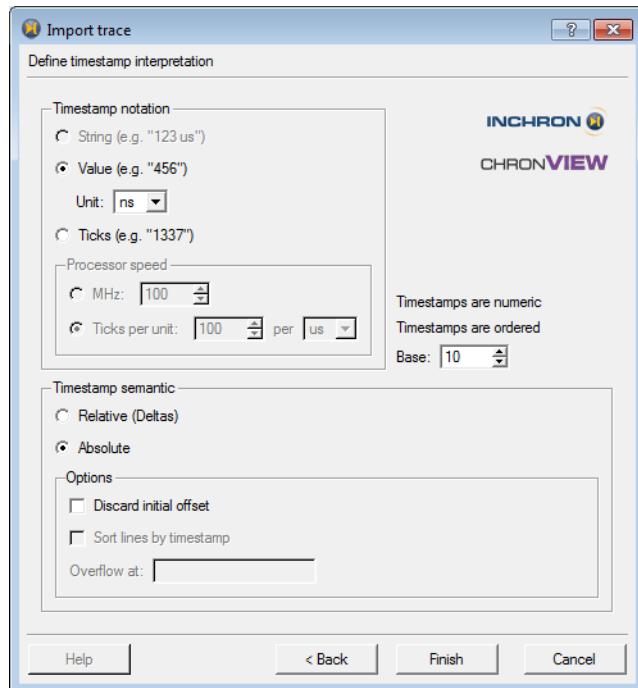
#### 5. Specify tasks, ISRs and functions.



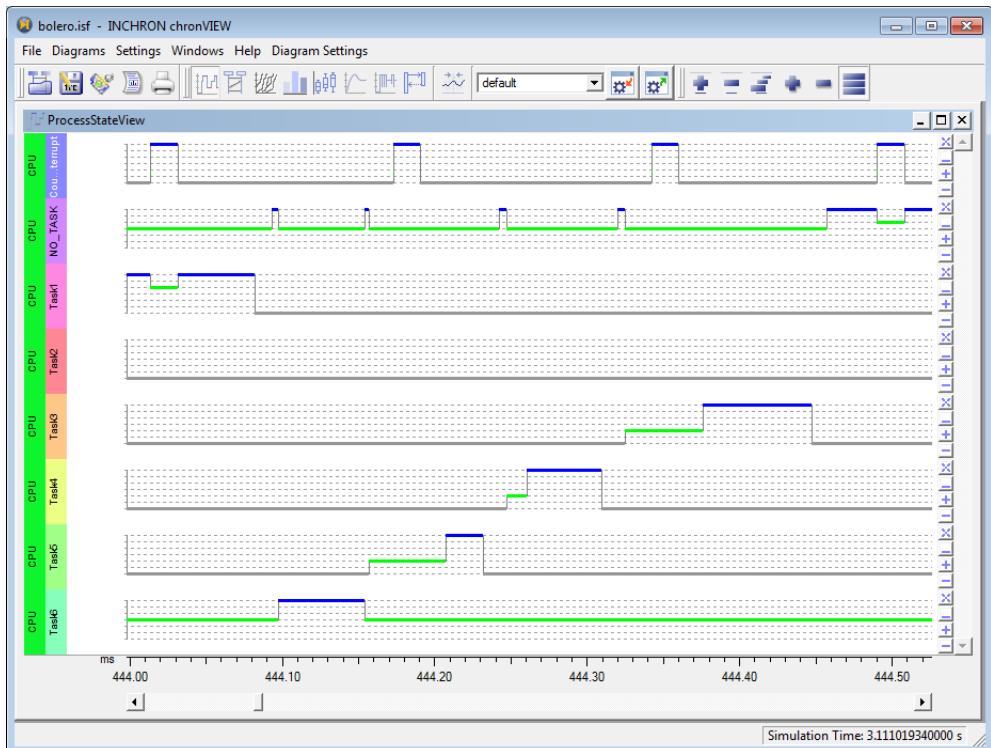
## 6. Define action semantics.



7. Define timestamp interpretation.

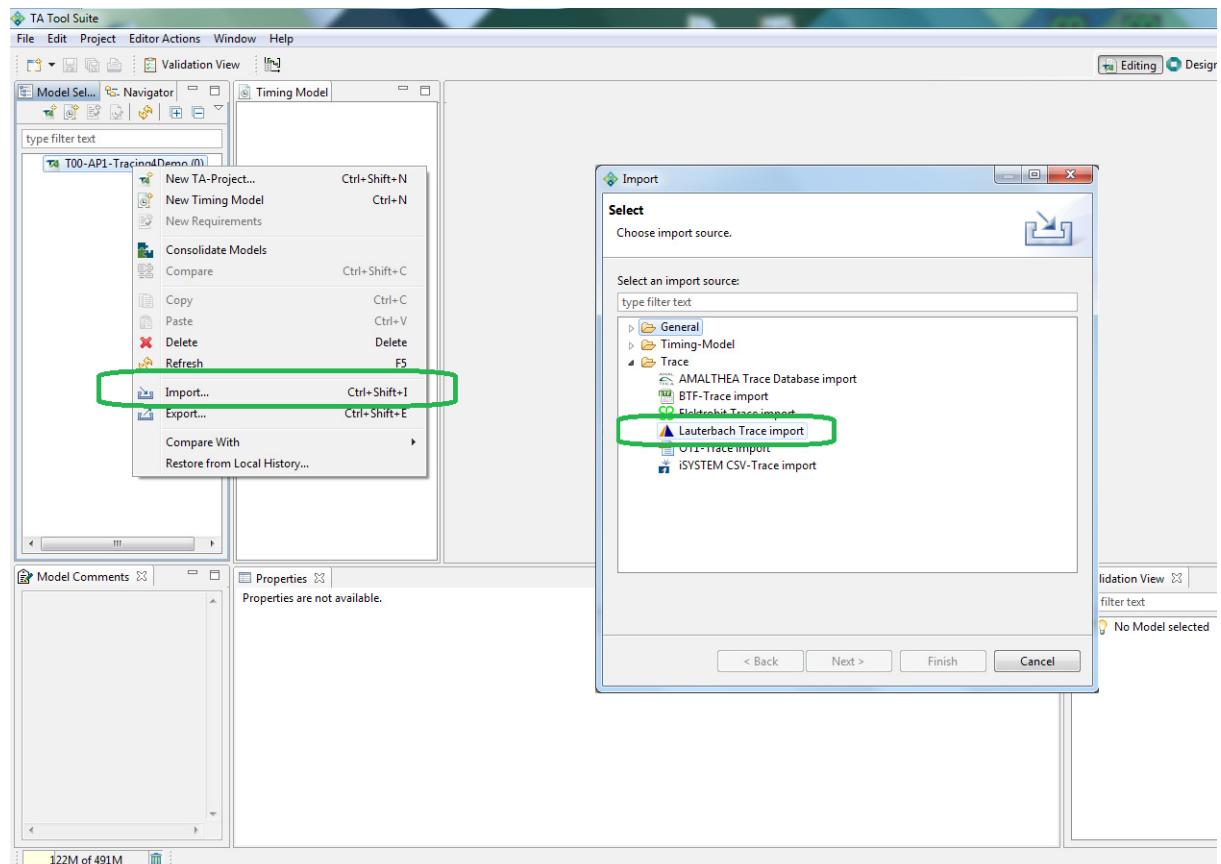


8. Press the **Finish** button to get the result.



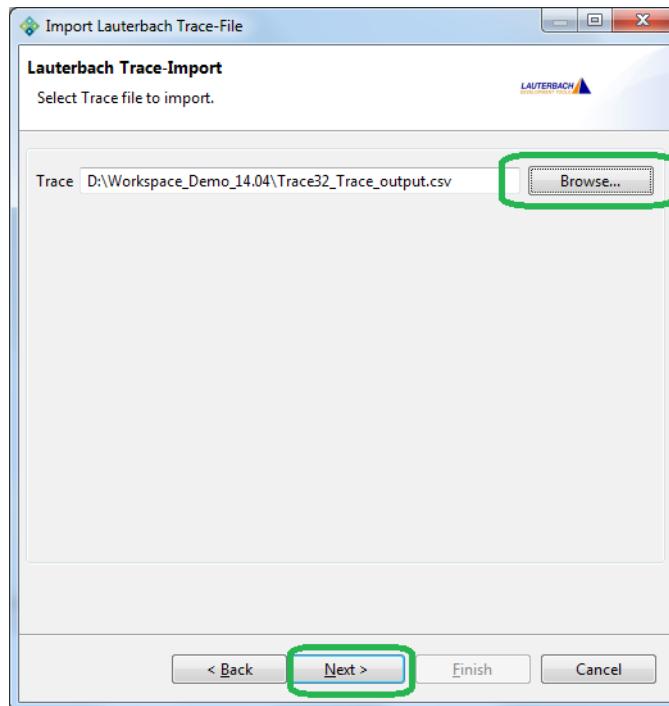
In order to analyze your trace recording with the TA inspector proceed as follows:

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.

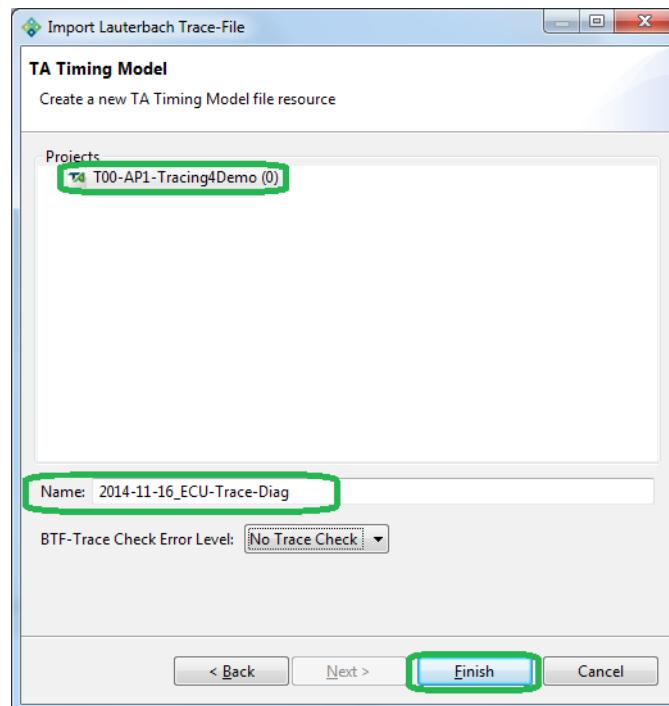


3. Select **Import...** from the appearing context menu.
4. Inform TA that you will import a trace file exported by a Lauterbach TRACE32 tool by choosing **Lauterbach Trace import** from the Trace folder.

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

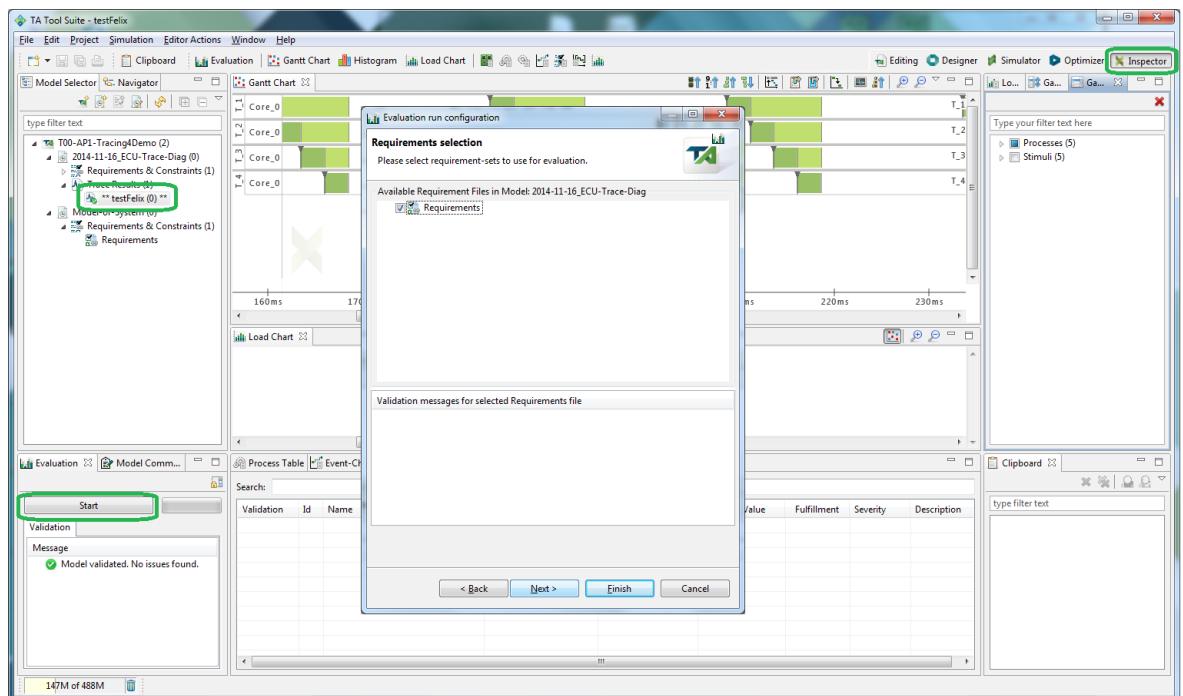


6. Then select the project and specify the name for the timing model.



7. Click finish to start the import process.

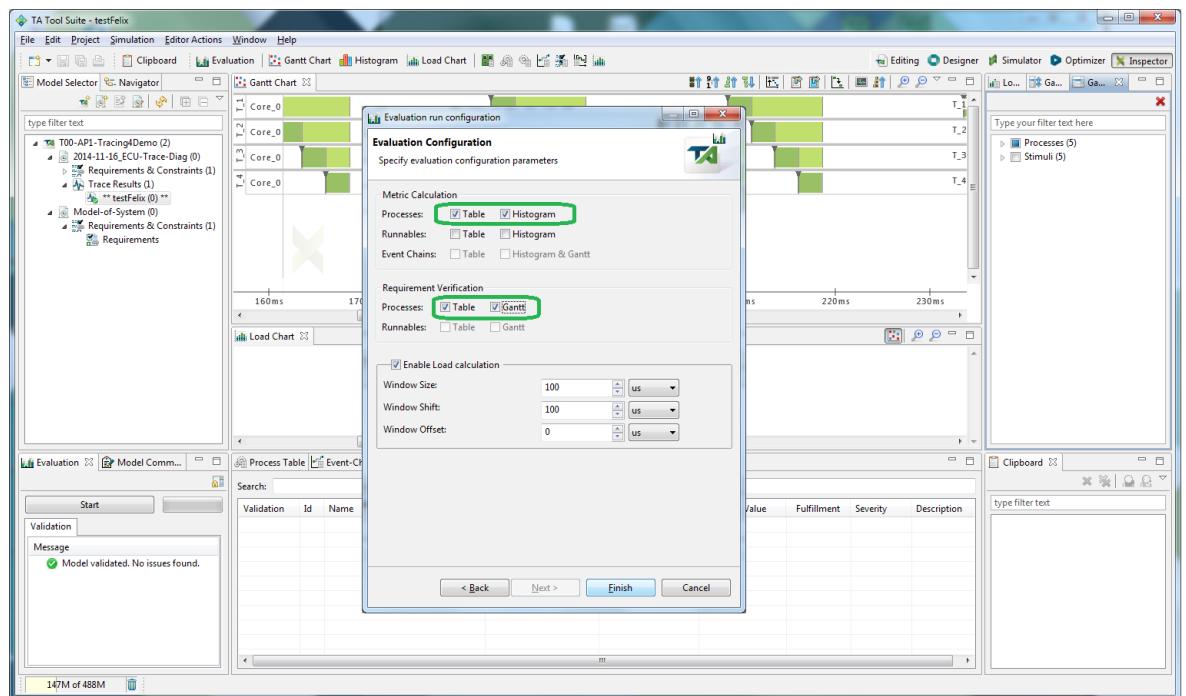
8. After the import is completed left-click the **Inspector** button (top right corner).



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.



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

