Trace Export for Third-Party Timing Tools

Trace Export for Third-Party Timing Tools

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

Trace Export for Third-Party Timing Tools

Version 06-Jun-2024

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.

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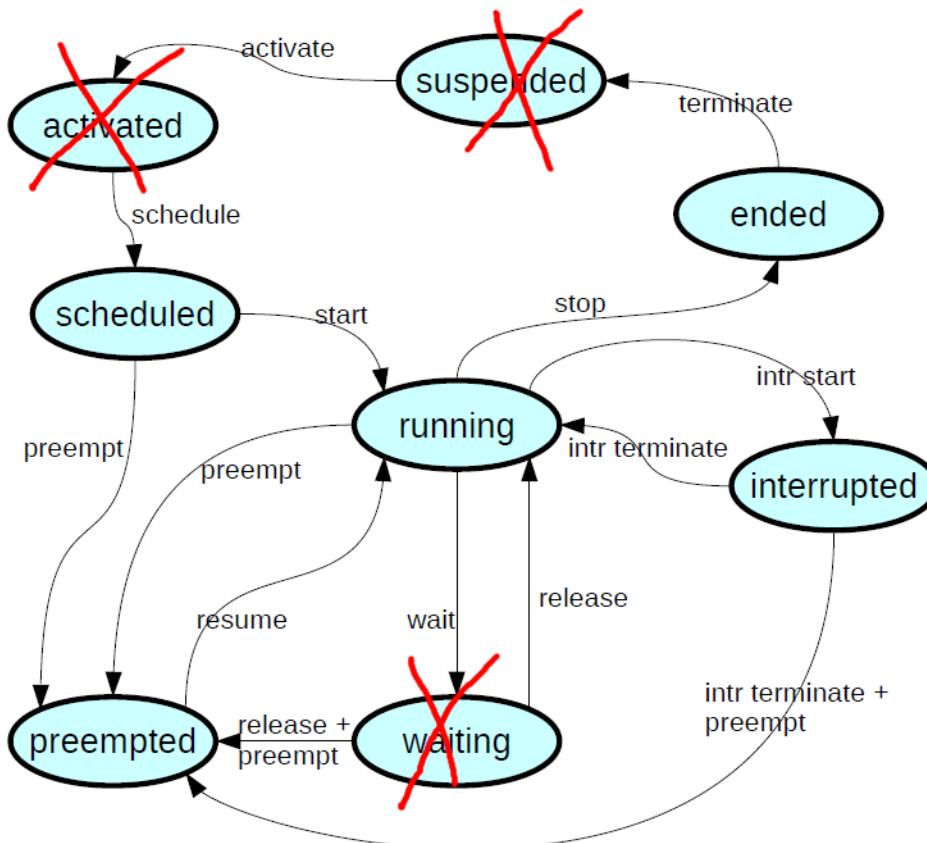
The **events** (state transitions) in the CSV file have the following meanings:

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).
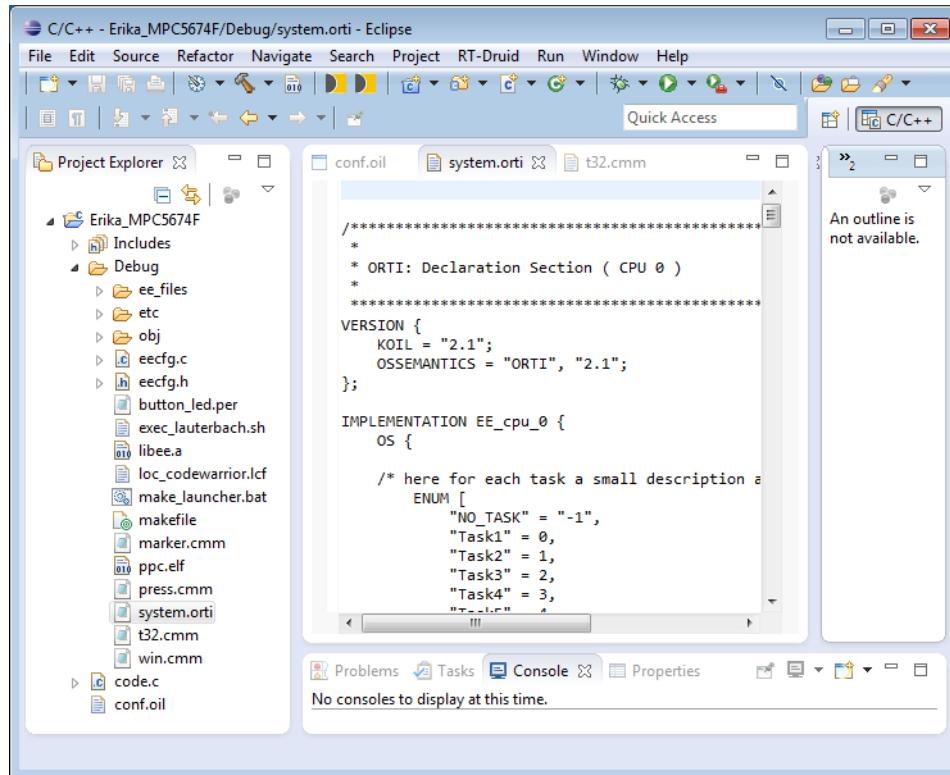
Step 2: Set up TRACE32 and Run the Application

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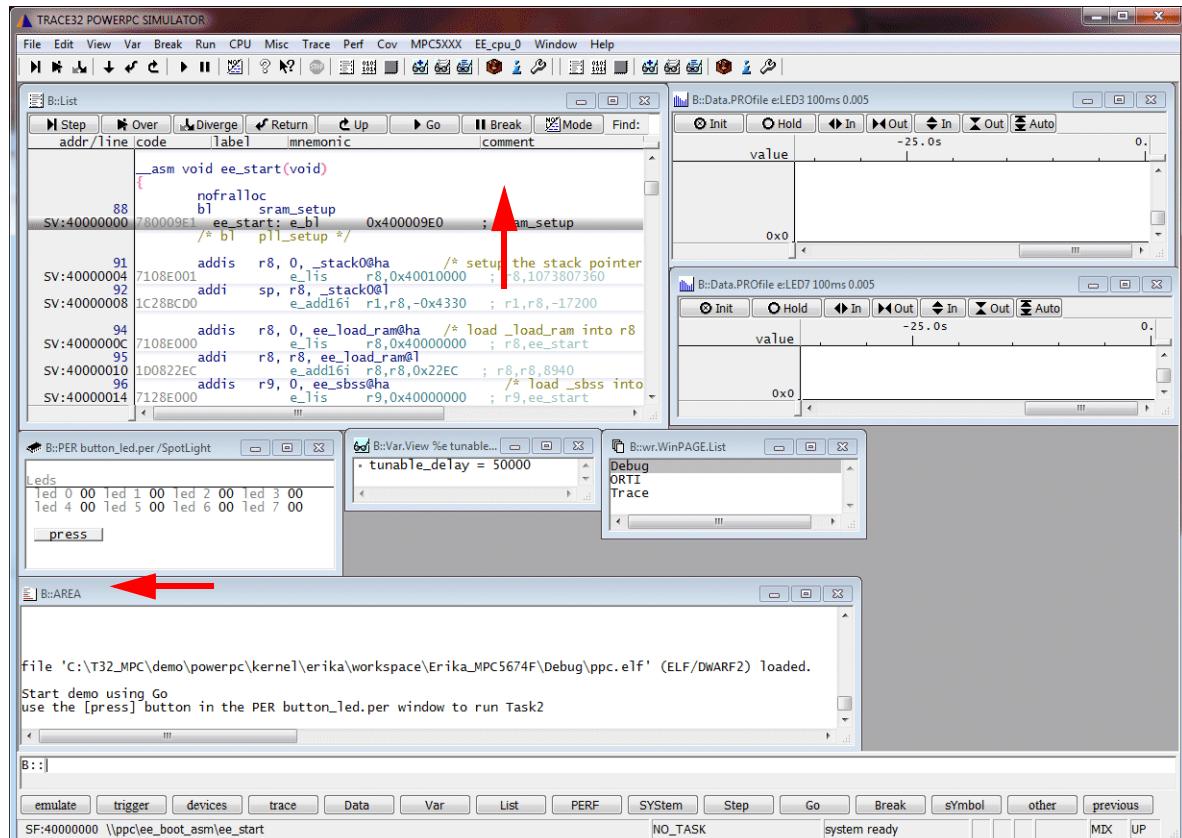
Use the **press** button in the **PER button_led.per** window to start the demo and use the **break** button to stop it.

TRACE32 PowerView shows the OS resources after the demo stopped.

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.
```
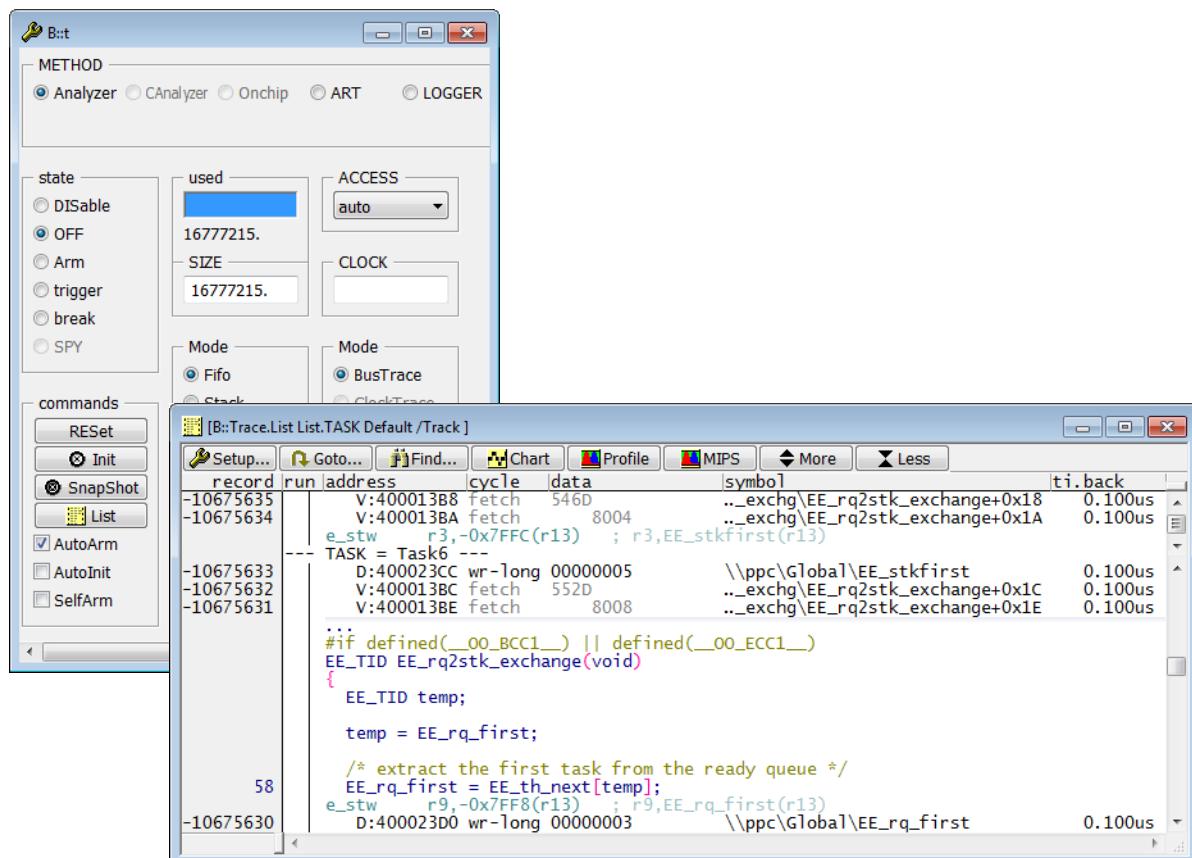
NEXUS Class 3

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
```
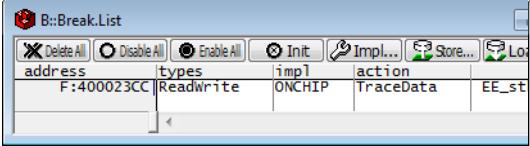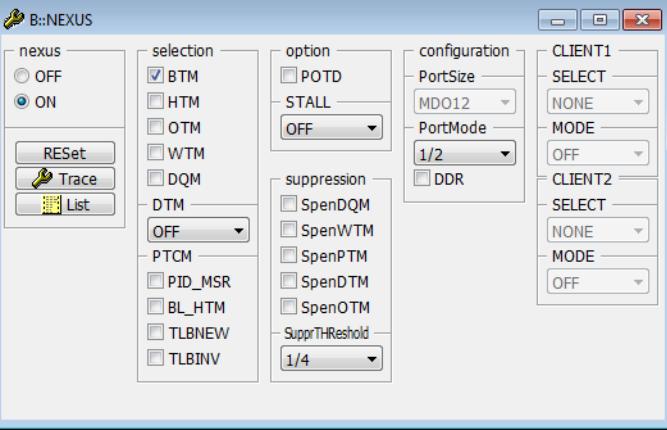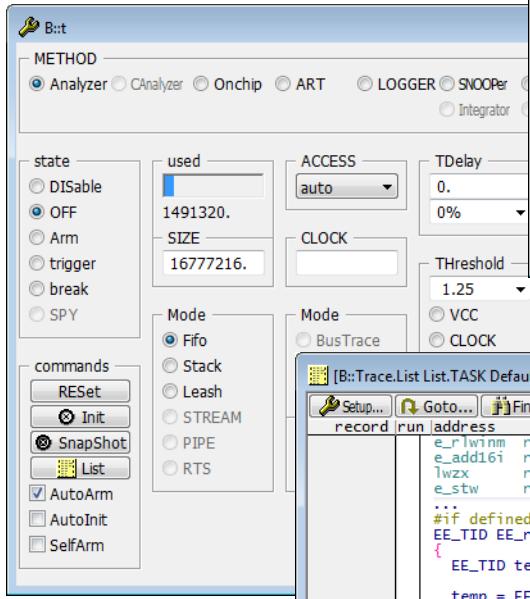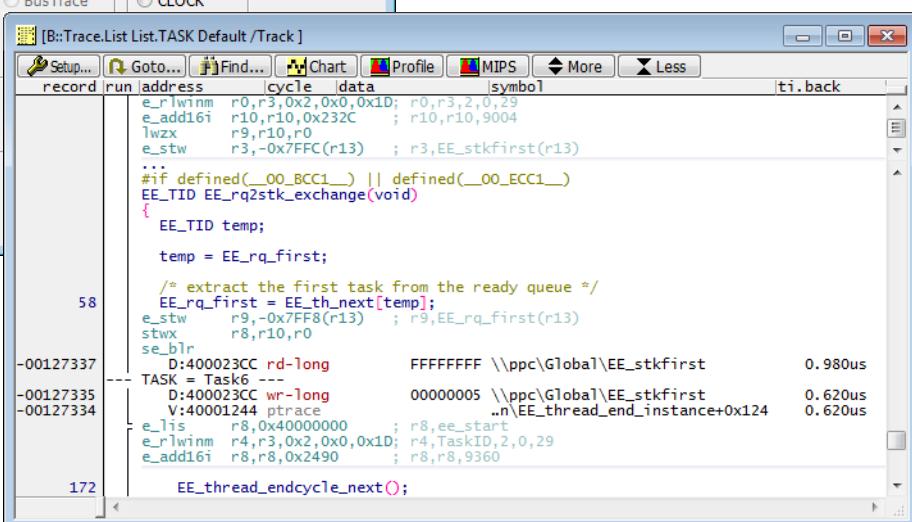
NEXUS Class 2

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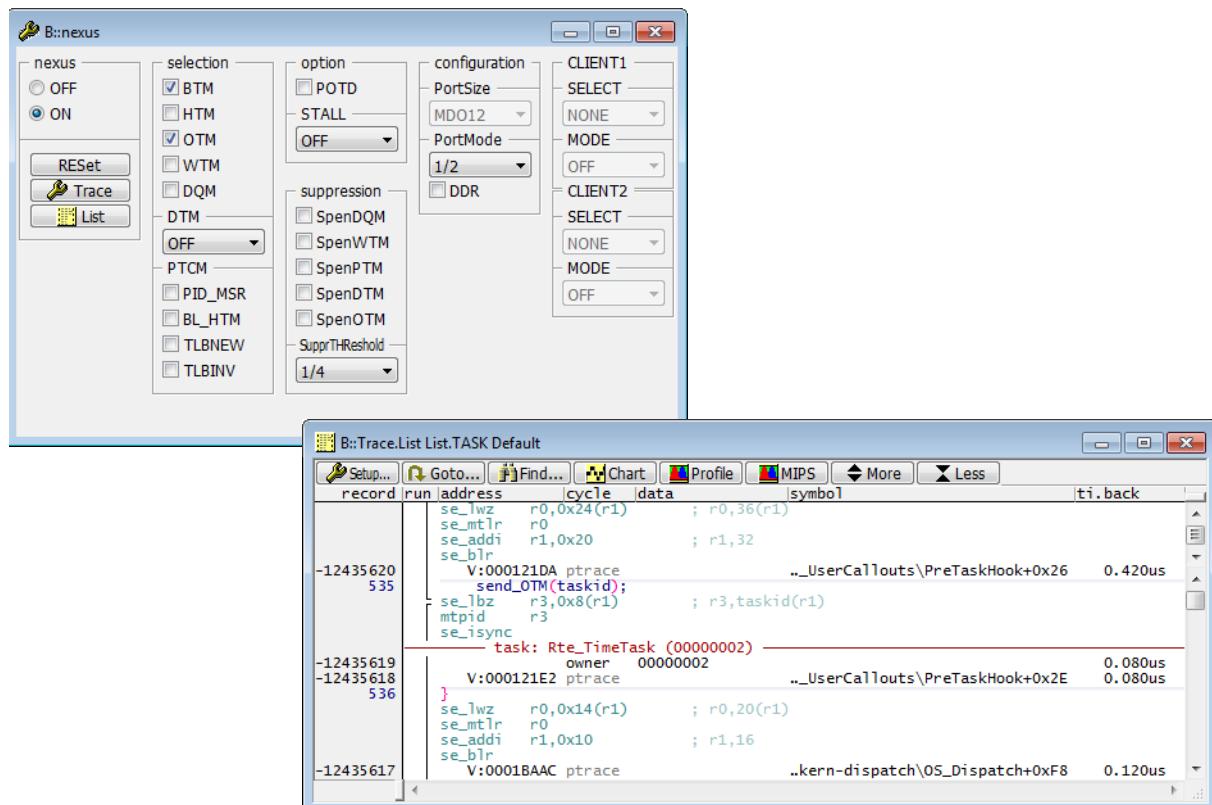
Step 4: Run the Program Execution to Fill the Trace

Display a Trace Configuration window (**Trace.state)** and start the program execution.
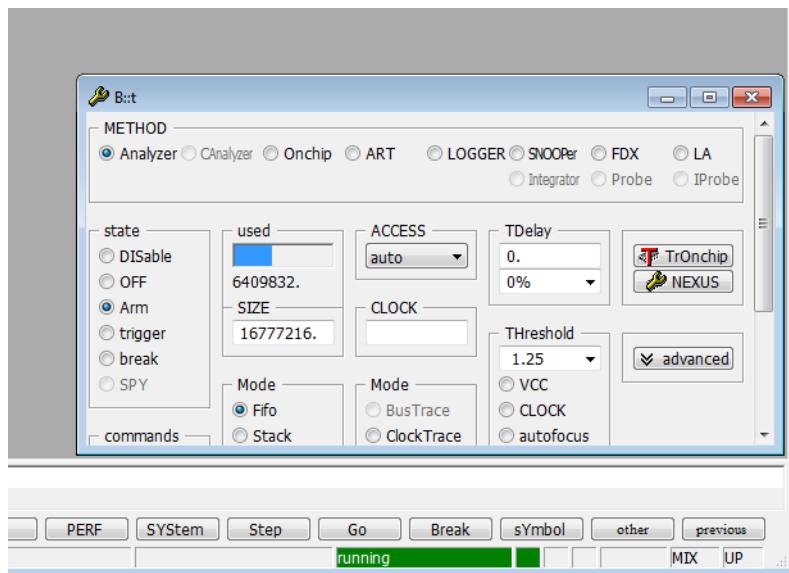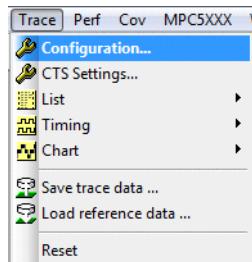
Stop the program execution by pushing the [Break] button.

Use the **Trace.Chart.sYmbol** command to check if the trace information was recorded without errors.
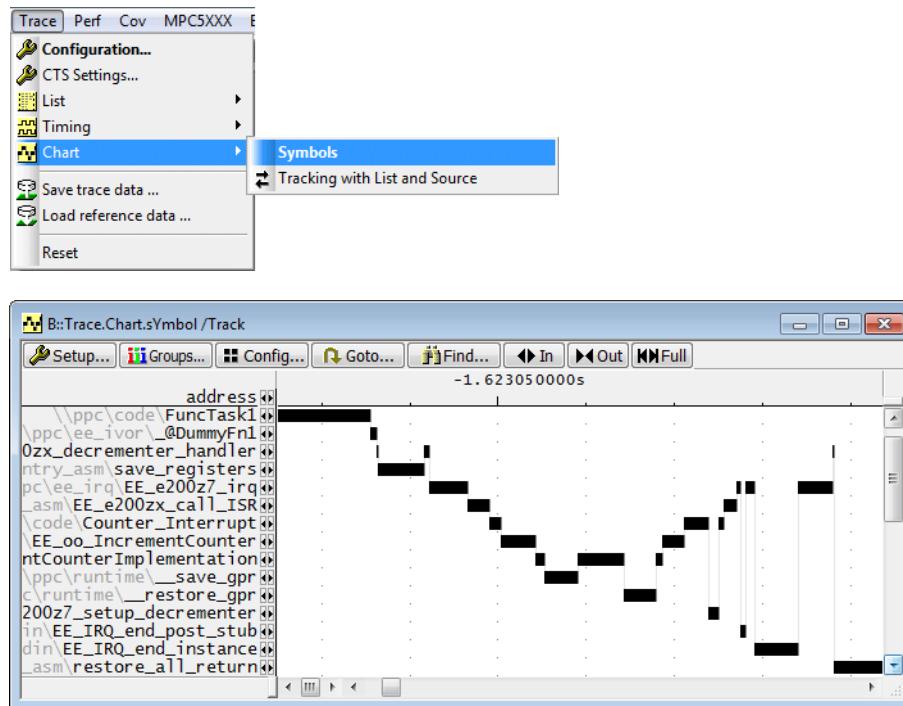
Details on possible errors and their causes can be found in **"FlowErrors"** in Nexus Training, page 48 (training_nexus.pdf) and **"FIFOFULL"** in Nexus Training, page 44 (training_nexus.pdf). Please be aware that an error-free trace is required in order to export task event information.
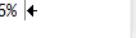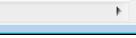
Use the **Trace.Chart.TASK** command to inspect the task switches.
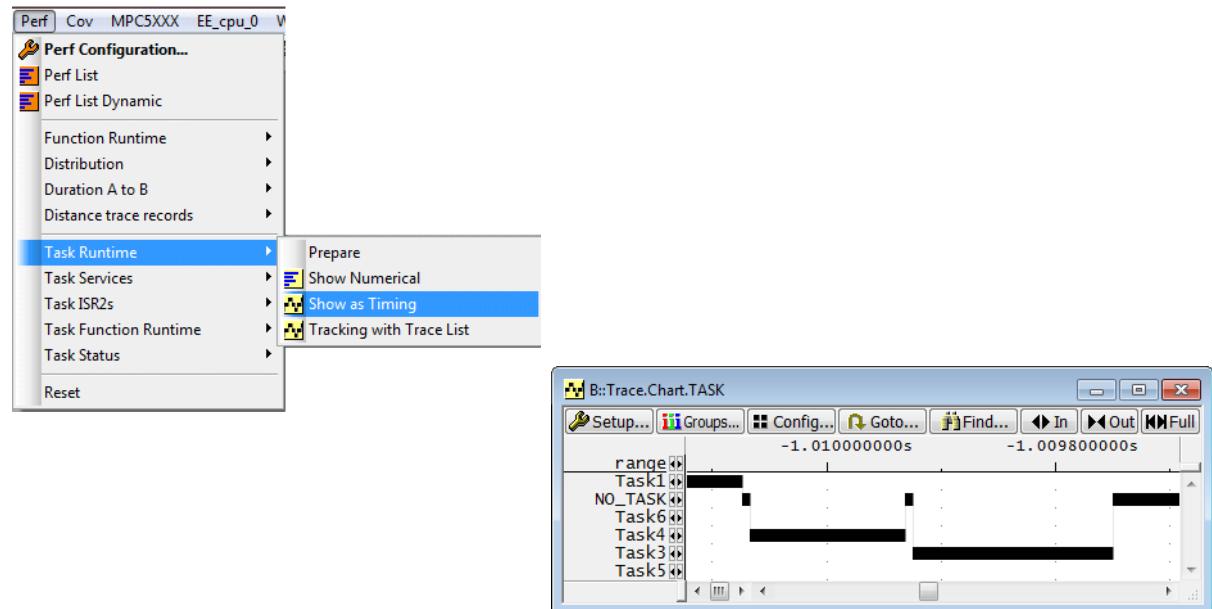
The command **Trace.STATistic.TASK** provides the same result in a numeric display.

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:

# Timing Tools

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.
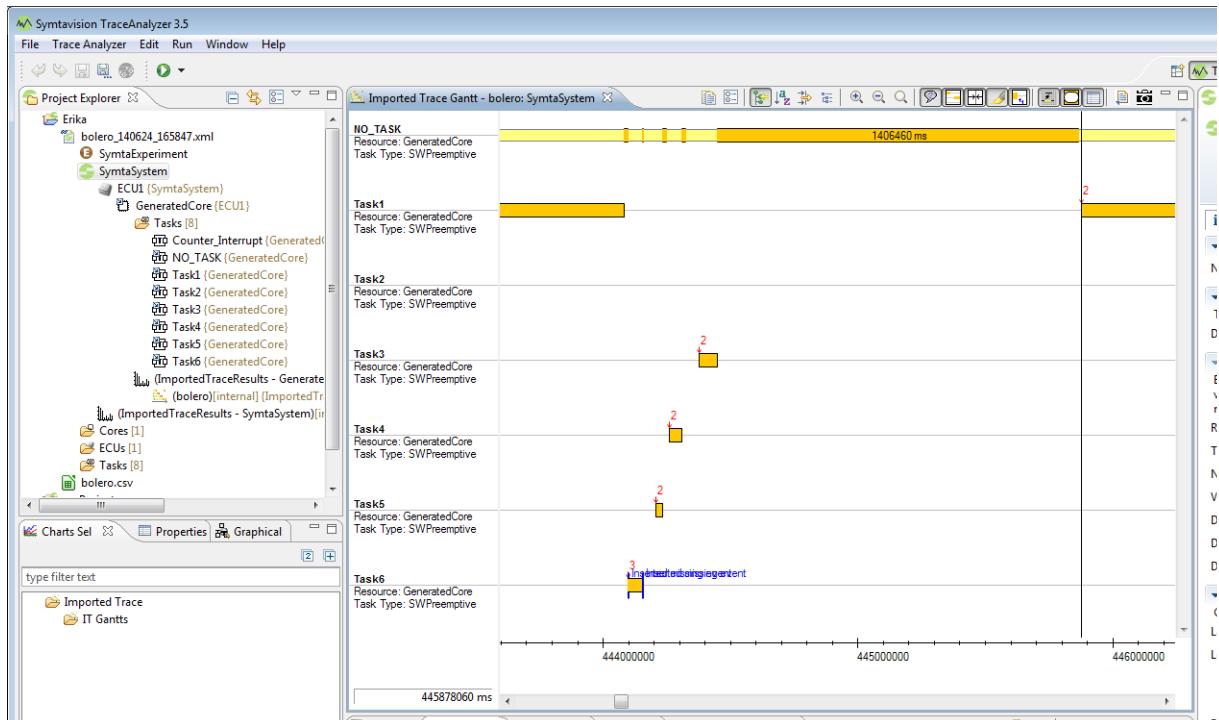
INCHRON chronVIEW

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.
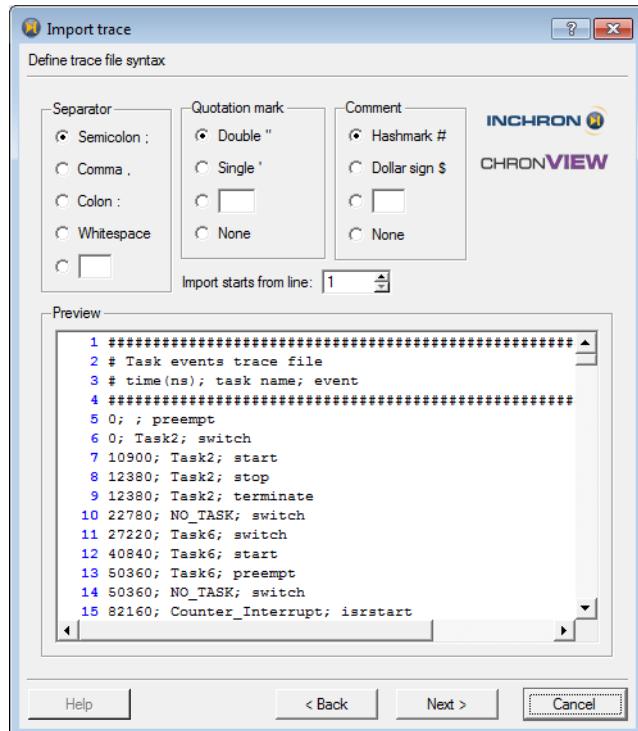
4. Define column semantics.
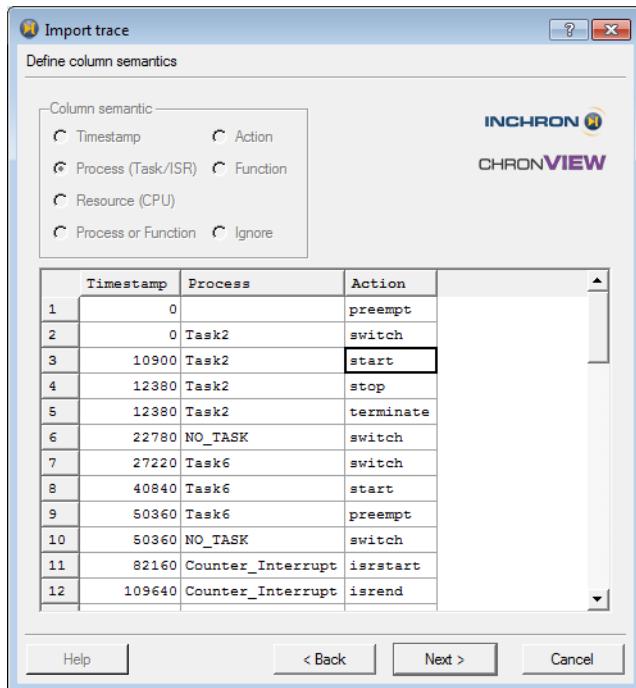
5. Specify tasks, ISRs and functions.
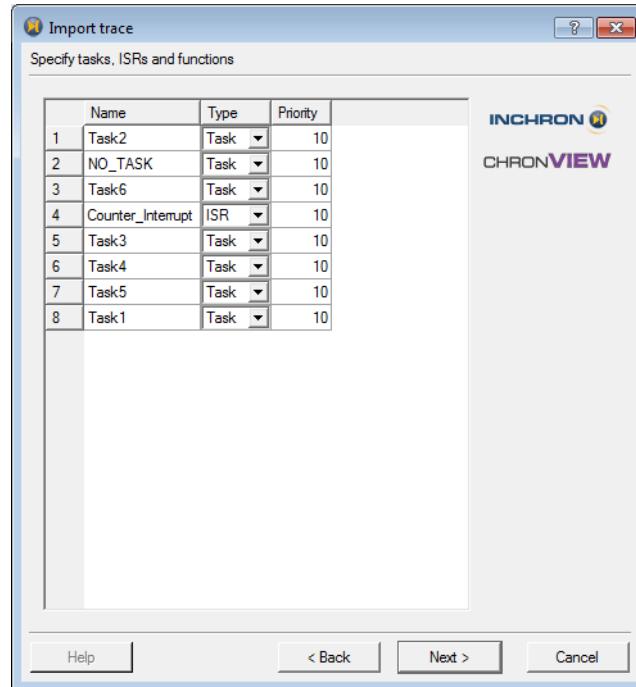
6. Define action semantics.
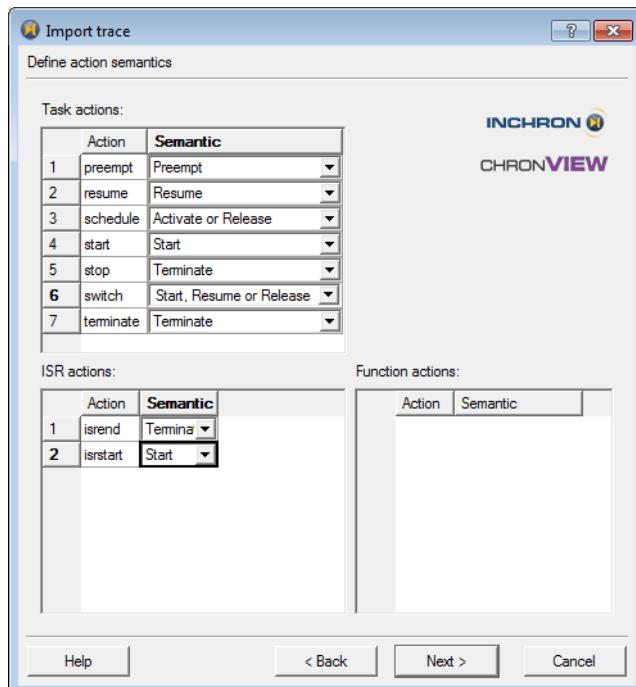
7.   Define timestamp interpretation.
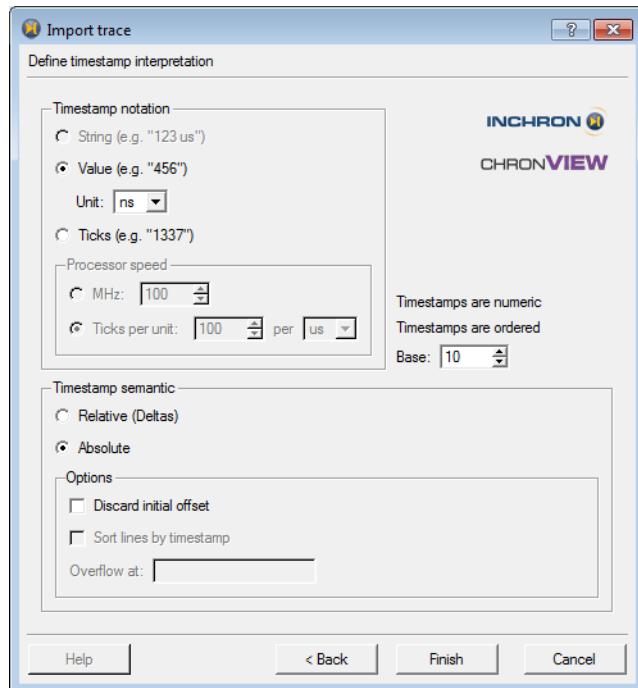
8.   Press the **Finish** button to get the result.

Timing Architects - Inspector
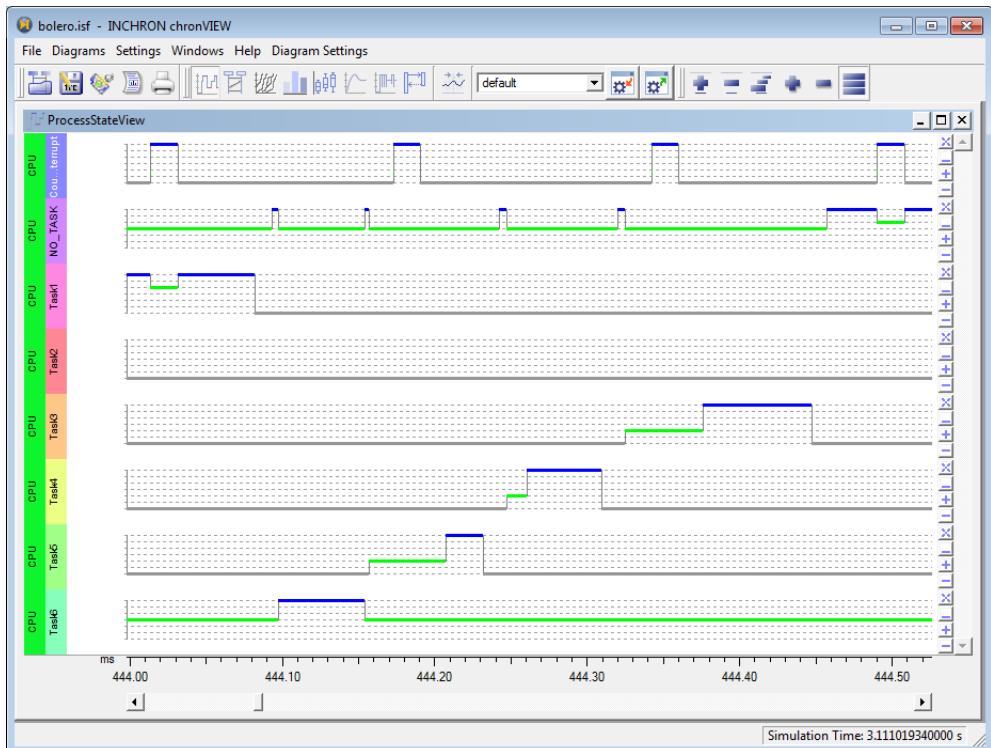
In order to analyze your trace recording with the TA inspector proceed as follows:

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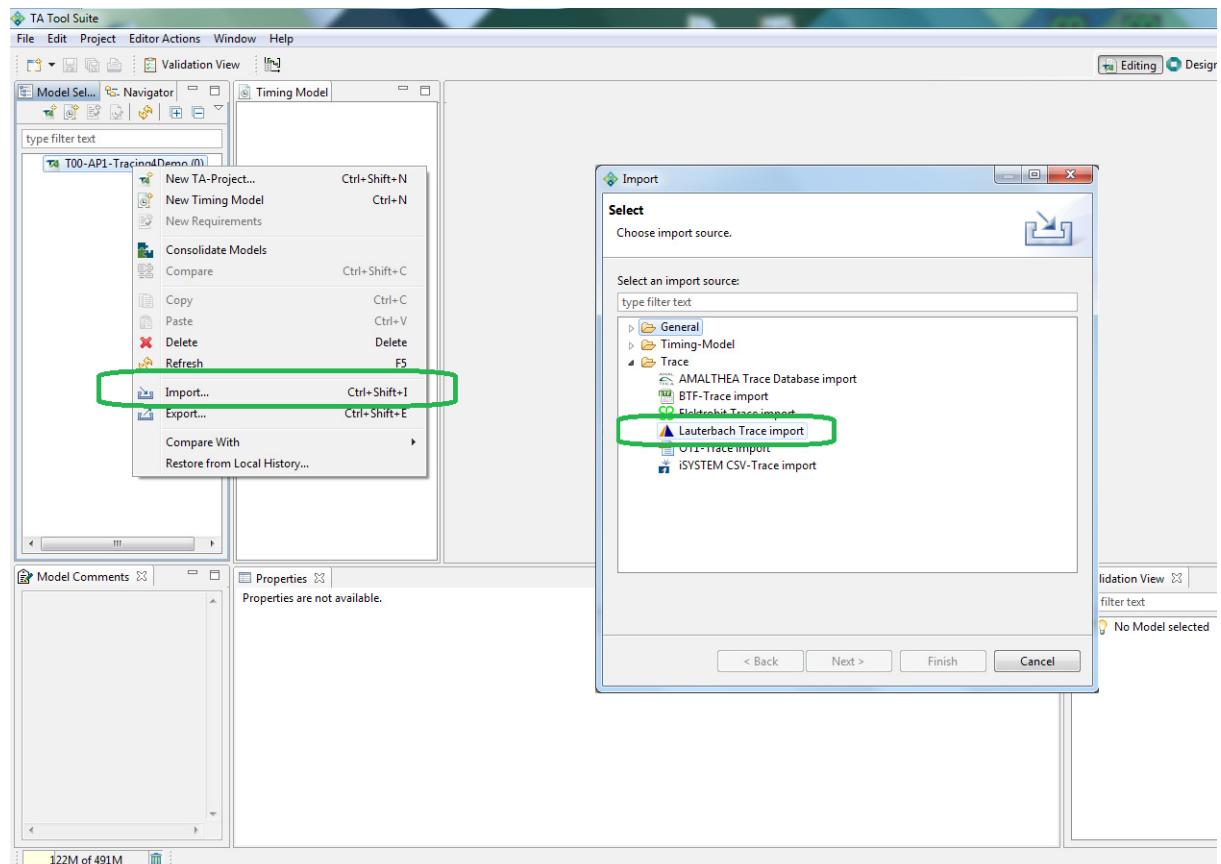
3. Select **Import…** from the appearing context menu.

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.
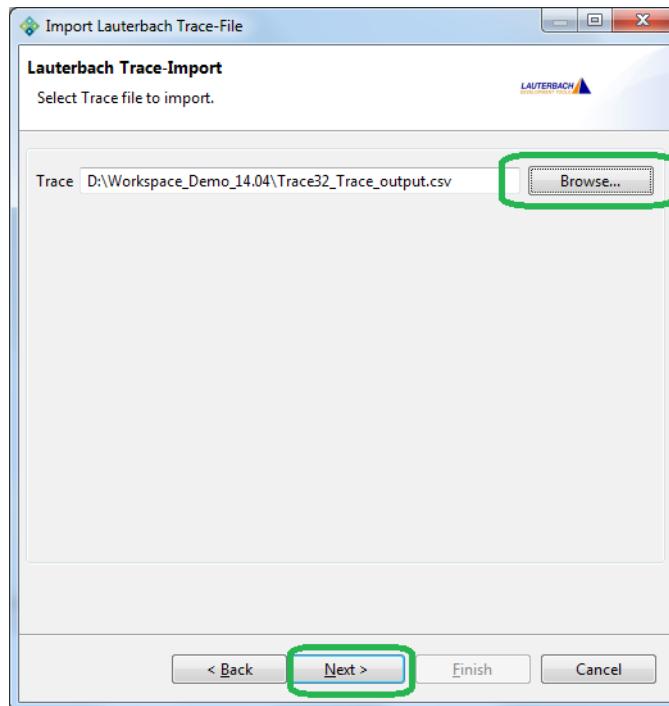
6. Then select the project and specify the name for the timing model.
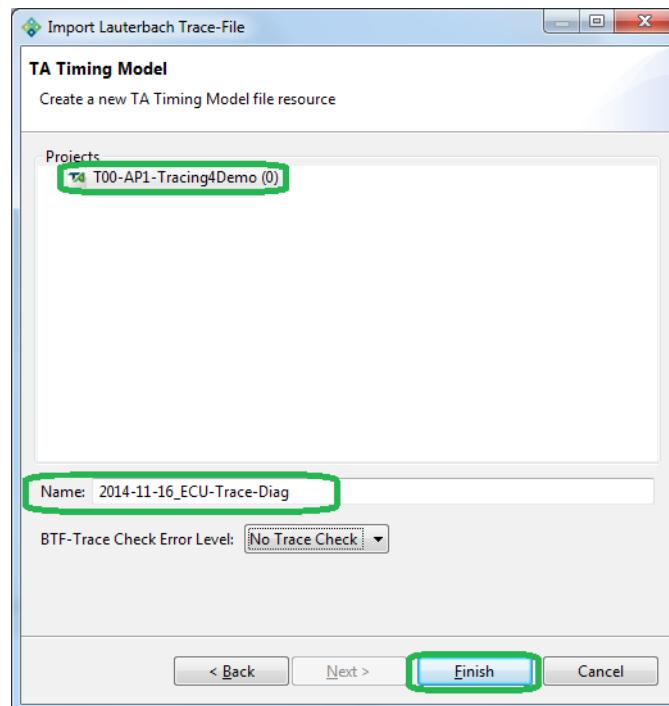
7. Click finish to start the import process.

8. After the import is completed left-click the **Inspector** button (top right corner).
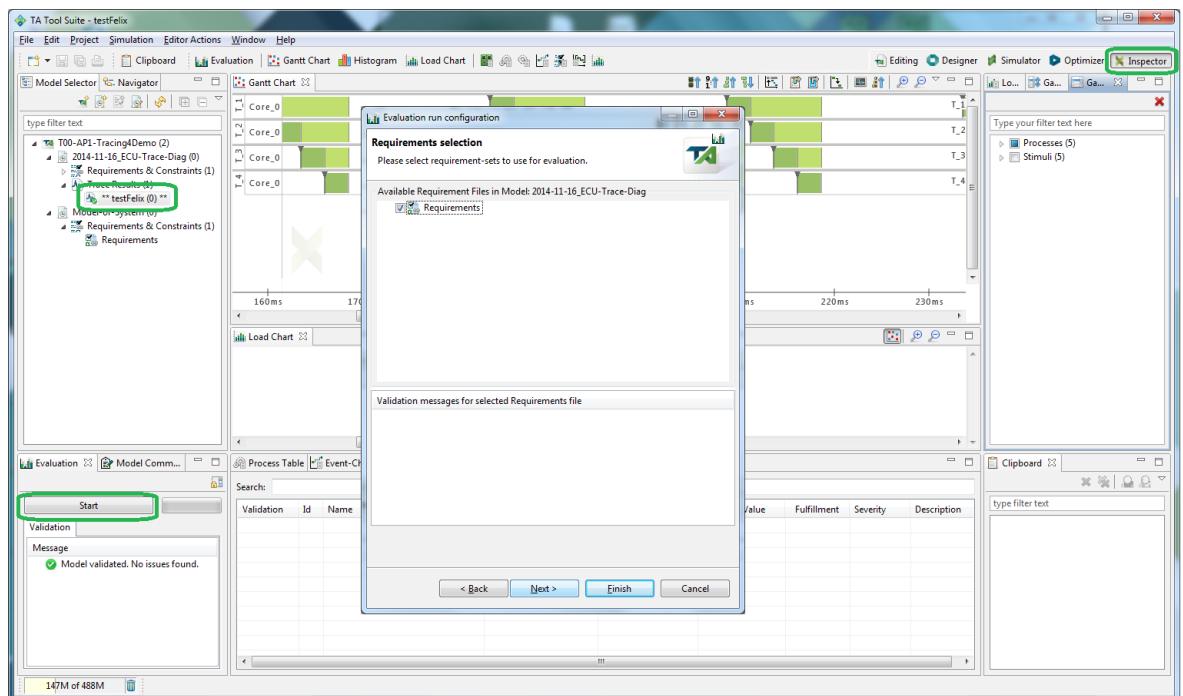
9. Select a trace file and start the calculation.

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

11. Additionally the **evaluation configuration parameters** need to be specified.
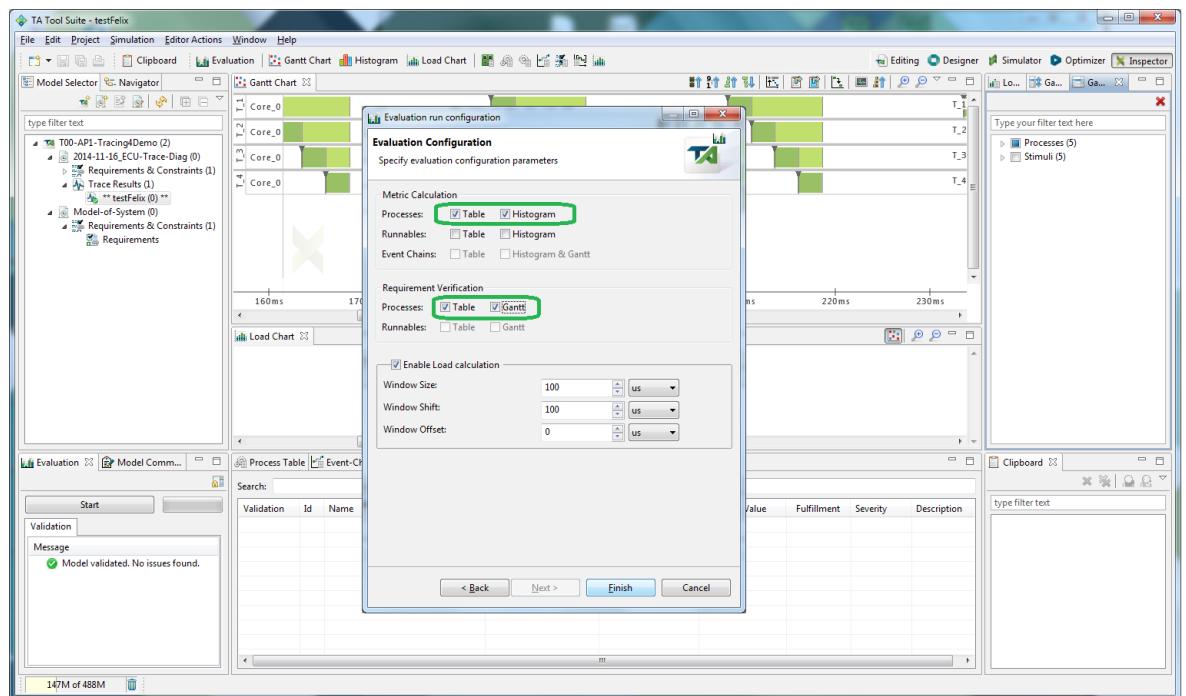
12. When the calculation is done, the results are displayed.