ETAS RTA-HVR Hypervisor & Multi RTA-OS Profiling

Publish Date: 02/01/2018

Table of Contents

1 Introduction 2
1.1 OS and Hypervisor Event Signaling for Trace 2
2 winIDEA Configuration for OS/Hypervisor Awareness 3
2.1 AUTOSAR OS on Master Core 4
2.2 Hypervisor on Slave Core 4
2.3 AUTOSAR OSes on the Slave Core 5
3 Trace & Profiler Configuration 7
3.1 OS Profiler Configuration 7
3.2 Manual Trace Configuration 9
4 Profiling 11
4.1 Sample Profiler Timeline 11
4.2 Task Running State within the VM Context 11
5 Technical Support 13
5.1 Online Resources 13
5.2 Contact 13

1 Introduction

This document describes how to use winIDEA for Multi-OS timing analysis (profiling) using the ETAS Lightwight Hypervisor, RTA-HVR. The description is based on a demo application provided by ETAS. The demo application executes on a dual-core SPC58EC *Chorus4M* device.
The master core, which is operational after reset, executes a standards AUTOSAR OS (RTA-OS) and starts the slave core. The slave core executes the hypervisor which provides two virtual machines (VM0 and VM1). Within each of these VMs an (adapted) AUTOSAR OS (RTA-OS) is running.

1.1 OS and Hypervisor Event Signaling for Trace

To allow a tracing/profiling tool to reconstruct the timing behavior of such a system, both the hypervisor and the individual OSes much signal specific events to the tool.

1.1.1 AUTOSAR OS on Master Core

The currently running task and running ISR Cat.2 are signaled by the OS according to the OSEK ORTI standard.

- **RUNNINGTASK**: The OS writes the currently running task pointer into a global variable.
- **RUNNINGISR2**: The OS writes the currently running ISR2 pointer into a global variable.

1.1.2 Hypervisor on Slave Core:

The hypervisor signals the currently running virtual machine. This is not covered by the ORTI standard.

- **Running VM:** The hypervisor writes the ID of the currently running VM into the PID0 register of the slave core, which subsequently emits a Nexus Ownership Trace Message (OTM). The PID0 register is only accessible in the Supervisor mode of the core. The hypervisor (kernel) executes in Supervisor mode.

1.1.3 AUTOSAR OSes on Virtual Machine of Slave Core

The currently running task and running ISR Cat.2 are signaled by the OS via techniques that are not covered by the ORTI standard.
Signaling by means of Ownership Trace (OTM) is not possible as the OS within a hypervisor VM runs in the lower privilege User mode of the core and thus cannot write to the PID0 register. Writing to global data objects (as used by the Master OS) is not also not applicable as a set of global variables for signaling running task and running ISR2 would be needed for each VM, i.e. for each OS. A real system may implement many VMs. However, typically the on-chip trace logic of a micro-controller is only capable of monitoring between 2 or 8 data objects simultaneously (4 in case of a Chorus4M).
Therefore, in this demo project, running on a SPC58x device we utilized the Nexus Data Acquisition Message (DQM) to signal running task and ISR2. Generating a DQM is triggered by writing at the DDAM register of the core, which is also supported in User mode.

- **RUNNINGTASK**: The OS writes the currently running task ID into the DDAM register of the slave core, which subsequently emits a Nexus Data Acquisition Message (DQM). The lower 2 LSBs of DDAM hold a message ID. For RUNNINGTASK, this message ID is set to 1.
- **RUNNINGISR2***:* The OS writes the currently running ISR2 ID into the DDAM register of the slave core, which subsequently emits a Nexus DQM. The lower 2 LSBs of DDAM hold a message ID. For RUNNINGISR2, this message ID is set to 2.

2 winIDEA Configuration for OS/Hypervisor Awareness

OS-Awareness means that winIDEA has information about the OS structure, i.e. existing tasks, ISRs and other OS objects such as alarms, etc. In addition, winIDEA knows how specific OS events, such as task switches, are signaled by the OS. This information is utilized for two different purposes:

- Display of OS status while the CPU is stopped, by reading out the associated data objects from memory.
- Tracing/Profiling of OS event, such as task and ISR scheduling while the CPU is running.

In case of an AUTOSAR OS, a project-specific ORTI file must be imported into winIDEA to make it AUTOSAR OS aware. The ORTI file in an optional output by the OS generator when generating the OS source code. However, in case of a hypervisor-based application, consisting of multiple AUTOSAR OSes, and the hypervisor itself, this ORTI-based approach is not sufficient anymore. Thus, winIDEA has been extended to allow for awareness of multiple AUTOSAR OSes, plus hypervisor.
The menu “Debug – Operating System…” now allows for specifying multiple OSes. One of these “OSes” is used for the hypervisor (HVR).

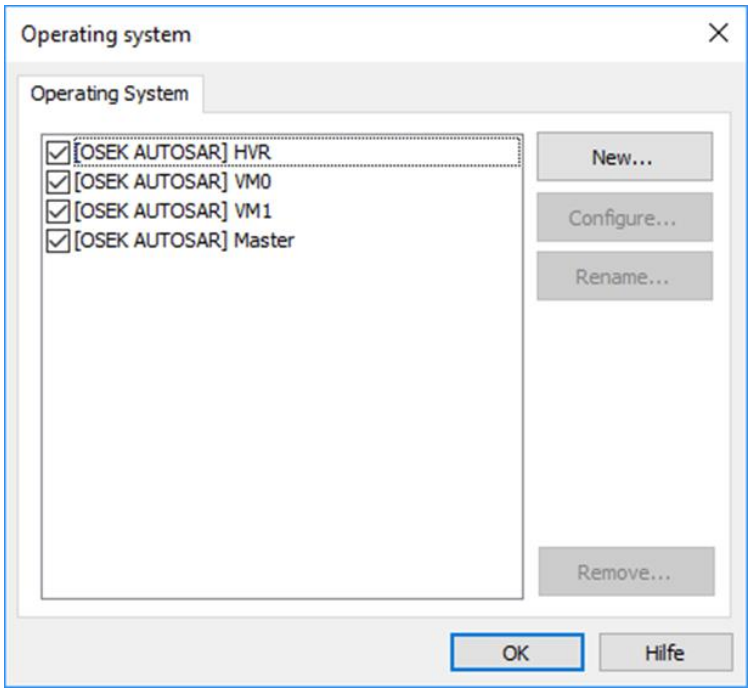

Figure 1: OS/HVR Awareness in winIDEA

Each OS can be described with two different methods. The “traditional” method is reading in an ORTI file (generated by the AUTOSAT OS generator). The new method applied here is reading in an iSYSTEM proprietary XML file.
Note: winIDEA reads in the ORTI/XML file contents on “Debug – Download” and “Debug – Load Symbols only”.

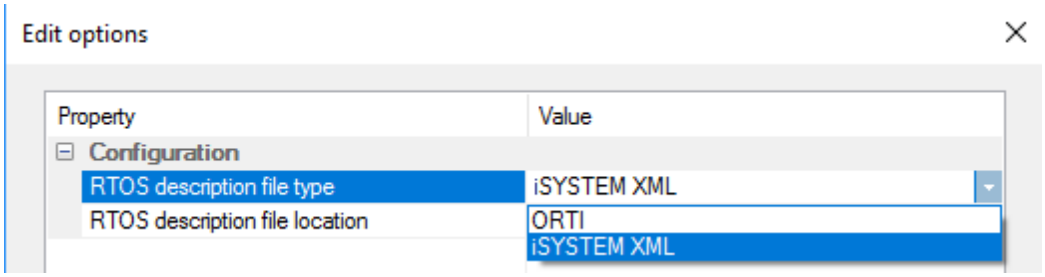

Figure 2: iSYSTEM Profiler XML File Option

2.1 AUTOSAR OS on Master Core

The file master.xml describes the AUTOSAR OS running on the master core.

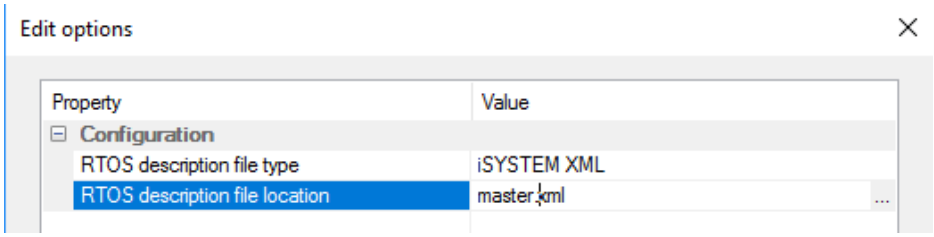

Figure 3: iSYSTEM Profiler XML file selection for the Master AUTOSAR OS

The master.xml file imports all OS information from the ORTI file iSystemExample.orti. The description of RUNNINGTASK and RUNNINGISR2 is then extended to indicate that these tasks and ISR2 are executed by the master core. Thus, the profiler uses the object names Master Tasks and Master ISR2.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <OperatingSystem>
3    <Name>Master</Name>
4    <ORTI>..\iSystemExample.orti</ORTI>
5    <NumCores>1</NumCores>
6
7    <Profiler>
8      <Object>
9        <Definition>RUNNINGTASK</Definition>
10       <Description>Master_Tasks</Description>
11     </Object>
12
13     <Object>
14       <Definition>RUNNINGISR2</Definition>
15       <Description>Master_ISR2</Description>
16     </Object>
17
18   </Profiler>
19 </OperatingSystem>
```

Figure 4: iSYSTEM Profiler XML for the Master AUTOSAR OS

2.2 Hypervisor on Slave Core

The file HVR.xml describes the hypervisor running on the slave core.

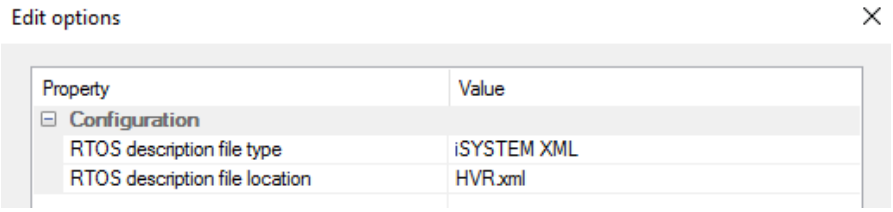

Figure 5: iSYSTEM Profiler XML file Selection for the Hypervisor

The HVR.xml file describes how the hypervisor signals virtual machine switches to the trace tool by writing a VM ID to the PID0 register (i.e. Nexus OTM). The mapping between VM name (“VM0”, “VM1” and “HVR”) is also given by means of an ENUM type.

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<OperatingSystem>
  <Name>RTA_HVR</Name>
  <NumCores>2</NumCores>
  <Types>
    <TypeEnum>
      <Name>Master</Name>
      <Enum><Name>Master</Name><Value>0</Value></Enum>
    </TypeEnum>

    <TypeEnum><Name>Application</Name>
      <Enum><Name>VM0</Name><Value>0</Value></Enum>
      <Enum><Name>VM1</Name><Value>1</Value></Enum>
      <Enum><Name>HVR</Name><Value>2</Value></Enum>
    </TypeEnum>

    </Types>
  <Profiler>
    <Object>
      <Definition>RUNNINGAPP1</Definition>
      <Level>Application</Level>
      <Name>RUNNINGAPP1</Name>
      <Core>1</Core>
      <DefaultValue>HVR</DefaultValue>
      <Description>RunningVM</Description>
      <Signaling>OTM</Signaling>
      <Type>Application</Type>
    </Object>

    <Object>
      <Definition>RUNNINGAPP0</Definition>
      <Level>Application</Level>
      <Name>RUNNINGAPP0</Name>
      <Core>0</Core>
      <DefaultValue>Master</DefaultValue>
      <Description>MasterApp</Description>
      <Signaling>OTM</Signaling>
      <Type>Master</Type>
    </Object>

  </Profiler>
</OperatingSystem>
```

Figure 6: iSYSTEM Profiler XML for the Hypervisor

2.3 AUTOSAR OSes on the Slave Core

The files VM[n].xml describe the AUTOSAR OSes running in the virtual machine n of the slave core.

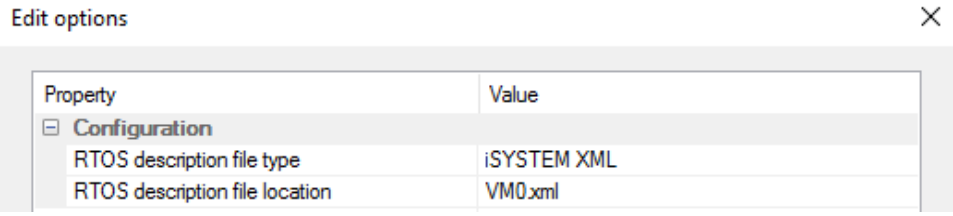

Figure 7: iSYSTEM Profiler XML Selection for the AUTOSAR OS in VM0

The VM[n].xml files import all OS information from the corresponding ORTI file.
The description of RUNNINGTASK and RUNNINGISR2 is then extended to indicate that these tasks and ISR2 are executed within the associated VM. Thus, the profiler uses the object names VM[n]_Tasks and VM[n]_ISR2. In addition, the XML file extends the RUNNINGTASK and RUNNINGISR2 definition of the ORTI file with a definition of “ISYS_RUNNINGTASK” and “ISYS_RUNNINGISR2”, using the DQM-based signaling approach.

```
<?xml version='1.0' encoding='UTF-8' ?>
<OperatingSystem>
  <Name>VM0</Name>
  <NumCores>1</NumCores>
  <ORTI>..\VM0\HelloWorld.orti</ORTI>
  <Profiler>
    <Object>
      <Definition>RUNNINGTASK</Definition>
      <Level>None</Level>
    </Object>
    <Object>
      <Definition>RUNNINGISR2</Definition>
      <Level>None</Level>
    </Object>

    <Object>
      <Definition>ISYS_RUNNINGTASK</Definition>
      <Name>ISYS_RUNNINGTASK</Name>
      <Type>OS:vs_Signal_RUNNINGTASK</Type>
      <Core>1</Core>
      <Description>VM0_Tasks</Description>
      <Level>Task</Level>
      <DefaultValue>idle</DefaultValue>
      <Signaling>DQM(0).0.2.1</Signaling>
    </Object>
    <Object>
      <Definition>ISYS_RUNNINGISR2</Definition>
      <Name>ISYS_RUNNINGISR2</Name>
      <Type>OS:vs_Signal_RUNNINGISR2</Type>
      <Core>1</Core>
      <Description>VM0_ISR2</Description>
      <Level>IRQ0</Level>
      <DefaultValue>NO_ISR</DefaultValue>
      <Signaling>DQM(0).0.2.2</Signaling>
    </Object>
  </Profiler>
</OperatingSystem>
```

Figure 8: iSYSTEM Profiler XML for an AUTOSAR OS within a VM

3 Trace & Profiler Configuration

3.1 OS Profiler Configuration

When enabling OS profiling by selecting “Profile – OS objects”, all OSes (and hypervisor) configured by means of the “Debug – Operating System…” will be included in the profiler analysis.

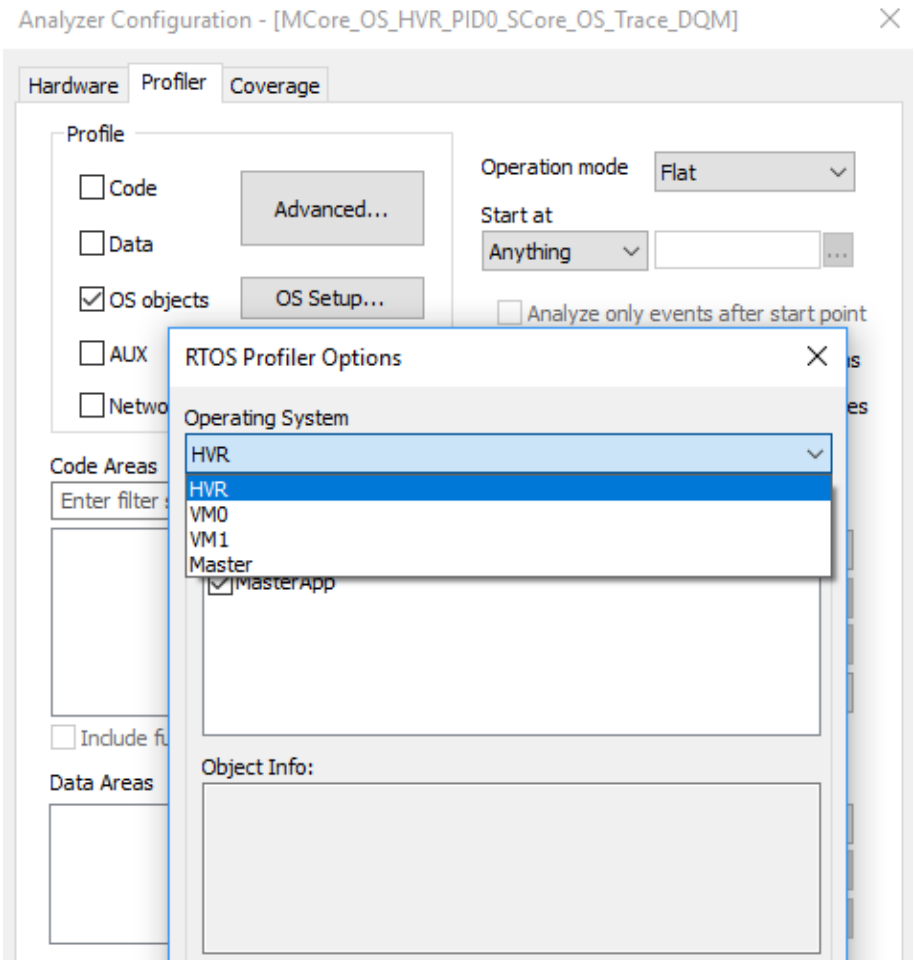

Figure 9: Hypervisor / Multi-OS Configuration in the Profiler

Individual objects of each OS can be included/excluded to/from the analysis via the OS selection opened by means of the “OS Setup…” button.

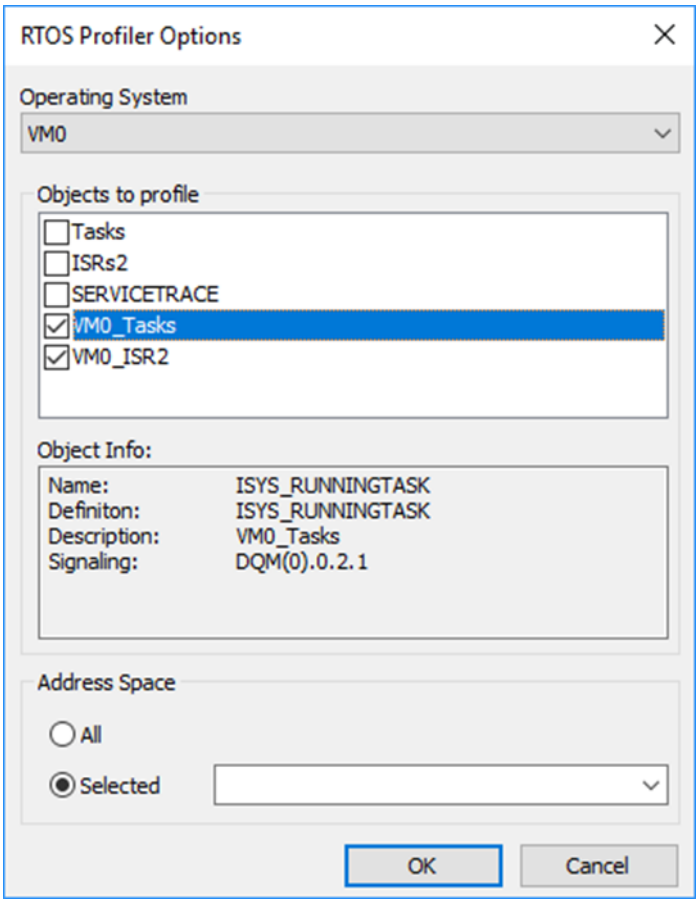

Figure 10: Object Selection for each OS

The objects listed for each OS correspond to the objects obtained via the iSYSTEM XML file and the ORTI file included into the XML file. In the example, shown in Figure 10, only the OS objects described in the iSYSTEM XML are selected for profiling. The objects “Tasks”, “ISRs2” and “SERVICETRACE” of the ORTI file are ignored.

3.2 Manual Trace Configuration

For many use-cases a manual trace configuration should not be necessary as the winIDEA Analyzer automatically configures the on-chip trace logic according to the settings in the profiler. However, for the sake of completeness, also a complete manual configuration is described in this section.
Master Core (CPU2): As mentioned previously, the AUTOSAR OS of the master core signals the currently running task and ISR2 by writing a pointer value to a global variable which is monitored by the trace tool by means of on-chip data trace.

Table 1: ORTI attribute to global variable mapping.

OS Object	Global Variable
RUNNINGTASK	Os_RunningTask,,<master_ELF_file>
RUNNINGISR2	Os_RunningISR,,<master_ELF_file>

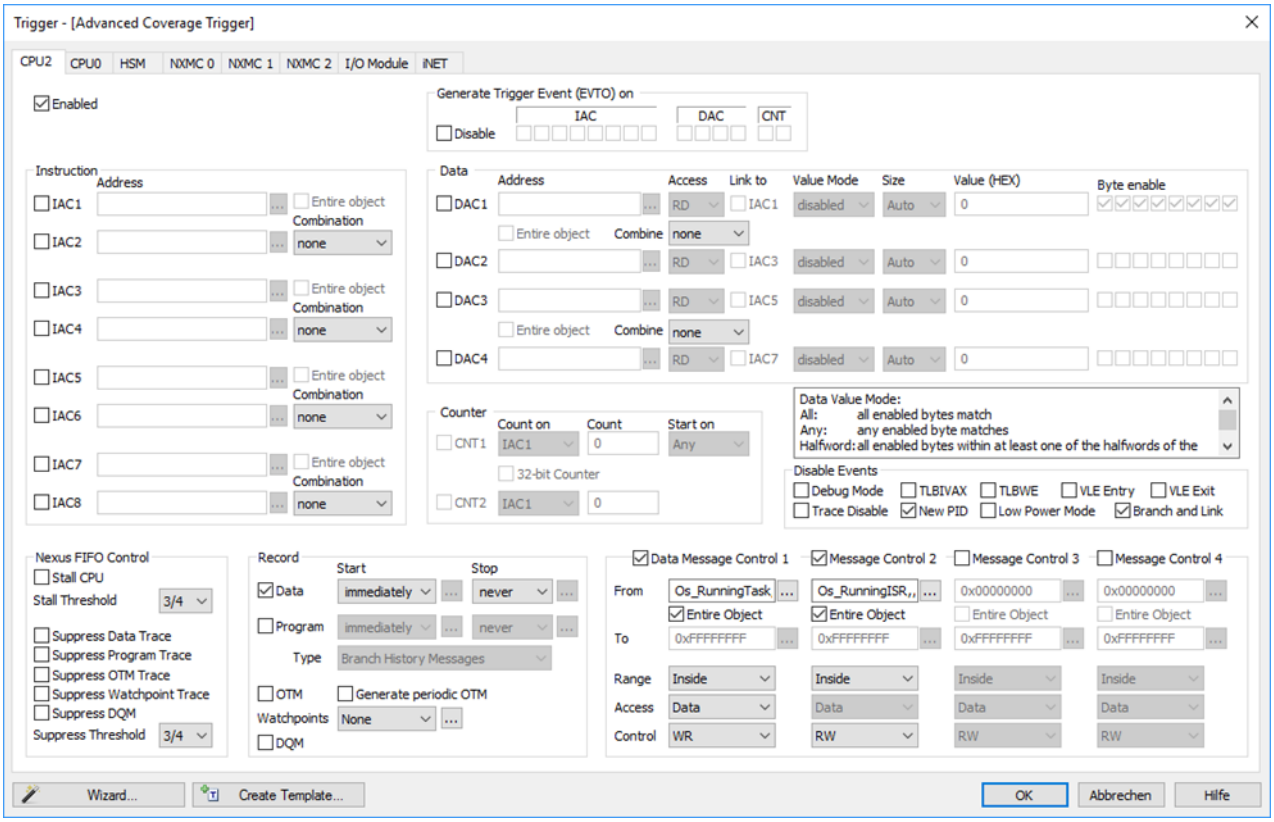

Figure 11: Manual Trace Configuration on Master Core (CPU2)

Slave Core (CPU0): As mentioned previously, the AUTOSAR OSes executing within the VMs of the slave core signal the currently running task and ISR2 by writing an ID value to the DDAM register of the core which emits a NEXUS DQM, monitored by the trace tool.
The hypervisor of the slave core signals the currently active VM by writing an ID value to the PID0 register of the core which emits a NEXUS OTM, monitored by the trace tool.

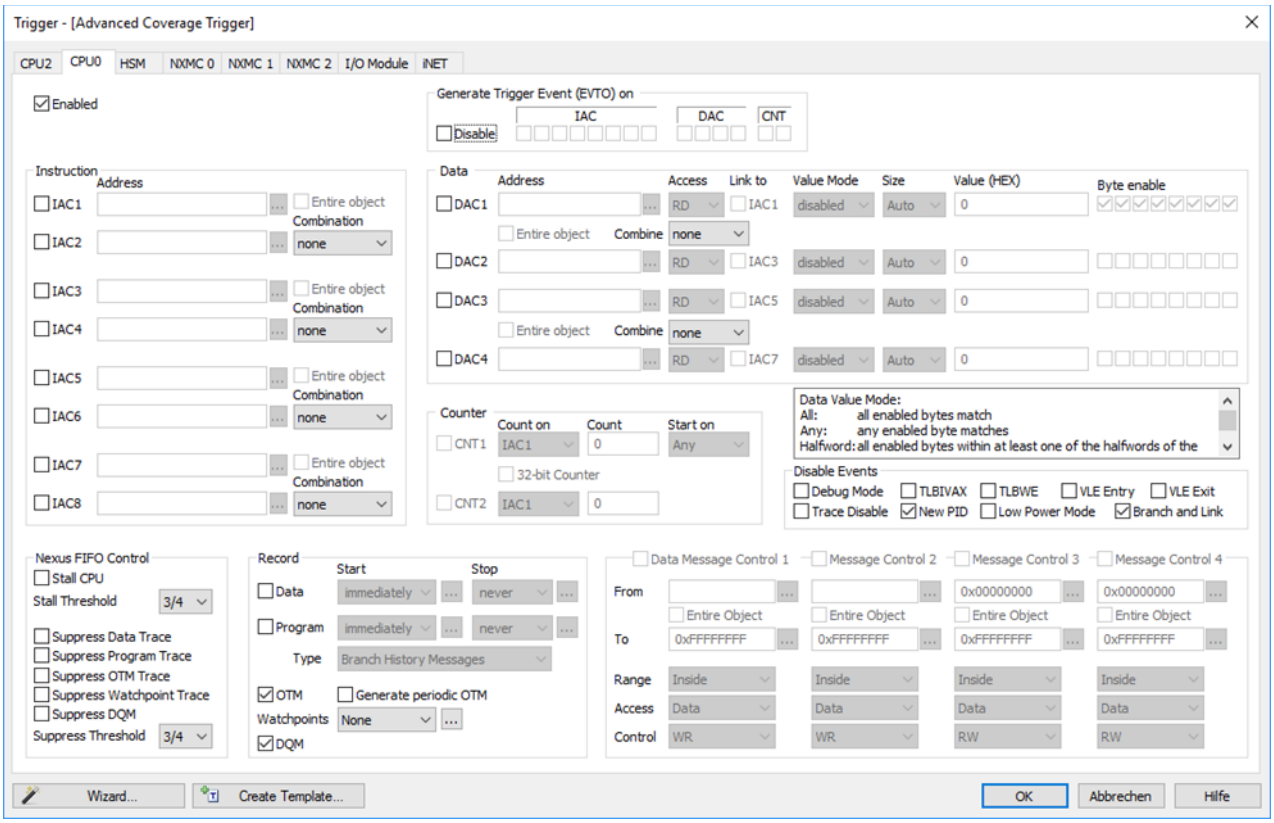

Figure 12: Manual Trace Configuration on Slave Core (CPU0)

4 Profiling

4.1 Sample Profiler Timeline

The winIDEA Analyzer allows a profiling and visualization of multiple OSes simultaneously.

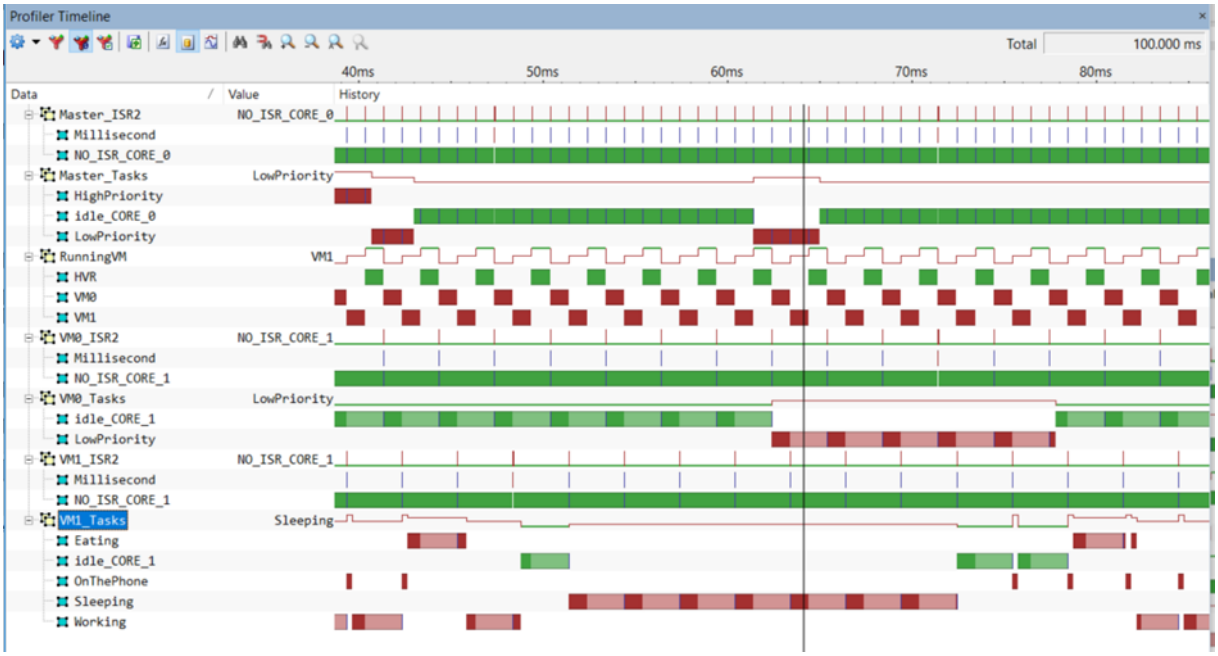

Figure 13: Sample Profiler Timeline

4.2 Task Running State within the VM Context

In the figures below, we take a closer look at the timing behavior of the task “HighPriority” executing within VM0 on the slave core.

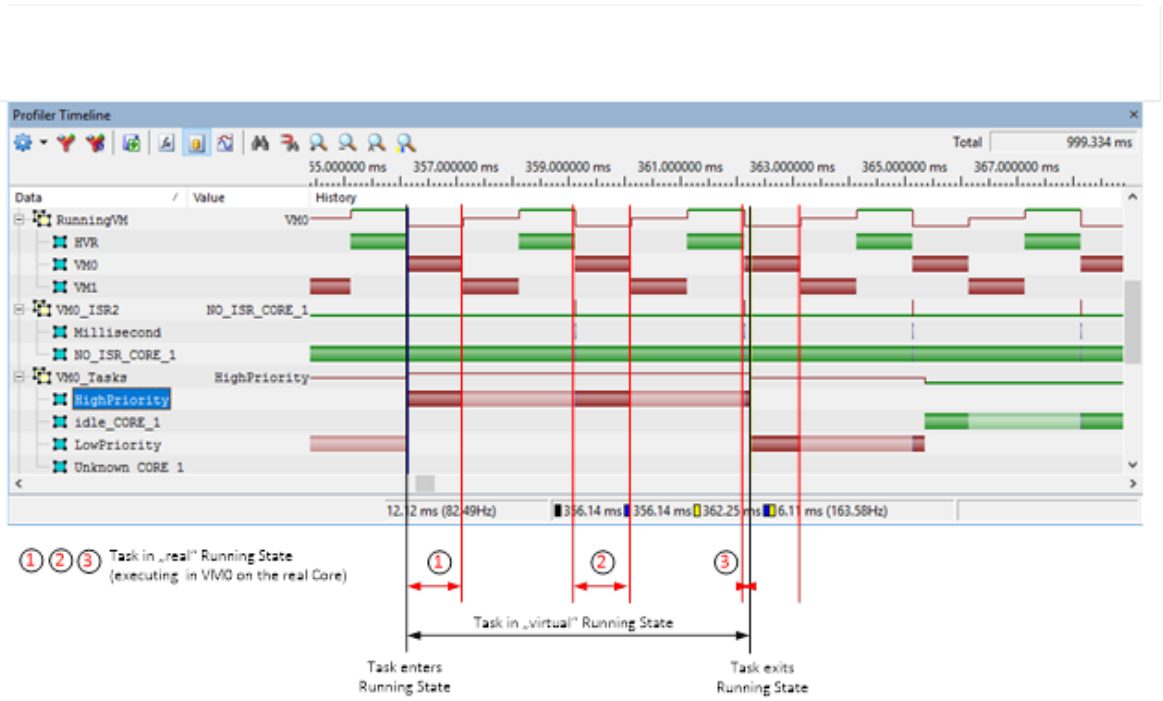

Figure 14: Sample Profiler Timeline, detailed Task Runtime with VM Switches

The dark red bar during the time periods (1), (2) and (3) indicate that the task “HighPriority” really executes on the real core while the associated VM0 is active (core execution time). The accumulated core executing time between the point in time when the task first enters running state until the time the task exists running state, represents the “Net Execution Time” of the task. In this example the Net Time of the task is 2.1ms.
The light red bars between the point in time when the task first enters running state until the time the task exists running state, represent the time when the task (and VM0) has been “preempted” by another VM (VM1 and the HVR).

The timing properties of the task HighPriority are calculated accordingly.

- **Net Time:** “Real” execution time of the task while the corresponding VM (VM0) is active.
- **Gross Time:** “Virtual” execution time of the task, including all hypervisor timeslots where VM0 is inactive.

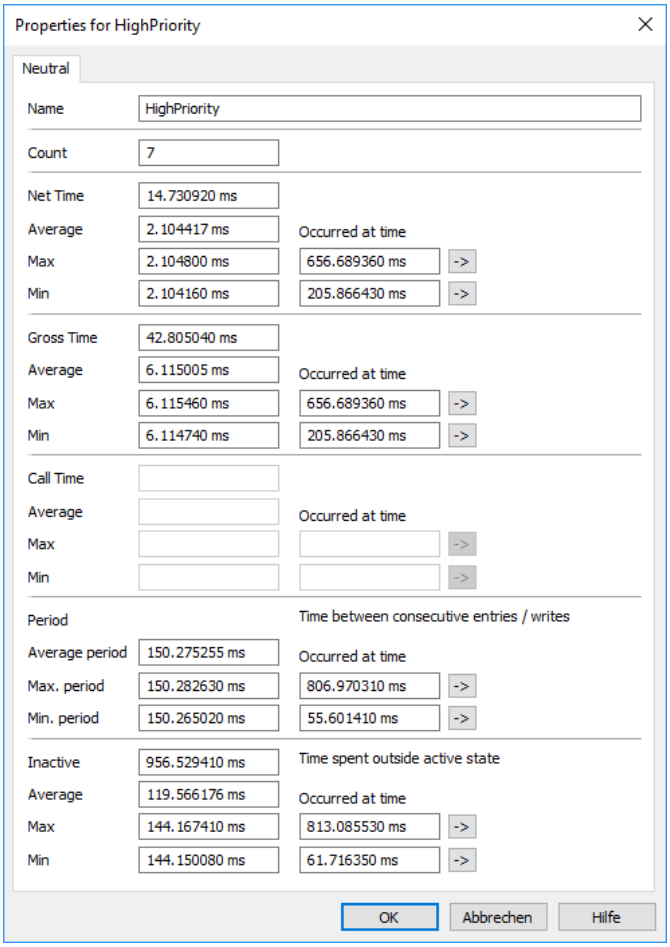

Figure 15: Sample Timing Properties of OS Task “HighPriority”

5 Technical Support

5.1 Online Resources

Online Help winIDEA and testIDEA online help	Knowledge Base Tips & tricks categorized by issue type and architecture	Tutorials From beginner to expert
Technical Notes How-tos for winIDEA functionalities with scripts	Application Notes How-to notes on advanced use-cases	Webinars Technical webinars about ISYSTEM tools with use cases

5.2 Contact

Please visit https://www.isystem.com/contact.html for contact details.