

TASKING[®]

***TASKING Multi Core
Performance Tool User Guide***

MA160-957 (v1.0r1) March 31, 2020

Copyright © 2020 TASKING BV.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of TASKING BV. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium[®], TASKING[®], and their respective logos are registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.

Table of Contents

Manual Purpose and Structure	v
1. Installing the Software	1
1.1. Installation for Windows	1
1.2. Licensing	1
1.2.1. Obtaining a License	3
1.2.2. Frequently Asked Questions (FAQ)	3
1.2.3. Installing a License	3
2. Introduction to the TASKING Multi Core Performance Tool	9
2.1. Core Load Analysis	11
2.2. RTOS Analysis	11
2.3. Trace Support	11
2.3.1. MCDS	13
2.3.2. miniMCDS	13
2.3.3. MCDS Light	13
3. Tutorial	15
3.1. Analyze a Project in TASKING Multi Core Performance Tool	15
3.2. Compare Results	23
3.3. Export Results	23
4. Effects on Profiling Analysis Results	25
4.1. Differences in Analysis Results Due to Compiler Optimizations	25
4.2. Effects of Interrupt Handlers on Interrupted Functions	25
4.2.1. Core Load Analysis	25
4.2.2. RTOS Analysis	26
5. Using the TASKING Multi Core Performance Tool	29
5.1. Run the Multi Core Performance Tool in Interactive Mode	29
5.2. Run the Multi Core Performance Tool from the Command Line	30
5.2.1. Command Line Tutorial	31
5.3. What to Do if Your Application Does not Start on a Board?	33
6. Reference	35
6.1. Settings Dialog	35
6.2. Summary Tab	36
6.2.1. Configuration	37
6.2.2. Info	37
6.2.3. Core Load	38
6.2.4. Core Load Per Period (only visible when there are multiple periods)	38
6.2.5. Top Activities	38
6.2.6. Top Activities Per Period (only visible when there are multiple periods)	39
6.3. Core Load Tab	40
6.4. Activities Tab	41
6.5. Raw Trace Data Tab	42

Manual Purpose and Structure

Manual Purpose

You should read this manual if you want to know:

- how to use the TASKING Multi Core Performance Tool
- the features of the TASKING Multi Core Performance Tool

Manual Structure

Chapter 1, *Installing the Software*

Explains how to install and license the TASKING Multi Core Performance Tool.

Chapter 2, *Introduction to the TASKING Multi Core Performance Tool*

Contains an introduction to the TASKING Multi Core Performance Tool and contains an overview of the features.

Chapter 3, *Tutorial*

Contains a step-by-step tutorial how to use the demo projects with the TASKING Multi Core Performance Tool.

Chapter 4, *Effects on Profiling Analysis Results*

Describes the differences in analysis results due to compiler optimizations and explains the effects of interrupt handlers on interrupted functions.

Chapter 5, *Using the TASKING Multi Core Performance Tool*

Explains how to use the TASKING Multi Core Performance Tool. You can run the TASKING Multi Core Performance Tool in two ways, via an interactive graphical user interface (GUI) or via the command line.

Chapter 6, *Reference*

Contains an overview of all the fields and columns in an analysis result output.

Related Publications

- Getting Started with the TASKING VX-toolset for TriCore
- TASKING VX-toolset for TriCore User Guide
- AURIX™ TC21x/TC22x/TC23x Family User's Manual, V1.1 [2014-12, Infineon]
- AURIX™ TC26x A-Step User's Manual, V1.1 [2013-12, Infineon]
- AURIX™ TC26x B-Step User's Manual, V1.2 [2014-02, Infineon]

TASKING Multi Core Performance Tool User Guide

- AURIX™ TC27x User's Manual, V1.4 [2013-11, Infineon]
- AURIX™ TC27x B-Step User's Manual, V1.4.1 [2014-02, Infineon]
- AURIX™ TC27x C-Step User's Manual, V2.2 [2014-12, Infineon]
- AURIX™ TC27x D-Step User's Manual, V2.2 [2014-12, Infineon]
- AURIX™ TC29x A-Step User's Manual, V1.1.1 [2014-01, Infineon]
- AURIX™ TC29x B-Step User's Manual, V1.3 [2014-12, Infineon]
- AURIX™ TC3xx Target Specification, V2.4 [2017-10, Infineon]
- AURIX™ TC3xx User's Manual, V1.4.0 [2019-12, Infineon]
- AURIX™ TC35x Appendix, V1.4.0 [2019-12, Infineon]
- AURIX™ TC37xEXT Appendix, V1.4.0 [2019-12, Infineon]
- AURIX™ TC38x Appendix, V1.4.0 [2019-12, Infineon]
- AURIX™ TC39x-B Appendix, V1.4.0 [2019-12, Infineon]

Chapter 1. Installing the Software

This chapter guides you through the installation process of the TASKING® Multi Core Performance Tool. It also describes how to license the software.

In this manual, **TASKING Multi Core Performance Tool** and **Multi Core Performance Tool** are used as synonyms.

1.1. Installation for Windows

System Requirements

Before installing, make sure the following minimum system requirements are met:

- 64-bit version of Windows 7 or higher
- 2 GHz Pentium class processor
- 4 GB memory
- 500 MB free hard disk space
- Screen resolution: 1024 x 768 or higher

Installation

1. If you received a download link, download the software and extract its contents.

- or -

If you received an USB flash drive, insert it into a free USB port on your computer.

2. Run the installation program (**setup.exe**).

The TASKING Setup dialog box appears.

3. Select a product and click on the **Install** button. If there is only one product, you can directly click on the **Install** button.
4. Follow the instructions that appear on your screen. During the installation you need to enter a license key, this is described in [Section 1.2, Licensing](#).

1.2. Licensing

TASKING products are protected with TASKING license management software (TLM). To use a TASKING product, you must install that product and install a license.

The following license types can be ordered from Altium.

Node-locked license

A node-locked license locks the software to one specific computer so you can use the product on that particular computer only.

For information about installing a node-locked license see [Section 1.2.3.2, *Installing Server Based Licenses \(Floating or Node-Locked\)*](#) and [Section 1.2.3.3, *Installing Client Based Licenses \(Node-Locked\)*](#).

Floating license

A floating license is a license located on a license server and can be used by multiple users on the network. Floating licenses allow you to share licenses among a group of users up to the number of users (seats) specified in the license.

For example, suppose 50 developers may use a client but only ten clients are running at any given time. In this scenario, you only require a ten seats floating license. When all ten licenses are in use, no other client instance can be used. Also a linger time is in place. This means that a license seat is locked for a period of time after a user has stopped using a client. The license seat is available again for other users when the linger time has finished.

For information about installing a floating license see [Section 1.2.3.2, *Installing Server Based Licenses \(Floating or Node-Locked\)*](#).

License service types

The license service type specifies the process used to validate the license. The following types are possible:

- **Client based** (also known as 'standalone'). The license is serviced by the client. All information necessary to service the license is available on the computer that executes the TASKING product. This license service type is available for node-locked licenses only.
- **Server based** (also known as 'network based'). The license is serviced by a separate license server program that runs either on your companies' network or runs in the cloud. This license service type is available for both node-locked licenses and floating licenses.

Licenses can be serviced by a cloud based license server called "**Remote TASKING License Server**". This is a license server that is operated by TASKING. Alternatively, you can install a license server program on your local network. Such a server is called a "**Local TASKING License Server**". You have to configure such a license server yourself. The installation of a local TASKING license server is not part of this manual. You can order it as a separate product (SW000089).

The benefit of using the Remote TASKING License Server is that product installation and configuration is simplified.

Unless you have an IT department that is proficient with the setup and configuration of licensing systems we recommend to use the facilities offered by the Remote TASKING License Server.

1.2.1. Obtaining a License

You need a license key when you install a TASKING product on a computer. If you have not received such a license key follow the steps below to obtain one. Otherwise, you cannot install the software.

Obtaining a server based license (floating or node-locked)

- Order a TASKING product from Altium or one of its distributors.

A license key will be sent to you by email or on paper.

If your node-locked server based license is not yet bound to a specific computer ID, the license server binds the license to the computer that first uses the license.

Obtaining a client based license (node-locked)

To use a TASKING product on one particular computer with a license file, Altium needs to know the computer ID that uniquely identifies your computer. You can do this with the **getcid** program that is available on the TASKING website. The detailed steps are explained below.

1. Download the **getcid** program from <http://www.tasking.com/support/tlm/downloads>.
2. Execute the **getcid** program on the computer on which you want to use a TASKING product. The tool has no options. For example,

```
C:\Tasking\getcid_version
Computer ID: 5Dzm-L9+Z-WFbO-aMkU-5Dzm-L9+Z-WFbO-aMkU-MDAy-Y2Zm
```

The computer ID is displayed on your screen.

3. Order a TASKING product from Altium or one of its distributors and supply the computer ID.

A license key and a license file will be sent to you by email or on paper.

When you have received your TASKING product, you are now ready to install it.

1.2.2. Frequently Asked Questions (FAQ)

If you have questions or encounter problems you can check the support page on the TASKING website.

<http://www.tasking.com/support/tlm/faqs>

This page contains answers to questions for the TASKING license management system TLM.

If your question is not there, please contact your nearest Altium Sales & Support Center or Value Added Reseller.

1.2.3. Installing a License

The license setup procedure is done by the installation program.

TASKING Multi Core Performance Tool User Guide

If the installation program can access the internet then you only need the licence key. Given the license key the installation program retrieves all required information from the remote TASKING license server. The install program sends the license key and the computer ID of the computer on which the installation program is running to the remote TASKING license server, no other data is transmitted.

If the installation program cannot access the internet the installation program asks you to enter the required information by hand. If you install a node-locked client based license you should have the license file at hand (see [Section 1.2.1, Obtaining a License](#)).

Floating licenses are always server based and node-locked licenses can be server based. All server based licenses are installed using the same procedure.

1.2.3.1. Configure the Firewall in your Network

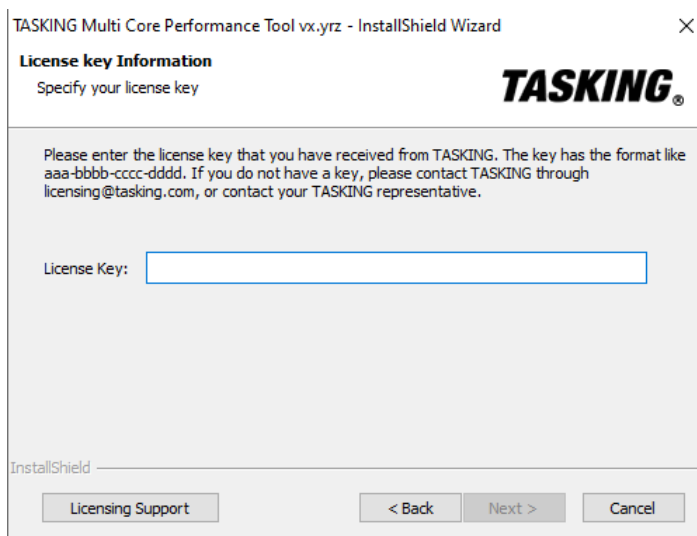
For using the TASKING license servers the TASKING license manager tries to connect to the Remote TASKING servers `lic1.tasking.com`, `lic2.tasking.com`, `lic3.tasking.com`, `lic4.tasking.com` at the TCP ports 8080, 8936 or 80. Make sure that the firewall in your network is transparently enabled for one of these ports.

1.2.3.2. Installing Server Based Licenses (Floating or Node-Locked)

If you do not have received your license key, read [Section 1.2.1, Obtaining a License](#) before you continue.

1. If you want to use a local license server, first install and run the local license server before you continue with step 2. You can order a local license server as a separate product (product code SW000089).
2. Install the TASKING product and follow the instruction that appear on your screen.

The installation program asks you to enter the license information.



The screenshot shows a window titled "TASKING Multi Core Performance Tool vx.yrz - InstallShield Wizard". The main heading is "License key Information" with the instruction "Specify your license key". The TASKING logo is prominently displayed. Below the heading, there is a text box containing the following instructions: "Please enter the license key that you have received from TASKING. The key has the format like aaa-bbbb-cccc-dddd. If you do not have a key, please contact TASKING through licensing@tasking.com, or contact your TASKING representative." A text input field labeled "License Key:" is provided for the user to enter the key. At the bottom of the window, there are three buttons: "Licensing Support", "< Back", and "Next >" (disabled), and a "Cancel" button.

3. In the **License Key** field enter the license key you have received from Altium and click **Next** to continue.

*The installation program tries to retrieve the license information from a remote TASKING license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*

4. Select your **License Type** and click **Next** to continue. If the license type is already filled in and grayed out, you can just click **Next** to continue.

You can find the license type in the email or paper that contains the license key.

5. (For floating licenses only) Select **Remote TASKING license server** to use one of the remote TASKING license servers, or select **Local TASKING license server** for a local license server. The latter requires optional software.

(For local license server only) specify the **Server name** and **Server port** of the local license server.

Note that a Node-locked server based license always uses the Remote TASKING license server.

6. Click **Next** and follow the rest of the instructions to complete the installation.

1.2.3.3. Installing Client Based Licenses (Node-Locked)

If you do not have received your license key and license file, read [Section 1.2.1, Obtaining a License](#) before continuing.

1. Install the TASKING product and follow the instruction that appear on your screen.

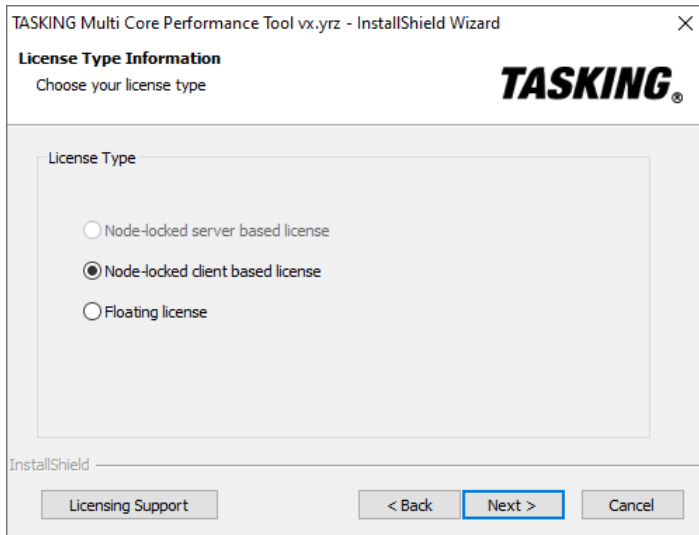
The installation program asks you to enter the license information.

The screenshot shows a dialog box titled "TASKING Multi Core Performance Tool vx.yrz - InstallShield Wizard". The main heading is "License key Information" with the instruction "Specify your license key". The TASKING logo is displayed in the top right. Below the heading, there is a text box containing the following instructions: "Please enter the license key that you have received from TASKING. The key has the format like aaa-bbbb-cccc-dddd. If you do not have a key, please contact TASKING through licensing@tasking.com, or contact your TASKING representative." Below this text is a text input field labeled "License Key:". At the bottom of the dialog, there are four buttons: "Licensing Support", "< Back", "Next >", and "Cancel".

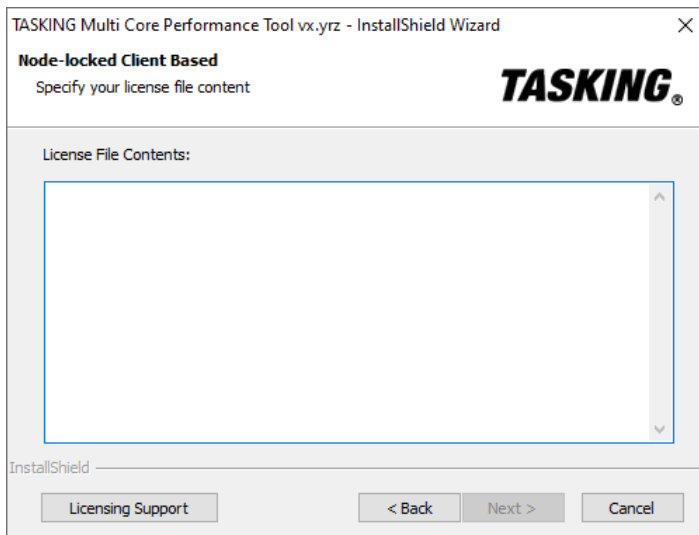
TASKING Multi Core Performance Tool User Guide

- In the **License Key** field enter the license key you have received from Altium and click **Next** to continue.

*The installation program tries to retrieve the license information from a remote TASKING license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*



- Select **Node-locked client based license** and click **Next** to continue.



4. In the **License File Contents** field enter the contents of the license file you have received from Altium.

The license data is stored in the file licfile.txt in the etc directory of the product (<install_dir>\etc).

5. Click **Next** and follow the rest of the instructions to complete the installation.

Chapter 2. Introduction to the TASKING Multi Core Performance Tool

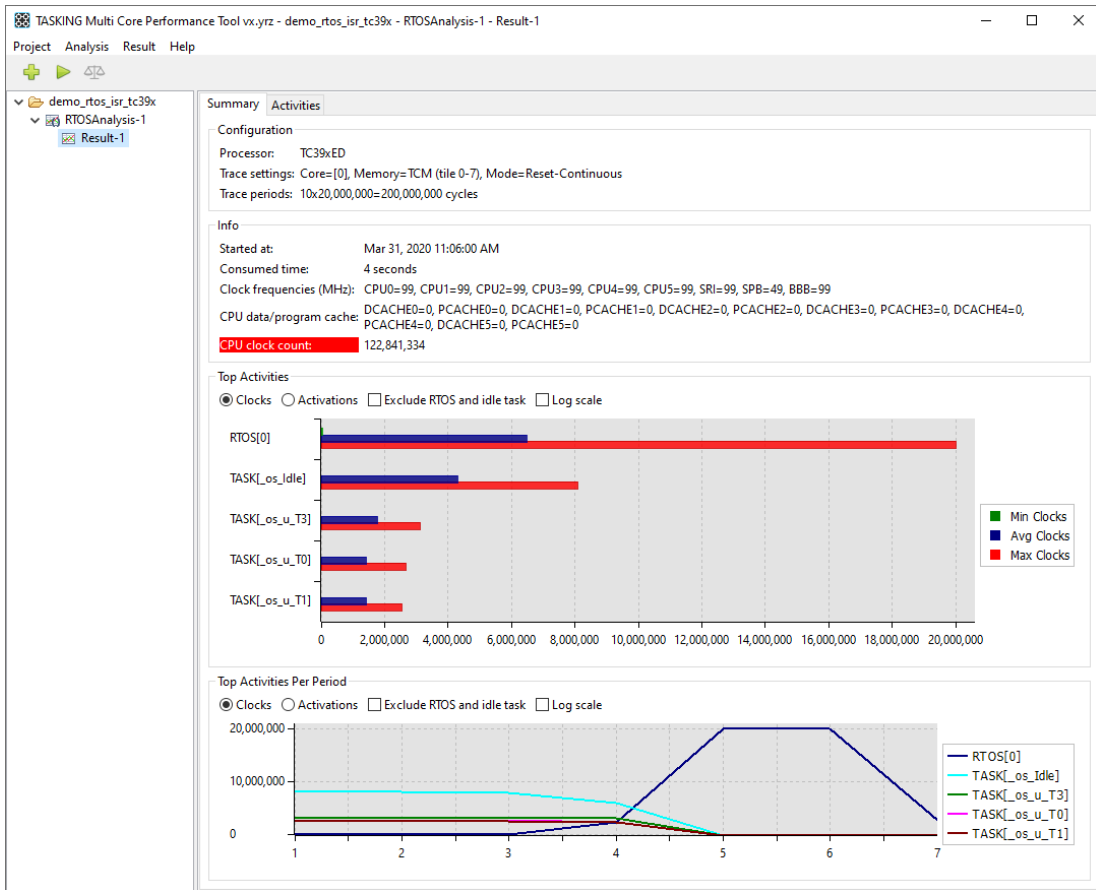
The embedded computer market and especially the safety critical embedded computer market is in the process to completely switch from single-core systems to multi-core systems. In parallel also the amount of functionality on these systems is growing enormously. As a result the performance gain of a multi-core system is already consumed by most of the software and applications that have to run on such systems. One critical hurdle to take is the architectural design which defines which software/application will execute on which core. There is no other way to accurately measure up-front the behavior and interaction of all software running on a multi-core device than executing on real hardware and having a tool that is able to display a real picture and pin points to corrective actions.

The TASKING Multi Core Performance Tool can be used by software architects to define a well leveraged multi-core setup. It can be used by function architects to prove and analyze the given load on each core is kept and it can be used by system testers to continuously have a clear status of the good behavior of the system during stress test conditions.

The TASKING Multi Core Performance Tool communicates with the selected central processor unit (CPU) to gather real-time data of the target and generates a representation of the load of each core in a graphical user interface (GUI). The load of each core is displayed in the GUI in a graphical and numerical way. In the GUI you can dig deeper by selecting a core and get the load view for each process running on the selected target. The minimal, maximal and average load for each core, process and function is made visible.

If you see that one of the cores is too busy, you could check if another core has some idle time to spare and you can relocate some of the load to another core.

TASKING Multi Core Performance Tool User Guide



Features of the TASKING Multi Core Performance Tool

- Core load analysis
- Real-Time Operating System (RTOS) analysis
- Compare analysis runs of the same kind
- Organize analyses and results in projects
- Load/store analysis results
- Graphical user interface (GUI) and command line interface (CLI) support
- Support for the latest Device Access Server driver (DAS, see www.infineon.com/das) and support for all Device Access Port (DAP) miniWigglers that are supported by the DAS drivers. All debug interfaces supported by the Infineon miniWiggler used can also be used to connect to the target hardware. This can be 10-pin DAP / 20-pin Automotive JTAG connector or 10-pin DAP / 16-pin OCDL1, depending on which miniWiggler version is used.

- Export analysis results to comma separated values (CSV) files

2.1. Core Load Analysis

This type of analysis collects data to visualize the minimum, average and maximum load for each core of the selected processor. It measures the time in the given idle function(s) and or range(s) for the given time period(s). The idle function is a function that will always run with the lowest possible priority. This means that if the program is very busy the less time is spent in the idle function. Core Load Analysis shows if you can better off-load some functionality to another core. If the analysis shows 100% for a core that means that there was no time spent in the idle function. So, if the core load analysis shows less than 100%, that means that there is time left to add functionality. Time in the interrupt handler is not added to the idle time, this also means that if for some reason a function has been selected as idle while it is in the interrupt handler, the clocks never go to the idle function.

2.2. RTOS Analysis

This type of analysis traces the load of Real-Time Operating System (RTOS) activities. The RTOSes that are supported by the Multi Core Performance Tool are statically configured.

A global variable containing something recognizable as the 'current task' can and will be used to determine the current task. It is also used to make an accumulated task load. To determine individual task runs the task starting point is taken into account as well. This is all done by the Multi Core Performance Tool by selecting which RTOS you want to use.

In operation systems that can schedule a process to run on any of the cores, determining the process load may not be possible due to the trace hardware not being able to trace all cores at the same time. Thread ID and thread start address are used to indicate individual runs of a thread to be able to calculate minimum, average and maximum thread processor load values.

2.3. Trace Support

The standard TriCore/AURIX™ processors (production devices) lack debug trace functionality. However, this functionality is very useful when you develop and test your application.

The TASKING Multi Core Performance Tool uses the Multi-Core Debug Solution (MCDS) for on-chip trace support. The following table shows the devices that are supported by the TASKING Multi Core Performance Tool.

Device	MCDS type	Trace memory	Number of cores that can be traced at the same time
TC23xED	MCDS	TCM / XTM	1
TC26xED	MCDS	TCM / XTM	2
TC27xED	MCDS	TCM	2
TC29x	miniMCDS	TRAM	1
TC29xED	MCDS	TCM / XTM	2
TC35x	MCDSLigh	TCM / XTM	2
TC37xEXT	MCDS	TCM / XTM	3 (of which one must be core 0)
TC38x	miniMCDS	TRAM	1
TC39xAED	MCDS	TCM / XTM	2
TC39xED	MCDS	TCM / XTM	3 (of which one must be core 0)

Naming convention

You can see by the name on the processor what type of device it is. For example, with SAK-TC299TE the last letter indicates the "Feature Package". If this letter is an 'E' or 'F' you have an Emulation Device. For the TC3xx devices, if the "Feature Package" letter is an 'E' you have an Emulation Device.

For a detailed naming convention see the Infineon website:

- [AURIX™ Product Naming](#)

Trace memory

Trace information is stored in a dedicated trace buffer. With an Emulation Device (ED) you can allocate part of the Emulation Memory (EMEM) as trace buffer memory. The Emulation Memory is divided in RAM blocks, the so-called 'tiles', which can be used as Calibration or Trace memory. These memory tiles consists of TCM, XCM and XTM. TCM (Trace Calibration Memory) can be used for Trace memory or Calibration, XCM (Extended Calibration Memory) can only be used for Calibration memory and XTM (Extended Trace Memory) can only be used for Trace memory.

Production Devices that are equipped with miniMCDS use TRAM for trace memory.

Which trace memory you can select depends on the selected processor.

Tile memory range

For TCM, you can choose which part of the Emulation Memory should be used for tracing. For XTM always both tiles are used for tracing.

Be careful that the same tile memory range used for tracing is not used by the target application, as this can lead to unexpected trace results. The number of tiles vary per Emulation Device.

Trace mode

When you run a trace analysis in the TASKING Multi Core Performance Tool, you can set the trace mode:

- **One shot mode.** In this mode the analysis will run until the trace buffer is full, or when the application finishes or when you stop the analysis manually. This is non-intrusive, meaning that the trace does not interfere the running processor. After the trace has stopped the Multi Core Performance Tool reads the collected data.
- **Continuous trace.** In this mode the analysis will run until the application finishes or when you stop the analysis manually. This mode is intrusive, meaning that the processor is stopped temporarily every time the trace buffer has been filled, so that the Multi Core Performance Tool can read the collected data. After that the processor continues execution and continues writing to the trace buffer.

2.3.1. MCDS

MCDS (Multi-Core Debug Solution) uses ED (Emulation Devices). An Emulation Device has an Emulation Extension Chip (EEC) added to the same silicon, which is accessible through the JTAG or DAP interface (EEC is only for devices that have tile range supports like MCDS and MCDSLigh.). The TASKING Multi Core Performance Tool supports the on-chip trace feature of the Emulation Device.

The MCDS has support for a maximum of three processor observation blocks (POBs). This means that, depending on the device, up to three cores can be traced at once. For the hardware that can trace three cores at the same time one of the three cores must be core 0. For detailed information about MCDS we recommend that you read the processor documentation.

2.3.2. miniMCDS

Some devices that do not have MCDS come with miniMCDS. miniMCDS is a subset of the on-chip trace feature that is available on Emulation Devices. The mini-MCDS memory is not suitable for safety related data and must not be used for data storage by safety applications.

The miniMCDS devices have no EEC and therefore only TRAM memory. These kind of devices have only one POB, this means that only one core can be traced at the same time. See the processor documentation for detailed information about the device.

2.3.3. MCDS Light

In AURIX 2G devices a new type of Multi-Core Debug Solution is introduced, called MCDS Light. MCDS Light is a subset of MCDS. The difference with MCDS is that MCDS Light has a maximum of two POBs, which means that a maximum of two cores can be traced at once.

One of the advantages of MCDS Light over miniMCDS is that it comes with EMEM and thus is not limited to the very small TRAM memory.

Chapter 3. Tutorial

The `mcptool\tutorials` directory of the TASKING Multi Core Performance Tool installation contains several examples. They serve as a good starting point for your own profiling analysis project. All examples are present for the TC29xB and the TC39xB.

- `demo_rtos_isr` - A single task RTOS project causing multiple interrupts of both ISR category 1 and 2.
- `demo_rtos_leds_tb` - A multi task RTOS project. The tasks are driven by different timer intervals, each task blinking a LED.

All examples come with TASKING Multi Core Performance Tool projects (files with the `.McpTool` extension), with pre-run analyses. You can open a project in the TASKING Multi Core Performance Tool to inspect the RTOS analysis results, without having to run the examples on a target board.

All examples also contain an ELF file (`.elf`) that you can use with the TASKING Multi Core Performance Tool to flash the example application on a target board.

In this tutorial we will use the `demo_rtos_isr` example for the TC39xB to go through the process of preparing your project from scratch and running an RTOS analysis. After this tutorial you can use the other tutorials yourself in a similar way.

3.1. Analyze a Project in TASKING Multi Core Performance Tool

This section describes how to analyze a demo project.

Unzip the example project

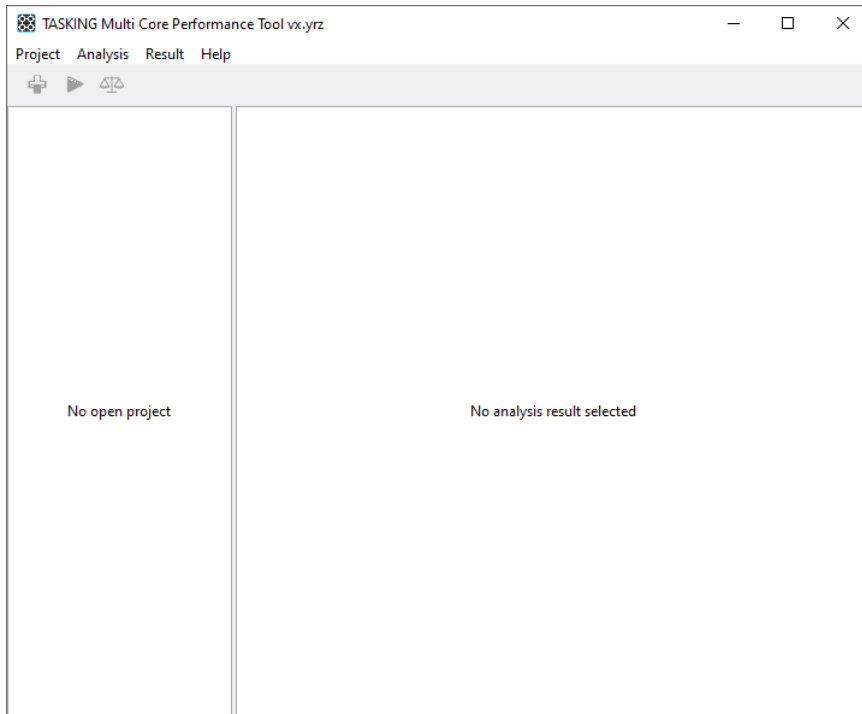
Since the tutorials in `mcptool\tutorials` are in Zip files, you need to extract them first to a location of your choice. You can use the native Windows Explorer or use your favorite unzip tool.

- Unzip the file `mcptool\tutorials\demo_rtos_isr_tc39x.zip` to a directory of your choice. From the Windows Explorer you can right-click on the Zip file and select **Extract All...**, select the destination and click **Extract**.

In this tutorial we assume that the extracted files are in
`C:\Users\name\workspace_mcptool\demo_rtos_isr_tc39x`.

Create a project

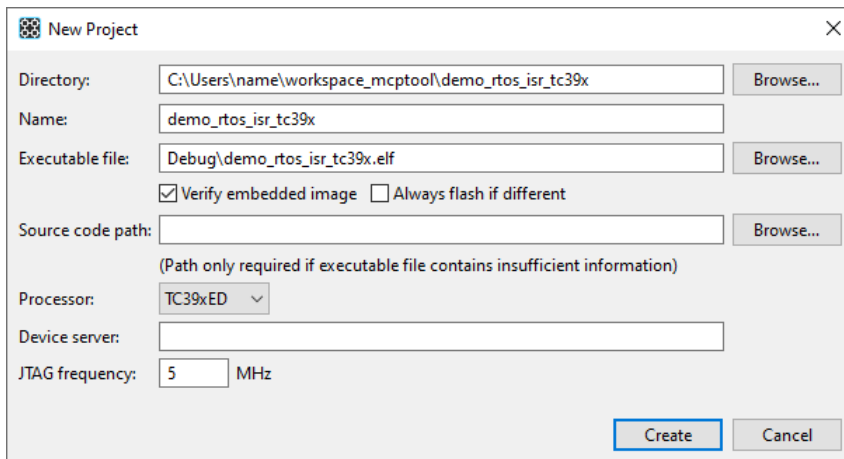
1. Start the TASKING Multi Core Performance Tool.



The TASKING Multi Core Performance Tool window is divided into two panes. The left pane is reserved for the project tree and the right pane is reserved for analysis results.


2. From the **Project** menu, select **New Project**.

The New Project dialog appears.



3. In the **Directory** field, specify the directory where you want to store the Multi Core Performance Tool project file (file with extension `.McpTool`).
4. In the **Name** field, enter the name of the project, for example `demo_rtos_isr_tc39x`. By default, the name is derived from the selected directory, but you can change it.
5. In the **Executable file** field, specify the name of the ELF file. For standard TASKING projects this file is usually in the Debug directory relative to the project directory. If the executable file is stored in another directory, the full path name is shown.
6. Enable **Verify embedded image** to compare the contents of the flash image to the ELF file before a run is started. If there is a difference you are asked if the flash should be updated before the run starts, unless you also enable **Always flash if different**.
7. Optionally specify a **Source code path** (a semi-colon separated directory list). Normally, the location of the source files is taken from the ELF file. This field has no relevance for RTOS projects.
8. Select the **Processor**. For example, `TC39xED`.
9. For the **Device server**, enter the name or address of a remote PC that the TriCore hardware is connected to (leave blank for `localhost`). See also the DAS documentation.
10. Optionally specify the **JTAG frequency** in MHz.
11. Leave the rest of the dialog as is and click **Create**.

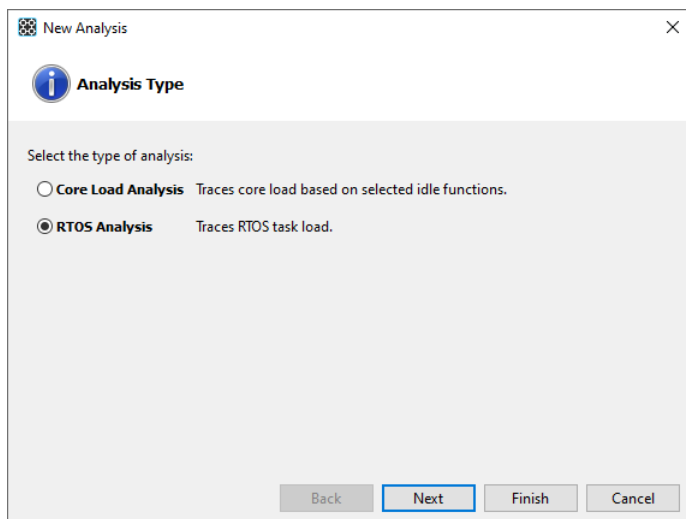
The new project is created and opened.

 `demo_rtos_isr_tc39x`

Create an RTOS analysis

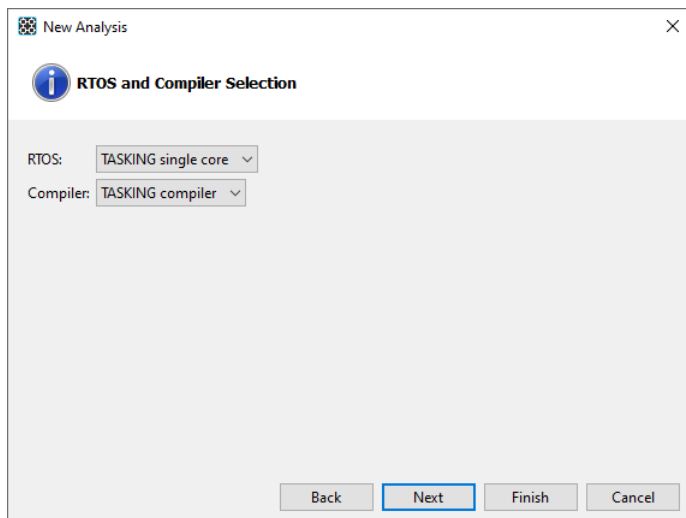
1. From the **Analysis** menu, select **New Analysis**.

The New Analysis wizard appears.



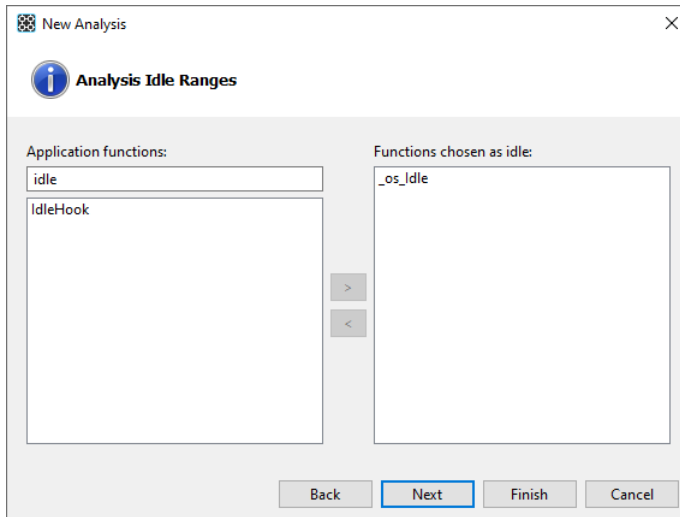
2. Two types of analyses are possible. Select **RTOS Analysis** and click **Next**.

The RTOS and Compiler Selection page appears.



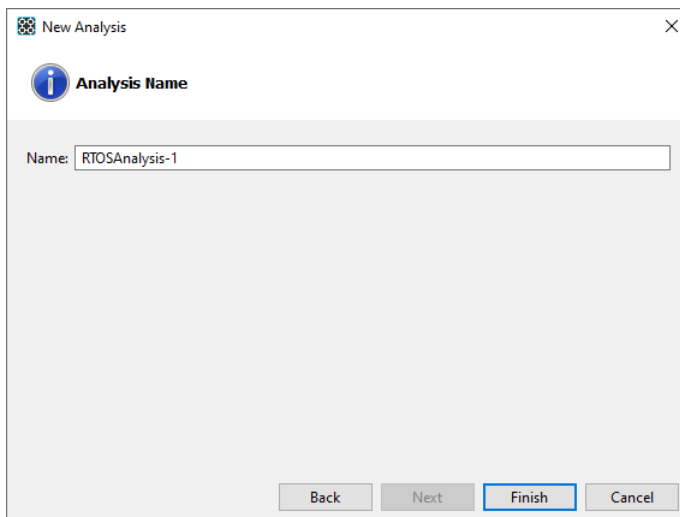
3. Select the **RTOS** used in your project. Several RTOSes are supported, for this tutorial select **TASKING single core**.
4. Select the **Compiler** used to build your project. For this tutorial select **TASKING compiler** and click **Next**.

The Analysis Idle Ranges page appears.



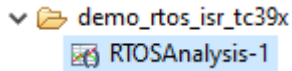
5. Select one or more **Application functions** that serve as the idle routine and click >. In the **Filter** field you can enter a part of the function name to reduce the list. When you have chosen an idle function, the result summary can filter out the data for this item. In this example, select **_os_Idle**.
6. Click **Next**.

The Analysis Name page appears.



7. Specify the analysis name. A default name has already been filled in based on the analysis type and a sequence number, but you can specify your own name.
8. Click **Finish**.

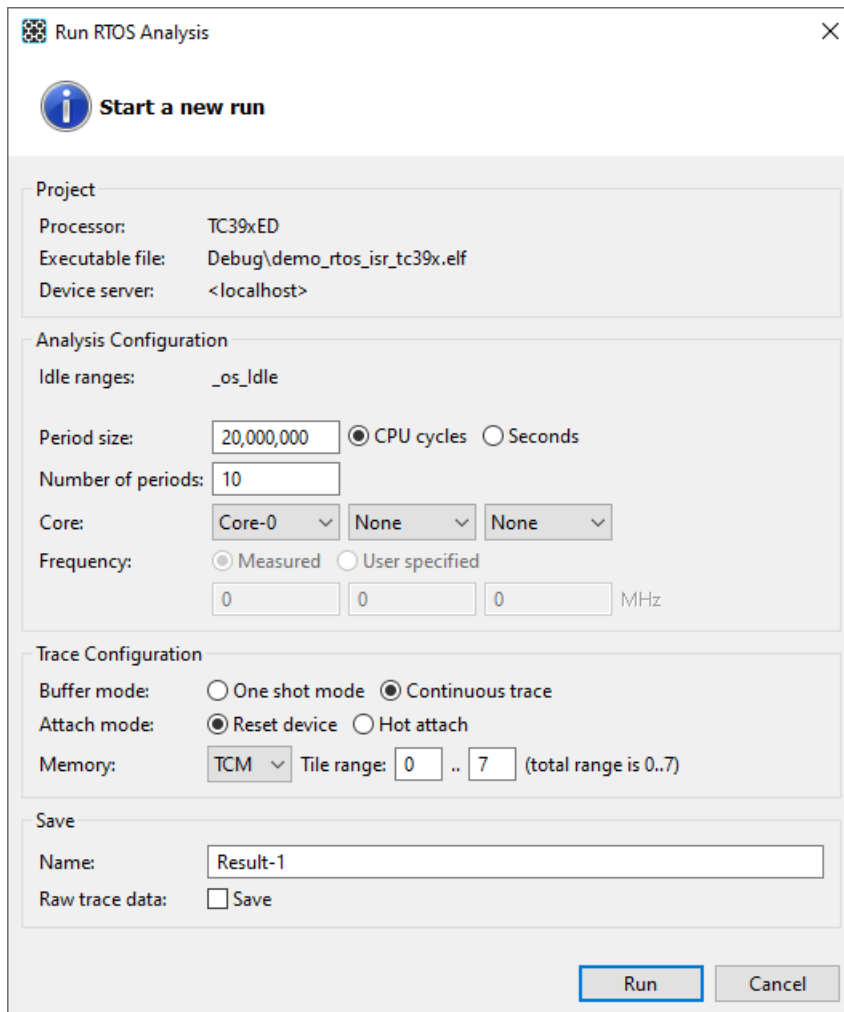
The new analysis is created and is visible in the project tree.



Run the analysis

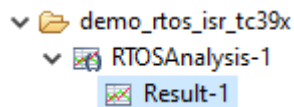
1. In the project tree select the analysis you want to run.
2. From the **Analysis** menu, select **Run Analysis**.

The *Run Analysis dialog* appears.



3. Enter a **Period size** in **CPU cycles** or **Seconds** and the **Number of periods** that is suitable for your application. After each period information is collected.
4. In the **Core** field, select the TriCore core(s) for which you want to run the analysis.
5. Optionally, specify the **User specified core Frequency** in MHz. **Measured** means that the previously measured frequency is used. This field is only visible when you have a period size in seconds.
6. Select a trace **Buffer mode**. A **One shot mode** trace ends when the hardware trace buffer is full, or when the application finishes or when you stop the analysis manually. A **Continuous trace** ends when the application finishes or when you stop the analysis manually. This mode is intrusive, meaning that the processor is stopped temporarily every time the trace buffer has been filled, so that the Multi Core Performance Tool can read the collected data. After that the processor continues execution and continues writing to the trace buffer. Enable **Continuous trace** when the amount of trace data exceeds the processor internal trace memory. An indication of this may be that the trace run ends before all specified periods are captured. An other possible cause of not receiving all periods can be that the program finished, as is the case with the `demo_rtos_isr` example project.
7. Select an **Attach mode**. With **Reset device**, tracing starts by running the program in the embedded device from the reset vector. With **Hot attach**, tracing starts by continuing tracing from the current program counter location.
8. Select the type of **Memory** that should be used for tracing, **TCM** (Trace Calibration Memory) or **XTM** (Extended Trace Memory). Production Devices that are equipped with mini-MCDS always use TRAM for trace memory.
9. For trace calibration memory (TCM) on emulation devices only, enter a **Trace memory tile range**. Trace calibration memory (TCM) of emulation devices consists of a consecutive number of tiles. Select the first and last tile index you want to use for trace memory.
10. Enter an analysis result **Name** (default `Result-` and a sequence number).
11. Optionally enable **Raw trace data**. Raw trace data is for advanced users who want to examine program flow. Raw trace data is useful, for example, to see why stall cycles are assigned to instructions that do not access memory. If you enable this option, an extra **Raw Trace Data** tab appears in the analysis result.
12. Click **Run**.

The default setting is that before a run is started the contents of the flash is compared to the ELF file and you are asked if the flash should be updated before the run starts. You can change this in the project settings. When the ELF file is flashed, the analysis starts. After the analysis is finished the result is present in the project tree.

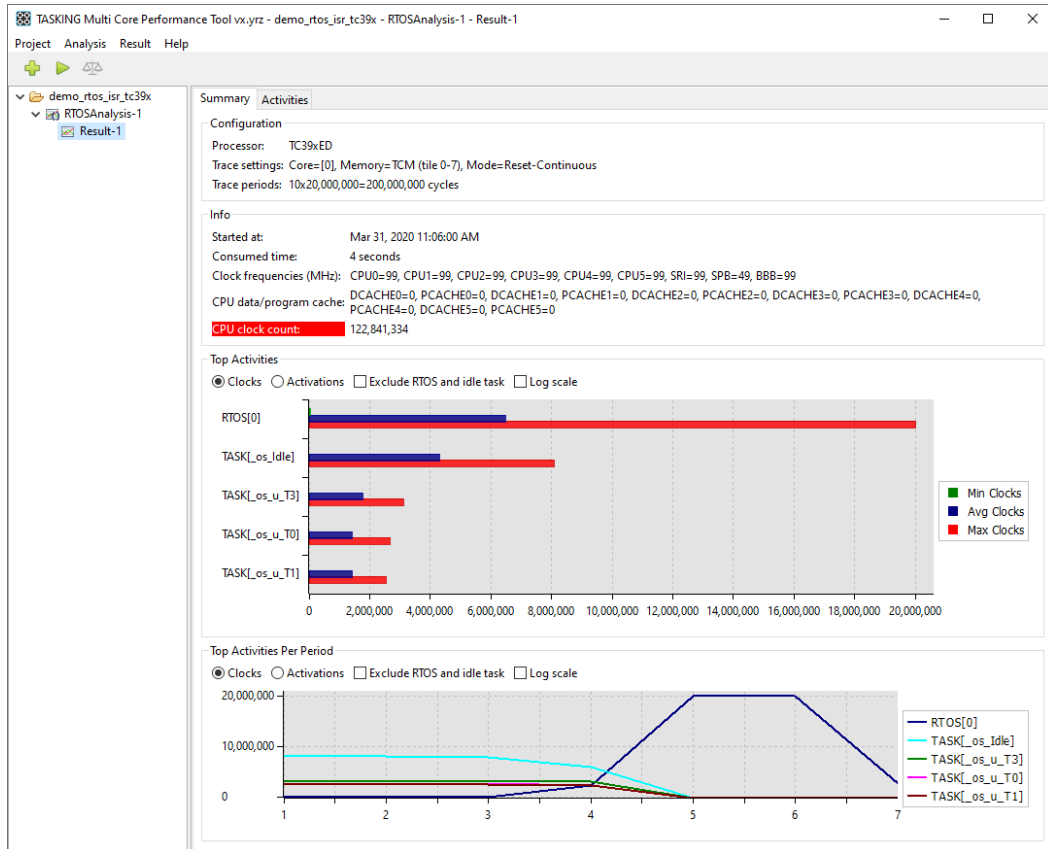


Inspect the result of the RTOS analysis

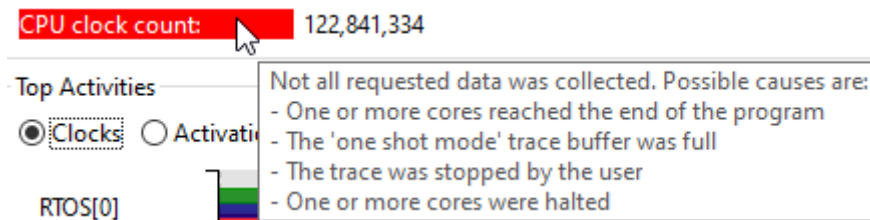
1. In the project tree select the result you want to inspect (`RTOSAnalysis-1`, `Result-1`).

TASKING Multi Core Performance Tool User Guide

The result appears in several tabs. The summary shows the top activities, this can be on clock cycles or on number of activations.



- On the **Summary** tab, notice the red **CPU clock count**. If you hover the mouse over it, you get a popup with additional information.



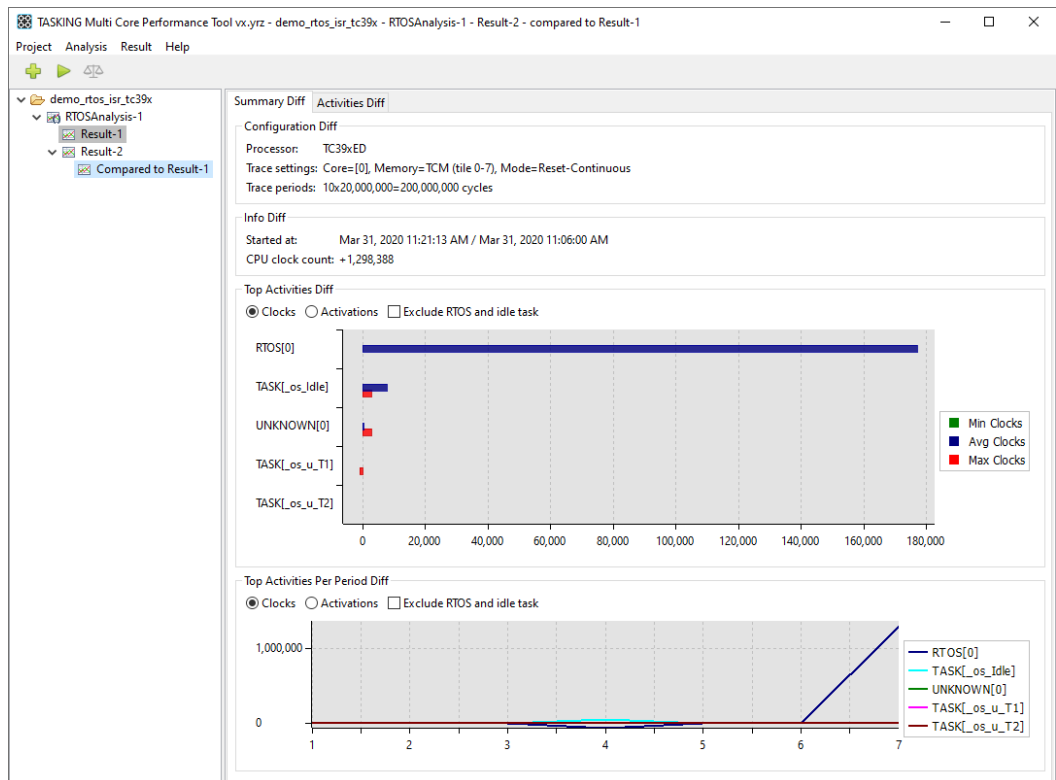
See [Chapter 6, Reference](#) for detailed information of each tab.

3.2. Compare Results

The Multi Core Performance Tool has a feature to compare results. This is very useful to see the differences before and after a fix. Note that you can only compare results from the same analysis.

1. In the Multi Core Performance Tool, select a result. For example, when you have a `Result-2` of `RTOSAnalysis-1`.
2. From the **Result** menu, select **Compare Results**.
3. Select another result, for example `Result-1`. The results you can select are marked yellow.

The comparison starts and a difference report is created. The numbers in the report are calculated as the "first selected result" minus the "second selected result".



3.3. Export Results

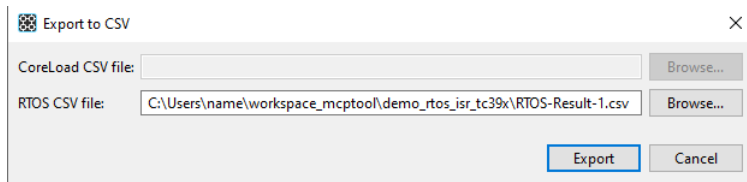
You can export analysis results and comparison results to comma separated values (CSV) files. You can choose to export instructions, functions or memory depending on the analysis type.

1. In the Multi Core Performance Tool, select a result. For example, `Result-1` of `RTOSAnalysis-1`.

TASKING Multi Core Performance Tool User Guide

- From the **Result** menu, select **Export Result to CSV**.

The Export to CSV dialog appears.



- Enter the filename(s) and click **Export**.

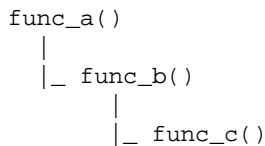
Chapter 4. Effects on Profiling Analysis Results

This chapter describes the differences in analysis results due to compiler optimizations and explains the effects of interrupt handlers on interrupted functions.

4.1. Differences in Analysis Results Due to Compiler Optimizations

Analysis results of the same kind may have different cycle counts. This can happen due to the tail call optimization of the C compiler, which is part of the peephole optimization of the C compiler. This optimization is enabled by default for the TASKING VX-toolset for TriCore. This optimization causes a leaf function that is called at the last line of a function to not show up in the analysis result while the function's code is executed. The reason for this is that the leaf function is entered with a jump instruction and the leaf function's return instruction performs the return that the calling function would have done.

Without the tail call optimization, the normal function flow is: `func_a()` calls `func_b()` which calls `func_c()`. `func_c()` returns to `func_b()` which returns to `func_a()`.



With tail call optimization, the function flow becomes: `func_a()` jumps to `func_b()` which jumps to `func_c()`. `func_c()` returns to the function that called `func_a()`.

Because of this optimization (jumps instead of calls), less return statements are needed and therefore less clock cycles.

4.2. Effects of Interrupt Handlers on Interrupted Functions

MCDS supports instruction tracing, flow tracing and function call tracing. The TASKING Multi Core Performance Tool uses flow tracing to gather the data needed for the analyses (RTOS and Core Load).

4.2.1. Core Load Analysis

For Core load analysis clocks in the interrupt handling are not added to the idle time, this also means that if for some reason a function has been selected as idle while it is in the interrupt handler the clocks never go to the idle function.

4.2.2. RTOS Analysis

The following items are of interest:

- Tasks
- Interrupt Service Routines category 1 (ISR1)
- Interrupt Service Routines category 2 (ISR2)
- RTOS code
- System Start-up and Shutdown

OSEK defines three processing levels:

- Interrupt level
- Logical level for scheduler
- Task level

Depending on the implementation interrupt handlers can be called directly from the interrupt router or via a generic interrupt handler that calls the specific handler functions from an internal handlers table. OSEK defines two categories of ISRs, ISRs of category 2 are allowed to call a limited set of operating system services which in turn may cause switching of tasks.

4.2.2.1. Tasks

Tasks are started by the operation system and can be suspended in waiting for events and or data. Task termination is always performed via the `TerminateTask` and `ChainTask` functions.

When tasks are waiting for events and or messages in the mean time other tasks may be running. These tasks may also end up waiting for events and or messages therefor just looking at the addresses executed by the processor will not be enough to determine which is the current running task e.g. they may be interrupted before getting back into task specific code therefore tracing of more than just the executed addresses is required.

Determining what is the current task is done by looking at something maintained by the operating system. This may be something like a current task ID or a current task pointer.

The MCDS has functionality to trace memory access (including values, this can also be traced with `miniMCDS` and `MCDSlight`). Enabling logging of memory access to specific memory addresses including values gives the Multi Core Performance Tool knowledge about current active task for cycle accounting.

4.2.2.2. Interrupt Service Routines Category 1

Interrupts of category 1 do not influence task management. As stated in the introduction there may be a generic handler that calls the specific interrupt handlers. This results in one interrupt handler with children for each of the called interrupt functions. When this generic interrupt handler and its children are needed to be shown separately, extra processing is required.

4.2.2.3. Interrupt Service Routines Category 2

Interrupts of category 2 can influence task management and may contain calls to some of the system services. Rescheduling is normally performed at the return from this category of interrupt service routines.

4.2.2.4. RTOS Code

This is code that handles for instance context switching, looking for rescheduling, handling of timers, clean up after task termination and such. This is a kind of overhead that is preferably not added to a specific task but to the generic operating system execution.

4.2.2.5. System Start-up and Shutdown

Lots of things happen before an actual program or operating system becomes active, generic processor initialization, memory configuration, clock configuration, global variables initialization, etc. In a TASKING environment these are handled/initiated by the cstart code.

4.2.2.6. Software Traps

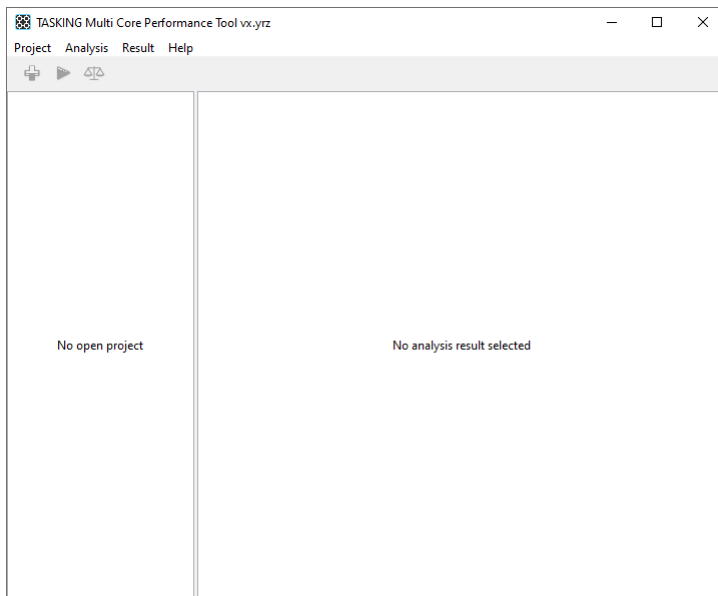
Software traps are generated as an intentional result of executing a system call or an assertion instruction. The supported assertion instructions are TRAPV (Trap on overflow) and TRAPSV (Trap on sticky overflow). System calls are generated by the SYSCALL instruction. These traps run at interrupt level and will thus require extra processing. The TASKING RTOS makes extensive use of syscalls. In the TASKING RTOS the use of syscall depends on the use of the RTOS timer and the definition of interrupts of category 2 in the RTOS configuration.

Chapter 5. Using the TASKING Multi Core Performance Tool

You can run the TASKING Multi Core Performance Tool in two ways, via an interactive graphical user interface (GUI) or via the command line. The GUI variant is useful in showing graphical analysis results with hints how to improve the code. The command line interface is useful in automated scripts and makefiles to generate analysis results in comma separated values (CSV) files.

5.1. Run the Multi Core Performance Tool in Interactive Mode

To start the Multi Core Performance Tool select **Multi Core Performance Tool** from the Windows **Start** menu. The program starts with an empty window except for a menu bar and a toolbar at the top. The area below that consists of two panes. The left pane is used to display a project tree, with a project name, one or more analysis names and one or more result names. The right pane is used to display an analysis result. You can resize a pane by dragging one of its four corners and you can move a pane by dragging its title. You can drag the button toolbar to another place, for example vertically to the left side or even detach it from the main window.



Normal project management is available. You can create, open, edit, close or delete a project. A project filename will have the extension `.McpTool`.

The steps to:

- create a project

- create an analysis
- run an analysis

are described in [Section 3.1, Analyze a Project in TASKING Multi Core Performance Tool](#).

See also [Section 3.2, Compare Results](#) and [Section 3.3, Export Results](#). For details about the Results see [Chapter 6, Reference](#).

5.2. Run the Multi Core Performance Tool from the Command Line

To run the Multi Core Performance Tool from the command line use the **McpToolCmd** batch file in a Windows Command Prompt. Enter the following command to see the usage:

```
McpToolCmd --help
```

The general invocation syntax is:

```
McpToolCmd options project.McpTool
```

where, *project.McpTool* refers to an existing Multi Core Performance Tool project file.

The following *options* are available:

Option	Description
-? / --help	This option causes the program to display an overview of all command line options.
--compare=result -mresult	This option allows you to compare the results of a run with another result. You must specify the name of an existing reference result. Option --run should be used together with this option.
--continuous -c	This option allows you to run the analysis in continuous trace mode. Without this option, the default is one shot mode.
--core=core-nr	This option allows you to specify the core index number. Without this option, the default is core 0. To select multiple cores repeat this option.
--frequency=frequency	This option allows you to specify the core <i>frequency</i> in Mhz. Without this option, the previously measured frequency is used.
--jtag=speed -jspeed	This option allows you to set the JTAG <i>speed</i> in MHz.
--memorytype=type -ttype	This option allows you to specify the trace memory type. <i>type</i> can be TCM, XTM or TRAM.
--periodclocks=clocks	This option allows you to specify the duration of the trace in number of CPU <i>clocks</i> .
--periodcount=count	This option allows you to set the number of periods that the analysis should run.

Option	Description
--periodseconds=seconds	This option allows you to specify the duration of the trace in <i>seconds</i> .
--run=analysis -ranalysis	This option allows you to run an existing analysis.
--server=hostname -shostname	This option allows you to specify the device server name. Without this option, the default is <code>localhost</code> .
--tilerange=from-to -xfrom-to	This option allows you to specify the tile memory range for the TCM memory type.
--version -v	This option shows the program version header.

To run an existing analysis

Use the following syntax to run an existing analysis from the command line:

```
McpToolCmd --run=analysis project.McpTool
```

where, `project.McpTool` refers to an existing Multi Core Performance Tool project file.

To run and compare an existing analysis

Use the following syntax to run an existing analysis and compare the results with a previous result from the command line:

```
McpToolCmd --run=analysis --compare=result project.McpTool
```

where, `project.McpTool` refers to an existing Multi Core Performance Tool project file.

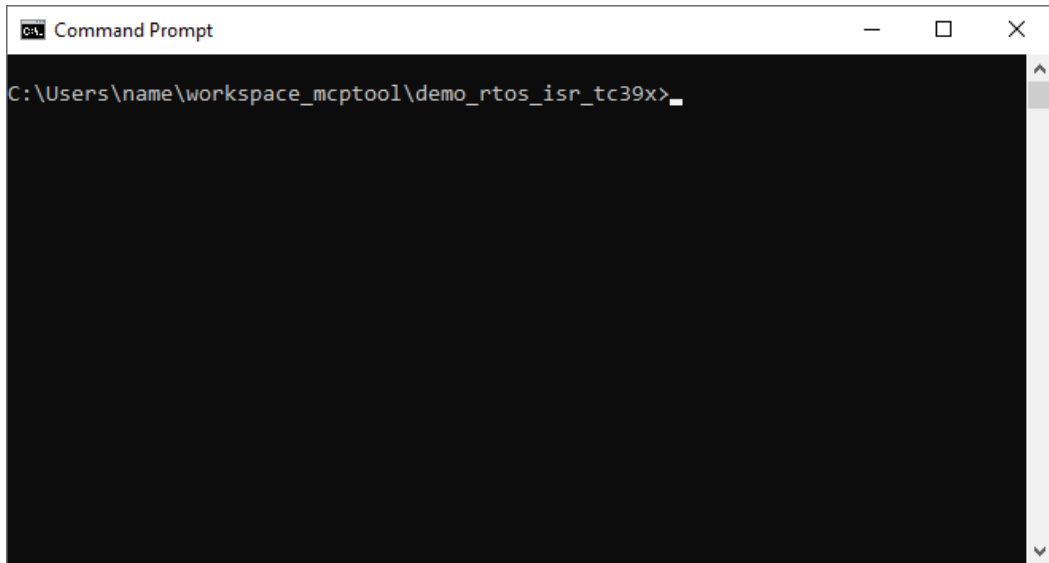
5.2.1. Command Line Tutorial

In this section we use tutorial `demo_rtos_isr_tc39x` with the delivered `demo_rtos_isr_tc39x.McpTool` to illustrate the use of the command line options of the Multi Core Performance Tool.

Prepare command line

Before you run the Multi Core Performance Tool from the command line, follow these steps to configure the Windows command prompt.

1. Start the Windows Command Prompt and go to the workspace directory containing the tutorial `demo_rtos_isr_tc39x`.



2. Add the executable directory of the Multi Core Performance Tool to the environment variable PATH. The executable directory is the Multi Core Performance Tool directory in the installation directory. Substitute *version* with the correct version number.

```
set PATH=%PATH%;"C:\Program Files\TASKING\MCPT version\mcptool"
```

Command line examples

1. To run an RTOS analysis on `demo_rtos_isr_tc39x` using one shot trace mode on a specific core, use option `--core=core-nr`. For the TC39x derivative you can use the values 0, 1, 2, 3, 4 and 5. Repeat this option to trace multiple cores at once. The amount of cores that can be run depends on the device (see the table in [Section 2.3, Trace Support](#)). So, for the TC39x you can run 3 cores at once and core 0 must be one of them. Be aware that the cores need to be enabled in the startup code of the application. Otherwise the analysis run will not terminate.

```
McpToolCmd --run=RTOSAnalysis-1 demo_rtos_isr_tc39x.McpTool --core=0  
--core=1 --core=2 --periodclocks=100000 --periodcount=10
```

The results are exported to the CSV file `demo_rtos_isr_tc39x_rtos_activities.csv`. You can inspect this file with any text editor. The first line in a CSV file shows the columns that are used.

Note that the command line invocation does not add a new result entry to the `demo_rtos_isr_tc39x.McpTool` file.

2. To run a Core Load analysis on `demo_rtos_isr_tc39x` using one shot trace mode and compare the results with `Result-1`, enter:

```
McpToolCmd --run=RTOSAnalysis-1 demo_rtos_isr_tc39x.McpTool --core=0  
--core=1 --core=2 --periodclocks=100000 --periodcount=10  
--compare=Result-1
```

The results of the comparison are exported to the CSV file

`demo_rtos_isr_tc39x_diff_rtos_activities.csv`. If all value fields are zero, this indicates that the results are identical. Fields that contain zeros indicate no change. Fields with negative values indicate an improvement, fields with positive values indicate worse performance. In this example the comparison is worse, because we compare the original result (non-fixed sources) with a version where the sources have been fixed. Normally, you compare your results with a previous result.

3. To run an analysis using continuous trace mode use option **--continuous**. A continuous run ends when the specified number of periods are collected.

```
McpToolCmd --run=RTOSAnalysis-1 demo_rtos_isr_tc39x.McpTool --core=0
--core=1 --core=2 --periodclocks=100000 --periodcount=10
--compare=Result-1 --continuous
```

4. To specify a remote host to connect to the target, use option **--server=hostname**. The default, if you do not specify this option, is `localhost`.

```
McpToolCmd --run=RTOSAnalysis-1 demo_rtos_isr_tc39x.McpTool --core=0
--core=1 --core=2 --periodclocks=100000 --periodcount=10
--compare=Result-1 --continuous --server=myservername
```

5.3. What to Do if Your Application Does not Start on a Board?

When you profile an application and you encounter the error message:

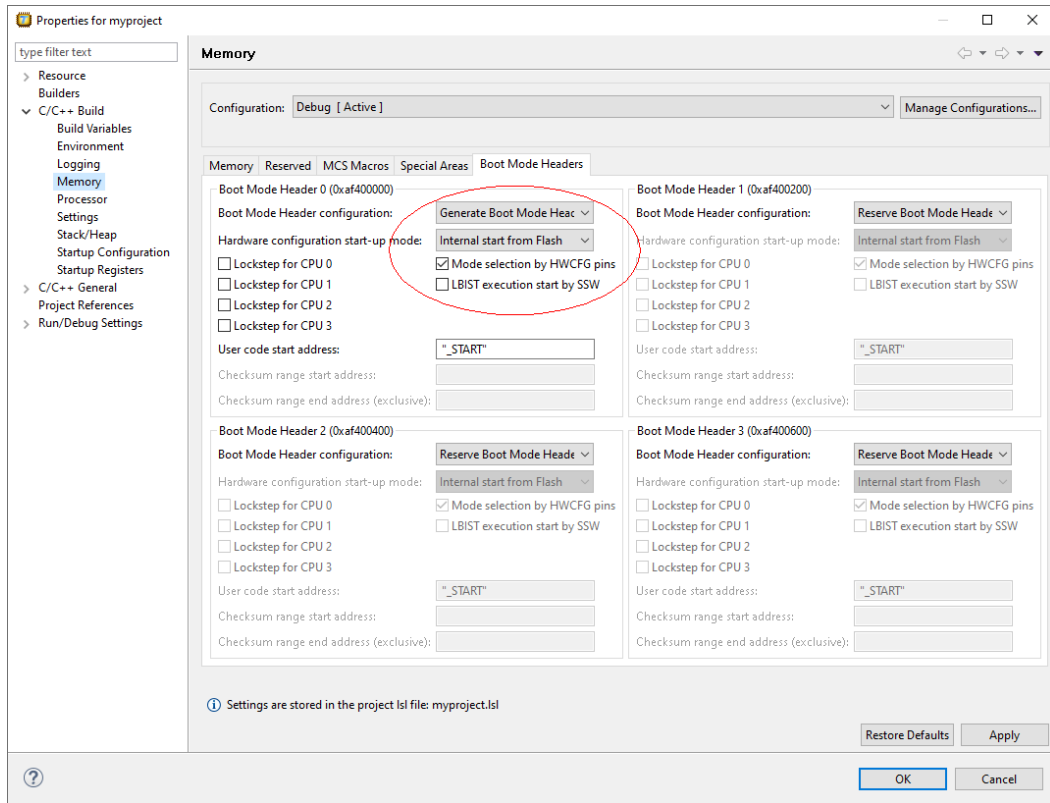
```
Trace error: cannot find code at address address
Do you want to continue the run?
```

it might be the case that the application does not include a valid Boot Mode Header 0 (BMHD0) configuration, or that the start address in the Boot Mode Header on the target does not match the start address of the application. In order to fix this you need to initialize a Boot Mode Header for your target. But be careful, you need to know what you are doing, because wrong use of the Boot Mode Headers might brick the device. Therefore, we advice you to first read chapter 4 TC29x BootROM Content of the AURIX™ TC29x B-Step User's Manual, or similar chapter in the User's Manual for other devices. Also read sections 7.9.13 Boot Mode Headers, and section 9.7.1. Boot Mode Headers in the TriCore User Guide.

To initialize the Boot Mode Header using Eclipse in the TASKING VX-toolset for TriCore:

1. From the **Project** menu, select **Properties for » C/C++ Build » Memory**, and open the **Boot Mode Headers** tab.
2. In **Boot Mode Header 0**, from the **Boot Mode Header configuration**, select **Generate Boot Mode Header**.

TASKING Multi Core Performance Tool User Guide



3. Leave the other default settings untouched and select **OK**.

This will initialize the Boot Mode Header to allow for stand-alone execution of the target.

Chapter 6. Reference

Every analysis result shows a number of tabs with information. What information is shown depends on the type of the analysis: Core Load Analysis or RTOS Analysis.

Furthermore there is a Settings dialog where you can specify values that influence the way information is shown in the analysis results.

This chapter contains a description of the Settings dialog and contains an overview of all the fields and columns in an analysis result.

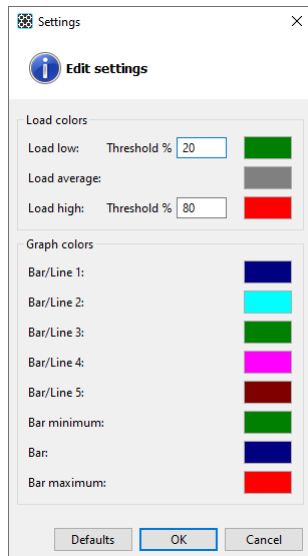
6.1. Settings Dialog

In the Settings dialog you can specify values that influence the way information is shown in the analysis results.

To open the Settings dialog

1. From the **Project** menu, select **Settings**.

The Settings dialog appears.



2. Change the values and/or colors and click **OK**.

All results will be updated to reflect the new settings.

When you click the **Defaults** button all the values of the Settings dialog are reset to their initial values. By clicking on a color a color selector pops up where you can change the color.

In the **Graph colors** group you can change the colors of the graph lines.

When you run a Core Load analysis, the settings in the **Load** group are visible on the Core Load tab in the Load column. The load threshold values determine when the cells on the Core Load tab in the Load column are changing their colors.

When you run an RTOS analysis, the settings in the **Load** group are visible on the Activities tab in the Clocks column. The load threshold values determine when the cells on the Activities tab in the Clocks column are changing their colors.

6.2. Summary Tab

On the Summary tab the following information is available for the different analysis types

Core Load analysis

- Configuration
- Info
- Core Load
- Core Load Per Period (only visible if there are multiple periods)

RTOS analysis

- Configuration
- Info
- Top Activities
- Top Activities Per Period (only visible if there are multiple periods)

6.2.1. Configuration

The **Configuration** part of the Summary tab contains the following information.

Configuration	
Processor:	TC39xED
Trace settings:	Core=[0, 3, 5], Memory=TCM (tile 0-7), Mode=Reset-Continuous
Trace periods:	5x80.000=400.000 cycles

Information	Description
Processor	The name of the selected processor device
Trace settings	The trace configuration settings (e.g. selected cores, buffer mode, attach mode, memory type, memory range, etc.)
Trace periods	The period configuration settings (e.g. number of periods, period size, etc.)

Items that are marked red have additional information, hover the mouse over a value to see this additional information.

6.2.2. Info

The **Info** part of the Summary tab contains the following information.

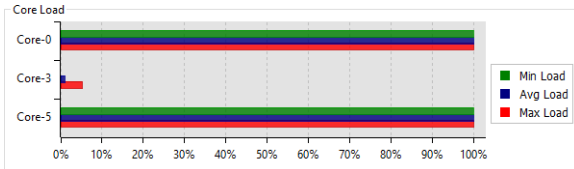
Info	
Started at:	Feb 26, 2020 3:49:59 PM
Consumed time:	2 seconds
Clock frequencies (MHz):	CPU0=98, CPU1=98, CPU2=98, CPU3=98, CPU4=98, CPU5=98, SRI=98, SPB=49, BBB=98
CPU data/program cache:	DCACHE0=0, PCACHE0=0, DCACHE1=0, PCACHE1=0, DCACHE2=0, PCACHE2=0, DCACHE3=0, PCACHE3=0, DCACHE4=0, PCACHE4=0, DCACHE5=0, PCACHE5=0
CPU clock count:	Core-0=400.000, Core-3=400.000, Core-5=400.000

Information	Description
Started at	The date and time the analysis was run
Consumed time	The time (in seconds) it took for the analysis to complete
Clock frequencies (MHz)	The values of several clock frequencies. The values are read at the start of the analysis before any reset. If the CPU was reset or halted at analysis start, the clock frequencies are not measured.
CPU data/program cache	The CPU 0, 1, 2, ... data cache (DCache) and program cache (PCache) settings. DCACHE0=1 means CPU0.DCACHED is enabled, PCACHE1=0 means CPU1.PCACHED is disabled. The values are read at the start of the analysis before any reset.
CPU clock count	The number of CPU clock cycles on the board it took to run the analysis

Items that are marked red have additional information, hover the mouse over a value to see this additional information.

6.2.3. Core Load

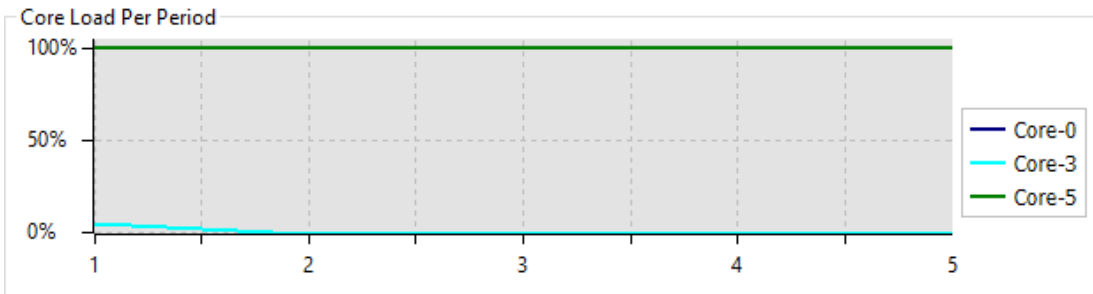
The **Core Load** part of the Summary tab shows the minimum, maximum and average load for each core. This chart is available for all Core Load Analyses. As you can see in the following example, core 0 and core 5 do not spend time in the idle function.



If you double-click on a bar, the Core Load tab opens at the selected core.

6.2.4. Core Load Per Period (only visible when there are multiple periods)

The **Core Load Per Period** is a part of the Core Load analysis displayed on the Summary tab and is only visible when there are multiple periods. This graph shows the load of each core per period.



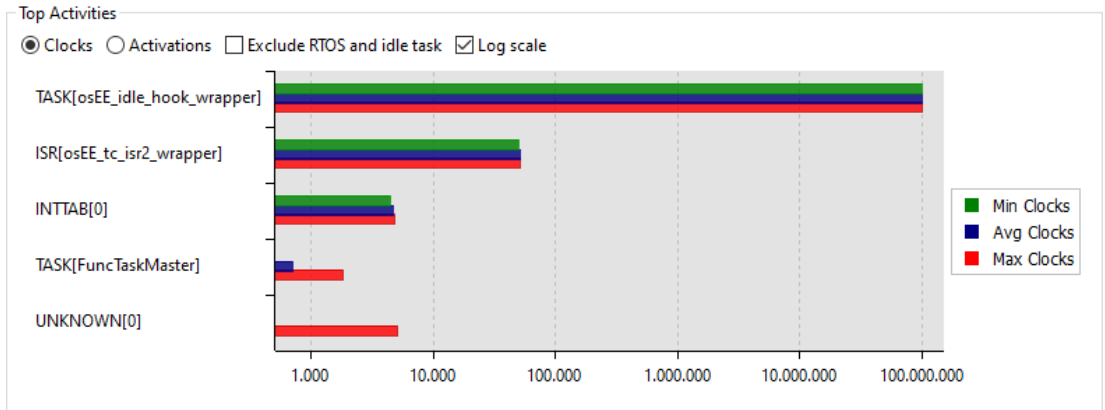
If you double-click on a line, the Core Load tab opens at the selected core.

6.2.5. Top Activities

The **Top Activities** is a part of the RTOS analysis displayed on the Summary tab. This graph shows the top activities sorted on either the **Clocks** or the **Activations**.

The **Exclude RTOS and idle task** check box removes the RTOS activity and the given idle task (the idle task is set in the analysis wizard), when this check box is enabled.

When the check box **Log scale** is enabled the graph is displayed logarithmic.

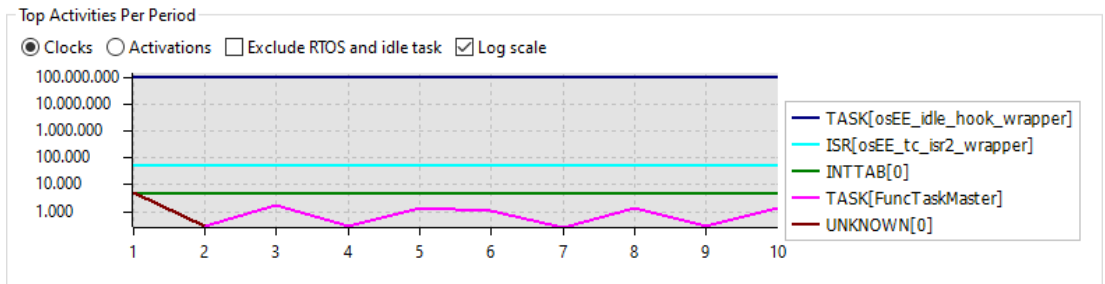


6.2.6. Top Activities Per Period (only visible when there are multiple periods)

The **Top Activities Per Period** is a part of the RTOS analysis displayed on the Summary tab and is only visible when there are multiple periods. This graph shows the top activities per period sorted on either the **Clocks** or the **Activations**.

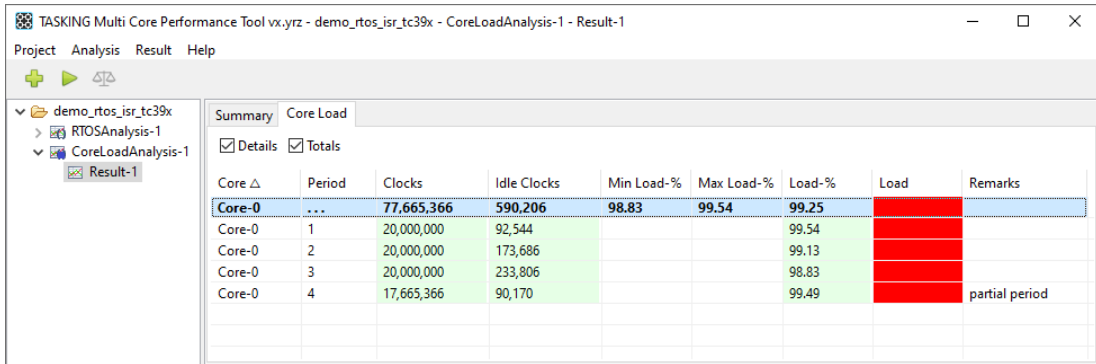
The **Exclude RTOS and idle task** check box removes the RTOS activity and the given idle task (the idle task is set in the analysis wizard), when this check box is enabled.

When the check box **Log scale** is enabled the graph is displayed logarithmic.



6.3. Core Load Tab

The Core Load tab shows a table with all the measured clocks for a Core Load analysis.



Click on a column to sort the table according to the information in that column. For sub sorting keep the Ctrl key pressed while clicking on another column.

Uncheck the **Details** check box to hide the detail information. Uncheck the **Totals** check box to hide the totals (At least one of the 2 check boxes must be enabled, this will be forced by the program).

Hover the mouse over a column to see additional information.

The Core Load tab contains the following information:

Information	Description
Core	The measured core
Period	The measured period
Clocks	The measured total clocks
Idle Clocks	The measured total idle clocks
Min Load - %	The lowest measured load in percentage
Max Load - %	The highest measured load in percentage
Load - %	The average measured load in percentage
Load	The average measured load displayed as color in percentage
Remarks	Remarks

6.4. Activities Tab

The Activities tab shows a table with all the measured clocks for an RTOS analysis.

Core	Activity Δ	Period	Activations	Clocks Δ-1	Clocks Period-% Δ-1	Clocks Total-% Δ-1
Core-0	ISR[os_tricore_isr2_srb03_interr...	...	8	147,176		
Core-0	ISR[os_tricore_isr2_srb03_interrupt...	2	2	36,776		
Core-0	ISR[os_tricore_isr2_srb03_interrupt...	3	2	36,776		
Core-0	ISR[os_tricore_isr2_srb03_interrupt...	4	2	36,776		
Core-0	ISR[os_tricore_isr2_srb03_interrupt...	1	2	36,848		
Core-0	RTOS[0]	...	-	45,442,958		
Core-0	RTOS[0]	1	-	52,044		
Core-0	RTOS[0]	2	-	52,726		
Core-0	RTOS[0]	3	-	52,760		
Core-0	RTOS[0]	4	-	2,444,094		
Core-0	RTOS[0]	7	-	2,841,334		
Core-0	RTOS[0]	5	-	20,000,000		
Core-0	RTOS[0]	6	-	20,000,000		
Core-0	TASK[os_idle]	...	586	30,063,450		
Core-0	TASK[os_idle]	4	120	6,040,318		
Core-0	TASK[os_idle]	3	154	7,896,838		
Core-0	TASK[os_idle]	1	156	8,034,974		
Core-0	TASK[os_idle]	2	156	8,091,320		
Core-0	TASK[os_u_T0]	...	64	10,085,226		
Core-0	TASK[os_u_T0]	4	15	2,363,726		
Core-0	TASK[os_u_T0]	1	16	2,520,758		
Core-0	TASK[os_u_T0]	2	17	2,525,766		
Core-0	TASK[os_u_T0]	3	16	2,674,976		
Core-0	TASK[os_u_T1]	...	1	9,981,888		
Core-0	TASK[os_u_T1]	4	-	2,450,254		
Core-0	TASK[os_u_T1]	2	-	2,494,696		
Core-0	TASK[os_u_T1]	1	1	2,497,800		
Core-0	TASK[os_u_T1]	3	-	2,539,138		

Click on a column to sort the table according to the information in that column. For sub sorting keep the Ctrl key pressed while clicking on another column.

Uncheck the **Details** check box to hide the detail information. Uncheck the **Totals** check box to hide the totals (At least one of the 2 check boxes must be enabled, this will be forced by the program).

Hover the mouse over a column to see additional information.

The Activities tab contains the following information:

Information	Description
Core	The measured core
Activity	The name of the RTOS activity (e.g. task name, interrupt name, etc.)
Period	The measured period
Activations	The number of time the start address of the activity occurs
Clocks	The measured total clocks
Clocks Period - %	The measured total clocks per period in percentage

Information	Description
Clocks Total - %	The measured total clocks in percentage

6.5. Raw Trace Data Tab

The Raw Trace Data tab is for advanced users who want to examine program flow. For RTOS analyses the raw trace can be searched through for memory writes to see task switches. This tab is available for all analysis types, but only when you enable **Raw data** in the **Run Analysis** dialog.

Hover the mouse over a value to see additional information.

Nr	Ticks	OPoint	Origin	Operation	Data	Address
19,812	4	CPU0	CPU0	IP_CALL	0x0	0x80004118
19,813	51	CPU0	CPU0	IP_RET	0x0	0x8000437e
19,814	6	CPU0	CPU0	IP	0x0	0x8000411c
19,815	1	CPU0	CPU0	IP_RET	0x0	0x80004120
19,816	6	CPU0	CPU0	IP_CALL	0x0	0x800032b2
19,817	44	CPU0	CPU0	IP_RET	0x0	0x80004f80
19,818	6	CPU0	CPU0	IP_CALL	0x0	0x800032b6
19,819	61	CPU0	CPU0	IP_RET	0x0	0x80004fb2
19,820	20	CPU0	CPU0	IP	0x0	0x800032ba
19,821	2	CPU0	CPU0	MEMORY_READ	0x100000f0	0x10000000
19,822	55	CPU0	CPU0	IP_CALL	0x0	0x800032dc
19,823	37	CPU0	CPU0	IP	0x0	0x80005290
19,824	3	CPU0	CPU0	IP	0x0	0x8000527e
19,825	19	CPU0	CPU0	IP_RET	0x0	0x8000529e
19,826	4	CPU0	CPU0	IP	0x0	0x800032e0
19,827	2	CPU0	CPU0	IP_CALL	0x0	0x800032e4
19,828	43	CPU0	CPU0	IP_RET	0x0	0x80004f80
19,829	6	CPU0	CPU0	IP	0x0	0x800032e8
19,830	7	CPU0	CPU0	IP_RFE	0x0	0x800032f0
19,831	0	CPU0	CPU0	STATE	0x100	0x0
19,832	4	CPU0	CPU0	IP	0x0	0x80004764
19,833	0	CPU0	CPU0	STATE	0x180	0x0
19,834	1	CPU0	CPU0	IP	0x0	0x80004766
19,835	3	CPU0	CPU0	IP	0x0	0x80004770

In the **Search** field you specify a search pattern. When you hover the mouse over the **Search** field, you can see all search possibilities.

Search:

Search multiple criteria (space delimited, case insensitive, arbitrary order), use at most one of each of the following:

- OPoint/Origin: name
- Operation: name
- data/address: value (integer/octal/hex)
- data/address: value+range (both integer/octal/hex)
- minimum ticks: >ticks

Example: IP_CALL 0x8000300+0x100 >50

(Ctrl+F to focus this field)

All matches are marked with a gray bar. With the buttons you can navigate to the Next, Previous, First or Last occurrence.

The Raw Trace Data tab contains the following information:

Column	Description
Nr	The sequence number for every raw trace operation.
Ticks	The MCDS clock Ticks between trace messages. Please note that one Tick is equal to two CPU cycles.
OPoint	Displays the Observation Point of the trace data. The observation point is the physical data acquisition point inside the SoC (System-on-Chip). For example the CPU0, CPU1, SRI bus, and so on.
Origin	The origin of the activity. In most cases this is the same as OPoint.
Operation	The operation being executed but not on the level of assembler mnemonics for program trace. It displays a more abstract type of the operation. For example, IP_CALL, IP_RET, MEMORY_READ or MEMORY_WRITE.
Data	The data written or read.
Address	The pointer of the instruction (IP) which is being executed. If the Operation column displays an R/W Operation, the Address column displays the address where data is read or written to.

