

TASKING[®]

***TASKING Embedded Profiler
User Guide***

Copyright © 2017 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	2
1.2.2. Frequently Asked Questions (FAQ)	3
1.2.3. Installing a License	3
2. Introduction to the TASKING Embedded Profiler	7
2.1. Emulation Device (ED)	9
2.2. Trace Support	9
3. Tutorial	11
3.1. Prepare Demo Project in Eclipse	11
3.2. Analyze Project in TASKING Embedded Profiler	16
3.3. Fix the Problem	25
3.4. Verify Fix in TASKING Embedded Profiler	26
3.5. Compare Results	29
3.6. Export Results	30
4. Using the TASKING Embedded Profiler	31
4.1. Run the Embedded Profiler in Interactive Mode	31
4.2. Run the Embedded Profiler from the Command Line	32
4.2.1. Command Line Tutorial	33
5. Reference	35
5.1. Summary Tab	35
5.1.1. Info	36
5.1.2. Performance Hotspots	37
5.1.3. Data Access Intensive Functions	38
5.1.4. Memory Access Conflicts	39
5.1.5. ICache Misses	40
5.1.6. DCache Misses	40
5.2. Hot Functions Tab	41
5.3. Source Line Results Tab	42
5.4. Instruction Results Tab	43
5.5. Source Tab	44
5.6. Disassembly Tab	44

Manual Purpose and Structure

Manual Purpose

You should read this manual if you want to know:

- how to use the TASKING Embedded Profiler
- the features of the TASKING Embedded Profiler

Manual Structure

Chapter 1, *Installing the Software*

Explains how to install and license the TASKING Embedded Profiler.

Chapter 2, *Introduction to the TASKING Embedded Profiler*

Contains an introduction to the TASKING Embedded Profiler 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 Embedded Profiler.

Chapter 4, *Using the TASKING Embedded Profiler*

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

Chapter 5, *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]
- 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]

TASKING Embedded Profiler User Guide

- 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]

Chapter 1. Installing the Software

This chapter guides you through the installation process of the TASKING[®] Embedded Profiler. It also describes how to license the software.

In this manual, **TASKING Embedded Profiler** and **Embedded Profiler** are used as synonyms.

1.1. Installation for Windows

System Requirements

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

- Windows 7 or higher
- 2 GHz Pentium class processor
- 1 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 a 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.

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/download.shtml>.
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  
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/faq.shtml>

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.

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.

TASKING Embedded Profiler User Guide

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

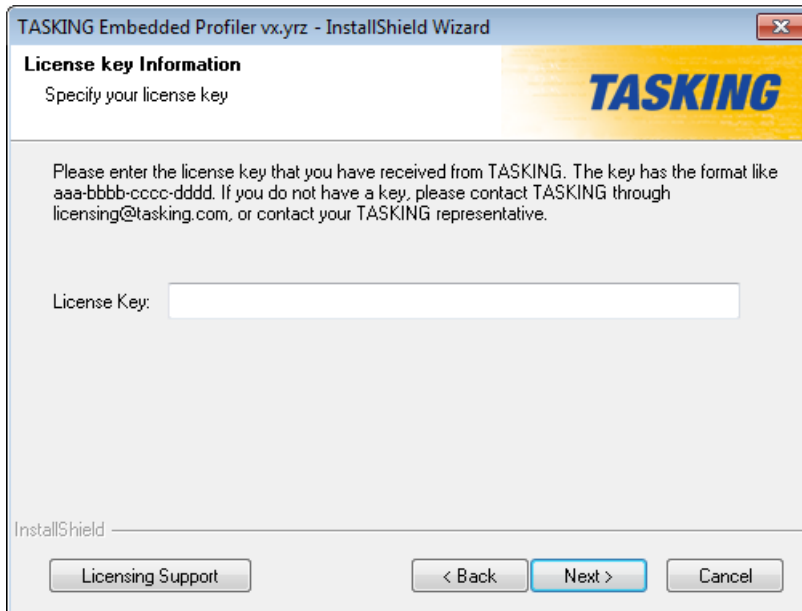
For using the TASKING license servers the TASKING license manager tries to connect to the Remote TASKING servers `lic1.tasking.com` .. `lic4.tasking.com` at the TCP ports 8080, 8936 or 80. Make sure that the firewall in your network has transparent access enabled for one of these ports.

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 Embedded Profiler vx.yrz - InstallShield Wizard". The main heading is "License key Information" with the instruction "Specify your license key". On the right, there is a yellow banner with the "TASKING" logo. Below the heading, there is a text box containing the following text: "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 window, there are four buttons: "Licensing Support", "< Back", "Next >", and "Cancel". The "Next >" button is highlighted in blue.

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.

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

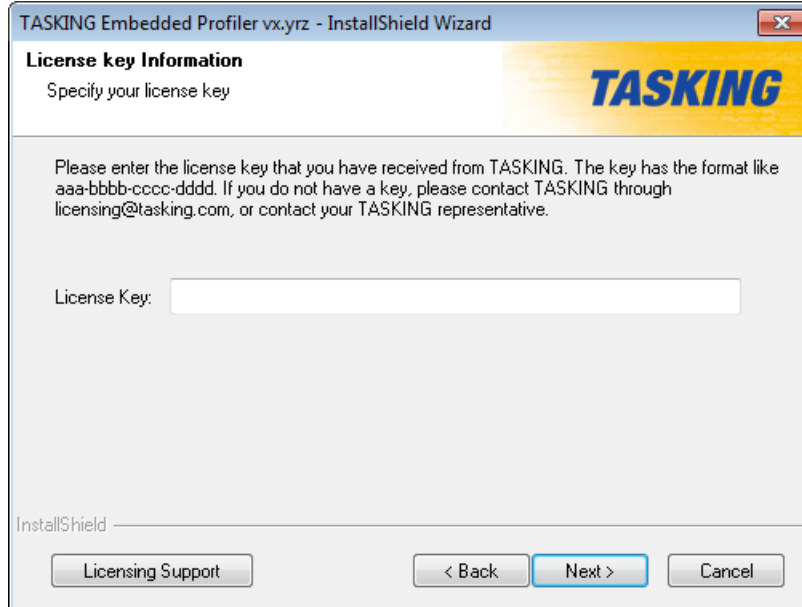
5. Select **Remote TASKING license server** to use one of the remote TASKING license servers, or select **Local TASKING license server** for a local license server. The latter requires optional software.
6. (For local license server only) specify the **Server name** and **Port number** of the local license server.
7. Click **Finish** 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.



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

TASKING Embedded Profiler User Guide

*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.*

3. Select **Node-locked client based license** and click **Next** to continue.
4. In the **License file content** 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.
5. Click **Finish** to complete the installation.

Chapter 2. Introduction to the TASKING Embedded Profiler

After your application has been verified, thoroughly tested and debugged, and by itself behaves correctly, you may still run into performance and timing issues. Many timing issues can be addressed simply by improving the performance of the applications that caused a missed deadline. Furthermore, by reducing the core load of your applications you may be able to go for a device that is cheaper because it has fewer cores. A way to address these issues is performance tuning.

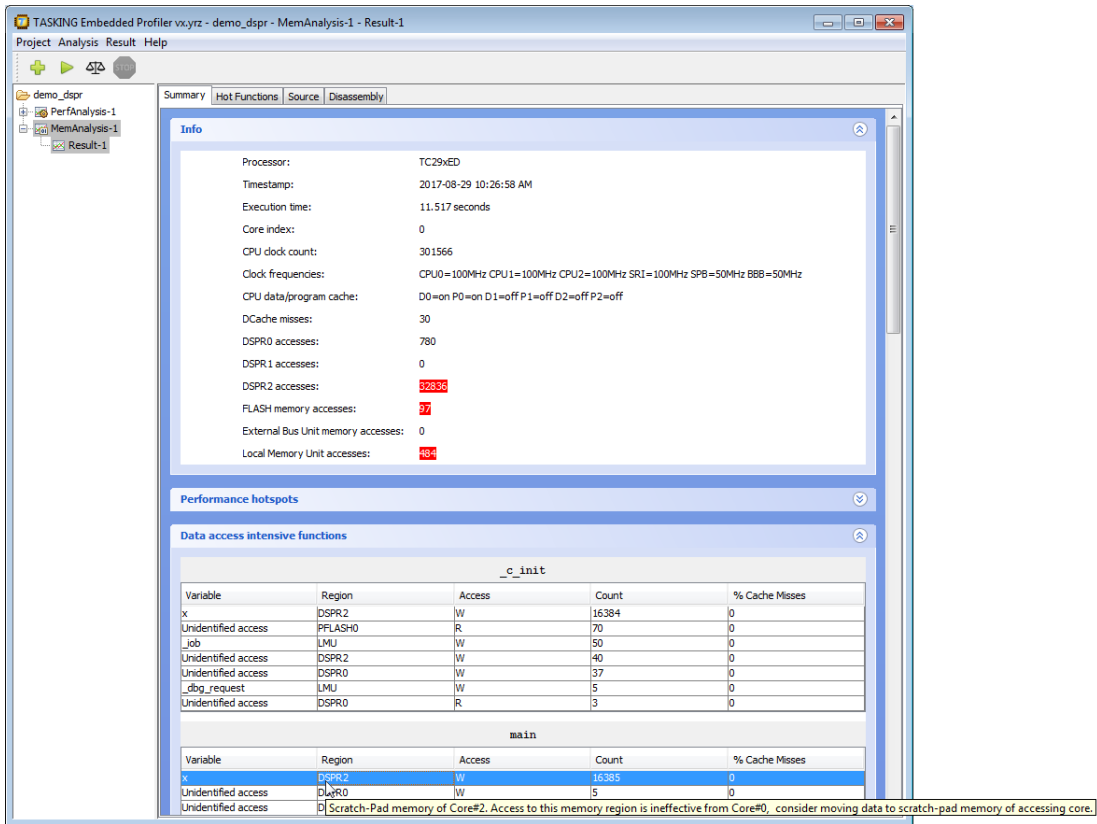
With performance tuning we refer to optimizing your application for a specific target device. Common situations where performance tuning of your application makes sense are:

- You are using self-made libraries that are called a lot and thus have a big impact on overall application performance.
- You develop/adapt low level drivers and basic software (BSW) or operating system (OS) components.
- You are close to or above your core load budget limit.
- You have a timing problem in your schedule that could be fixed by speeding up specific tasks but want to avoid changing the schedule.
- You want to try and target a smaller electronic control unit (ECU) in order to save costs.
- You care about easily and cost effectively tracking and improving the performance of your code on target devices.

Embedded hardware platforms are too complex for the average software developer to predict or understand the performance of his code. In order to optimize code for a specific platform (cores plus peripherals), developers need feedback from the hardware on which specific part of their code is suboptimal (in terms of memory consumption, jitter, execution time, ...) and what is the root cause of the performance impact. The TASKING Embedded Profiler is a smart profiling tool that provides this feedback.

The TASKING Embedded Profiler communicates with an embedded processor (CPU) to gather real-time tracing and performance data. The tool gives an overview over the current clock settings — no need to get an oscilloscope to verify that the clocks are configured properly for a benchmark run. After verification of correct clock setup, you are guided through a few easy steps that pinpoint the source lines that have the greatest performance impact. The tool indicates the root cause of the performance impact and gives simple instructions on how to address the problem. The data is presented in graphics and tables and into computer readable formats.

TASKING Embedded Profiler User Guide



Summary Hot Functions Source Disassembly

Info

Processor: TC29xED
Timestamp: 2017-08-29 10:26:58 AM
Execution time: 11.517 seconds
Core index: 0
CPU clock count: 301566
Clock frequencies: CPU0=100MHz CPU1=100MHz CPU2=100MHz SRI=100MHz SPB=50MHz BBB=50MHz
CPU data/program cache: D0=on P0=on D1=off P1=off D2=off P2=off
DCache misses: 30
DSPR0 accesses: 780
DSPR1 accesses: 0
DSPR2 accesses: 32385
FLASH memory accesses: 37
External Bus Unit memory accesses: 0
Local Memory Unit accesses: 484

Performance hotspots

Data access intensive functions

_c_init

Variable	Region	Access	Count	% Cache Misses
x	DSPR2	W	16384	0
Unidentified access	PFLASH0	R	70	0
_job	LMU	W	50	0
Unidentified access	DSPR2	W	40	0
Unidentified access	DSPR0	W	37	0
_dbg_request	LMU	W	5	0
Unidentified access	DSPR0	R	3	0

main

Variable	Region	Access	Count	% Cache Misses
x	DSPR2	W	16385	0
Unidentified access	DSPR0	W	5	0
Unidentified access	Scratch-Pad memory of Core#2	W	5	0

Scratch-Pad memory of Core#2. Access to this memory region is ineffective from Core#0, consider moving data to scratch-pad memory of accessing core.

After applying the suggested mitigation, you can use the TASKING Embedded Profiler to confirm that the problem has indeed been fixed. With the default settings of the tool this all happens non-intrusively with real data collected from the application running on the real device. Using such a performance tuning tool, non-expert users can often highly speed up untuned applications.

Features of the TASKING Embedded Profiler

- Performance analysis
- Memory access analysis
- Function-level 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 support

- Support for Device Access Server (DAS) v6.0 and Device Access Port (DAP) miniWiggler

Performance analysis

This type of analysis traces instructions and performance events. It measures the CPU clock count and it finds branch misses, cache misses and stalls due to memory access delays or pipeline hazards. You can run this type of analysis on the whole application or select specific functions.

Memory access analysis

This type of analysis traces function calls, function returns and data accesses. You can run this type of analysis on the whole application or select specific functions.

Function-level analysis

This type of analysis traces all function calls and function returns. This is the fastest analysis.

2.1. Emulation Device (ED)

The standard TriCore/AURIX™ processors (production devices) lack debug trace functionality. However, this functionality is very useful when you develop and test your application. Therefore pin compatible Emulation Devices (ED) are available. An Emulation Device has an Emulation Extension Chip (EEC) added to the same silicon, which is accessible through the JTAG or DAP interface. The TASKING Embedded Profiler supports the on-chip trace feature of the Emulation Device. See the processor documentation for detailed information about the device.

Some Production Devices, such as the TC29x, are equipped with a mini-MCDS, which 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. See the processor documentation for detailed information about the device.

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 a detailed naming convention see the [AURIX™ Product Naming PDF on the Infineon website](#).

2.2. Trace Support

The TASKING Embedded Profiler uses the on-chip trace (MCS) concept. For detailed information about the Multi-Core Debug Solution (MCDS) we recommend that you read the processor documentation belonging to the Emulation Device.

Tile memory range

Trace information is stored in a dedicated trace buffer. With an Emulation Device you can allocate part of the Emulation Memory (EMEM) as trace buffer memory. The Emulation Memory is divided in so-called 'tiles', and you can choose which part of the Emulation Memory should be 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 Embedded Profiler, you can set the trace mode:

- **One shot mode.** In this mode the analysis will run until the trace buffer is full. This is non-intrusive, meaning that the trace does not interfere the running processor. After the trace has stopped the profiler reads the collected data.
- **Continuous trace.** In this mode the analysis will run until the application stops 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 profiler can read the collected data. After that the processor continues writing to the buffer.

Attach mode

When you run a trace analysis in the TASKING Embedded Profiler, you can set the attach mode:

- **Reset device.** In this mode the device is reset first and then the analysis starts.
- **Hot attach.** In this mode the analysis will start at the current execution position of the running application.

Chapter 3. Tutorial

The `profiler\tutorials` directory of the TASKING Embedded Profiler installation contains several examples. They serve as a good starting point for your own profiling analysis project.

- `demo_dspr` - A project for the TC29xB demonstrating how defaulting to the wrong scratch pad memory results in a penalty in stalls.
- `demo_dcache` - A project for the TC29xB demonstrating how multiple passes over a large buffer can cause many data cache misses.
- `demo_concurrent` - A project for the TC29xB demonstrating how accessing the same memory from multiple cores causes stalls.

All examples come with embedded profiler projects (files with the `.EmbProf` extension), with pre-run analyses. You can open a project in the TASKING Embedded Profiler to inspect the various analysis results, without having to run the examples on a target board.

In this tutorial we will use the `demo_dspr` example to go through the process of preparing your project from scratch, running a profiling analysis, fixing the problem and rerunning a profiling analysis to see the improvement. After this tutorial you can use the other tutorials yourself in a similar way.

3.1. Prepare Demo Project in Eclipse

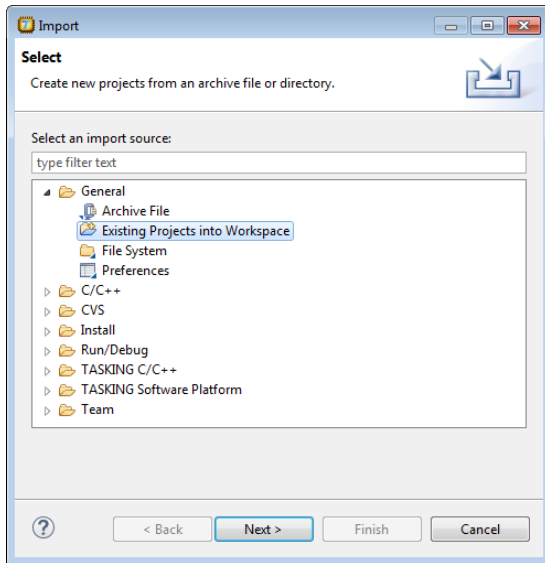
Before you can use the TASKING Embedded Profiler, you must have an application ELF file with debug information and the application must be downloaded onto a target board.

The example projects delivered with the TASKING Embedded Profiler are Eclipse projects suitable for the TASKING VX-toolset for TriCore v6.2r1 or higher. For this part of the tutorial it is assumed that you have this toolset installed.

Import an example project

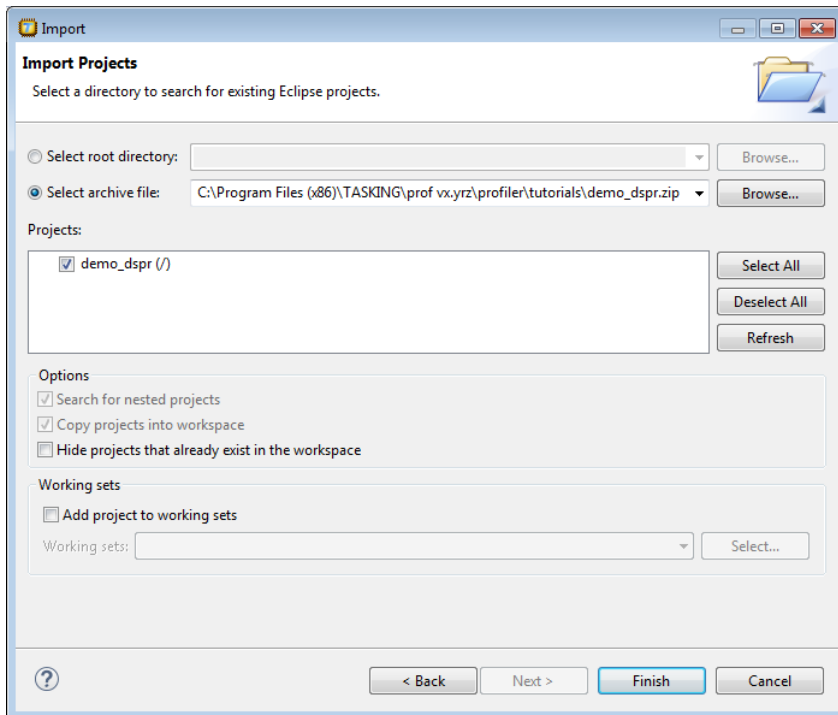
1. Start the TASKING VX-toolset for TriCore Eclipse IDE.
2. From the **File** menu, select **Import**.

The Import dialog appears.



3. Select **General » Existing Projects into Workspace** and click **Next**.

The Import Projects dialog appears.



4. Click **Select archive file** and browse to the example ZIP file delivered with the TASKING Embedded Profiler.
5. Leave the other settings in this dialog as is and click **Finish**.

The project will be added to your workspace.

You can now examine the source files, build the project (for your target) and flash the application.

Examine source file

1. In the C/C++ Projects view double-click on the source file `demo_dspr.c`.

The file will be opened in the source editor.


2. Examine the source file and make sure that the following define has the value 0:

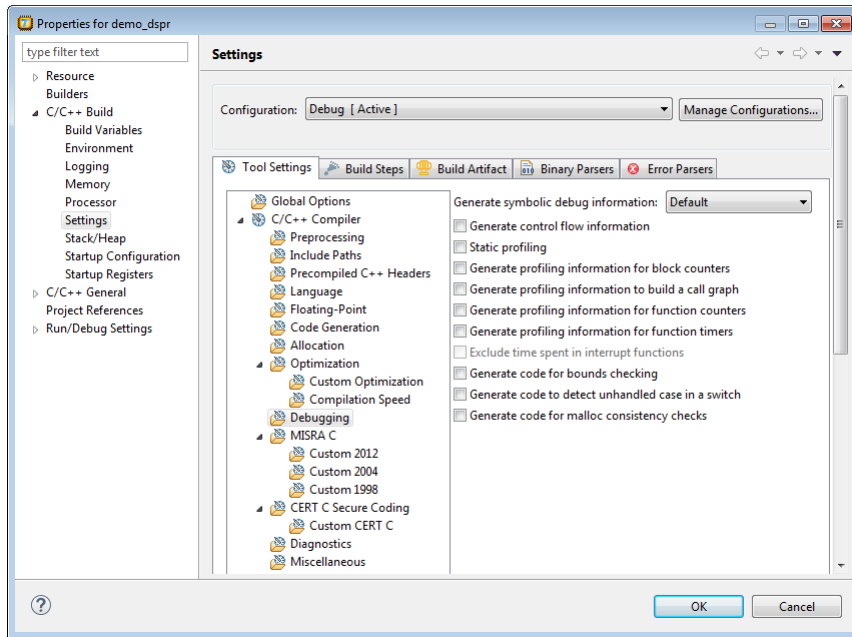
```
#define FIXED 0
```

This define is used to demonstrate the different profiler results before and after fixing the source file.


Set project options

The resulting application ELF file must contain debug information. The demo projects already have debugging enabled by default. So, for the demo projects you can skip this step. For your own project, make sure that debugging is enabled.

1. From the **Project** menu, select **Properties for**. Alternatively, you can click the  button.
The Properties for demo_dspr dialog appears.
2. If not selected, expand **C/C++ Build** and select **Settings** to access the TriCore tool settings.
3. On the Tool Settings tab, expand **C/C++ Compiler » Debugging**, set option **Generate symbolic debug information** to **Default** or **Full** and click **OK**.



Build the project

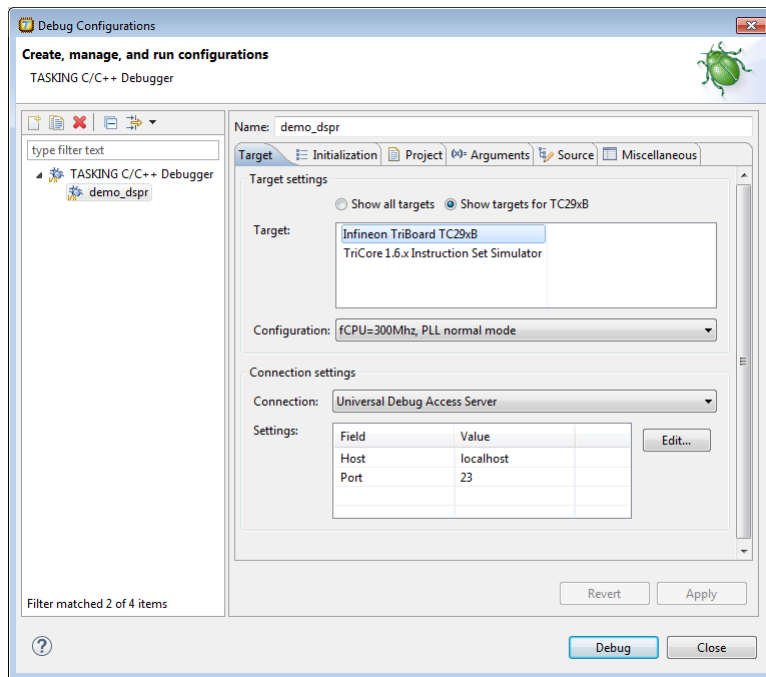
- From the **Project** menu, select **Build demo_dspr**, or click  from the toolbar.

Run the debugger to flash the application onto the target board

1. Connect the Infineon TriBoard TC29xB to your computer. See the documentation that came with the board for more information.
2. From the **Debug** menu, select **Debug project**.

Alternatively you can click the  button in the main toolbar.

Before you can debug a project, you need a Debug launch configuration. Such a configuration, identified by a name, contains all information about the debug project: which debugger is used, which project is used, which binary debug file is used, ... and so forth. So, initially the Debug Configurations dialog appears.



3. On the **Target** tab, select the **Infineon Triboard TC29xB** and click **Debug**.

The TASKING Debug perspective is associated with the TASKING C/C++ Debugger. Because the TASKING C/C++ perspective is still active, Eclipse asks to open the TASKING Debug perspective.

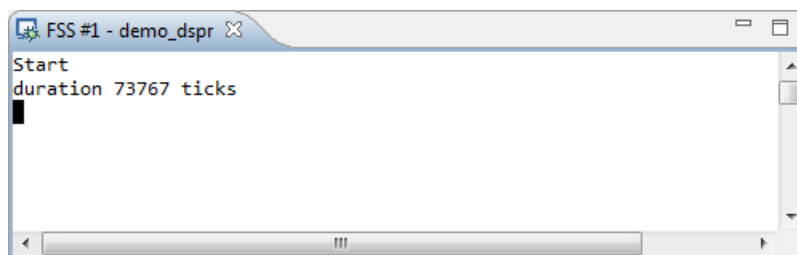
4. Optionally, enable the option **Remember my decision** and click **Yes**.

The debug session is launched. This may take a few seconds.

5. From the **Debug** menu, select **Resume** (▶) to run the application on the target board.

The output of the application appears in the FSS (File System Simulation) view.

6. Inspect the FSS view and notice the number of ticks.



TASKING Embedded Profiler User Guide

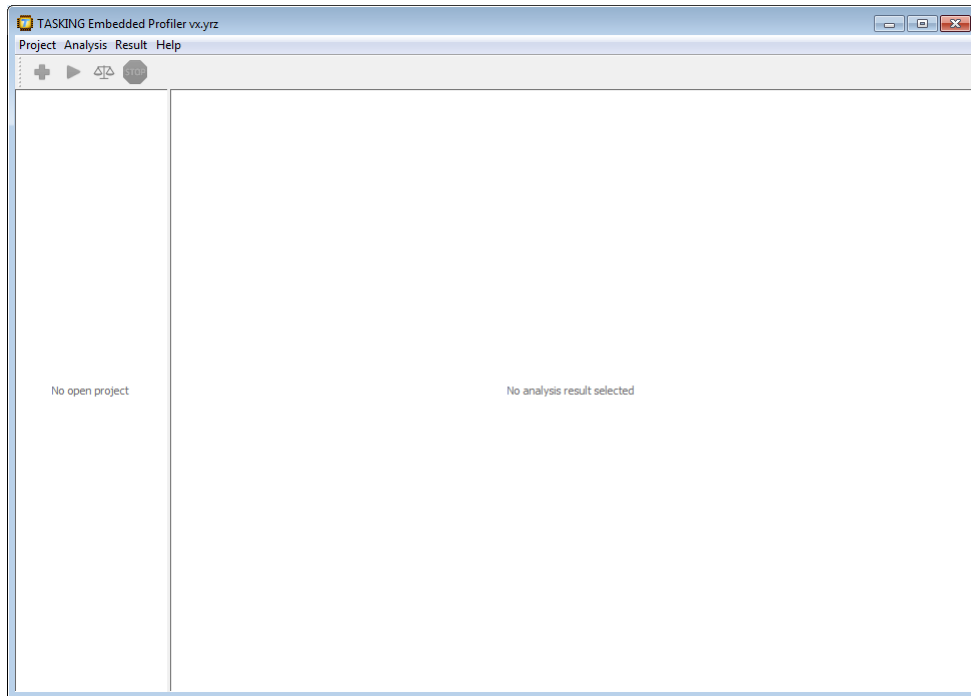
- From the **Debug** menu, select **Terminate** (■) to stop the debugging session. This is necessary to free the connection with the target board.

3.2. Analyze Project in TASKING Embedded Profiler

Now it is time to start analyzing the demo project.

Create a project

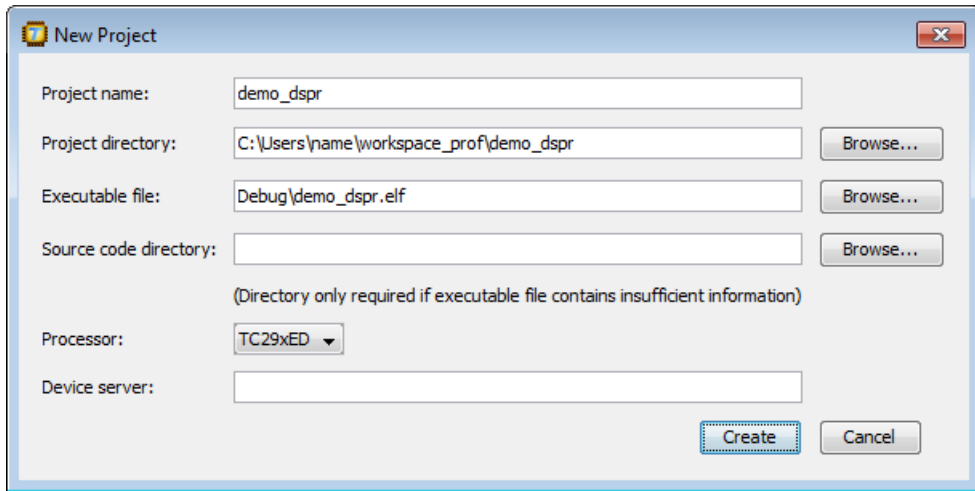
- Start the TASKING Embedded Profiler.



The TASKING Embedded Profiler window is divided into two panes. The left pane is reserved for the project tree and the right pane is reserved for analysis results.


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

The New Project dialog appears.



3. In the **Project name** field, enter the name of the project (for example, you can use the same name as the Eclipse project, `demo_dspr`).
4. In the **Project directory** field, specify the directory where you want to store the Embedded Profiler project file (file with extension `.EmbProf`).
5. In the **Executable file** field, specify the name of the ELF file. This file is usually relative to the project directory. If the executable file is stored in another directory, the full path name is shown.
6. Optionally specify a **Source code directory**. Normally, the location of the source files is taken from the ELF file.
7. Select the **Processor**. For example, `TC29xED`.
8. For the **Device server**, enter the server name (leave blank for `localhost`).
9. Leave the rest of the dialog as is and click **Create**.

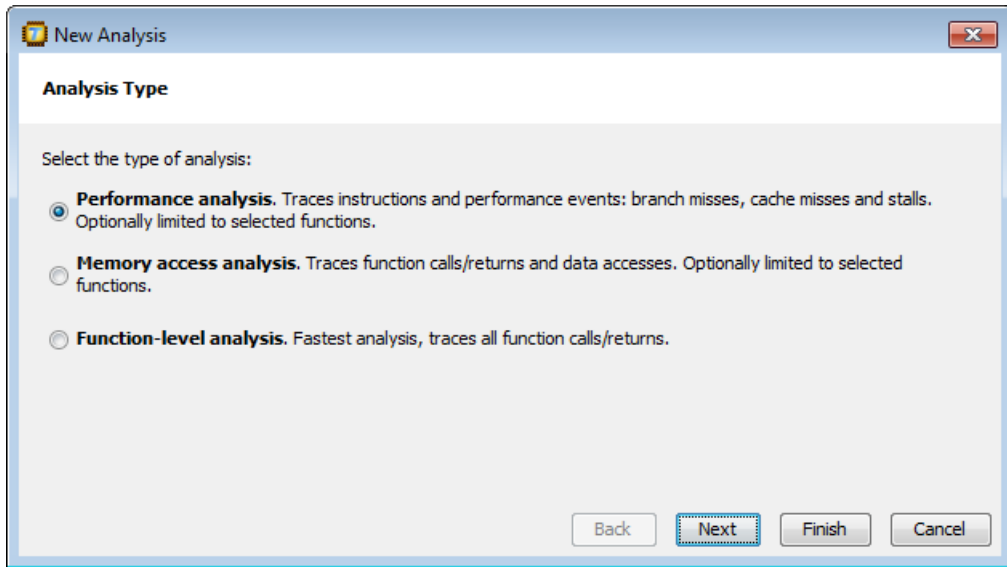
The new project is created and opened.

 `demo_dspr`

Create a Performance analysis

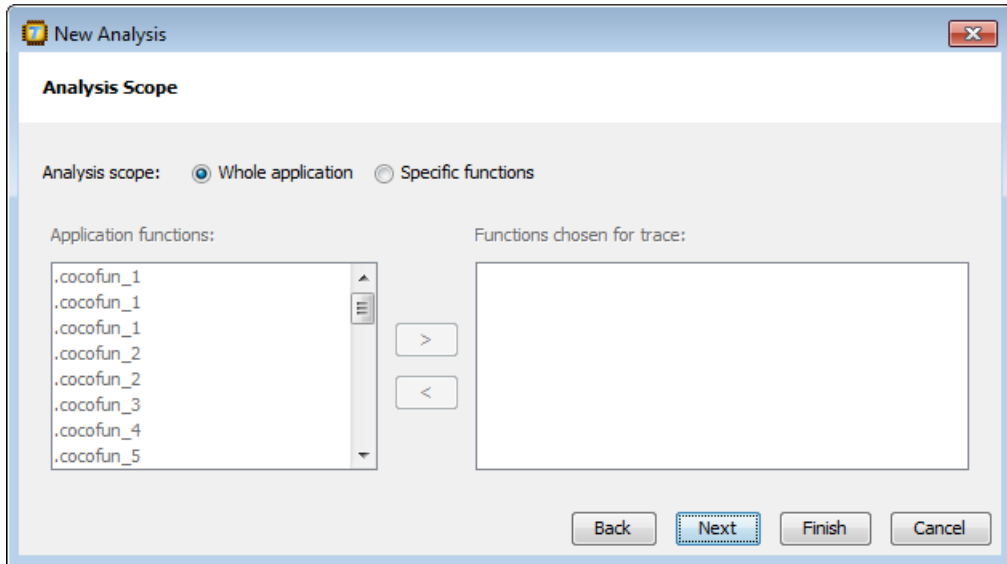
1. From the **Analysis** menu, select **New Analysis**.

The New Analysis wizard appears.



2. Three types of analyses are possible. Select **Performance analysis** and click **Next**.

The Analysis Scope page appears.

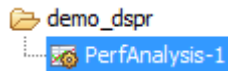


3. For this tutorial select **Whole Application**. If you select **Specific Functions**, select one or more **Application functions** and click **>**.
4. Click **Next**.

The *Analysis Name* page appears.

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

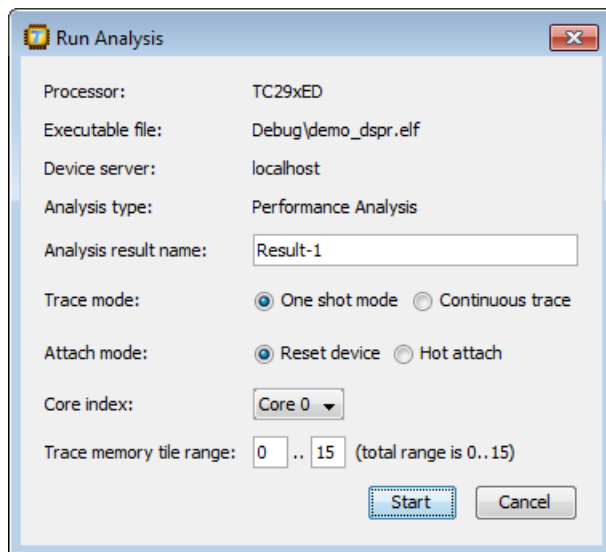
The new analysis is created and is visible in the project tree.



Run the analysis

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

The *Run Analysis* dialog appears.

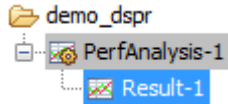


- Enter an **Analysis result name** (default `Result-` and a sequence number).
- Select a **Trace mode**. A **One shot mode** trace ends when the hardware trace buffer is full. A **Continuous trace** ends when the program finishes (ends at a break instruction) or when it is stopped explicitly by the user.
- 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.

TASKING Embedded Profiler User Guide

- In the **Core index** field, select the TriCore core for which you want to run the analysis.
- For emulation devices only, enter a **Trace memory tile range**. Trace memory of emulation devices consists of a consecutive number of tiles. Select the first and last tile index you want to use for trace memory.
- Click **Start**.

The analysis starts. After the analysis is finished the result is present in the project tree.

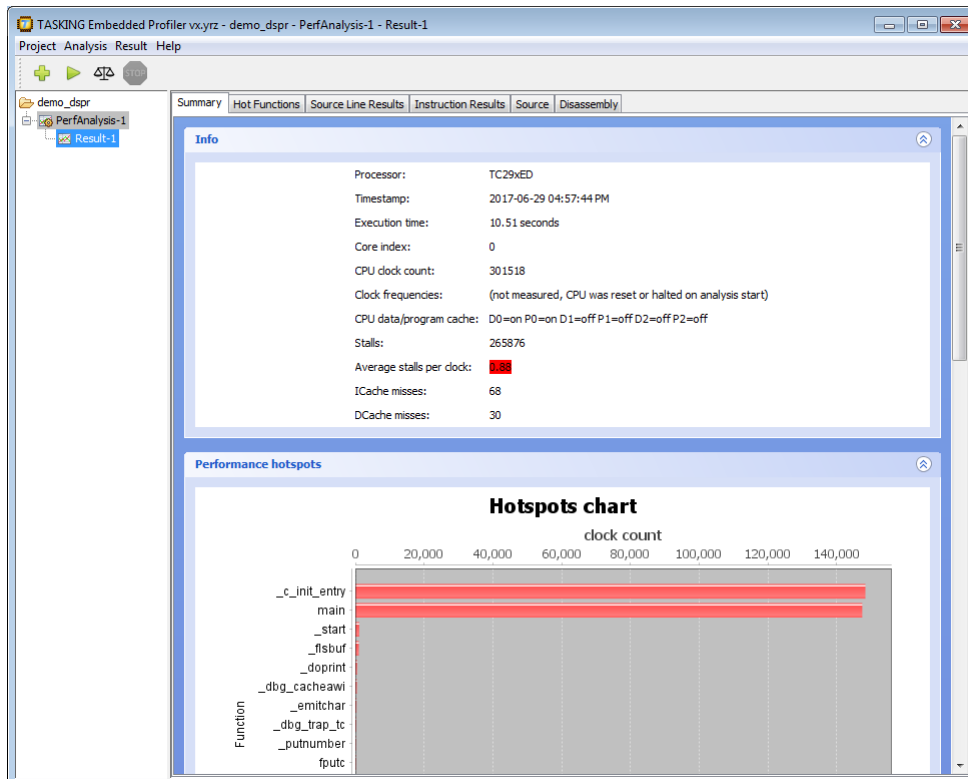


- If a Windows Security Alert appears that the firewall has blocked some features, select a network type and click **Allow access**.

Inspect the result of the Performance analysis

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

The result appears in several tabs.



- On the **Summary** tab, notice the high number of **Average stalls per clock** (0.88).
- On the **Hot Functions** tab, notice the high number of **Stalls** with functions `_c_init_entry` and `main`.

TASKING Embedded Profiler vx.yz - demo_dspr - PerfAnalysis-1 - Result-1

Project Analysis Result Help

Summary Hot Functions Source Line Results Instruction Results Source Disassembly

Funcnt...	Sourc...	Funcnt...	Clocks	% Of...	Clock...	Entries	Avg. ...	Max ...	Min Cl...	Jitter...	Branc...	ICac...	DCac...	Stalls
_c_init_...	0x8000...	148364	49.21	148364	1	148364	148364	148364	0	2	4	16	131727	
main	.. \cstart.c	0x8000...	147452	48.90	150740	1	147452	147452	0	0	3	0	131100	
_start	.. \cstart.c	0x8000...	1130	0.37	300652	1	1130	1130	0	1	11	0	683	
_fsbuf	0x8000...	1014	0.34	1422	27	37	68	34	34	2	6	2	16	
_doprint	0x8000...	516	0.17	3062	2	258	412	104	308	3	18	1	219	
_dbg_c...	dbg_tra...	0x8000...	454	0.15	454	8	56	86	16	70	0	0	23	
_emitchar	_doprin...	0x8000...	312	0.10	312	27	11	44	8	36	0	1	2	
_dbg_tr...	0x8000...	264	0.09	718	6	44	88	20	68	0	4	0	25	
_puthu...	_doprin...	0x8000...	188	0.06	350	1	188	188	188	0	8	0	144	
fputc	0x8000...	178	0.06	178	27	6	22	6	16	0	1	0	0	
_jo_putc	0x8000...	158	0.05	158	27	5	30	4	26	0	1	0	2	
clock	0x8000...	156	0.05	156	2	78	92	64	28	0	0	0	48	
_ltoa	_doprin...	0x8000...	122	0.04	122	1	122	122	122	0	1	0	53	
fclose	0x8000...	118	0.04	650	3	39	40	38	2	0	1	1	35	
.cocofu...	.. \cstart.c	0x8000...	98	0.03	98	2	49	52	46	6	1	0	16	
_fflush	0x8000...	98	0.03	98	3	32	52	22	30	2	1	0	17	
_dbg_trap	0x8000...	98	0.03	98	6	16	20	4	16	0	0	5	15	
_putstring	_doprin...	0x8000...	92	0.03	406	1	92	92	92	0	1	2	91	
.cocofu...	.. \cstart.c	0x8000...	84	0.03	84	2	42	48	36	12	0	0	19	
.cocofu...	.. \cstart.c	0x8000...	78	0.03	78	2	39	40	38	2	0	0	19	
_dodose	0x8000...	78	0.03	728	1	78	78	78	0	2	0	0	57	
.cocofu...	.. \cstart.c	0x8000...	76	0.03	76	2	38	46	30	16	1	0	17	
printf	0x8000...	70	0.02	3132	2	35	50	20	30	0	3	0	23	
strlen	0x8000...	64	0.02	64	2	32	34	30	4	0	0	0	15	
.cocofu...	.. \cstart.c	0x8000...	50	0.02	50	1	50	50	50	0	0	0	14	
.cocofu...	.. \cstart.c	0x8000...	50	0.02	50	5	10	24	6	18	1	0	0	
host...	.. \cstart.c	0x8000...	42	0.01	408	2	21	30	12	18	0	2	1	26
_c_init	0x8000...	32	0.01	32	1	32	32	32	0	0	1	1	7	
_host_c...	.. \cstart.c	0x8000...	32	0.01	434	3	10	12	10	2	0	0	3	4
_init_sp	.. \cstart.c	0x8000...	18	0.01	18	1	18	18	18	0	0	0	9	
_weaks...	0x8000...	12	0.00	12	1	12	12	12	0	0	0	0	8	
.cocofu...	_doprin...	0x8000...	10	0.00	10	1	10	10	10	0	0	0	0	
_exit	0x8000...	10	0.00	10	1	10	10	10	0	0	0	0	0	
exit	0x8000...	0	0.00	0	1	0	0	0	0	0	0	0	0	

- Double-click on `main`.

The Source tab opens.

TASKING Embedded Profiler User Guide

The screenshot displays the TASKING Embedded Profiler interface for a project named 'demo_dspr'. The main window shows the 'Source' tab with the following C code:

```
17 #define ARRAY_SIZE (16 * 1024)
18
19
20 // 0 = original
21 // 1 = problem fixed
22 #define FIXED 0
23
24
25 #if !FIXED
26
27 // this is the original line
28 // x[] is by default allocated in DSPR2
29 volatile int x[ARRAY_SIZE];
30
31 #else
32
33 // this is the fixed line
34 // we allocate x[] in DSPR0 to avoid the penalty in stalls
35 volatile int __private0 x[ARRAY_SIZE];
36
37 #endif
38
39 int main(void)
40 {
41     printf("Start\n");
42     clock_t clockstart = clock();
43
44     for (int i = 0; i < ARRAY_SIZE; ++i)
45     {
46         x[i] = 1;
47     }
48
49     int duration = (int) (clock() - clockstart);
50     printf("duration %i ticks\n", duration);
51 }
52
53
```

The right-hand side of the interface shows a performance metrics table with the following columns: Clods, Branch Misses, ICache Misses, DCache Misses, and Stalls. The data is as follows:

Line	Clods	Branch Misses	ICache Misses	DCache Misses	Stalls
1	0	0	1	0	19
21	0	0	0	0	0
1	0	0	1	0	16
45	155605	0	1	0	131040
11	0	0	0	0	7
1	0	0	0	0	0
14	0	0	0	0	7
2	0	0	0	0	6

5. Notice the high number of stalls is in the `for` loop.
6. Enable **Show disassembly** on the **Source** tab to show disassembly intermixed with the source lines, or open the **Disassembly** tab. Notice that the stalls are related to memory access.

The screenshot shows the TASKING Embedded Profiler interface for a performance analysis. The main window displays assembly code with columns for Clocks, Branches, ICache, DCache, and Stalls. The row for instruction 0x8000DE2 is highlighted in yellow, showing 147409 clocks and 131029 stalls.

Address	Instruction	Clocks	Branch...	ICache...	DCach...	Stalls
0x8000DA6	jlt d2,d1,0x8000d80	49	0	1	0	21
0x8000DAA	mtcr #0xfe38,d0	1	0	0	0	0
0x8000DAE	isync	10	0	0	0	0
0x8000DB2	call 0x80001d4	1	0	0	0	3
0x8000DB6	mov d4,#0x0	1	0	0	0	3
0x8000DB8	mov.a a4,#0x0	24	0	0	0	0
0x8000DBA	call 0x8000dc4	1	0	0	0	0
0x8000DBE	mov d4,d2	1	0	0	0	3
0x8000DC0	j 0x8000dfe	1	0	0	0	0
main:						
0x8000DC4	sub.a sp,#0x8	1	0	1	0	19
0x8000DC6	lea a4,0x8000024	20	0	0	0	0
0x8000DCA	call 0x8000e5c	1	0	0	0	0
0x8000DCE	call 0x8000bc0	1	0	1	0	16
0x8000DD2	mov d8,d2	8	0	0	0	7
0x8000DD4	movh.a a15,#0x5000	2	0	0	0	0
0x8000DD8	lea a15,[a15]0x5000	1	0	0	0	0
0x8000DDC	mov d15,#0x1	1	0	0	0	0
0x8000DDE	lea a2,0x3fff	2	0	0	0	0
0x8000DE2	st.w [a15]0x4,d15	147409	0	1	0	131029
0x8000DE4	loop a2,0x8000de2	8193	0	0	0	11
0x8000DE6	call 0x8000bc0	1	0	0	0	0
0x8000DEA	sub d2,d8	8	0	0	0	7
0x8000DEC	st.w [sp],d2	2	0	0	0	0
0x8000DEE	movh.a a4,#0x8000	1	0	0	0	0
0x8000DF2	lea a4,[a4]0xf1a	1	0	0	0	0
0x8000DF6	call 0x8000e5c	2	0	0	0	0
0x8000DFA	mov d2,#0x0	1	0	0	0	6
0x8000DFC	ret	1	0	0	0	0
exit:						
0x8000DFE	mov d15,d4	10	0	0	0	0
0x8000E00	call 0x80000f6	1	0	1	0	15
0x8000E04	call 0x8000396	1	0	0	0	8
0x8000E08	mov d4,d15	4	0	0	0	3
0x8000E0A	call 0x80006cc	1	0	0	0	3
0x8000E0E	mov d4,d15	1	0	0	0	3
0x8000E10	j 0x80000f6	1	0	0	0	0

Create and run a Memory access analysis

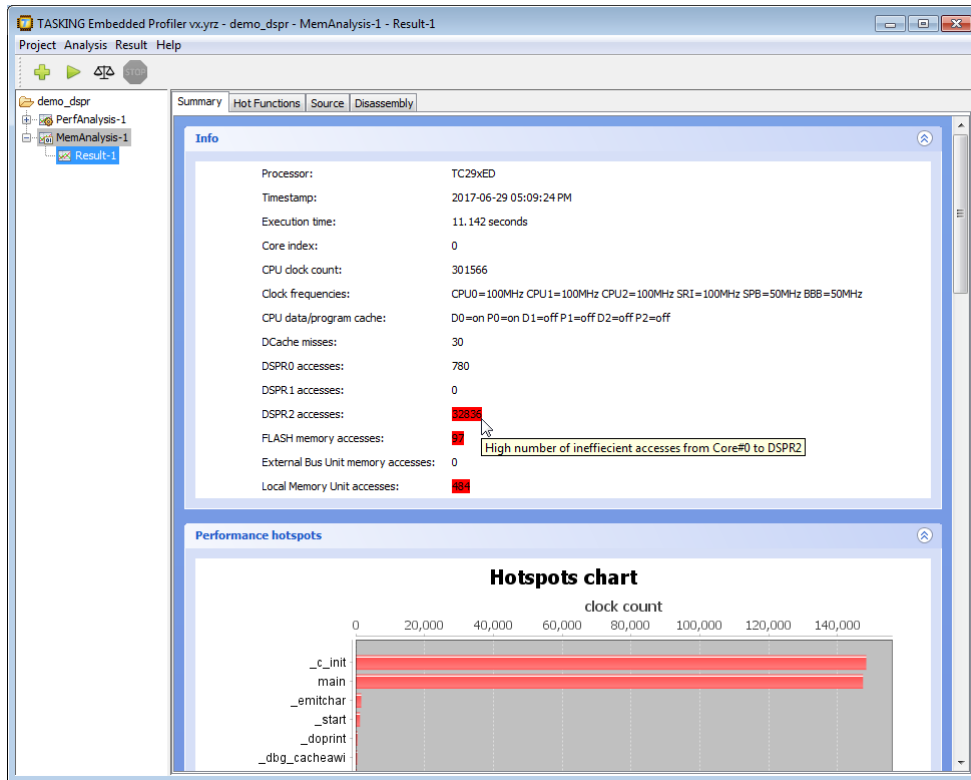
1. Repeat the steps described above with *Create a Performance analysis*, but in Step 2 select **Memory access analysis**.
2. Run the new analysis similar as described above with *Run the analysis*.

Inspect the result of the Memory access analysis

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

The result appears in several tabs.

TASKING Embedded Profiler User Guide



2. On the **Summary** tab, notice the high number of **DSPR2 accesses** (32836). When you hover the mouse over a value, a context sensitive help box with additional information can appear.
3. Scroll down to the **Data access intensive functions** and notice that `_c_init` and `main` both access variable `x` in DSPR2

Summary Hot Functions Source Disassembly

demo_dspr
PerfAnalysis-1
MemAnalysis-1
Result-1

Data access intensive functions

_c_init

Variable	Region	Access	Count	% Cache Misses
x	DSPr2	W	16384	0
Unidentified access	PFLASH0	R	70	0
_job	LMU	W	50	0
Unidentified access	DSPr2	W	40	0
Unidentified access	DSPr0	W	37	0
_dbg_request	LMU	W	5	0
Unidentified access	DSPr0	R	3	0

main

Variable	Region	Access	Count	% Cache Misses
x	DSPr2	W	16385	0
Unidentified access	DSPr0	W	5	0
Unidentified access	DSPr0	R	3	0

_emitchar

Variable	Region	Access	Count	% Cache Misses
_job	LMU	R	271	0
Unidentified access	DSPr0	W	122	0
_job	LMU	W	81	0
Unidentified access	DSPr0	R	75	0
x	DSPr0	R	54	0
x	DSPr0	W	29	0
Unidentified access	DSPr2	W	27	0

_doprint

Variable	Region	Access	Count	% Cache Misses
Unidentified access	PFLASH0	R	27	0
Unidentified access	DSPr0	W	22	0
Unidentified access	DSPr0	R	6	0
x	DSPr0	W	3	0
x	DSPr0	R	3	0

4. Hover the mouse over DSPr2 in main.

A context sensitive help box appears with a suggestion to solve the problem.

main

Variable	Region	Access	Count	% Cache Misses
x	DSPr2	W	16385	0
Unidentified access	DSPr0	W	5	0
Unidentified access	DSPr0	R	3	0

Scratch-Pad memory of Core#2. Access to this memory region is ineffective from Core#0. Consider moving data to scratch-pad memory of accessing core.

3.3. Fix the Problem

Now that we have analyzed the problem, we can fix it.

1. In the TASKING TriCore Eclipse IDE, double-click on the source file demo_dspr.c.

The file will be opened in the source editor.

2. Change the following source line:

```
#define FIXED 0
```

TASKING Embedded Profiler User Guide

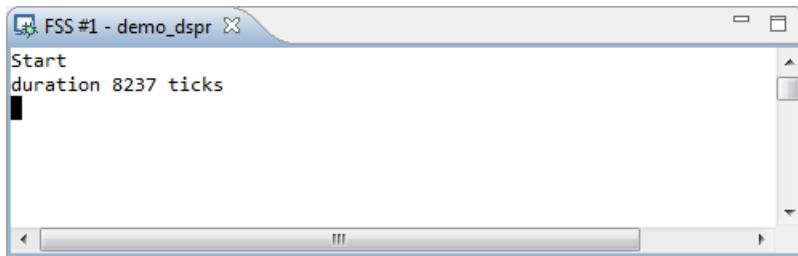
into:

```
#define FIXED 1
```

3. From the **Project** menu, select **Rebuild demo_dspr** (🔧).
4. From the **Debug** menu, select **Debug project** (🔧).
5. From the **Debug** menu, select **Resume** (▶) to run the application on the target board.

The output of the application appears in the FSS (File System Simulation) view.

6. Inspect the FSS view and notice the number of ticks has reduced significantly.



7. From the **Debug** menu, select **Terminate** (■) to stop the debugging session. This is necessary to free the connection with the target board.

3.4. Verify Fix in TASKING Embedded Profiler

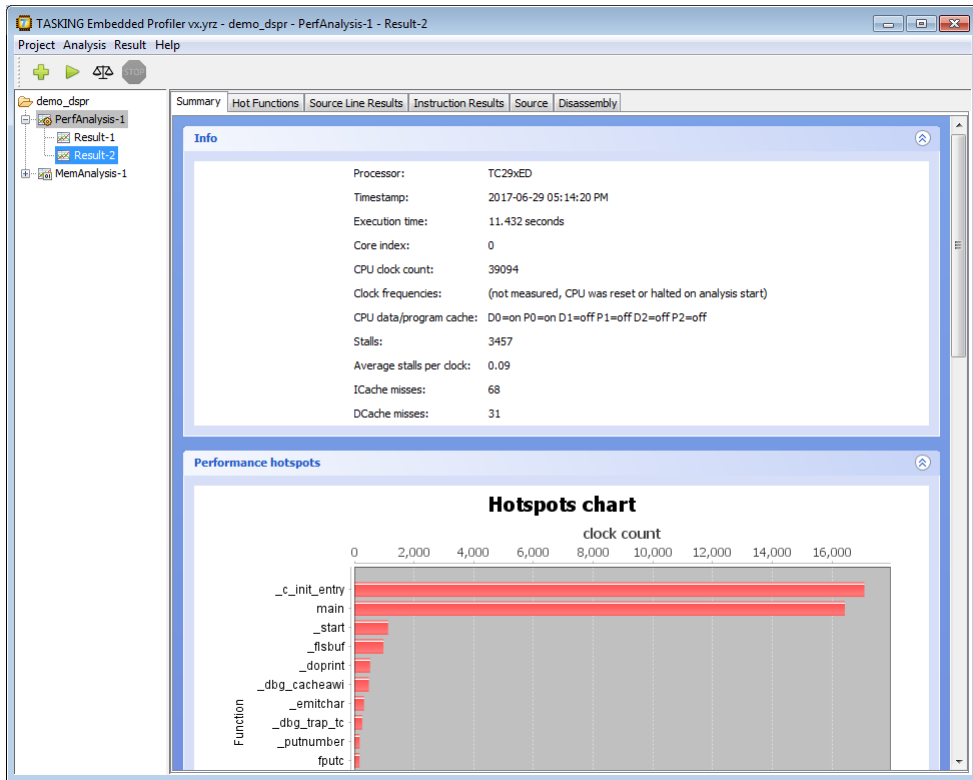
Now that we have fixed the problem, we can use the TASKING Embedded Profiler to rerun both the Performance analysis and the Memory access analysis mentioned in [Section 3.2, Analyze Project in TASKING Embedded Profiler](#) and see the new results of the analyses.

Rerun the Performance analysis and inspect the result

1. In the TASKING Embedded Profiler, select `PerfAnalysis-1`.
2. From the **Analysis** menu, select **Run Analysis**.
3. Click **Start**.

This creates a Result-2.

4. Select `Result-2` and notice that on the **Summary** tab, the number of **Average stalls per clock** has reduced significantly from 0.88 to 0.09.



5. Also inspect the other tabs yourself to see the results.

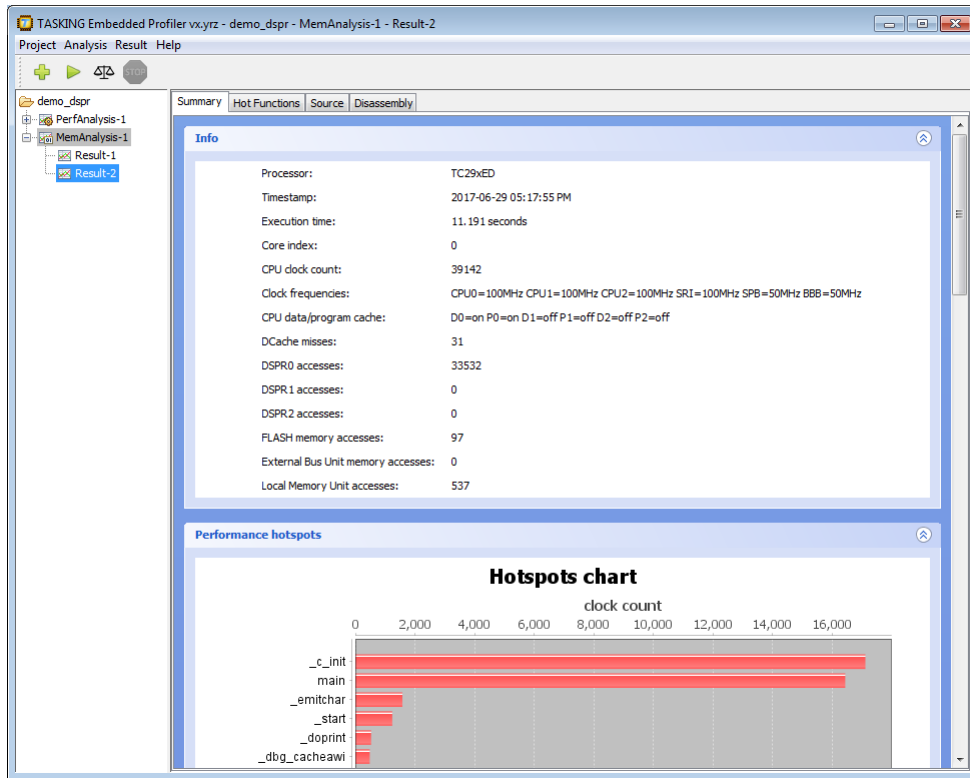
Rerun the Memory access analysis

1. In the TASKING Embedded Profiler, select MemAnalysis-1.
2. From the **Analysis** menu, select **Run Analysis**.
3. Click **Start**.

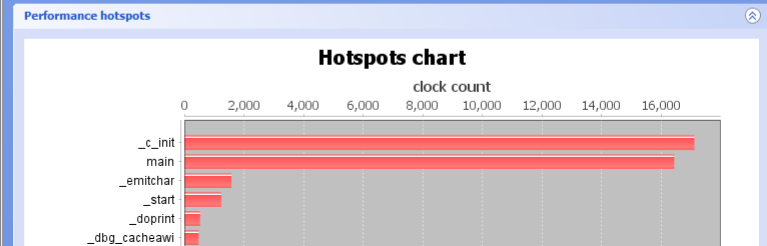
This creates a Result-2.

4. Select Result-2 and notice that on the **Summary** tab, the accesses are now in DSPR0. And notice that main is no longer listed in the list of Data access intensive functions.

TASKING Embedded Profiler User Guide



Info	
Processor:	TC29MED
Timestamp:	2017-06-29 05:17:55 PM
Execution time:	11.191 seconds
Core index:	0
CPU clock count:	39142
Clock frequencies:	CPU0=100MHz CPU1=100MHz CPU2=100MHz SRI=100MHz SPB=50MHz BBB=50MHz
CPU data/program cache:	D0=on P0=on D1=off P1=off D2=off P2=off
DCache misses:	31
DSPR0 accesses:	33532
DSPR1 accesses:	0
DSPR2 accesses:	0
FLASH memory accesses:	97
External Bus Unit memory accesses:	0
Local Memory Unit accesses:	537



The screenshot shows the TASKING Embedded Profiler interface. The main window displays the results of a memory analysis for 'demo_dspr - MemAnalysis-1 - Result-2'. The 'Data access intensive functions' section is expanded, showing four tables of data access statistics:

_emitchar				
Variable	Region	Access	Count	% Cache Misses
_job	LMJ	R	261	0
Unidentified access	DSPRO	W	139	0
Unidentified access	DSPRO	R	124	0
_job	LMJ	W	78	0
Unidentified access	LMJ	W	26	0

_c_init				
Variable	Region	Access	Count	% Cache Misses
x	DSPRO	W	16384	0
Unidentified access	PFLASH0	R	70	0
_job	LMJ	W	50	0
Unidentified access	LMJ	W	40	0
Unidentified access	DSPRO	W	36	0
_dbg_request	LMJ	W	5	0
Unidentified access	DSPRO	R	3	0

_doprint				
Variable	Region	Access	Count	% Cache Misses
Unidentified access	DSPRO	W	32	0
Unidentified access	PFLASH0	R	27	0
Unidentified access	DSPRO	R	9	0

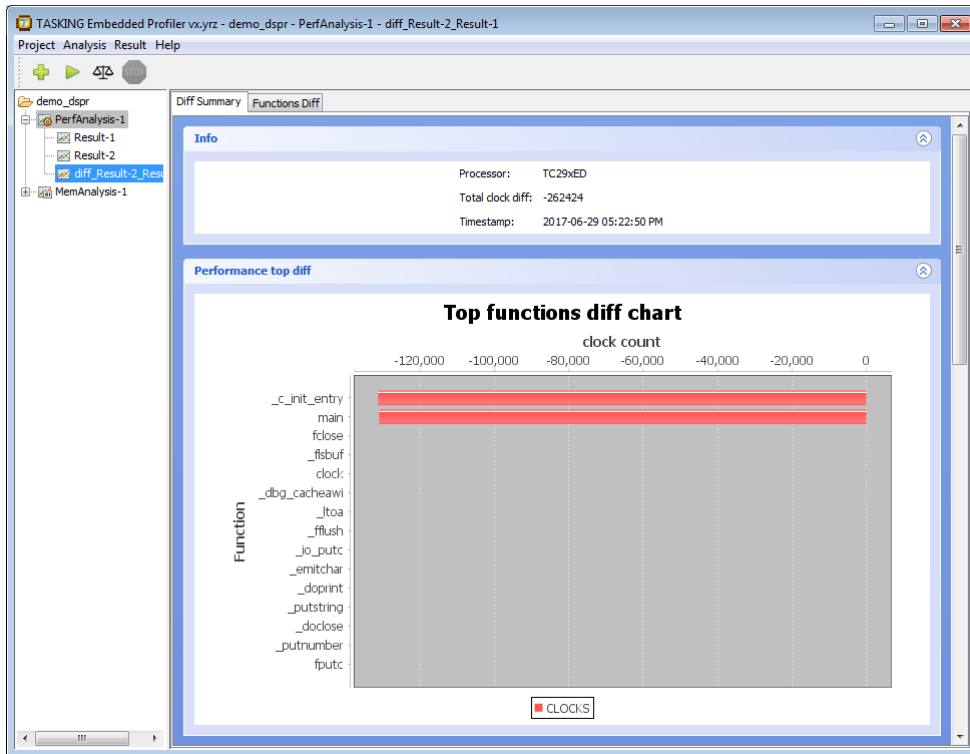
_fflush				
Variable	Region	Access	Count	% Cache Misses
Unidentified access	DSPRO	W	14	0
_job	LMJ	R	10	0
Unidentified access	DSPRO	R	9	0
_job	LMJ	W	4	0

3.5. Compare Results

The Embedded Profiler 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 TASKING Embedded Profiler, select a result. For example, `Result-2` of `PerfAnalysis-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".

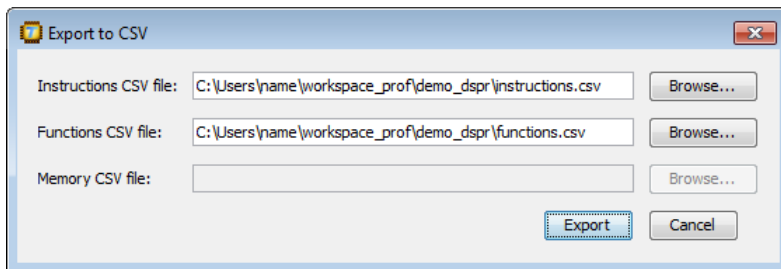


3.6. Export Results

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 TASKING Embedded Profiler, select a result. For example, `Result-1` of `PerfAnalysis-1`.
2. From the **Result** menu, select **Export to CSV**.

The Export to CSV dialog appears.



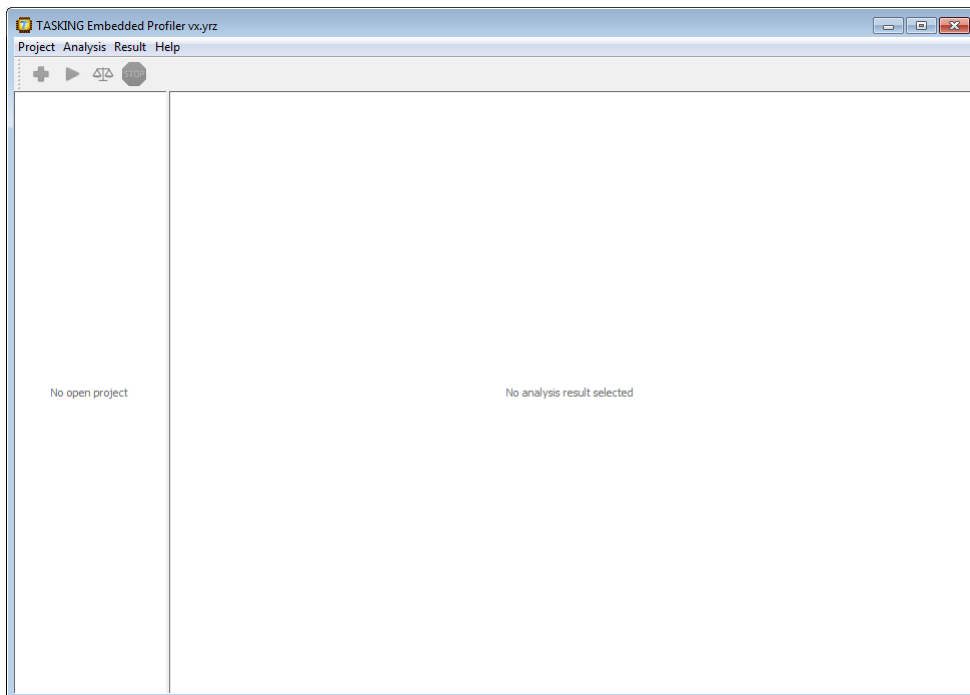
3. Enter the filename(s) and click **Export**.

Chapter 4. Using the TASKING Embedded Profiler

You can run the TASKING Embedded Profiler 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.

4.1. Run the Embedded Profiler in Interactive Mode

To start the Embedded Profiler select **Embedded Profiler** 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 `.EmbProf.`

TASKING Embedded Profiler User Guide

The steps to:

- create a project
- create an analysis
- run an analysis

are described in [Section 3.2, Analyze Project in TASKING Embedded Profiler](#).

See also [Section 3.5, Compare Results](#) and [Section 3.6, Export Results](#). For details about the Results see [Chapter 5, Reference](#).

4.2. Run the Embedded Profiler from the Command Line

To run the Embedded Profiler from the command line use the **EmbProfCmd** batch file in a Windows Command Prompt. Enter the following command to see the usage:

```
EmbProfCmd --help
```

The general invocation syntax is:

```
EmbProfCmd options project.EmbProf
```

where, *project.EmbProf* refers to an existing Embedded Profiler 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	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	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.
--run=analysis	This option allows you to run an existing analysis.
--server=hostname	This option allows you to specify the device server name. If you omit this option, the default is localhost.
--version	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:

```
EmbProfCmd --run=analysis project.EmbProf
```

where, `project.EmbProf` refers to an existing Embedded Profiler 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:

```
EmbProfCmd --run=analysis --compare=result project.EmbProf
```

where, `project.EmbProf` refers to an existing Embedded Profiler project file.

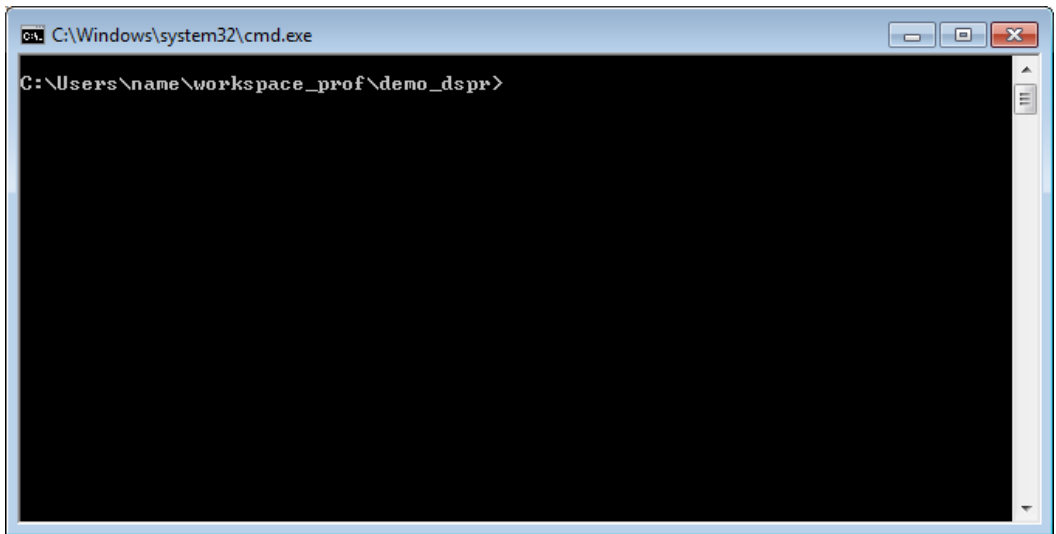
4.2.1. Command Line Tutorial

In this section we use tutorial `demo_dspr` with the delivered `demo_dspr.EmbProf` to illustrate the use of the command line options of the Embedded Profiler.

Prepare command line

Before you run the Embedded Profiler 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_dspr`.



2. Add the executable directory of the Embedded Profiler to the environment variable PATH. The executable directory is the `profiler` directory in the installation directory. Substitute `version` with the correct version number.

```
set PATH=%PATH%;"C:\Program Files (x86)\TASKING\prof version\profiler"
```

Command line examples

1. To run a performance analysis in `demo_dspr` using one shot trace mode, enter:

```
EmbProfCmd --run=PerfAnalysis demo_dspr.EmbProf
```

The results are exported to the CSV files `functions.csv` and `instructions.csv`. You can inspect these files 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_dspr.EmbProf` file.

2. To run a performance analysis in `demo_dspr` using one shot trace mode and compare the results with `original`, enter:

```
EmbProfCmd --run=PerfAnalysis --compare=original demo_dspr.EmbProf
```

The results of the comparison are exported to the CSV file `functions.csv`. If all value fields are zero, this indicates that the results are identical. This should be the case with this example.

3. To run a performance analysis in `demo_dspr` using one shot trace mode and compare the results with `fixed`, enter:

```
EmbProfCmd --run=PerfAnalysis --compare=fixed demo_dspr.EmbProf
```

The results of the comparison are exported to the CSV file `functions.csv`. 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.

4. To run an analysis using continuous trace mode use option **--continuous**. Be aware that this mode requires that the application ends and does not contain endless `while` loops. Otherwise an analysis run will not end.

```
EmbProfCmd --run=PerfAnalysis --compare=fixed --continuous  
demo_dspr.EmbProf
```

5. To run an analysis on a specific core, use option **--core=core-nr**. For the TC29x derivative you can use the values 0, 1 and 2. Be aware that a core needs to be enabled in the startup code of the application. Otherwise the analysis run will not terminate.

```
EmbProfCmd --run=PerfAnalysis --compare=fixed --continuous  
--core=0 demo_dspr.EmbProf
```

6. 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`.

```
EmbProfCmd --run=PerfAnalysis --compare=fixed --continuous  
--core=0 --server=myservername demo_dspr.EmbProf
```


Chapter 5. Reference

Every analysis result shows a number of tabs with information. What information is shown depends on the type of the analysis: performance analysis, memory access analysis or function-level analysis.

This chapter contains an overview of all the fields and columns in an analysis result output.

5.1. Summary Tab

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

Performance analysis

- Info
- Performance hotspots
- ICache misses
- DCache misses

Memory access analysis

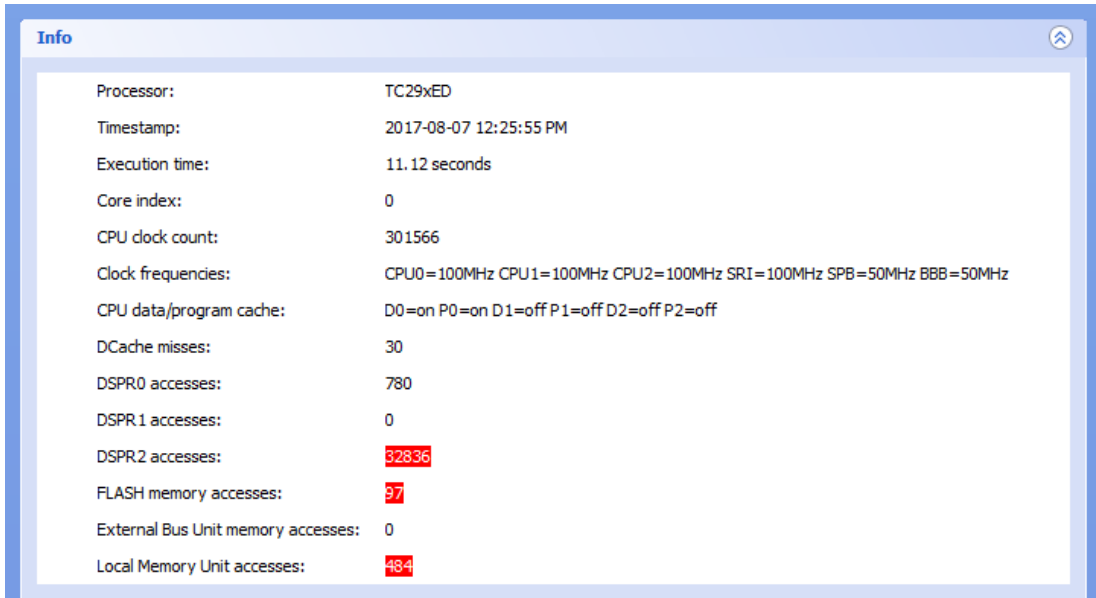
- Info
- Performance hotspots
- Data access intensive functions
- Memory access conflicts
- DCache misses

Function-level analysis

- Info
- Performance hotspots

5.1.1. Info

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



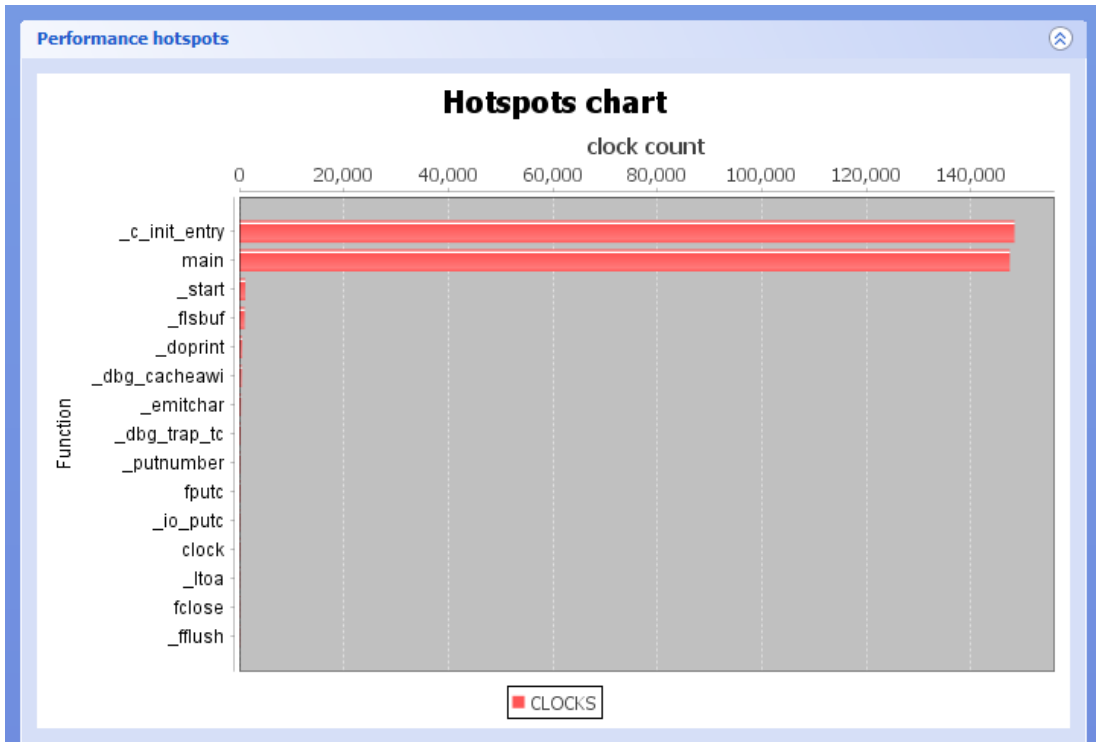
Information	Description	Perf Analysis	Mem Analysis	Func Analysis
Processor	The name of the selected processor device	✓	✓	✓
Timestamp	The date and time the analysis was run	✓	✓	✓
Execution time	The time it took on the PC to run the analysis	✓	✓	✓
Core index	The TriCore core (0, 1, 2, ...) the analysis was run for	✓	✓	✓
CPU clock count	The number of CPU clock cycles on the board it took to run the analysis	✓	✓	✓
Clock frequencies	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. D0=on means CPU0.DCACHE is enabled, P1=off means CPU1.PCACHE is disabled. The values are read at the start of the analysis before any reset.	✓	✓	✓

Information	Description	Perf Analysis	Mem Analysis	Func Analysis
Stalls	The number of clock cycles the CPU stalls on branch misses, ICache misses and/or DCache misses	✓		
Average stalls per clock	The average of stalls / CPU clock count	✓		
ICache misses	The number of failed attempts to read or write instructions from the instruction cache (ICache)	✓		
DCache misses	The number of failed attempts to read or write data from the data cache (DCache)	✓	✓	
DSPR0 accesses	The number of read or write accesses to Data Scratchpad RAM 0		✓	
DSPR1 accesses	The number of read or write accesses to Data Scratchpad RAM 1		✓	
DSPR2 accesses	The number of read or write accesses to Data Scratchpad RAM 2		✓	
FLASH memory accesses	The number of read or write accesses to flash memory		✓	
External Bus Unit memory accesses	The number of read or write accesses to the EBU		✓	
Local Memory Unit accesses	The number of read or write accesses to the LMU		✓	

Items that are marked red are high values that may be improved. Hover the mouse over a value to see additional information.

5.1.2. Performance Hotspots

The **Performance hotspots** part of the Summary tab show a Hotspots chart. It show the functions with the highest clock count. This chart is available for all analysis types. As you can see in the following example, most of the time is spent in the functions `_c_init` and `main`.



If you double-click on a function, the Source tab opens at the selected function.

5.1.3. Data Access Intensive Functions

The **Data access intensive functions** part of the Summary tab shows the functions with the highest number of data accesses to memory. This chart is available for memory access analyses only.

The first column is the name of the global variable, if the address is associated with a variable, otherwise "Unidentified access" is shown. The second column is the name of the memory. The third column shows the type of access read (R) or write (W). The fourth column shows the total number of accesses. The fifth column shows the approximate cache miss rate for this specific access.

Hover the mouse over a value to see additional information.

Data access intensive functions				
_c_init				
Variable	Region	Access	Count	% Cache Misses
x	DSPR2	W	16384	0
Unidentified access	PFLASH0	R	70	0
_job	LMU	W	50	0
Unidentified access	DSPR2	W	40	0
Unidentified access	DSPR0	W	37	0
_dbg_request	LMU	W	5	0
Unidentified access	DSPR0	R	3	0
main				
Variable	Region	Access	Count	% Cache Misses
x	DSPR2	W	16385	0
Unidentified access	DSPR0	W	5	0
Unidentified access	DSPR0	R	3	0
_emitchar				
Variable	Region	Access	Count	% Cache Misses
_job	LMU	R	271	0
Unidentified access	DSPR0	W	122	0

5.1.4. Memory Access Conflicts

The **Memory access conflicts** part of the Summary tab shows the conflicts where two variables from different cores access the same memory at the same time. This is called concurrent access. The `demo_concurrent` tutorial delivered with the product demonstrates this problem. This chart is available for memory access analyses only.

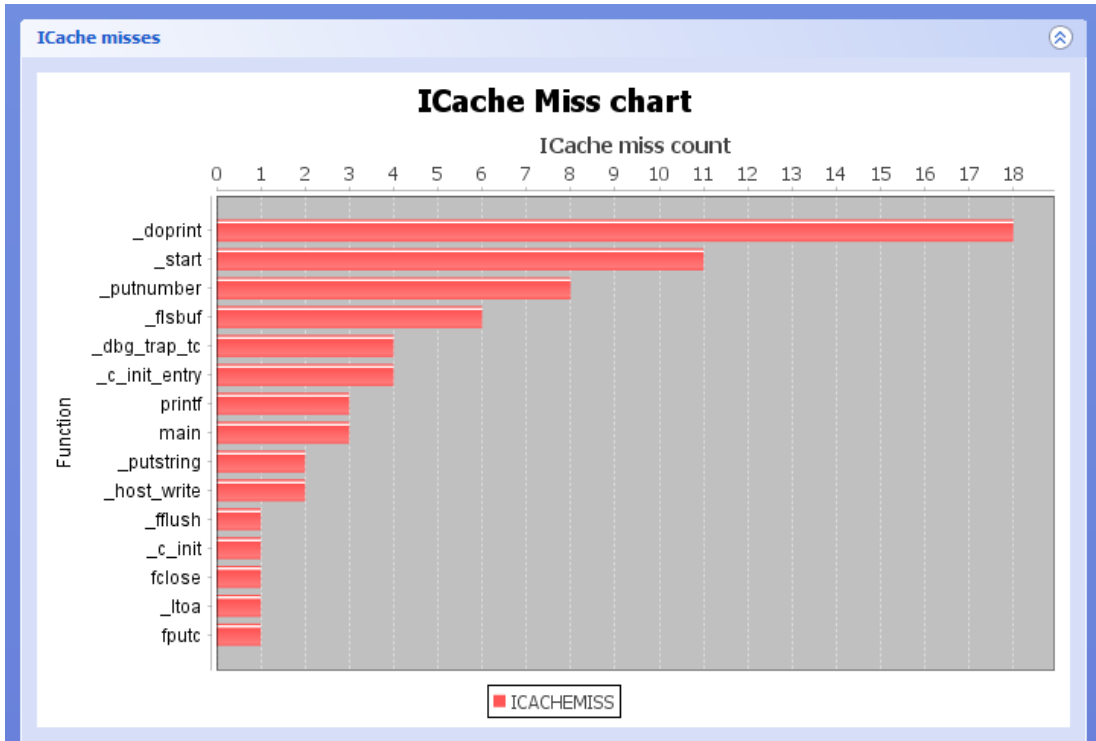
The first column is the name of the first global variable that accesses the memory. The second column shows the type of access read (R) or write (W) of the first variable. The third column is the name of the second global variable that accesses the memory and causes the conflict. The fourth column shows the type of access read (R) or write (W) of the second variable. The fifth column shows the core from which the conflicting access originated. The sixth column shows the total number of access conflicts.

Hover the mouse over a value to see additional information.

Memory access conflicts					
main					
Variable 1	Access 1	variable 2	Access 2	Core	Count
var0	W	var1	W	CPU1	997
var0	W	var2	R	CPU2	1
var0	W	var2	W	CPU2	1

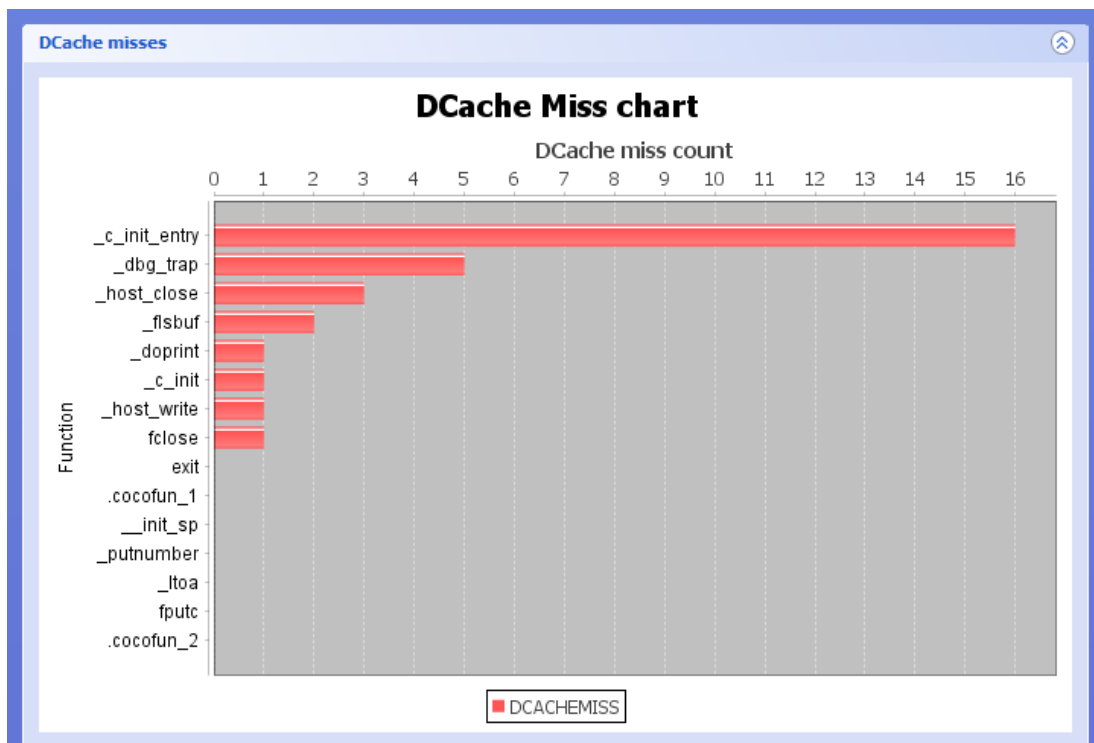
5.1.5. ICache Misses

The **ICache misses** part of the Summary tab show an ICache Miss chart. It show the functions with the highest number of instruction cache (ICache) misses. This chart is available for performance analyses only.



5.1.6. DCache Misses

The **DCache misses** part of the Summary tab show a DCache Miss chart. It show the functions with the highest number of data cache (DCache) misses. This chart is available for performance analyses and memory access analyses.



5.2. Hot Functions Tab

The Hot Functions tab shows a list with all the measured functions. This tab is available in all analysis types. The performance analysis contains the most columns. Click on a column to sort the list according to the information in that column. If you double-click on a function, the Source tab opens at the selected function. If no source lines can be displayed, the Disassembly tab opens. Hover the mouse over a column to see additional information.

The Hot Functions tab contains the following information:

Column	Description	Perf Analysis	Mem Analysis	Func Analysis
Function Name	The name of the measured function	✓	✓	✓
Source File	The relative path to the source file as stored in the application ELF file	✓	✓	✓
Function Address	The address of the function in the application ELF file	✓	✓	✓
Clocks	The total number of CPU clocks spent in the function	✓	✓	✓

Column	Description	Perf Analysis	Mem Analysis	Func Analysis
% Of Total Time	The application execution time spent in the function as a percentage of the total application execution time	✓	✓	✓
Clocks With Children	The total number of CPU clocks spent in the function and call tree descendents	✓	✓	✓
Entries	The total number of times the function is called	✓	✓	✓
Avg. clocks/Entry	The average number of CPU clocks spent in a function per function entry	✓	✓	✓
Max Clocks/Entry	The highest number of CPU clocks spent in a function per function entry	✓	✓	✓
Min Clocks/Entry	The lowest number of CPU clocks spent in a function per function entry	✓	✓	✓
Jitter/Entry	The difference between the highest and lowest number of CPU clocks spent in a function. This is the difference of the previous two columns.	✓	✓	✓
Branch Misses	The total number of branch misses	✓		
ICache Misses	The total number of instruction cache misses	✓		
DCache Misses	The total number of data cache misses	✓	✓	
Stalls	The total number of stalls due to memory access delays or pipeline hazards	✓		

5.3. Source Line Results Tab

The Source Line Results tab shows a list with all the source lines of the measured functions where branch misses, instruction cache misses, data cache misses and/or stalls appear. This tab is available for performance analyses only. Click on a column to sort the list according to the information in that column. Hover the mouse over a column to see additional information.

If you double-click on a row, the Source tab opens at the selected source line.

The Source Line Results tab contains the following information:

Column	Description
Position	The source line number, function name and relative path to the source file where the problem occurred
Clocks	The total number of CPU clocks spent on the source line
Branch Misses	The total number of branch misses
ICache Misses	The total number of instruction cache misses

Column	Description
DCache Misses	The total number of data cache misses
Stalls	The total number of stalls due to memory access delays or pipeline hazards

5.4. Instruction Results Tab

The Instruction Results tab shows a list with all the instructions of the measured functions where branch misses, instruction cache misses, data cache misses and/or stalls appear. This tab is available for performance analyses only. Click on a column to sort the list according to the information in that column. Hover the mouse over a column to see additional information.

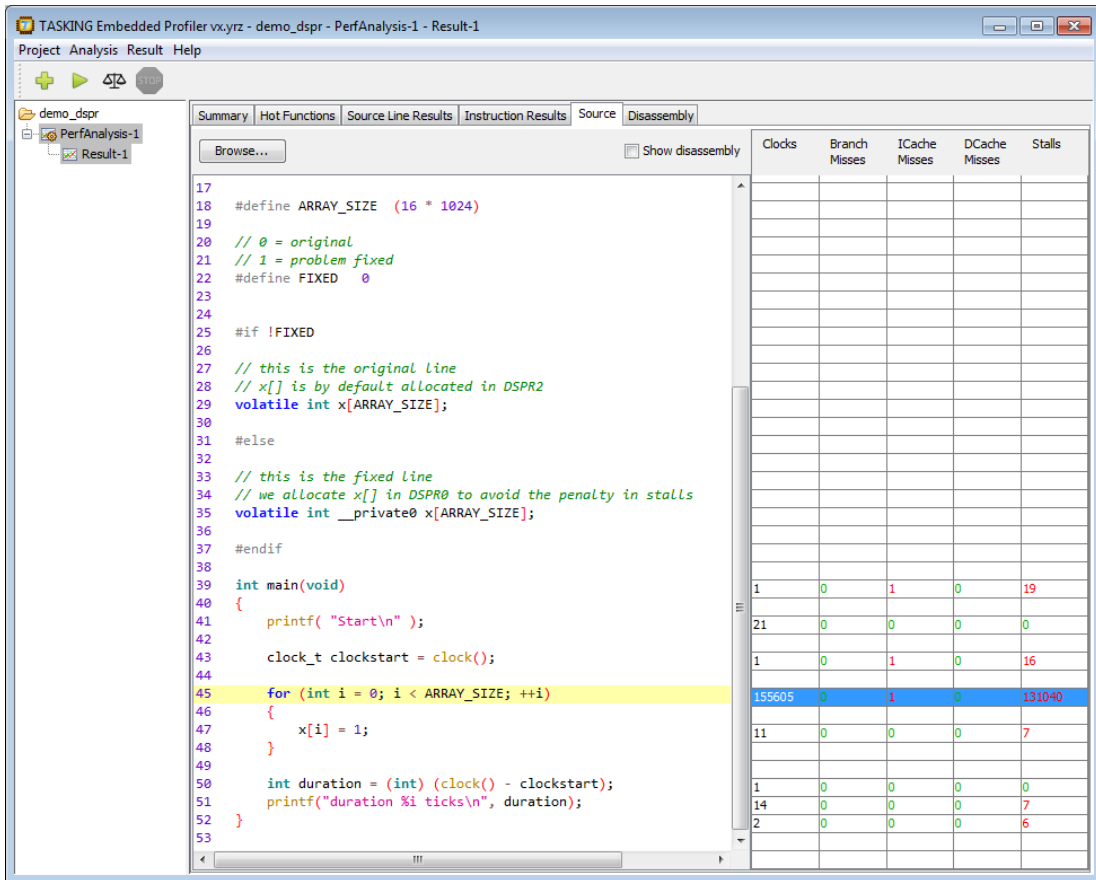
If you double-click on a row, the Disassembly tab opens at the selected instruction.

The Instruction Results tab contains the following information:

Column	Description
Address	The instruction address and function name where the problem occurred
Clocks	The total number of CPU clocks spent on the instruction
Branch Misses	The total number of branch misses
ICache Misses	The total number of instruction cache misses
DCache Misses	The total number of data cache misses
Stalls	The total number of stalls due to memory access delays or pipeline hazards

5.5. Source Tab

The Source tab shows the source code for the selected 'hot function'. For performance analyses only, trace data is also present grouped by source line.



The columns are the same as explained in [Section 5.3, Source Line Results Tab](#). Red values indicate a miss or a stall. Hover the mouse over a value to see additional information.

With the **Browse** button you can open another source file.

When you enable **Show disassembly**, the disassembly will be intermixed with the source lines.

5.6. Disassembly Tab

The Disassembly tab shows the instructions for the selected 'hot function'. For performance analyses only, trace data is also present grouped by instruction address.

TASKING Embedded Profiler vx.yrz - demo_dspr - PerfAnalysis-1 - Result-1

Project Analysis Result Help

demo_dspr

PerfAnalysis-1

Result-1

Summary Hot Functions Source Line Results Instruction Results Source Disassembly

	Clocks	Branch...	ICache...	DCach...	Stalls
0x8000DA6: jlt d2,d1,0x8000d80	49	0	1	0	21
0x8000DAA: mtc r #0xfe38,d0	1	0	0	0	0
0x8000DAE: isync	10	0	0	0	0
0x8000DB2: call 0x80001d4	1	0	0	0	3
0x8000DB6: mov d4,#0x0	1	0	0	0	3
0x8000DB8: mov.a a4,#0x0	24	0	0	0	0
0x8000DBA: call 0x8000dc4	1	0	0	0	0
0x8000DBE: mov d4,d2	1	0	0	0	3
0x8000DC0: j 0x8000dfe	1	0	0	0	0
main:					
0x8000DC4: sub.a sp,#0x8	1	0	1	0	19
0x8000DC6: lea a4,0x8000024	20	0	0	0	0
0x8000DCA: call 0x8000e5c	1	0	0	0	0
0x8000DCE: call 0x8000bc0	1	0	1	0	16
0x8000DD2: mov d8,d2	8	0	0	0	7
0x8000DD4: movh.a a15,#0x5000	2	0	0	0	0
0x8000DD8: lea a15,[a15]0x5000	1	0	0	0	0
0x8000DDC: mov d15,#0x1	1	0	0	0	0
0x8000DDE: lea a2,0x3fff	2	0	0	0	0
0x8000DE2: st.w [a15+]0x4,d15	147409	0	1	0	131029
0x8000DE4: loop a2,0x8000de2	8193	0	0	0	11
0x8000DE6: call 0x8000bc0	1	0	0	0	0
0x8000DEA: sub d2,d8	8	0	0	0	7
0x8000DEC: st.w [sp],d2	2	0	0	0	0
0x8000DEE: movh.a a4,#0x8000	1	0	0	0	0
0x8000DF2: lea a4,[a4]0xf1a	1	0	0	0	0
0x8000DF6: call 0x8000e5c	2	0	0	0	0
0x8000DFA: mov d2,#0x0	1	0	0	0	6
0x8000DFC: ret	1	0	0	0	0
exit:					
0x8000DFE: mov d15,d4	10	0	0	0	0
0x8000E00: call 0x80000f6	1	0	1	0	15
0x8000E04: call 0x8000396	1	0	0	0	8
0x8000E08: mov d4,d15	4	0	0	0	3
0x8000E0A: call 0x80006cc	1	0	0	0	3
0x8000E0E: mov d4,d15	1	0	0	0	3
0x8000E10: j 0x80000f8	1	0	0	0	0

The columns are the same as explained in [Section 5.4, Instruction Results Tab](#). Red values indicate a miss or a stall. Hover the mouse over a value to see additional information.

Note that due to hardware constraints, a miss or a stall cannot always be linked to the exact assembly instruction.

