CodeWarrior Development Studio for StarCore SC3900FP DSP Architectures Tracing and Analysis Tools User Guide

Document Number: CWSCSWAUG
Rev. 10.9.0, 11/2015

Contents

Section number　　Title　　Page

Chapter 1
Introduction

Chapter 2
Getting Started

Chapter 3
Collecting Data

Section number Title Page

Chapter 4
Setting Tracepoints

Section number Title Page

Chapter 5
Setting Counterpoints

Chapter 6
Performance Analysis for B4 Devices

Section number Title Page

Chapter 7
Launching Scripts

Chapter 8
Remote Launch API

Section number Title Page

Section number Title Page

Section number Title Page

Chapter 1
Introduction

The CodeWarrior Tracing and Analysis tool set to develop software for Freescale StarCore DSP processors.

This chapter explains:

- Release Notes
- About this Guide
- Accompanying Documentation

1.1 Release Notes

Release notes include information about new features, last-minute changes, bug fixes, incompatible elements, or other sections that may not be included in this manual.

You should read release notes before using the CodeWarrior IDE.

NOTE

The release notes for specific components of the CodeWarrior IDE are located in the `Release_Notes` folder in the CodeWarrior installation directory.

1.2 About this Guide

Each chapter of this manual describes a different area of software development.

The table below lists each chapter in the manual.

Table 1-1. Chapters in this Guide

Chapter	Description
Introduction	(this chapter)
Getting Started	Describes the analysis tool and the trace and profiling data
Collecting Data	Describes how to collect the data from simulator SC3900 and Qonverge targets
Setting Tracepoints	Describes how to collect selective data after setting the tracepoints
Setting Counterpoints	Describes how to count events between two points of the executed code
Performance Analysis for B4 Devices	Describes how to configure, collect, and analyze scenarios in the B4 devices.
Launching Scripts	Describes how to collect and export trace data automatically using remote launch.
Remote Launch API	Lists the classes and interface for the remote launch API.

1.3 Accompanying Documentation

The Documentation Roadmap page describes the documentation included in this version of CodeWarrior Development Studio for StarCore DSP Architectures.

You access Documentation Roadmap by:

- a shortcut link on the Desktop that the installer creates by default, or
- opening START_HERE.html in CWInstallDir\SC\Help.

Chapter 2
Getting Started

CodeWarrior Tracing and Analysis Tools provide visibility into an application as it runs on the simulator SC3900 and the Qonverge targets. This visibility can help you understand how your application runs, as well as identify operational problems.

Tracing and analysis results are accurate for O0 optimization level. For an optimization level higher than O0, the results that are provided by this tool can be affected. The affected results depends on the type of code and its optimization, whereas the results that are based on asm code, analysis are not affected by optimization level. The results based on source code and name of symbols can be affected by optimization level. For an accurate analysis of the libraries linked to your project, make sure to generate debugging information for them so that all the debug symbols are found and included in the statistics.

However, it is possible to obtain limited accuracy only for profiling trace(Trace, Timeline, Performance and Call Tree results) on O3 optimization level independent of the type of code. The limited accuracy is defined due to the following reasons, when:

- There is no debug information generated for inlined functions
- The compiler has tail call optimization enabled by default, which means that a `jsr` is transformed into a `jmp (bsr vs. bra)`. It is possible to disable this optimization using the compiler option `-Xllt --tailjsr2jmp=0`

This chapter explains:

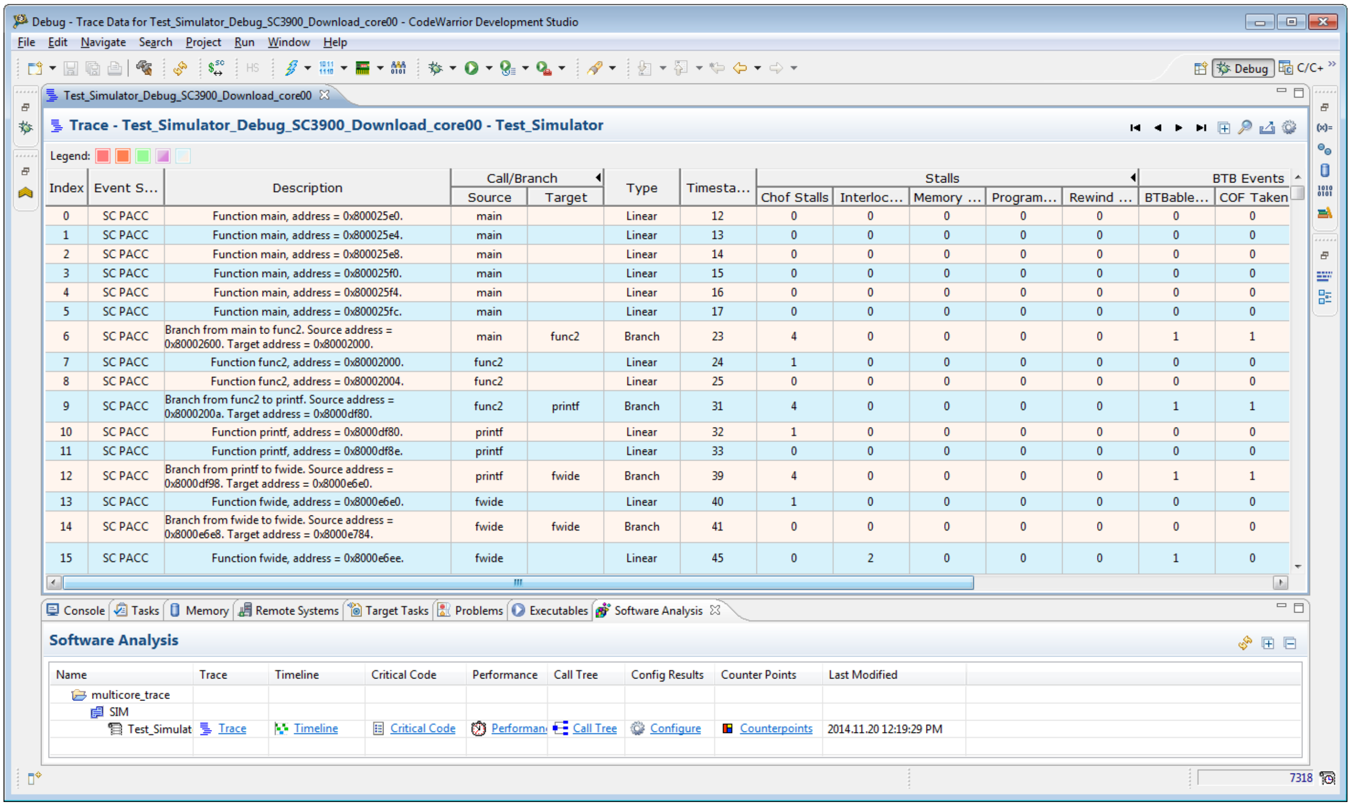
- About CodeWarrior Interface
- About Data Collection

2.1 About CodeWarrior Interface

About CodeWarrior Interface

The CodeWarrior Development Studio provides a common interface for developing, debugging, and analyzing your applications. The project-oriented Workbench window provides numerous perspectives containing views, editors, and controls that appear in menus and tool bars.

After creating a project, build your application, define a launch configuration, and then wait for data collection and data display.


Debug - Trace Data for Test_Simulator_Debug_SC3900_Download_core00 - CodeWarrior Development Studio
File
Edit
Navigate
Search
Project
Run
Window
Help
Debug
C/C++
Test_Simulator_Debug_SC3900_Download_core00
Trace - Test_Simulator_Debug_SC3900_Download_core00 - Test_Simulator
Legend: 🟥 🟧 🟩 🟪 ⬜
Console
Tasks
Memory
Remote Systems
Target Tasks
Problems
Executables
Software Analysis
7318

Index	Event S...	Description	Call/Branch		Type	Timesta...	Stalls					BTB Events	
			Source	Target			Chof Stalls	Interloc...	Memory ...	Program...	Rewind ...	BTBable...	COF Taken
0	SC PACC	Function main, address = 0x800025e0.	main		Linear	12	0	0	0	0	0	0	0
1	SC PACC	Function main, address = 0x800025e4.	main		Linear	13	0	0	0	0	0	0	0
2	SC PACC	Function main, address = 0x800025e8.	main		Linear	14	0	0	0	0	0	0	0
3	SC PACC	Function main, address = 0x800025f0.	main		Linear	15	0	0	0	0	0	0	0
4	SC PACC	Function main, address = 0x800025f4.	main		Linear	16	0	0	0	0	0	0	0
5	SC PACC	Function main, address = 0x800025fc.	main		Linear	17	0	0	0	0	0	0	0
6	SC PACC	Branch from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Branch	23	4	0	0	0	0	1	1
7	SC PACC	Function func2, address = 0x80002000.	func2		Linear	24	1	0	0	0	0	0	0
8	SC PACC	Function func2, address = 0x80002004.	func2		Linear	25	0	0	0	0	0	0	0
9	SC PACC	Branch from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Branch	31	4	0	0	0	0	1	1
10	SC PACC	Function printf, address = 0x8000df80.	printf		Linear	32	1	0	0	0	0	0	0
11	SC PACC	Function printf, address = 0x8000df8e.	printf		Linear	33	0	0	0	0	0	0	0
12	SC PACC	Branch from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Branch	39	4	0	0	0	0	1	1
13	SC PACC	Function fwide, address = 0x8000e6e0.	fwide		Linear	40	1	0	0	0	0	0	0
14	SC PACC	Branch from fwide to fwide. Source address = 0x8000e6e8. Target address = 0x8000e784.	fwide	fwide	Branch	41	0	0	0	0	0	0	0
15	SC PACC	Function fwide, address = 0x8000e6ee.	fwide		Linear	45	0	2	0	0	0	1	0

Software Analysis

Name	Trace	Timeline	Critical Code	Performance	Call Tree	Config Results	Counter Points	Last Modified
multicore_trace								
SIM								
Test_Simulat	Trace	Timeline	Critical Code	Performan	Call Tree	Configure	Counterpoints	2014.11.20 12:19:29 PM

Figure 2-1. Workbench Window

The perspective is a collection of views within the Workbench window used by the Tracing and Analysis tools.

Standard perspective views:

- **CodeWarrior Projects** view provides a hierarchical view of the project resources in the Workbench. Right-click context menus provide file-management controls.
- **Console** view shows the process output including actions, messages, and errors. It display messages indicating when data collection is enabled, in process, and complete.
- **Software Analysis** view provides a hierarchical view of the project's data sources, data files, and data sets of the simulator.

2.2 About Data Collection

This section lists the types of data collected for an application when it runs on the simulator SC3900 and Qonverge targets.

You can collect the following types of data for an application:

- Trace Data
- Timeline Data
- Critical Code Data
- Performance Data
- Call Tree Data
- Counterpoints

2.2.1 Trace Data

The Trace viewer displays the trace data collected by the simulator and the targets. The features available for the Trace viewer include:

- Export trace data to an Excel file
- Configure Timestamp column of the trace data as absolute or delta values in decimal or hexadecimal format
- Set CPU frequency and convert the clock cycles into real time in milliseconds, microseconds, or nanoseconds, displayed in the **Timestamp** column
- Copy and paste a cell or a line of the trace data
- Auto resize the column width or a row
- Quick navigation through trace using navigation buttons
- Expand the trace lines for more detailed information on the events
- Search capability

2.2.2 Timeline Data

The timeline data displays a graphical view of the functions that are executed in the application and the number of cycles each function takes when the application is run.

2.2.3 Critical Code Data

The critical code data is generated based on the trace data. The Critical Code Data viewer displays name, start address, number of times each instruction is executed, and code size of each function in the program. The Critical Code Data viewer displays the detailed information of every instruction traced in the data.

The features available for the Critical Code Data viewer include:

- statistics at function level,
- statistics at instruction level,
- code view in source editor, and
- column ordering and sorting at function level.

2.2.4 Performance Data

The performance data includes the metric and invocation information for each function that executes in the application. The performance data during measurement enables you to compare the relative efficiencies of various portions of your target program. Both exclusive and inclusive timing measurements are provided in the performance data. The parent-child calling relationships between your program's functions are also provided. Each function pair consists of a caller and a callee with data provided for each.

2.2.5 Call Tree Data

The Call Tree data shows the general application flow in a hierarchical tree in which statistics are displayed for each function.

2.2.6 Counterpoints

Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. When counterpoints are found in the executed code, the delta statistics is computed between two counterpoints.

Chapter 3
Collecting Data

This chapter explains how the program runs on the simulator and Qonverge targets to collect the data.

This chapter describes:

- Process for Collecting Data
- Creating New Project
- Configuring Launcher
- Collecting Trace
- Viewing Data
- Exclude Symbols from Statistics
- Trace on Attach
- Viewing Interrupts
- Preparing Aurora Interface for Collecting Aurora Nexus Trace
- Inline Functions
- Importing Offline SC3900 Trace with Hardware Trace Configuration
- Multicore Tracing

3.1 Process for Collecting Data

This section describes the process for setting up the tools for data collection.

To collect data:

- Create and configure a project for the simulator
- Set up the debugger launch configuration to collect the analysis data from the simulator
- Run the application on the simulator to collect data

3.2 Creating New Project

This section explains the process to create new projects for both simulator and hardware profiling.

You can use the CodeWarrior Bareboard Project Wizard to create new projects.

In this topic:

- Simulator Profiling
- Hardware Profiling

3.2.1 Simulator Profiling

This section describes the process to create a simulator project for collecting trace data.

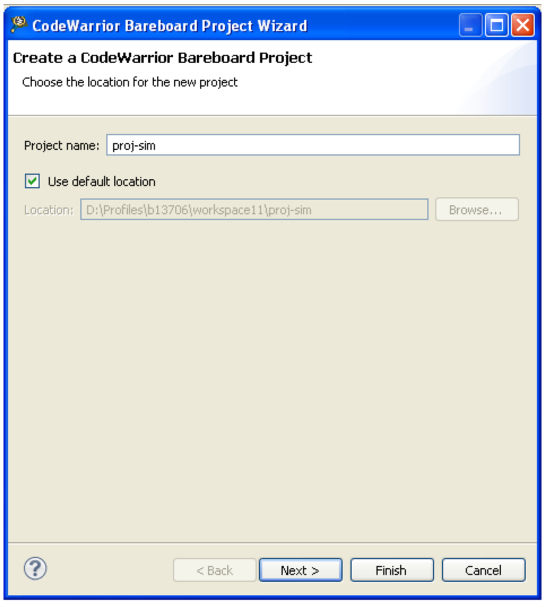
You can use the **CodeWarrior Bareboard Project Wizard** to create new projects for tracing.

The CodeWarrior IDE is a project-oriented interface. You must create a new project or open an existing project before using the Analysis tools.

To create a new project:

1. Select **File > New > CodeWarrior Bareboard Project Wizard**.

   The **CodeWarrior Bareboard Project Wizard** page appears.



CodeWarrior Bareboard Project Wizard

Create a CodeWarrior Bareboard Project

Choose the location for the new project

Project name: proj-sim

☑ Use default location

Location: D:\Profiles\b13706\workspace11\proj-sim

Figure 3-1. Create a CodeWarrior Bareboard Project Page

2. On the **Create a CodeWarrior Bareboard Project** page, enter the name of your project and also specify the location of the project.

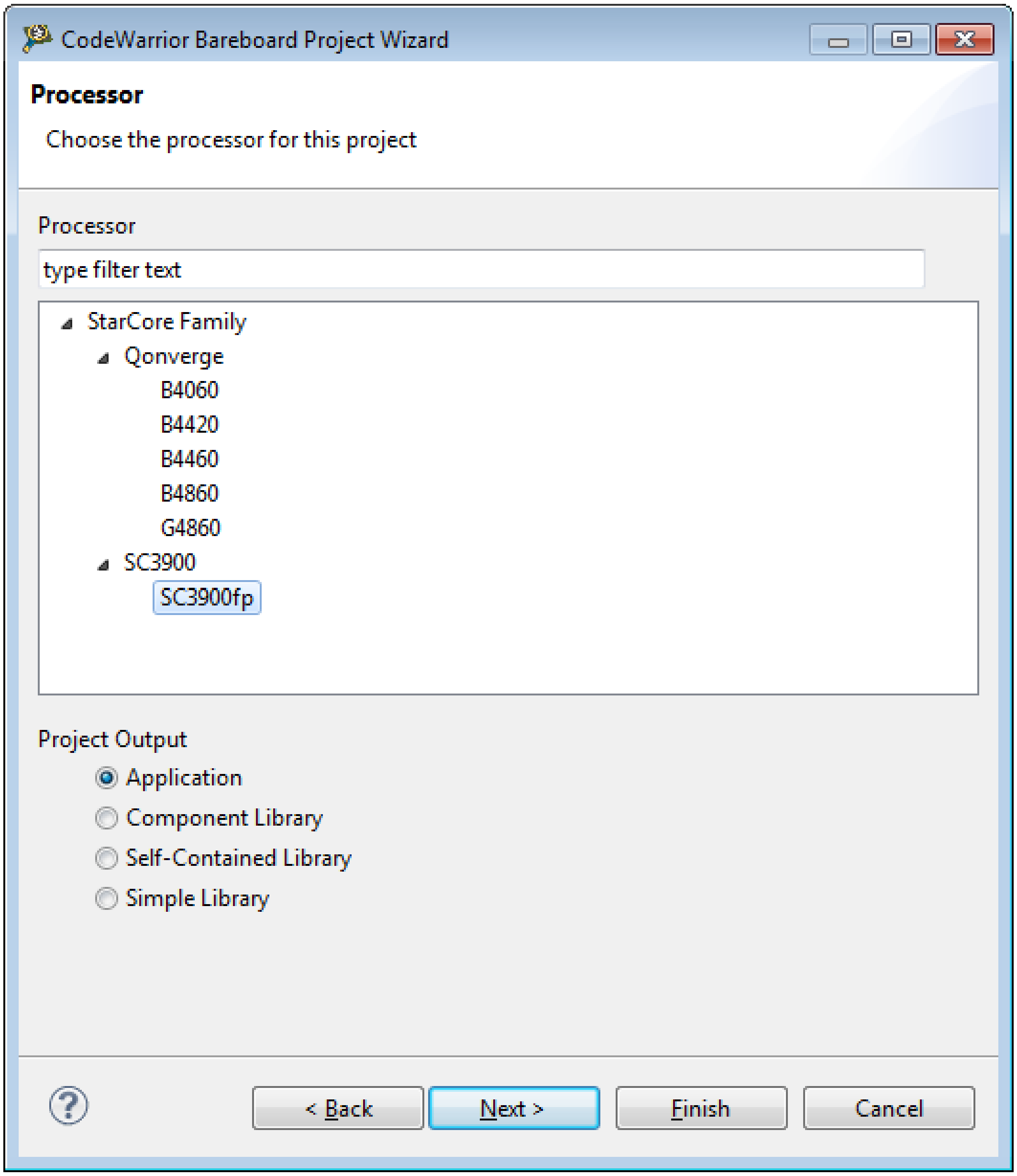
   The following table describes the StarCore Project settings.

Table 3-1. Create a StarCore Project Page Settings

Option	Description
New Project Name	Enter the name of the project in this text box
Use default location	• Checked - Project stores the files required to build the program in the Workbench's current workspace directory • Unchecked - Project files are located in the directory you specify. Use the Location text box to select the directory
Location	Specifies the directory that contains the project files. Use the **Browse** button to navigate to the desired directory. This option is only available when **Use default location** is unchecked

3. Click **Next**.

   The **Processor** page appears.



CodeWarrior Bareboard Project Wizard

Processor

Choose the processor for this project

Processor

type filter text

- StarCore Family
  - Qonverge
    - B4060
    - B4420
    - B4460
    - B4860
    - G4860
  - SC3900
    - SC3900fp

Project Output

● Application

○ Component Library

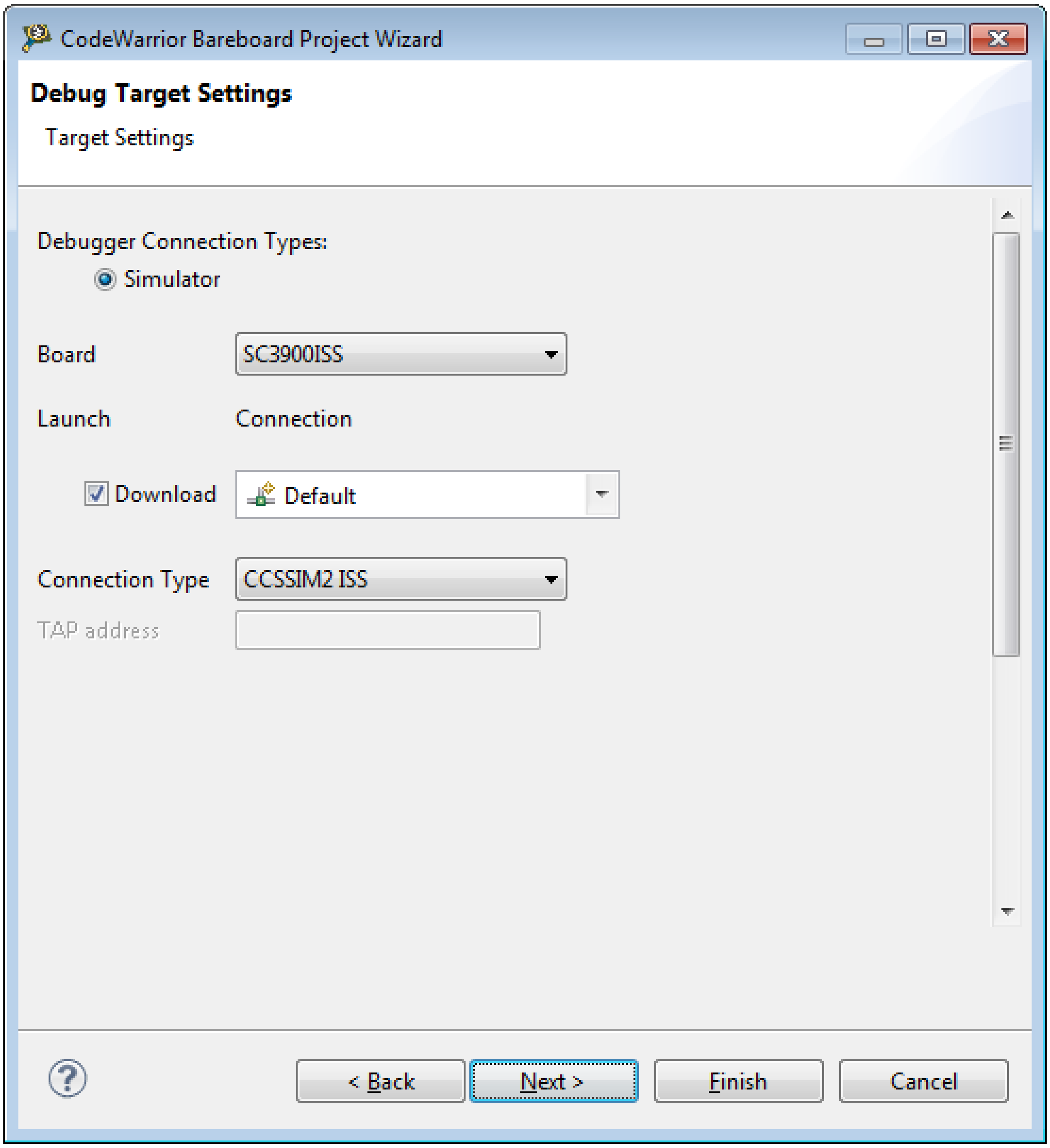
○ Self-Contained Library

○ Simple Library

Figure 3-2. Processor Page

4. On the **Processor** page, choose the target device for your project. Select **SC3900fp** from **StarCore Family** options.
5. Click **Next**.

   The **Debug Target Settings** page appears.



CodeWarrior Bareboard Project Wizard

Debug Target Settings

Target Settings

Debugger Connection Types:

● Simulator

Board: SC3900ISS

Launch: Connection

☑ Download: Default

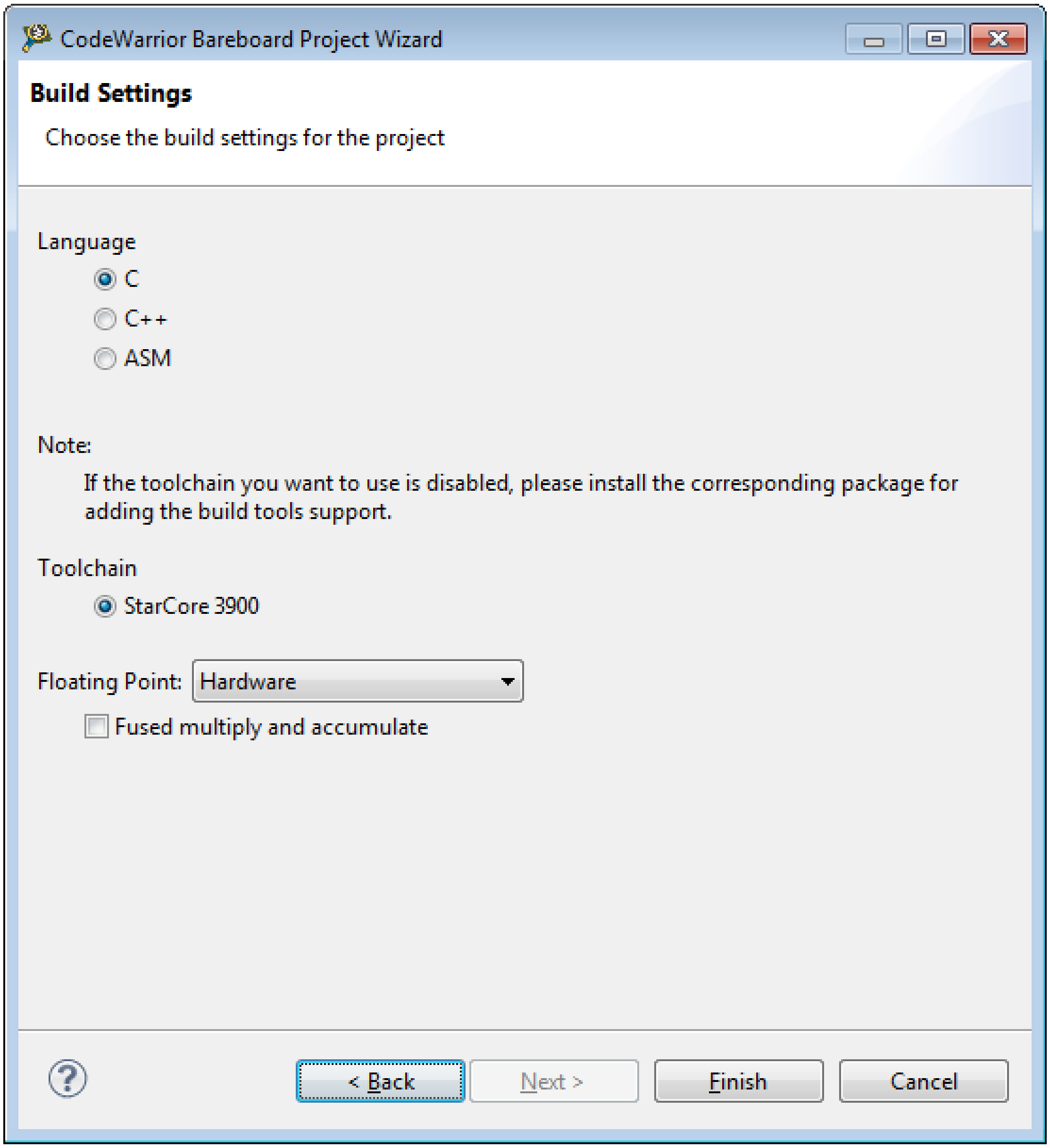
Connection Type: CCSSIM2 ISS

TAP address:

Figure 3-3. Debug Target Settings Page

6. Select **SC3900PACC** board from the list.
7. Click **Next**.

   The **Build Settings** page appears.



CodeWarrior Bareboard Project Wizard

Build Settings

Choose the build settings for the project

Language

● C

○ C++

○ ASM

Note:

If the toolchain you want to use is disabled, please install the corresponding package for adding the build tools support.

Toolchain

● StarCore 3900

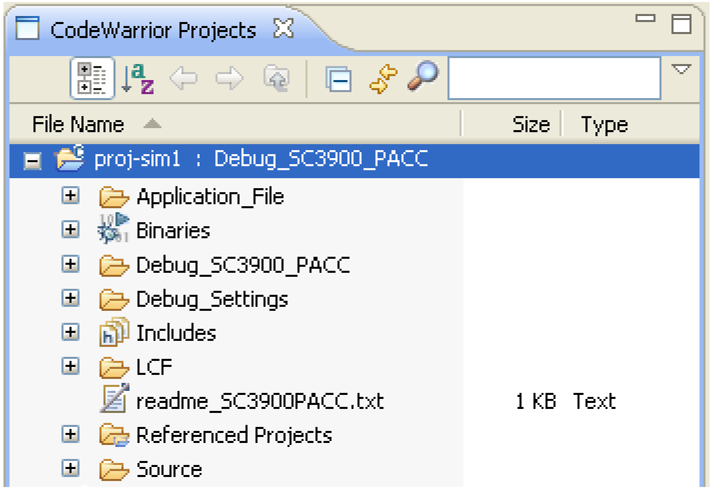
Floating Point: Hardware

□ Fused multiply and accumulate

Figure 3-4. Build Settings Page

8. Do not change the default settings on the **Build Settings** page.
9. Click **Finish**.

   The project is created in the **CodeWarrior Projects** view. In this view, you can see that the *proj-sim* project is created.



File Name	Size	Type
proj-sim1 : Debug_SC3900_PACC		
Application_File		
Binaries		
Debug_SC3900_PACC		
Debug_Settings		
Includes		
LCF		
readme_SC3900PACC.txt	1 KB	Text
Referenced Projects		
Source		

Figure 3-5. CodeWarrior Projects View Page

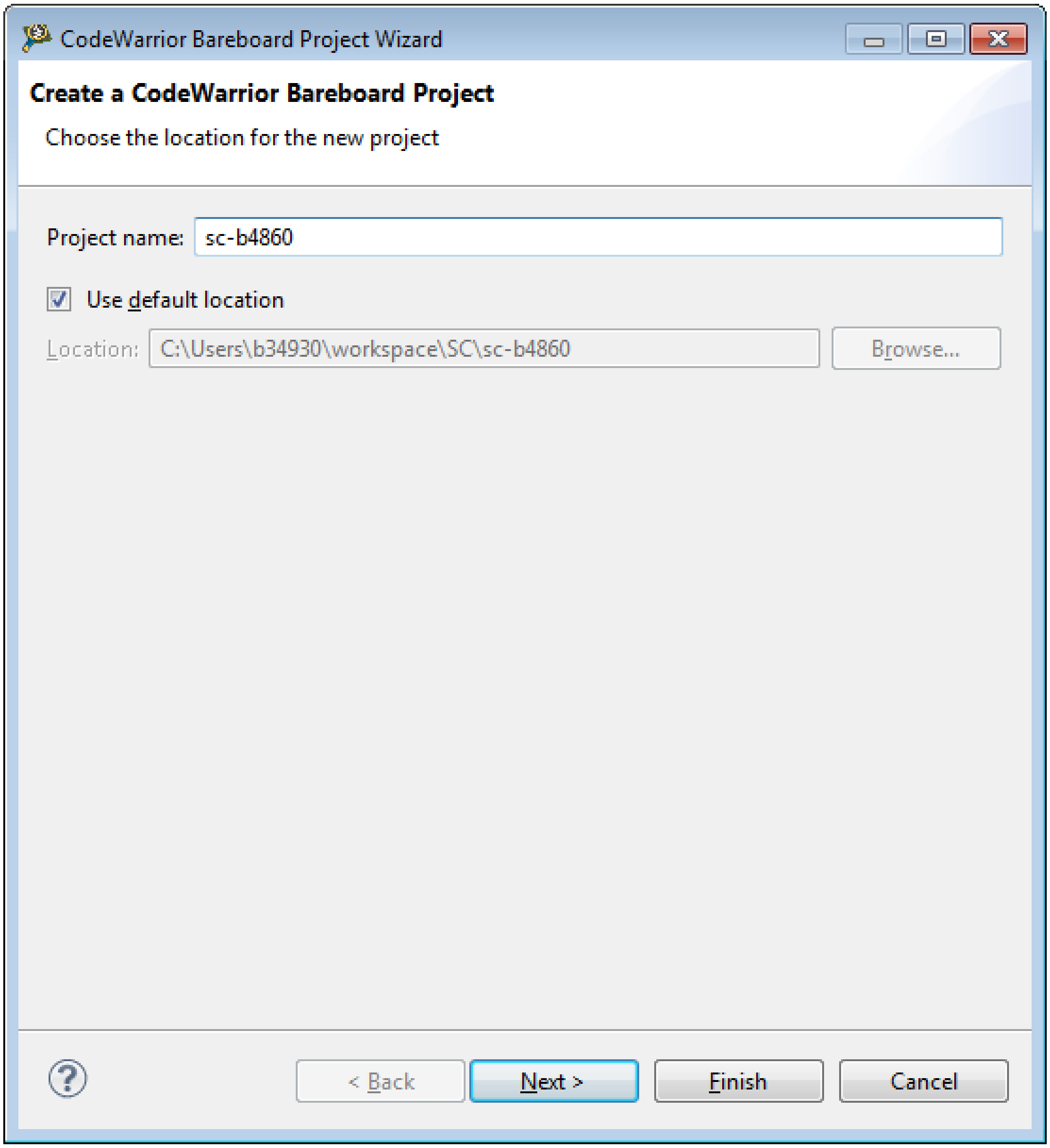
3.2.2 Hardware Profiling

This section describes the process to create a new project for collecting trace data.

You must create a new project or open an existing project before using the Analysis tools for hardware profiling.

To create a new project:

1. Select **File > New > CodeWarrior Bareboard Project Wizard**. The **CodeWarrior Bareboard Project Wizard** appears.
2. Enter the name of your project and also specify the location of the project.



CodeWarrior Bareboard Project Wizard

Create a CodeWarrior Bareboard Project

Choose the location for the new project

Project name: sc-b4860

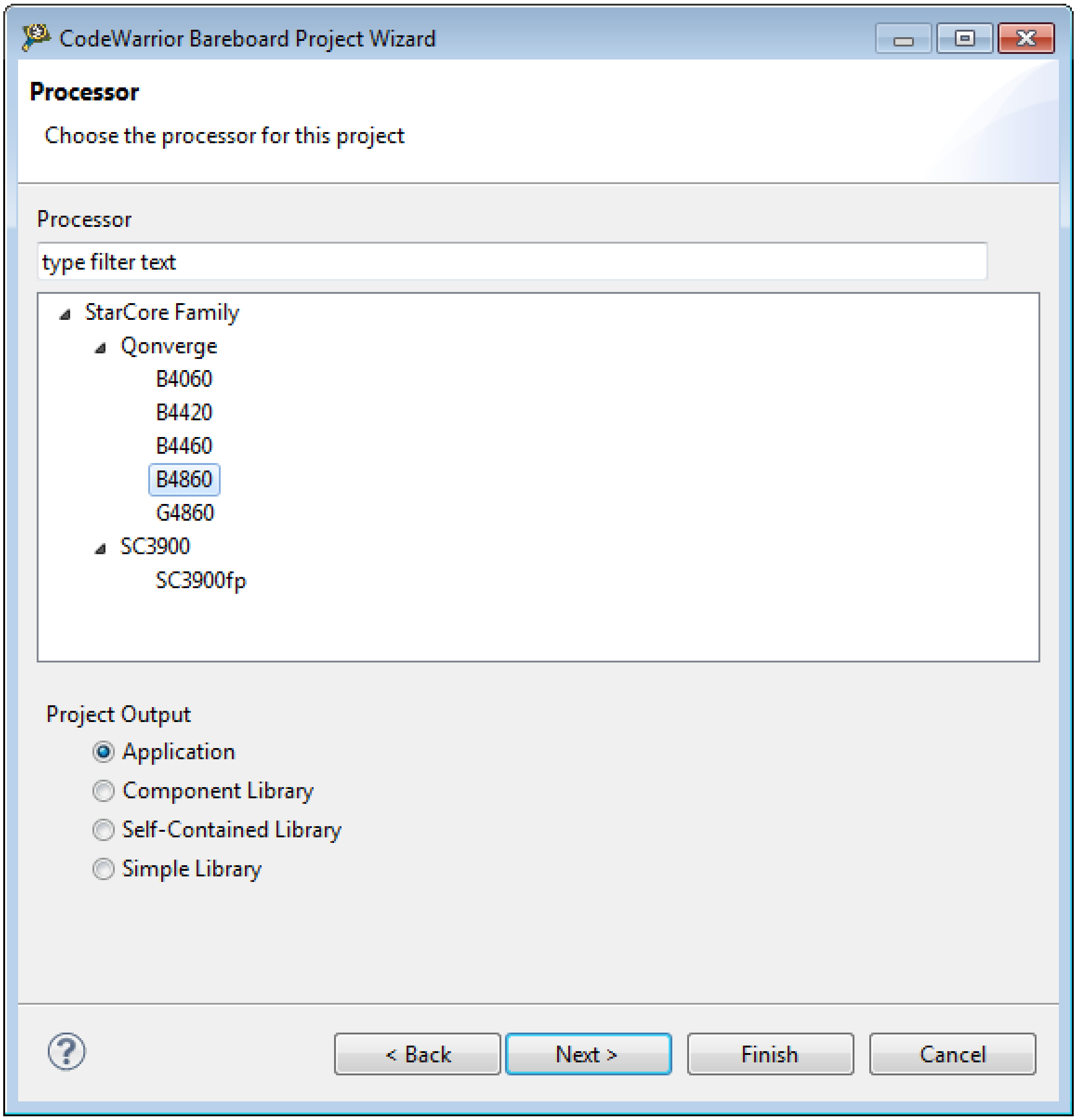
☑ Use default location

Location: C:\Users\b34930\workspace\SC\sc-b4860

Figure 3-6. Create a CodeWarrior Bareboard Project Page

3. Click **Next**.

   The **Processor** page appears.



CodeWarrior Bareboard Project Wizard

Processor

Choose the processor for this project

Processor

type filter text

- StarCore Family
  - Qonverge
    - B4060
    - B4420
    - B4460
    - B4860
    - G4860
  - SC3900
    - SC3900fp

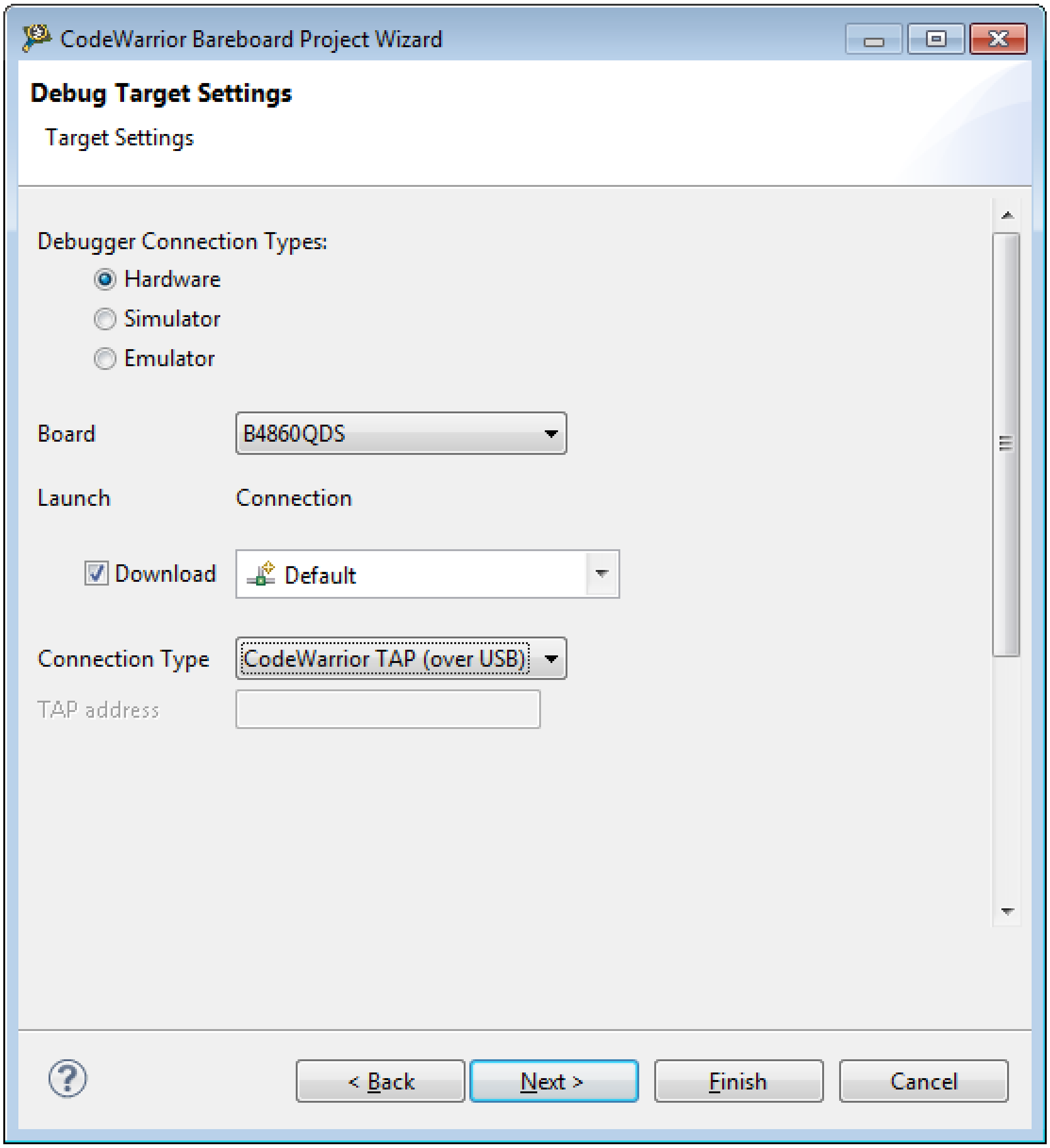
Project Output

- ● Application
- ○ Component Library
- ○ Self-Contained Library
- ○ Simple Library

Figure 3-7. Processor Page

4. For hardware profiling, select the desired target (for example, **B4860**) from **StarCore Family** group.
5. Click **Next**.

   The **Debug Target Settings** page appears. Do not change the default settings of this page.



Debugger Connection Types:

● Hardware

○ Simulator

○ Emulator

Board: B4860QDS

Launch: Connection

☑ Download: Default

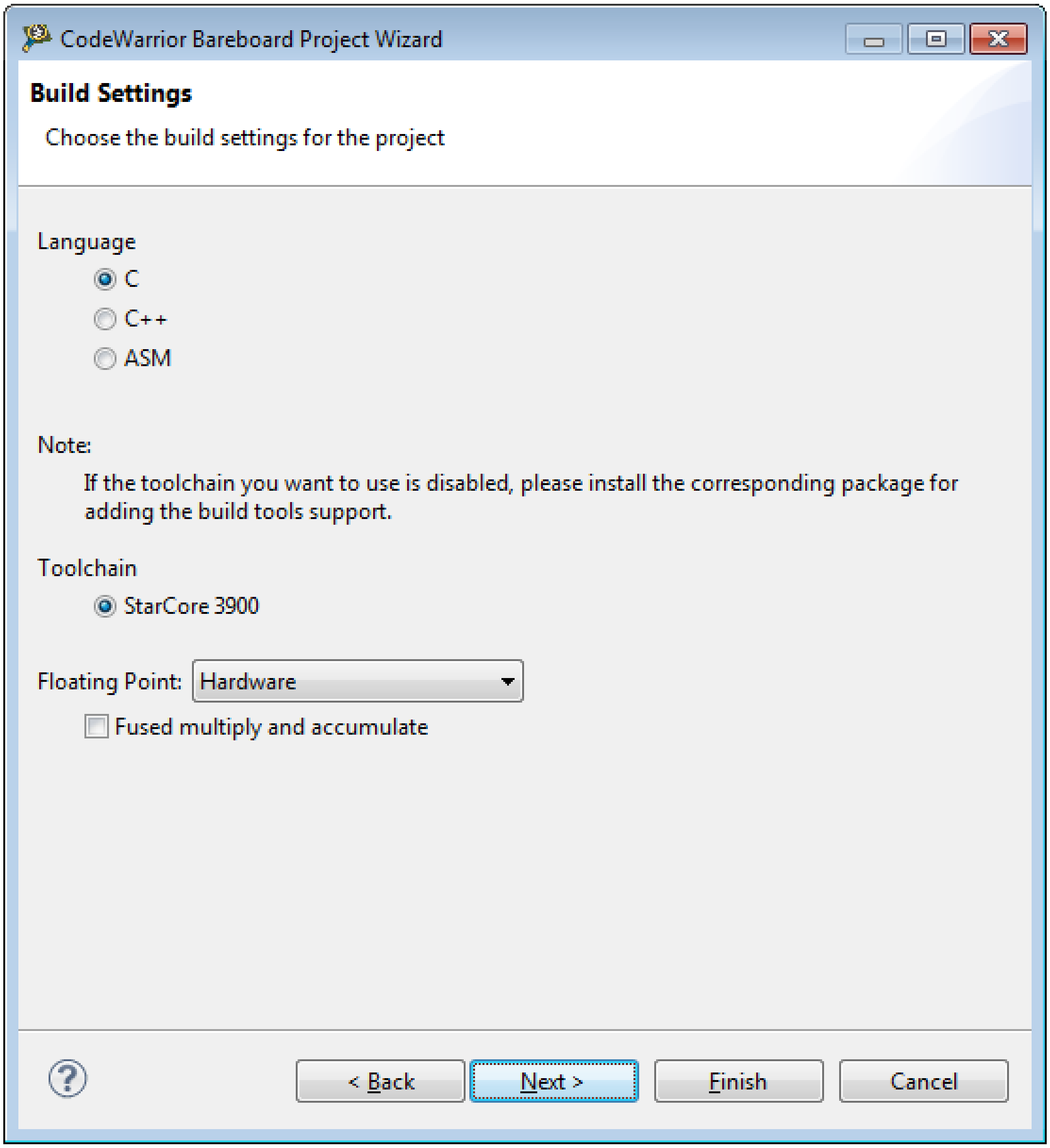
Connection Type: CodeWarrior TAP (over USB)

TAP address:

Figure 3-8. Debug Target Settings Page

6. Click **Next**.

   The **Build Settings** page appears.



Build Settings

Choose the build settings for the project

Language

● C

○ C++

○ ASM

Note:

If the toolchain you want to use is disabled, please install the corresponding package for adding the build tools support.

Toolchain

● StarCore 3900

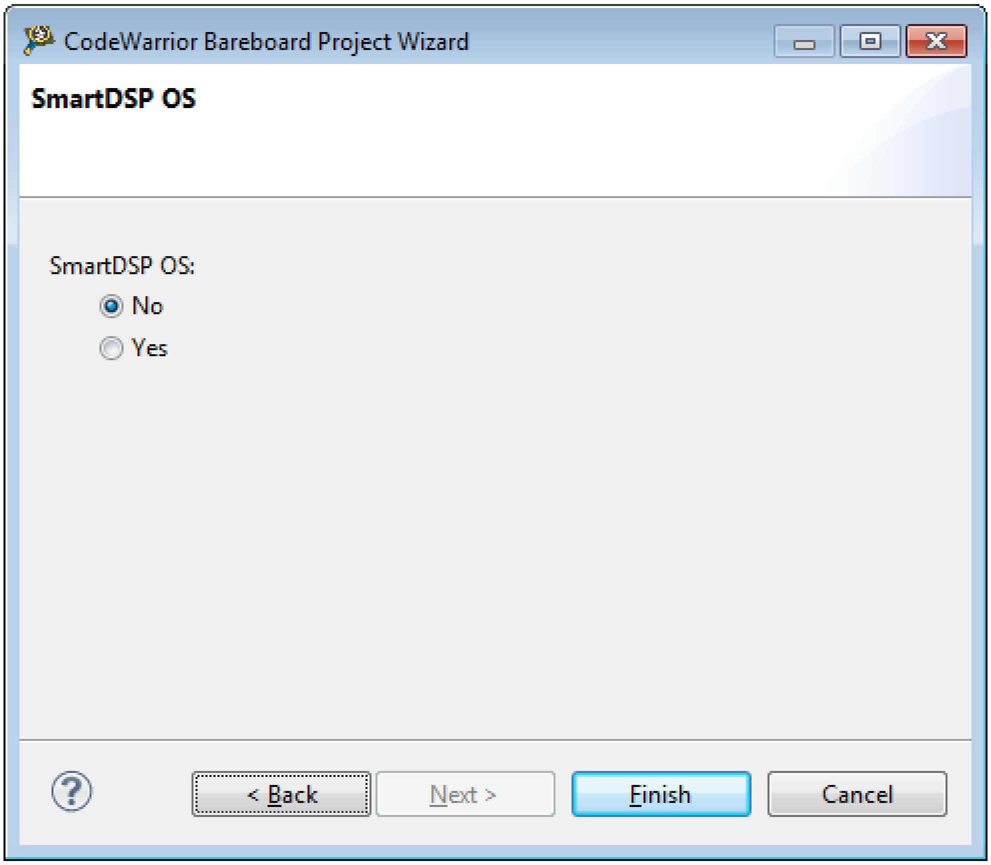
Floating Point: Hardware

□ Fused multiply and accumulate

Figure 3-9. Build Settings Page

7. Do not change the default settings on the **Build Settings** page.
8. Click **Next**.

   The **SmartDSP OS** page appears.



CodeWarrior Bareboard Project Wizard

SmartDSP OS

SmartDSP OS:

● No

○ Yes

Figure 3-10. SmartDSP OS Page

9. Select the SmartDSP OS option you want to use.
   - Select No to exclude SmartDSP OS support from your project.
   - Select Yes to include SmartDSP OS support in your project.
10. Click **Finish**.
11. The project is created in the **Projects Explorer** view. In this view, you can see that `sc-b4860` project is created.

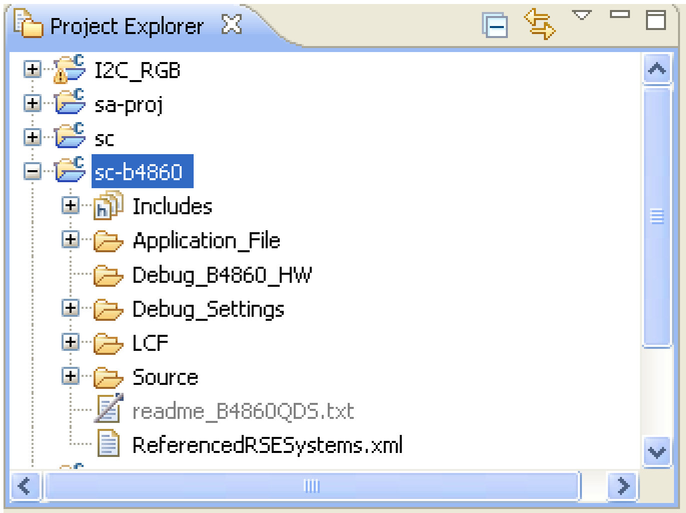


Figure 3-11. Project Explorer View

12. Select **Project > Build Project** to build your project.

3.3 Configuring Launcher

You need to configure the launcher for simulator and hardware profiling, before debugging your application.

In this topic:

- Simulator Profiling
- Hardware Profiling

3.3.1 Simulator Profiling

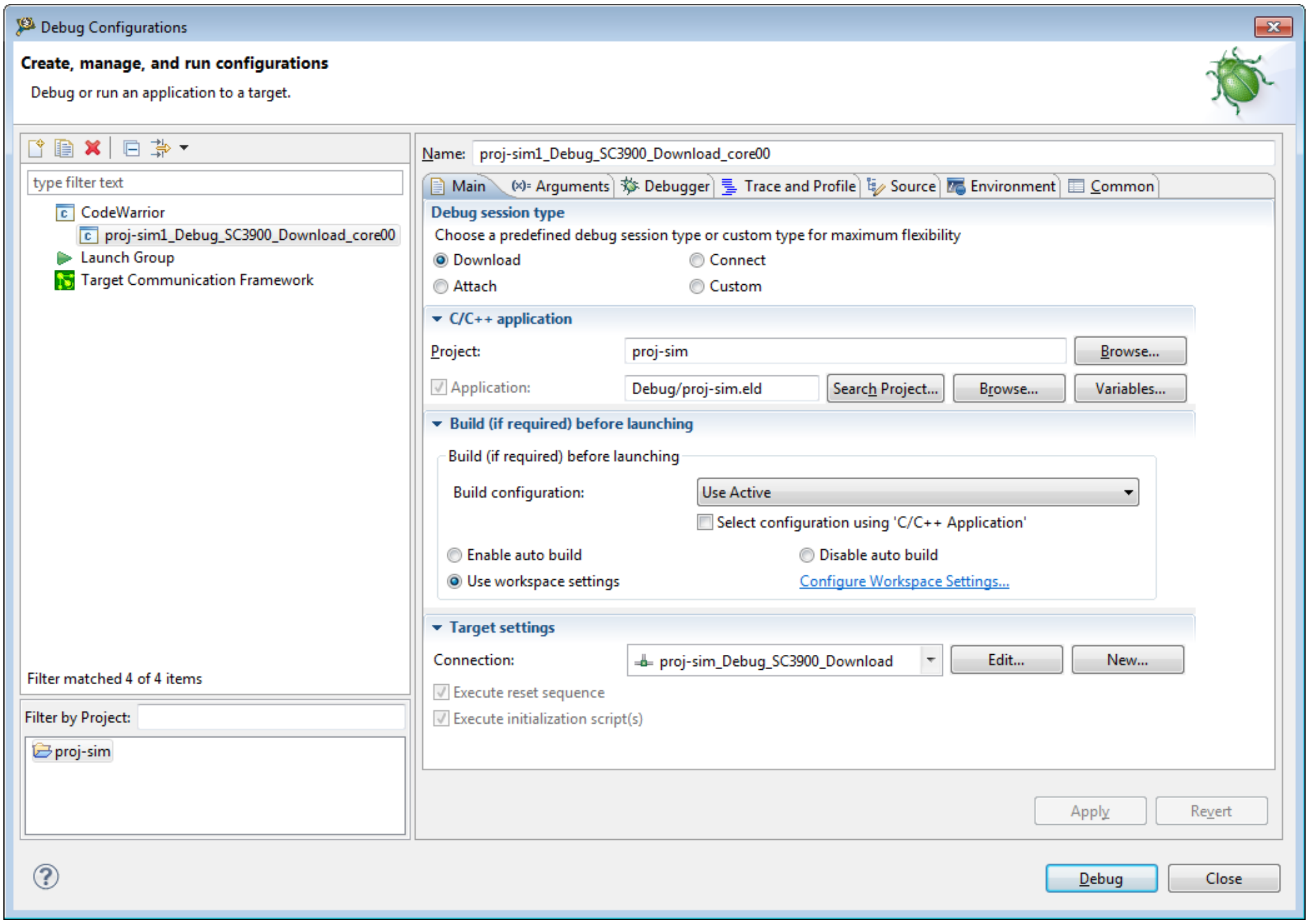
Before debugging your application, you need to configure the launcher for simulator tracing.

To configure the launch configuration:

1. In the **CodeWarrior Projects** view, right-click on the project and select Debug as > Debug Configurations from the context menu.

   The **Debug Configuration** dialog box appears.

2. Expand the CodeWarrior group in the tree structure on the left, and select the launch configuration corresponding to the project you are using from the list to create a launch configuration, for example, `proj-sim1_Debug_SC3900_Download_core0`.
3. In the **Main** tab of the **Debug Configurations** dialog box, verify that proj-sim1 is displayed in the **Project** field. If it does not appear, browse to the project.
4. If the application is not displayed in the **Application** field, click **Search Project** to select the application image.



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

- CodeWarrior
  - proj-sim1_Debug_SC3900_Download_core00
- Launch Group
- Target Communication Framework

Filter matched 4 of 4 items

Filter by Project:

proj-sim

Name: proj-sim1_Debug_SC3900_Download_core00

Main | Arguments | Debugger | Trace and Profile | Source | Environment | Common

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

● Download ○ Connect

○ Attach ○ Custom

C/C++ application

Project: proj-sim [Browse...]

☑ Application: Debug/proj-sim.eld [Search Project...] [Browse...] [Variables...]

Build (if required) before launching

Build (if required) before launching

Build configuration: Use Active

□ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

Target settings

Connection: proj-sim_Debug_SC3900_Download [Edit...] [New...]

☑ Execute reset sequence

☑ Execute initialization script(s)

[Apply] [Revert]

[Debug] [Close]

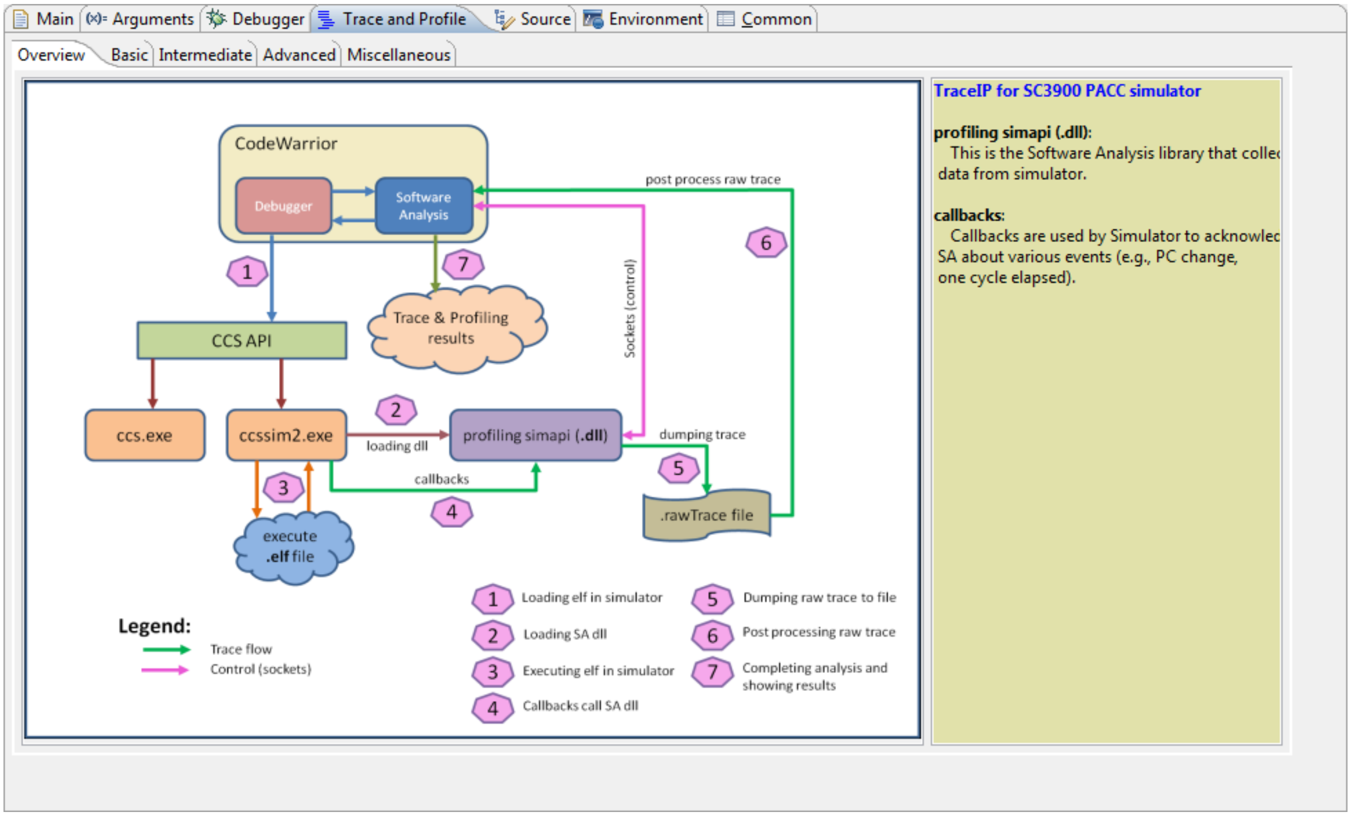
Figure 3-12. Debug Configurations Dialog Box - Main Page

To configure the launch configuration for measurement of data:

1. Click the **Trace and Profile** tab. This tab has **Overview** page, **Basic** page, **Intermediate** page, **Advanced** and **Miscellaneous** page.
   The **Overview** page displays the flow diagram for collecting trace.

NOTE

The **Advanced** tab is not applicable for Simulator profiling.



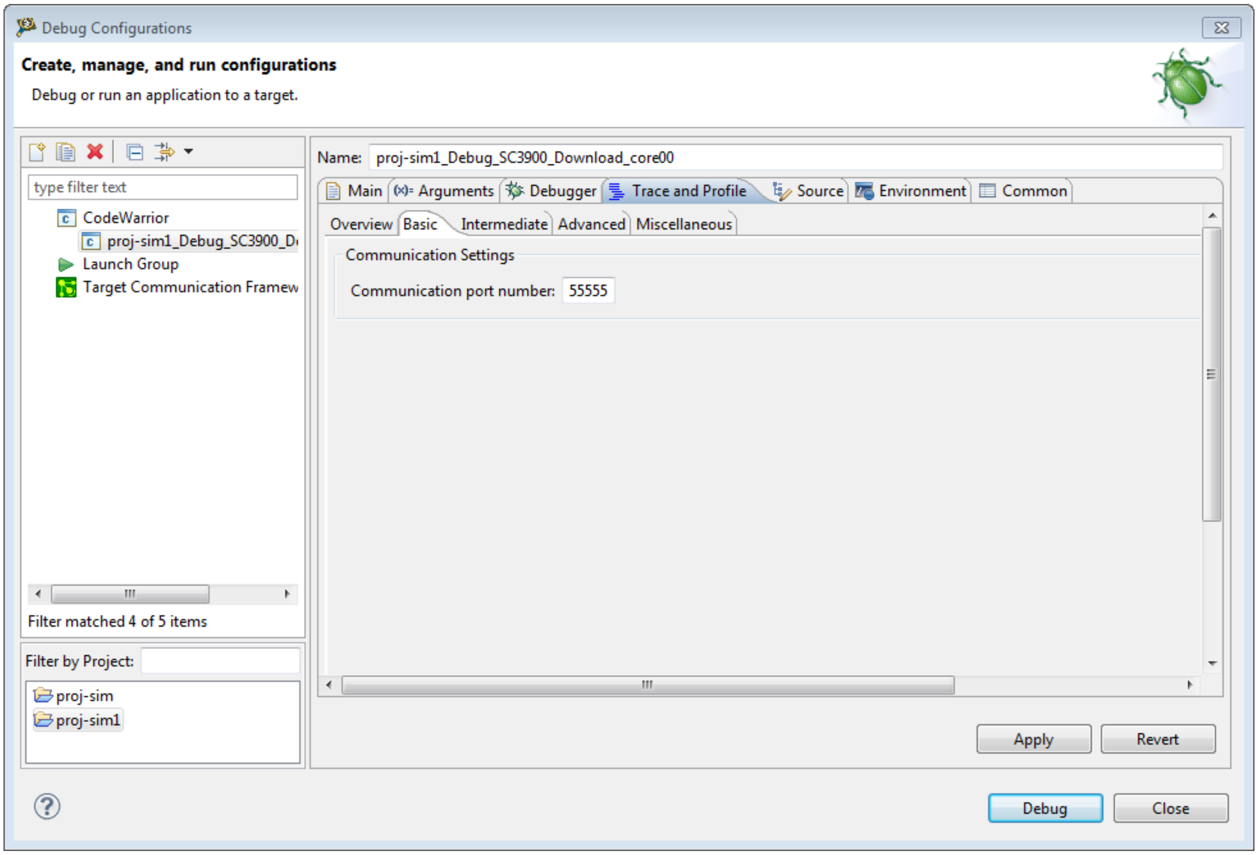
TraceIP for SC3900 PACC simulator

profiling simapi (.dll):
This is the Software Analysis library that collec
data from simulator.

callbacks:
Callbacks are used by Simulator to acknowlec
SA about various events (e.g., PC change,
one cycle elapsed).

Figure 3-13. Trace and Profile Tab - Overview Page

2. In the **Basic** page, specify the **Communication Settings**.



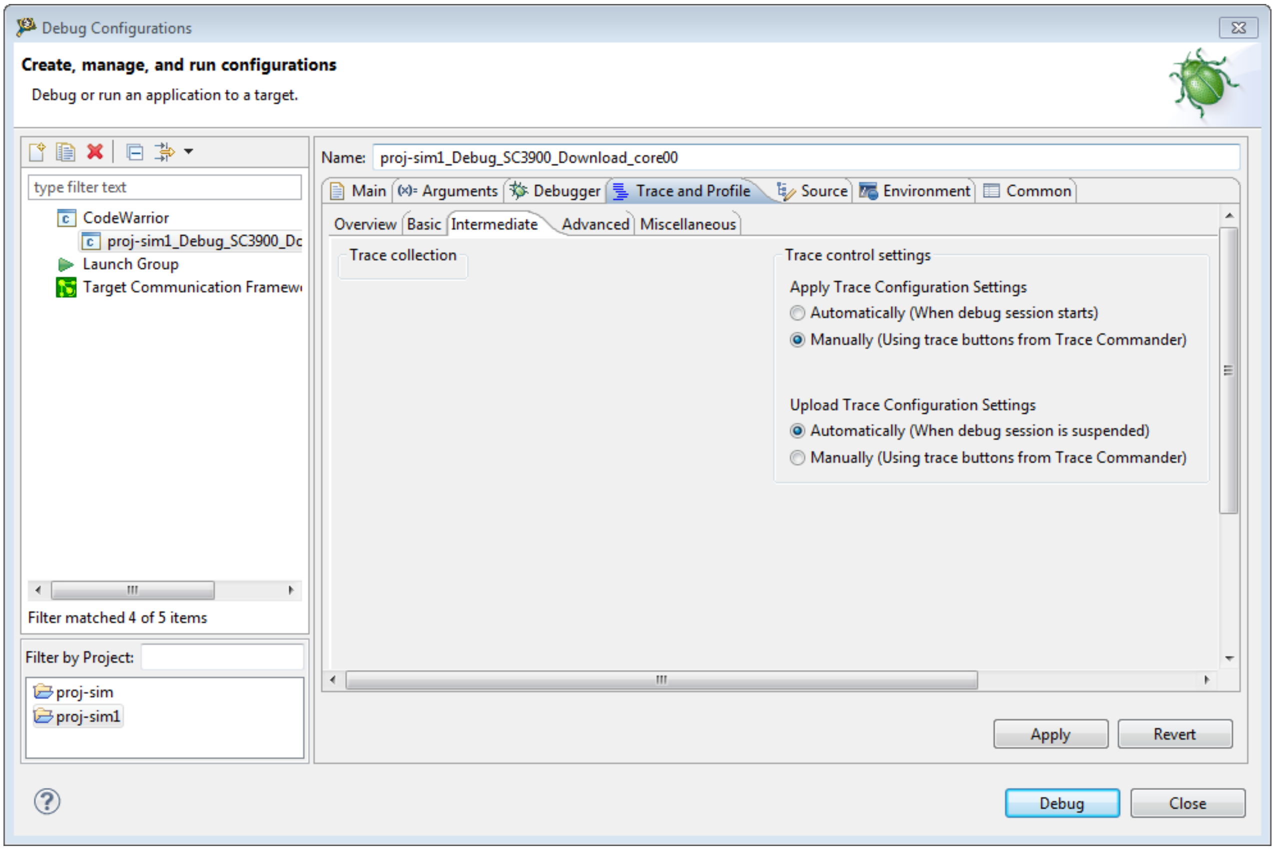
Name: proj-sim1_Debug_SC3900_Download_core00

Communication port number: 55555

Figure 3-14. Trace and Profile Tab - Basic Page

- Choose the free port number from interval 0-65535 for communication settings. The default port number is 55555. This is used in TCP/IP communication between software analysis and CCSSIM2.

3. In the **Intermediate** page, specify **Trace control settings** option.



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

Name: proj-sim1_Debug_SC3900_Download_core00

Trace and Profile > Intermediate

Trace control settings

Apply Trace Configuration Settings

○ Automatically (When debug session starts)

● Manually (Using trace buttons from Trace Commander)

Upload Trace Configuration Settings

● Automatically (When debug session is suspended)

○ Manually (Using trace buttons from Trace Commander)

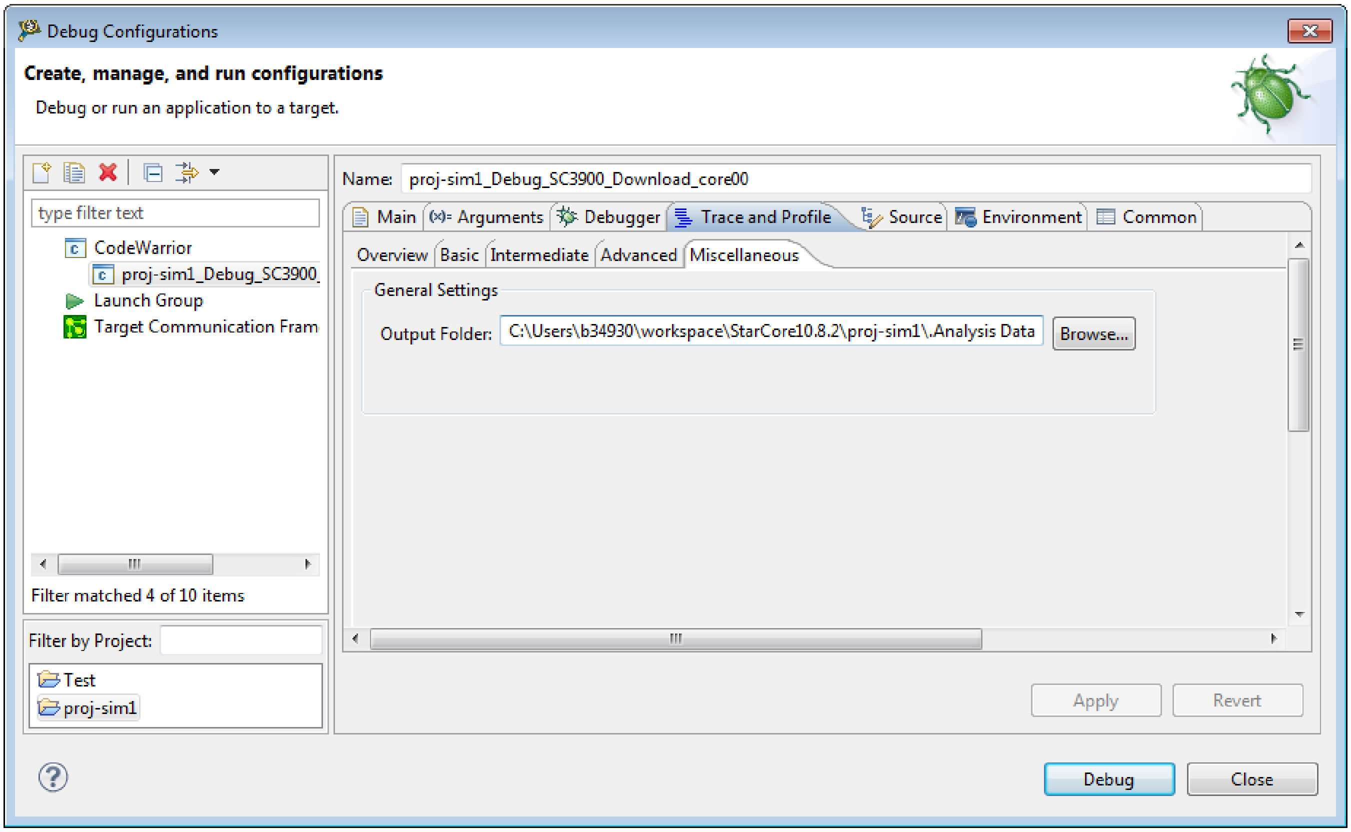
Figure 3-15. Trace and Profile Tab - Intermediate Page

- Apply Trace Configuration Settings
  - **Automatically (When debug session starts)** - The trace collection process is started automatically when the debug session is launched.
  - **Manually (Using trace buttons from Trace Commander)** - This is the default behavior. This option is used when you want to manually control the trace collection. The trace collection process will not be started automatically on debug session launch, but the trace configuration options are applied, when you click on **Trace Start/Stop** button.
- Upload Trace Configuration Settings
  - **Automatically (When debug session is suspended)** - This is the default behavior. This option is used to upload the trace data automatically.
  - **Manually (Using trace buttons from Trace Commander)** - This option is used when you want to manually upload the trace data using **Upload Trace** button available in the **Trace Commander** view.

NOTE

Trace configuration settings will be disabled during debug session.

4. In the **Miscellaneous** tab, specify the path for storing the trace and profile results. Click **Browse** to select the path for the **Output Folder**.



Name: proj-sim1_Debug_SC3900_Download_core00

Output Folder: C:\Users\b34930\workspace\StarCore10.8.2\proj-sim1\.Analysis Data

Figure 3-16. Trace and Profile Tab - Miscellaneous Page

NOTE

Before using a simulator to collect trace data, a configuration must be made. You can create the configuration in the debug mode. If the project is started without a configuration, a warning message will appear when trying to start the trace collection. No trace data will be collected during that session.

3.3.2 Hardware Profiling

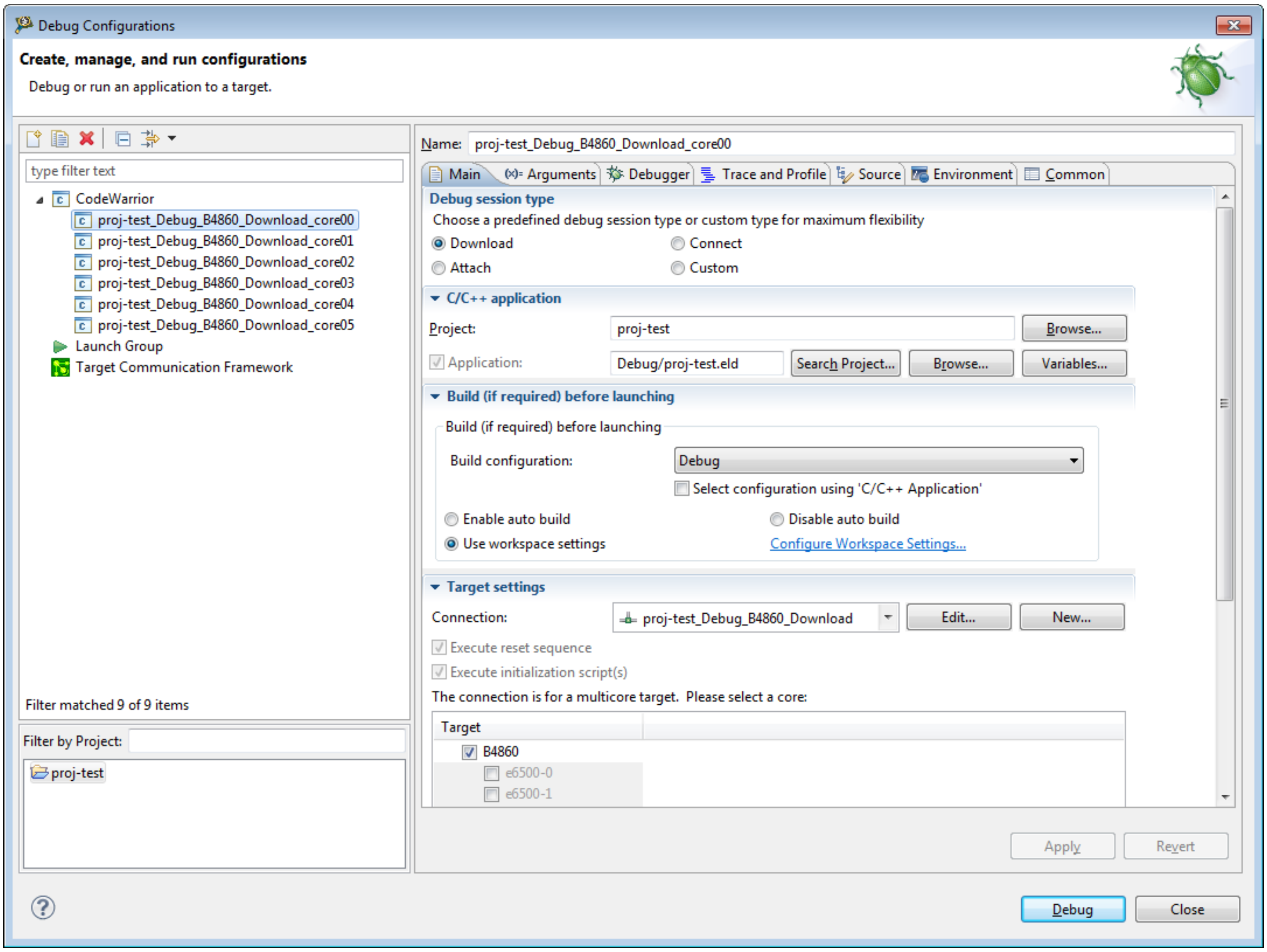
You need to configure the launch configuration for hardware profiling.

Perform the following steps to configure the launch configuration:

1. In the CodeWarrior Projects view, right-click on the project and select **Debug as > Debug Configurations** from the context-menu.

The Debug Configurations dialog box appears.

2. Select **CodeWarrior Download** in the tree structure on the left and select the launch configuration corresponding to the project you are using from the list to create a launch configuration, for example, *proj-test_Debug_B4860_Download_core0 (0...5)*.
3. In the Main page, verify that *proj-test* is displayed in the **Project** field. If it does not appear, browse to the project.
4. If the application is not displayed in the **C/C++Application** field, click **Search Project** to select the application image.



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

- CodeWarrior
  - proj-test_Debug_B4860_Download_core00
  - proj-test_Debug_B4860_Download_core01
  - proj-test_Debug_B4860_Download_core02
  - proj-test_Debug_B4860_Download_core03
  - proj-test_Debug_B4860_Download_core04
  - proj-test_Debug_B4860_Download_core05
- Launch Group
- Target Communication Framework

Filter matched 9 of 9 items

Filter by Project:

proj-test

Name: proj-test_Debug_B4860_Download_core00

Main | Arguments | Debugger | Trace and Profile | Source | Environment | Common

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

● Download ○ Connect

○ Attach ○ Custom

C/C++ application

Project: proj-test

☑ Application: Debug/proj-test.eld

Build (if required) before launching

Build (if required) before launching

Build configuration: Debug

☐ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

Target settings

Connection: proj-test_Debug_B4860_Download

☑ Execute reset sequence

☑ Execute initialization script(s)

The connection is for a multicore target. Please select a core:

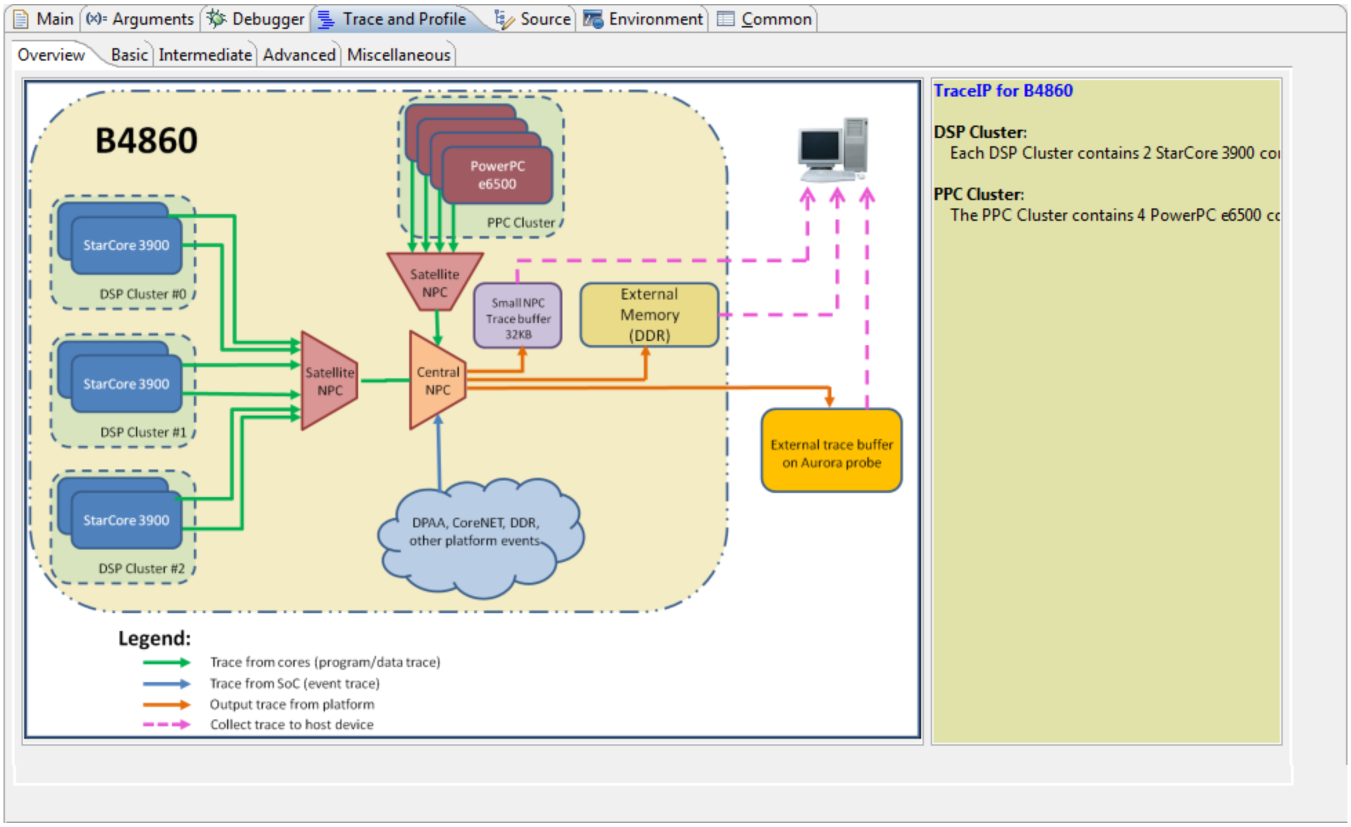
Target
☑ B4860
☐ e6500-0
☐ e6500-1

Figure 3-17. Debug Configurations Dialog Box - Main Page

To configure the launch configuration for measurement of data:

1. Click the **Trace and Profile** tab. This tab has **Overview** page, **Basic** page, **Intermediate** page, **Advanced** and **Miscellaneous** page.

   The **Overview** page displays the flow diagram for collecting trace.



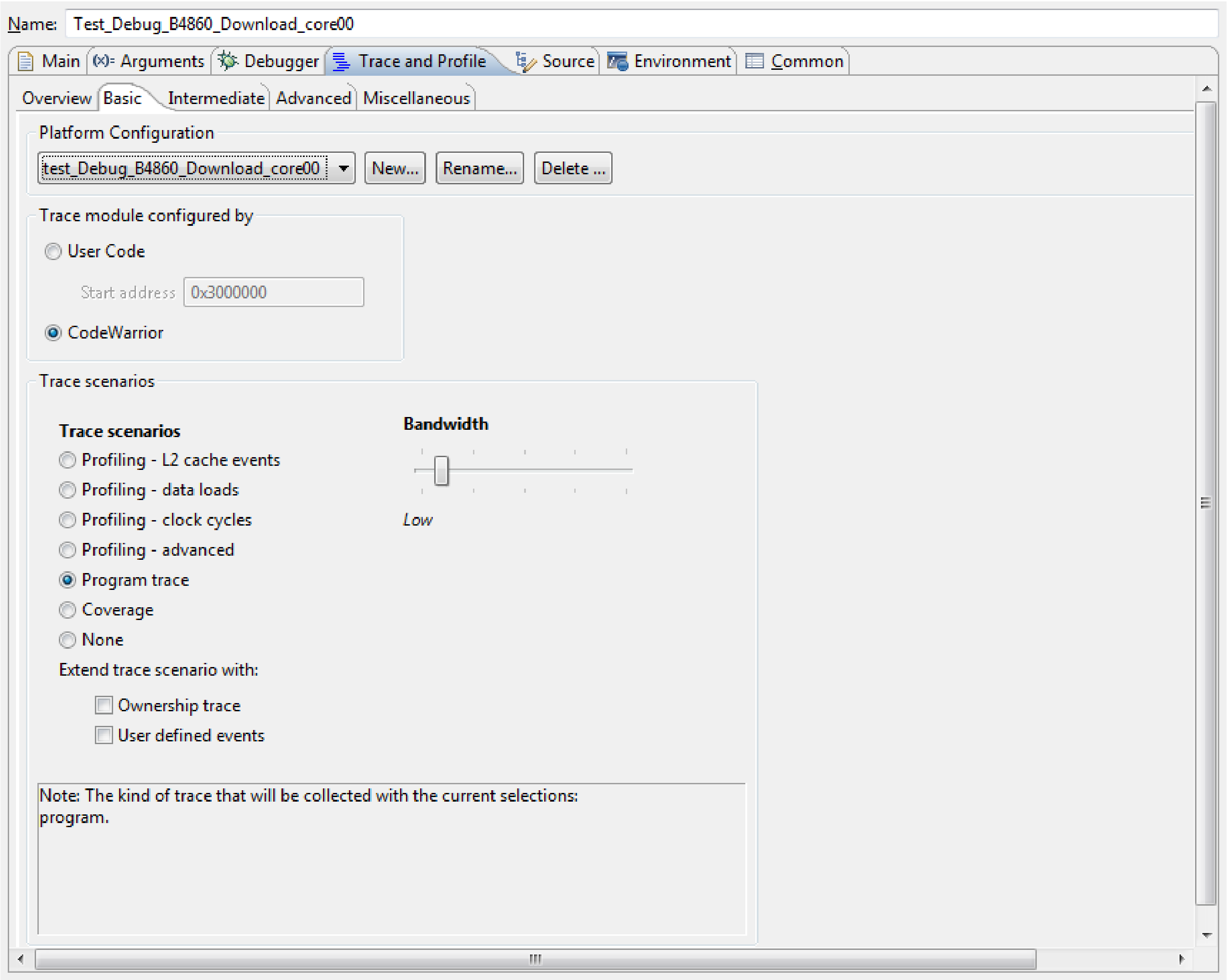
TraceIP for B4860

DSP Cluster:
Each DSP Cluster contains 2 StarCore 3900 co

PPC Cluster:
The PPC Cluster contains 4 PowerPC e6500 c

Figure 3-18. Trace and Profile Tab - Overview Page

2. In the **Basic** page, you need to configure platform configuration, trace module and trace scenarios.



Name: Test_Debug_B4860_Download_core00

Platform Configuration

test_Debug_B4860_Download_core00 ▾ New... Rename... Delete ...

Trace module configured by

○ User Code

Start address 0x3000000

● CodeWarrior

Trace scenarios

Trace scenarios

○ Profiling - L2 cache events

○ Profiling - data loads

○ Profiling - clock cycles

○ Profiling - advanced

● Program trace

○ Coverage

○ None

Extend trace scenario with:

□ Ownership trace

□ User defined events

Bandwidth

Low

Note: The kind of trace that will be collected with the current selections:
program.

Figure 3-19. Trace and Profile Tab - Basic Page

The below table lists the basic trace and profile options required for hardware profiling.

Table 3-2. Trace and Profile Basic Settings

Options	Description
Platform Configuration	allows to select the **Platform Configuration** file from the drop-down list. A platform configuration file holds the settings made in the Trace and Profile tab page for a project. Changes made in the configuration file of a project will automatically gets reflected in all the projects sharing the same configuration file. You can also perform the following actions on a configuration file: • **New**: Creates a new configuration file with default settings

Table continues on the next page...

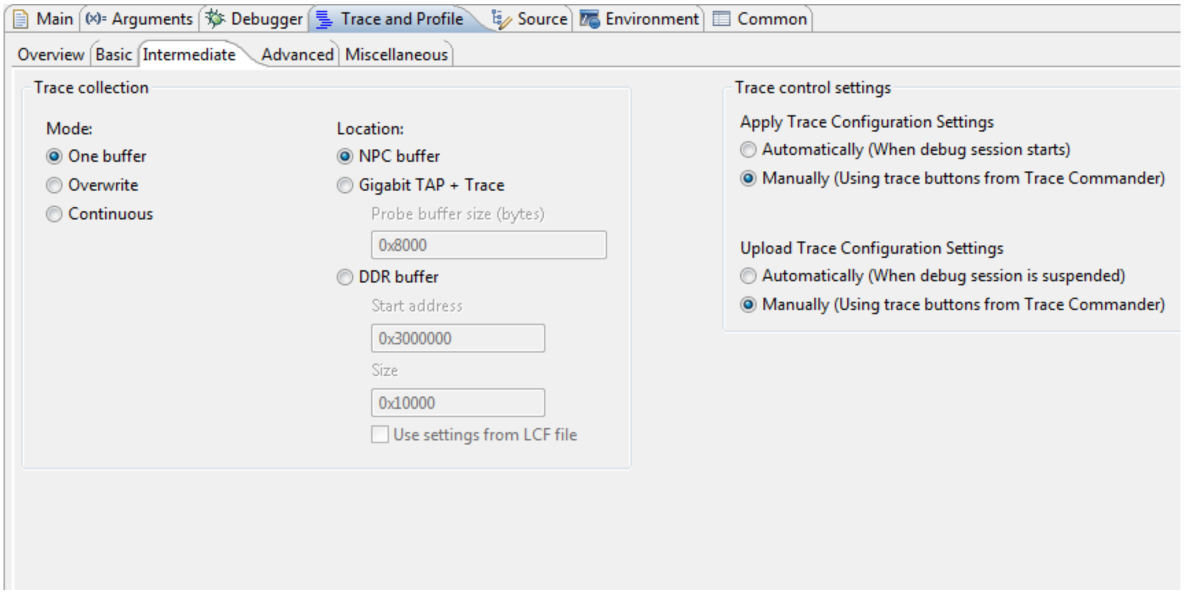
Table 3-2. Trace and Profile Basic Settings (continued)

Options	Description
	• **Rename**: Renames the currently selected configuration file. • **Delete**: Deletes the currently selected configuration file.
Trace module configured by	
User code	allows to perform trace configuration settings from coderun on target
CodeWarrior	allows to do the trace settings using CodeWarrior configuration windows
Trace Scenarios	
Profiling - L2 cache events	trace values of counters of both triad A and B on subroutine/interrupt call/return instructions
Profiling - data loads	trace values of counters of both triad A and B on subroutine/interrupt call/return instructions
Profiling - clock cycles	trace values of counters of triad A on subroutine/interrupt call/return instructions
Profiling - advanced	traces each change of flow instructions
Program trace	allows program trace to be collected
Coverage	for C source lines, displays the percentage of number of assembly instructions executed from the total number of assembly instructions corresponding to the source line. For assembly source lines, it shows if the instructions were executed or not
None	no trace will be collected
Extend Trace Scenario with:	
Ownership Trace	traces information on current task ID. Ownership trace facilitates tracking the active operating system task by providing visibility to the special purpose registers designated for use by the OS for process ID
User defined events	traces any write to TMDAT and TMTAG core registers

NOTE

For B4 targets, the ownership trace messages will be generated into trace stream only if **Program trace** or **Coverage trace** options available under Trace scenarios are also selected with **Ownership trace** option.

3. In the **Intermediate** tab, you need to configure mode of trace collection and trace control settings.



Trace collection

Mode:

● One buffer
○ Overwrite
○ Continuous

Location:

● NPC buffer
○ Gigabit TAP + Trace
Probe buffer size (bytes) 0x8000
○ DDR buffer
Start address 0x3000000
Size 0x10000
□ Use settings from LCF file

Trace control settings

Apply Trace Configuration Settings
○ Automatically (When debug session starts)
● Manually (Using trace buttons from Trace Commander)

Upload Trace Configuration Settings
○ Automatically (When debug session is suspended)
● Manually (Using trace buttons from Trace Commander)

Figure 3-20. Trace and Profile Tab - Intermediate Page

The following table describes the Intermediate trace and profile options required for hardware profiling.

Table 3-3. Trace and Profile Intermediate Settings

Options	Description
Trace Collection - Mode	
One Buffer	Stops trace collection when trace buffer gets filled.
Overwrite	Allows trace buffer to work in wrap mode. When trace buffer gets full, trace data continues to get collected overwriting existing data.
Continuous	The Continuous mode collects trace continuously till you suspend the target application. The continuous trace collection is available only when trace buffer is placed in DDR.
Trace Collection - Location	
NPC buffer	Saves trace data in NPC internal buffer.
Gigabit TAP + Trace Probe buffer size (bytes)	Saves trace data in probe buffer.
DDR buffer	Saves trace data in DDR. Magenta is the bus that transfers trace data from NPC internal buffer to DDR.
Buffer start address	Specifies the start address of the DDR where the trace data is saved. **NOTE:** The Trace buffer start address should be aligned to a block boundary of 128 bytes.
Buffer size	Specifies the DDR memory size of the region used as trace buffer.

Table continues on the next page...

Table 3-3. Trace and Profile Intermediate Settings (continued)

Options	Description
Use settings from LCF file	Uses trace buffer start address and size from linker files. File common.l3k contains _TRACE_BUFFER_size and _TRACE_BUFFER_start variables that allows you to change the start address and size. File mmu_attr.l3k contains _ENABLE_TB variable that allows to enable and disable settings for trace buffer from linker's files. **NOTE:** The Trace buffer start address should be aligned to a block boundary of 128 bytes.
Trace control settings - Apply Trace Configuration	
Automatically (When debug session starts)	The trace collection process is started automatically when the debug session is launched. **NOTE:** To successfully collect multicore trace, master core needs to be run into execution before all other cores at first resume of the debug session. If master core is core 0, use multicore resume from Debug toolbar. But if master core is not core 0, master core needs to be resumed first and then click multicore resume from Debug toolbar for all the other cores.
Manually (Using trace buttons from Trace Commander)	This is the default behavior. This option is used when you want to manually control the trace collection. The trace collection process will not be started automatically on debug session launch, but the trace configuration options are applied, when you click on **Trace Start/Stop** button available in the Trace Commander view. **NOTE:** To successfully collect multicore trace, master core needs to be run into execution before all other cores at first resume after trace is being started from Trace Commander. If master core is core 0, use multicore resume from Debug toolbar. But if master core is not core 0, master core needs to be resumed first and then click multicore resume from Debug toolbar for all the other cores.
Trace control settings - Upload Trace Configuration Settings	
Automatically (When debug session is suspended)	This is the default behavior. This option is used to upload the trace data automatically.

Table continues on the next page...

Table 3-3. Trace and Profile Intermediate Settings (continued)

Options	Description
	NOTE: The automatic upload trace configuration setting option works only when every suspend of the master trace generator/ master core is done after all other trace generators/cores have been suspended to ensure no trace generator generates trace in trace buffer while master trace generator is uploading trace. Otherwise, the trace uploaded will be corrupted.
Manually (Using trace buttons from Trace Commander)	This option is used when you want to manually upload the trace data using **Upload Trace** button available in the Trace Commander view.

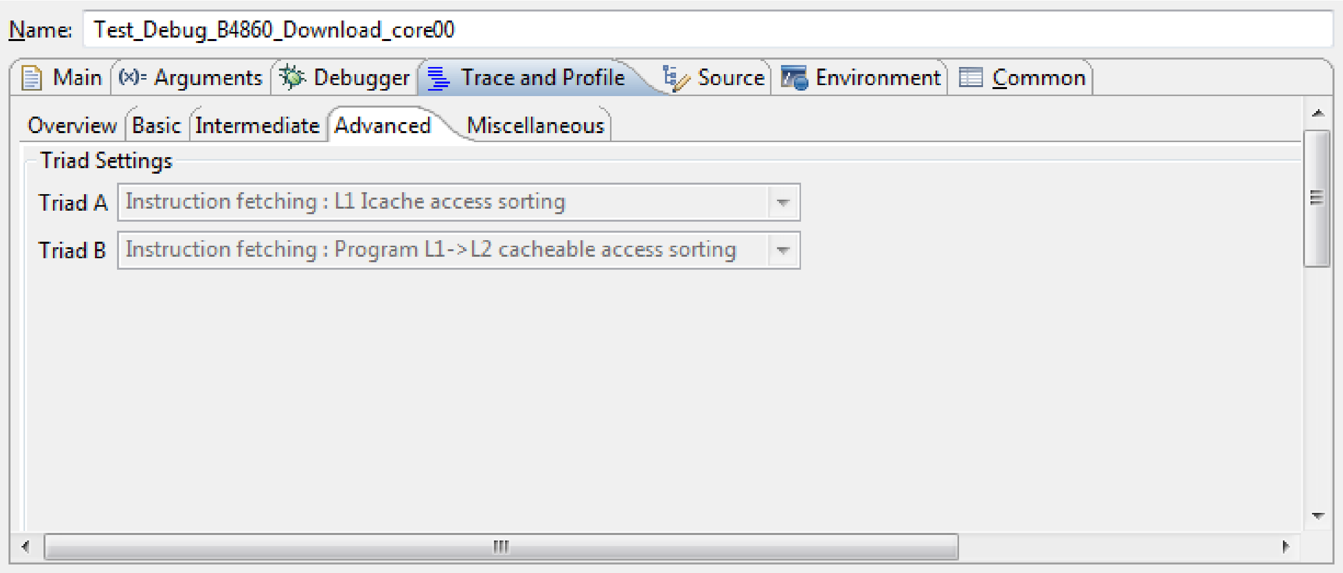
NOTE

Apply Trace Configuration and **Upload Trace Configuration** settings are disabled during debug session.

NOTE

Trace buffer for B4860 is not per core, but it uses one single buffer for all cores.

4. In the **Advanced** tab, you need to configure triad settings if you have selected **Profiling - advanced** option available under the **Trace scenarios** list in **Basic** tab. After selecting **Profiling - advanced** option, the default profiling events are mapped on **Triad A - L1 Icache Access Sorting** and **Triad B - Program L1-> L2 Cacheable Access Sorting**.



Name: Test_Debug_B4860_Download_core00

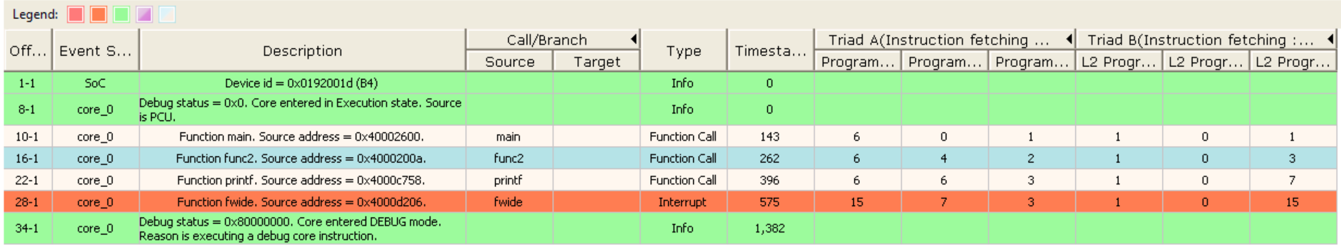
Triad A: Instruction fetching : L1 Icache access sorting

Triad B: Instruction fetching : Program L1->L2 cacheable access sorting

Figure 3-21. Trace and Profile Tab - Advanced Page

The profiling sampling events can be associated to triad A or triad B or both of them.

For example, if you select **Core Events - branch prediction : Branch Prediction 1: High level** events in Triad A and for sampling **Instruction Fetching : Program access holds** in Triad B the following trace is displayed.



Legend: 🟥 🟧 🟩 🟪 ⬜

Off...	Event S...	Description	Call/Branch		Type	Timesta...	Triad A(Instruction fetching ...			Triad B(Instruction fetching :...		
			Source	Target			Program...	Program...	Program...	L2 Progr...	L2 Progr...	L2 Progr...
1-1	SoC	Device id = 0x0192001d (B4)			Info	0						
8-1	core_0	Debug status = 0x0. Core entered in Execution state. Source is PCU.			Info	0						
10-1	core_0	Function main. Source address = 0x40002600.	main		Function Call	143	6	0	1	1	0	1
16-1	core_0	Function func2. Source address = 0x4000200a.	func2		Function Call	262	6	4	2	1	0	3
22-1	core_0	Function printf. Source address = 0x4000c758.	printf		Function Call	396	6	6	3	1	0	7
28-1	core_0	Function fwide. Source address = 0x4000d206.	fwide		Interrupt	575	15	7	3	1	0	15
34-1	core_0	Debug status = 0x80000000. Core entered DEBUG mode. Reason is executing a debug core instruction.			Info	1,382						

Figure 3-22. Trace Data after Configuring Triads A and B

5. In the **Miscellaneous** tab, specify the path for storing the trace and profile results. Click **Browse** to select the path for the **Output Folder**.

3.4 Collecting Trace

This section explains the trace collection methods for hardware and simulator based projects.

In this topic:

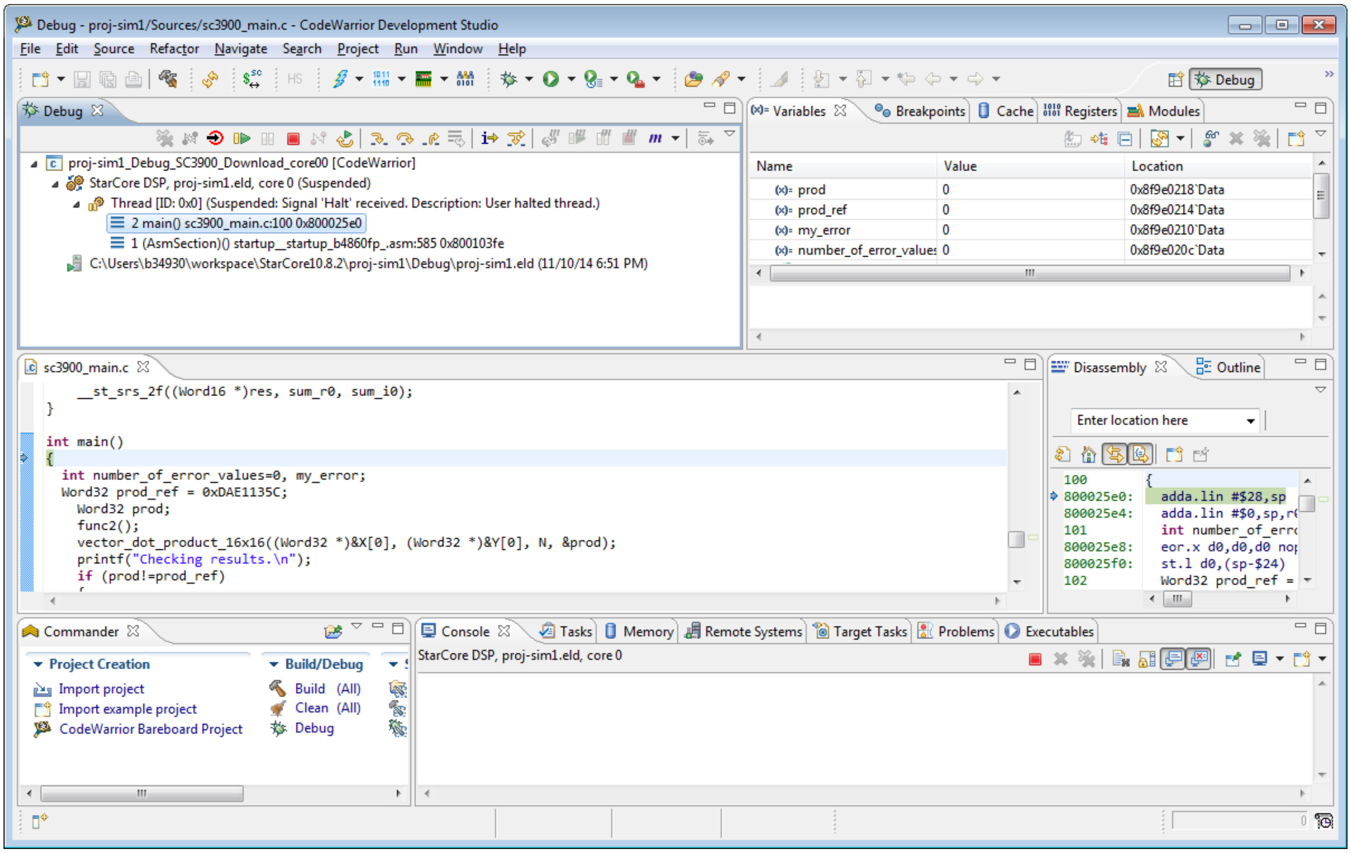
- Using Simulator
- Using Hardware

3.4.1 Using Simulator

This section explains the trace collection methods for simulator projects.

To start the trace collection:

In the Debug window, click **Debug** to launch the project. The application should halt at the beginning of main in the **Debug** perspective. If you want to modify the trace configuration, click **Configure Trace** button from the **Trace commander** view. You can reconfigure the trace during the debug session. You can apply the new settings, all the data collected before will be deleted and when resuming the session, new trace with be collected with the new configuration.



proj-sim1_Debug_SC3900_Download_core00 [CodeWarrior]
StarCore DSP, proj-sim1.eld, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
2 main() sc3900_main.c:100 0x800025e0
1 (AsmSection)() startup__startup_b4860fp_.asm:585 0x800103fe
C:\Users\b34930\workspace\StarCore10.8.2\proj-sim1\Debug\proj-sim1.eld (11/10/14 6:51 PM)

Name	Value	Location
prod	0	0x8f9e0218 Data
prod_ref	0	0x8f9e0214 Data
my_error	0	0x8f9e0210 Data
number_of_error_values	0	0x8f9e020c Data

```
    __st_srs_2f((Word16 *)res, sum_r0, sum_i0);
}

int main()
{
  int number_of_error_values=0, my_error;
  Word32 prod_ref = 0xDAE1135C;
    Word32 prod;
    func2();
    vector_dot_product_16x16((Word32 *)&X[0], (Word32 *)&Y[0], N, &prod);
    printf("Checking results.\n");
    if (prod!=prod_ref)
```

```
100          {
800025e0:      adda.lin #$28,sp
800025e4:      adda.lin #$0,sp,r
101            int number_of_erro
800025e8:      eor.x d0,d0,d0 nop
800025f0:      st.l d0,(sp-$24)
102            Word32 prod_ref =
```

StarCore DSP, proj-sim1.eld, core 0

Figure 3-23. Debug Perspective - Simulator Screen

If you have selected **Manual** option in the **Intermediate** page of **Trace and Profile** tab, you should first click **Start Trace Collection** button available in the **Trace commander** view and then **Resume** button to resume the execution and begin measurement. Let the application run for several seconds before performing the next step. During the debug session, the cycle counter value is enabled and non-zero. When the measurement is stopped, you can view the value of the cycle counter on the status bar as shown below. This value changes after every second when the application is running.

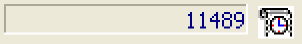


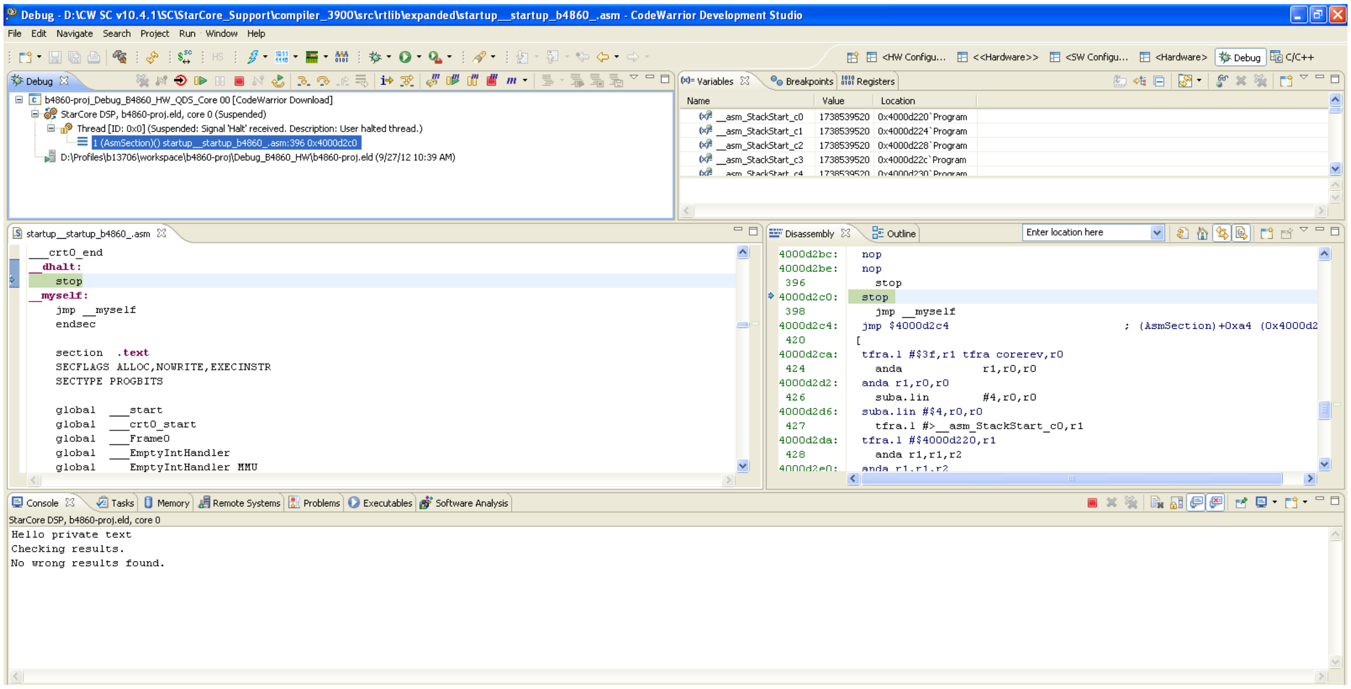
Figure 3-24. Cycle Counter

If you want to stop the execution of the application while it is running, click **Suspend**. When you suspend the application, the cycle counter also stops. When the application has executed completely, click **Terminate** to stop the measurement. When the debug session is terminated, the value of cycle counter becomes zero and disabled.

3.4.2 Using Hardware

This section explains the trace collection methods for hardware projects.

To start the trace collection, in the Debug window, click **Debug** to launch the project.:



b4860-proj_Debug_B4860_HW_QDS_Core 00 [CodeWarrior Download]
StarCore DSP, b4860-proj.eld, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
1 (AsmSection)() startup__startup_b4860_.asm:396 0x4000d2c0
D:\Profiles\b13706\workspace\b4860-proj\Debug_B4860_HW\b4860-proj.eld (9/27/12 10:39 AM)

Name	Value	Location
__asm_StackStart_c0	1738539520	0x4000d220`Program
__asm_StackStart_c1	1738539520	0x4000d224`Program
__asm_StackStart_c2	1738539520	0x4000d228`Program
__asm_StackStart_c3	1738539520	0x4000d22c`Program
__asm_StackStart_c4	1738539520	0x4000d230`Program

```
___crt0_end
__dhalt:
    stop
__myself:
    jmp __myself
    endsec

    section  .text
    SECFLAGS ALLOC,NOWRITE,EXECINSTR
    SECTYPE PROGBITS

    global  ___start
    global  ___crt0_start
    global  ___Frame0
    global  ___EmptyIntHandler
    global     EmptyIntHandler MMU
```

```
4000d2bc:  nop
4000d2be:  nop
 396         stop
4000d2c0:  stop
 398         jmp __myself
4000d2c4:  jmp $4000d2c4                        ; (AsmSection)+0xa4  (0x4000d2
 420       [
4000d2ca:  tfra.l #$3f,r1 tfra corerev,r0
 424         anda            r1,r0,r0
4000d2d2:  anda r1,r0,r0
 426         suba.lin        #4,r0,r0
4000d2d6:  suba.lin #$4,r0,r0
 427         tfra.l #>__asm_StackStart_c0,r1
4000d2da:  tfra.l #$4000d220,r1
 428         anda r1,r1,r2
4000d2e0:  anda r1,r1,r2
```

StarCore DSP, b4860-proj.eld, core 0

```
Hello private text
Checking results.
No wrong results found.
```

Figure 3-25. Debug Perspective - Hardware Screen

For more information on debugging and collecting trace data, refer to the Using Simulator topic.

3.5 Viewing Data

You can view various types of data collected on an application using the Software Analysis view.

On resuming the application, the measurement starts and the trace, timeline, critical code, performance, and call tree data is generated.

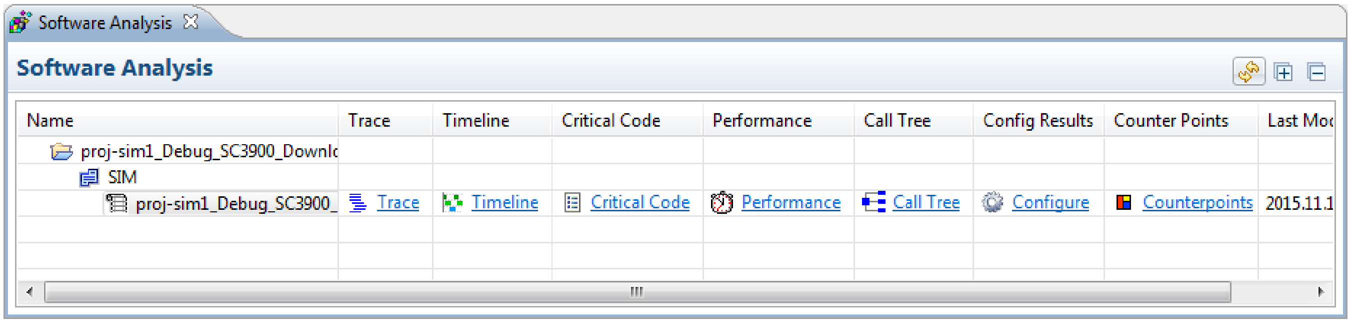
Profiling data is not available during debug session, it is available only when you terminate the application. The hyperlinks of profiling results of **Software Analysis** view are disabled during debug session. Also, if trace collection is stopped before debug session gets terminated, the hyperlinks from Software Analysis view for profiling features will be disabled. Click **Refresh** button in the Software Analysis view to update the hyperlinks to view the profiling trace data.

To view a data file that is created while running the application on the simulator or hardware:

1. From the menu bar, select **Window > Show View > Other > Software Analysis > Software Analysis** to open the **Software Analysis** view.
2. After collecting the data, the data file will be listed in the **Software Analysis** view as shown in the figure.
   In the **Software Analysis** view, you can **Copy Line** or **Cell** and also **Delete Results**. These options are available in the context menu of this view.

NOTE

The Software Analysis view will list the results from the output path in correspondence with the platform configuration selected for that respective project.



Name	Trace	Timeline	Critical Code	Performance	Call Tree	Config Results	Counter Points	Last Moc
proj-sim1_Debug_SC3900_Downlc								
SIM								
proj-sim1_Debug_SC3900_	Trace	Timeline	Critical Code	Performance	Call Tree	Configure	Counterpoints	2015.11.1

Figure 3-26. Software Analysis View Screen

3. Expand the datafile to view the trace and profiling data.

The data can be viewed either generated by simulator or hardware.

- Viewing Data Generated by Simulator
- Viewing Data Generated by Hardware

3.5.1 Viewing Data Generated by Simulator

This section describes how to view trace data for a simulator project in Software Analysis View.

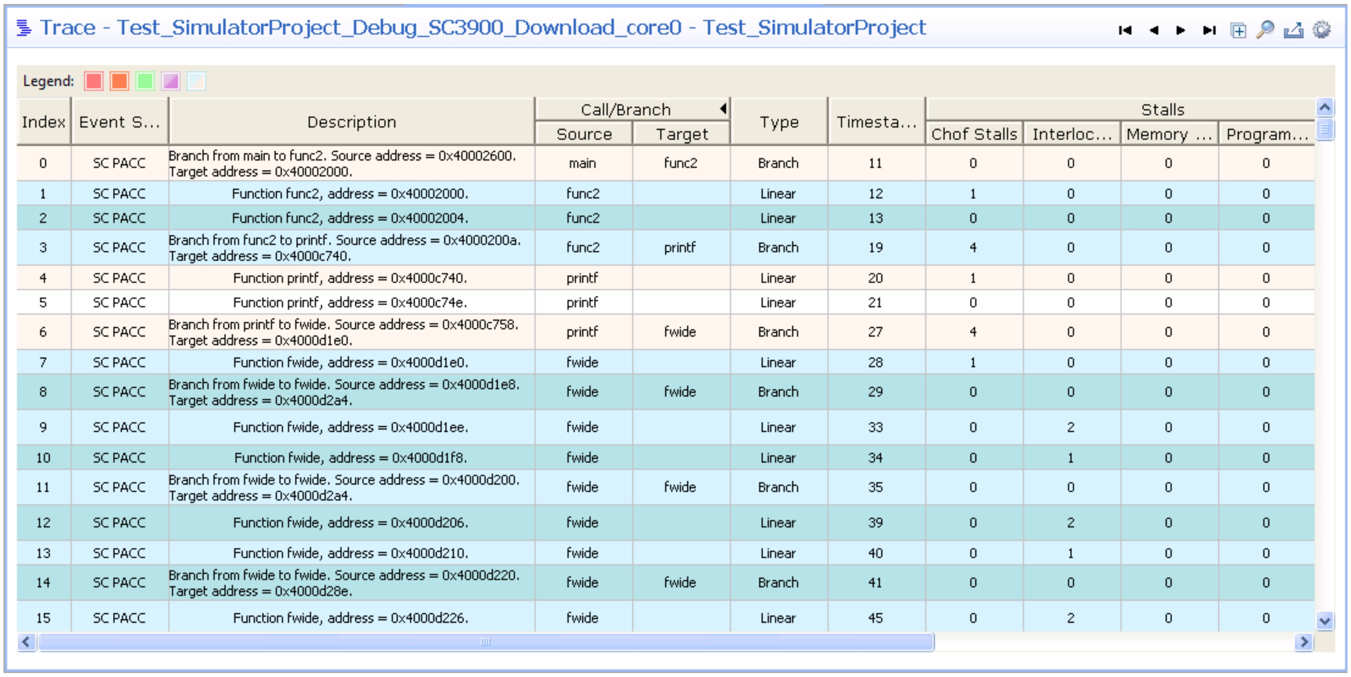
The following data can be viewed:

- Viewing Trace Data
- Viewing Timeline Data
- Viewing Critical Code Data
- Viewing Performance Data
- Viewing Call Tree Data
- Viewing Counterpoints

3.5.1.1 Viewing Trace Data

Click the **Trace** link to view the trace data generated in Software Analysis view.

The following figure shows the trace data collected for a simulator project.



Trace - Test_SimulatorProject_Debug_SC3900_Download_core0 - Test_SimulatorProject

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...	Stalls			
			Source	Target			Chof Stalls	Interloc...	Memory ...	Program...
0	SC PACC	Branch from main to func2. Source address = 0x40002600. Target address = 0x40002000.	main	func2	Branch	11	0	0	0	0
1	SC PACC	Function func2, address = 0x40002000.	func2		Linear	12	1	0	0	0
2	SC PACC	Function func2, address = 0x40002004.	func2		Linear	13	0	0	0	0
3	SC PACC	Branch from func2 to printf. Source address = 0x4000200a. Target address = 0x4000c740.	func2	printf	Branch	19	4	0	0	0
4	SC PACC	Function printf, address = 0x4000c740.	printf		Linear	20	1	0	0	0
5	SC PACC	Function printf, address = 0x4000c74e.	printf		Linear	21	0	0	0	0
6	SC PACC	Branch from printf to fwide. Source address = 0x4000c758. Target address = 0x4000d1e0.	printf	fwide	Branch	27	4	0	0	0
7	SC PACC	Function fwide, address = 0x4000d1e0.	fwide		Linear	28	1	0	0	0
8	SC PACC	Branch from fwide to fwide. Source address = 0x4000d1e8. Target address = 0x4000d2a4.	fwide	fwide	Branch	29	0	0	0	0
9	SC PACC	Function fwide, address = 0x4000d1ee.	fwide		Linear	33	0	2	0	0
10	SC PACC	Function fwide, address = 0x4000d1f8.	fwide		Linear	34	0	1	0	0
11	SC PACC	Branch from fwide to fwide. Source address = 0x4000d200. Target address = 0x4000d2a4.	fwide	fwide	Branch	35	0	0	0	0
12	SC PACC	Function fwide, address = 0x4000d206.	fwide		Linear	39	0	2	0	0
13	SC PACC	Function fwide, address = 0x4000d210.	fwide		Linear	40	0	1	0	0
14	SC PACC	Branch from fwide to fwide. Source address = 0x4000d220. Target address = 0x4000d28e.	fwide	fwide	Branch	41	0	0	0	0
15	SC PACC	Function fwide, address = 0x4000d226.	fwide		Linear	45	0	2	0	0

Figure 3-27. Trace Data

The below table explains various fields of the trace data.

Table 3-4. Trace Data

Field	Description
Index	Displays the order number of the instructions.
Event Source	Displays program trace messages from PACC simulator.
Description	Displays detailed information about the trace line.
Source	Displays the source function of the trace line if it is a call or a branch.
Target	Displays the target function of the trace line if it is a call or a branch.
Type	Displays the type of the trace line, which can either be a linear instruction, a branch, a call, or a custom message.
Timestamp	Displays the absolute clock cycles that the instruction takes to execute.
Stalls shows the number of stalls that occurred on a PC and measured in cycles.	
ChoF Stalls	Displays the number of bubble cycles incurred by change of program flow for each instruction.
Interlock Stalls	Displays the number of bubble cycles incurred by core resource interlocks for each instruction.
Memory Stalls	Displays the number of bubble cycles incurred by data bus hold for each instruction.
Program Stalls	Displays the number of bubble cycles incurred by program starvation for each instruction.
Rewind Stalls	Displays the number of bubble cycles incurred by a rewind on the core buses for each instruction.

Table continues on the next page...

Table 3-4. Trace Data (continued)

Field	Description
BTB events shows which BTB event has taken place at the PC. It can be 0 or 1:1 if the event has taken place and 0 if no event has taken place.	
BTBable COF	The decoded COF is BTBable.
COF Taken	The COF is taken; changes the program flow.
False BTB Hit	Indicates False hit.
BTB Prediction	BTB prediction shows the status of the BTB prediction and it can be any of the following: • COF predicted correctly: Set of the COF instruction did not cause a pipe flush. • COF in BTB: BTBable COF not in the BTB and miss-predicted. It can be taken or not. • COF wrongly predicted: BTBable COF, predicted as taken in the BTB, resolved as not taken (wrongly predicted). • Buffered sequential loop wrongly predicted: Buffered LPEND incorrectly predicted (Non-BTBable LPEND.SQ with LC>1 and no internal match). • BTBable loop wrongly predicted: Either BTB hit on LPEND in last iteration or LPEND taken without BTB hit.

You can also also count the events by the profiling counters. The events are organized into groups of 3, each counted by either Triad A or B. In many cases, two events groups were defined for concurrent profiling, as they belong to the same event category.

The following table summarizes the different counting groups.

Table 3-5. Summary of event groups for profiling

Event Category	Event Group	Triad	Counter 0	Counter 1	Counter 2	Comments
Core events - interrupts	Exceptions 1	A	Trap instructions	Critical interrupts	Non-critical interrupts	-
	Exceptions 2	B	MMU exceptions	Debug exceptions	Other exceptions	-
Core events - branch prediction	Branch predication 1: High level	A	Total VLES	COF VLES (w/o hardware loops)	COF correctly predicted (w/o hardware loops)	Includes both taken and not-taken COFs.
	Branch prediction 2: BTB	B	All BTB-able COF	BTB-able COFs, wrongly predicted, not in the BTB	BTB-able COFs, wrongly predicted, in the BTB	Includes both taken and not-taken COFs.
	Hardware loop prediction	B	Total hardware end-of-loop COF (not including CONT and BREAK)	Incorrectly predicted buffered end-of-loop COF	Other incorrectly predicted end-of-loop COF (not in the BTB, in the BTB wrongly predicted, loop re-learn)	Buffered loops: those sequential loops which fit in the buffer

Table continues on the next page...

Table 3-5. Summary of event groups for profiling (continued)

Event Category	Event Group	Triad	Counter 0	Counter 1	Counter 2	Comments
Core cycles	Cycle high level	A	Application cycles	-	Bubbles	App.cycles: not in debug and low power modes
	Stall cycles 1	A	No bubbles (once per VLES)	COF (pipe flush) & dispatch starvation cycles	interlocks (RSU) w/o holds	-
	Stall cycles 2	B	Data memory holds and freezes	&program cache holds	Rewind cycles (prog & data)	-
	Fetch & Dispatch stall cycles	B	Pipe flush cycles: mispredicted COF, interdicts, exit from idle states	Stalls due to COF target spread over two fetch sets	BTB bubbles	-
	Rewind events	B	Data rewind events	Program rewind events	DTU rewind events	-
Debug events	Debug Events 1	A	IEU Q0	IEU Q1	Reload counter 0 RZ	-
	Debug Events 2	B	IEU Q2	IEU Q3	-	
	Watchpoints 1	A	-	PCDA0	PCDA2	-
	Watchpoints 2	B	-	PCDA4	PCDA6	-
	DEBUGEV 1	A	DEBUGEV.0	DEBUGEV.2	DEBUGEV.3	-
	DEBUGEV 2	B	DEBUGEV.4	DEBUGEV.6	DEBUGEV.7	
Trace performance	Trace messages	A	Total trace messages	Error messages	Messages lost	-
	Trace cycle overhead	B	Dual trace beats leaving the queue	Freeze cycles due to main queue (if not frozen due to the input queues)	Freeze cycles due to input queues	Each increment of B0 means two beats sent in parallel to the NPC
Instruction fetching	L1 Icache access sorting	A	Program access L1 hits	Program access L1 pre-fetch hits	Program access L1 miss	-
	Program L1->L2 cacheable access sorting	B	L2 Program access hits (shared/ exclusive)	L2 Program access hits (modified)	L2 Program access miss	Only cacheable accesses, including CME accesses
	L1 Program pre-fetch accesses	A	CME/granular PF accesses - L1 hit (including pre-fetch hit)	-	CME granular - L1 miss	Only program CME and cache granular accesses. ctr1 does not count
	Program access holds	B	Program cacheable hold + contention cycles	Program non-cacheable holds + miscellaneous	Program rewind events	The core may continue to execute during program holds

Table continues on the next page...

Table 3-5. Summary of event groups for profiling (continued)

Event Category	Event Group	Triad	Counter 0	Counter 1	Counter 2	Comments
Data channel - general	Data access holds - HL	B	Data cache holds	Data store freeze	Data rewind events	B0 and B1 events may overlap
	Coherency - invalidation events	A	L1 data invalidate from L2	CME/granular invalidation accesses	L1 data invalidation from internal conflicts	Only actual invalidation is counted
Data Loads	L1 Dcache load access sorting	A	Data access L1 hits	Data access L1 pre-fetch hits	Data access L1 miss	-
	Data L1-> L2 cacheable load access sorting	B	L2 Data access hits (shared/ exclusive)	L2 Data access hits (modified)	L2 Data access miss	Only cacheable accesses, including CME accesses
	L1 Data pre-fetch accesses	A	CME/granular PF accesses - L1 hit (including pre-fetch hit)	CME/granular cache pre-fetch accesses that were dropped	CME granular - L1 miss	Only data CME and cache granular accesses
	Data load holds 1	A	Data load cacheable hold cycles	Data loads non-cacheable hold cycles	Data cache contention cycles	Holds for parallel accesses counted once
	Data load holds 2	B	Full fetch queue hold cycles	miscellaneous hold cycles (barriers, special commands)	Address queue load after store hazard cycles	Address queue holds may overlap the data cache holds
	Data load special cases	A	Total memory loads	Non-aligned 4K loads	Load rewind events	-
	Data load access speculation	B	Total core accesses without cache commands and barriers	Total valid accesses	Killed accesses with cache miss	W bus accesses
Data stores	Store access sorting	A	Total store accesses	SGB merges	SGB write out hot (DLINK to L1)	No cache commands and barriers, after non-alignm ent split, merges may be 4 at a time
	SGB events	B	SGB freeze cycles	SGB read-modify-write accesses to L2	SGB accepter rewind events	For parallel accesses, a freeze is counted once
CME	Data CME commands	A	CME data commands	CME SkyBlue data commands	Granular data commands	CME command = actual channel allocation. A2 is not implemente d in the CME

Table continues on the next page...

Table 3-5. Summary of event groups for profiling (continued)

Event Category	Event Group	Triad	Counter 0	Counter 1	Counter 2	Comments
	Program CME commands	B	CME program commands	CME SkyBlue program commands	Granular program commands	-
	CME allocation	A	Total CME allocation requests (program + data)	Total successful allocations (program + data)	Total CME channel substitutions (program + data)	Substitution s = incomplete pre-fetch channel
Bus load	L1-L2 bus load	A	DLINK read beats	DLINK write beats	ILINK beats	A single load access (A0,A2) can be 2 bus cycles (beats) if cacheable. Cache commands (no-tag access) counted as writes
	SkyBlue bus load	B	Core access to local subsystem registers in the CCSR space	SkyBlue access from the SoC or adjacent core in the cluster to local subsystem registers in the CCSR space	SkyBlue access from any source to the local subsystem registers in the DCSR space	-
L2 cache statistics	L2 demand (Elink) accesses - 1	A	Total L2 demand accesses	L2 program accesses	L2 data accesses	Each counter counts up to 4 events per cycle
	Each counter counts up to 4 events L2 demand per cycle	B	Total L2 hit	L2 instruction miss	L2 data miss	
	L2 AOUT master requests to CoreNet	A	Total L2 AOUT requests	L2 AOUT requests sent as global (M=1)	L2 AOUT data side requests (including barriers)	-
	L2 total traffic	B	Total L2 demand (ELINK) accesses	L2 Snoop requestsc	Tot. L2 accesses from all sources	Counters 0 and 2 may count up to 4 events per cycle
	L2 external accesses	A	L2 Total stashes	L2 Snoop requests	L2 Stash requests degraded to snoop	KEXT should be counted with either KDIN or KSNP
	L2 DIN	B(a)	L2 reload requests from CoreNet	L2 Store allocate	-	
	L2 snooping	B(b)	L2 Snoop hit	L2 snoops causing MINT	L2 snoops causing SINT	

To copy the cell that is currently selected, select the **Copy Cell** option from the context menu. To copy the complete line of the data source, select the **Copy Line** option. If you want to resize the row or column, select the **Auto Resize Row** or **Auto Resize Column Width** option from the context menu.

The Trace viewer displays two types of trace data:

- Partially decoded trace data
- Fully decoded trace data

Full decoded trace is displayed if prior to opening Trace viewer any of the profiling viewers (Timeline, Critical Code, Performance, Call Tree, Counterpoints) is already open. If none of the profiling viewers has been opened, then Trace viewer will display the partial decoded trace or trace decoded on chunks. Once trace is decoded, Trace viewer will always display full decoded trace.

Partial decoded trace has certain limitations because current chunk of trace to be decoded might not contain all information needed to decode that chunk, for example, chunk might lack PC synchronization or RAS stack information in scenerios when trace is collected in overwrite mode or multicore core. Such limitations happen only on program trace and not on profiling trace.

The following table list the details of each trace viewer events showing the limitations of partial decoded trace:

Table 3-6. Trace viewer events

Trace Events	Description
Periodic synchronization	After periodic synchronization message no trace dropping should happen.
Trace dropped - no PC synchronization	Trace has been dropped because no message with PC (program counter) has been analyzed.
Trace partially recovered - PC synchronization but no RAS stack synchronization	Trace has partially recovered due to a message providing PC. Message however, does not ensures a synchronization for RAS stack. Therefore, it is expected that trace to be dropped once destination address for a RTS instruction from RAS stack will be needed.
Trace dropped - RTS treated as direct COF but RAS stack is empty	Trace has been dropped, because RTS instruction analyzed requires its destination from RAS stack but stack is empty.
Trace fully recovered - PC and RAS stack synchronization	Trace has fully recovered due to a message that provides both PC and RAS stack synchronized. This kind of message is expected to come once with Periodic synchronization message.
Trace dropped - certain type of instruction incorrectly traced	Trace has been dropped due to an unexpected error in decoding algorithm.
Call	It reports a call instruction from a source function to a destination function.

Table continues on the next page...

Table 3-6. Trace viewer events (continued)

Trace Events	Description
Return	It reports a return instruction from a source function to a destination function.
Branch	It reports a branch instruction from a source function to a destination function.
Linear	It reports a linear instruction.
Debug Status	It reports core state.
Device Id	It reports device id.
User defined	It reports any write made by user to TMDAT and TMTAG core registers.
Ownership	It reports information on current task ID.
Profiling counter overflow	It reports an overflow of the profiling counters.

The **Trace Data** viewer provides a color convention in the form of **Legend** to highlight trace events and source changes. The details on the color convention is listed in the table below:

Table 3-7. Trace Data

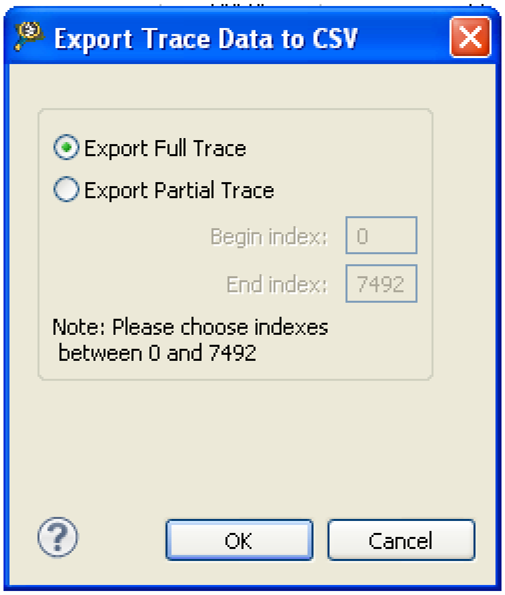
Color	Description
Red	Highlights the error events reported in the trace data.
Orange	Highlights interrupt jumps reported in both program and profiling trace data.
Green	Highlights the info events reported in the trace data.
Purple (Light and Dark)	Highlights the data events reported in the trace data; light purple alternates with dark purple for odd and even lines.
Blue (Light and Dark) and White (White and Seashell)	Highlights all other trace events. Blues alternate with whites as source changes. Light blue alternates with dark blue for odd and even lines. White alternates with seashell for odd and even lines.

3.5.1.1.1 Exporting Trace Data

In the trace data viewer you can export trace data to a CSV file.

Perform the following steps to export trace data:

1. Click to display the **Exporting Trace Data** dialog box.



Export Trace Data to CSV

● Export Full Trace

○ Export Partial Trace

Begin index: 0

End index: 7492

Note: Please choose indexes between 0 and 7492

Figure 3-28. Exporting Trace Data Dialog Box

2. Select the **Export Full Trace** option to export complete trace data to a CSV file else select **Export Partial Trace**.
3. Specify the begin and end index of the trace data if you have selected the **Export Partial Trace** option.
4. Click **OK**. The **Export Trace Data to CSV** dialog box appears.
5. Browse to the location where you want to save the trace data.

3.5.1.1.2 Configuring Time Unit and Time Format

The **Configure Table** button allows you to **Configure Time Unit** and **Configure Time Format** actions.

- **Configure Time Unit** - Lets you set CPU frequency and convert the clock cycles, displayed in the **Timestamp** column, into real time in milliseconds, microseconds, or nanoseconds. Click this button and select the **Configure Time Unit** option as shown in the figure. To convert the clock cycles into milliseconds, microseconds, or nanoseconds, select the corresponding option.

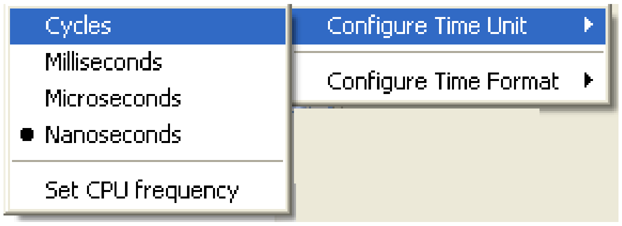


Figure 3-29. Configure Time Unit

The **Set CPU Frequency** option allows you to set the CPU frequency needed to convert the clock cycles into real time. Select this option to display the **Set CPU Frequency** dialog box. Set the new CPU frequency according to requirements.

- **Configure Time Format** - Enables the **Timestamp** column of the Trace Data viewer for time formatting, **Radix**: *Hexadecimal or Decimal* and **Value**: *Absolute or Delta* as shown in the figure below. Time radix formatting is available only if time unit is set to **Time in cycles**. The default format of time is displayed in Decimal and Absolute value. Delta is calculated for same source event. The Delta value for a specific timestamp is the difference between current timestamp and previous timestamp with same source event as current one. If Delta value is negative, it will be displayed as zero.

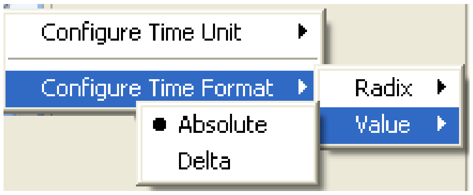
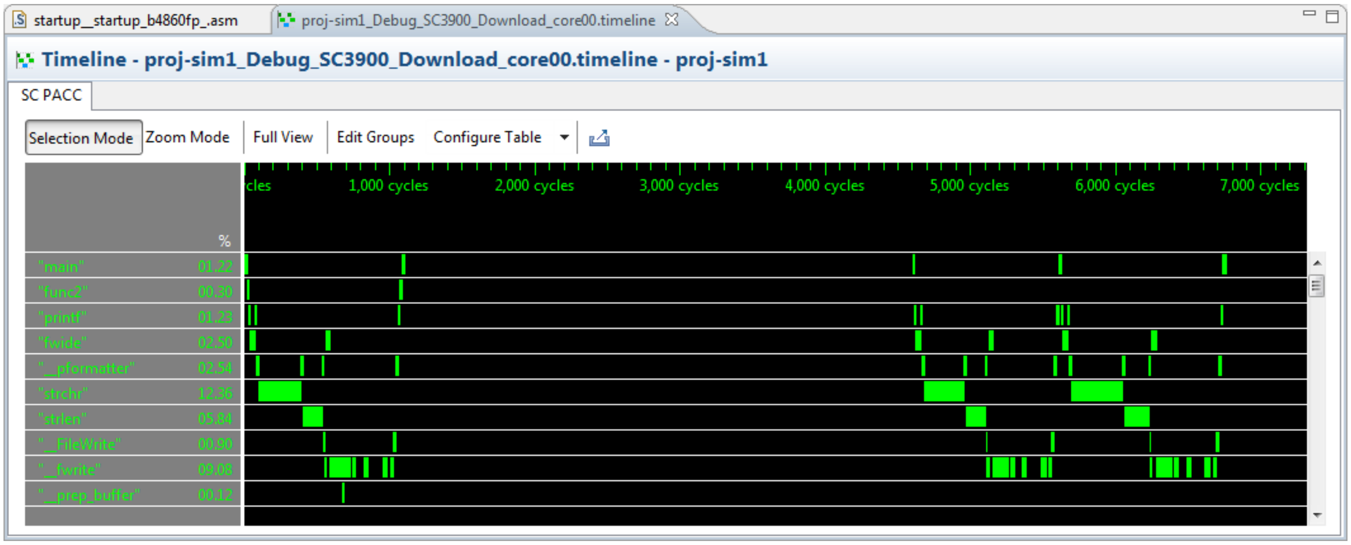


Figure 3-30. Configure Time Format

3.5.1.2 Viewing Timeline Data

To view the timeline data, click the **Timeline** link.

The following figure shows an example of Timeline data.



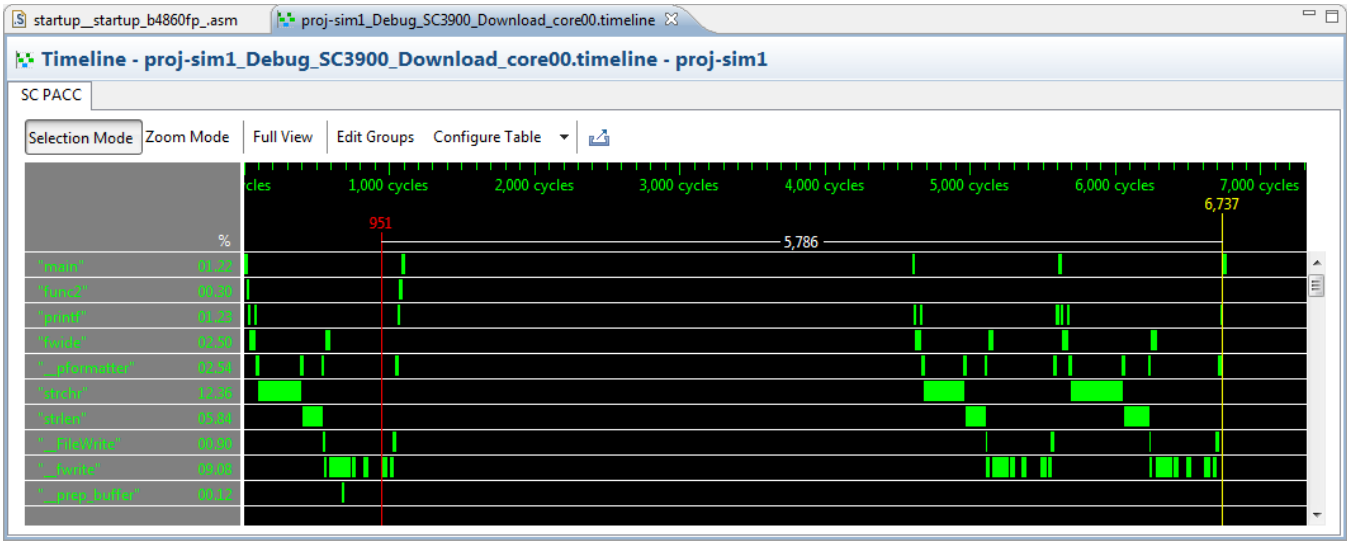
	%
"main"	01.22
"func2"	00.30
"printf"	01.23
"fwide"	02.50
"__pformatter"	02.54
"strchr"	12.36
"strlen"	05.84
"__FileWrite"	00.90
"__fwrite"	09.08
"__prep_buffer"	00.12

Figure 3-31. Timeline Data

The timeline data displays the functions that are executed in the application and the number of cycles each function takes when the application is run. The **Timeline** viewer shows the function percent of the completion status for each function in the left panel and timeline graph in the right panel with the functions aligned on y-axis and the number of cycles on x-axis. The green-colored bars show the time and cycles that the function takes.

The Timeline viewer have the following buttons:

- **Selection Mode** - Allows you to mark points in the function bars to measure the difference of cycles between those points. To mark a point in the bar:
  - Click **Selection Mode**.
  - Click on the bar where you want to mark the point.
  - A yellow vertical line appears displaying the number of cycles at that point.
  - Right-click on another point in the bar.
  - A red vertical line appears displaying the number of cycles at that point along with the difference of cycles between two marked points as shown in the below figure.



	%
"main"	01.22
"func2"	00.30
"printf"	01.23
"fwide"	02.50
"__pformatter"	02.54
"strchr"	12.36
"strlen"	05.84
"__FileWrite"	00.90
"__fwrite"	09.08
"__prep_buffer"	00.12

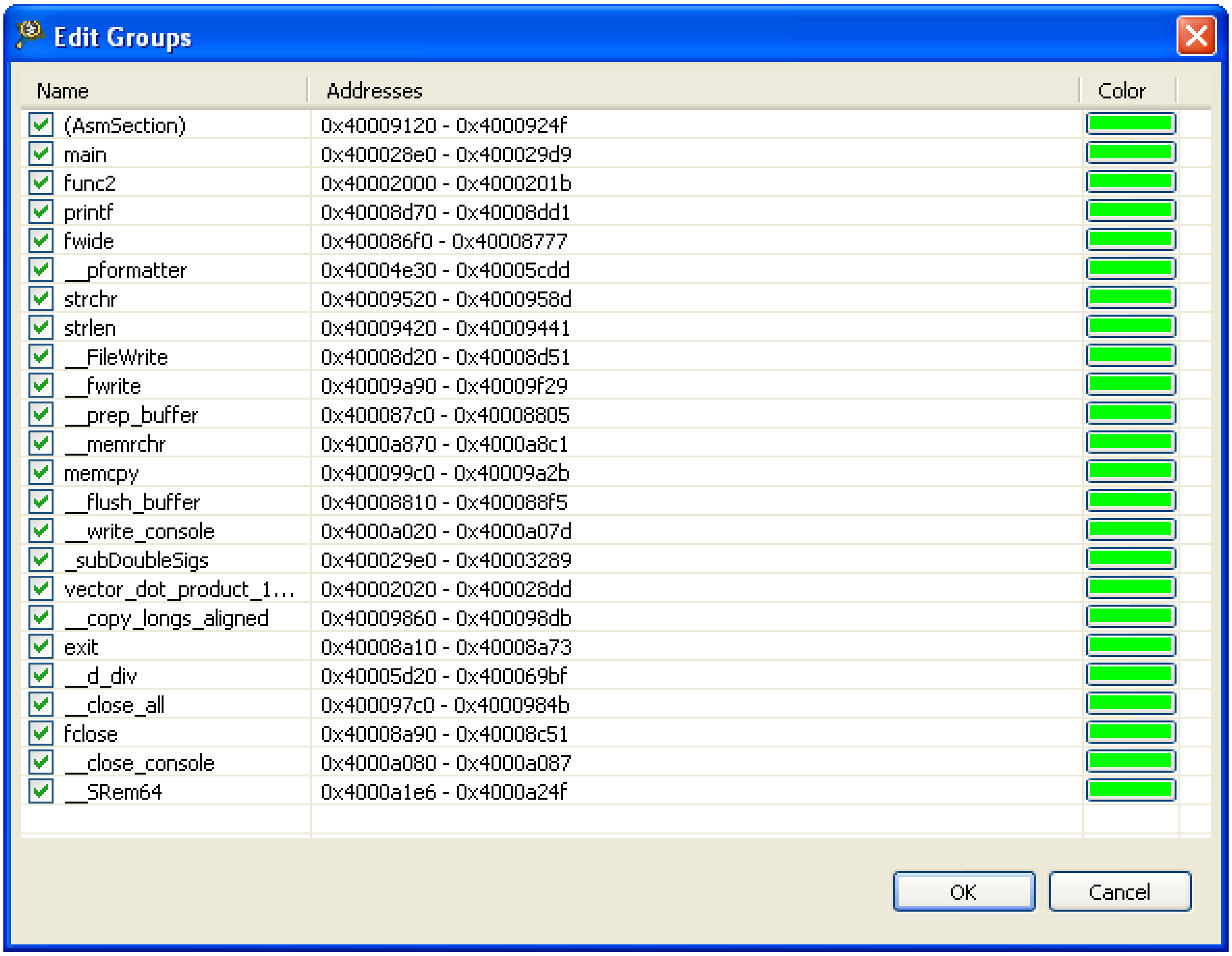
Figure 3-32. Selection Mode to Measure Difference of Cycles Between Functions

- **Zoom Mode** - Allows you to zoom-in and zoom-out in the timeline graph. Click **Zoom Mode** and then click on the timeline graph to zoom-in. To zoom-out, right-click on the timeline graph. You can also move the mouse wheel up and down to zoom-in and zoom-out.
- **Full View** - Allows you to get back to the original view if you selected the zoom mode.

NOTE

The **Selection Mode** is the default mode of the timeline view.

- **Configure Table** - Lets you configure time unit in cycles, milliseconds, microseconds, and nanoseconds. For example, to display time displayed in the timeline graph in milliseconds, select **Configure Table > Configure Time Unit > Milliseconds**. It also allows you to set CPU frequency. For details, refer to the Configuring Time Unit and Time Format topic.
- **Export timeline graphic to file** - Allows you to save the data visible in the timeline view as an image in .JPG or .PNG format.
- **Edit Groups** - Lets you customize the timeline according to your requirements. For example, you can change the default color of the line bars representing the functions to differentiate between them. You can add/remove a function to/from the timeline. To perform these functions, select **Edit Groups**. The **Edit Groups** dialog box appears.



Name	Addresses	Color
☑ (AsmSection)	0x40009120 - 0x4000924f	🟩
☑ main	0x400028e0 - 0x400029d9	🟩
☑ func2	0x40002000 - 0x4000201b	🟩
☑ printf	0x40008d70 - 0x40008dd1	🟩
☑ fwide	0x400086f0 - 0x40008777	🟩
☑ __pformatter	0x40004e30 - 0x40005cdd	🟩
☑ strchr	0x40009520 - 0x4000958d	🟩
☑ strlen	0x40009420 - 0x40009441	🟩
☑ __FileWrite	0x40008d20 - 0x40008d51	🟩
☑ __fwrite	0x40009a90 - 0x40009f29	🟩
☑ __prep_buffer	0x400087c0 - 0x40008805	🟩
☑ __memrchr	0x4000a870 - 0x4000a8c1	🟩
☑ memcpy	0x400099c0 - 0x40009a2b	🟩
☑ __flush_buffer	0x40008810 - 0x400088f5	🟩
☑ __write_console	0x4000a020 - 0x4000a07d	🟩
☑ _subDoubleSigs	0x400029e0 - 0x40003289	🟩
☑ vector_dot_product_1…	0x40002020 - 0x400028dd	🟩
☑ __copy_longs_aligned	0x40009860 - 0x400098db	🟩
☑ exit	0x40008a10 - 0x40008a73	🟩
☑ __d_div	0x40005d20 - 0x400069bf	🟩
☑ __close_all	0x400097c0 - 0x4000984b	🟩
☑ fclose	0x40008a90 - 0x40008c51	🟩
☑ __close_console	0x4000a080 - 0x4000a087	🟩
☑ __SRem64	0x4000a1e6 - 0x4000a24f	🟩

Figure 3-33. Edit Group Dialog Box

You can perform the following operations in the **Edit Groups** dialog box:

- Add or Remove Function
- Edit Address Range of Function

- Change Color
- Add or Remove Group
- Merge Groups or Functions

3.5.1.2.1 Add or Remove Function

This section explains how to add or remove a function from the **Edit Groups** dialog box.

Right-click on the function name in the Name column, and select **Insert Function** or press **Ctrl+F** to add a function. Select **Delete Selected** from the context menu to delete the function from the graph. You can disable a function from the graph by clearing the corresponding checkbox in the **Name** column. Check it again to include it in the graph.

3.5.1.2.2 Edit Address Range of Function

This section explains how to edit the address range of a function from the **Edit Groups** dialog box.

Perform the following steps:

1. Select the function of which you want to change the address range.
2. Double-click on the cell of the **Addresses** column of the selected function.

   The cell becomes editable.

3. Type an address range for the group or function in the cell.

NOTE

You can specify multiple address ranges to a function. The multiple address ranges are separated by a comma.

3.5.1.2.3 Change Color

This section explains how to change the color of a function in the timeline graph.

The color appears as a horizontal bar in the graph. Click on the **Color** column of the corresponding function, and select the color of your choice from the **Color** window that appears.

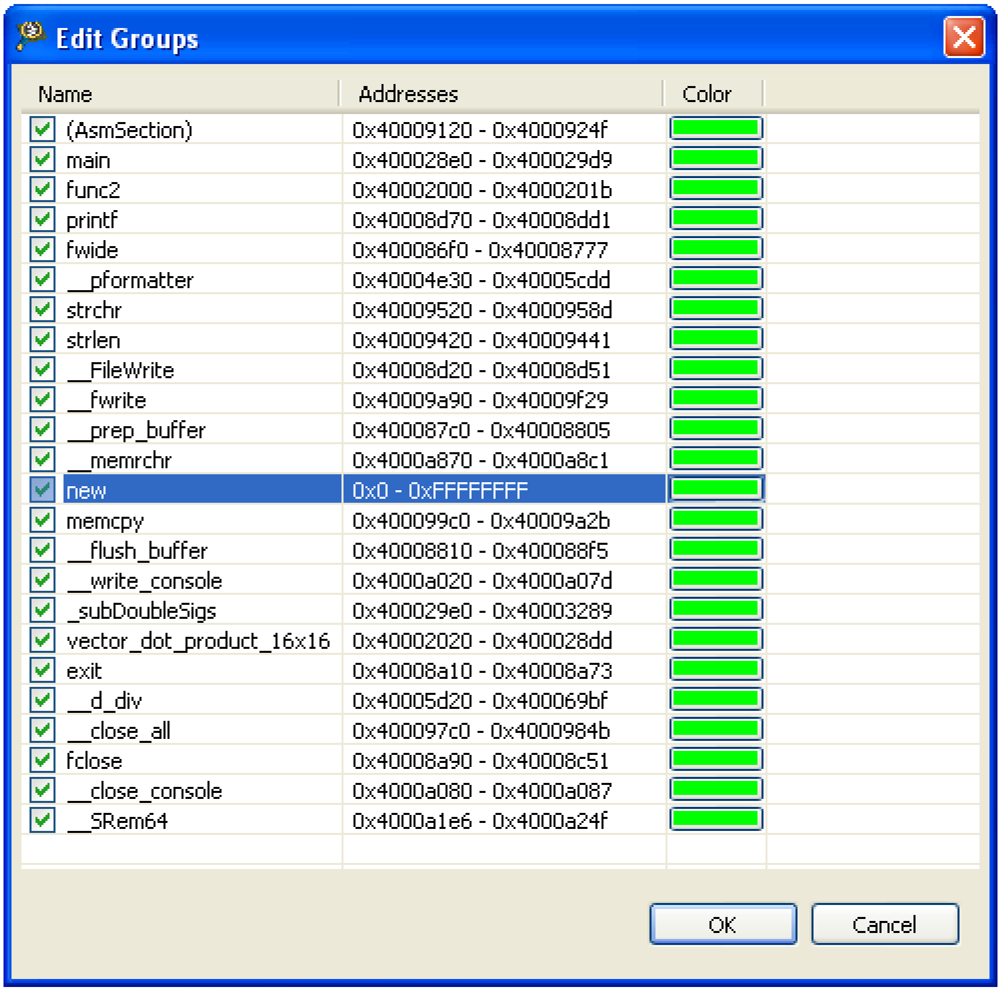
3.5.1.2.4 Add or Remove Group

This section explains how to add or remove a group from **Edit Groups** dialog box.

A group is a range of addresses. In case, you want to view trace of a part of a function only, for example, for loop, you can find the addresses of the loop and create a group for those addresses. Perform the following steps to add a group:

1. Right-click on the row, in the **Edit Groups** dialog box, where you want to insert a group, and select **Insert Group** from the context menu. Alternatively, press the *Ctrl +G* key.

   A row is added to the table with `new` as function name.



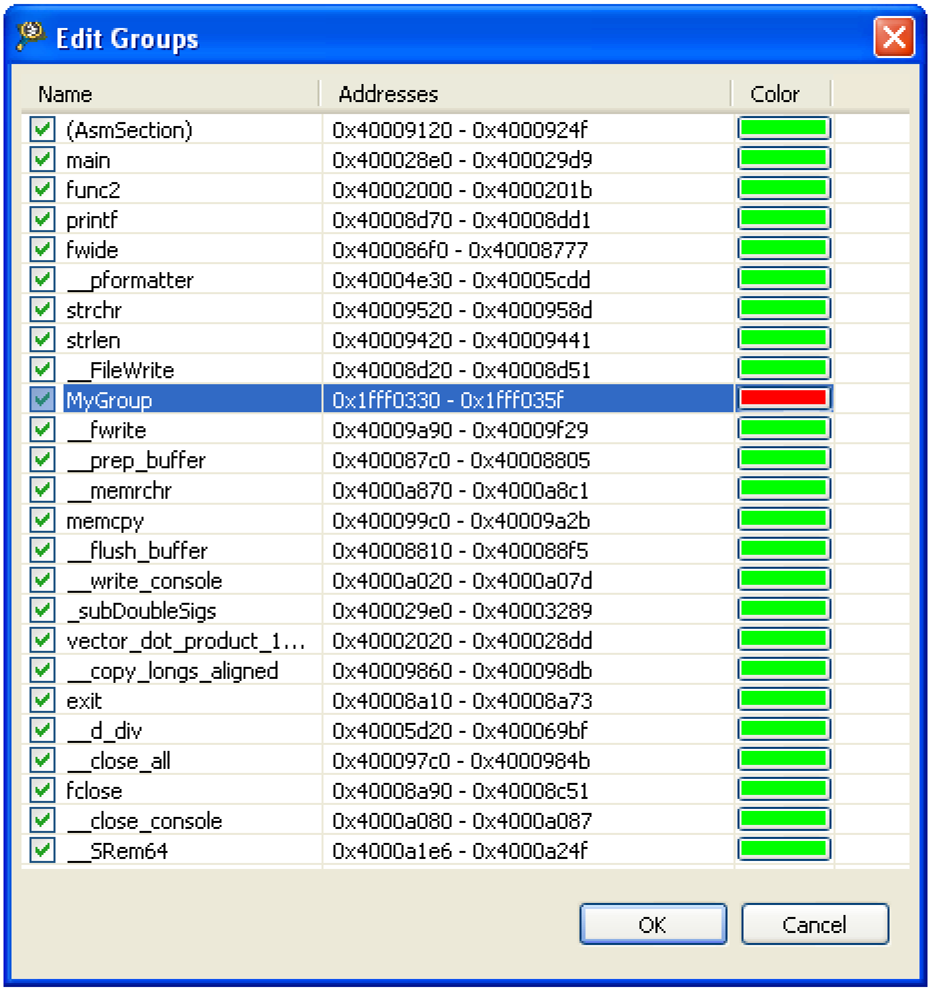
Name	Addresses	Color
☑ (AsmSection)	0x40009120 - 0x4000924f	🟩
☑ main	0x400028e0 - 0x400029d9	🟩
☑ func2	0x40002000 - 0x4000201b	🟩
☑ printf	0x40008d70 - 0x40008dd1	🟩
☑ fwide	0x400086f0 - 0x40008777	🟩
☑ __pformatter	0x40004e30 - 0x40005cdd	🟩
☑ strchr	0x40009520 - 0x4000958d	🟩
☑ strlen	0x40009420 - 0x40009441	🟩
☑ __FileWrite	0x40008d20 - 0x40008d51	🟩
☑ __fwrite	0x40009a90 - 0x40009f29	🟩
☑ __prep_buffer	0x400087c0 - 0x40008805	🟩
☑ __memrchr	0x4000a870 - 0x4000a8c1	🟩
☑ new	0x0 - 0xFFFFFFFF	🟩
☑ memcpy	0x400099c0 - 0x40009a2b	🟩
☑ __flush_buffer	0x40008810 - 0x400088f5	🟩
☑ __write_console	0x4000a020 - 0x4000a07d	🟩
☑ _subDoubleSigs	0x400029e0 - 0x40003289	🟩
☑ vector_dot_product_16x16	0x40002020 - 0x400028dd	🟩
☑ exit	0x40008a10 - 0x40008a73	🟩
☑ __d_div	0x40005d20 - 0x400069bf	🟩
☑ __close_all	0x400097c0 - 0x4000984b	🟩
☑ fclose	0x40008a90 - 0x40008c51	🟩
☑ __close_console	0x4000a080 - 0x4000a087	🟩
☑ __SRem64	0x4000a1e6 - 0x4000a24f	🟩

Figure 3-34. Adding Group Dialog Box

2. Double-click on the `new` group cell.

   The cell becomes editable.

3. Type a name for the group, for example, `MyGroup`.
4. Double-click on the cell of the corresponding **Addresses** column, and edit the address range according to requirements, for example, `0x1fff0330-0x1fff035f`.
5. Change the color of the group.



Name	Addresses	Color
☑ (AsmSection)	0x40009120 - 0x4000924f	🟩
☑ main	0x400028e0 - 0x400029d9	🟩
☑ func2	0x40002000 - 0x4000201b	🟩
☑ printf	0x40008d70 - 0x40008dd1	🟩
☑ fwide	0x400086f0 - 0x40008777	🟩
☑ __pformatter	0x40004e30 - 0x40005cdd	🟩
☑ strchr	0x40009520 - 0x4000958d	🟩
☑ strlen	0x40009420 - 0x40009441	🟩
☑ __FileWrite	0x40008d20 - 0x40008d51	🟩
☑ MyGroup	0x1fff0330 - 0x1fff035f	🟥
☑ __fwrite	0x40009a90 - 0x40009f29	🟩
☑ __prep_buffer	0x400087c0 - 0x40008805	🟩
☑ __memrchr	0x4000a870 - 0x4000a8c1	🟩
☑ memcpy	0x400099c0 - 0x40009a2b	🟩
☑ __flush_buffer	0x40008810 - 0x400088f5	🟩
☑ __write_console	0x4000a020 - 0x4000a07d	🟩
☑ _subDoubleSigs	0x400029e0 - 0x40003289	🟩
☑ vector_dot_product_1...	0x40002020 - 0x400028dd	🟩
☑ __copy_longs_aligned	0x40009860 - 0x400098db	🟩
☑ exit	0x40008a10 - 0x40008a73	🟩
☑ __d_div	0x40005d20 - 0x400069bf	🟩
☑ __close_all	0x400097c0 - 0x4000984b	🟩
☑ fclose	0x40008a90 - 0x40008c51	🟩
☑ __close_console	0x4000a080 - 0x4000a087	🟩
☑ __SRem64	0x4000a1e6 - 0x4000a24f	🟩

Figure 3-35. After Editing Address Range and Color of Group Screen

To delete a group, select it, right-click on the **Edit Groups** dialog box, and select the **Delete Selected** option from the context menu. You can also remove a group from the graph by clearing the corresponding checkbox in the **Name** column. Check it again to include it in the graph.

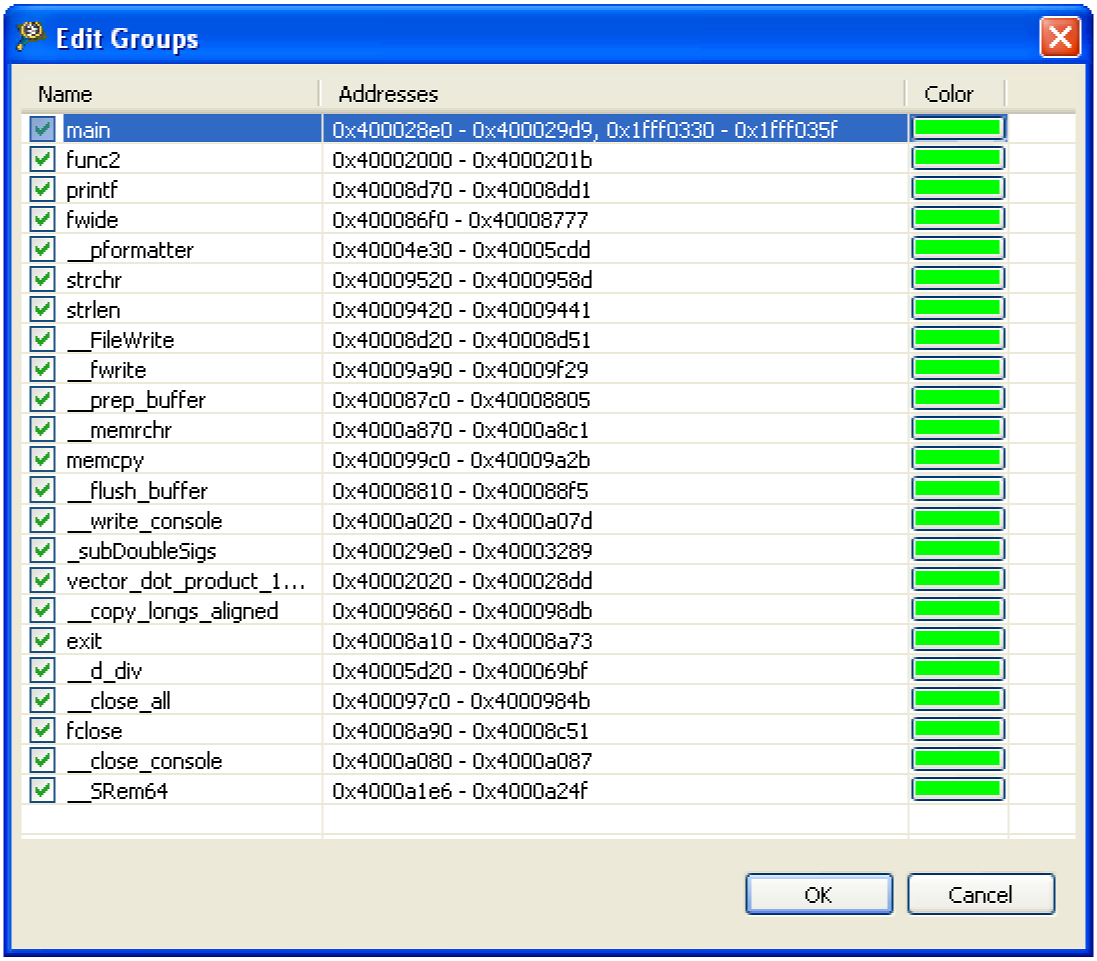
3.5.1.2.5 Merge Groups or Functions

This section explains how to merge the groups or functions available.

Merging is useful in case there are many functions and you do not want to view the trace of each and every function. You cannot undo this operation, that is you cannot separate the merged functions or groups. To view the original trace data, reopen the Trace Data viewer. Perform the following steps to merge the group or function:

1. In the **Edit Groups** dialog box, select the function or group to be merged.
2. Drag and drop it in the function or group with which you want it to get merged with.

3. Both the functions or groups merge into a single function or group that covers both address ranges, where the function, `main` is merged with the group, `MyGroup`.



Name	Addresses	Color
☑ main	0x400028e0 - 0x400029d9, 0x1fff0330 - 0x1fff035f	🟩
☑ func2	0x40002000 - 0x4000201b	🟩
☑ printf	0x40008d70 - 0x40008dd1	🟩
☑ fwide	0x400086f0 - 0x40008777	🟩
☑ __pformatter	0x40004e30 - 0x40005cdd	🟩
☑ strchr	0x40009520 - 0x4000958d	🟩
☑ strlen	0x40009420 - 0x40009441	🟩
☑ __FileWrite	0x40008d20 - 0x40008d51	🟩
☑ __fwrite	0x40009a90 - 0x40009f29	🟩
☑ __prep_buffer	0x400087c0 - 0x40008805	🟩
☑ __memrchr	0x4000a870 - 0x4000a8c1	🟩
☑ memcpy	0x400099c0 - 0x40009a2b	🟩
☑ __flush_buffer	0x40008810 - 0x400088f5	🟩
☑ __write_console	0x4000a020 - 0x4000a07d	🟩
☑ _subDoubleSigs	0x400029e0 - 0x40003289	🟩
☑ vector_dot_product_1...	0x40002020 - 0x400028dd	🟩
☑ __copy_longs_aligned	0x40009860 - 0x400098db	🟩
☑ exit	0x40008a10 - 0x40008a73	🟩
☑ __d_div	0x40005d20 - 0x400069bf	🟩
☑ __close_all	0x400097c0 - 0x4000984b	🟩
☑ fclose	0x40008a90 - 0x40008c51	🟩
☑ __close_console	0x4000a080 - 0x4000a087	🟩
☑ __SRem64	0x4000a1e6 - 0x4000a24f	🟩

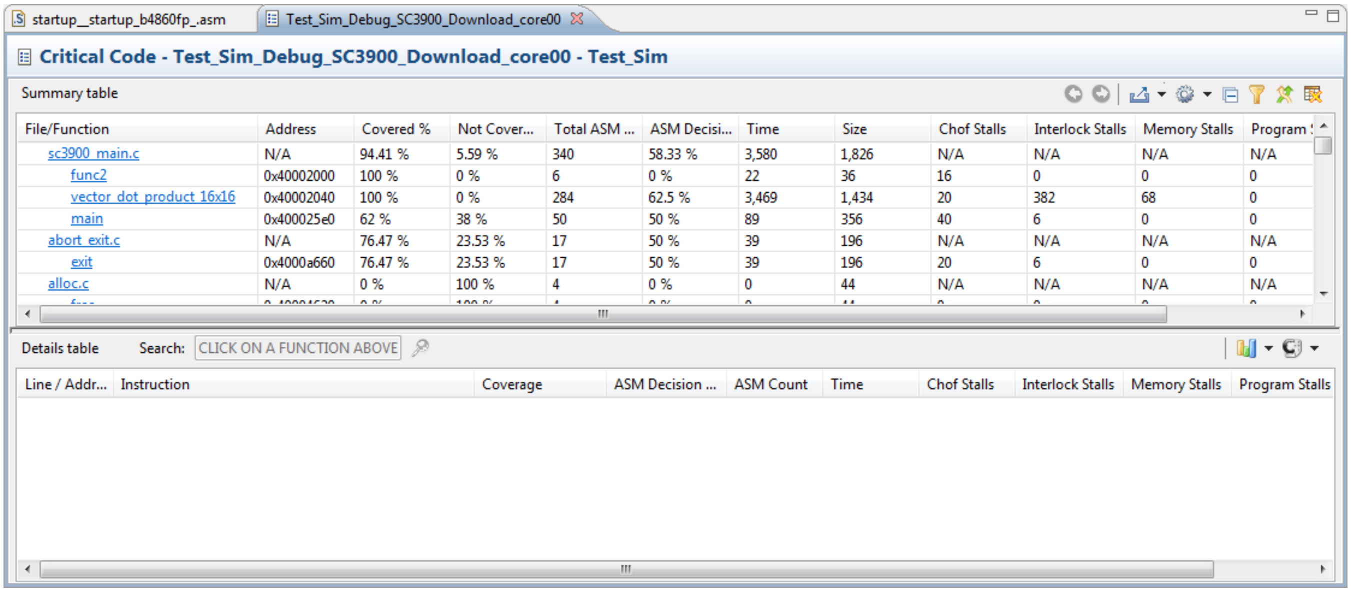
Figure 3-36. Merging Functions or Groups Screen

4. Click **OK**.

3.5.1.3 Viewing Critical Code Data

To view the critical code data, click the **Critical Code** link.

The following diagram shows an example of Critical Code data.



startup__startup_b4860fp_.asm | Test_Sim_Debug_SC3900_Download_core00

Critical Code - Test_Sim_Debug_SC3900_Download_core00 - Test_Sim

Summary table

File/Function	Address	Covered %	Not Cover...	Total ASM ...	ASM Decisi...	Time	Size	Chof Stalls	Interlock Stalls	Memory Stalls	Program S
sc3900_main.c	N/A	94.41 %	5.59 %	340	58.33 %	3,580	1,826	N/A	N/A	N/A	N/A
func2	0x40002000	100 %	0 %	6	0 %	22	36	16	0	0	0
vector_dot_product_16x16	0x40002040	100 %	0 %	284	62.5 %	3,469	1,434	20	382	68	0
main	0x400025e0	62 %	38 %	50	50 %	89	356	40	6	0	0
abort_exit.c	N/A	76.47 %	23.53 %	17	50 %	39	196	N/A	N/A	N/A	N/A
exit	0x4000a660	76.47 %	23.53 %	17	50 %	39	196	20	6	0	0
alloc.c	N/A	0 %	100 %	4	0 %	0	44	N/A	N/A	N/A	N/A

Details table Search: CLICK ON A FUNCTION ABOVE

Line / Addr...	Instruction	Coverage	ASM Decision ...	ASM Count	Time	Chof Stalls	Interlock Stalls	Memory Stalls	Program Stalls

Figure 3-37. Critical Code Viewer

The critical code data displays the summarized data of a function in a tabular form. The columns are movable; you can drag and drop the columns to move them according to your requirements.

The below table explains the various fields of Critical Code tab.

Table 3-8. Critical Code Data

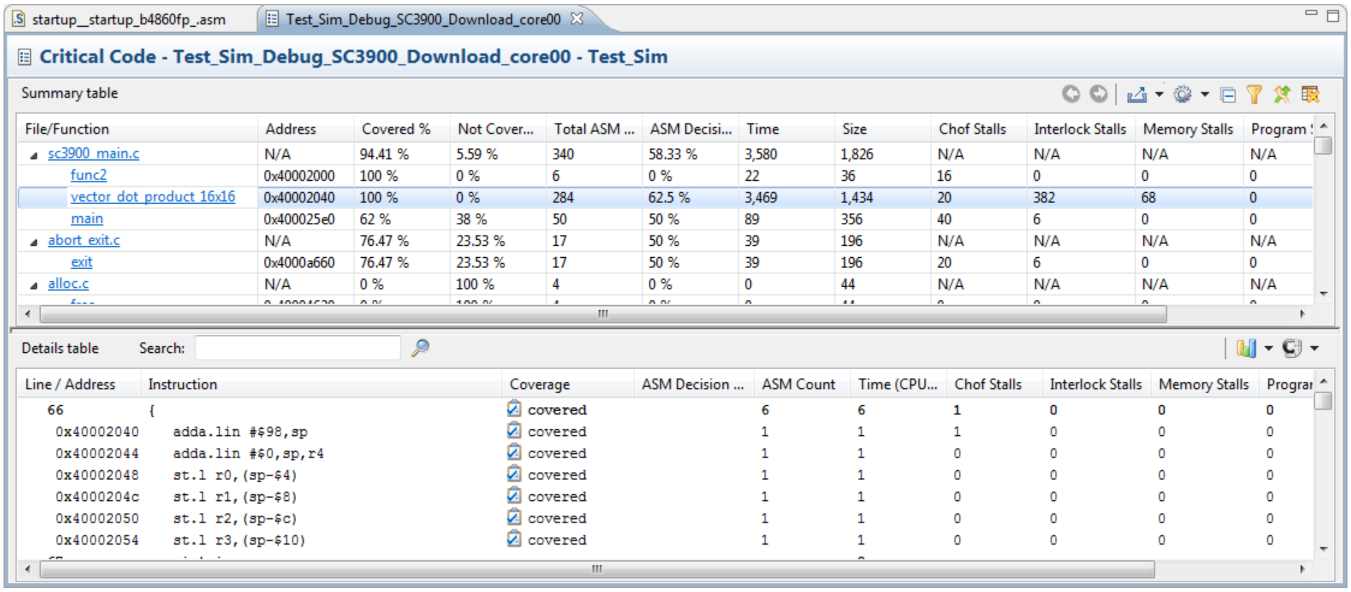
Field	Description
Address	Displays the start address of the function.
Function	Displays the name of the function that has executed.
Coverage %	Displays the percentage of number of assembly instructions executed from the total number of assembly instructions in a function.
ASM Decision Coverage %	Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. So if a function has two conditional instructions, one with 100% and another with 50% decision coverage, the decision coverage would be (100 + 50) / 2 = 75% . It is calculated only for assembly instructions and not for C source code.
ASM Count	Displays the number of lines executed in the function.
Time (Microsecond)	Displays the execution time of each function.
Size	Displays the number of bytes required by each function.
Chof Stalls	Displays the number of bubble cycles incurred by change of program flow for each function.
Interlock Stalls	Displays the number of bubble cycles incurred by core resource interlocks for each function.
Memory Stalls	Displays the number of bubble cycles incurred by data bus hold for each function.

Table continues on the next page...

Table 3-8. Critical Code Data (continued)

Field	Description
Program Stalls	Displays the number of bubble cycles incurred by program starvation for each function.
Rewind Stalls	Displays the number of bubble cycles incurred by a rewind on the core buses for each function.

The **Critical Code** tab displays data into two views; the top view displays the summary of the functions, and the bottom view displays the statistics for all the instructions executed in a particular function. Click on a hyperlinked function in the top view of the **Critical Code** viewer to view the corresponding statistics for the instructions executed in that function. For example, the statistics of the `main()` function is shown below.


startup__startup_b4860fp_.asm
Test_Sim_Debug_SC3900_Download_core00

Critical Code - Test_Sim_Debug_SC3900_Download_core00 - Test_Sim

Summary table

File/Function	Address	Covered %	Not Cover...	Total ASM ...	ASM Decisi...	Time	Size	Chof Stalls	Interlock Stalls	Memory Stalls	Program S
sc3900_main.c	N/A	94.41 %	5.59 %	340	58.33 %	3,580	1,826	N/A	N/A	N/A	N/A
func2	0x40002000	100 %	0 %	6	0 %	22	36	16	0	0	0
vector_dot_product_16x16	0x40002040	100 %	0 %	284	62.5 %	3,469	1,434	20	382	68	0
main	0x400025e0	62 %	38 %	50	50 %	89	356	40	6	0	0
abort_exit.c	N/A	76.47 %	23.53 %	17	50 %	39	196	N/A	N/A	N/A	N/A
exit	0x4000a660	76.47 %	23.53 %	17	50 %	39	196	20	6	0	0
alloc.c	N/A	0 %	100 %	4	0 %	0	44	N/A	N/A	N/A	N/A

Details table Search:

Line / Address	Instruction	Coverage	ASM Decision ...	ASM Count	Time (CPU...	Chof Stalls	Interlock Stalls	Memory Stalls	Prograr
66	{	☑ covered		6	6	1	0	0	0
0x40002040	adda.lin #$98,sp	☑ covered		1	1	1	0	0	0
0x40002044	adda.lin #$0,sp,r4	☑ covered		1	1	0	0	0	0
0x40002048	st.l r0,(sp-$4)	☑ covered		1	1	0	0	0	0
0x4000204c	st.l r1,(sp-$8)	☑ covered		1	1	0	0	0	0
0x40002050	st.l r2,(sp-$c)	☑ covered		1	1	0	0	0	0
0x40002054	st.l r3,(sp-$10)	☑ covered		1	1	0	0	0	0

Figure 3-38. Statistics of Critical Code Data

The below table describes the fields of the statistics of the critical code data.

Table 3-9. Description of Statistics of Critical Code Data

Field	Description
Line/Address	Displays either the line number for each instruction in the source code or the address for the assembly code.
Instruction	Displays all the instructions executed in the selected function.
Coverage %	For C source lines, displays the percentage of number of assembly instructions executed from the total number of assembly instructions corresponding to the source line. For assembly source lines, it shows if the instructions were executed or not.

Table continues on the next page...

Table 3-9. Description of Statistics of Critical Code Data (continued)

Field	Description
ASM Decision Coverage	Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. So if a function has two conditional instructions, one with 100% and another with 50% decision coverage, the decision coverage would be (100 + 50) / 2 = 75% . It is calculated only for assembly instructions and not for C source code.
ASM Count	Displays the number of times each instruction is executed.
Time (CPU Cycles)	Displays the total time taken by each instruction in the function.
Chof Stalls	Displays the number of bubble cycles incurred by change of program flow for each function.
Interlock Stalls	Displays the number of bubble cycles incurred by core resource interlocks for each instruction in the function.
Memory Stalls	Displays the number of bubble cycles incurred by data bus hold for each function.
Program Stalls	Displays the number of bubble cycles incurred by program starvation for each function.
Rewind Stalls	Displays the number of bubble cycles incurred by a rewind on the core buses for each function.

NOTE

In the Critical Code viewer, all functions in all files associated with the project are displayed irrespective of coverage percentage. However, the 0% coverage functions do not appear in the **Performance** and **Call Tree** viewers because these functions are not considered to be computed and are not a part of caller-called pair.

Click on the column header to sort the critical code data by that column. However, you can only sort the critical code data available on the top view. The icons available in the statistics view of the Critical Code tab allow you to perform the following actions:

- **Previous function** - Lets you navigate to the previous function you selected in the **File/Function** column.
- **Next function** - Lets you navigate to the next function in the **File/Function** column.
- **Export** - Lets you export the critical code data of both top and bottom views in a CSV or HTML file.
- **Configure Table** - Lets you show and hide column(s) of the critical code data. Click and select the **Configure the table above option** to show/hide columns of the top view or the **Configure the table below** option to show/hide columns of the bottom view. The **Drag and drop to order columns** dialog box appears in which you can check/uncheck the checkboxes corresponding to the available columns to show/hide

them in the **Critical Code** viewer. The option also allows you to set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds.

- **Collapse/Expand all files** - Lets you collapse/expand all files in the **Summary Table**.
- **Filter files** - Lets you filter the desired files to display in the **Summary Table**. Click this button and select the desired files from the **Choose files to be listed** dialog.
- **Switch to executable source line statistics/Switch to ASM instructions statistics** - Lets you display the executable source line statistics/ASM instructions statistics data format in the **Summary Table**.
- **Graphics** - Lets you display the colored bars for the columns in the bottom view of the critical code data. For example, click and select the **Assembly > ASM Count**, option to display colored bars in the ASM Count. The colors in these columns differentiate source code with the assembly code.
- **Show code** - Lets you display the assembly or mixed code in the statistics of the critical code data. Click and select the **Assembly** option to display the assembly code. Similarly, select the **Source** or the **Mixed** option to display the source or mixed code in the statistical details.

3.5.1.3.1 Export Critical Code Data

You can export critical code statistical data into CSV or HTML file format.

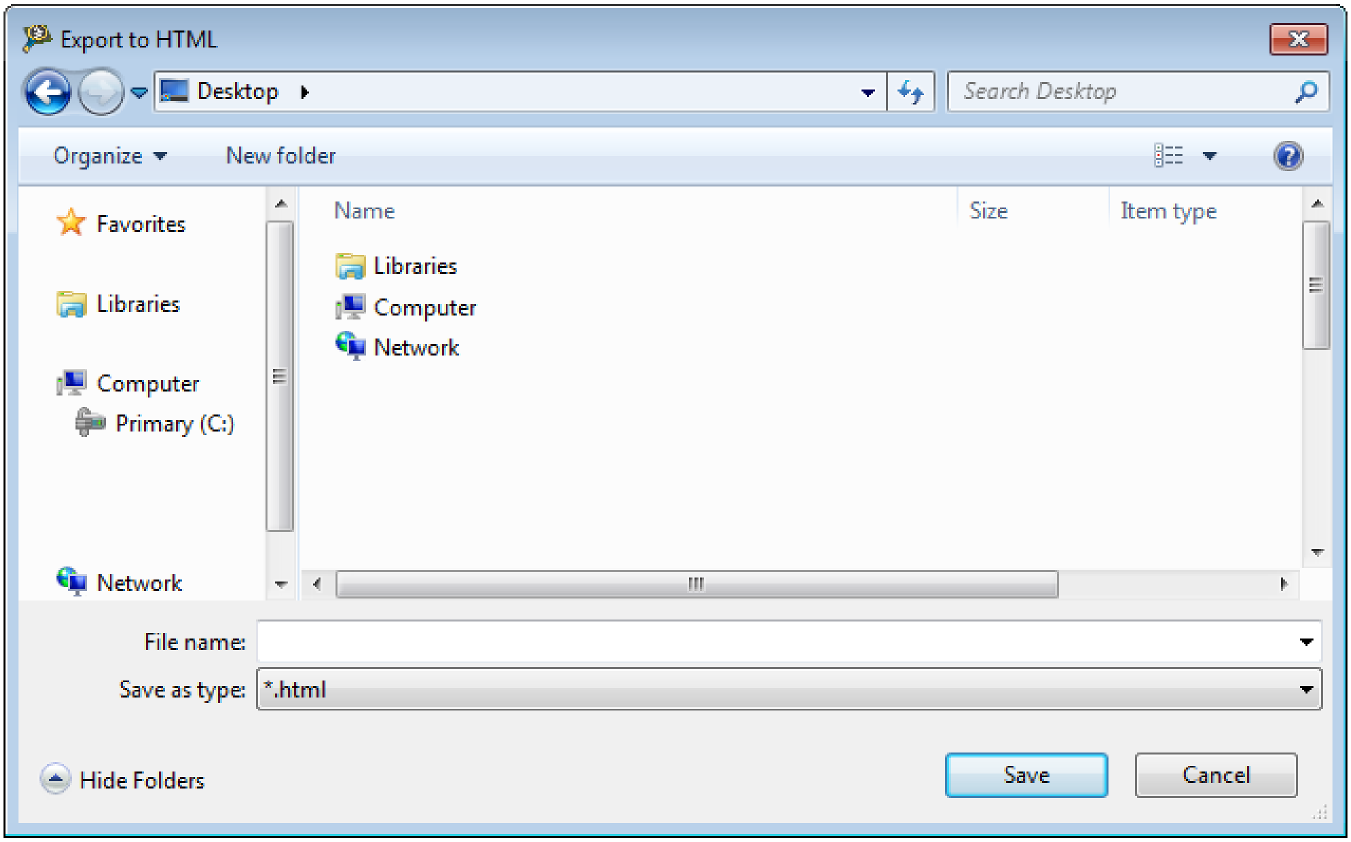
To export the data, follow the given procedure:

1. Click the export button available on the Critical Code view.

   The drop-down menu shows Export to CSV or Export to HTML option.

2. Select any one of the following option to export the statistical data in the respective format:
   - **Export to CSV**: Select the **Export summary table statistics** option to export the details of the top view or the **Export Details table statistics** option to export the details of the bottom view respectively.
   - **Export to HTML**: Select Export as source option to export the data in function details table that contains source code lines, Export as asm option to export the data in function details table that contains address and assembly code of the function, and Export as mixed option to export the data in function details table that contains both addresses and line numbers of the instructions with their corresponding source and assembly code.

   The **Export to** dialog box appears.



File name:

Save as type: *.html

Figure 3-39. Export To Dialog Box

3. Specify the **File name** and browse the location where you want to save the data.
4. Click **Save**.

The exported data will be saved in the respective format (`.csv` or `.html`).

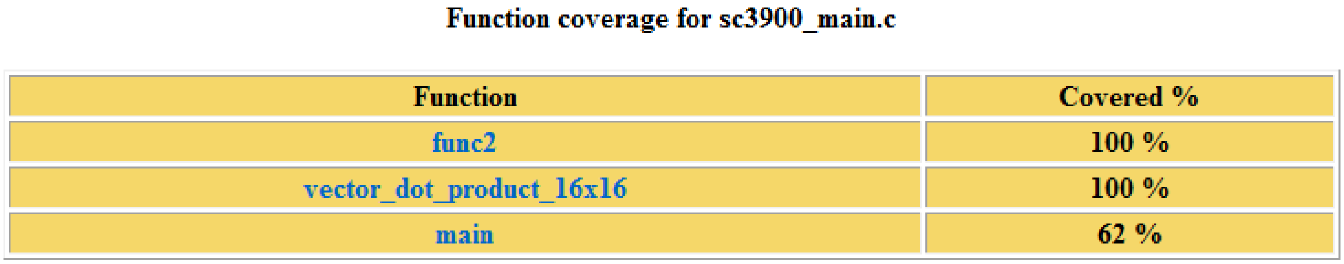
The data exported on the html page is structured in the given format:

- **Files**: Lists all the analyzed files
- **File Analysis Content**: Provides details on **Function Coverage** table, **Each function detail** (including Details table, Code coverage metrics table, and ASM decision metrics table), **Summary table**, and **ASM decision coverage table** of the selected file.

NOTE

While reading the data from the html page, the content of Function Details table suggests what option was selected from Export to HTML menu (Export as source, asm or mixed). For example, if the first column name of the Function Details table is Line, the option selected is Export as source. Similarly, if the first column name is Address, then the option selected is Export as asm, and if the first column name is Line/Address then the option selected is Export as mixed.

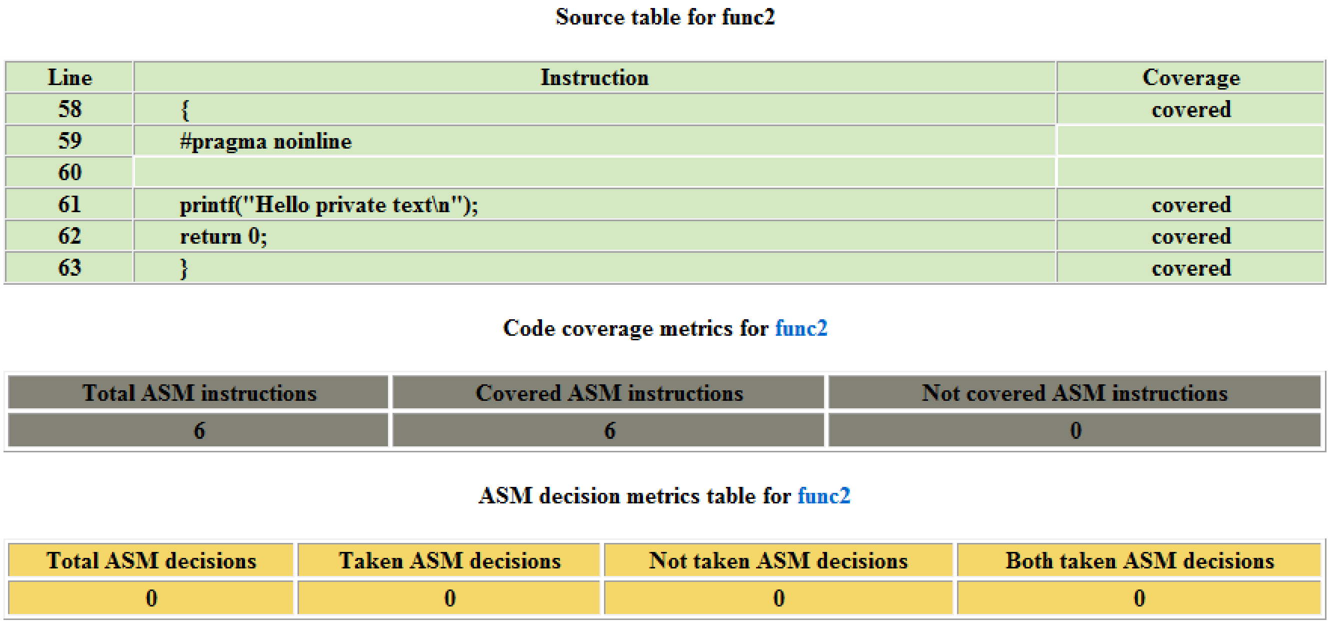
The screens given below shows the data structured in different tables:



Function coverage for sc3900_main.c

Function	Covered %
func2	100 %
vector_dot_product_16x16	100 %
main	62 %

Figure 3-40. Function Coverage Table



Source table for func2

Line	Instruction	Coverage
58	{	covered
59	#pragma noinline	
60		
61	printf("Hello private text\n");	covered
62	return 0;	covered
63	}	covered

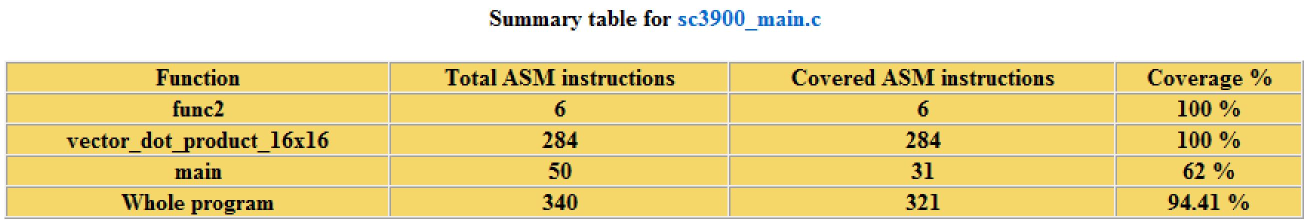
Code coverage metrics for func2

Total ASM instructions	Covered ASM instructions	Not covered ASM instructions
6	6	0

ASM decision metrics table for func2

Total ASM decisions	Taken ASM decisions	Not taken ASM decisions	Both taken ASM decisions
0	0	0	0

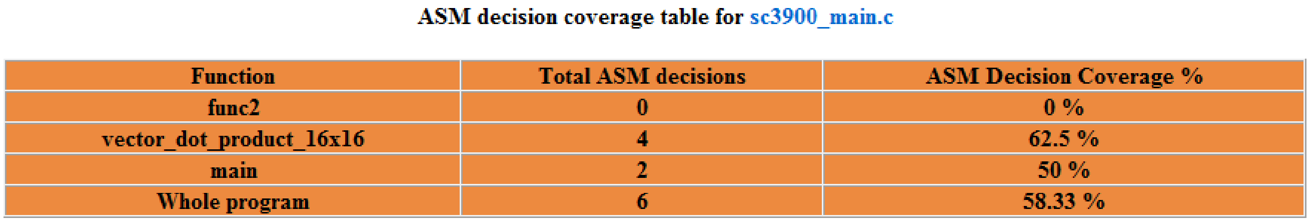
Figure 3-41. Function Details Table



Summary table for sc3900_main.c

Function	Total ASM instructions	Covered ASM instructions	Coverage %
func2	6	6	100 %
vector_dot_product_16x16	284	284	100 %
main	50	31	62 %
Whole program	340	321	94.41 %

Figure 3-42. Summary Table



ASM decision coverage table for sc3900_main.c

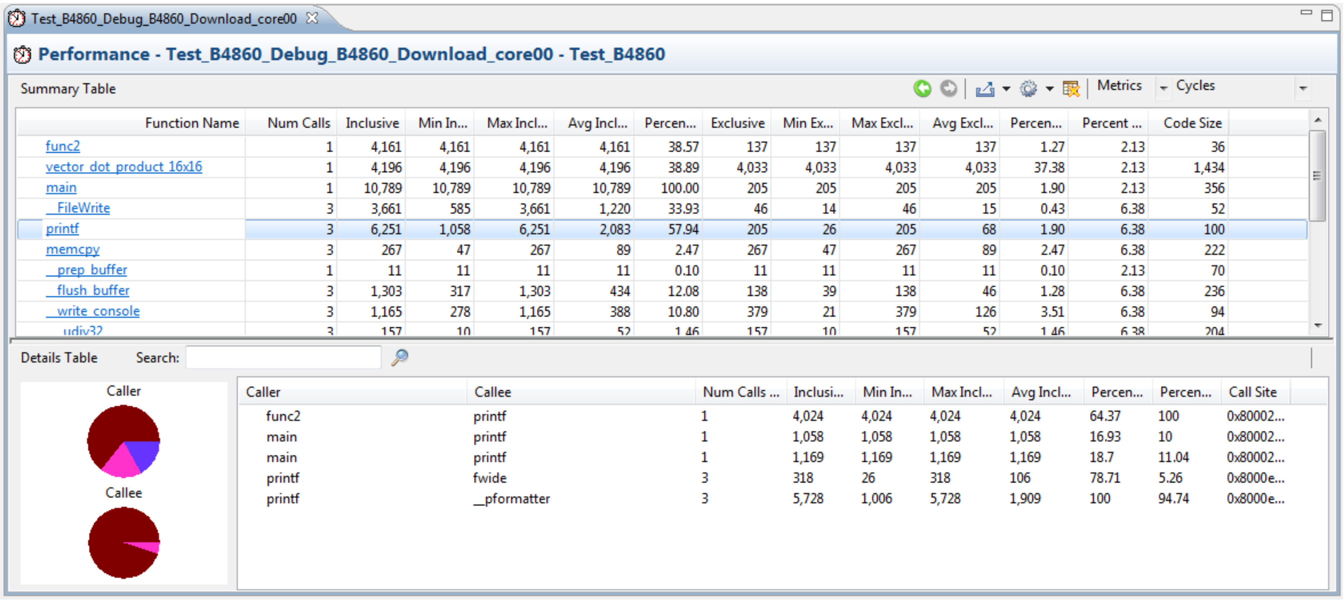
Function	Total ASM decisions	ASM Decision Coverage %
func2	0	0 %
vector_dot_product_16x16	4	62.5 %
main	2	50 %
Whole program	6	58.33 %

Figure 3-43. ASM Decision Coverage Table

3.5.1.4 Viewing Performance Data

To view performance data, click on the **Performance** hyperlink.

The Performance viewer appears as shown below.



Performance - Test_B4860_Debug_B4860_Download_core00 - Test_B4860

Summary Table

Function Name	Num Calls	Inclusive	Min In...	Max Incl...	Avg Incl...	Percen...	Exclusive	Min Ex...	Max Excl...	Avg Excl...	Percen...	Percent ...	Code Size
func2	1	4,161	4,161	4,161	4,161	38.57	137	137	137	137	1.27	2.13	36
vector_dot_product_16x16	1	4,196	4,196	4,196	4,196	38.89	4,033	4,033	4,033	4,033	37.38	2.13	1,434
main	1	10,789	10,789	10,789	10,789	100.00	205	205	205	205	1.90	2.13	356
__FileWrite	3	3,661	585	3,661	1,220	33.93	46	14	46	15	0.43	6.38	52
printf	3	6,251	1,058	6,251	2,083	57.94	205	26	205	68	1.90	6.38	100
memcpy	3	267	47	267	89	2.47	267	47	267	89	2.47	6.38	222
__prep_buffer	1	11	11	11	11	0.10	11	11	11	11	0.10	2.13	70
__flush_buffer	3	1,303	317	1,303	434	12.08	138	39	138	46	1.28	6.38	236
__write_console	3	1,165	278	1,165	388	10.80	379	21	379	126	3.51	6.38	94
udiv32	3	157	10	157	52	1.46	157	10	157	52	1.46	6.38	204

Details Table Search:

Caller	Callee	Num Calls ...	Inclusi...	Min In...	Max Incl...	Avg Incl...	Percen...	Percen...	Call Site
func2	printf	1	4,024	4,024	4,024	4,024	64.37	100	0x80002...
main	printf	1	1,058	1,058	1,058	1,058	16.93	10	0x80002...
main	printf	1	1,169	1,169	1,169	1,169	18.7	11.04	0x80002...
printf	fwide	3	318	26	318	106	78.71	5.26	0x8000e...
printf	__pformatter	3	5,728	1,006	5,728	1,909	100	94.74	0x8000e...

Figure 3-44. Performance Viewer

The Performance viewer is divided into two views:

- The top view presents function performance data in the **Summary** table. It displays the count and invocation information for each function that executes during the measurement, enabling you to compare the relative data for various portions of your target program. The information in the Summary table can be sorted by column in ascending or descending order. Click the column header to sort the corresponding data.

The following table explains the fields of the Summary table.

Table 3-10. Field description of Summary table

Field	Description
Function Name	Name of the function that has executed.
Num Calls	Number of times the function has executed.
Inclusive	Cumulative metric count during execution time spent from function entry to exit.
Min Inclusive	Minimum metric count during execution time spent from function entry to exit.
Max Inclusive	Maximum metric count during execution time spent from function entry to exit.
Avg Inclusive	Average metric count during execution time spent from function entry to exit.
Percent Inclusive	Percentage of total metric count spent from function entry to exit.
Exclusive	Cumulative metric count during execution time spent within function.
Min Exclusive	Minimum metric count during execution time spent within function.
Max Exclusive	Maximum metric count during execution time spent within function.
Avg Exclusive	Average metric count during execution time spent within function.
Percent Exclusive	Percentage of total metric count spent within function.
Percent Total Calls	Percentage of the calls to the function compared to the total calls.
Code Size	Number of bytes required by each function.

- The bottom view or the **Details** table presents call pair data for the function selected in the **Summary** table. It displays call pair relationships for the selected function, that is which function called the another function. Each function pair consists of a caller and a callee. The percent caller and percent callee data is also displayed graphically. The functions are represented in different colors in the pie chart, you can move the mouse cursor over the color to see the corresponding function.

The below table describes the fields of the **Details** table. You cannot sort the columns of this table.

Table 3-11. Field description of Details table

Field	Description
Caller	Name of the calling function.

Table continues on the next page...

Table 3-11. Field description of Details table (continued)

Field	Description
Callee	Name of the function that is called by the calling function.
Num Calls	Number of times the caller called the callee.
Inclusive	Cumulative metric count during execution time spent from function entry to exit.
Min Inclusive	Minimum metric count during execution time spent from function entry to exit.
Max Inclusive	Maximum metric count during execution time spent from function entry to exit.
Avg Inclusive	Average metric count during execution time spent from function entry to exit.
Percent Callee	Percent of total metric count during the time the selected function is the caller of a specific callee. The data is also shown in the Caller pie chart.
Percent Caller	Percent of total metric count during the time the selected function is the callee of a specific caller. The data is also shown in the Callee pie chart.
Call Site	Address from where the function was called.

The table below lists the buttons available in the Summary table of the Performance viewer.

Table 3-12. Buttons Available in Summary Table of Performance Viewer

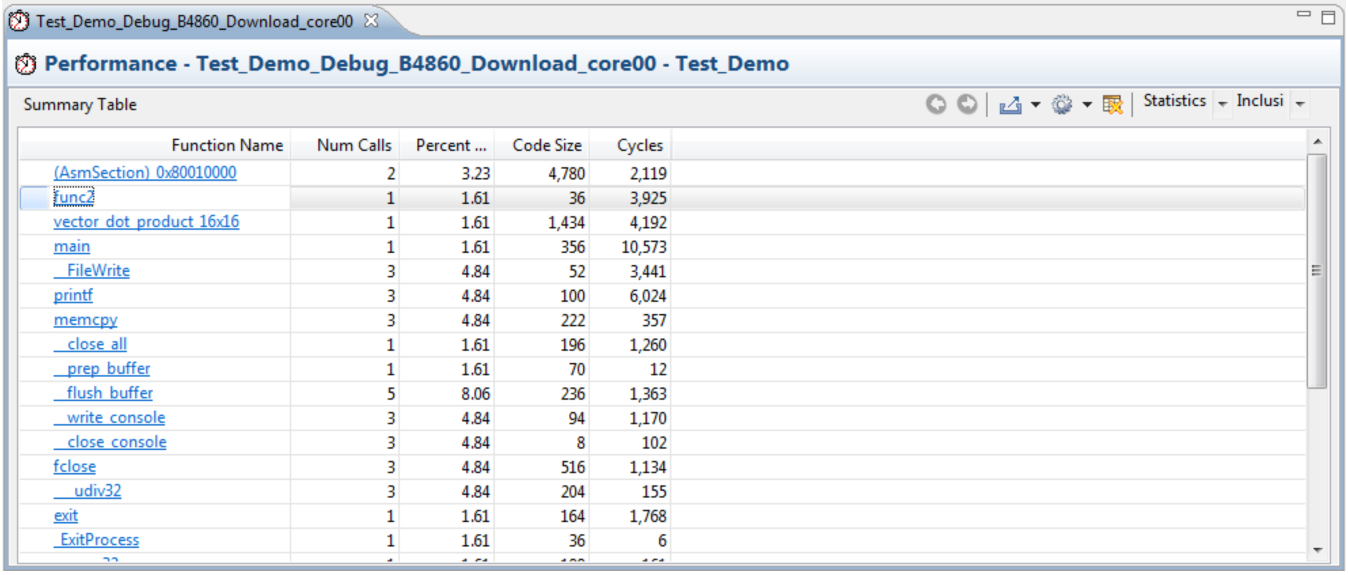
Name	Button	Description
Previous Function	⬅	Lets you view the details of the previous function that was selected in the bottom view before the currently selected function. Click it to view the details of the previous function.
Next Function	➡	Lets you view the details of the next function that was selected in the bottom view. **NOTE**: The Previous and Next buttons are contextual and go to previous/next function according to the history of selections. So if you select a single line in the view, these buttons will be disabled because there is no history.
Export	↗ ▾	Lets you export the performance data of both top and bottom views in a CSV file. Click the button and select the **Export the statistics above** option to export the details of the top view or the **Export the statistics below** option to export the details of the bottom view respectively.
Configure Table	⚙ ▾	Lets you show and hide column(s) of the performance data. Click the button and select the **Configure the table above** option to show/hide columns of the top view or the **Configure the table below** option to show/hide columns of the bottom view. The **Drag and drop to order columns** dialog box appears in which you can check/clear the checkboxes corresponding to the available columns to show/hide them in the **Performance** viewer. The Configuring Time Unit option allows you to set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds.

Table continues on the next page...

Table 3-12. Buttons Available in Summary Table of Performance Viewer (continued)

Name	Button	Description
Exclude Symbols		Lets you exclude statistics libraries or symbols from the performance data. Click this button to select/specify a function or library you want to exclude from the performance data. After excluding the selected function, library, or symbol, the statistics will be recomputed and reloaded in the Performance data viewer. For details, see Exclude Symbols from Statistics.
Choose view	Metrics ▾	Lets you to switch between the data available in the Metrics view and Statistics view of the summary table. Metrics view displays all available performance statistics details for a specific metric; for example, a Metrics view for Cycles metric displays all available performance statistics details for metric Cycles. Statistics view displays all performance metrics details for a specific statistic; for example, a Statistics view for Inclusive statistic displays all available performance metrics details for Inclusive statistic. **NOTE:** Performance details table is available only for metrics view. The data in Details table is refreshed when a user selects a new metric. NOTE: For Statistics view **Configure Columns for Details Table** and **Export Details table statistics** options are disabled.
Choose metric/statistic	Cycles ▾	Lets you load and list the metrics name (Cycles) or statistics name (Inclusive) in the summary table of the Performance data. The data displayed in the Summary table will correspond to the selection made in the **Choose view** and **Choose metric/statistic** fields.

By default, the data collected in the Performance data viewer is based on Cycle metric and is displayed in the Metrics view. The below figure shows the Statistics view of the Performance data viewer.


Test_Demo_Debug_B4860_Download_core00
Performance - Test_Demo_Debug_B4860_Download_core00 - Test_Demo
Summary Table
Statistics
Inclusi

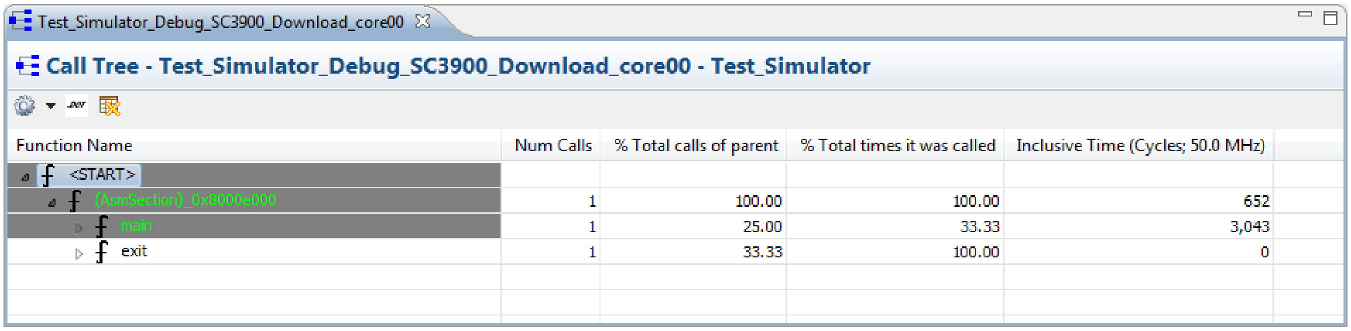
Function Name	Num Calls	Percent ...	Code Size	Cycles
(AsmSection)_0x80010000	2	3.23	4,780	2,119
func2	1	1.61	36	3,925
vector_dot_product_16x16	1	1.61	1,434	4,192
main	1	1.61	356	10,573
__FileWrite	3	4.84	52	3,441
printf	3	4.84	100	6,024
memcpy	3	4.84	222	357
__close_all	1	1.61	196	1,260
__prep_buffer	1	1.61	70	12
__flush_buffer	5	8.06	236	1,363
__write_console	3	4.84	94	1,170
__close_console	3	4.84	8	102
fclose	3	4.84	516	1,134
___udiv32	3	4.84	204	155
exit	1	1.61	164	1,768
_ExitProcess	1	1.61	36	6

Figure 3-45. Performance Viewer - Statistics View

3.5.1.5 Viewing Call Tree Data

To view the call tree data, click the **Call Tree** link.

The **Call Tree** viewer appears as shown below.



Test_Simulator_Debug_SC3900_Download_core00

Call Tree - Test_Simulator_Debug_SC3900_Download_core00 - Test_Simulator

Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles; 50.0 MHz)
<START>				
(AsmSection)_0x8000e000	1	100.00	100.00	652
main	1	25.00	33.33	3,043
exit	1	33.33	100.00	0

Figure 3-46. Call Tree Viewer

In the **Call Tree** viewer, **START** is the root of the tree. You can click on "+" to expand the tree and "-" to collapse the tree. It shows the biggest depth for stack utilization in **Call Tree** and the functions on this call path are displayed in green color. The Call Tree nodes are synchronized with the source code. You can double-click on the node to view the source code. The columns are movable; you can move the columns to the left or right of another column depending on your requirements by dragging and dropping.

The following table describe the fields of **Call Tree** data.

Table 3-13. Call Tree Table

Field	Description
Function Name	Name of function that has executed.
Num Calls	Number of times function has executed.
% Total calls of Parent	Percent of number of function calls from total number of calls in the application.
% Total times it was called	Percent of number of times a function was called.
Inclusive Time	Cumulative count during execution time spent from function entry to exit.

You can set **CPU frequency** and **Inclusive Time** displayed in the **Call Tree** viewer in cycles, milliseconds, microseconds, and nanoseconds. Click **Configure Table** to configure time unit and set CPU frequency. For details, refer to the Configuring Time Unit and Time Format topic.

The table below lists the toolbar buttons available in the **Call Tree** view.

Table 3-14. Toolbar Buttons Available in Call Tree View

Name	Button	Description
Configure Table	⚙▾	Lets you to set CPU frequency and set time in cycles, milliseconds, microseconds, and nanoseconds using the **Configuring Time Unit** option.
Export to dot	.DOT	Lets you export the call tree data to `.dot` file. Click this button to open the **Export to dot** dialog and save the data as `.dot` file in the desired location.
Exclude Symbols	☒	Lets you exclude statistics libraries or symbols from the performance data. Click this button to select/specify a function or library you want to exclude from the call tree data. After excluding the selected function, library, or symbol, the statistics will be recomputed and reloaded in the Call Tree data viewer. For details, see Exclude Symbols from Statistics.

3.5.1.6 Viewing Counterpoints

To view Counterpoints information, click on the **Counterpoints** hyperlink.

Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. For information on counterpoints, refer to the Viewing Counterpoints in Analysispoints topic.

3.5.2 Viewing Data Generated by Hardware

This section describes how to view trace data for a hardware project.

In the **Basic** subtab of **Trace and Profile** tab of **Debug** window, the following viewers are available in the **Software Analysis** view:

- If you have selected either **Program Trace** or **Coverage** scenarios, only **Trace** and **Critical Code** viewers are displayed.
- If you have selected any profiling trace scenario, **Trace**, **Performance**, **Timeline**, **Critical Code**, and **Call Tree** viewers are displayed.
- If you have selected **User defined events** or/and **Ownership** (used with **None** scenario), only **Trace** viewer is displayed.
- Log and Counterpoints viewers are displayed with any combination of trace scenarios.

Also, a note has been added in **Basic** tab of Trace and Profile that informs you about the kind of trace that will be collected with the chosen configuration. In case **Profiling-advanced** scenario is selected, you will have to customize the configuration in **Advanced** tab. In case **User defined events** option is selected, the events can be generated by writing TMDAT and TMTAG core registers.

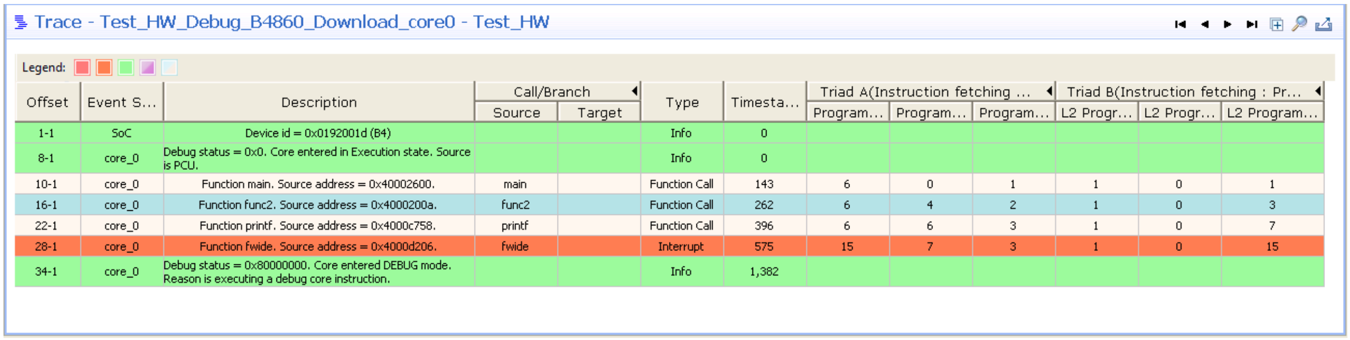
Each default trace scenario has a specific bandwidth. This indicates how many messages are routed in trace stream (on hardware), it depends on the trace scenario that you have selected to collect trace.

The following data can be viewed:

- Viewing Trace Data
- Viewing Timeline Data
- Viewing Critical Code Data
- Viewing Performance Data
- Viewing Counterpoints

3.5.2.1 Viewing Trace Data

Click the **Trace** link to view the trace data.



Trace - Test_HW_Debug_B4860_Download_core0 - Test_HW

Legend: 🟥 🟧 🟩 🟪 ⬜

Offset	Event S...	Description	Call/Branch		Type	Timesta...	Triad A(Instruction fetching ...			Triad B(Instruction fetching : Pr...		
			Source	Target			Program...	Program...	Program...	L2 Progr...	L2 Progr...	L2 Program...
1-1	SoC	Device id = 0x0192001d (B4)			Info	0						
8-1	core_0	Debug status = 0x0. Core entered in Execution state. Source is PCU.			Info	0						
10-1	core_0	Function main. Source address = 0x40002600.	main		Function Call	143	6	0	1	1	0	1
16-1	core_0	Function func2. Source address = 0x4000200a.	func2		Function Call	262	6	4	2	1	0	3
22-1	core_0	Function printf. Source address = 0x4000c758.	printf		Function Call	396	6	6	3	1	0	7
28-1	core_0	Function fwide. Source address = 0x4000d206.	fwide		Interrupt	575	15	7	3	1	0	15
34-1	core_0	Debug status = 0x80000000. Core entered DEBUG mode. Reason is executing a debug core instruction.			Info	1,382						

Figure 3-47. Trace Viewer

The below table explains the values of each counter for each trace scenario.

Table 3-15. Counter Values for each Trace Scenario

Trace Scenario	Traid A			Traid B		
	Counter 0	Counter 1	Counter 2	Counter 0	Counter 1	Counter 2
Profiling - L2 cache events	Total L2 demand accesses	L2 program accesses	L2 data accesses	Total L2 hit	L2 instruction miss	L2 data miss

Table continues on the next page...

Table 3-15. Counter Values for each Trace Scenario (continued)

Trace Scenario	Traid A			Traid B		
	Counter 0	Counter 1	Counter 2	Counter 0	Counter 1	Counter 2
Profiling - data loads	Data access L1 hits	Data access L1 pre-fetch hits	Data access L1 miss	L2 Data access hits	L2 Data access hits	L2 Data access miss
Profiling - clock cycles	Application cycles	Inactive counter	Bubbles	Inactive counter	Inactive counter	Inactive counter

The following table explains fields of trace data.

Table 3-16. Trace Data - Description of Fields

Field	Description
Offset	First number represents trace message offset in raw trace file. Second number represents trace event index. One trace message can generate one or more trace events.
Event Source	Displays source of trace event
Description	Displays description of trace event
Call/Branch	
Source	Displays source function of trace event
Target	Displays target function of trace event
Type	Displays type of trace event
Timestamp (Cycles)	Displays the absolute clock cycles that the instruction takes to execute
Traid A	
Total L2 demand accesses	Displays number of events counted by counter 0 of triad A
L2 program accesses	Displays number of events counted by counter 1 of triad A
L2 data accesses	Displays number of events counted by counter 2 of triad A
Triad B	
Total L2 hit	Displays number of events counted by counter 0 of triad B
L2 instruction miss	Displays number of events counted by counter 1 of triad B.
L2 data miss	Displays number of events counted by counter 0 of triad B.

For more information on trace data, refer to the Viewing Trace Data topic.

3.5.2.2 Viewing Timeline Data

For information on Timeline data, refer to the Viewing Timeline Data topic.

3.5.2.3 Viewing Critical Code Data

For information on Critical Code data, refer to the Viewing Critical Code Data topic.

3.5.2.4 Viewing Performance Data

For information on Performance data, refer to the Viewing Performance Data topic.

3.5.2.5 Viewing Counterpoints

For information on counterpoints, refer to the Viewing Counterpoints in Analysispoints topic.

3.6 Exclude Symbols from Statistics

Exclude symbols feature allows you to configure the statistics before or after the data is processed.

The configure results dialog is available from:

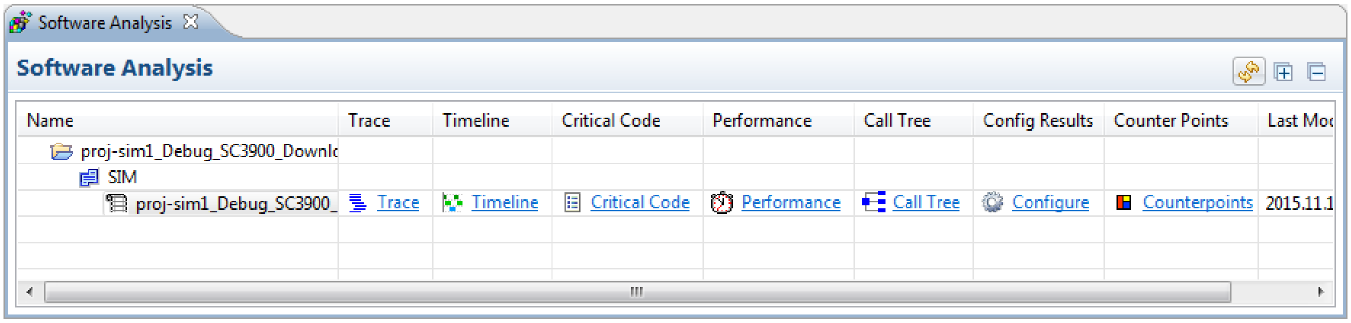
- Software Analysis View
- Critical Code View
- Performance View
- Call Tree View

3.6.1 Software Analysis View

In Software Analysis view, the column **Config Results** provides a hyperlink to the result configuration view.

The configuration is saved in a `.sym` file inside the **Analysis Data** folder available at the project folder. Perform the following steps to configure the statistics before computing the results:

1. From the menu bar, select **Window > Show View > Other > Software Analysis > Software Analysis** to open the **Software Analysis** view.
2. After collecting the data, the data file will be listed in the **Software Analysis** view. The exclude symbols option will be available for each set of results.



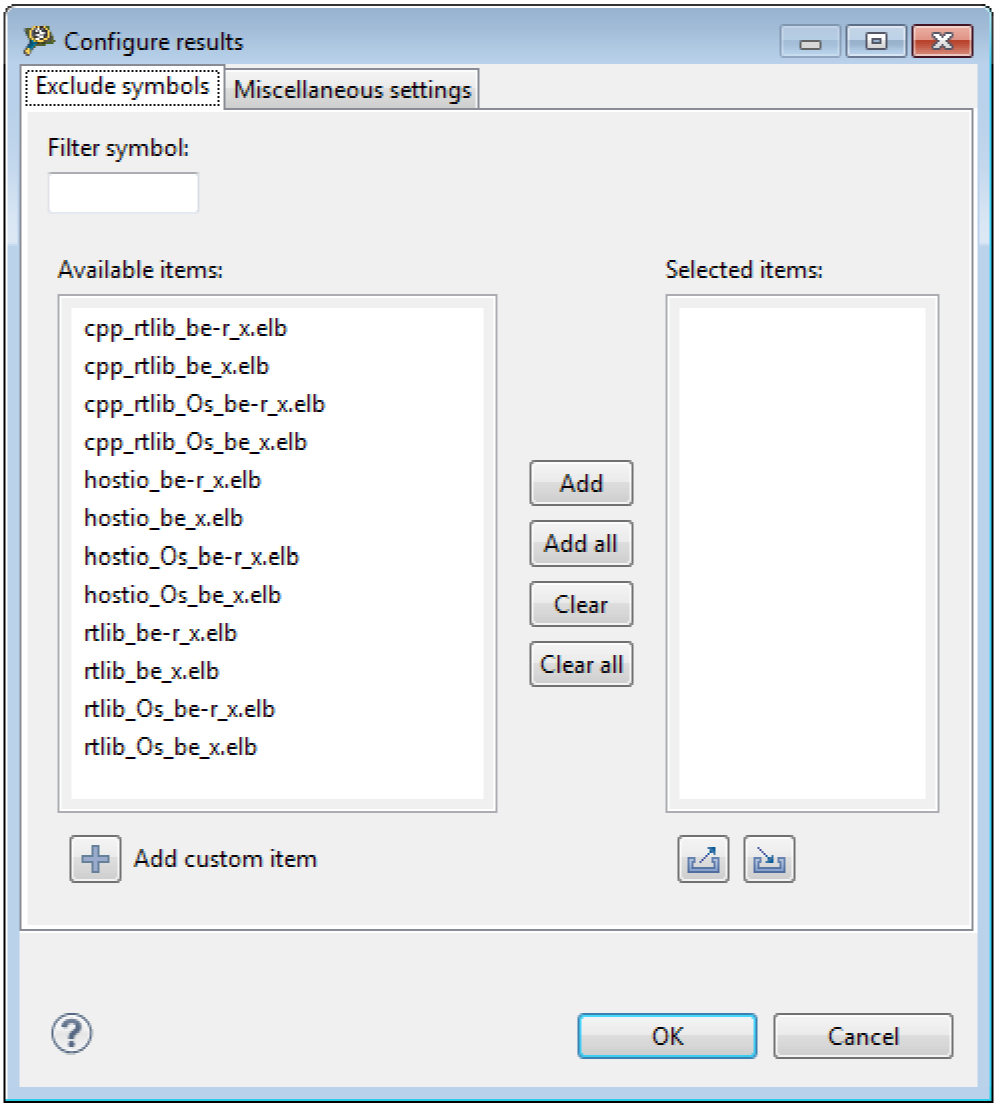
Software Analysis

Name	Trace	Timeline	Critical Code	Performance	Call Tree	Config Results	Counter Points	Last Moc
proj-sim1_Debug_SC3900_Downlc								
SIM								
proj-sim1_Debug_SC3900_	Trace	Timeline	Critical Code	Performance	Call Tree	Configure	Counterpoints	2015.11.1

Figure 3-48. Software Analysis View

3. Click the **Configure** hyperlink available under the **Config Results** column.

   The **Exclude symbols** dialog appears.



Configure results

Exclude symbols | Miscellaneous settings

Filter symbol:

Available items:

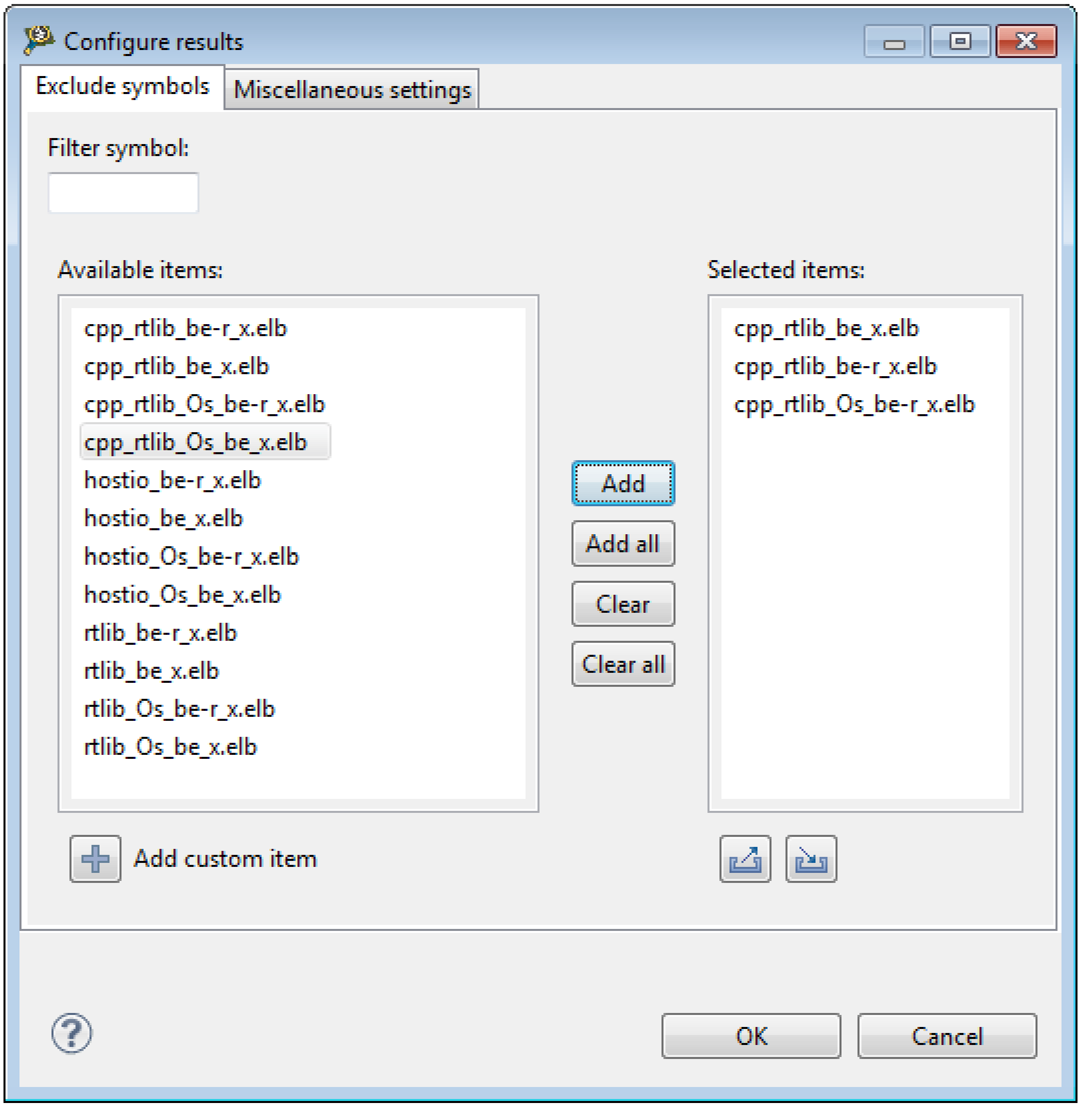
cpp_rtlib_be-r_x.elb
cpp_rtlib_be_x.elb
cpp_rtlib_Os_be-r_x.elb
cpp_rtlib_Os_be_x.elb
hostio_be-r_x.elb
hostio_be_x.elb
hostio_Os_be-r_x.elb
hostio_Os_be_x.elb
rtlib_be-r_x.elb
rtlib_be_x.elb
rtlib_Os_be-r_x.elb
rtlib_Os_be_x.elb

Selected items:

Figure 3-49. Exclude Symbols Dialog Box

4. Select the libraries from the **Available items** list.
5. Click **Add**.

   The selected libraries now appear under the **Selected items** list.



Available items:

cpp_rtlib_be-r_x.elb
cpp_rtlib_be_x.elb
cpp_rtlib_Os_be-r_x.elb
cpp_rtlib_Os_be_x.elb
hostio_be-r_x.elb
hostio_be_x.elb
hostio_Os_be-r_x.elb
hostio_Os_be_x.elb
rtlib_be-r_x.elb
rtlib_be_x.elb
rtlib_Os_be-r_x.elb
rtlib_Os_be_x.elb

Selected items:

cpp_rtlib_be_x.elb
cpp_rtlib_be-r_x.elb
cpp_rtlib_Os_be-r_x.elb

Figure 3-50. Exclude Symbols Dialog Box

6. Click **OK**.
   The configuration is saved with the list of all symbols and libraries that needs to be excluded from statistics.

NOTE

If the result configurator is opened for a new project, the latest saved configuration will be loaded into tables as a default configuration. If the configurator is not used, the results will be computed with all libraries and symbols.

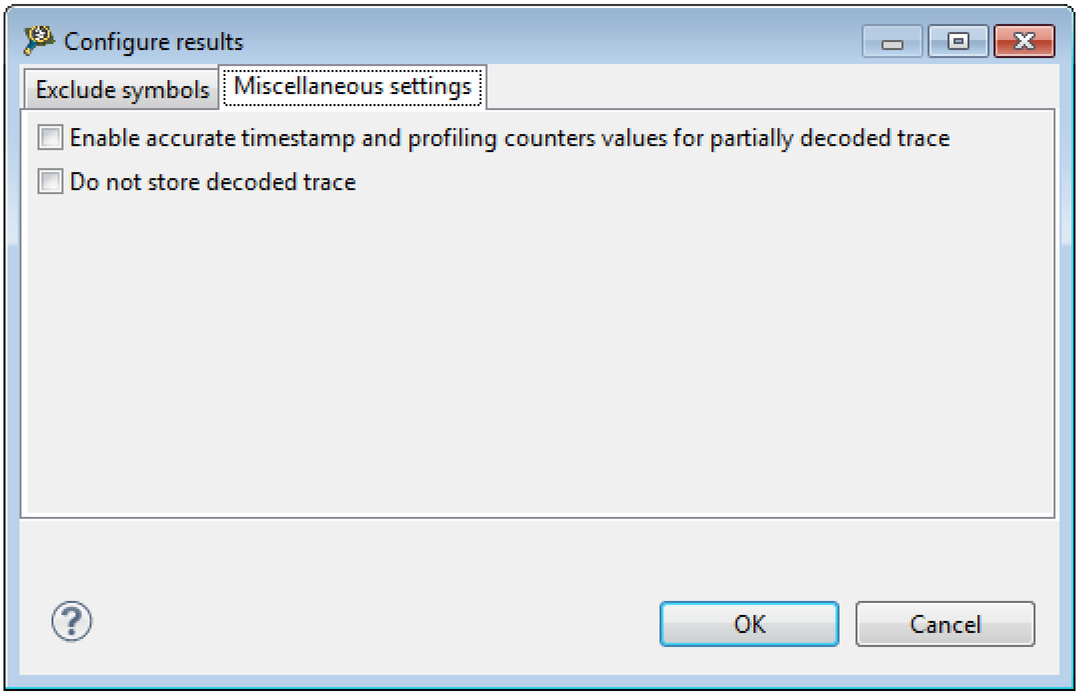
You can also perform the following actions using **Exclude symbols** dialog:

Table 3-17. Exclude Symbols Options

Options	Description
Filter symbol	Acts as a real time filter that allows to search a symbol in a library from **Available items** list. The **Available items** list shows all the libraries that contain symbols with the same name available in the **Filter symbol** field.
Add custom item	Allows to add a customized symbol or an entire library under the **Available items** list.
Export item list to a file	Exports the configuration in a file. The file is saved in the **Analysis Results** folder of the current project location.
Import item list from file	Loads an existing configuration.

When scrolling through unprocessed data in the **Trace** viewer, the timestamp values and profiling counters values increases from the beginning of the trace, and at some point of time, the value gets reset. Also, the timestamp value does not increase beyond 16 million (approximately) and the profiling counters values beyond 256.000 (approximately).You can use the **Configure results** dialog to control the overflow of counters in the **Trace** viewer. Click the **Miscellaneous settings** tab, and select the **Enable accurate timestamp and profiling counters values for partially decoded trace** checkbox to enable the feature for handling timestamp overflow. When this checkbox is selected, the timestamps value count indefinitely, that is the values will not drop or reset while navigating through unprocessed data in the **Trace** viewer. If not selected, the behavior remains unchanged and the counters (timestamp and profiling) value increases till their maximum values and reset at some point.

Configure results has been enriched with the possibility to not store decoded trace while processing trace data for profiling viewers: Timeline, Critical Code, Performance, Call Tree and Counterpoints. By default processing trace data for these viewers is done with storing decoded trace. Processing trace data without storing decoded trace for profiling viewers is quicker than processing trace data with storing decoded trace, impact of storing or not decoded trace can be noticed for higher size of raw trace and executable. Select **Do not store decoded trace** option to disable storing of decoded trace.



☐ Enable accurate timestamp and profiling counters values for partially decoded trace

☐ Do not store decoded trace

Figure 3-51. Configure results dialog - Miscellaneous settings tab

3.6.2 Critical Code View

In Critical Code view, use the **Exclude symbols** icon to view the Exclude symbols dialog box.

Perform the following steps to view the configured statistics in Critical Code view:

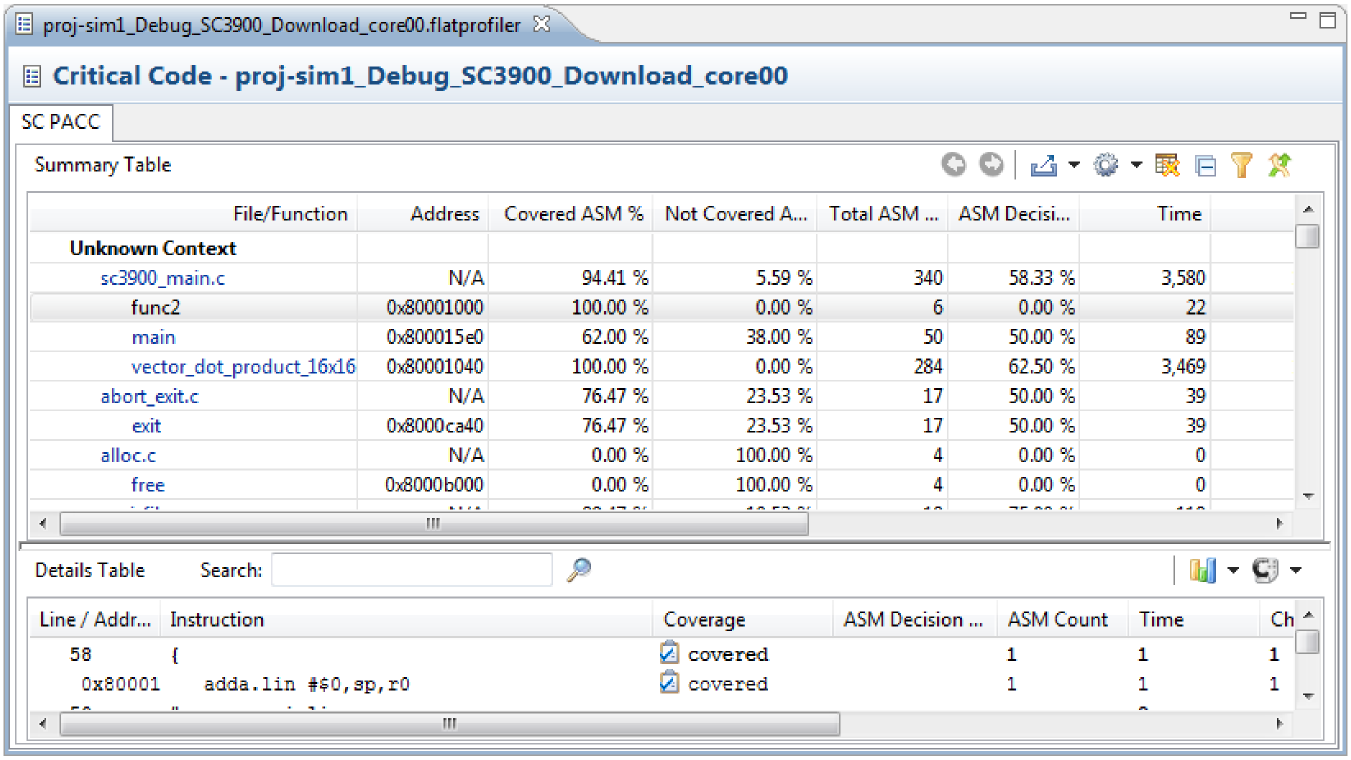
1. Click the **Exclude symbols** icon available on the toolbar.

   The **Exclude symbols** dialog appears.

2. Select the existing configuration or modify the configuration by adding or deleting the libraries or symbols from the **Selected items** list.
3. Click **OK**.

   The configuration is applied on the Critical Code data. The old statistics will be deleted and the new statistics will be computed based on the configuration, and viewer will be repopulated with the new results.

Once the data is computed as per the statistics, the respective view will display the filtered data. The following figure shows an example that the func2 symbol is excluded from the computed data:



proj-sim1_Debug_SC3900_Download_core00.flatprofiler

Critical Code - proj-sim1_Debug_SC3900_Download_core00

SC PACC

Summary Table

File/Function	Address	Covered ASM %	Not Covered A...	Total ASM ...	ASM Decisi...	Time
Unknown Context						
sc3900_main.c	N/A	94.41 %	5.59 %	340	58.33 %	3,580
func2	0x80001000	100.00 %	0.00 %	6	0.00 %	22
main	0x800015e0	62.00 %	38.00 %	50	50.00 %	89
vector_dot_product_16x16	0x80001040	100.00 %	0.00 %	284	62.50 %	3,469
abort_exit.c	N/A	76.47 %	23.53 %	17	50.00 %	39
exit	0x8000ca40	76.47 %	23.53 %	17	50.00 %	39
alloc.c	N/A	0.00 %	100.00 %	4	0.00 %	0
free	0x8000b000	0.00 %	100.00 %	4	0.00 %	0

Details Table Search:

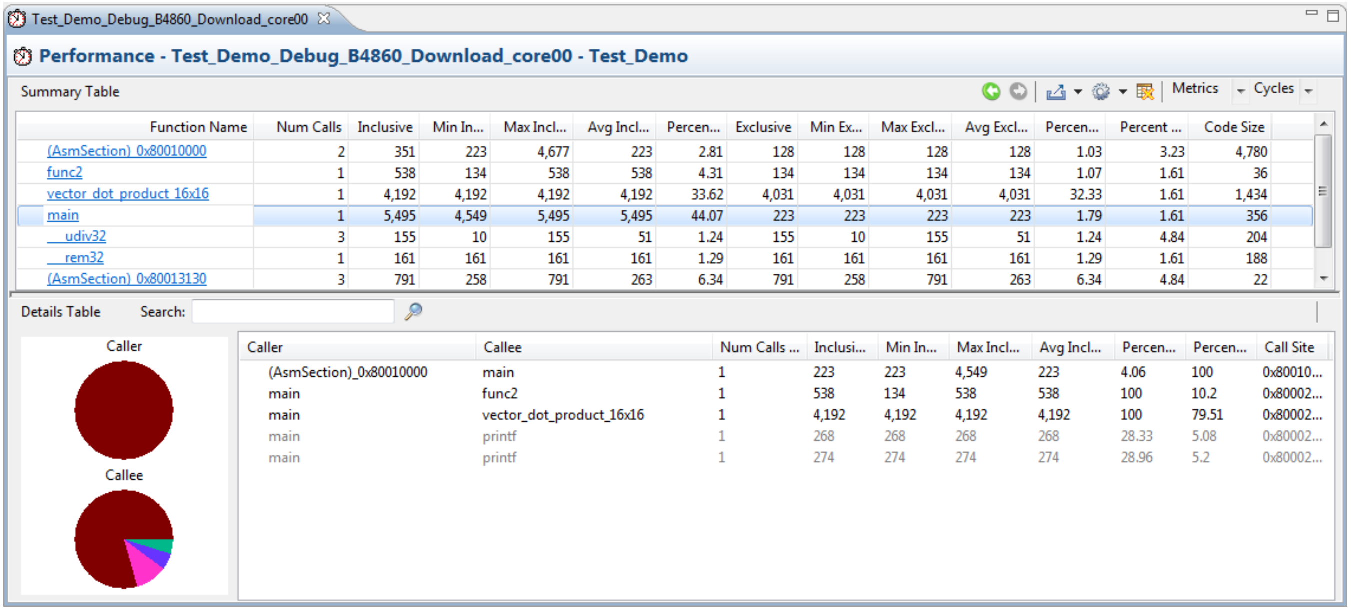
Line / Addr...	Instruction	Coverage	ASM Decision ...	ASM Count	Time	Ch
58	{	☑ covered		1	1	1
0x80001	adda.lin #$0,sp,r0	☑ covered		1	1	1

Figure 3-52. Critical Code View

3.6.3 Performance View

In Performance view, click the **Exclude symbols** icon available on the toolbar to filter out the symbols from the summary view table.

If the filtered symbol is called from another non filtered functions they will appear grayed in details table, in caller-callee pairs.



Performance - Test_Demo_Debug_B4860_Download_core00 - Test_Demo

Summary Table

Function Name	Num Calls	Inclusive	Min In...	Max Incl...	Avg Incl...	Percen...	Exclusive	Min Ex...	Max Excl...	Avg Excl...	Percen...	Percent ...	Code Size
(AsmSection)_0x80010000	2	351	223	4,677	223	2.81	128	128	128	128	1.03	3.23	4,780
func2	1	538	134	538	538	4.31	134	134	134	134	1.07	1.61	36
vector_dot_product_16x16	1	4,192	4,192	4,192	4,192	33.62	4,031	4,031	4,031	4,031	32.33	1.61	1,434
main	1	5,495	4,549	5,495	5,495	44.07	223	223	223	223	1.79	1.61	356
__udiv32	3	155	10	155	51	1.24	155	10	155	51	1.24	4.84	204
__rem32	1	161	161	161	161	1.29	161	161	161	161	1.29	1.61	188
(AsmSection)_0x80013130	3	791	258	791	263	6.34	791	258	791	263	6.34	4.84	22

Details Table Search:

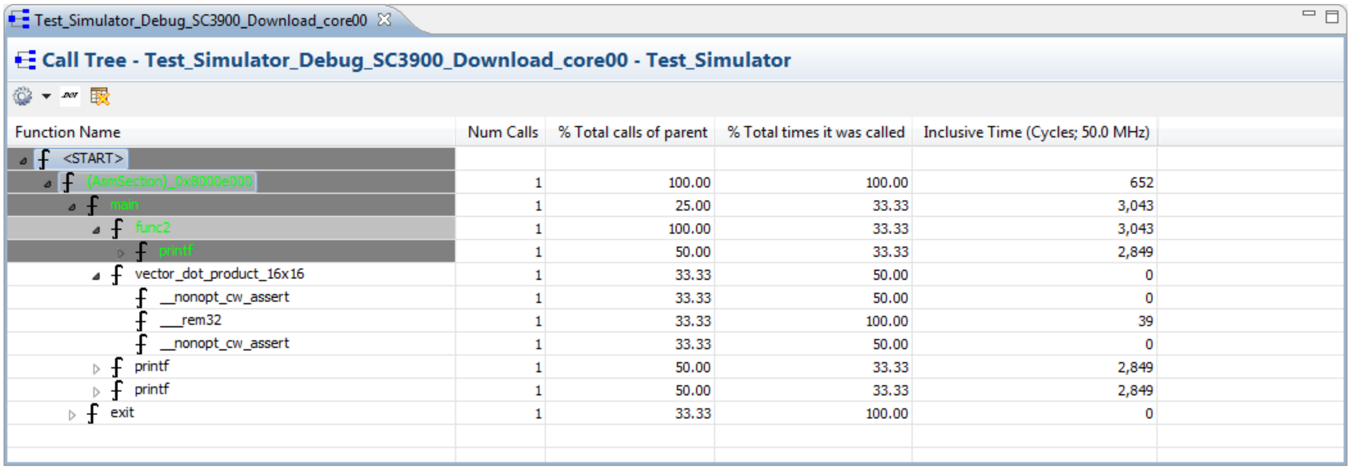
Caller	Callee	Num Calls ...	Inclusi...	Min In...	Max Incl...	Avg Incl...	Percen...	Percen...	Call Site
(AsmSection)_0x80010000	main	1	223	223	4,549	223	4.06	100	0x80010...
main	func2	1	538	134	538	538	100	10.2	0x80002...
main	vector_dot_product_16x16	1	4,192	4,192	4,192	4,192	100	79.51	0x80002...
main	printf	1	268	268	268	268	28.33	5.08	0x80002...
main	printf	1	274	274	274	274	28.96	5.2	0x80002...

Figure 3-53. Performance View

3.6.4 Call Tree View

In the Call Tree view, click the **Exclude symbols** icon to filter out the symbols or libraries.

Filtered symbols would appear as grayed in the tree hierarchy.



Call Tree - Test_Simulator_Debug_SC3900_Download_core00 - Test_Simulator

Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles; 50.0 MHz)
<START>				
(AsmSection)_0x8000e000	1	100.00	100.00	652
main	1	25.00	33.33	3,043
func2	1	100.00	33.33	3,043
printf	1	50.00	33.33	2,849
vector_dot_product_16x16	1	33.33	50.00	0
__nonopt_cw_assert	1	33.33	50.00	0
___rem32	1	33.33	100.00	39
__nonopt_cw_assert	1	33.33	50.00	0
printf	1	50.00	33.33	2,849
printf	1	50.00	33.33	2,849
exit	1	33.33	100.00	0

Figure 3-54. Call Tree View

NOTE

Applying the configuration in a result viewer (critical code, performance or call tree) will reload the new recomputed results in all opened corresponding viewers.

3.7 Trace on Attach

Use trace on attach feature to help within a debug session of type Attach.

Following are the two main cases where this feature is useful:

- On Application Crash Investigation
- On Changing Trace Configuration

3.7.1 On Application Crash Investigation

Trace on attach launch configuration feature is a very handful way of investigating application crashes as it provides the retrieving of trace buffer that contains information on the context of the crash.

To perform this; enable the trace (program trace or profiling trace) and configure the trace buffer in overwrite mode in the moment of crash. For more information on configuring trace buffer, refer Hardware Profiling.

NOTE

Trace buffer can be configured in three different modes:

Overwrite: It provides trace just before the application crashes. It is considered as the most recommended mode of configuring trace buffer while investigating an application crash.

One buffer: It provides trace from the first time filled trace buffer and usually trace buffer gets filled before crash. It is recommended to use one buffer mode only if size of trace until the application reaches crash is small enough to fit in trace buffer or if trace buffer size is increased so that trace until application reaches crash fits into it.

Continuous: It is not recommended to configure trace buffer in continuous mode because in this mode every time trace buffer gets filled; core enters into a freeze state and waits for the trace buffer to be filled out. So if trace buffer is configured in continuous mode application will not reach crash section as core will be freeze waiting for trace buffer to be filled out.

Perform the following steps to investigate application crash using trace on attach feature:

- Create an Attach Launch Configuration
- Upload Trace
- Investigate Crash

3.7.1.1 Create an Attach Launch Configuration

Follow the steps given below to create an attach launch configuration:

1. In the CodeWarrior Projects view, right-click on the project that was crashed and select **Debug as >Debug Configurations** option from the context menu.
2. In the Debug window, select **CodeWarrior** configuration and select **New** from the context menu.
3. On the **Main** page, select **Attach** as the **Debug session type**.
4. Specify the project name that crashed in the **Project** field. If the application is not displayed in the C/C++ Application field, click **Search Project** to select the application image.
5. Click **Debug**.

NOTE

By default, the **Trace control settings** will be set on **Manually (Using trace buttons from Trace Commander)** option while using the **Attach** mode configuration. Hence, the trace collection will not start automatically. To enable the trace collection, use the trace control button in the **Trace Commander** view.

3.7.1.2 Upload Trace

Click on the **Upload Trace** button available in the Trace Commander view to retrieve the trace buffer from target hardware to host machine.

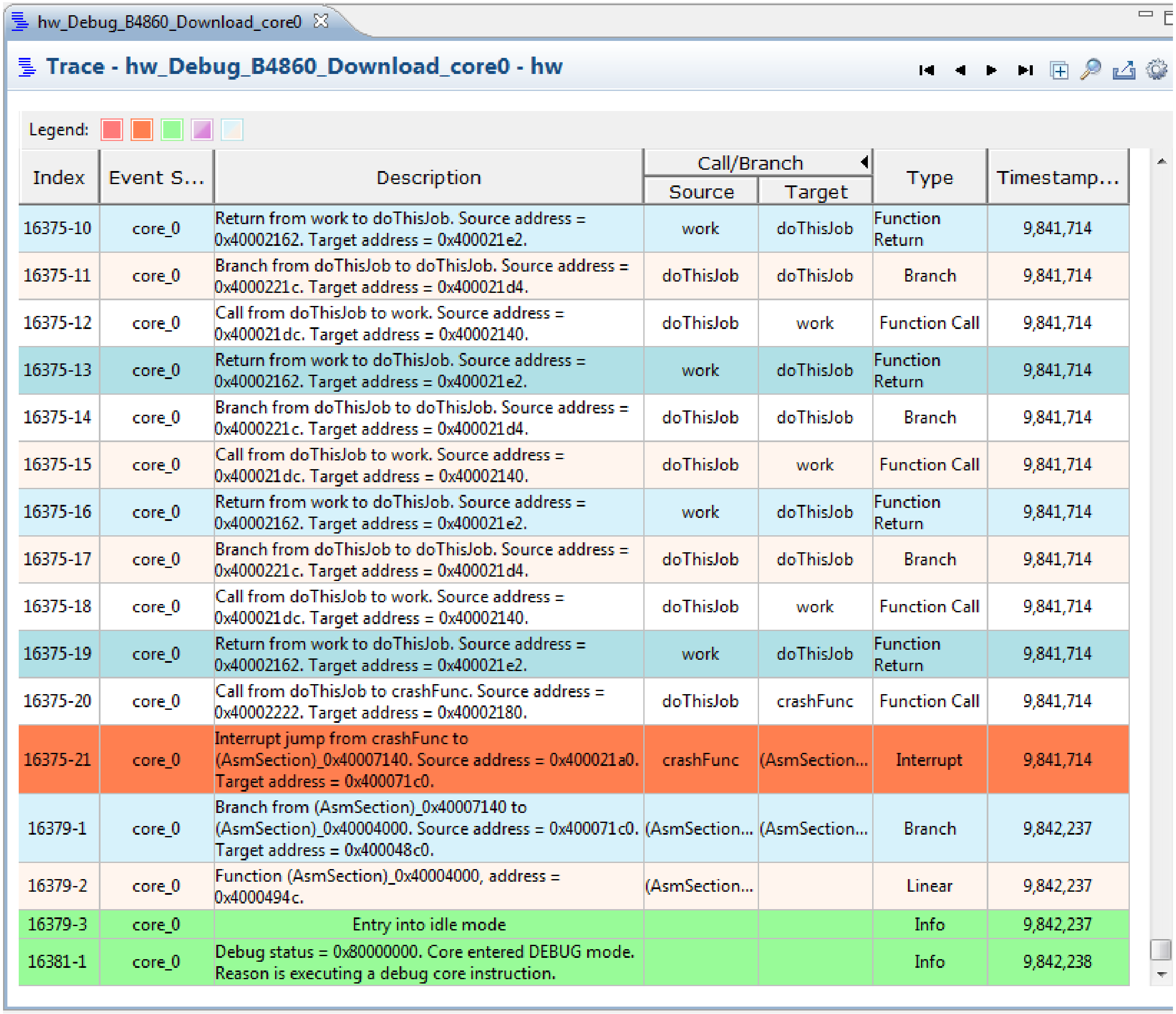
The **Upload Trace** button gets enabled every time whenever the **Trace collection is ON**, therefore the trace can be uploaded even when trace has already been collected at a suspend state (when debug session is not in the attach mode). Following are the effects while uploading trace data after trace has already been collected:

1. If the trace buffer is configured in **One buffer** mode or **Overwrite** mode and the upload trace button is hit to upload the trace data then the already collected trace data will be removed. The trace buffer will be read once again and trace data should look same as before hitting the upload trace button.
2. If trace buffer is configured in **Continuous** mode and upload trace button is hit then the trace data already collected is removed, the trace buffer will be read once again and the trace will become empty (reset the trace buffer in the continuous mode to unfreeze the core).

3.7.1.3 Investigate Crash

Inspect trace in Trace viewer towards its end.

The following figure displays the trace data in Trace viewer when an application crashes and when trace mode has been set to Program trace. For more information on how trace mode can be set to Program trace or Profiling trace, refer Hardware Profiling.



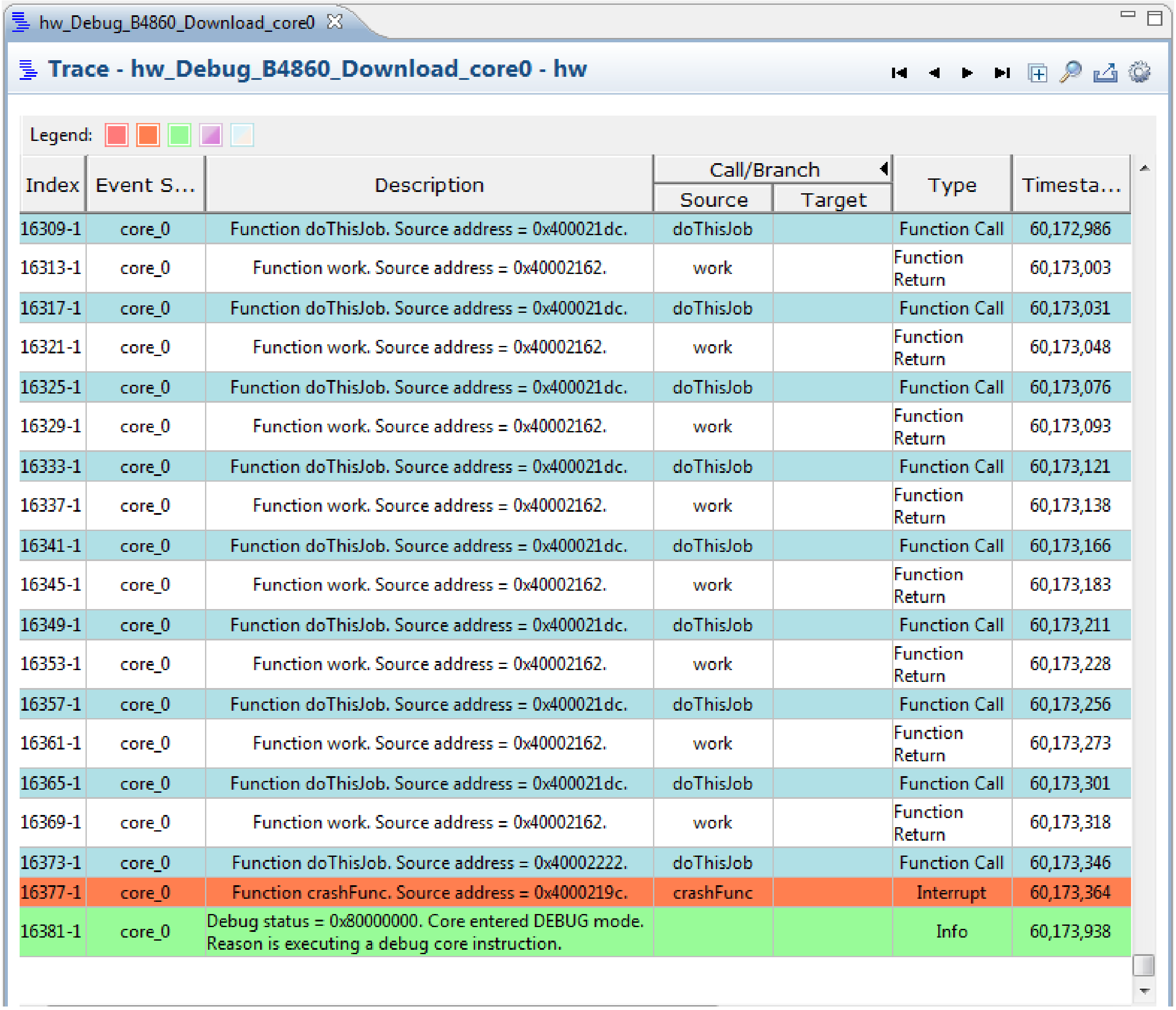
hw_Debug_B4860_Download_core0
Trace - hw_Debug_B4860_Download_core0 - hw

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timestamp...
			Source	Target		
16375-10	core_0	Return from work to doThisJob. Source address = 0x40002162. Target address = 0x400021e2.	work	doThisJob	Function Return	9,841,714
16375-11	core_0	Branch from doThisJob to doThisJob. Source address = 0x4000221c. Target address = 0x400021d4.	doThisJob	doThisJob	Branch	9,841,714
16375-12	core_0	Call from doThisJob to work. Source address = 0x400021dc. Target address = 0x40002140.	doThisJob	work	Function Call	9,841,714
16375-13	core_0	Return from work to doThisJob. Source address = 0x40002162. Target address = 0x400021e2.	work	doThisJob	Function Return	9,841,714
16375-14	core_0	Branch from doThisJob to doThisJob. Source address = 0x4000221c. Target address = 0x400021d4.	doThisJob	doThisJob	Branch	9,841,714
16375-15	core_0	Call from doThisJob to work. Source address = 0x400021dc. Target address = 0x40002140.	doThisJob	work	Function Call	9,841,714
16375-16	core_0	Return from work to doThisJob. Source address = 0x40002162. Target address = 0x400021e2.	work	doThisJob	Function Return	9,841,714
16375-17	core_0	Branch from doThisJob to doThisJob. Source address = 0x4000221c. Target address = 0x400021d4.	doThisJob	doThisJob	Branch	9,841,714
16375-18	core_0	Call from doThisJob to work. Source address = 0x400021dc. Target address = 0x40002140.	doThisJob	work	Function Call	9,841,714
16375-19	core_0	Return from work to doThisJob. Source address = 0x40002162. Target address = 0x400021e2.	work	doThisJob	Function Return	9,841,714
16375-20	core_0	Call from doThisJob to crashFunc. Source address = 0x40002222. Target address = 0x40002180.	doThisJob	crashFunc	Function Call	9,841,714
16375-21	core_0	Interrupt jump from crashFunc to (AsmSection)_0x40007140. Source address = 0x400021a0. Target address = 0x400071c0.	crashFunc	(AsmSection...	Interrupt	9,841,714
16379-1	core_0	Branch from (AsmSection)_0x40007140 to (AsmSection)_0x40004000. Source address = 0x400071c0. Target address = 0x400048c0.	(AsmSection...	(AsmSection...	Branch	9,842,237
16379-2	core_0	Function (AsmSection)_0x40004000, address = 0x4000494c.	(AsmSection...		Linear	9,842,237
16379-3	core_0	Entry into idle mode			Info	9,842,237
16381-1	core_0	Debug status = 0x80000000. Core entered DEBUG mode. Reason is executing a debug core instruction.			Info	9,842,238

Figure 3-55. Cash Investigation in Trace Viewer using Program Trace

The following figure shows the trace data in Trace viewer when an application crashes and when trace mode has been set to profiling trace.



hw_Debug_B4860_Download_core0

Trace - hw_Debug_B4860_Download_core0 - hw

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
16309-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,172,986
16313-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,003
16317-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,031
16321-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,048
16325-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,076
16329-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,093
16333-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,121
16337-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,138
16341-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,166
16345-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,183
16349-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,211
16353-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,228
16357-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,256
16361-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,273
16365-1	core_0	Function doThisJob. Source address = 0x400021dc.	doThisJob		Function Call	60,173,301
16369-1	core_0	Function work. Source address = 0x40002162.	work		Function Return	60,173,318
16373-1	core_0	Function doThisJob. Source address = 0x40002222.	doThisJob		Function Call	60,173,346
16377-1	core_0	Function crashFunc. Source address = 0x4000219c.	crashFunc		Interrupt	60,173,364
16381-1	core_0	Debug status = 0x80000000. Core entered DEBUG mode. Reason is executing a debug core instruction.			Info	60,173,938

Figure 3-56. Cash Investigation in Trace Viewer using Profiling Trace

Tip

Exceptions can be easily recognized in Trace viewer by their orange color.

3.7.2 On Changing Trace Configuration

Trace on attach feature can also be used to change trace configuration on board when a process is already running on it.

For instance you can attach to the running process and enable trace if it was not enabled before or you can change from program trace to profiling trace. To change the trace configuration using trace on attach feature, perform the following steps:

- Create an Attach Launch Configuration
- Change Trace Configuration

3.7.2.1 Create an Attach Launch Configuration

To create an Attach launch configuration, refer Create an Attach Launch Configuration.

3.7.2.2 Change Trace Configuration

Perform the following steps after creating the attach launch configuration:

1. Suspend the debug session to configure trace.
2. Click **Configure Trace** icon available under the **Trace configure** column in the Trace Commander view.

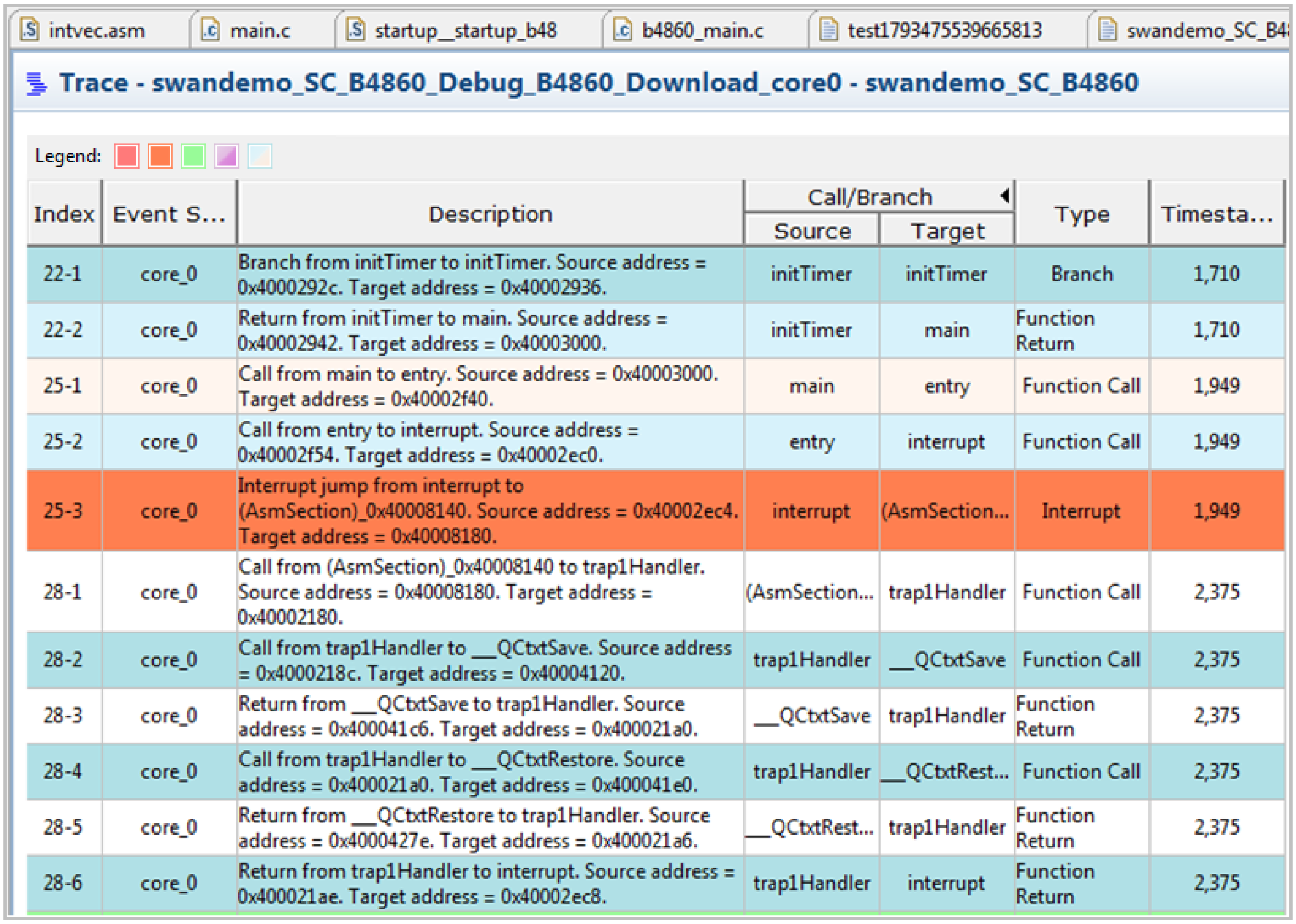
   The **Configure Trace** dialog box appears.
3. Select **Program trace** option from the **Trace scenarios** list.
4. Click **OK**.

The new configuration gets applied on the target when debug session is resumed.

3.8 Viewing Interrupts

The interrupts in an application flow can be easily recognized in Trace viewer with their 'Orange' color and their **Type** flag.

The following figure shows an example on how interrupts are displayed for Program trace in Trace viewer:



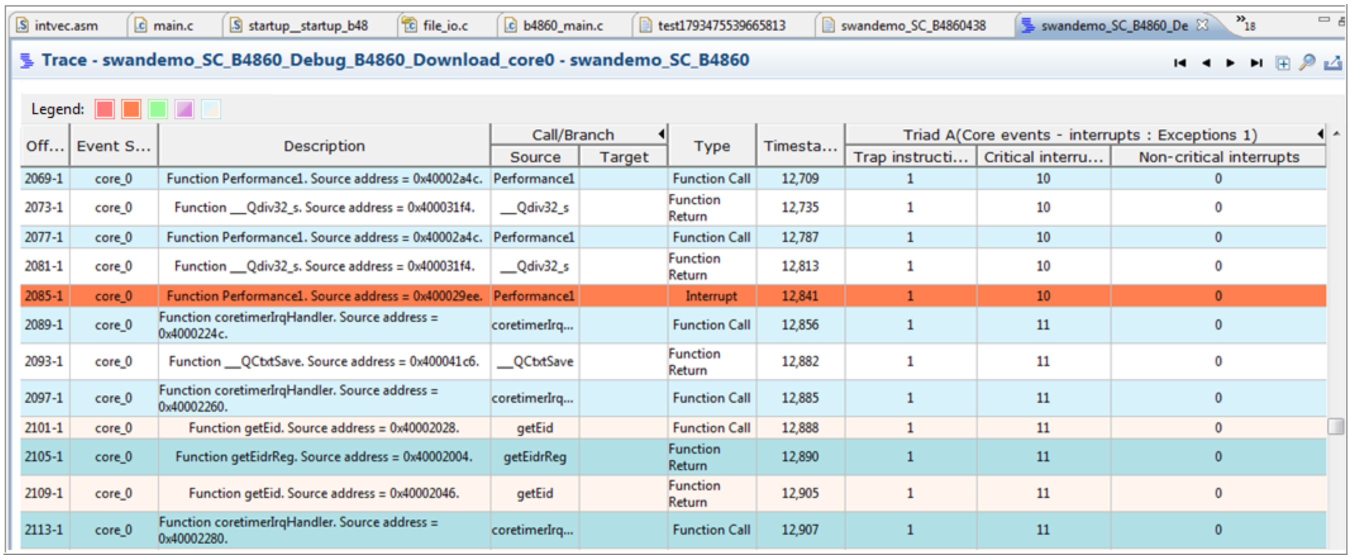
Trace - swandemo_SC_B4860_Debug_B4860_Download_core0 - swandemo_SC_B4860

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
22-1	core_0	Branch from initTimer to initTimer. Source address = 0x4000292c. Target address = 0x40002936.	initTimer	initTimer	Branch	1,710
22-2	core_0	Return from initTimer to main. Source address = 0x40002942. Target address = 0x40003000.	initTimer	main	Function Return	1,710
25-1	core_0	Call from main to entry. Source address = 0x40003000. Target address = 0x40002f40.	main	entry	Function Call	1,949
25-2	core_0	Call from entry to interrupt. Source address = 0x40002f54. Target address = 0x40002ec0.	entry	interrupt	Function Call	1,949
25-3	core_0	Interrupt jump from interrupt to (AsmSection)_0x40008140. Source address = 0x40002ec4. Target address = 0x40008180.	interrupt	(AsmSection...	Interrupt	1,949
28-1	core_0	Call from (AsmSection)_0x40008140 to trap1Handler. Source address = 0x40008180. Target address = 0x40002180.	(AsmSection...	trap1Handler	Function Call	2,375
28-2	core_0	Call from trap1Handler to __QCtxtSave. Source address = 0x4000218c. Target address = 0x40004120.	trap1Handler	__QCtxtSave	Function Call	2,375
28-3	core_0	Return from __QCtxtSave to trap1Handler. Source address = 0x400041c6. Target address = 0x400021a0.	__QCtxtSave	trap1Handler	Function Return	2,375
28-4	core_0	Call from trap1Handler to __QCtxtRestore. Source address = 0x400021a0. Target address = 0x400041e0.	trap1Handler	__QCtxtRest...	Function Call	2,375
28-5	core_0	Return from __QCtxtRestore to trap1Handler. Source address = 0x4000427e. Target address = 0x400021a6.	__QCtxtRest...	trap1Handler	Function Return	2,375
28-6	core_0	Return from trap1Handler to interrupt. Source address = 0x400021ae. Target address = 0x40002ec8.	trap1Handler	interrupt	Function Return	2,375

Figure 3-57. Program Trace Interrupts Displayed in Trace Viewer

The below figure shows an example on how interrupts are displayed for Profiling trace in Trace viewer:



Trace - swandemo_SC_B4860_Debug_B4860_Download_core0 - swandemo_SC_B4860

Legend: 🟥 🟧 🟩 🟪 ⬜

Off...	Event S...	Description	Call/Branch		Type	Timesta...	Triad A(Core events - interrupts : Exceptions 1)		
			Source	Target			Trap instructi...	Critical interru...	Non-critical interrupts
2069-1	core_0	Function Performance1. Source address = 0x40002a4c.	Performance1		Function Call	12,709	1	10	0
2073-1	core_0	Function __Qdiv32_s. Source address = 0x400031f4.	__Qdiv32_s		Function Return	12,735	1	10	0
2077-1	core_0	Function Performance1. Source address = 0x40002a4c.	Performance1		Function Call	12,787	1	10	0
2081-1	core_0	Function __Qdiv32_s. Source address = 0x400031f4.	__Qdiv32_s		Function Return	12,813	1	10	0
2085-1	core_0	Function Performance1. Source address = 0x400029ee.	Performance1		Interrupt	12,841	1	10	0
2089-1	core_0	Function coretimerIrqHandler. Source address = 0x4000224c.	coretimerIrq...		Function Call	12,856	1	11	0
2093-1	core_0	Function __QCtxtSave. Source address = 0x400041c6.	__QCtxtSave		Function Return	12,882	1	11	0
2097-1	core_0	Function coretimerIrqHandler. Source address = 0x40002260.	coretimerIrq...		Function Call	12,885	1	11	0
2101-1	core_0	Function getEid. Source address = 0x40002028.	getEid		Function Call	12,888	1	11	0
2105-1	core_0	Function getEidrReg. Source address = 0x40002004.	getEidrReg		Function Return	12,890	1	11	0
2109-1	core_0	Function getEid. Source address = 0x40002046.	getEid		Function Return	12,905	1	11	0
2113-1	core_0	Function coretimerIrqHandler. Source address = 0x40002280.	coretimerIrq...		Function Call	12,907	1	11	0

Figure 3-58. Profiling Trace Interrupts Displayed in Trace Viewer

3.9 Preparing Aurora Interface for Collecting Aurora Nexus Trace

This topic describes how to prepare Aurora interface to extract Nexus trace from the Aurora port (for processors that support Aurora) using a Gigabit tap with Aurora daughter card.

If you want to collect Nexus trace from the target using Aurora, you need to first prepare the Aurora interface to be used by the trace probe. A TCL script or the Aurora training script is used to train Aurora on the target. This topic contains the following sub-topics:

- Enabling Aurora
- Executing Aurora Training Script

3.9.1 Enabling Aurora

You can enable Aurora on B4860QDS board using Reset Configuration Word (RCW). For B4860QDS boards, Aurora lanes need to be properly routed to Aurora connector on the board. This is done automatically from u-boot, when a RCW with Aurora enabled is detected.

When Aurora is enabled, the Aurora interface needs to be trained so that it can run properly.

3.9.2 Executing Aurora Training Script

The Aurora training script should be executed before configuring or extracting trace through the Aurora port. You can execute the training script to train Aurora on B4860 targets as follows.

For B4860QDS target, the TCL script that is used to train Aurora on the target is available in the CodeWarrior installation folder at:

```
SC/StarCore_Support/Initialization_Files/Aurora/train_aurora_b4.tcl
```

The routine inside this file which executes the required training is called **train_aurora**.

To train Aurora on the B4860QDS target:

1. Ensure that the RCW on board is having Aurora enabled on SERDES#1 and the speed is set to 2.5 Gbps. You can modify RCW fields for different SERDES protocol values to have Aurora enabled, for example:
   - For bits 128-134 in RCW, configure Serdes#1 protocol. Make sure that protocol selected is 0x30. It will enable 2 Aurora lanes.
   - For bit 176 in RCW, configure Aurora divisor. Make sure this bit has value 1, so that Aurora operates at 2.5Gbps.

   You can write a new RCW to target using the CodeWarrior Flash Programmer utility.

2. Execute u-boot. A proper u-boot detects that Aurora is enabled in RCW and initializes cross-points for routing Aurora signals to Aurora connector. If you do not have an u-boot or you have an u-boot without Aurora routing support, you can write it using Flash programmer from CodeWarrior.

   NOTE

   For more information on writing u-boot using Flash programmer, refer to *CodeWarrior Common Features Guide.pdf* in the folder: `<CWInstallDir>\SC\Help\PDF`, where `<CWInstallDir>` is the directory where CodeWarrior for StarCore V10.x is installed.

3. Run the TCL script from the **CCS** console window or the **Debugger Shell** window.
   - From the **CCS** console window:
     1. Make sure you have a proper Command Converter started and configured.
     2. In console, type: `source {$path}/train_aurora_b4.tcl`, where `$path` is a valid system path to the TCL script.
     3. In console, type: `train_aurora` to call the training procedure.
     4. Look for a confirmation message that Aurora interface was successfully trained. If it was not successfully trained, repeat step c.
   - From the **Debugger Shell** window:
     1. In shell, type: `source {$path}/train_aurora_b4.tcl`, where `$path` is a valid system path to the TCL script.
     2. In shell: `train_aurora_dbg` to call the training procedure.
     3. Look for a confirmation message that Aurora interface was successfully trained. If it was not successfully trained, repeat step b.

     NOTE

     A target reset after u-boot initialization for Aurora will result into a link down. This means that Aurora interface cannot be used anymore. If you use CodeWarrior for downloading the projects on the target, make sure you have unchecked the option to reset the target. If you need to reset the target, ensure after reset that u-boot has a chance to run and initialize routing signals for using Aurora (estimated time is 10 seconds). You can use an initialization script that can handle a target already initialized by u-boot.

NOTE

If you see HBDP errors in the collected trace, it is useful to run train_aurora_dbg in the Debugger Shell window. This helps in eliminating most of the HBDP errors from trace.

3.10 Inline Functions

Inline functions are used to optimize the source code. It is typically used for functions that execute frequently.

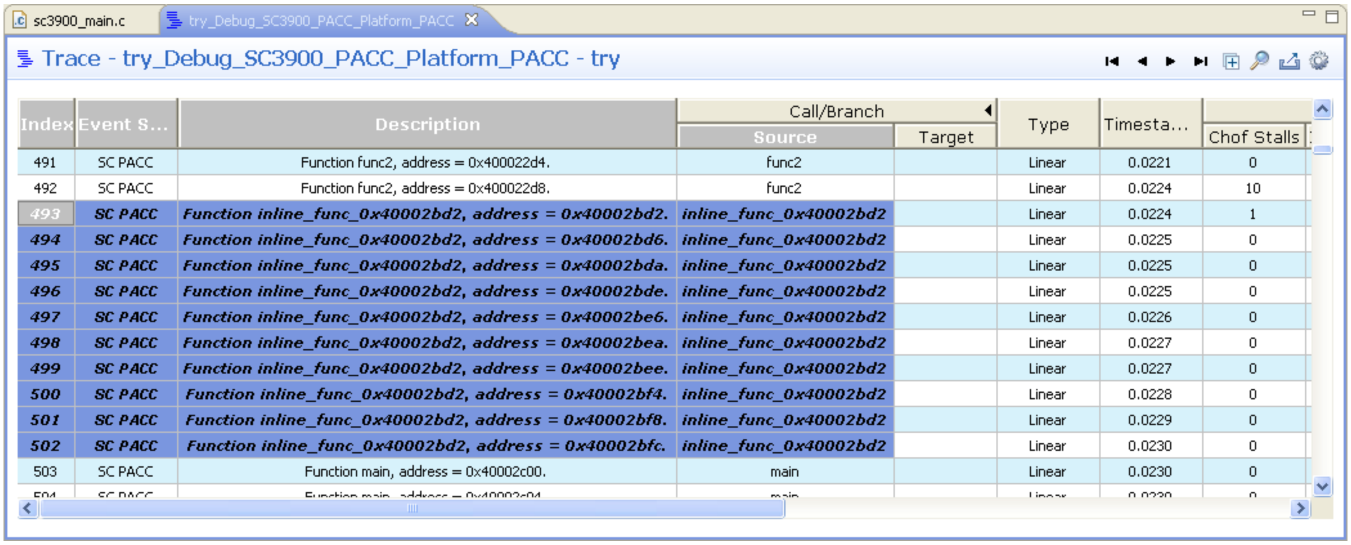
Perform the following steps to add inline function:

1. Create a Stationary project for SC3900 PACC.
2. Add inline function in source file, `sc3900_main.c`. The following example is taken: `int inline_func(int i){#pragma inline;i++;i+=200;return i;}`
3. Select the project and right-click on it. From the context menu, select **Properties**. The **Properties** dialog box appears. Select **C/C++ Build -> Settings -> Starcore 3900 C/C++ Linker Application -> Linker Settings**. Uncheck **Strip Dead Code** checkbox.
4. Build the project.
5. Open Debug Configurations dialog box and check **Enable Trace and Profile** checkbox in the **Trace and Profile** tab.
6. Click **Apply** and debug the application.
7. Resume debug session and then suspend the debug session after sometime.
8. To open Software Analysis view, click **Windows -> Show View -> Software Analysis**.
9. Check for inline function in the Trace, Timeline, Critical Code, Performance, Call Tree viewers.

NOTE

On B4860 SC3900 targets, the only profiling feature providing support for inline functions is Critical Code. Timeline, Performance and Call Tree features do not provide support for inline functions because of hardware trace limitation. Profiling trace messages used by these features are being generated only on call and return instructions. Since body of inline functions is embedded into parent function that calls inline function, trace profiler features based on functions calls are not aware of inline functions.

In Trace data viewer, inlined functions are displayed as <function_name_address>, for example, inline_func_0x40002bd2.

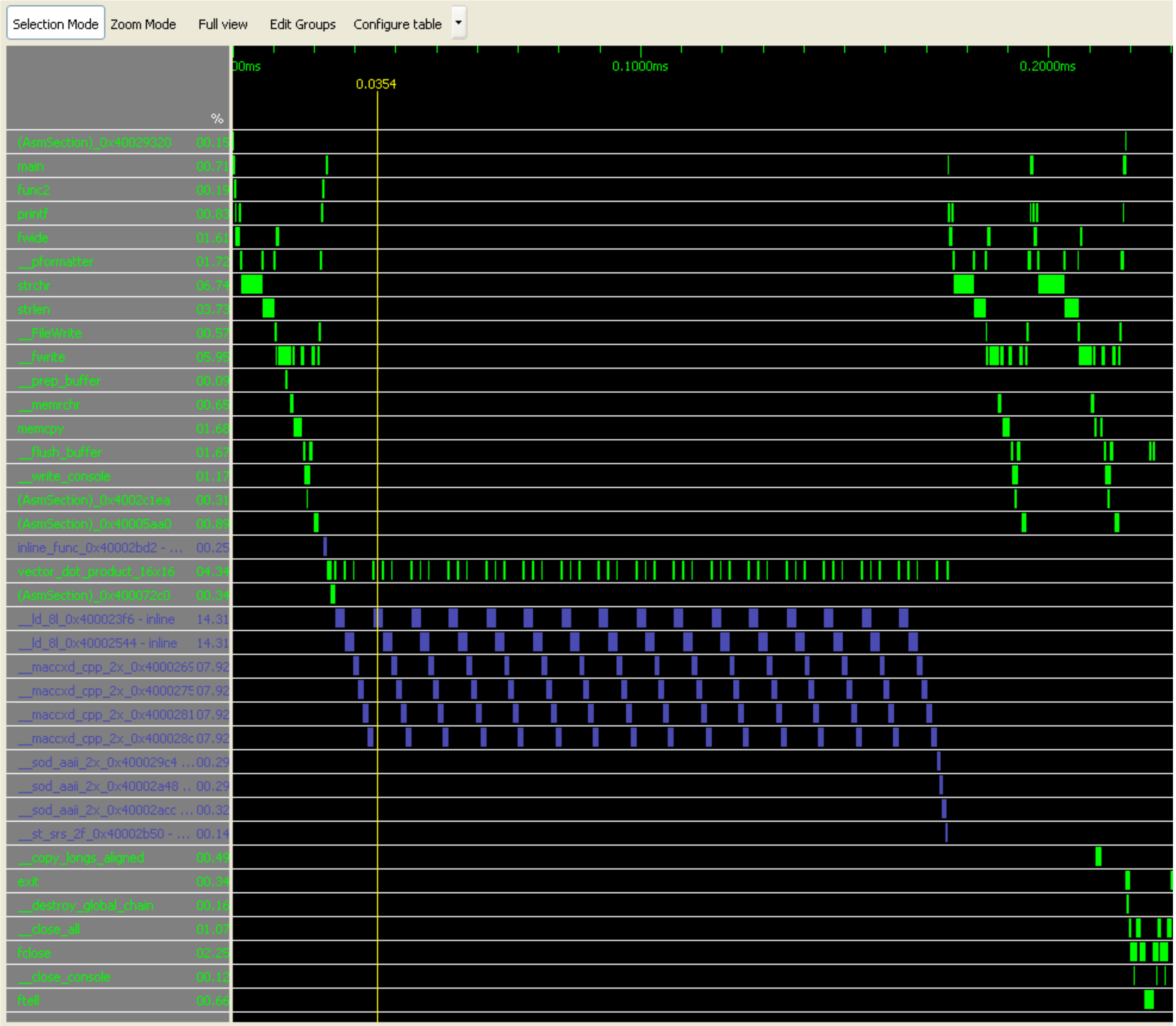


sc3900_main.c
try_Debug_SC3900_PACC_Platform_PACC
Trace - try_Debug_SC3900_PACC_Platform_PACC - try

Index	Event S...	Description	Call/Branch		Type	Timesta...	
			Source	Target			Chof Stalls
491	SC PACC	Function func2, address = 0x400022d4.	func2		Linear	0.0221	0
492	SC PACC	Function func2, address = 0x400022d8.	func2		Linear	0.0224	10
493	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bd2.	inline_func_0x40002bd2		Linear	0.0224	1
494	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bd6.	inline_func_0x40002bd2		Linear	0.0225	0
495	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bda.	inline_func_0x40002bd2		Linear	0.0225	0
496	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bde.	inline_func_0x40002bd2		Linear	0.0225	0
497	SC PACC	Function inline_func_0x40002bd2, address = 0x40002be6.	inline_func_0x40002bd2		Linear	0.0226	0
498	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bea.	inline_func_0x40002bd2		Linear	0.0227	0
499	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bee.	inline_func_0x40002bd2		Linear	0.0227	0
500	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bf4.	inline_func_0x40002bd2		Linear	0.0228	0
501	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bf8.	inline_func_0x40002bd2		Linear	0.0229	0
502	SC PACC	Function inline_func_0x40002bd2, address = 0x40002bfc.	inline_func_0x40002bd2		Linear	0.0230	0
503	SC PACC	Function main, address = 0x40002c00.	main		Linear	0.0230	0
504	SC PACC	Function main, address = 0x40002c04	main		Linear	0.0230	0

Figure 3-59. Trace Data with Inline Function

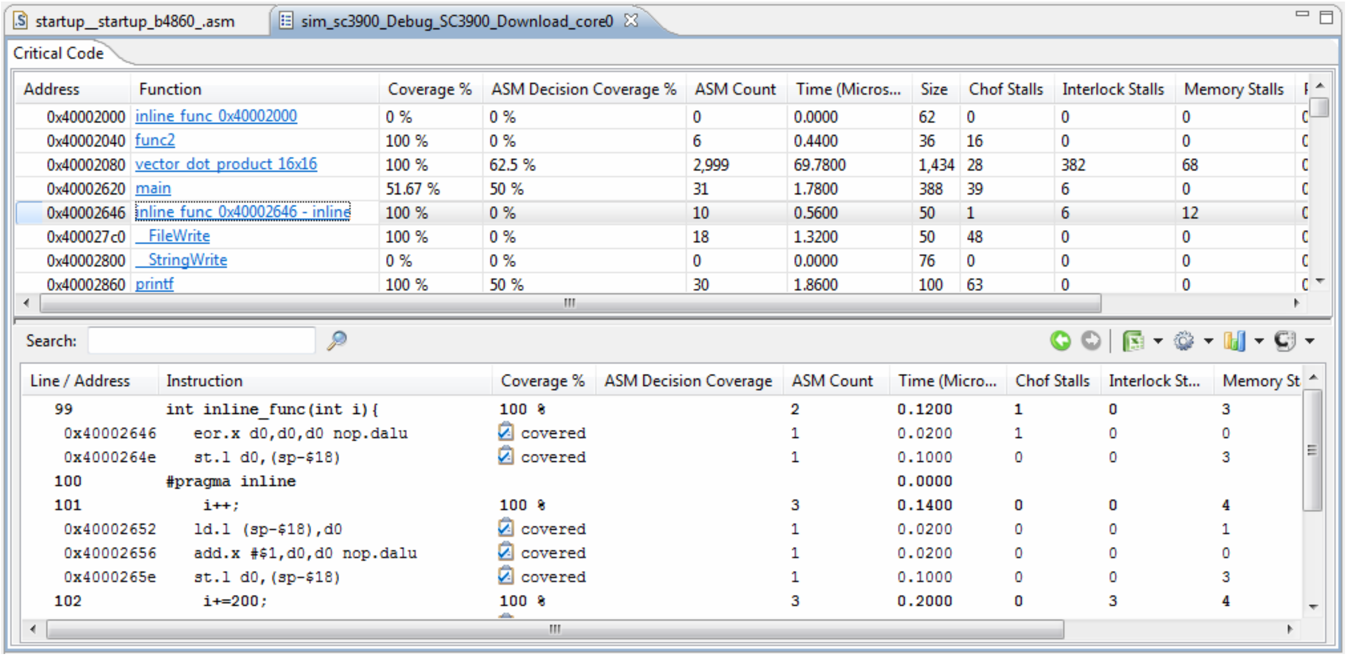
In Timeline data viewer, inlined functions are represented in a different color than default one. For example, inlined function details are shown in blue.



	%
(AsmSection)_0x40029320	00.15
main	00.71
func2	00.19
printf	00.83
fwide	01.61
__pformatter	01.72
strchr	06.74
strlen	03.73
__FileWrite	00.57
__fwrite	05.95
__prep_buffer	00.09
__memrchr	00.65
memcpy	01.68
__flush_buffer	01.67
__write_console	01.17
(AsmSection)_0x4002c1ea	00.31
(AsmSection)_0x40005aa0	00.89
inline_func_0x40002bd2 - ...	00.25
vector_dot_product_16x16	04.34
(AsmSection)_0x400072c0	00.34
__ld_8l_0x400023f6 - inline	14.31
__ld_8l_0x40002544 - inline	14.31
__maccxd_cpp_2x_0x4000269	07.92
__maccxd_cpp_2x_0x4000275	07.92
__maccxd_cpp_2x_0x4000281	07.92
__maccxd_cpp_2x_0x400028c	07.92
__sod_aaii_2x_0x400029c4 ...	00.29
__sod_aaii_2x_0x40002a48 ..	00.29
__sod_aaii_2x_0x40002acc ...	00.32
__st_srs_2f_0x40002b50 - ...	00.14
__copy_longs_aligned	00.49
exit	00.34
__destroy_global_chain	00.16
__close_all	01.07
fclose	02.25
__close_console	00.12
ftell	00.66

Figure 3-60. Timeline Data with Inline Function

In Critical Code data viewer, inlined functions are suffixed with `inline`.



startup__startup_b4860_.asm
sim_sc3900_Debug_SC3900_Download_core0
Critical Code

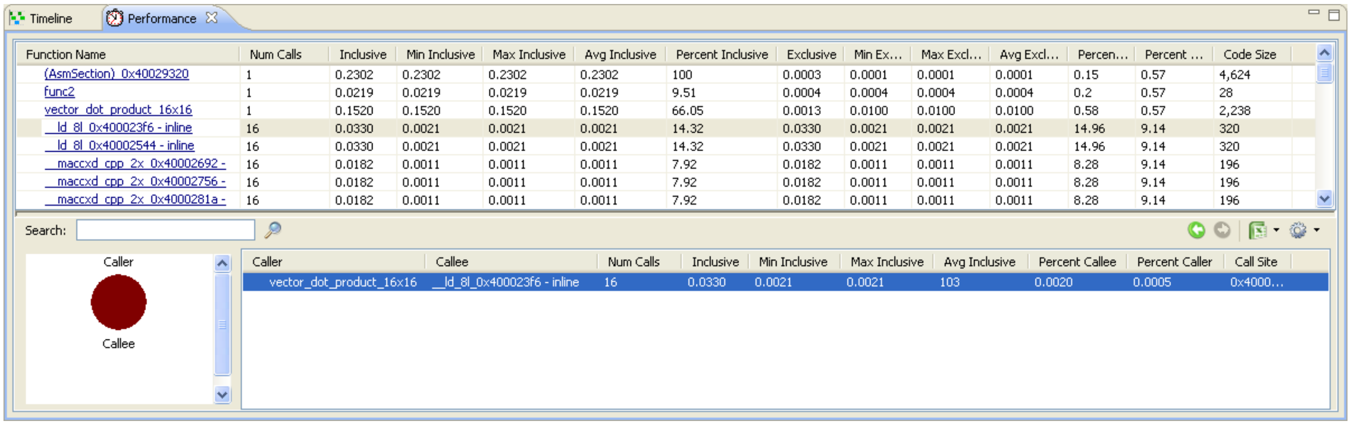
Address	Function	Coverage %	ASM Decision Coverage %	ASM Count	Time (Micros...	Size	Chof Stalls	Interlock Stalls	Memory Stalls	F
0x40002000	inline_func_0x40002000	0 %	0 %	0	0.0000	62	0	0	0	C
0x40002040	func2	100 %	0 %	6	0.4400	36	16	0	0	C
0x40002080	vector_dot_product_16x16	100 %	62.5 %	2,999	69.7800	1,434	28	382	68	C
0x40002620	main	51.67 %	50 %	31	1.7800	388	39	6	0	C
0x40002646	inline_func_0x40002646 - inline	100 %	0 %	10	0.5600	50	1	6	12	C
0x400027c0	__FileWrite	100 %	0 %	18	1.3200	50	48	0	0	C
0x40002800	__StringWrite	0 %	0 %	0	0.0000	76	0	0	0	C
0x40002860	printf	100 %	50 %	30	1.8600	100	63	0	0	C

Search:

Line / Address	Instruction	Coverage %	ASM Decision Coverage	ASM Count	Time (Micro...	Chof Stalls	Interlock St...	Memory St
99	int inline_func(int i){	100 %		2	0.1200	1	0	3
0x40002646	eor.x d0,d0,d0 nop.dalu	☑ covered		1	0.0200	1	0	0
0x4000264e	st.l d0,(sp-$18)	☑ covered		1	0.1000	0	0	3
100	#pragma inline				0.0000			
101	i++;	100 %		3	0.1400	0	0	4
0x40002652	ld.l (sp-$18),d0	☑ covered		1	0.0200	0	0	1
0x40002656	add.x #$1,d0,d0 nop.dalu	☑ covered		1	0.0200	0	0	0
0x4000265e	st.l d0,(sp-$18)	☑ covered		1	0.1000	0	0	3
102	i+=200;	100 %		3	0.2000	0	3	4

Figure 3-61. Critical Code Data with Inline Function

In Performance data viewer, inlined functions are suffixed with `inline`.



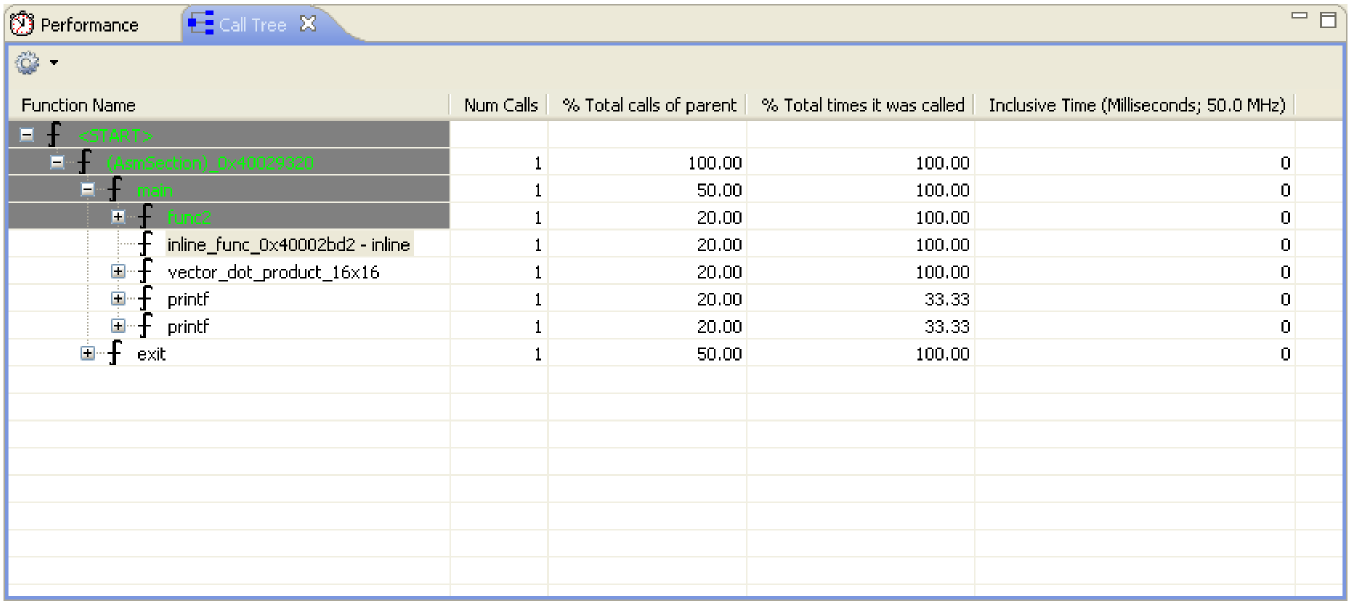
Function Name	Num Calls	Inclusive	Min Inclusive	Max Inclusive	Avg Inclusive	Percent Inclusive	Exclusive	Min Ex...	Max Excl...	Avg Excl...	Percen...	Percent ...	Code Size
(AsmSection)_0x40029320	1	0.2302	0.2302	0.2302	0.2302	100	0.0003	0.0001	0.0001	0.0001	0.15	0.57	4,624
func2	1	0.0219	0.0219	0.0219	0.0219	9.51	0.0004	0.0004	0.0004	0.0004	0.2	0.57	28
vector_dot_product_16x16	1	0.1520	0.1520	0.1520	0.1520	66.05	0.0013	0.0100	0.0100	0.0100	0.58	0.57	2,238
__ld_8l_0x400023f6 - inline	16	0.0330	0.0021	0.0021	0.0021	14.32	0.0330	0.0021	0.0021	0.0021	14.96	9.14	320
__ld_8l_0x40002544 - inline	16	0.0330	0.0021	0.0021	0.0021	14.32	0.0330	0.0021	0.0021	0.0021	14.96	9.14	320
__maccxd_cpp_2x_0x40002692 -	16	0.0182	0.0011	0.0011	0.0011	7.92	0.0182	0.0011	0.0011	0.0011	8.28	9.14	196
__maccxd_cpp_2x_0x40002756 -	16	0.0182	0.0011	0.0011	0.0011	7.92	0.0182	0.0011	0.0011	0.0011	8.28	9.14	196
__maccxd_cpp_2x_0x4000281a -	16	0.0182	0.0011	0.0011	0.0011	7.92	0.0182	0.0011	0.0011	0.0011	8.28	9.14	196

Search:

Caller	Callee	Num Calls	Inclusive	Min Inclusive	Max Inclusive	Avg Inclusive	Percent Callee	Percent Caller	Call Site
vector_dot_product_16x16	__ld_8l_0x400023f6 - inline	16	0.0330	0.0021	0.0021	103	0.0020	0.0005	0x4000...

Figure 3-62. Performance Data with Inline Function

In Call Tree data viewer, inlined functions are suffixed with `inline`.



Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Milliseconds; 50.0 MHz)
<START>				
(AsmSection)_0x40029320	1	100.00	100.00	0
main	1	50.00	100.00	0
func2	1	20.00	100.00	0
inline_func_0x40002bd2 - inline	1	20.00	100.00	0
vector_dot_product_16x16	1	20.00	100.00	0
printf	1	20.00	33.33	0
printf	1	20.00	33.33	0
exit	1	50.00	100.00	0

Figure 3-63. Call Tree Data with Inline Function

3.11 Importing Offline SC3900 Trace with Hardware Trace Configuration

This feature allows you to import offline SC3900 trace that is configured with hardware profiling.

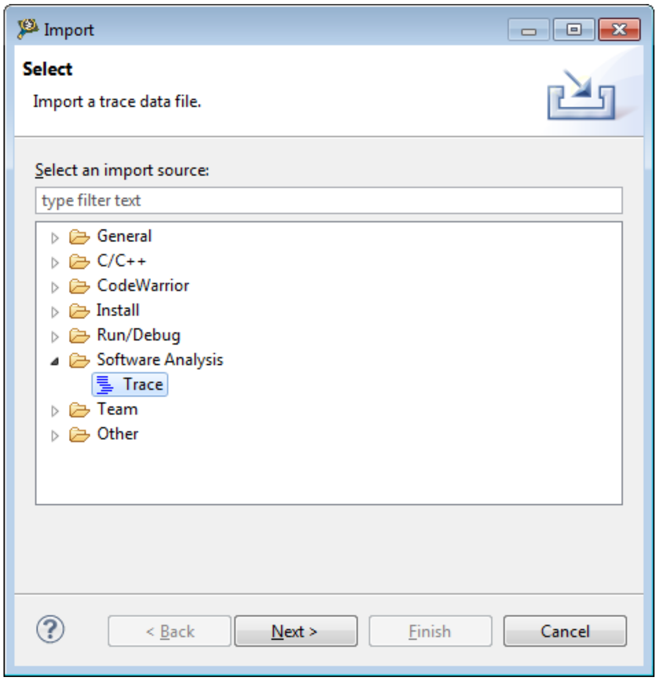
Before importing SC3900 trace, it is important to know that profiling triads were active on board or not when trace was collected.

There are 2 triads (each triad is a group of 3 profiling counters) that are used when trace is imported, triad A and triad B.

The profiling triads should be selected while importing, only if trace contains values for selected counters. If trace does not contain values for selected counters, there is no need to select profiling triads while importing the trace.

To import offline SC3900 trace data into your project:

1. Select **File > Import** to open the Import wizard.
2. Expand the **Software Analysis** node and select **Trace**.



Import

Select

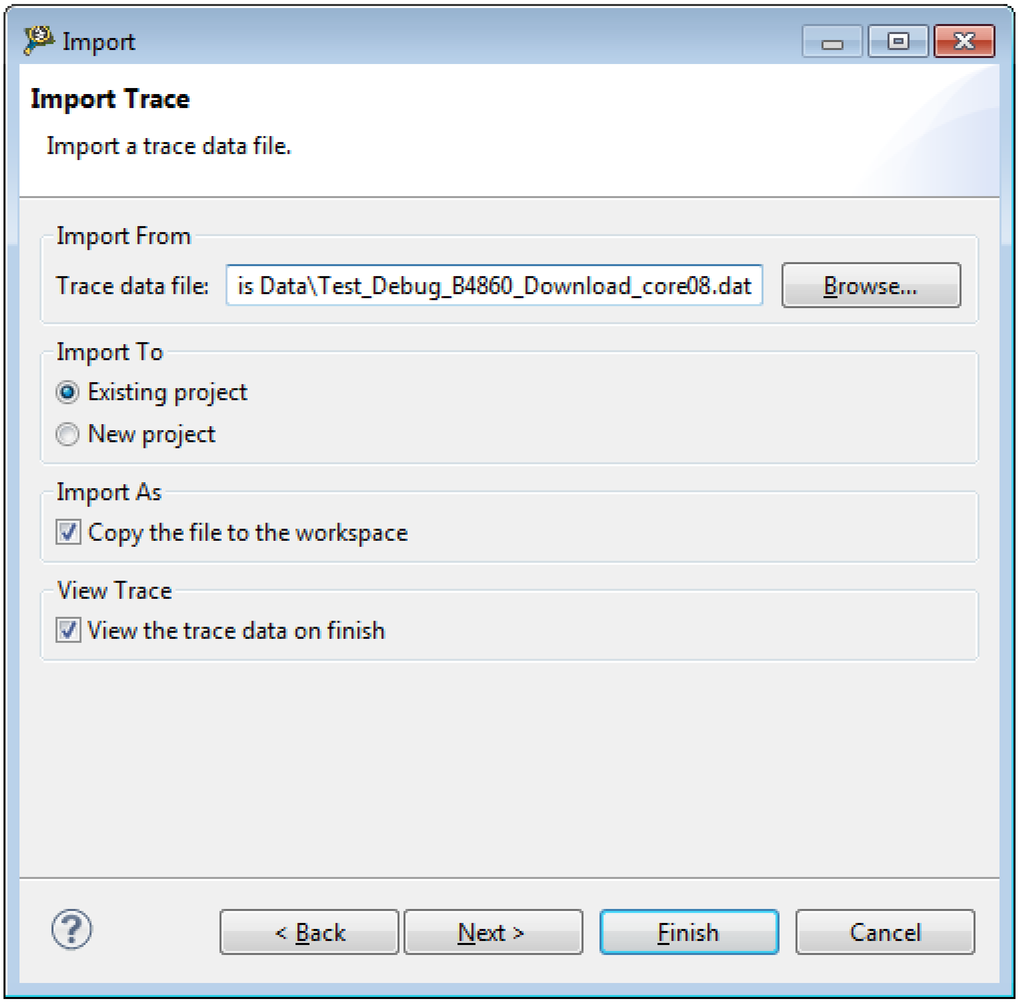
Import a trace data file.

Select an import source:

type filter text

Figure 3-64. Import Wizard

3. Click **Next** to display the **Import Trace** page of the Import wizard.



Import

Import Trace

Import a trace data file.

Import From

Trace data file: is Data\Test_Debug_B4860_Download_core08.dat

Import To

● Existing project

○ New project

Import As

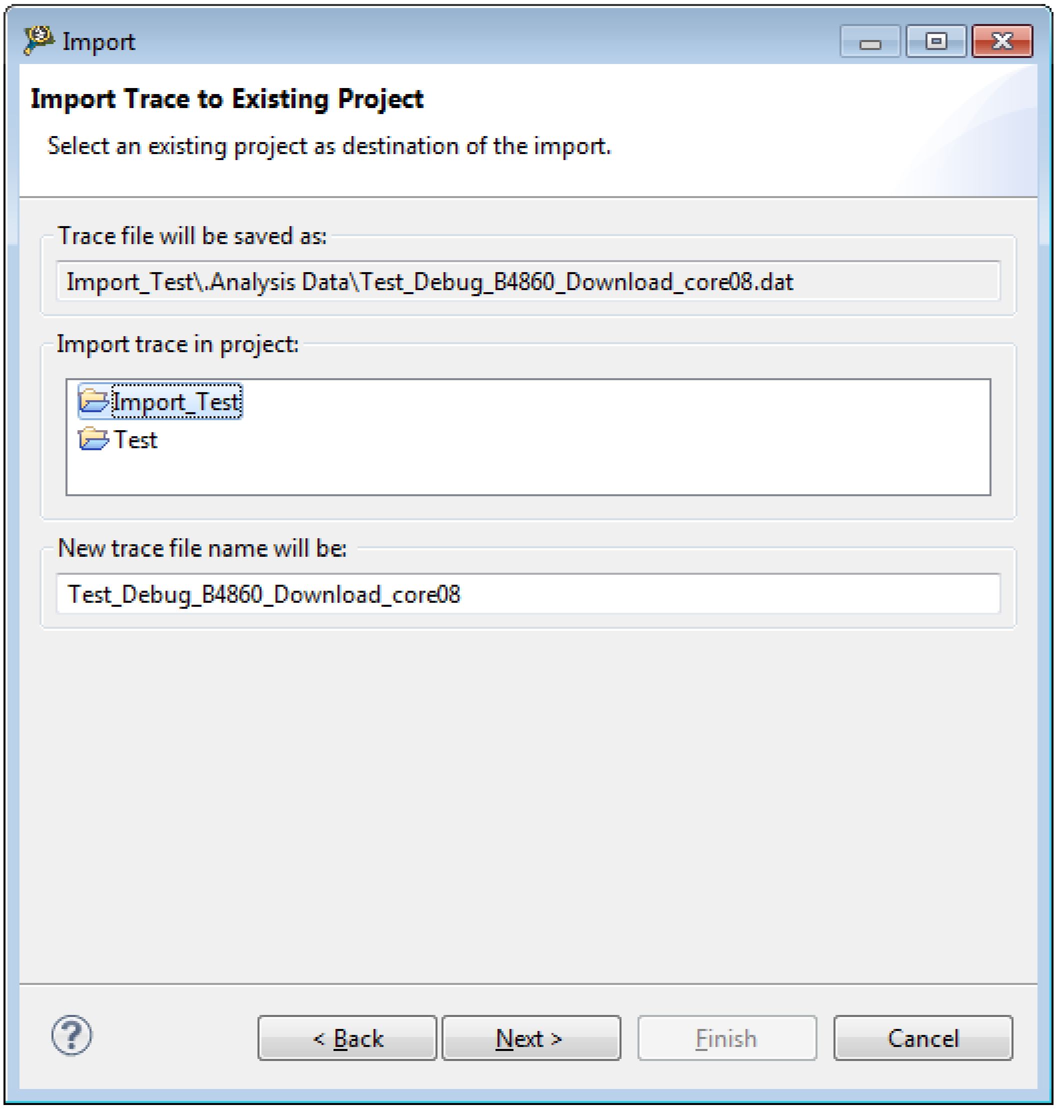
☑ Copy the file to the workspace

View Trace

☑ View the trace data on finish

Figure 3-65. Import Wizard - Import Trace Page

4. Click **Browse** and locate the trace data file of the project that you want to import into your project. The trace data file is located in the **.Analysis Data** folder of the project in the workspace.
5. If you select the **View the trace data on finish** checkbox, the imported trace data will automatically open in the **Trace Data** viewer of your project. If you clear this option, the imported data will be visible from the **Software Analysis** view. The imported trace data is also saved in the `.Analysis Data` folder of your project.
6. Click **Next** to display the **Import Trace to Existing Project** page.

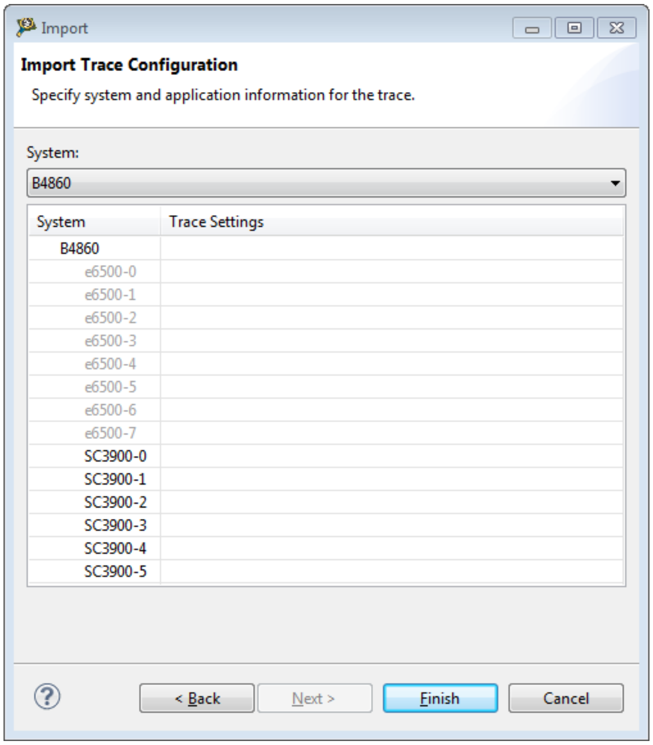

Import
Import Trace to Existing Project
Select an existing project as destination of the import.
Trace file will be saved as:
Import trace in project:
Import_Test
Test
New trace file name will be:
< Back
Next >
Finish
Cancel

Trace file will be saved as: Import_Test\.Analysis Data\Test_Debug_B4860_Download_core08.dat

New trace file name will be: Test_Debug_B4860_Download_core08

Figure 3-66. Import Wizard - Import Trace to Existing Project Page

7. Type or select the project in which you want to import the trace data.
8. Type a new file name for the trace data file in the **New trace file name will be** text box. This field is optional.
9. Click **Next** to display the **Import Trace Configuration** page. This page allows you to specify the trace configuration details of the project to display imported trace data based on the executable file or launch configuration used during trace collection.



Import

Import Trace Configuration

Specify system and application information for the trace.

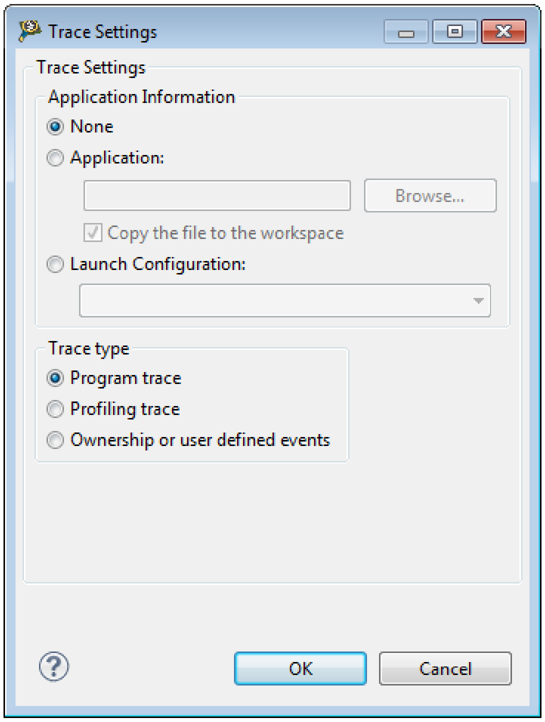
System:

B4860

System	Trace Settings
B4860	
e6500-0	
e6500-1	
e6500-2	
e6500-3	
e6500-4	
e6500-5	
e6500-6	
e6500-7	
SC3900-0	
SC3900-1	
SC3900-2	
SC3900-3	
SC3900-4	
SC3900-5	

Figure 3-67. Import Wizard - Import Trace Configuration Page

10. Select the target of your project from the **System** drop-down box.
11. Select the desired core from the **System** list.
12. Left click In the corresponding **Trace Settings** group, and click … .
    The **Trace Settings** dialog appears.



Trace Settings

Trace Settings

Application Information

● None

○ Application:

Browse...

☑ Copy the file to the workspace

○ Launch Configuration:

Trace type

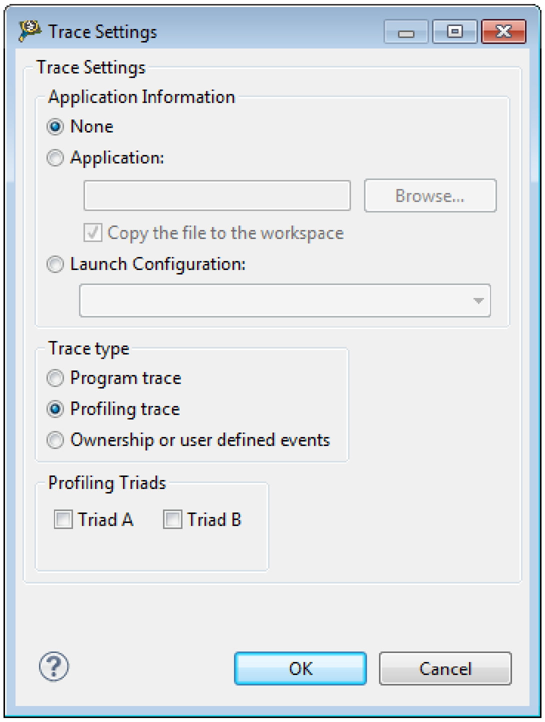
● Program trace

○ Profiling trace

○ Ownership or user defined events

Figure 3-68. Import Wizard - Trace Settings dialog

13. Select any of the following options from the **Trace Settings** group:
    - **None** if you do not want to correlate trace data with the source of your project, that is when the trace being imported is not program trace. The trace data will appear without displaying the source lines corresponding to the trace events.
    - **Application** if you want to correlate trace data with the source. Click **Browse** to locate the executable file (`.abs`/`.elf`/`.afx`) of the application used during trace collection.
    - **Launch Configuration** to select the launch configuration of the application used during collection to identify the executable file or processor/system of the application.
14. Check **Profiling trace** option available under the **Trace type** list.



Trace Settings

Trace Settings

Application Information

● None

○ Application:

☑ Copy the file to the workspace

○ Launch Configuration:

Trace type

○ Program trace

● Profiling trace

○ Ownership or user defined events

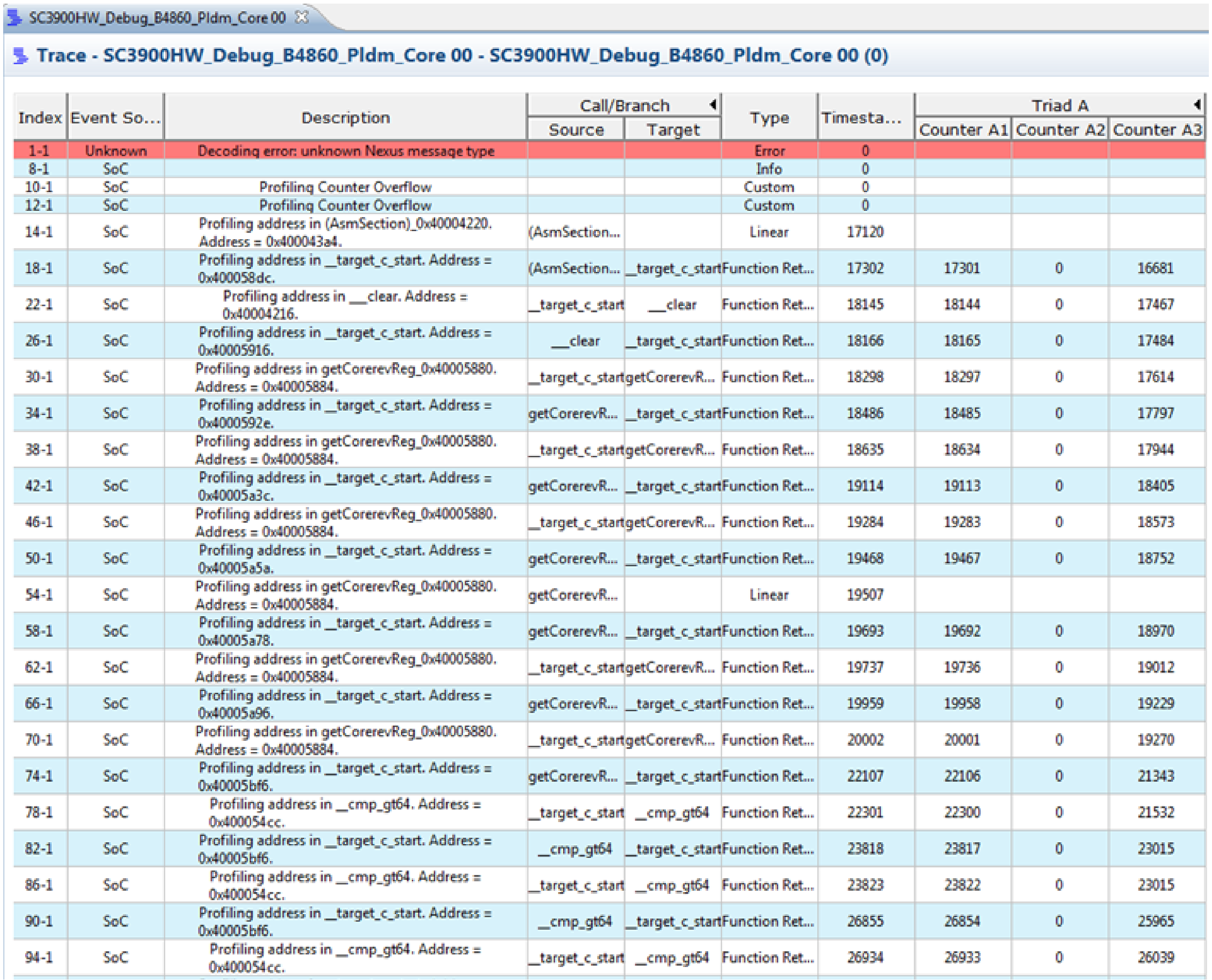
Profiling Triads

□ Triad A □ Triad B

Figure 3-69. Trace Settings dialog - Trace type

15. Select the **Profiling Triads** if the trace that you are importing contains the counter values. The counters of both triads have default name. To change the name of the counters of a triad, select the triad checkbox and the **Edit** button is enabled. Click **Edit** to rename the counters. Legal characters for counter name are A-Z, a-z, ., _, - and whitespace. First legal character is A-Z or a-z. Also, names of counters should be distinct.
16. Click **OK**.
17. Click **Finish**.

If trace data file contains values for Triad A, only Traid A was configured during trace collection and **Profiling - clock cycles** scenario was selected. You should select same counters while importing trace data. The trace viewer displays the 3 counter values of triad A.



SC3900HW_Debug_B4860_Pldm_Core 00

Trace - SC3900HW_Debug_B4860_Pldm_Core 00 - SC3900HW_Debug_B4860_Pldm_Core 00 (0)

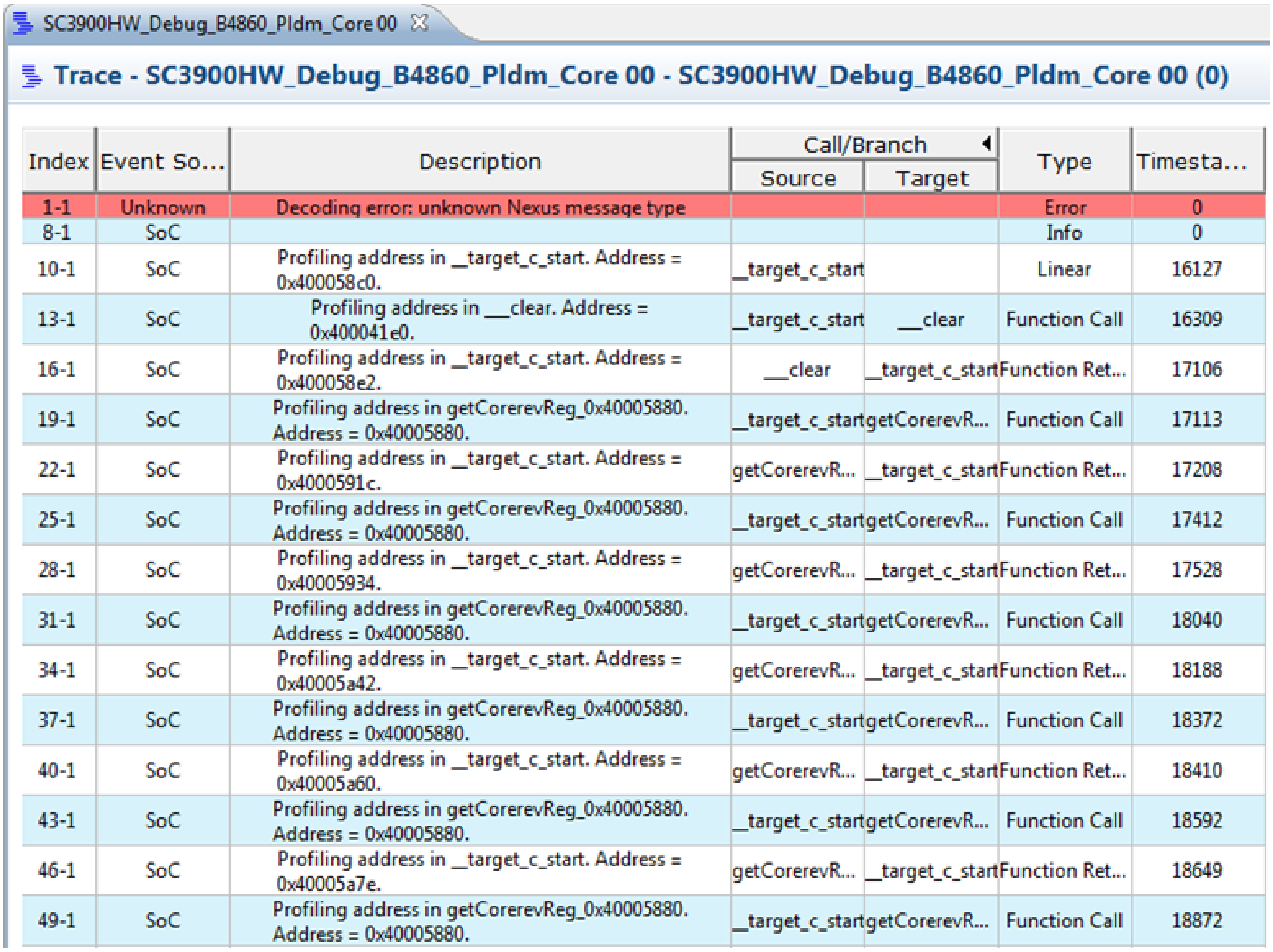
Index	Event So...	Description	Call/Branch		Type	Timesta...	Triad A		
			Source	Target			Counter A1	Counter A2	Counter A3
1-1	Unknown	Decoding error: unknown Nexus message type			Error	0			
8-1	SoC				Info	0			
10-1	SoC	Profiling Counter Overflow			Custom	0			
12-1	SoC	Profiling Counter Overflow			Custom	0			
14-1	SoC	Profiling address in (AsmSection)_0x40004220. Address = 0x400043a4.	(AsmSection...		Linear	17120			
18-1	SoC	Profiling address in __target_c_start. Address = 0x400058dc.	(AsmSection...	__target_c_start	Function Ret...	17302	17301	0	16681
22-1	SoC	Profiling address in __clear. Address = 0x40004216.	__target_c_start	__clear	Function Ret...	18145	18144	0	17467
26-1	SoC	Profiling address in __target_c_start. Address = 0x40005916.	__clear	__target_c_start	Function Ret...	18166	18165	0	17484
30-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	18298	18297	0	17614
34-1	SoC	Profiling address in __target_c_start. Address = 0x4000592e.	getCorerevR...	__target_c_start	Function Ret...	18486	18485	0	17797
38-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	18635	18634	0	17944
42-1	SoC	Profiling address in __target_c_start. Address = 0x40005a3c.	getCorerevR...	__target_c_start	Function Ret...	19114	19113	0	18405
46-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	19284	19283	0	18573
50-1	SoC	Profiling address in __target_c_start. Address = 0x40005a5a.	getCorerevR...	__target_c_start	Function Ret...	19468	19467	0	18752
54-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	getCorerevR...		Linear	19507			
58-1	SoC	Profiling address in __target_c_start. Address = 0x40005a78.	getCorerevR...	__target_c_start	Function Ret...	19693	19692	0	18970
62-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	19737	19736	0	19012
66-1	SoC	Profiling address in __target_c_start. Address = 0x40005a96.	getCorerevR...	__target_c_start	Function Ret...	19959	19958	0	19229
70-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	20002	20001	0	19270
74-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	getCorerevR...	__target_c_start	Function Ret...	22107	22106	0	21343
78-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__target_c_start	__cmp_gt64	Function Ret...	22301	22300	0	21532
82-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	__cmp_gt64	__target_c_start	Function Ret...	23818	23817	0	23015
86-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__target_c_start	__cmp_gt64	Function Ret...	23823	23822	0	23015
90-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	__cmp_gt64	__target_c_start	Function Ret...	26855	26854	0	25965
94-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__target_c_start	__cmp_gt64	Function Ret...	26934	26933	0	26039

Figure 3-70. Trace Data with Triad A Counter Values

The trace data file contains profiling messages for 6 counters (values of triad A and B) if both triads were configured during trace collection and trace scenario **Profiling - L2 cache events** was selected. While importing, if only triad A (3 counters) is selected, trace viewer will not display any counter value (as shown below). Number of triad counters within profiling messages from trace data file must match with number of triad counters selected at the time of importing for trace viewer to display counter columns.

NOTE

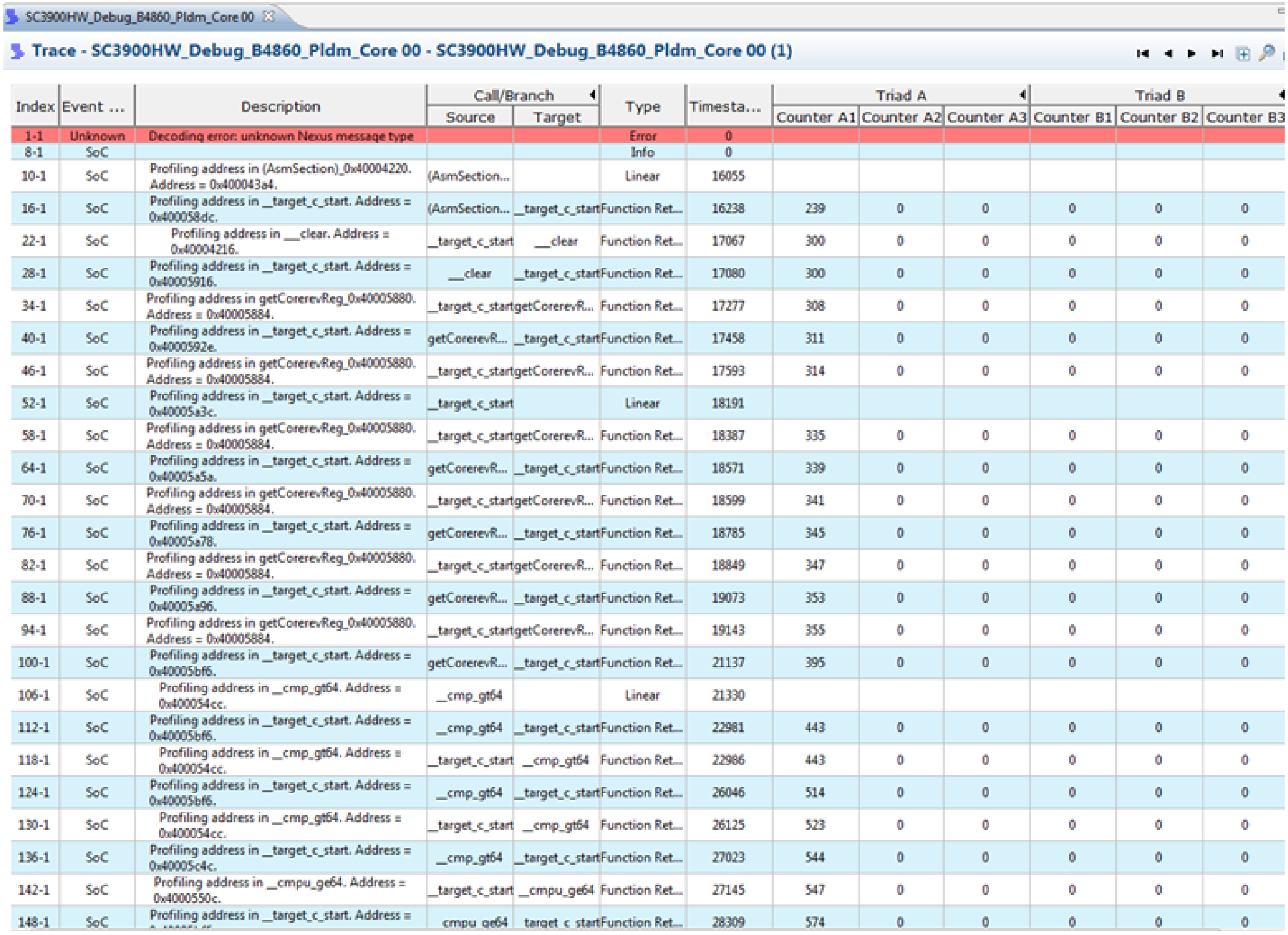
Trace viewer displays trace containing no counter values if the trace was collected using **Program Trace** configuration.


SC3900HW_Debug_B4860_Pldm_Core 00
Trace - SC3900HW_Debug_B4860_Pldm_Core 00 - SC3900HW_Debug_B4860_Pldm_Core 00 (0)

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
1-1	Unknown	Decoding error: unknown Nexus message type			Error	0
8-1	SoC				Info	0
10-1	SoC	Profiling address in __target_c_start. Address = 0x400058c0.	_target_c_start		Linear	16127
13-1	SoC	Profiling address in __clear. Address = 0x400041e0.	_target_c_start	__clear	Function Call	16309
16-1	SoC	Profiling address in __target_c_start. Address = 0x400058e2.	__clear	_target_c_start	Function Ret...	17106
19-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	17113
22-1	SoC	Profiling address in __target_c_start. Address = 0x4000591c.	getCorerevR...	_target_c_start	Function Ret...	17208
25-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	17412
28-1	SoC	Profiling address in __target_c_start. Address = 0x40005934.	getCorerevR...	_target_c_start	Function Ret...	17528
31-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	18040
34-1	SoC	Profiling address in __target_c_start. Address = 0x40005a42.	getCorerevR...	_target_c_start	Function Ret...	18188
37-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	18372
40-1	SoC	Profiling address in __target_c_start. Address = 0x40005a60.	getCorerevR...	_target_c_start	Function Ret...	18410
43-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	18592
46-1	SoC	Profiling address in __target_c_start. Address = 0x40005a7e.	getCorerevR...	_target_c_start	Function Ret...	18649
49-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005880.	_target_c_start	getCorerevR...	Function Call	18872

Figure 3-71. Trace Data with No Counters

You should select both triad A and B while importing, if both triads were configured during trace collection when trace scenario **Profiling -L2 cache events** was selected.



SC3900HW_Debug_B4860_Pldm_Core 00

Trace - SC3900HW_Debug_B4860_Pldm_Core 00 - SC3900HW_Debug_B4860_Pldm_Core 00 (1)

Index	Event ...	Description	Call/Branch		Type	Timesta...	Triad A			Triad B		
			Source	Target			Counter A1	Counter A2	Counter A3	Counter B1	Counter B2	Counter B3
1-1	Unknown	Decoding error: unknown Nexus message type			Error	0						
8-1	SoC				Info	0						
10-1	SoC	Profiling address in (AsmSection)_0x40004220. Address = 0x400043a4.	(AsmSection...		Linear	16055						
16-1	SoC	Profiling address in __target_c_start. Address = 0x400058dc.	(AsmSection...	__target_c_start	Function Ret...	16238	239	0	0	0	0	0
22-1	SoC	Profiling address in __clear. Address = 0x40004216.	__target_c_start	__clear	Function Ret...	17067	300	0	0	0	0	0
28-1	SoC	Profiling address in __target_c_start. Address = 0x40005916.	__clear	__target_c_start	Function Ret...	17080	300	0	0	0	0	0
34-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	17277	308	0	0	0	0	0
40-1	SoC	Profiling address in __target_c_start. Address = 0x4000592e.	getCorerevR...	__target_c_start	Function Ret...	17458	311	0	0	0	0	0
46-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	17593	314	0	0	0	0	0
52-1	SoC	Profiling address in __target_c_start. Address = 0x40005a3c.	__target_c_start		Linear	18191						
58-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	18387	335	0	0	0	0	0
64-1	SoC	Profiling address in __target_c_start. Address = 0x40005a5a.	getCorerevR...	__target_c_start	Function Ret...	18571	339	0	0	0	0	0
70-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	18599	341	0	0	0	0	0
76-1	SoC	Profiling address in __target_c_start. Address = 0x40005a78.	getCorerevR...	__target_c_start	Function Ret...	18785	345	0	0	0	0	0
82-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	18849	347	0	0	0	0	0
88-1	SoC	Profiling address in __target_c_start. Address = 0x40005a96.	getCorerevR...	__target_c_start	Function Ret...	19073	353	0	0	0	0	0
94-1	SoC	Profiling address in getCorerevReg_0x40005880. Address = 0x40005884.	__target_c_start	getCorerevR...	Function Ret...	19143	355	0	0	0	0	0
100-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	getCorerevR...	__target_c_start	Function Ret...	21137	395	0	0	0	0	0
106-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__cmp_gt64		Linear	21330						
112-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	__cmp_gt64	__target_c_start	Function Ret...	22981	443	0	0	0	0	0
118-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__target_c_start	__cmp_gt64	Function Ret...	22986	443	0	0	0	0	0
124-1	SoC	Profiling address in __target_c_start. Address = 0x40005bf6.	__cmp_gt64	__target_c_start	Function Ret...	26046	514	0	0	0	0	0
130-1	SoC	Profiling address in __cmp_gt64. Address = 0x400054cc.	__target_c_start	__cmp_gt64	Function Ret...	26125	523	0	0	0	0	0
136-1	SoC	Profiling address in __target_c_start. Address = 0x40005c4c.	__cmp_gt64	__target_c_start	Function Ret...	27023	544	0	0	0	0	0
142-1	SoC	Profiling address in __cmpu_ge64. Address = 0x4000550c.	__target_c_start	__cmpu_ge64	Function Ret...	27145	547	0	0	0	0	0
148-1	SoC	Profiling address in __target_c_start. Address =	__cmpu_ge64	__target_c_start	Function Ret...	28309	574	0	0	0	0	0

Figure 3-72. Trace Data when both Triads are Selected

3.12 Multicore Tracing

This section explains the various actions performed on the multicore trace data.

Before debugging a multicore project and collecting trace data on the B4 target hardware, create and build a B4 project with Trace and Profile enabled.

NOTE

To successfully apply multicore trace configuration, specify or set the platform configuration (default one, a new one or one already created) for each of the launched configuration using Debug Configurations window. If launch configuration is not edited, then the platform configuration is not set. Hence, no trace data is collected for the core associated with that launch configuration.

This section includes the following:

- Debugging Multicore Project
- Collecting Multicore Trace Data
- Viewing multicore trace data
- Importing multicore trace data
- Trace commander view
- Multicore continuous trace

3.12.1 Debugging Multicore Project

This section explains how to debug a multicore trace project.

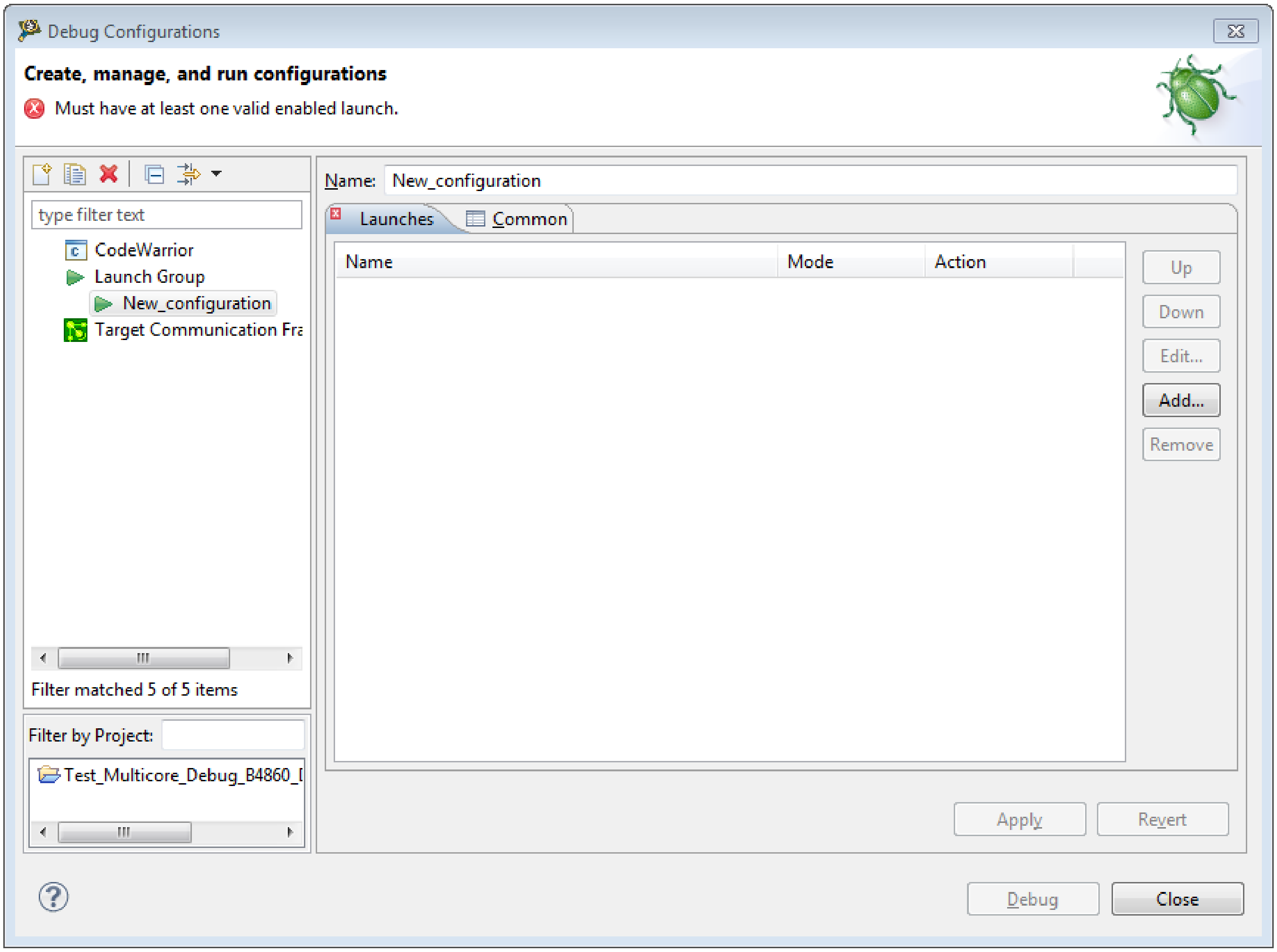
To debug a multicore project:

1. Right-click the B4860 project in the CodeWarrior Projects view and select **Debug As > Debug Configurations**.

   The **Debug Configurations** dialog box appears.

2. Double-click the Launch Group node in the tree structure on the left.

   The **New_configuration** node is added.



Debug Configurations

Create, manage, and run configurations

Must have at least one valid enabled launch.

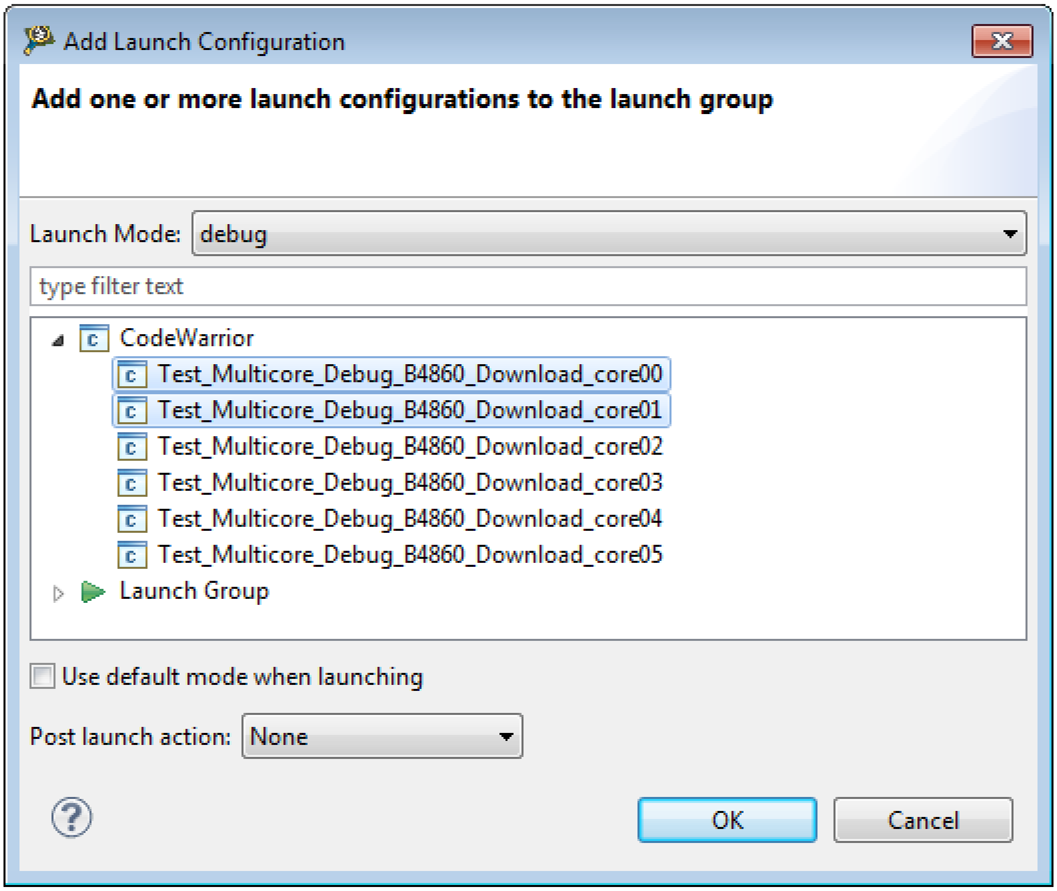
Name: New_configuration

Figure 3-73. Creating new launch group dialog box

3. Type a name for the new launch group in the **Name** text box, for example, Test_Multicore_TraceData_cores.
4. Click **Add**.

   The **Add Launch Configuration** dialog box appears.

   Expand the **CodeWarrior Download** node, and select the first two cores with `Shift` key pressed.



Add Launch Configuration

Add one or more launch configurations to the launch group

Launch Mode: debug

type filter text

- CodeWarrior
  - Test_Multicore_Debug_B4860_Download_core00
  - Test_Multicore_Debug_B4860_Download_core01
  - Test_Multicore_Debug_B4860_Download_core02
  - Test_Multicore_Debug_B4860_Download_core03
  - Test_Multicore_Debug_B4860_Download_core04
  - Test_Multicore_Debug_B4860_Download_core05
- Launch Group

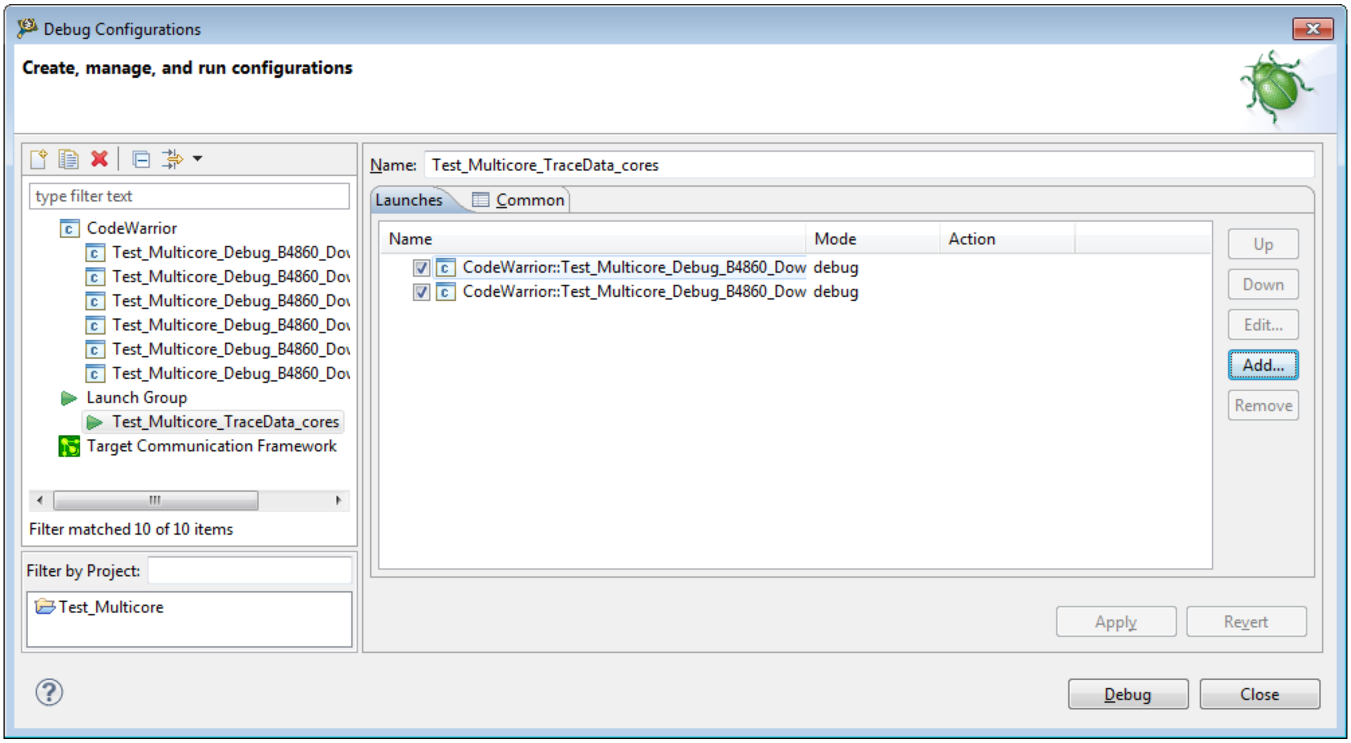
□ Use default mode when launching

Post launch action: None

Figure 3-74. Add launch configuration dialog box

5. Click **OK**.

   The cores get added to the new launch group.



Debug Configurations

Create, manage, and run configurations

type filter text

- CodeWarrior
  - Test_Multicore_Debug_B4860_Do\
  - Test_Multicore_Debug_B4860_Do\
  - Test_Multicore_Debug_B4860_Do\
  - Test_Multicore_Debug_B4860_Do\
  - Test_Multicore_Debug_B4860_Do\
  - Test_Multicore_Debug_B4860_Do\
- Launch Group
  - Test_Multicore_TraceData_cores
- Target Communication Framework

Filter matched 10 of 10 items

Filter by Project:

Test_Multicore

Name: Test_Multicore_TraceData_cores

Launches | Common

Name	Mode	Action
☑ CodeWarrior::Test_Multicore_Debug_B4860_Dow	debug	
☑ CodeWarrior::Test_Multicore_Debug_B4860_Dow	debug	

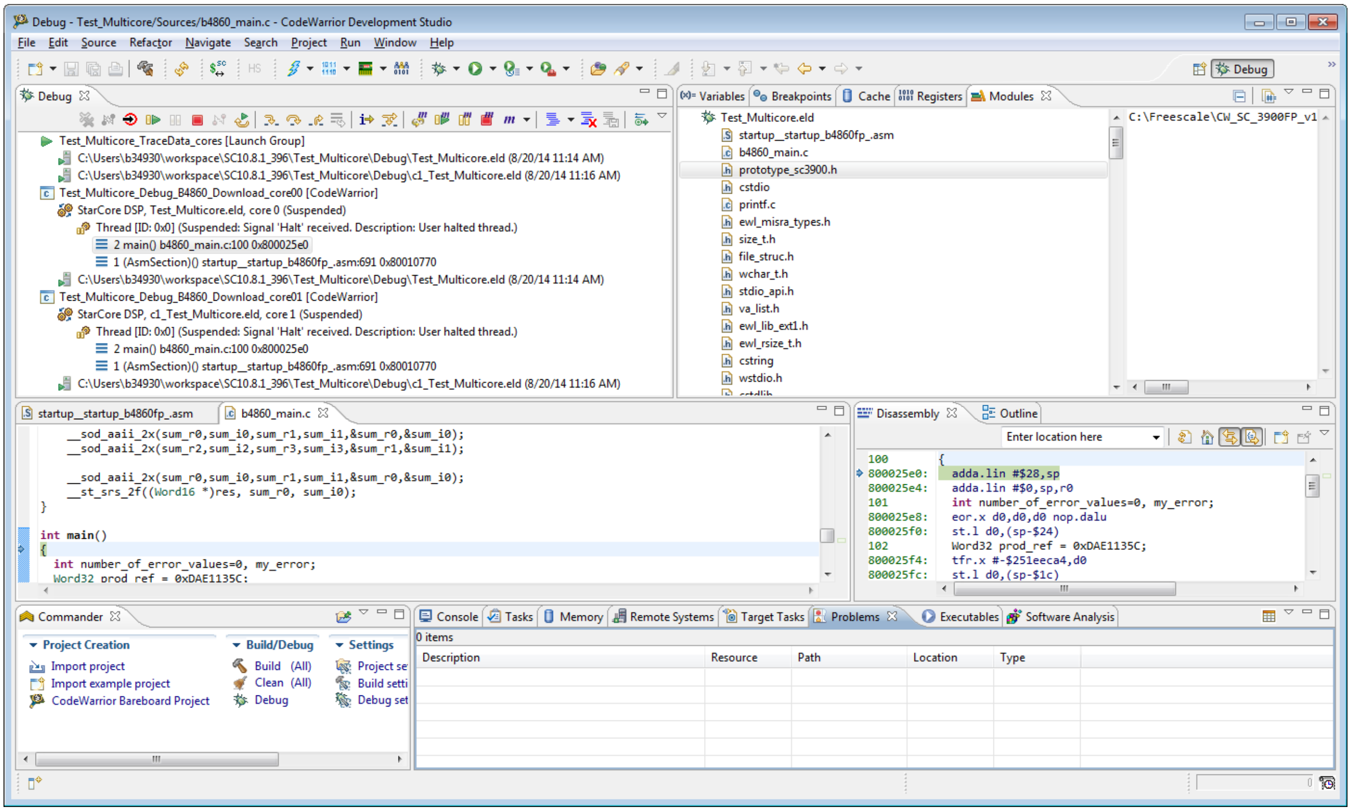
Figure 3-75. Cores Added to New Launch Group

NOTE

The cores selected to be part of the launch group should be configured before debugging the launch group.

6. Click **Apply** to save the settings.

   Click **Debug** to start the debug session. The Debug perspective appears and the execution halts at the first statement of `main()`.



```
Test_Multicore_TraceData_cores [Launch Group]
  C:\Users\b34930\workspace\SC10.8.1_396\Test_Multicore\Debug\Test_Multicore.eld (8/20/14 11:14 AM)
  C:\Users\b34930\workspace\SC10.8.1_396\Test_Multicore\Debug\c1_Test_Multicore.eld (8/20/14 11:16 AM)
Test_Multicore_Debug_B4860_Download_core00 [CodeWarrior]
  StarCore DSP, Test_Multicore.eld, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() b4860_main.c:100 0x800025e0
      1 (AsmSection)() startup__startup_b4860fp_.asm:691 0x80010770
  C:\Users\b34930\workspace\SC10.8.1_396\Test_Multicore\Debug\Test_Multicore.eld (8/20/14 11:14 AM)
Test_Multicore_Debug_B4860_Download_core01 [CodeWarrior]
  StarCore DSP, c1_Test_Multicore.eld, core 1 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() b4860_main.c:100 0x800025e0
      1 (AsmSection)() startup__startup_b4860fp_.asm:691 0x80010770
  C:\Users\b34930\workspace\SC10.8.1_396\Test_Multicore\Debug\c1_Test_Multicore.eld (8/20/14 11:16 AM)
```

```
    __sod_aaii_2x(sum_r0,sum_i0,sum_r1,sum_i1,&sum_r0,&sum_i0);
    __sod_aaii_2x(sum_r2,sum_i2,sum_r3,sum_i3,&sum_r1,&sum_i1);

    __sod_aaii_2x(sum_r0,sum_i0,sum_r1,sum_i1,&sum_r0,&sum_i0);
    __st_srs_2f((Word16 *)res, sum_r0, sum_i0);
}

int main()
{
  int number_of_error_values=0, my_error;
  Word32 prod_ref = 0xDAE1135C;
```

```
100         {
800025e0:     adda.lin #$28,sp
800025e4:     adda.lin #$0,sp,r0
101           int number_of_error_values=0, my_error;
800025e8:     eor.x d0,d0,d0 nop.dalu
800025f0:     st.l d0,(sp-$24)
102           Word32 prod_ref = 0xDAE1135C;
800025f4:     tfr.x #-$251eeca4,d0
800025fc:     st.l d0,(sp-$1c)
```

0 items

Description	Resource	Path	Location	Type

Figure 3-76. Debug Perspective Window

3.12.2 Collecting Multicore Trace Data

This section explains the ways to collect multicore trace data.

Follow the steps to collect multicore trace data:

1. In the Debug view, click **Multicore Resume** . The execution begins and data measurement starts.

   Wait for some time.

2. When output starts displaying in the Console view for all the two cores, click **Multicore Terminate** .

3.12.3 Viewing multicore trace data

The trace data is collected in the Software Analysis view. Click on the respective hyperlinks to view the collected data.

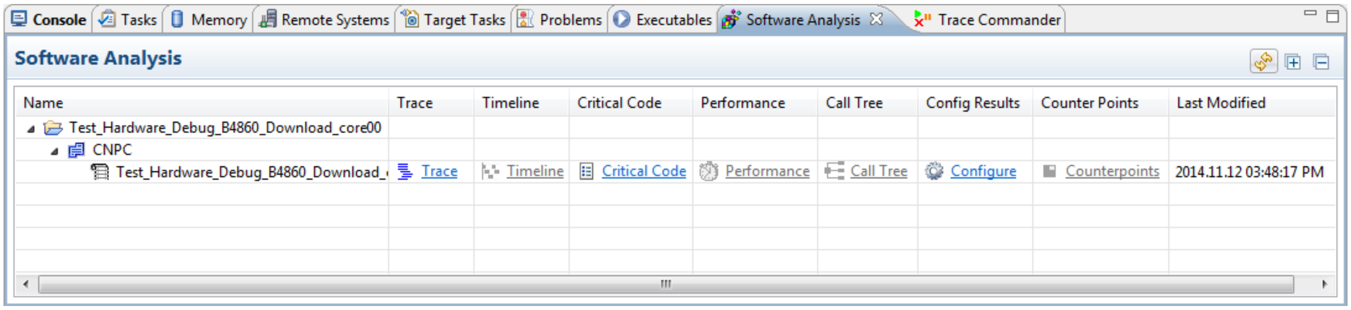
NOTE

CriticalCode link is available only when all cores that are active in a debug session are configured with Program Trace scenario. Whereas, Timeline, Performance, and Call Tree links are available only when cores are configured with Profiling scenario.

NOTE

If the launch group contains cores configured with both Profiling and Program Trace, then only the Trace link will be available in Software Analysis view.

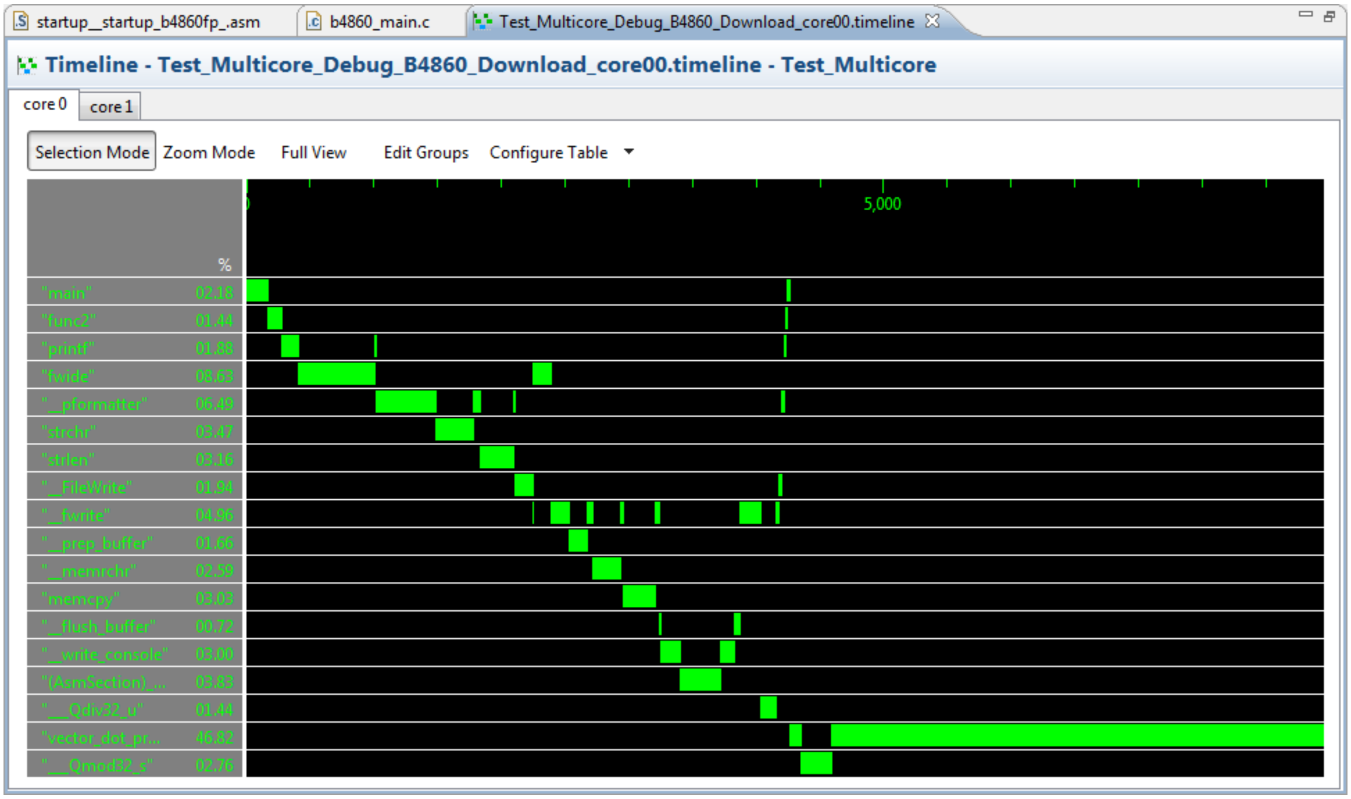
The following figure shows an example of Software Analysis view:



Name	Trace	Timeline	Critical Code	Performance	Call Tree	Config Results	Counter Points	Last Modified
Test_Hardware_Debug_B4860_Download_core00								
CNPC								
Test_Hardware_Debug_B4860_Download_	Trace	Timeline	Critical Code	Performance	Call Tree	Configure	Counterpoints	2014.11.12 03:48:17 PM

Figure 3-77. Software Analysis View

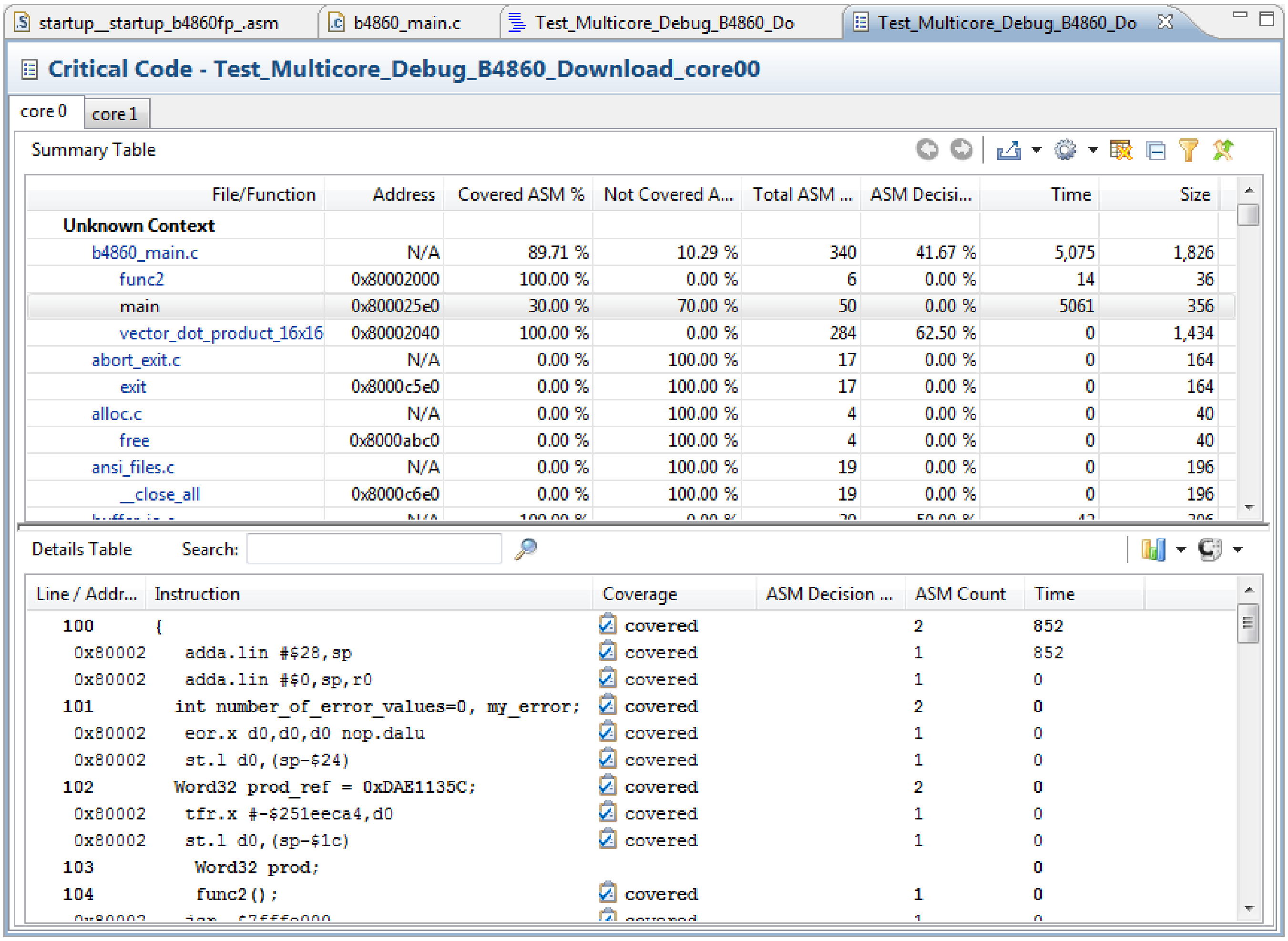
Click on the Timeline link to view the timeline data.


startup__startup_b4860fp_.asm
b4860_main.c
Test_Multicore_Debug_B4860_Download_core00.timeline
Timeline - Test_Multicore_Debug_B4860_Download_core00.timeline - Test_Multicore
core 0
core 1
Selection Mode
Zoom Mode
Full View
Edit Groups
Configure Table
0
5,000

	%
"main"	02.18
"func2"	01.44
"printf"	01.88
"fwide"	08.63
"__pformatter"	06.49
"strchr"	03.47
"strlen"	03.16
"__FileWrite"	01.94
"__fwrite"	04.96
"__prep_buffer"	01.66
"__memrchr"	02.59
"memcpy"	03.03
"__flush_buffer"	00.72
"__write_console"	03.00
"(AsmSection)_...	03.83
"___Qdiv32_u"	01.44
"vector_dot_pr...	46.82
"___Qmod32_s"	02.76

Figure 3-78. Timeline View

Click on the Critical Code link to view the critical code data.



startup__startup_b4860fp_.asm
b4860_main.c
Test_Multicore_Debug_B4860_Do
Test_Multicore_Debug_B4860_Do

Critical Code - Test_Multicore_Debug_B4860_Download_core00

Summary Table

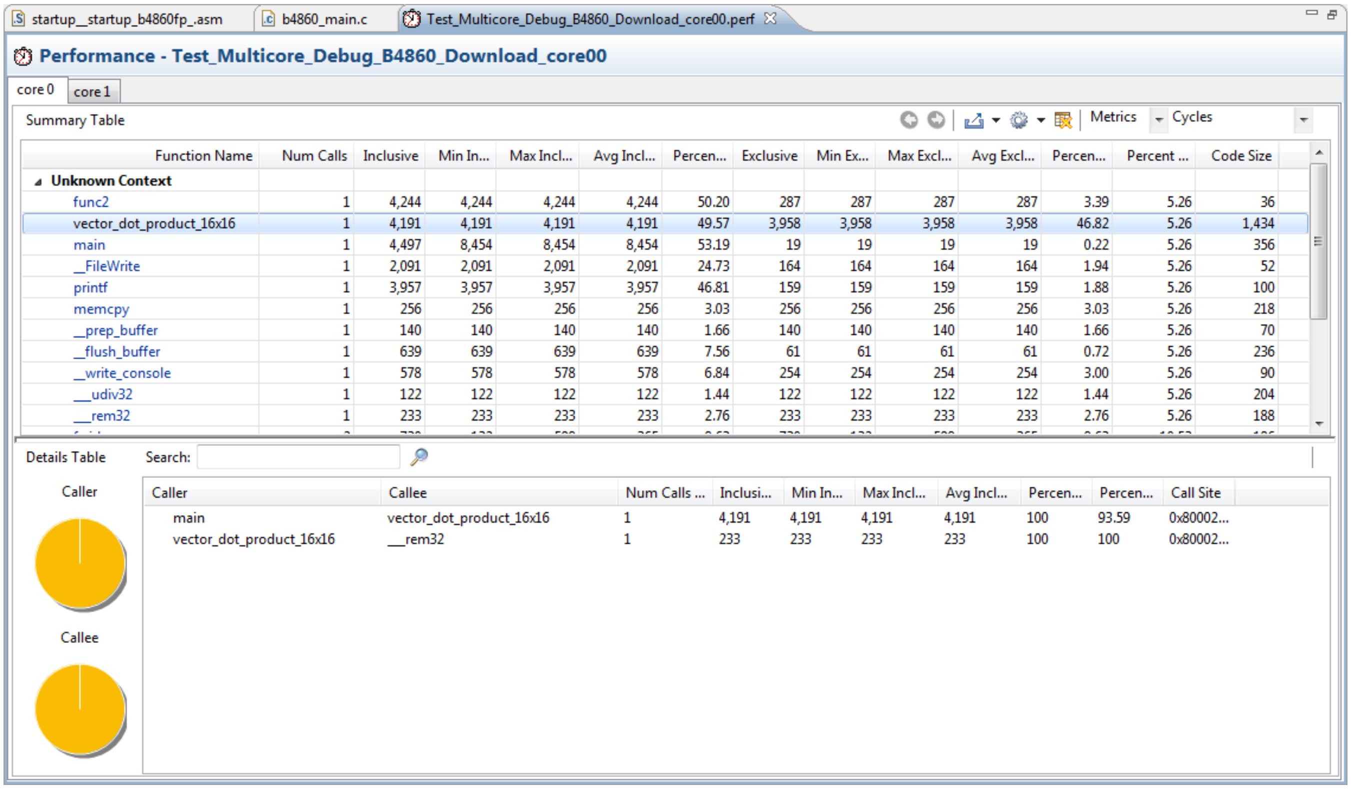
File/Function	Address	Covered ASM %	Not Covered A...	Total ASM ...	ASM Decisi...	Time	Size
Unknown Context							
b4860_main.c	N/A	89.71 %	10.29 %	340	41.67 %	5,075	1,826
func2	0x80002000	100.00 %	0.00 %	6	0.00 %	14	36
main	0x800025e0	30.00 %	70.00 %	50	0.00 %	5061	356
vector_dot_product_16x16	0x80002040	100.00 %	0.00 %	284	62.50 %	0	1,434
abort_exit.c	N/A	0.00 %	100.00 %	17	0.00 %	0	164
exit	0x8000c5e0	0.00 %	100.00 %	17	0.00 %	0	164
alloc.c	N/A	0.00 %	100.00 %	4	0.00 %	0	40
free	0x8000abc0	0.00 %	100.00 %	4	0.00 %	0	40
ansi_files.c	N/A	0.00 %	100.00 %	19	0.00 %	0	196
__close_all	0x8000c6e0	0.00 %	100.00 %	19	0.00 %	0	196

Details Table Search:

Line / Addr...	Instruction	Coverage	ASM Decision ...	ASM Count	Time
100	{	☑ covered		2	852
0x80002	adda.lin #$28,sp	☑ covered		1	852
0x80002	adda.lin #$0,sp,r0	☑ covered		1	0
101	int number_of_error_values=0, my_error;	☑ covered		2	0
0x80002	eor.x d0,d0,d0 nop.dalu	☑ covered		1	0
0x80002	st.l d0,(sp-$24)	☑ covered		1	0
102	Word32 prod_ref = 0xDAE1135C;	☑ covered		2	0
0x80002	tfr.x #-$251eeca4,d0	☑ covered		1	0
0x80002	st.l d0,(sp-$1c)	☑ covered		1	0
103	Word32 prod;				0
104	func2();	☑ covered		1	0

Figure 3-79. Critical Code View

Click on the Performance link to view the Performance data.



startup__startup_b4860fp_.asm
b4860_main.c
Test_Multicore_Debug_B4860_Download_core00.perf

Performance - Test_Multicore_Debug_B4860_Download_core00

core 0 | core 1

Summary Table

Function Name	Num Calls	Inclusive	Min In...	Max Incl...	Avg Incl...	Percen...	Exclusive	Min Ex...	Max Excl...	Avg Excl...	Percen...	Percent ...	Code Size
Unknown Context													
func2	1	4,244	4,244	4,244	4,244	50.20	287	287	287	287	3.39	5.26	36
vector_dot_product_16x16	1	4,191	4,191	4,191	4,191	49.57	3,958	3,958	3,958	3,958	46.82	5.26	1,434
main	1	4,497	8,454	8,454	8,454	53.19	19	19	19	19	0.22	5.26	356
__FileWrite	1	2,091	2,091	2,091	2,091	24.73	164	164	164	164	1.94	5.26	52
printf	1	3,957	3,957	3,957	3,957	46.81	159	159	159	159	1.88	5.26	100
memcpy	1	256	256	256	256	3.03	256	256	256	256	3.03	5.26	218
__prep_buffer	1	140	140	140	140	1.66	140	140	140	140	1.66	5.26	70
__flush_buffer	1	639	639	639	639	7.56	61	61	61	61	0.72	5.26	236
__write_console	1	578	578	578	578	6.84	254	254	254	254	3.00	5.26	90
___udiv32	1	122	122	122	122	1.44	122	122	122	122	1.44	5.26	204
___rem32	1	233	233	233	233	2.76	233	233	233	233	2.76	5.26	188

Details Table

Search:

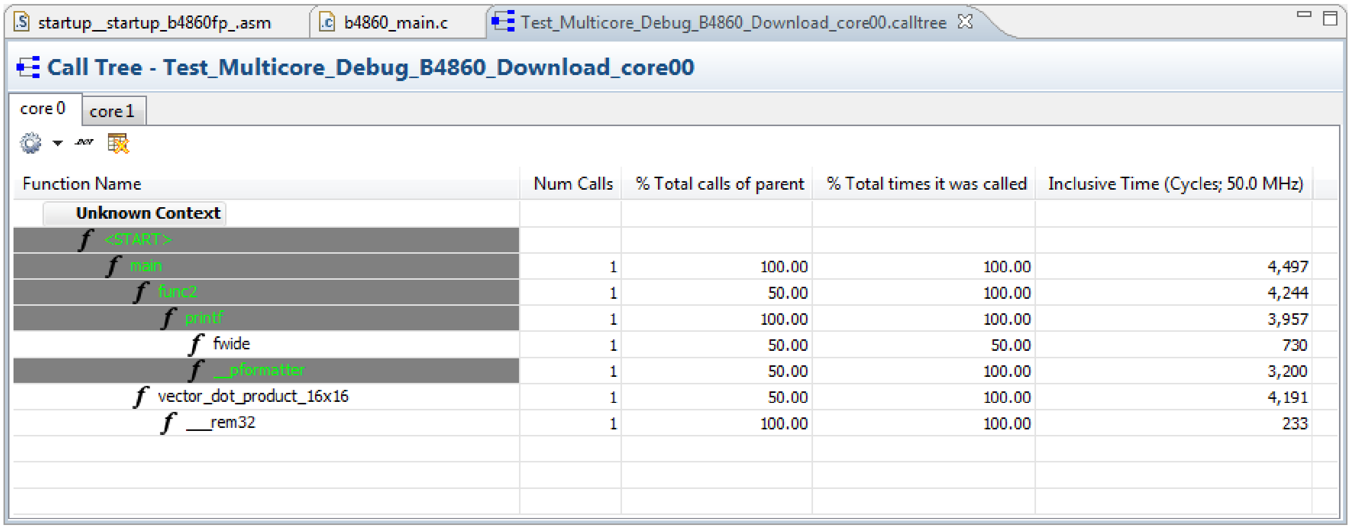
Caller	Callee	Num Calls ...	Inclusi...	Min In...	Max Incl...	Avg Incl...	Percen...	Percen...	Call Site
main	vector_dot_product_16x16	1	4,191	4,191	4,191	4,191	100	93.59	0x80002...
vector_dot_product_16x16	__rem32	1	233	233	233	233	100	100	0x80002...

Figure 3-80. Performance View

NOTE

The caller-callee pie chart will move according to the function pressed in details table and exclude symbols will exclude the selected symbols for all cores.

Click the Call Tree link to view the call tree data.



startup__startup_b4860fp_.asm
b4860_main.c
Test_Multicore_Debug_B4860_Download_core00.calltree

Call Tree - Test_Multicore_Debug_B4860_Download_core00

core 0 | core 1

Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles; 50.0 MHz)
Unknown Context				
<START>				
main	1	100.00	100.00	4,497
func2	1	50.00	100.00	4,244
printf	1	100.00	100.00	3,957
fwide	1	50.00	50.00	730
__pformatter	1	50.00	100.00	3,200
vector_dot_product_16x16	1	50.00	100.00	4,191
__rem32	1	100.00	100.00	233

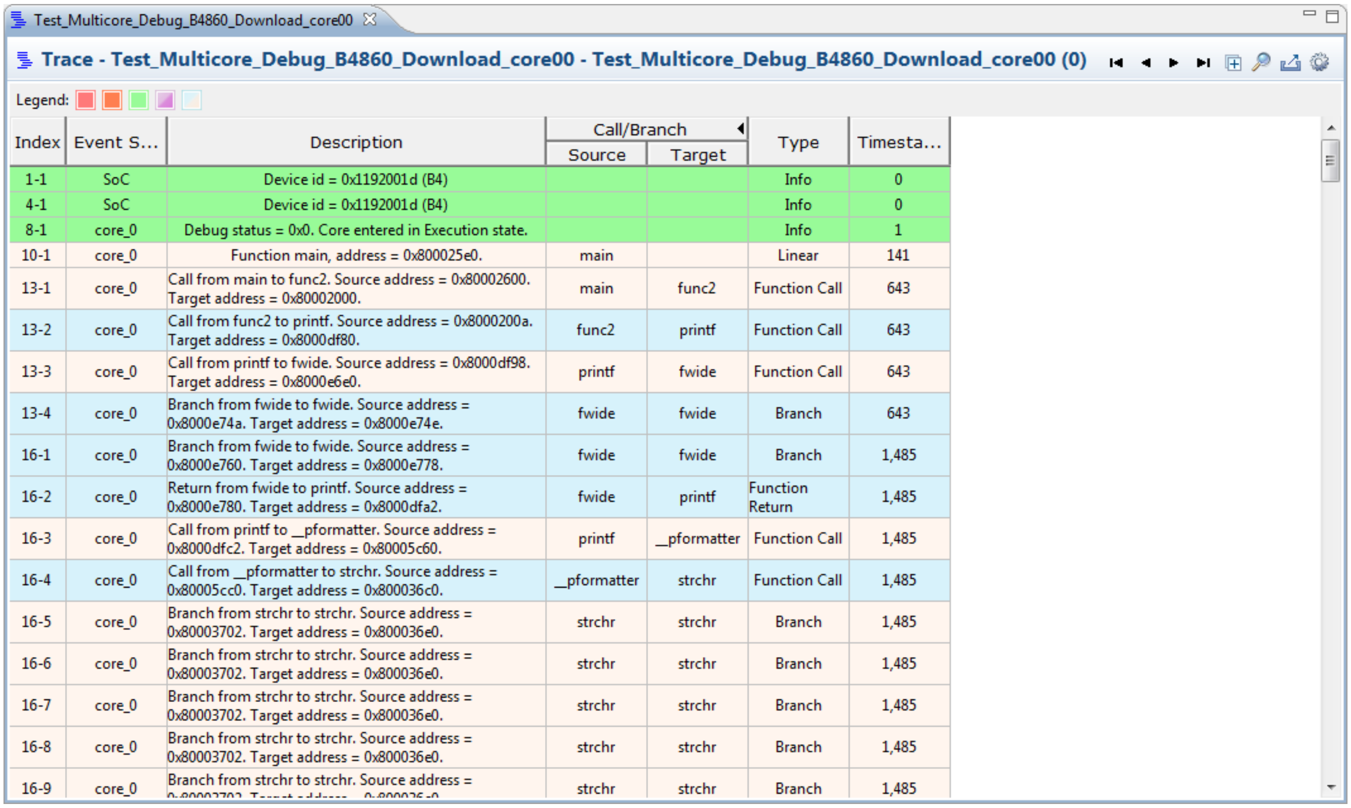
Figure 3-81. Call Tree View

3.12.4 Importing multicore trace data

This feature allows you to import the trace data collected for multicore B4 hardware project.

To import the multicore trace data, perform the following steps:

1. Collect the multicore trace data.
2. Switch to a new workspace.
3. Select **File > Import** to open the Import wizard.
4. Expand the **Software Analysis** node and select **Trace** option.
5. Click **Next** to display the **Import Trace** page of the **Import** wizard.
6. Click **Browse** and locate the trace data file of the project that you want to import into your project. The trace data file is located in the **.Analysis Data** folder of the project in the workspace.
7. Select either **Existing project** or **New project** from the **Import To** list, depending upon the requirements of your project.
8. Click **Next** to display the **Import Trace to Existing Project** page.
9. Type or select the project in which you want to import the trace data. Type a new file name for the trace data file in the **New trace file name** will be text box. This field is optional.
10. Click **Next** to display the **Import Trace Configuration** page.
11. Select the **System** corresponding to the trace that will be imported.
12. Specify the **Source Code Correlation** details for the cores on which trace was generated, for example SC3900-0 and SC3900-1.
13. Specify the **Application** or launch configuration and the respective **Trace type** for each core in the **Source Code Correlation** dialog box.
14. Click **Finish**.
15. The trace viewer will display the imported multicore data.



Test_Multicore_Debug_B4860_Download_core00

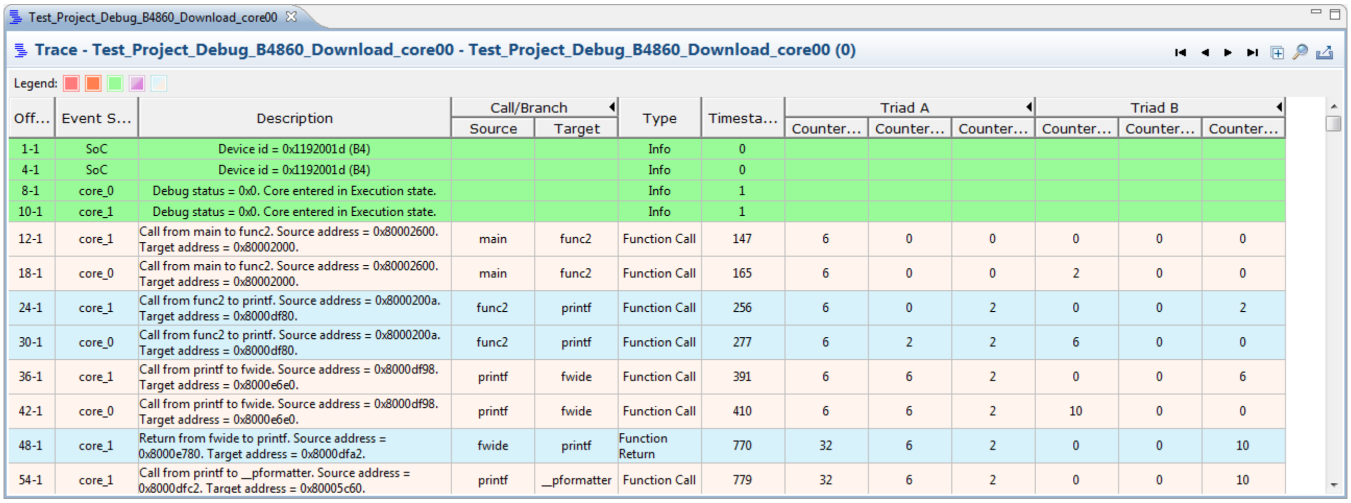
Trace - Test_Multicore_Debug_B4860_Download_core00 - Test_Multicore_Debug_B4860_Download_core00 (0)

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
1-1	SoC	Device id = 0x1192001d (B4)			Info	0
4-1	SoC	Device id = 0x1192001d (B4)			Info	0
8-1	core_0	Debug status = 0x0. Core entered in Execution state.			Info	1
10-1	core_0	Function main, address = 0x800025e0.	main		Linear	141
13-1	core_0	Call from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Function Call	643
13-2	core_0	Call from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Function Call	643
13-3	core_0	Call from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Function Call	643
13-4	core_0	Branch from fwide to fwide. Source address = 0x8000e74a. Target address = 0x8000e74e.	fwide	fwide	Branch	643
16-1	core_0	Branch from fwide to fwide. Source address = 0x8000e760. Target address = 0x8000e778.	fwide	fwide	Branch	1,485
16-2	core_0	Return from fwide to printf. Source address = 0x8000e780. Target address = 0x8000dfa2.	fwide	printf	Function Return	1,485
16-3	core_0	Call from printf to _pformatter. Source address = 0x8000dfc2. Target address = 0x80005c60.	printf	_pformatter	Function Call	1,485
16-4	core_0	Call from _pformatter to strchr. Source address = 0x80005cc0. Target address = 0x800036c0.	_pformatter	strchr	Function Call	1,485
16-5	core_0	Branch from strchr to strchr. Source address = 0x80003702. Target address = 0x800036e0.	strchr	strchr	Branch	1,485
16-6	core_0	Branch from strchr to strchr. Source address = 0x80003702. Target address = 0x800036e0.	strchr	strchr	Branch	1,485
16-7	core_0	Branch from strchr to strchr. Source address = 0x80003702. Target address = 0x800036e0.	strchr	strchr	Branch	1,485
16-8	core_0	Branch from strchr to strchr. Source address = 0x80003702. Target address = 0x800036e0.	strchr	strchr	Branch	1,485
16-9	core_0	Branch from strchr to strchr. Source address =	strchr	strchr	Branch	1,485

Figure 3-82. Imported multicore trace data

The following image shows the imported multicore trace data with profiling traids.



Test_Project_Debug_B4860_Download_core00

Trace - Test_Project_Debug_B4860_Download_core00 - Test_Project_Debug_B4860_Download_core00 (0)

Legend: 🟥 🟧 🟩 🟪 ⬜

Off...	Event S...	Description	Call/Branch		Type	Timesta...	Triad A			Triad B		
			Source	Target			Counter...	Counter...	Counter...	Counter...	Counter...	Counter...
1-1	SoC	Device id = 0x1192001d (B4)			Info	0						
4-1	SoC	Device id = 0x1192001d (B4)			Info	0						
8-1	core_0	Debug status = 0x0. Core entered in Execution state.			Info	1						
10-1	core_1	Debug status = 0x0. Core entered in Execution state.			Info	1						
12-1	core_1	Call from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Function Call	147	6	0	0	0	0	0
18-1	core_0	Call from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Function Call	165	6	0	0	2	0	0
24-1	core_1	Call from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Function Call	256	6	0	2	0	0	2
30-1	core_0	Call from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Function Call	277	6	2	2	6	0	0
36-1	core_1	Call from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Function Call	391	6	6	2	0	0	6
42-1	core_0	Call from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Function Call	410	6	6	2	10	0	0
48-1	core_1	Return from fwide to printf. Source address = 0x8000e780. Target address = 0x8000dfa2.	fwide	printf	Function Return	770	32	6	2	0	0	10
54-1	core_1	Call from printf to _pformatter. Source address = 0x8000dfc2. Target address = 0x80005c60.	printf	_pformatter	Function Call	779	32	6	2	0	0	10

Figure 3-83. Imported multicore trace data - Traid counter values

3.12.5 Trace commander view

Trace commander view is used to manage the multi-core trace collection data. The view is used to perform actions such as, starting or stopping the tracing on different cores, manual upload, trace configure or changing the master core of the collected trace data.

Trace commander displays current trace state for each core.

Perform the following steps to manage the multi-core trace data using a trace commander:

1. Create a multi-core project.
2. Right-click the project and select **Debug As > Debug Configurations**.

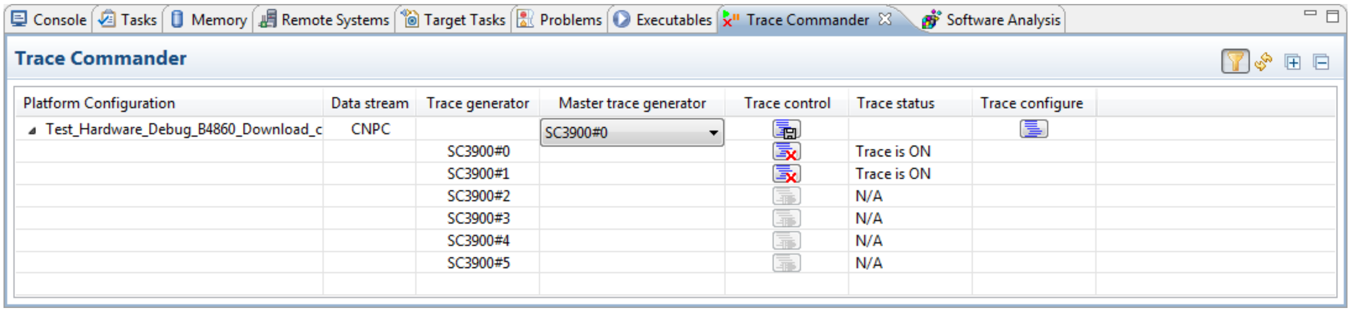
   The **Debug Configurations** dialog box appears.

3. Create a **New_Launch** group by adding the cores on which trace needs to be collected, for example core 00 and core 01.
4. Click **OK**.

   The cores get added to the new launch group.

5. Set the **Trace control** settings for the selected cores on which multi-core debugging needs to be performed. The trace control settings are available on the **Intermediate** page of **Trace and Profile** tab.
6. Click **Apply**.
7. Debug the **New_configuration** group to collect the trace data.
8. Select **Window > Show View > Other > Software Analysis > Trace Commander**.

   The **Trace Commander view** is displayed.



Trace Commander

Platform Configuration	Data stream	Trace generator	Master trace generator	Trace control	Trace status	Trace configure
Test_Hardware_Debug_B4860_Download_c	CNPC		SC3900#0			
		SC3900#0			Trace is ON	
		SC3900#1			Trace is ON	
		SC3900#2			N/A	
		SC3900#3			N/A	
		SC3900#4			N/A	
		SC3900#5			N/A	

Figure 3-84. Trace Commander View

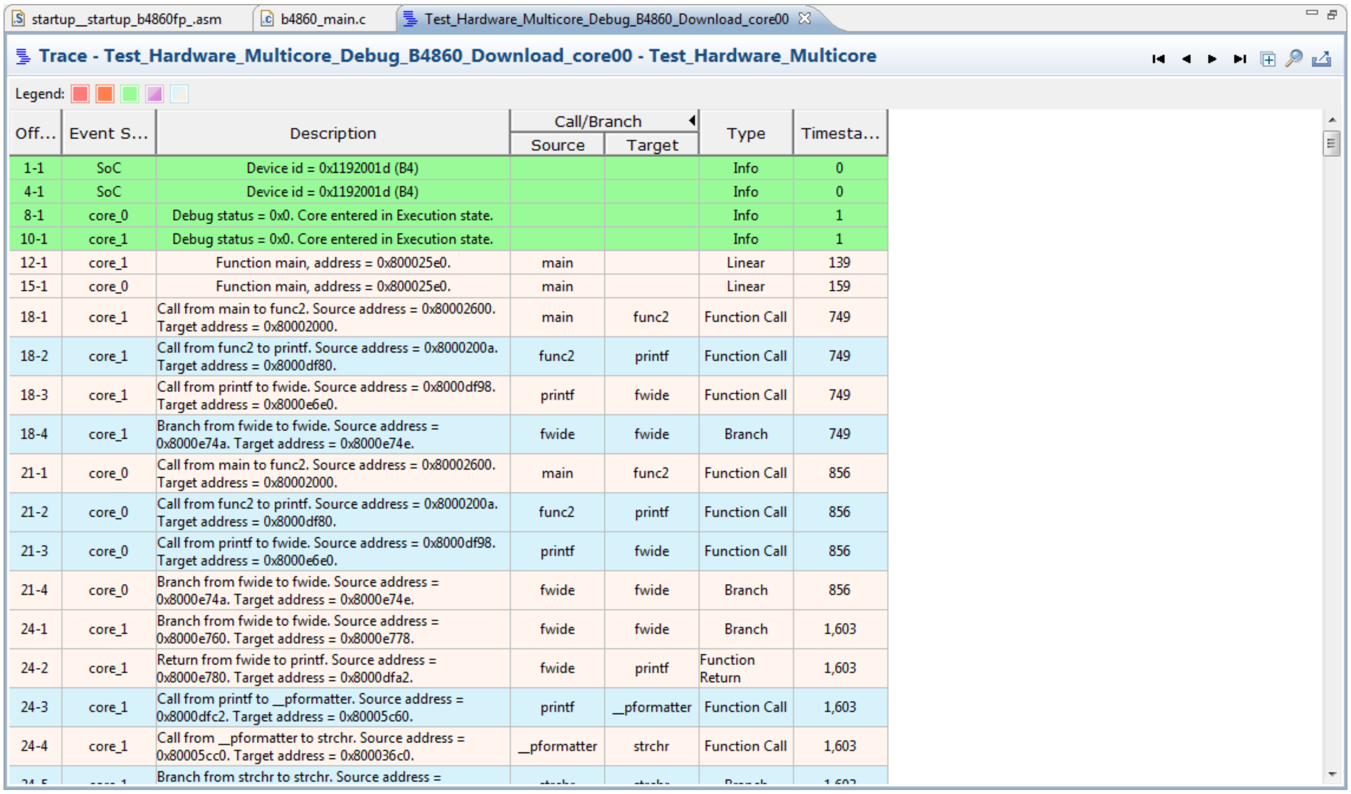
The following table lists the description of each column available in the **Trace commander** view:

.

Table 3-18. Trace Commander view settings

Option	Description
Platform Configuration	Shows the platform configuration file used by projects.
Data stream	Shows the current data stream selected in configurator.
Trace generator	Displays the current cores that are available for collecting trace.
Master trace generator	Displays the current core selected as the master core and allows changing the master core.
Trace control	Allows you to start/stop and upload the trace data. The **Upload** button is enabled only when trace is available for collection.
Trace status	Displays the current trace status. `N/A' is applicable when cores are not in the debug state. `Trace stop' or `Trace start' appears when cores are in the debug state, depending on the trace state for that core.
Trace configure	Allows you to configure the trace settings. **NOTE:** If trace configuration settings are applied manually, then after reconfiguring the trace using **Trace configure** button, Start/ Stop trace button corresponding to master core will be reset to the **Start – Trace is OFF** state and the Upload button will be disabled if it was previously enabled.

9. Click the **Trace** link available in the **Software Analysis** view to view the trace collected on the respective cores.



startup__startup_b4860fp_.asm | b4860_main.c | Test_Hardware_Multicore_Debug_B4860_Download_core00

Trace - Test_Hardware_Multicore_Debug_B4860_Download_core00 - Test_Hardware_Multicore

Legend: 🟥 🟧 🟩 🟪 ⬜

Off...	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
1-1	SoC	Device id = 0x1192001d (B4)			Info	0
4-1	SoC	Device id = 0x1192001d (B4)			Info	0
8-1	core_0	Debug status = 0x0. Core entered in Execution state.			Info	1
10-1	core_1	Debug status = 0x0. Core entered in Execution state.			Info	1
12-1	core_1	Function main, address = 0x800025e0.	main		Linear	139
15-1	core_0	Function main, address = 0x800025e0.	main		Linear	159
18-1	core_1	Call from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Function Call	749
18-2	core_1	Call from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Function Call	749
18-3	core_1	Call from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Function Call	749
18-4	core_1	Branch from fwide to fwide. Source address = 0x8000e74a. Target address = 0x8000e74e.	fwide	fwide	Branch	749
21-1	core_0	Call from main to func2. Source address = 0x80002600. Target address = 0x80002000.	main	func2	Function Call	856
21-2	core_0	Call from func2 to printf. Source address = 0x8000200a. Target address = 0x8000df80.	func2	printf	Function Call	856
21-3	core_0	Call from printf to fwide. Source address = 0x8000df98. Target address = 0x8000e6e0.	printf	fwide	Function Call	856
21-4	core_0	Branch from fwide to fwide. Source address = 0x8000e74a. Target address = 0x8000e74e.	fwide	fwide	Branch	856
24-1	core_1	Branch from fwide to fwide. Source address = 0x8000e760. Target address = 0x8000e778.	fwide	fwide	Branch	1,603
24-2	core_1	Return from fwide to printf. Source address = 0x8000e780. Target address = 0x8000dfa2.	fwide	printf	Function Return	1,603
24-3	core_1	Call from printf to __pformatter. Source address = 0x8000dfc2. Target address = 0x80005c60.	printf	__pformatter	Function Call	1,603
24-4	core_1	Call from __pformatter to strchr. Source address = 0x80005cc0. Target address = 0x800036c0.	__pformatter	strchr	Function Call	1,603
24-5	core_1	Branch from strchr to strchr. Source address =	strchr	strchr	Branch	1,603

Figure 3-85. Trace Data Collected

The trace viewer displays the trace collected for core00 and core01.

3.12.6 Multicore continuous trace

Multicore continuous trace stands for trace collected from multiple cores without trace being lost. For the B4860 targets, trace might be lost for two main reasons:

- Trace buffer size is not enough to accommodate all trace generated by the multicore trace session.
- Trace messages frequency/rate, even if trace buffer size is enough to accommodate all trace generated by the multicore trace session.

In order to avoid trace messages lost in any of the above cases trace collection mode should be set to continuous.

To collect the profiling trace in the multicore continous trace mode, follow the steps listed below:

1. Create a project using b4860 hardware target.
2. Use source `multicoreContinousTrace.c` with:
   - - undefine `FULL_PROGRAM_TRACE`
   - - set `TRACE_MES_FREQ to TRACE_MES_FREQ_HIGH` within `#ifdef PROFILING_TRACE` block.

3. Select Profiling – L2 cache events trace scenario for all cores.
4. Select DDR trace buffer size to 0x60000 and one buffer as trace collection mode for master core.
5. Start multicore debug and wait until application reaches its end. Multicore terminate.
6. Open Performance to fully decode trace and store decoded trace.
7. Open **Trace viewer** and press **Ctrl+F** to find ***error***.
8. Inspect Performance viewer.
9. Select DDR trace buffer size to 0x60000 and continuous as trace collection mode for master core.
10. Start multicore debug and wait until application reaches its end. Multicore terminate.
11. Open Performance to fully decode trace and store decoded trace.
12. Open **Trace viewer** and press **Ctrl+F** to find ***error***.
13. Inspect Performance viewer.
14. Redo steps 10 to 13 with:
    - DDR trace buffer size set to 0x10000 and continuous as trace collection mode for master core.
15. Redo steps 10 to 13 with:
    - DDR trace buffer size set to 0x60000 and one buffer as trace collection mode for master core.
    - TRACE_MES_FREQ set to TRACE_MES_FREQ_MEDIUM within #ifdef PROFILING_TRACE block. Rebuild the project.
16. Redo steps 10 to 13 with:
    - DDR trace buffer size set to 0x10000 and continuous as trace collection mode for master core.
    - TRACE_MES_FREQ set to TRACE_MES_FREQ_MEDIUM within #ifdef PROFILING_TRACE block. Rebuild the project.

Chapter 4
Setting Tracepoints

Tracepoint is a point in the target program where start or stop tracepoints are set at a line of the source code or the assembly code. The start and stop tracepoints are triggers for enabling and disabling the trace output. The advantage of setting start and stop tracepoints is to capture the trace data from the specific part of the program. This solves the problem of tracing a large application. A full trace would be extremely difficult to follow and would also require a large amount of target memory to store it. The trace data displays the trace result based on the tracepoints. If there is a stop tracepoint and no start tracepoint, the trace automatically starts at the beginning of the application. If there is at least one start tracepoint, the trace is automatically disabled at the beginning of the application. This chapter explains how to set start and stop tracepoints and also how to enable and disable a tracepoint.

This chapter explains:

- Setting Start and Stop Tracepoints
- Enabling and Disabling the Tracepoints
- Limitation of hardware tracepoints
- Multicore hardware tracepoints

4.1 Setting Start and Stop Tracepoints

This section explains how to start and stop tracepoints.

The tracepoint can be set in the Editor area and the **Disassembly** view. The source line tracepoint is set in the Editor area and the address tracepoint is set in the **Disassembly** view. You can see the tracepoints in the Editor area as well as in the **Disassembly** view.

NOTE

It is recommended that start and stop tracepoints should be set in the same function so that the trace that is collected is meaningful.

This topic explains how to start and stop tracepoints using the following:

- Simulator
- Target Hardware
- Viewing Tracepoints in Analysispoints

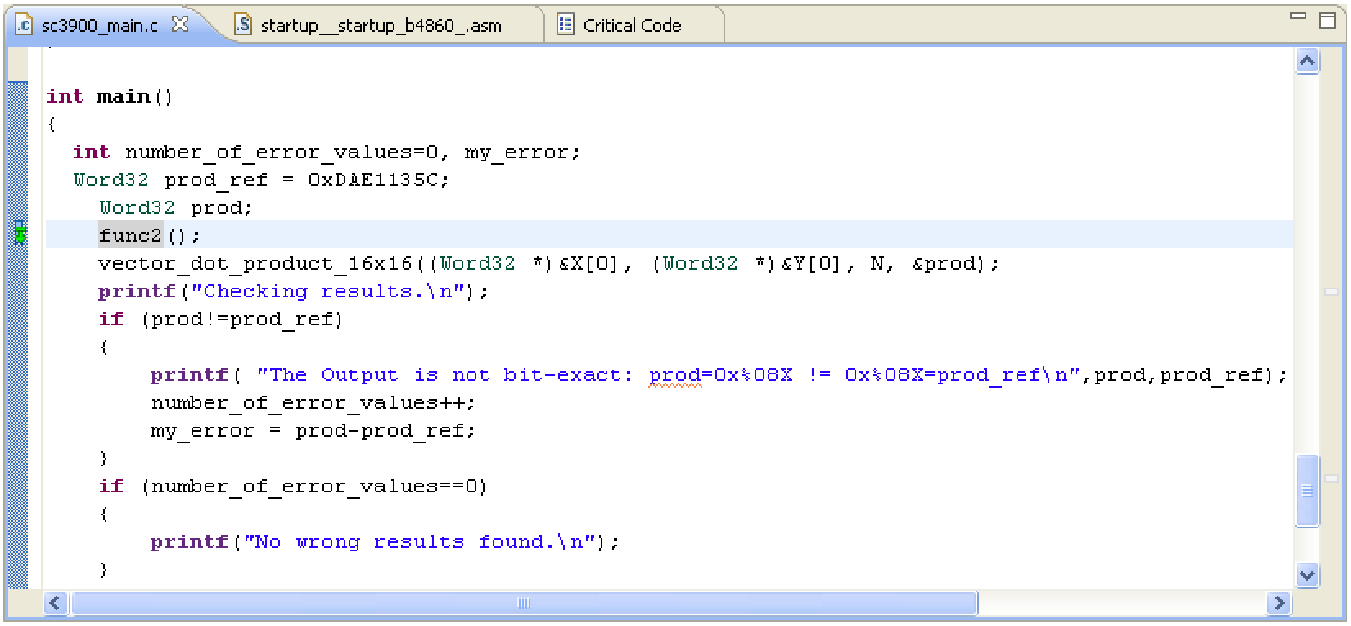
4.1.1 Simulator

This section explains how to set tracepoints in source code of a simulator project.

Before setting the tracepoints, refer to the topic Collecting Data for the procedure on how to collect the trace and profiling data using SC3900 simulator.

Perform the following steps to set the tracepoints in source code:

1. In the **CodeWarrior Projects** view, select the **Source** folder under the project.
2. Double-click on the source file, for example, sc3900_main.c. The contents of the file can be viewed on the right-side of the pane.
3. Select the line on which you want to set the start tracepoint.
4. Right-click on the marker bar, and select **Toggle Trace Start Point> Hardware Trace Point** option from the context menu as shown below. The same option is also used to remove the start tracepoint from the marker bar.



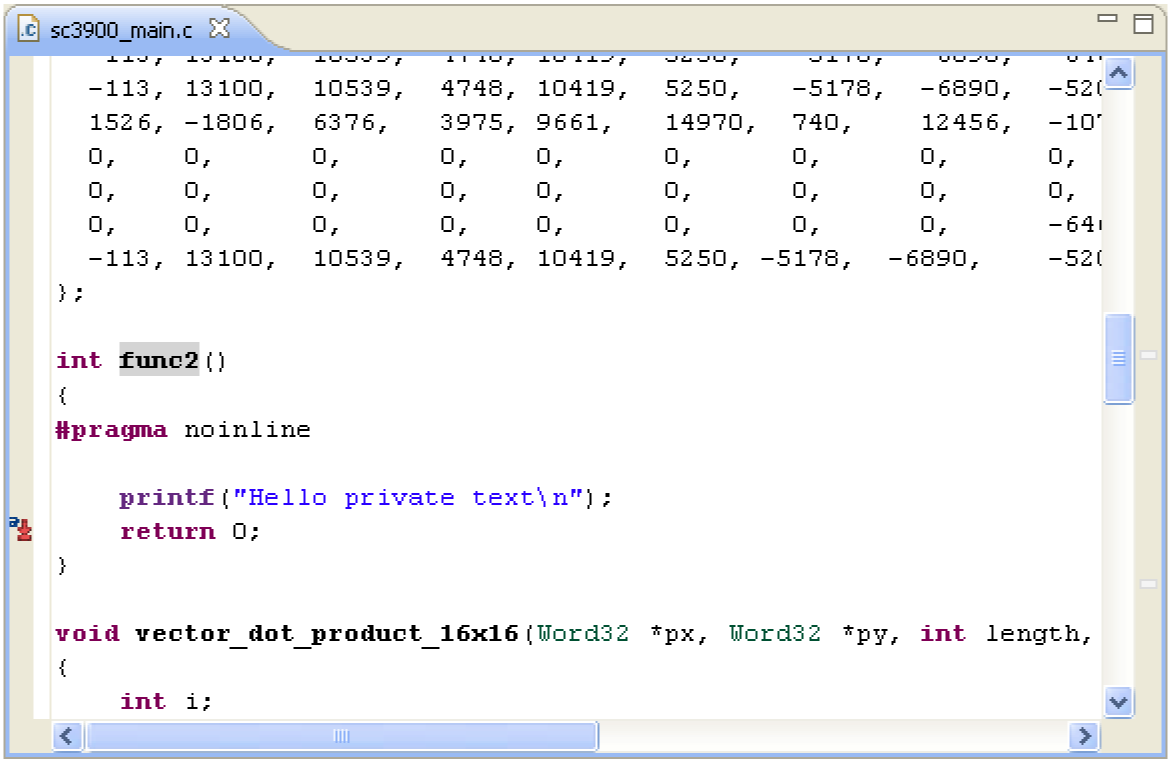
```
int main()
{
  int number_of_error_values=0, my_error;
  Word32 prod_ref = 0xDAE1135C;
    Word32 prod;
    func2();
    vector_dot_product_16x16((Word32 *)&X[0], (Word32 *)&Y[0], N, &prod);
    printf("Checking results.\n");
    if (prod!=prod_ref)
    {
        printf( "The Output is not bit-exact: prod=0x%08X != 0x%08X=prod_ref\n",prod,prod_ref);
        number_of_error_values++;
        my_error = prod-prod_ref;
    }
    if (number_of_error_values==0)
    {
        printf("No wrong results found.\n");
    }
```

Figure 4-1. Setting Start Tracepoint

5. Select the line on which you want to set the stop tracepoint, right-click on the marker bar, and select **Toggle Trace Stop Point> Hardware Trace Point** option from the context menu. The same option is also used to remove the stop tracepoint from the marker bar.

NOTE

The mouse pointer over a tracepoint icon displays the attributes of tracepoints on that line.



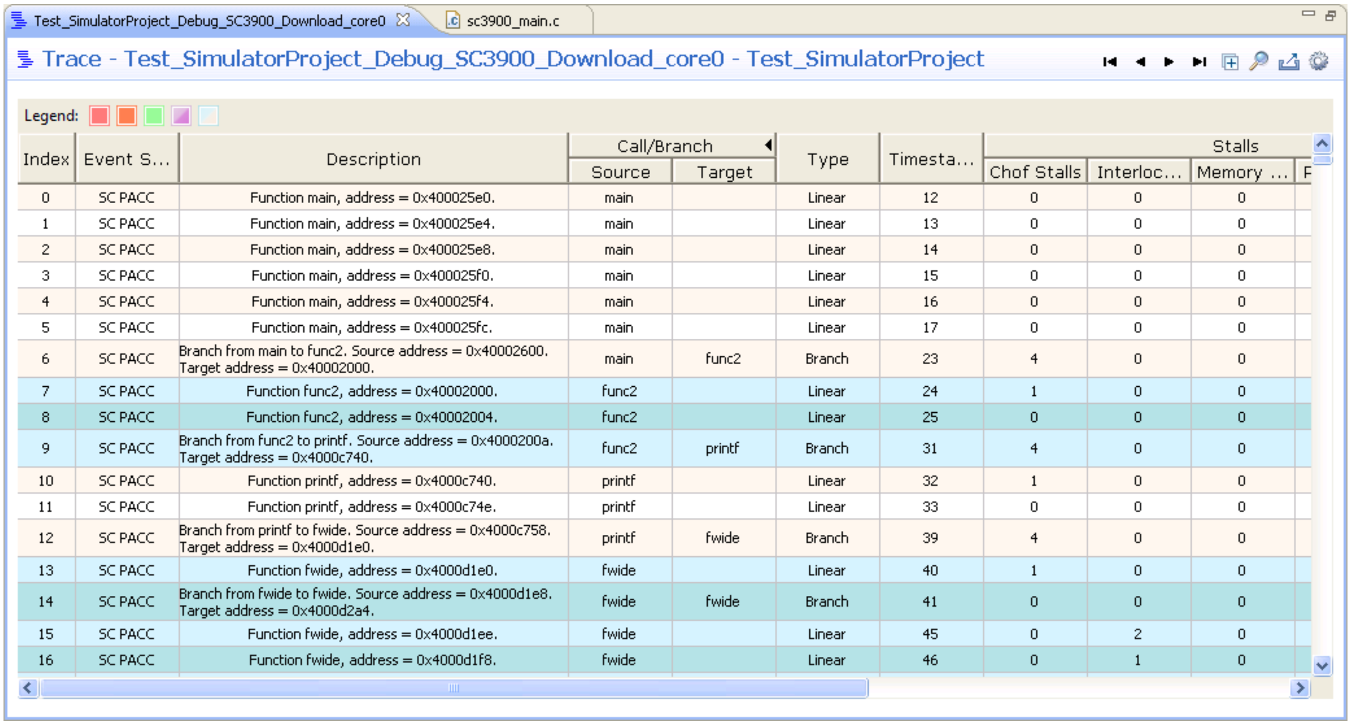
```
  -113, 13100,  10539,  4748, 10419,  5250,   -5178,  -6890,   -52
  1526, -1806,  6376,   3975, 9661,   14970,  740,    12456,   -10
  0,    0,      0,      0,    0,      0,      0,      0,       0,
  0,    0,      0,      0,    0,      0,      0,      0,       0,
  0,    0,      0,      0,    0,      0,      0,      0,       -64
  -113, 13100,  10539,  4748, 10419,  5250, -5178,  -6890,     -52
};

int func2()
{
#pragma noinline

    printf("Hello private text\n");
    return 0;
}

void vector_dot_product_16x16(Word32 *px, Word32 *py, int length,
{
    int i;
```

Figure 4-2. Setting Stop Tracepoint

NOTE

The start tracepoint is in green color and stop tracepoint is in red color. Do not set tracepoints on the lines containing only comments, brackets and variable declaration with no value.

6. After setting the tracepoints, debug the application and collect the data in the datafiles. The datafiles are generated by the simulator in which the data has been collected before setting the tracepoints in the source code. In this datafile, the **Description** field shows that the main function is called first and then it calls the func2 function.



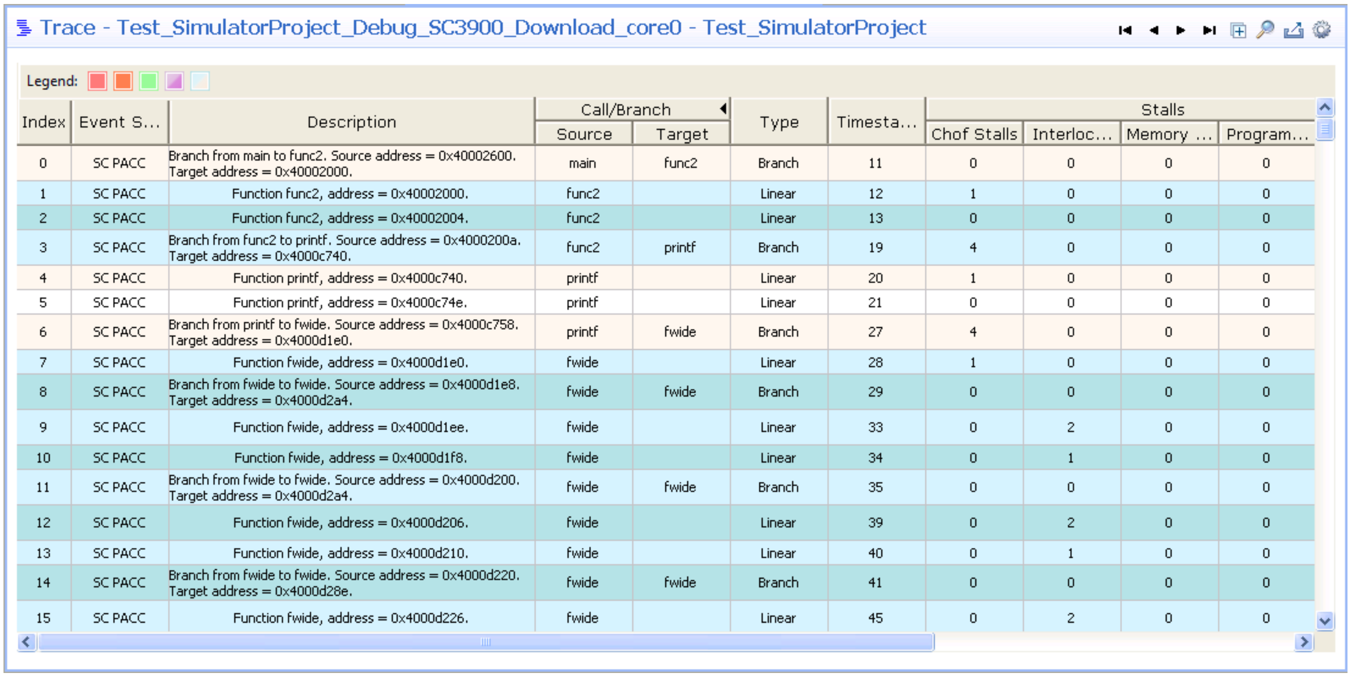
Trace - Test_SimulatorProject_Debug_SC3900_Download_core0 - Test_SimulatorProject

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...	Stalls			
			Source	Target			Chof Stalls	Interloc...	Memory ...	F
0	SC PACC	Function main, address = 0x400025e0.	main		Linear	12	0	0	0	
1	SC PACC	Function main, address = 0x400025e4.	main		Linear	13	0	0	0	
2	SC PACC	Function main, address = 0x400025e8.	main		Linear	14	0	0	0	
3	SC PACC	Function main, address = 0x400025f0.	main		Linear	15	0	0	0	
4	SC PACC	Function main, address = 0x400025f4.	main		Linear	16	0	0	0	
5	SC PACC	Function main, address = 0x400025fc.	main		Linear	17	0	0	0	
6	SC PACC	Branch from main to func2. Source address = 0x40002600. Target address = 0x40002000.	main	func2	Branch	23	4	0	0	
7	SC PACC	Function func2, address = 0x40002000.	func2		Linear	24	1	0	0	
8	SC PACC	Function func2, address = 0x40002004.	func2		Linear	25	0	0	0	
9	SC PACC	Branch from func2 to printf. Source address = 0x4000200a. Target address = 0x4000c740.	func2	printf	Branch	31	4	0	0	
10	SC PACC	Function printf, address = 0x4000c740.	printf		Linear	32	1	0	0	
11	SC PACC	Function printf, address = 0x4000c74e.	printf		Linear	33	0	0	0	
12	SC PACC	Branch from printf to fwide. Source address = 0x4000c758. Target address = 0x4000d1e0.	printf	fwide	Branch	39	4	0	0	
13	SC PACC	Function fwide, address = 0x4000d1e0.	fwide		Linear	40	1	0	0	
14	SC PACC	Branch from fwide to fwide. Source address = 0x4000d1e8. Target address = 0x4000d2a4.	fwide	fwide	Branch	41	0	0	0	
15	SC PACC	Function fwide, address = 0x4000d1ee.	fwide		Linear	45	0	2	0	
16	SC PACC	Function fwide, address = 0x4000d1f8.	fwide		Linear	46	0	1	0	

Figure 4-3. Trace Data Page Before Setting Tracepoints

In this datafile, after setting the start and stop tracepoints as displayed in the above figures, the **Description** field shows that the main function calls the func2 and the tracing starts from func2.

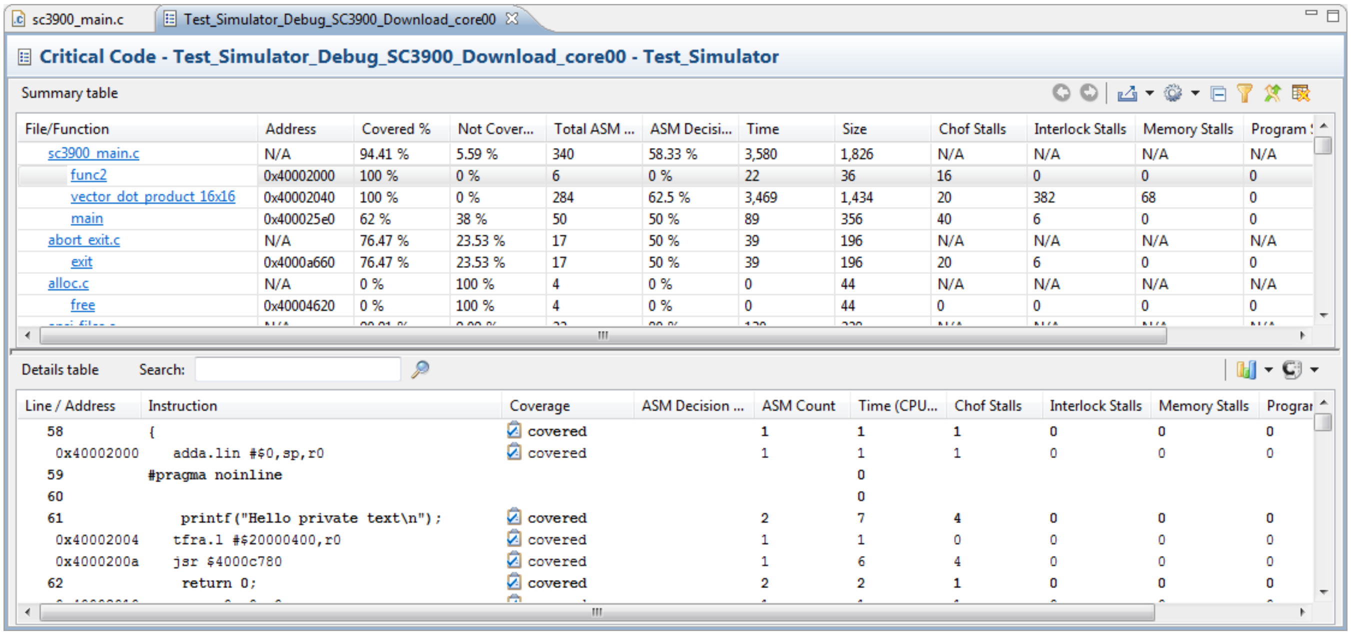

Trace - Test_SimulatorProject_Debug_SC3900_Download_core0 - Test_SimulatorProject

Legend: 🟥 🟧 🟩 🟪 ⬜

Index	Event S...	Description	Call/Branch		Type	Timesta...	Stalls			
			Source	Target			Chof Stalls	Interloc...	Memory ...	Program...
0	SC PACC	Branch from main to func2. Source address = 0x40002600. Target address = 0x40002000.	main	func2	Branch	11	0	0	0	0
1	SC PACC	Function func2, address = 0x40002000.	func2		Linear	12	1	0	0	0
2	SC PACC	Function func2, address = 0x40002004.	func2		Linear	13	0	0	0	0
3	SC PACC	Branch from func2 to printf. Source address = 0x4000200a. Target address = 0x4000c740.	func2	printf	Branch	19	4	0	0	0
4	SC PACC	Function printf, address = 0x4000c740.	printf		Linear	20	1	0	0	0
5	SC PACC	Function printf, address = 0x4000c74e.	printf		Linear	21	0	0	0	0
6	SC PACC	Branch from printf to fwide. Source address = 0x4000c758. Target address = 0x4000d1e0.	printf	fwide	Branch	27	4	0	0	0
7	SC PACC	Function fwide, address = 0x4000d1e0.	fwide		Linear	28	1	0	0	0
8	SC PACC	Branch from fwide to fwide. Source address = 0x4000d1e8. Target address = 0x4000d2a4.	fwide	fwide	Branch	29	0	0	0	0
9	SC PACC	Function fwide, address = 0x4000d1ee.	fwide		Linear	33	0	2	0	0
10	SC PACC	Function fwide, address = 0x4000d1f8.	fwide		Linear	34	0	1	0	0
11	SC PACC	Branch from fwide to fwide. Source address = 0x4000d200. Target address = 0x4000d2a4.	fwide	fwide	Branch	35	0	0	0	0
12	SC PACC	Function fwide, address = 0x4000d206.	fwide		Linear	39	0	2	0	0
13	SC PACC	Function fwide, address = 0x4000d210.	fwide		Linear	40	0	1	0	0
14	SC PACC	Branch from fwide to fwide. Source address = 0x4000d220. Target address = 0x4000d28e.	fwide	fwide	Branch	41	0	0	0	0
15	SC PACC	Function fwide, address = 0x4000d226.	fwide		Linear	45	0	2	0	0

Figure 4-4. Trace Data Page After Setting Tracepoints

In the critical code data, you can see that tracing has started from func2.



sc3900_main.c | Test_Simulator_Debug_SC3900_Download_core00

Critical Code - Test_Simulator_Debug_SC3900_Download_core00 - Test_Simulator

Summary table

File/Function	Address	Covered %	Not Cover...	Total ASM ...	ASM Decisi...	Time	Size	Chof Stalls	Interlock Stalls	Memory Stalls	Program !
sc3900_main.c	N/A	94.41 %	5.59 %	340	58.33 %	3,580	1,826	N/A	N/A	N/A	N/A
func2	0x40002000	100 %	0 %	6	0 %	22	36	16	0	0	0
vector_dot_product_16x16	0x40002040	100 %	0 %	284	62.5 %	3,469	1,434	20	382	68	0
main	0x400025e0	62 %	38 %	50	50 %	89	356	40	6	0	0
abort_exit.c	N/A	76.47 %	23.53 %	17	50 %	39	196	N/A	N/A	N/A	N/A
exit	0x4000a660	76.47 %	23.53 %	17	50 %	39	196	20	6	0	0
alloc.c	N/A	0 %	100 %	4	0 %	0	44	N/A	N/A	N/A	N/A
free	0x40004620	0 %	100 %	4	0 %	0	44	0	0	0	0

Details table Search:

Line / Address	Instruction	Coverage	ASM Decision ...	ASM Count	Time (CPU...	Chof Stalls	Interlock Stalls	Memory Stalls	Prograr
58	{	covered		1	1	1	0	0	0
0x40002000	adda.lin #$0,sp,r0	covered		1	1	1	0	0	0
59	#pragma noinline				0				
60					0				
61	printf("Hello private text\n");	covered		2	7	4	0	0	0
0x40002004	tfra.l #$20000400,r0	covered		1	1	0	0	0	0
0x4000200a	jsr $4000c780	covered		1	6	4	0	0	0
62	return 0;	covered		2	2	1	0	0	0

Figure 4-5. Critical Code Data After Setting Tracepoints

Similarly, you can view the other datafiles. For details, refer to the Viewing Data topic.

You can also set the tracepoints in the **Disassembly** view when the debug session is active. To get the **Disassembly** view, select **Windows > Show View > Disassembly**. (Debug perspective) The **Disassembly** view displays the C/C++ program in assembly language. When you set the tracepoints in **Disassembly** view, the tracepoints are automatically displayed in the Editor area.

To set the tracepoints in **Disassembly** view, follow the same steps as you followed for Editor area.

The tracepoint attributes of Source Code and **Disassembly** view are displayed in the **Analysispoints** view. To get the **Analysispoints** view, select **Windows > Show View > Other > Software Analysis > Analysispoints**.

4.1.2 Target Hardware

This section explains how to start ans stop hardware tracepoints.

For B4 boards, there are 3 start hardware tracepoints and 3 stop hardware tracepoints. If hardware tracepoints exceed this limit, only 3 start tracepoints and 3 stop tracepoints will control the trace, the other tracepoints will be ignored.

If only one stop hardware tracepoint is used then application will be traced from its beginning until tracepoint. Also, if only one start hardware tracepoint is used then application will be traced from tracepoint until the end of the application. When more than one hardware tracepoints are used then they should be used in pair, for each start hardware tracepoint set, a stop start hardware tracepoint should be set as well. A start hardware tracepoint marks the start of traced section while a stop hardware tracepoint marks the end of traced section.

Hardware tracepoints control only program trace and profiling trace, they have no effect on any other kind of trace, for example, debug messages, or user defined events.

NOTE

Start and stop tracepoints should be set on C/C++ or assembly linear instructions to ensure all change of flow instructions between start and stop are traced. If start and stop tracepoints are set on C/C++ or assembly change of flow instructions then these instructions are not traced.

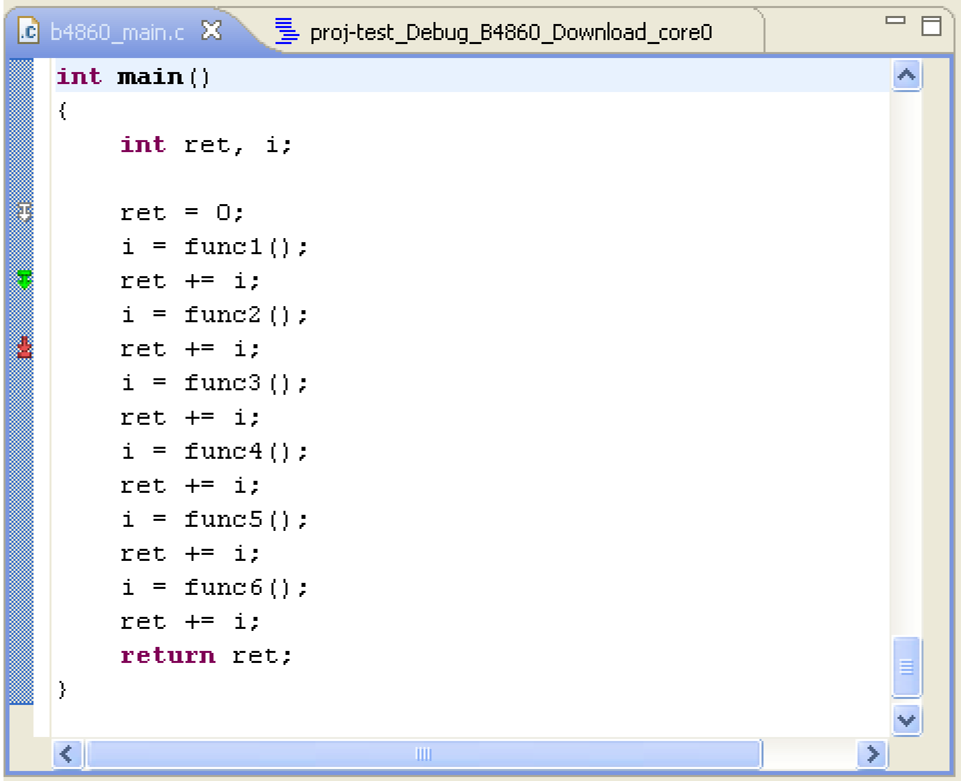
NOTE

Hardware or software breakpoints and stepping should not be used inside the traced section in order to obtain accurate trace and profile results. The reason of this behavior is that on single stepped VLES, the events generated in parallel like trace generation, address detection, and event counting are ignored. For experts only, trace and profile results can be accurate even though hardware or software breakpoints or stepping are used inside the traced section only if they are set such that none of single stepped VLES-es to interfere with the already mentioned events.

To set the hardware tracepoints in source code:

1. In the **CodeWarrior Projects** view, select the **Source** folder of b4860 under the project.
2. Double-click on the source file, for example, b4860_main.c. The content of the file can be viewed on the right-side of the pane.
3. Select the line on which you want to set the start hardware tracepoint. For example, set start tracepoint on C linear instruction before `func2` call as shown in figure below.
4. Right-click on the marker bar, and select the **Toggle Trace Start Point> Hardware Trace Point** option from the context menu.
5. Select the line on which you want to set the stop hardware tracepoint. For example, set stop tracepoint on C linear instruction after `func2` call as shown in figure below.

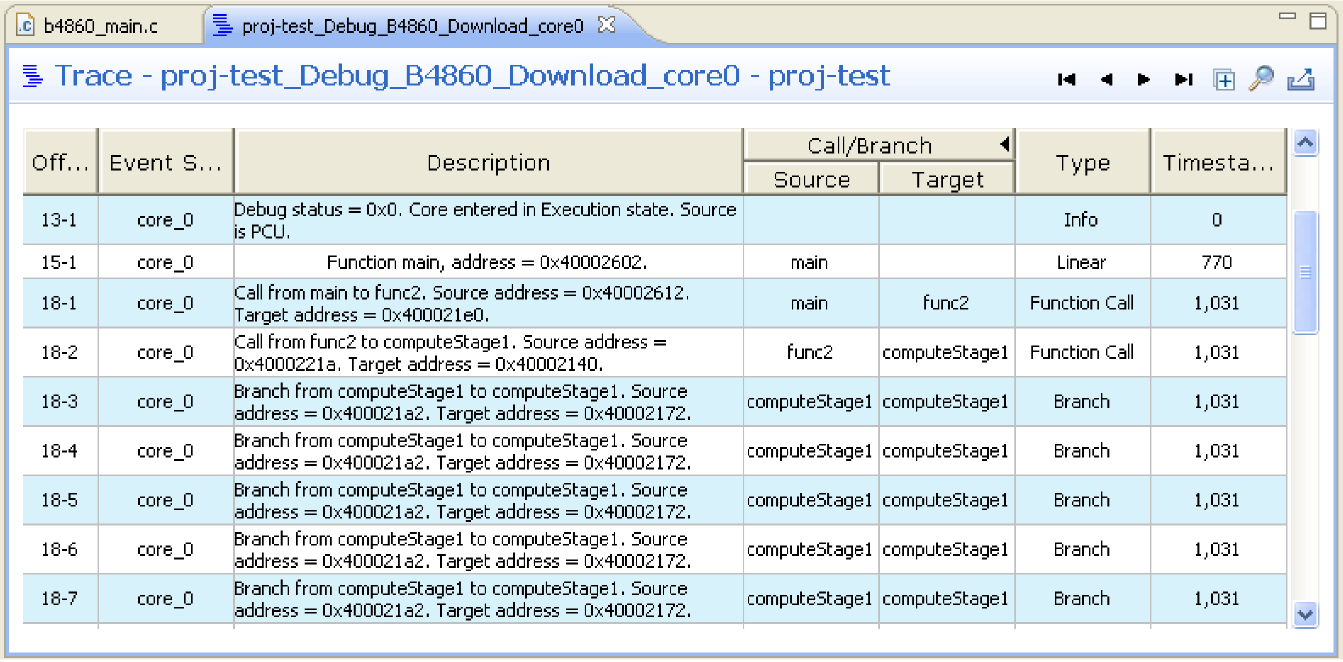
6. Select the line on which you want to set the stop hardware tracepoint, right-click on the marker bar, and select **Toggle Trace Stop Point > Hardware Trace Point** option from the context menu.



```
int main()
{
    int ret, i;

    ret = 0;
    i = func1();
    ret += i;
    i = func2();
    ret += i;
    i = func3();
    ret += i;
    i = func4();
    ret += i;
    i = func5();
    ret += i;
    i = func6();
    ret += i;
    return ret;
}
```

Figure 4-6. Setting Start and Stop Tracepoints

7. In the **Trace and Profile** tab of **Debug** window, select **Program trace** scenario.
8. Click **Debug** and wait until application reaches its end.
9. Terminate the application.
10. Inspect trace viewer and available profiling viewers.


b4860_main.c
proj-test_Debug_B4860_Download_core0
Trace - proj-test_Debug_B4860_Download_core0 - proj-test

Off...	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
13-1	core_0	Debug status = 0x0. Core entered in Execution state. Source is PCU.			Info	0
15-1	core_0	Function main, address = 0x40002602.	main		Linear	770
18-1	core_0	Call from main to func2. Source address = 0x40002612. Target address = 0x400021e0.	main	func2	Function Call	1,031
18-2	core_0	Call from func2 to computeStage1. Source address = 0x4000221a. Target address = 0x40002140.	func2	computeStage1	Function Call	1,031
18-3	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,031
18-4	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,031
18-5	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,031
18-6	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,031
18-7	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,031

Figure 4-7. Trace Data After Setting Hardware Tracepoints

The above figure shows the datafiles generated by the target hardware in which the data has been collected after setting the tracepoints in the source code. Tracing begins with a call from `main` to `func2` and ends with return `func2` to `main`.

If a B4 StarCore project contains a launch configuration with SC3900 simulator set as target, the hardware tracepoints will have no effect on trace while running the project on a B4 board. Perform the first two steps if tracepoints are already added on a source file from a project, otherwise skip to point 3 onwards if tracepoints have never been added on a source file:

1. Select **Window > Show View > Other > Software Analysis > Analysispoints**, the **Analysispoints** view is displayed. In the **Analysispoints** view, remove all the analysispoints.
2. Remove the `SaAnalysispointsManager.apconfig` file from the project folder for which the trace data has been collected.

NOTE

Project files are saved in <*workspace location* >/< *project* > folder.

3. Rename the extension of any `*.launch` file from the project location for which the target has been selected as SC3900 simulator.

NOTE

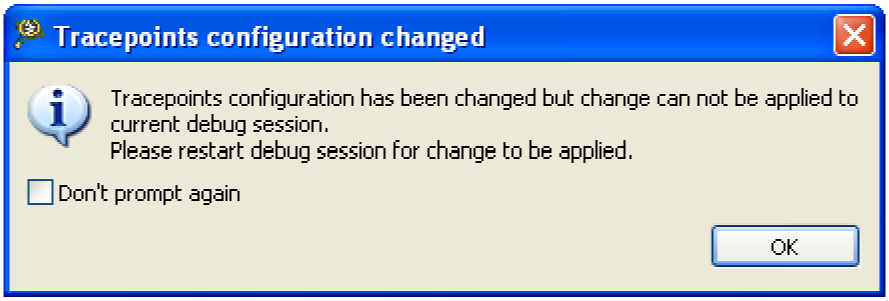
The launch files are saved in <*workspace location* >/< *project* >/< *Debug_Settings* > folder.

4. Select the existing project from **CodeWarrior Projects** view.
5. Right-click the project and select **Refresh** menu.
6. Add tracepoints on the source file of the selected project.

4.1.2.1 Modifying Hardware Tracepoint Configuration

This section explains how to modify the configuration of hardware tracepoints.

You can change the hardware tracepoints configuration during the debug session, such as adding, removing, enabling, disabling or ignoring a tracepoint. You will be notified when changes are made to hardware tracepoints configuration, but they are not applied to the current debug session.



Tracepoints configuration changed

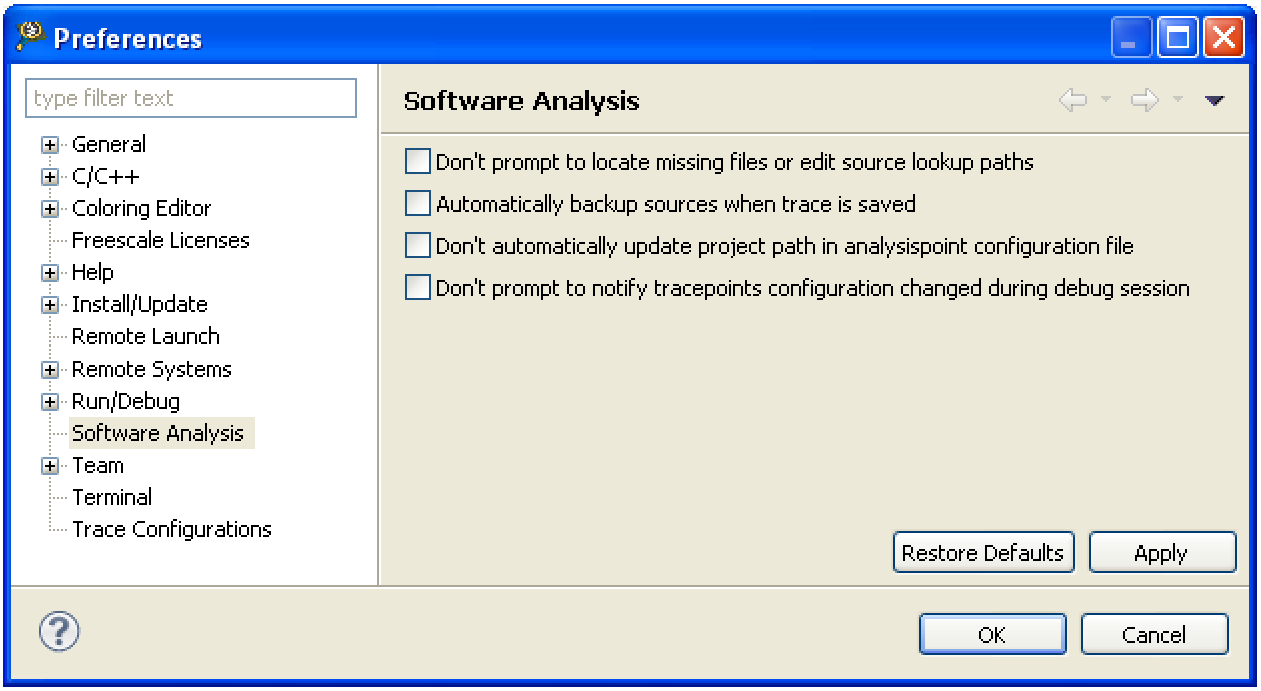
Tracepoints configuration has been changed but change can not be applied to current debug session.
Please restart debug session for change to be applied.

□ Don't prompt again

Figure 4-8. Tracepoints Configuration Changed

This message box appears at every resume if tracepoints configuration change is made and can not be applied to current debug session. If you have selected **Don't prompt again** checkbox then you will not be notified.

You can activate tracepoints configuration change notification by not selecting **Don't prompt to notify tracepoints configuration changed during debug session** in the **Preferences -> Software Analysis** window.



Software Analysis

□ Don't prompt to locate missing files or edit source lookup paths

□ Automatically backup sources when trace is saved

□ Don't automatically update project path in analysispoint configuration file

□ Don't prompt to notify tracepoints configuration changed during debug session

Figure 4-9. Preferences - Software Analysis

4.1.3 Viewing Tracepoints in Analysispoints

Use the Analysispoints view to view the the attributes of tracepoints.

This view display the attributes of tracepoints set in source code or/and assembly code. To view the attributes of tracepoints, (Debug perpective) select **Window > Show View > Other > Software Analysis > Analysispoints**, the **Analysispoints** view is displayed.

The source line tracepoint is set in the Editor area and the address tracepoint is set from the **Disassembly** view.

4.1.3.1 Disassembly View

This section lists the address tracepoint attributes set in **Disassembly**.

Following are the sddress tracepoint attributes set:

- Type of tracepoint (Software or Hardware)
- File name

- Address of the instruction of assembly code
- Action (Start or Stop)

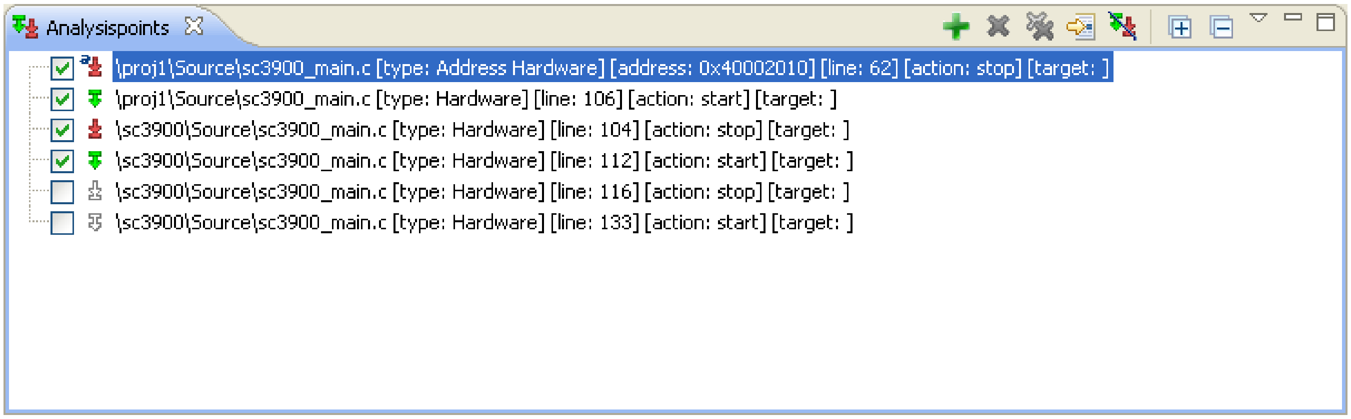
4.1.3.2 Editor Area

Lists the source code tracepoint attributes set in the Editor area.

Following are the source code tracepoint attributes set in the Editor area:

- Type of tracepoint (Software or Hardware)
- File name
- Line number
- Action (Start or Stop)

You can also enable and disable tracepoints from this view. To enable or disable a particular tracepoint, right-click on the attribute and select **EnableTracepoint** or **Disable Tracepoint** option from the context menu.



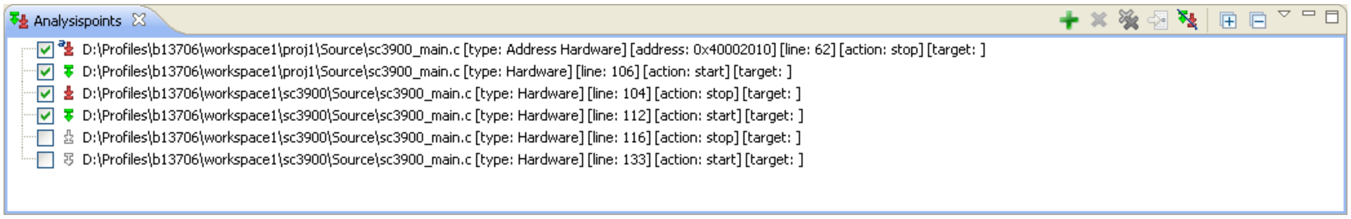
☑ \proj1\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002010] [line: 62] [action: stop] [target:]
☑ \proj1\Source\sc3900_main.c [type: Hardware] [line: 106] [action: start] [target:]
☑ \sc3900\Source\sc3900_main.c [type: Hardware] [line: 104] [action: stop] [target:]
☑ \sc3900\Source\sc3900_main.c [type: Hardware] [line: 112] [action: start] [target:]
□ \sc3900\Source\sc3900_main.c [type: Hardware] [line: 116] [action: stop] [target:]
□ \sc3900\Source\sc3900_main.c [type: Hardware] [line: 133] [action: start] [target:]

Figure 4-10. Analysispoints View

The **Analysispoints** view shows you where start and stop tracepoints are set in the Editor area or the **Disassembly** view. It helps you to navigate to the source code or assembly code where the tracepoint is set. If you want to go to the specific line of a source file where the tracepoint is set, click the **Go To File for Analysispoints** icon. This option will help you to navigate to that line.

If you want to ignore all the tracepoints at the time of execution, click the **Ignore All Analysispoints** icon. To remove a tracepoint or remove all the tracepoints, click **Remove Selected Analysispoint** icon or **Remove All Analysispoints** icon.

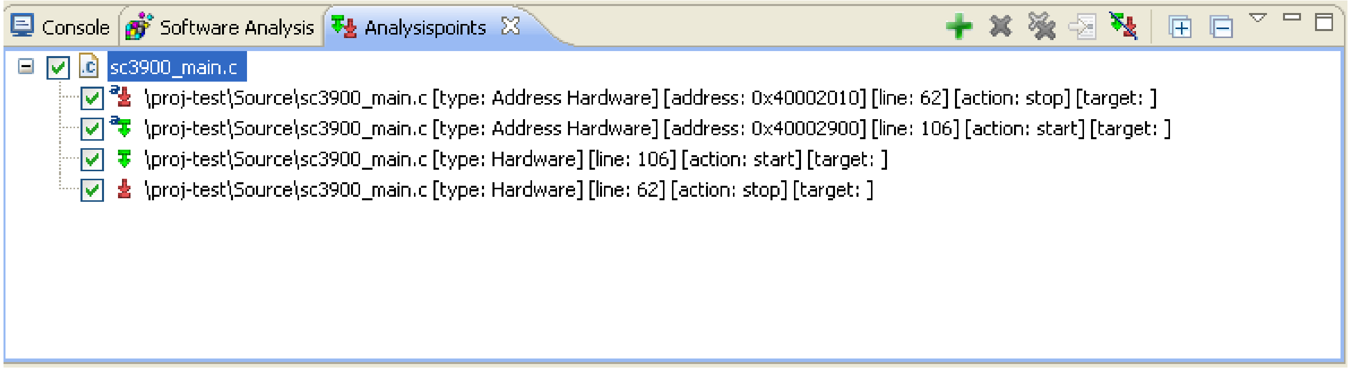
To view the complete path of the files of the attribute, click the **View Menu** of **Analysispoints** view and select **Show Full Paths** option.



☑ D:\Profiles\b13706\workspace1\proj1\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002010] [line: 62] [action: stop] [target:]
☑ D:\Profiles\b13706\workspace1\proj1\Source\sc3900_main.c [type: Hardware] [line: 106] [action: start] [target:]
☑ D:\Profiles\b13706\workspace1\sc3900\Source\sc3900_main.c [type: Hardware] [line: 104] [action: stop] [target:]
☑ D:\Profiles\b13706\workspace1\sc3900\Source\sc3900_main.c [type: Hardware] [line: 112] [action: start] [target:]
□ D:\Profiles\b13706\workspace1\sc3900\Source\sc3900_main.c [type: Hardware] [line: 116] [action: stop] [target:]
□ D:\Profiles\b13706\workspace1\sc3900\Source\sc3900_main.c [type: Hardware] [line: 133] [action: start] [target:]

Figure 4-11. Analysispoints View - Show Full Path

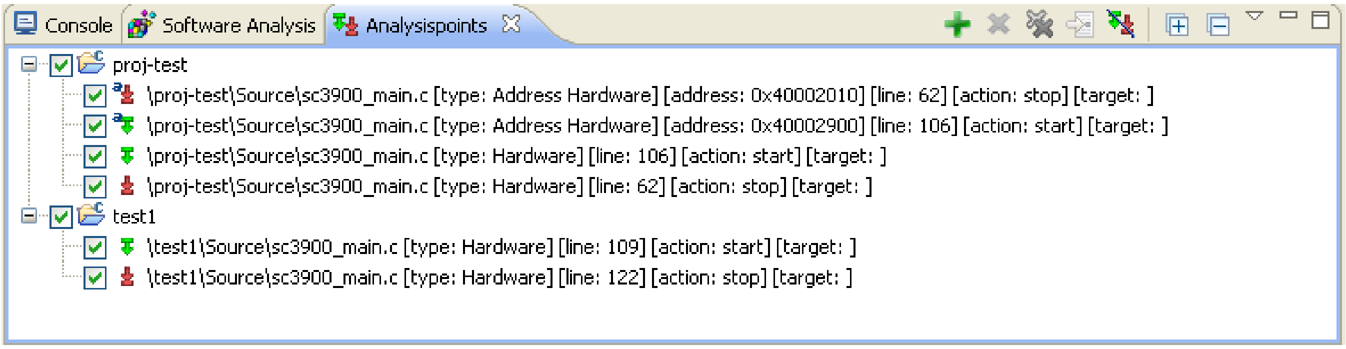
You can also group the attributes by files or projects. To group the attributes by files, click **ViewMenu** > **Group By> Files** option. The attributes are grouped by files as shown below.



☑ sc3900_main.c
☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002010] [line: 62] [action: stop] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002900] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 62] [action: stop] [target:]

Figure 4-12. Group By Files

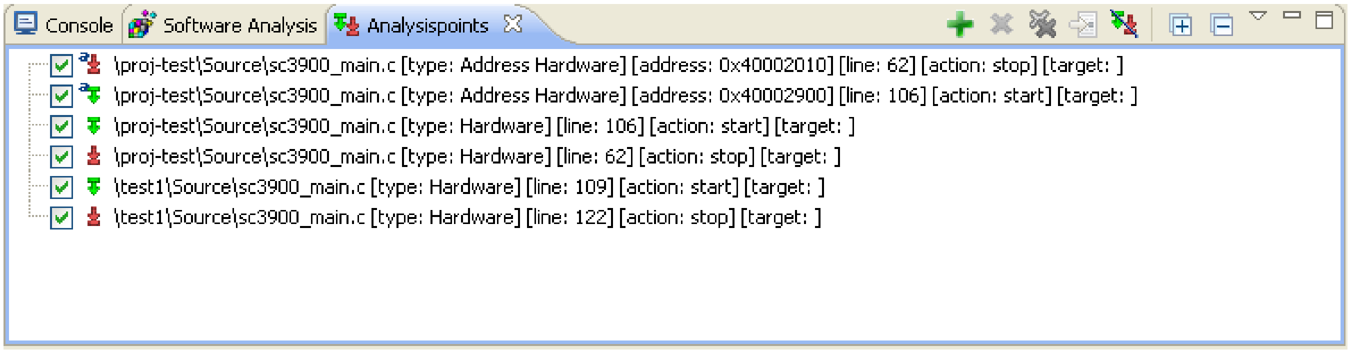
To group the attributes by projects, click **ViewMenu > Group By > Projects** option. The attributes are grouped by projects as shown below.



☑ proj-test
☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002010] [line: 62] [action: stop] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002900] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 62] [action: stop] [target:]
☑ test1
☑ \test1\Source\sc3900_main.c [type: Hardware] [line: 109] [action: start] [target:]
☑ \test1\Source\sc3900_main.c [type: Hardware] [line: 122] [action: stop] [target:]

Figure 4-13. Group By Projects

To ungroup the attributes, click **View Menu > Group By > Analysispoints** option as shown below.



☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002010] [line: 62] [action: stop] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Address Hardware] [address: 0x40002900] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 106] [action: start] [target:]
☑ \proj-test\Source\sc3900_main.c [type: Hardware] [line: 62] [action: stop] [target:]
☑ \test1\Source\sc3900_main.c [type: Hardware] [line: 109] [action: start] [target:]
☑ \test1\Source\sc3900_main.c [type: Hardware] [line: 122] [action: stop] [target:]

Figure 4-14. Ungroup the Attributes

4.2 Enabling and Disabling the Tracepoints

You can also enable and disable tracepoints using this Analysispoints view.

If you want to enable the Start or Stop tracepoint, right-click on the marker bar where Start or Stop tracepoints are already set and in disabled state, select the **Enable Tracepoint** option from the context menu.

If you want to disable the Start or Stop tracepoint, right-click on the marker bar where Start or Stop tracepoint is already set and enabled, select the **Disable Tracepoint** option from the context menu. A disabled tracepoint will have no effect during the collection of trace data. You can also disable the tracepoint from **Analysispoints** view. Right-click on the selected attribute and select **Disable** option. The unchecked attribute indicates the disabled tracepoint.

4.3 Limitation of hardware tracepoints

This section explains the various limitations of hardware tracepoints.

B4 SC3900 core has limited number of hardware resources allocated for hardware tracepoints. You can define upto 3 start hardware tracepoints and 3 stop tracepoints, but if the number of hardware tracepoints defined by user exceeds the number of defined tracepoints, then a notification is raised. The notification message specifies:

- the core id on which installation has failed
- the hardware tracepoints that are not successfully installed on target are identified through line, source file pair or address

- the reason for installation failure
- the total number of start/stop hardware tracepoints tried to be installed and number of them for which hardware resources were found free, only if install has failed for start/stop hardware tracepoints category

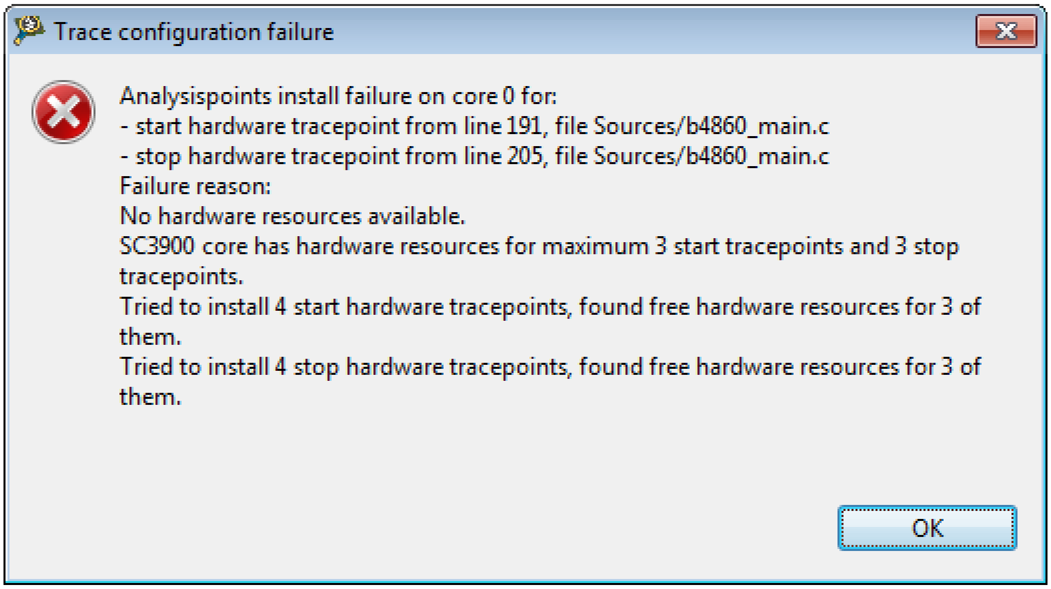
However, install is performed only on tracepoints that are enabled. The hardware tracepoints that are reported with install failure doesnot have any effect on trace in the ongoing debug session.

It is also possible that some of the hardware resources allocated for hardware tracepoints are already reserved or in use by some other module, for example, debugger is another module that can use these resources for hardware breakpoints.

Let us consider an example of hardware tracepoints notification failure, where start and stop hardware line tracepoints exceeds the maximum number of hardware resources available:

1. Create a b4860 hardware project. For details, refer to Hardware Profiling
2. Open the source file, `b4860_main.c`.
3. Set four start and stop tracepoints at the beginning and end of the functions in `b4860_main.c` file.
4. Build the project.
5. Open **Debug Configurations** dialog and select **Program trace** option available under the **Trace scenarios** list on the **Basic** tab page of **Trace and Profile** tab.
6. Click **Apply** and debug the application.
7. Resume the debug session and wait until application reaches its end.

   The **Trace configuration failure** dialog gets prompted, while the debug application keeps running till its end.



Trace configuration failure

Analysispoints install failure on core 0 for:
- start hardware tracepoint from line 191, file Sources/b4860_main.c
- stop hardware tracepoint from line 205, file Sources/b4860_main.c
Failure reason:
No hardware resources available.
SC3900 core has hardware resources for maximum 3 start tracepoints and 3 stop tracepoints.
Tried to install 4 start hardware tracepoints, found free hardware resources for 3 of them.
Tried to install 4 stop hardware tracepoints, found free hardware resources for 3 of them.

Figure 4-15. Trace configuration failure dialog

8. Open **Software Analysis** view and inspect the trace viewer.

The trace viewer shows the trace collected according to the successful installation of hardware tracepoints. If there is at least one start tracepoint successfully installed then trace is considered to be stopped, and will be started when first start tracepoint is hit.

4.4 Multicore hardware tracepoints

This section explains the support of hardware tracepoints for B4 SC 3900 on multicore.

For each core:

- There are up to 3 start hardware tracepoints and 3 stop hardware tracepoints available
- Hardware tracepoints control only program and profiling trace modes and have no effect on other trace modes
- When number of hardware tracepoints set exceeds above limitation user notification is raised

When a tracepoint is added, it is by default set (enabled) on all the available cores of the platform. For example, if the target is B4860 then the tracepoint is set on all the six cores of SC3900.

4.4.1 Set cores

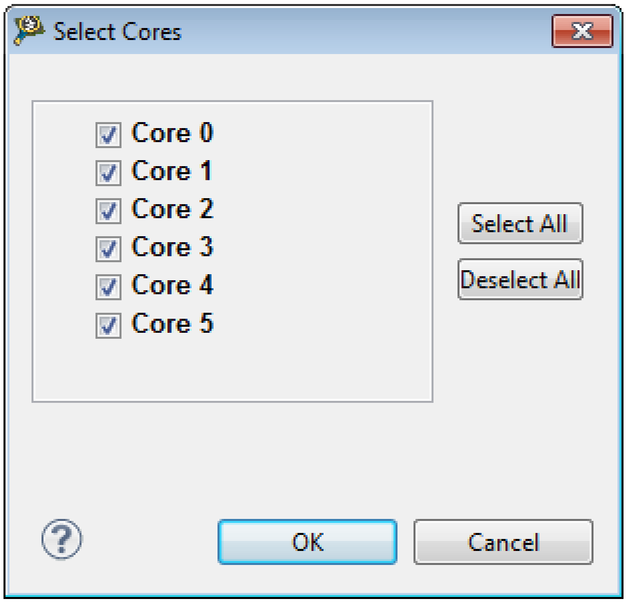
The Set cores feature is used to handle hardware tracepoints on multicore.

It is available for each hardware tracepoints.

Follow the steps given below to set hardware tracepoints on multicore:

1. Select **Window > Show View > Other > Software Analysis > Analysispoints**, the Analysispoints view is displayed.
2. Righ-click on the tracepoints listed in the Analysispoints view and select **Set cores** option.

   The **Select cores** dialog box appears.

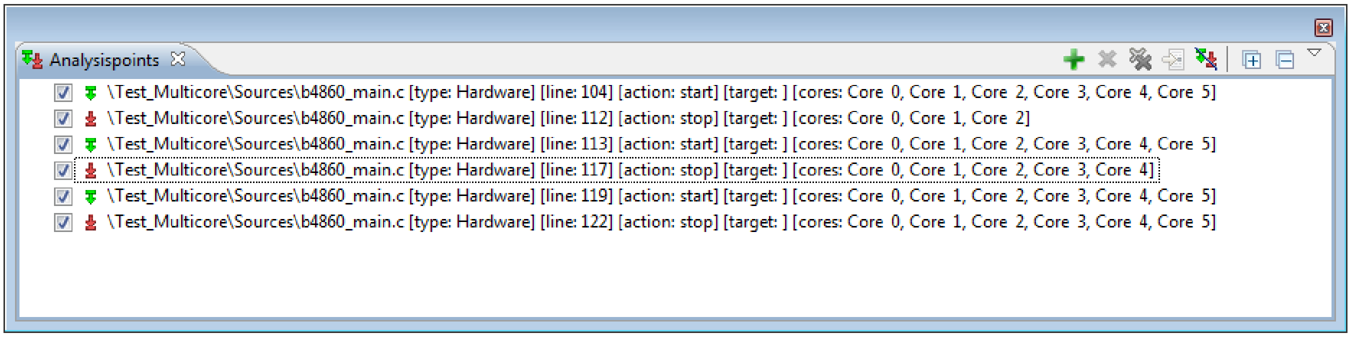


☑ Core 0
☑ Core 1
☑ Core 2
☑ Core 3
☑ Core 4
☑ Core 5

Figure 4-16. Select cores dialog

3. Select the cores on which hardware tracepoint needs to be enabled or disabled.
4. Click **OK**.

   The list of cores on which tracepoint is enabled gets added in Analysispoints view along with type and action performed.



```
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 104] [action: start] [target: ] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 112] [action: stop] [target: ] [cores: Core 0, Core 1, Core 2]
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 113] [action: start] [target: ] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 117] [action: stop] [target: ] [cores: Core 0, Core 1, Core 2, Core 3, Core 4]
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 119] [action: start] [target: ] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Multicore\Sources\b4860_main.c [type: Hardware] [line: 122] [action: stop] [target: ] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
```

Figure 4-17. Selected cores displayed in Analysispoints view

Also when start and stop hardware line tracepoints exceeds the maximum number of hardware resources available, the hardware tracepoints failure notification message will be prompted for each individual core.

Chapter 5
Setting Counterpoints

Counterpoints allow you to count events between two points of the executed code. The results are computed based on the collected trace data; the counterpoints are searched in trace by their addresses. When counterpoints are found in the executed code, the delta statistics is computed between two counterpoints. Counterpoints are only supported in Profiling trace.

This chapter explains:

5.1 Setting Counterpoints

This section explains setting the source line counterpoint in the Editor area and the address counterpoint in the **Disassembly** view.

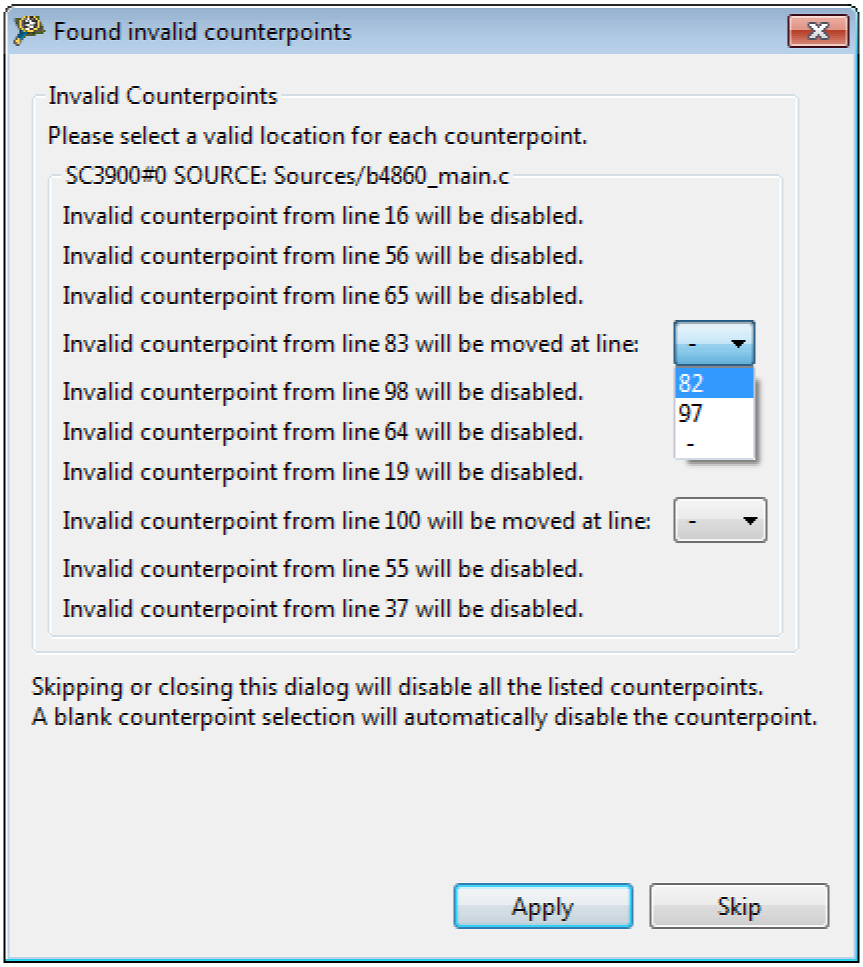
The counterpoints are set similarly as tracepoints.

Both types of counterpoints have associated address:

- For Disassembly counterpoints, it is the address of the assembly code where the counterpoint is placed in the **Disassembly** view.
- For source line counterpoints, it is the address of the source line where the counterpoint is placed. The address is determined at runtime.

  You can see the attributes of counterpoints from the **Analysispoints** view.

The counterpoints results are computed based on the traced data; you must place the counterpoints on the code that generates the trace. If the counterpoints are set on the lines containing only comments, brackets and variable declaration with no value, a **Found invalid counterpoints** dialog box gets prompted as shown below.



Found invalid counterpoints

Invalid Counterpoints

Please select a valid location for each counterpoint.

SC3900#0 SOURCE: Sources/b4860_main.c

Invalid counterpoint from line 16 will be disabled.

Invalid counterpoint from line 56 will be disabled.

Invalid counterpoint from line 65 will be disabled.

Invalid counterpoint from line 83 will be moved at line: - (82 / 97 / -)

Invalid counterpoint from line 98 will be disabled.

Invalid counterpoint from line 64 will be disabled.

Invalid counterpoint from line 19 will be disabled.

Invalid counterpoint from line 100 will be moved at line: -

Invalid counterpoint from line 55 will be disabled.

Invalid counterpoint from line 37 will be disabled.

Skipping or closing this dialog will disable all the listed counterpoints.
A blank counterpoint selection will automatically disable the counterpoint.

Figure 5-1. Found invalid counterpoints dialog box

You can either disable the counterpoint or move the counterpoint to location suggested in the dialog box.

This topic explains how to set counterpoints on the sample code of the simulator also how to view the counterpoints.

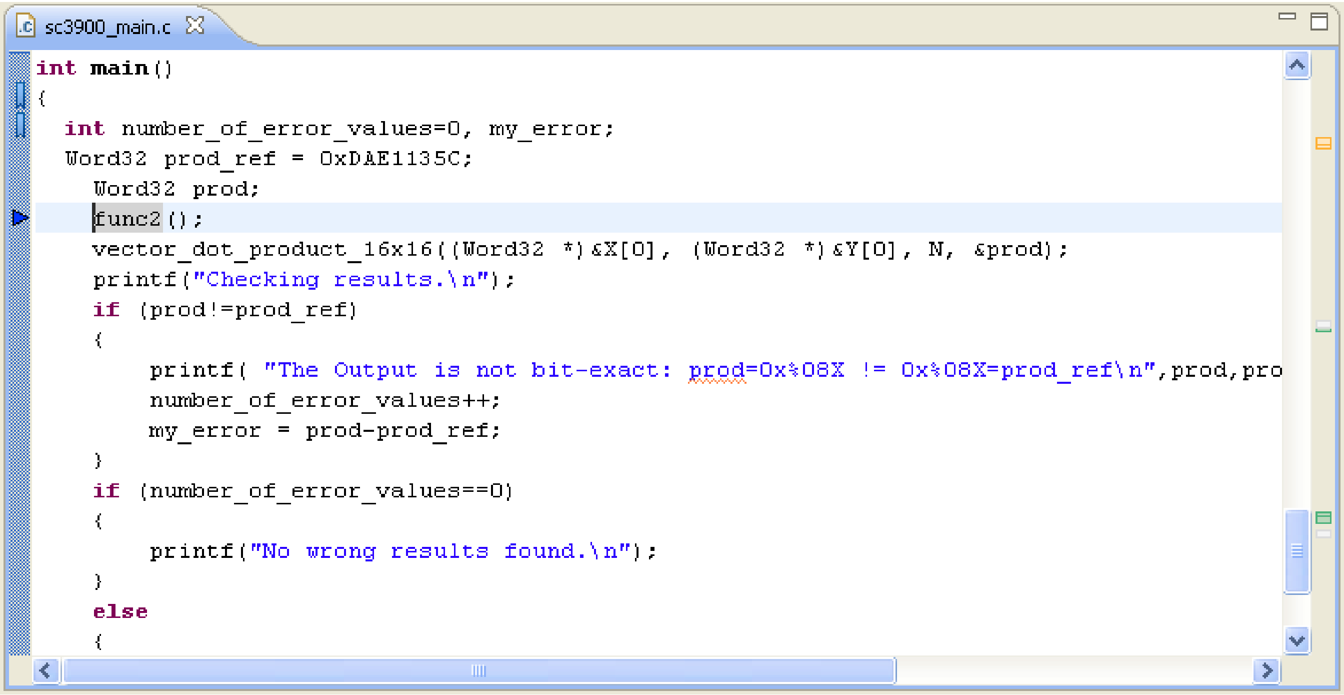
To set the counterpoints in source code:

1. In the **CodeWarrior Projects** view, expand the **Source** folder under the project.
2. Double-click on the source file, for example, SC3900_main.c.

   The contents of the file appear on the right-side of the pane.

3. Select the line on which you want to set the counterpoint, and right-click on the Marker bar.
4. Select the **Toggle Counter Point** option from the context menu. The same option is used to remove the counterpoint from the Marker bar.

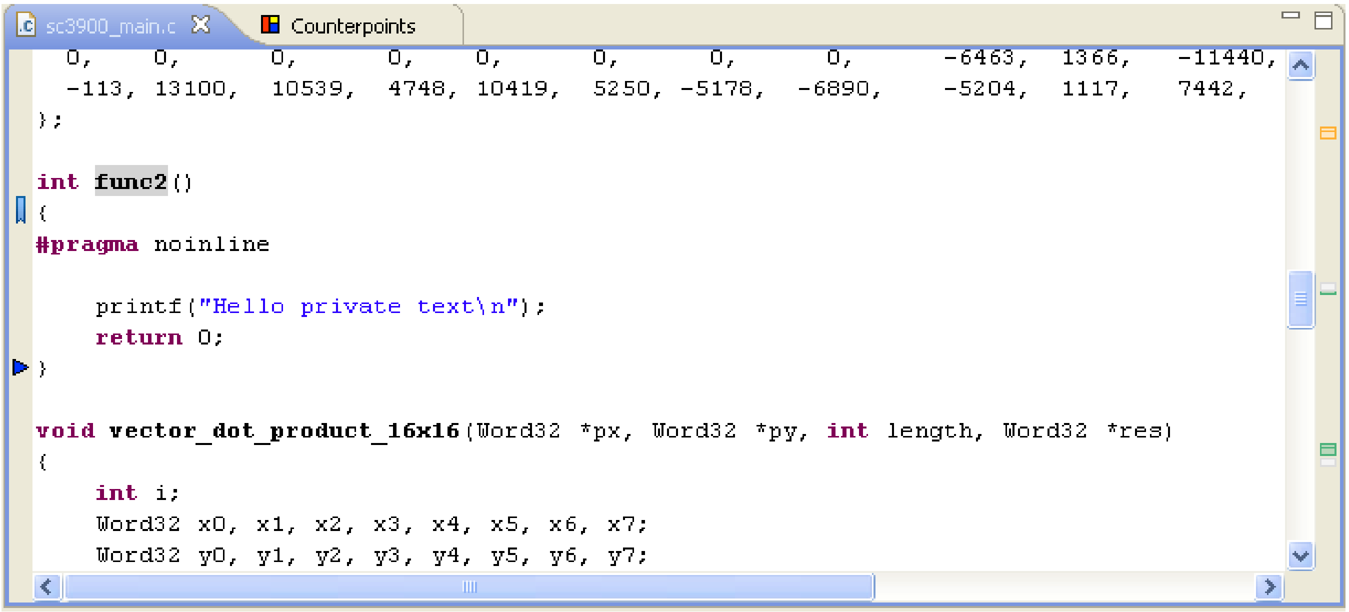
   The counterpoint appears in blue color as shown in the figure.



```
int main()
{
  int number_of_error_values=0, my_error;
  Word32 prod_ref = 0xDAE1135C;
    Word32 prod;
    func2();
    vector_dot_product_16x16((Word32 *)&X[0], (Word32 *)&Y[0], N, &prod);
    printf("Checking results.\n");
    if (prod!=prod_ref)
    {
        printf( "The Output is not bit-exact: prod=0x%08X != 0x%08X=prod_ref\n",prod,pro
        number_of_error_values++;
        my_error = prod-prod_ref;
    }
    if (number_of_error_values==0)
    {
        printf("No wrong results found.\n");
    }
    else
    {
```

Figure 5-2. First Counterpoints Set in Source Code

5. Select the line on which you want to set another counterpoint. You can also set the counterpoints in the **Disassembly** view when the application is in debug mode.

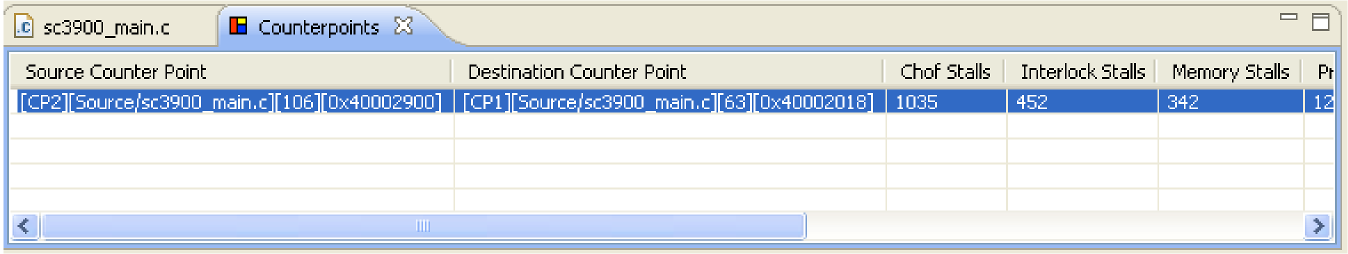


```
    0,      0,      0,      0,      0,      0,      0,      0,      -6463,  1366,   -11440,
    -113,   13100,  10539,  4748,   10419,  5250,   -5178,  -6890,  -5204,  1117,   7442,
};

int func2()
{
#pragma noinline

    printf("Hello private text\n");
    return 0;
}

void vector_dot_product_16x16(Word32 *px, Word32 *py, int length, Word32 *res)
{
    int i;
    Word32 x0, x1, x2, x3, x4, x5, x6, x7;
    Word32 y0, y1, y2, y3, y4, y5, y6, y7;
```

Figure 5-3. Second Counterpoint Set in Source Code

6. After setting the counterpoints, debug the application and collect the data in the datafiles.

   The datafile is generated after setting the counterpoints. If two different counterpoints are hit consecutively, the delta statistics will be computed between these two points as shown below.



Source Counter Point	Destination Counter Point	Chof Stalls	Interlock Stalls	Memory Stalls	Pr
[CP2][Source/sc3900_main.c][106][0x40002900]	[CP1][Source/sc3900_main.c][63][0x40002018]	1035	452	342	12

Figure 5-4. Events Collected After Setting Counterpoints

   You can match the counterpoint address with the trace address. A counterpoint is considered hit when its address is found in the collected trace.

The following table describes the fields of the counterpoints datafile.

Table 5-1. Description of Fields - Counter Points Datafile

Field	Description
Source Counter Point	The counterpoint that marks the beginning of the interval for which the delta statistics are computed
Destination Counter Point	The counterpoint that marks the end of the interval for which the delta statistics are computed

Table continues on the next page...

Table 5-1. Description of Fields - Counter Points Datafile (continued)

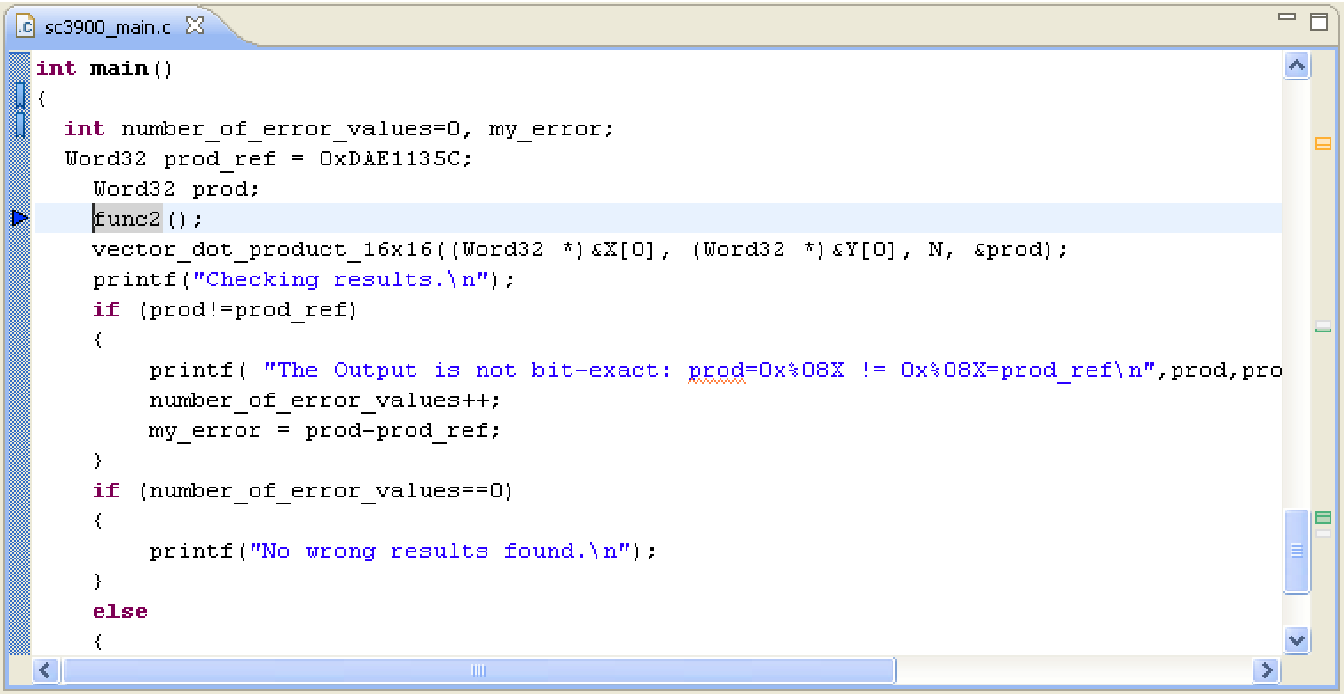
Field	Description
Chof Stalls	Displays the number of bubble cycles incurred by change of program flow for each instruction
Interlock Stalls	Stalls introduced by core to resolve resource conflicts
Memory Stalls	Displays the number of bubble cycles incurred by data bus hold for each instruction
Program Stalls	Displays the number of bubble cycles incurred by program starvation for each instruction
Rewind Stalls	Displays the number of bubble cycles incurred by a rewind on the core buses for each instruction
BTBable COF	The decoded COF is BTBable
COF Taken	The COF is taken; changes the program flow
False BTB Hit	Indicates False hit

NOTE

> The mouse pointer over a counterpoint icon displays the attributes of counterpoints on that line.

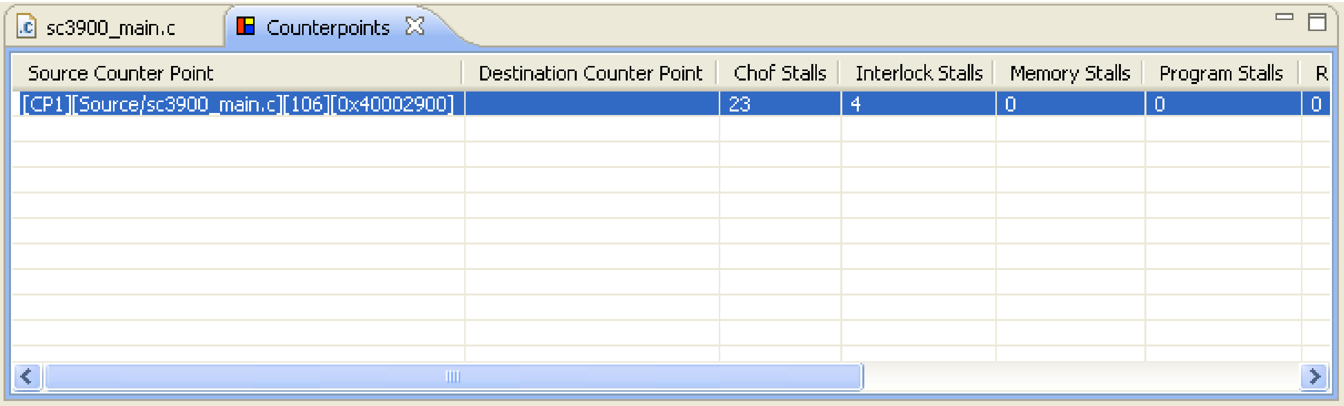
Few examples are given below to understand how result is computed in different situations.

- If there is only source counterpoint with no destination, the counterpoint is hit only once, the statistics will show the events counted till that point.



```
int main()
{
  int number_of_error_values=0, my_error;
  Word32 prod_ref = 0xDAE1135C;
    Word32 prod;
    func2();
    vector_dot_product_16x16((Word32 *)&X[0], (Word32 *)&Y[0], N, &prod);
    printf("Checking results.\n");
    if (prod!=prod_ref)
    {
        printf( "The Output is not bit-exact: prod=0x%08X != 0x%08X=prod_ref\n",prod,pro
        number_of_error_values++;
        my_error = prod-prod_ref;
    }
    if (number_of_error_values==0)
    {
        printf("No wrong results found.\n");
    }
    else
    {
```

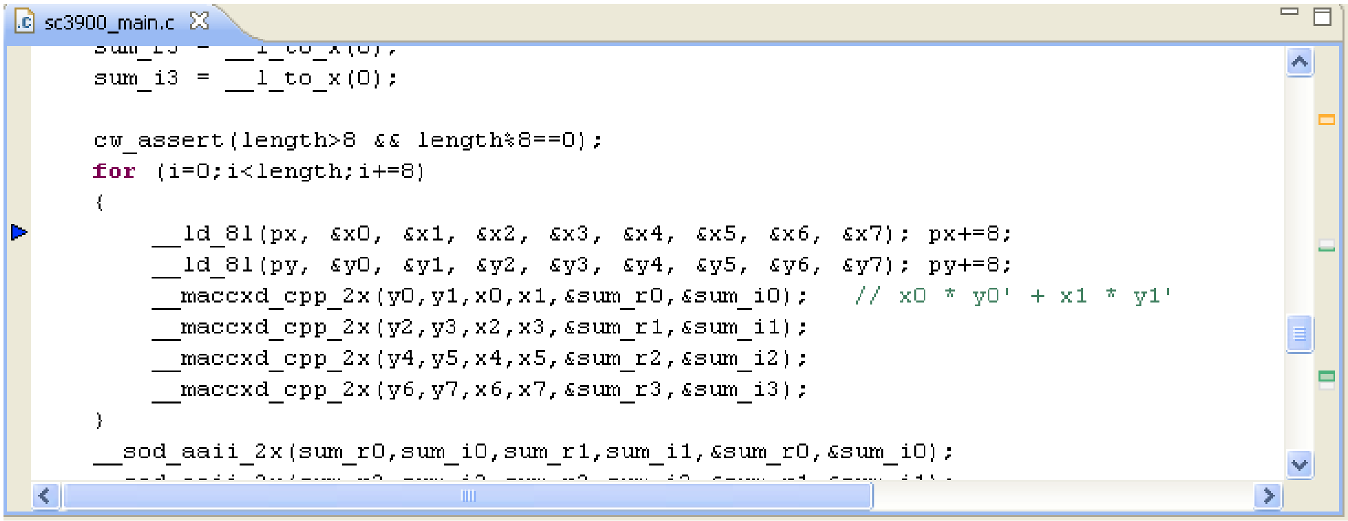
Figure 5-5. Setting Only Source Counterpoint Page



Source Counter Point	Destination Counter Point	Chof Stalls	Interlock Stalls	Memory Stalls	Program Stalls	R
[CP1][Source/sc3900_main.c][106][0x40002900]		23	4	0	0	0

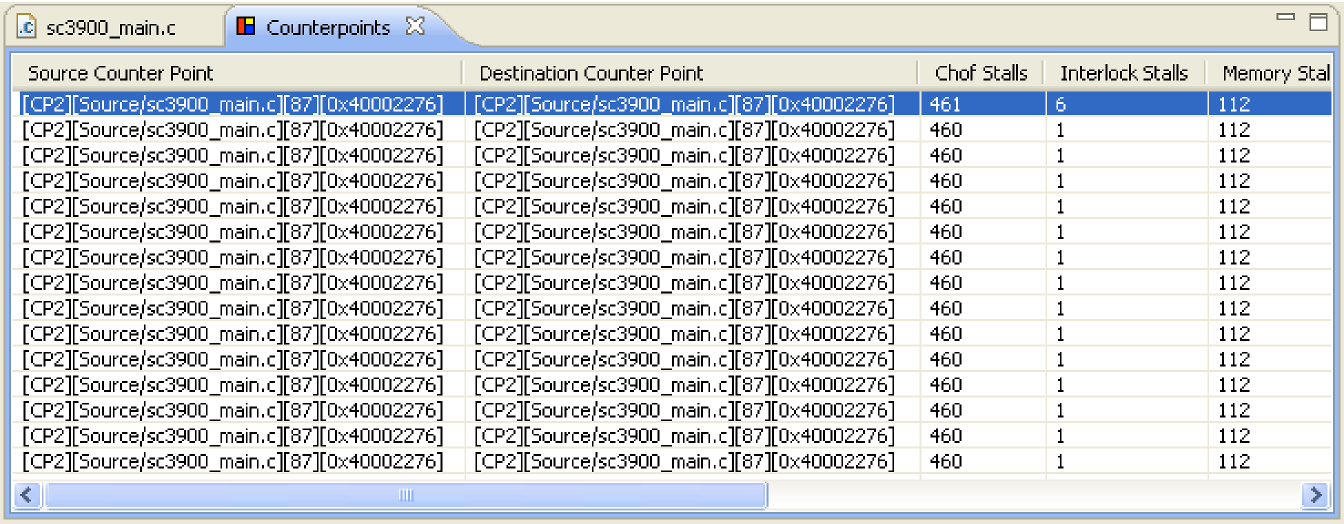
Figure 5-6. Result Computed for Source Counterpoint Page

- If there is a counterpoint set in trace loop as shown in the below figure. The same counterpoint is found in trace consecutively and the delta statistics will be computed each time when that counterpoint is hit.



```
    sum_i3 = __l_to_x(0);

    cw_assert(length>8 && length%8==0);
    for (i=0;i<length;i+=8)
    {
        __ld_8l(px, &x0, &x1, &x2, &x3, &x4, &x5, &x6, &x7); px+=8;
        __ld_8l(py, &y0, &y1, &y2, &y3, &y4, &y5, &y6, &y7); py+=8;
        __maccxd_cpp_2x(y0,y1,x0,x1,&sum_r0,&sum_i0);   // x0 * y0' + x1 * y1'
        __maccxd_cpp_2x(y2,y3,x2,x3,&sum_r1,&sum_i1);
        __maccxd_cpp_2x(y4,y5,x4,x5,&sum_r2,&sum_i2);
        __maccxd_cpp_2x(y6,y7,x6,x7,&sum_r3,&sum_i3);
    }
    __sod_aaii_2x(sum_r0,sum_i0,sum_r1,sum_i1,&sum_r0,&sum_i0);
```

Figure 5-7. Counterpoint Set after Loop Statement Page



Source Counter Point	Destination Counter Point	Chof Stalls	Interlock Stalls	Memory Stal
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	461	6	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112
[CP2][Source/sc3900_main.c][87][0x40002276]	[CP2][Source/sc3900_main.c][87][0x40002276]	460	1	112

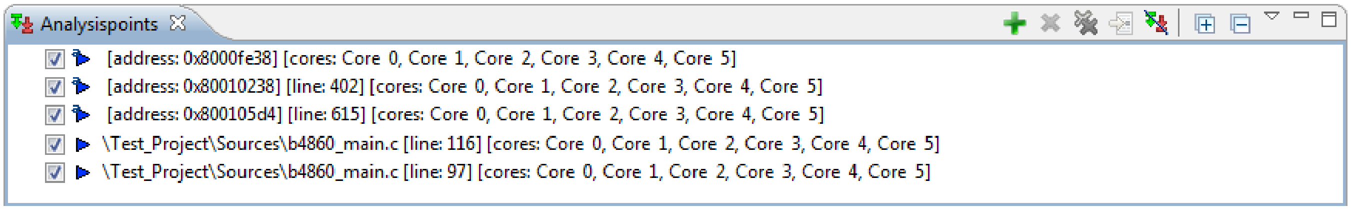
Figure 5-8. Result Page

5.2 Viewing Counterpoints in Analysispoints

You can view the attributes of counterpoints using **Analysispoints** view.

Analysis points view display the attributes of counterpoints set in source code or/and assembly code. To view the attributes of counterpoints, select **Window > Show View > Other > Analysis > Analysispoints**, the **Analysispoints** view is displayed in the below figure.

The source line counterpoints will have associated source file name and the line number, and the disassembly counterpoints will have associated source file name and the address.



☑ [address: 0x8000fe38] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ [address: 0x80010238] [line: 402] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ [address: 0x800105d4] [line: 615] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Project\Sources\b4860_main.c [line: 116] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Project\Sources\b4860_main.c [line: 97] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]

Figure 5-9. Analysispoints View

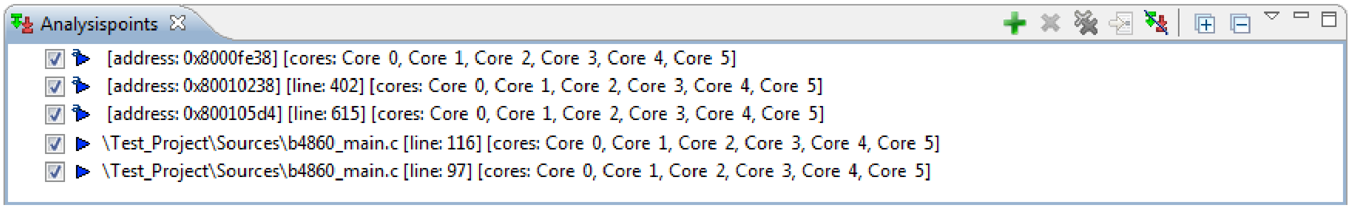
You can also enable and disable counterpoints from this view. To enable or disable a particular counterpoint, right-click on the attribute and select **Enable** or **Disable** option from the menu.

5.3 Enabling and Disabling the Counterpoints

This section explains the procedure to enable or disable counterpoints.

If you want to enable the counterpoint, right-click on the Marker bar where counterpoint is already set and in disabled state, select the **Enable Counterpoint** option from the context menu.

If you want to disable the counterpoint, right-click on the Marker bar where counterpoint is already set and enabled, select the **Disable Counterpoint** option from the context menu. A disabled counterpoint will have no effect and will not compute the delta statistics. You can also disable the counterpoint from **Analysispoints** view. Right-click on the selected attribute and select **Disable** option. The unchecked attribute indicates the disabled counterpoint.



☑ [address: 0x8000fe38] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ [address: 0x80010238] [line: 402] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ [address: 0x800105d4] [line: 615] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Project\Sources\b4860_main.c [line: 116] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]
☑ \Test_Project\Sources\b4860_main.c [line: 97] [cores: Core 0, Core 1, Core 2, Core 3, Core 4, Core 5]

Figure 5-10. Disable the Counterpoint Page

Chapter 6
Performance Analysis for B4 Devices

The Performance Analysis tool enables you to configure, collect, and analyze scenarios in the B4 devices. The tool helps you optimize your bareboard application by using core and platform counters to provide enhanced levels of visibility to the application. This tool is integrated in the CodeWarrior IDE to handle configuration and communication with the target system.

NOTE

> The Software Analysis and Performance Analysis tool cannot be used together as they use the same hardware resources. Currently, there is no mechanism for synchronization. You can either use Software Analysis or Performance Analysis.

This chapter explains:

- Performance Analysis Perspective
- Creating New Configuration
- Connecting to Board
- Selecting Scenarios
- Running Scenarios
- Viewing Results
- Custom scenarios

6.1 Performance Analysis Perspective

The **Performance Analysis** perspective provides a user-friendly interface to perform the scenario measurement activities, such as adding a configuration, configuring connection, defining scenarios, and collecting data.

The **Performance Analysis** perspective contains the following views:

- Performance Analysis View
- Configuration View
- Performance Analysis Session View
- Progress View

6.1.1 Performance Analysis View

The **Performance Analysis** view organizes measurement activities in terms of projects.

A project contains configuration information and the data that is collected as shown in the following figure.

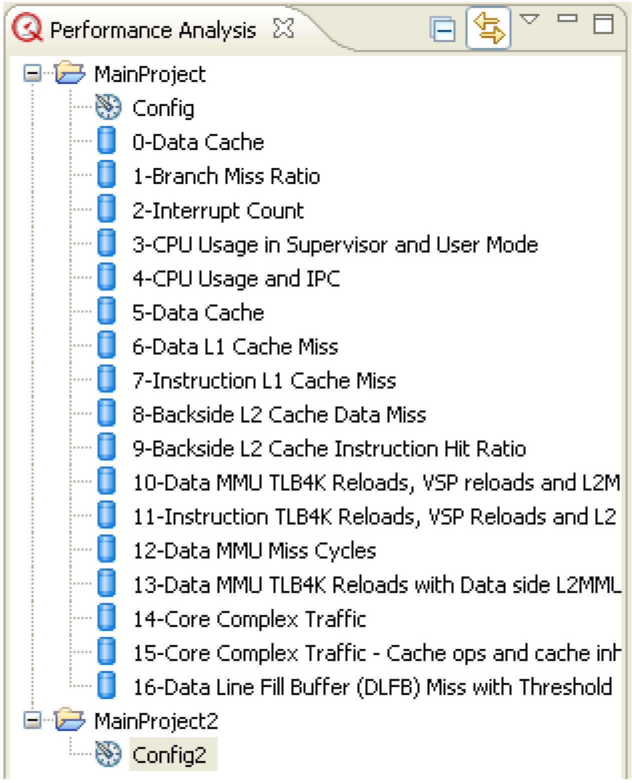

Performance Analysis
MainProject
Config
0-Data Cache
1-Branch Miss Ratio
2-Interrupt Count
3-CPU Usage in Supervisor and User Mode
4-CPU Usage and IPC
5-Data Cache
6-Data L1 Cache Miss
7-Instruction L1 Cache Miss
8-Backside L2 Cache Data Miss
9-Backside L2 Cache Instruction Hit Ratio
10-Data MMU TLB4K Reloads, VSP reloads and L2M
11-Instruction TLB4K Reloads, VSP Reloads and L2
12-Data MMU Miss Cycles
13-Data MMU TLB4K Reloads with Data side L2MML
14-Core Complex Traffic
15-Core Complex Traffic - Cache ops and cache inh
16-Data Line Fill Buffer (DLFB) Miss with Threshold
MainProject2
Config2

Figure 6-1. Performance Analysis View

You can import or export your project using the **Performance Analysis** view. To import a Performance Analysis configuration or project into your workspace, right-click the project and select the **Import** option from the context menu. In the **Import** wizard that appears, select the archived file or the root directory at which the project to be imported is located. Click **Finish**.

To export a Performance Analysis project, right-click the project and select the **Export** option from the context menu. In the **Export** wizard that appears, select the destination type and the directory where you want to export the project. Click **Finish**.

6.1.2 Configuration View

The **Configuration** view lets you configure connections and scenarios for measurement.

The following figure shows the **Configuration** view.

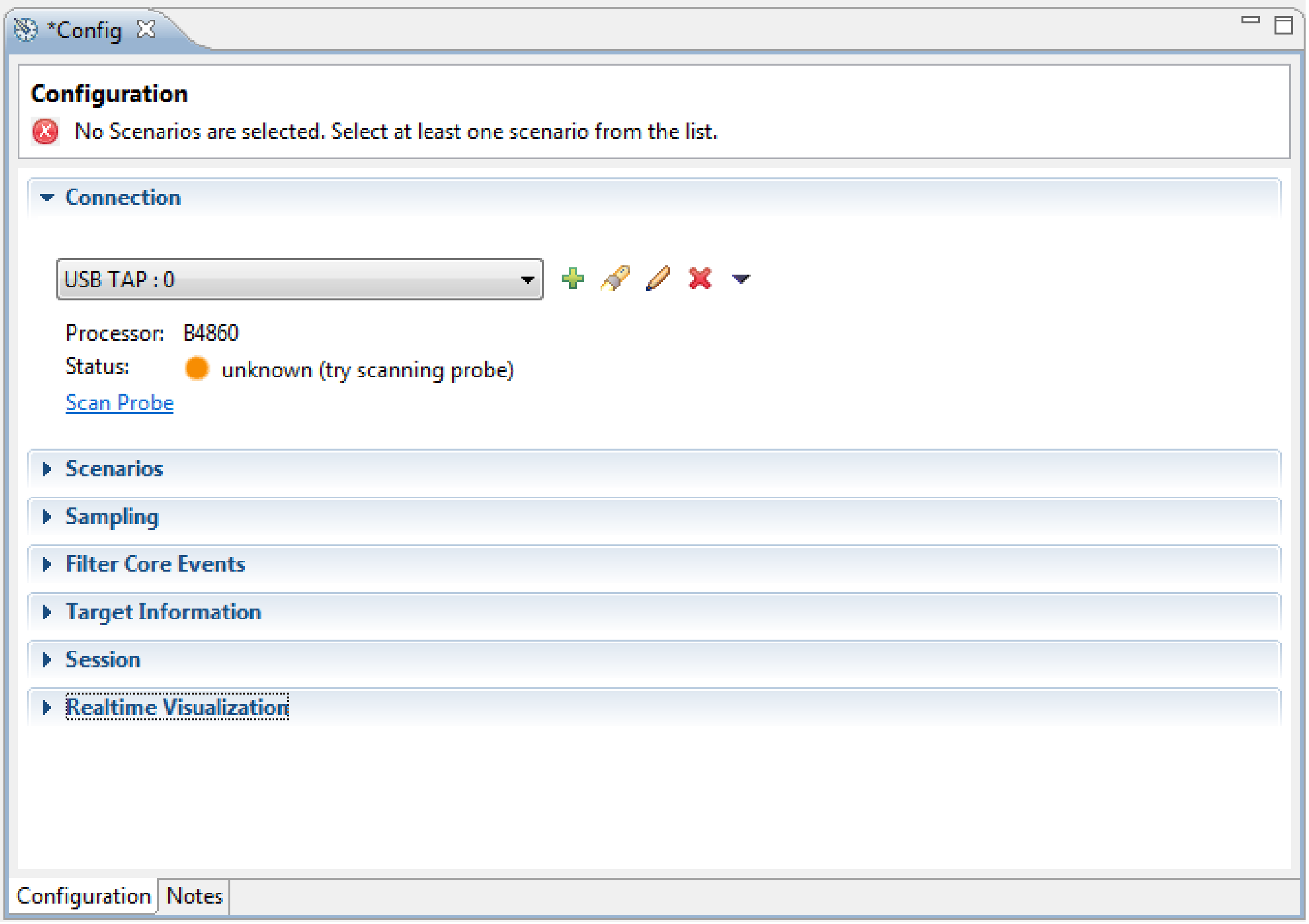


Figure 6-2. Configuration View

The **Configuration** view contains the following panes:

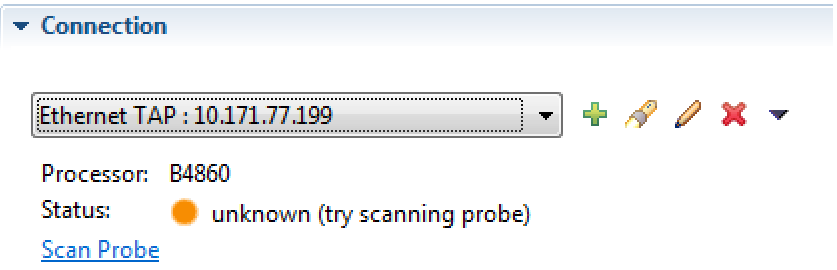
- Connection Pane
- Scenarios Pane
- Sampling Pane
- Filter Core Events pane

- Target Information Pane
- Session pane
- Realtime Visualization

6.1.2.1 Connection Pane

The **Connection** pane allows you to define the connection method that is used to communicate to the target hardware when a probe is being used in bareboard applications.

The following figure shows the **Connection Pane**.



Connection

Ethernet TAP : 10.171.77.199

Processor: B4860

Status: ● unknown (try scanning probe)

Scan Probe

Figure 6-3. Configuration View - Connection Pane

The following table lists the options available in the **Connection** pane.

Table 6-1. Configuration View - Connection Pane Options

Option	Name	Description
▾	Drop-down menu	Lists the previously selected connections and displays the currently selected connection on top. Appears blank if there is no selected connection.
✚	New Connection	Lets you add/configure a connection manually to a bareboard application. Refer Adding Connection Manually for more details.
✎	Edit Connection	Lets you modify an existing connection. The connection could be either automatically selected or manually added.
✖	Remove Connection	Removes an existing connection from the list. Select the connection you want to remove and click this button.
▾	Menu	Allows you to select an option to display the selected connection by IP address, host name or alias.

Table continues on the next page...

Table 6-1. Configuration View - Connection Pane Options (continued)

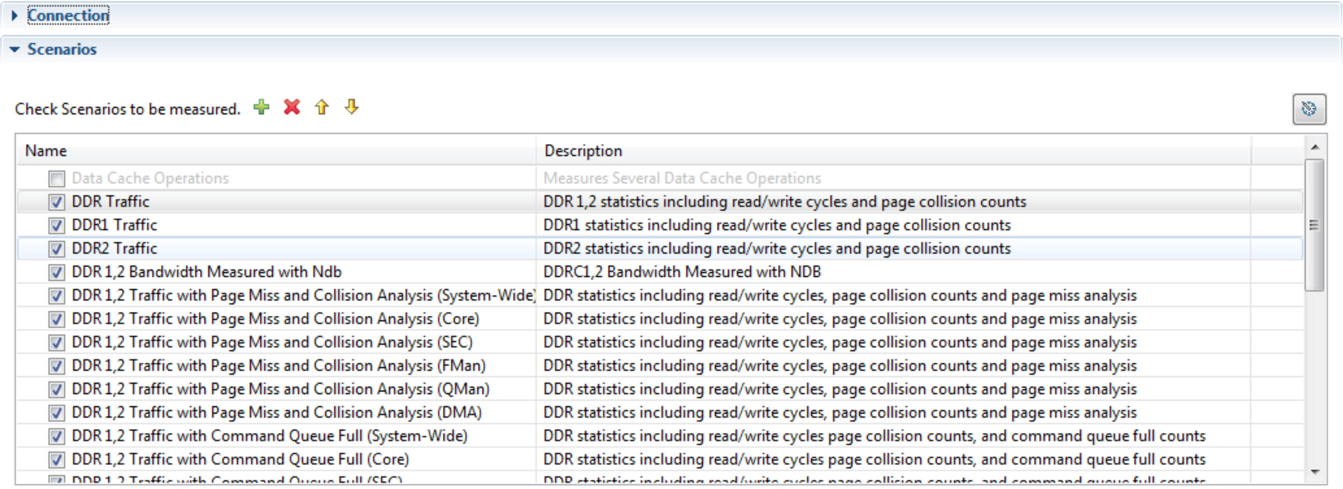
Option	Name	Description
	Processor	Displays the target processor. This determines what metrics and scenarios will be available to measure.

6.1.2.2 Scenarios Pane

The **Scenarios** pane lets you define the scenarios that will be measured.

A scenario contains the following:

- Events - Hardware occurrences that can be counted to any event in the B4 processor, including the output of other counters in a feedback loop.
- Counters - Counters that can be connected to any event in the B4 processor, including the output of other counters in a feedback loop.
- Metrics - A mathematical combination of counters (which count events) to provide an additional answer.



Check Scenarios to be measured.

Name	Description
□ Data Cache Operations	Measures Several Data Cache Operations
☑ DDR Traffic	DDR 1,2 statistics including read/write cycles and page collision counts
☑ DDR1 Traffic	DDR1 statistics including read/write cycles and page collision counts
☑ DDR2 Traffic	DDR2 statistics including read/write cycles and page collision counts
☑ DDR 1,2 Bandwidth Measured with Ndb	DDRC1,2 Bandwidth Measured with NDB
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (System-Wide)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (Core)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (SEC)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (FMan)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (QMan)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Page Miss and Collision Analysis (DMA)	DDR statistics including read/write cycles, page collision counts and page miss analysis
☑ DDR 1,2 Traffic with Command Queue Full (System-Wide)	DDR statistics including read/write cycles page collision counts, and command queue full counts
☑ DDR 1,2 Traffic with Command Queue Full (Core)	DDR statistics including read/write cycles page collision counts, and command queue full counts
☑ DDR 1,2 Traffic with Command Queue Full (SEC)	DDR statistics including read/write cycles page collision counts, and command queue full counts

Figure 6-4. Configuration View - Scenarios Pane

NOTE

For B4 devices, 2 DDR Controllers are supported to measure DDR bandwidth with NDB (NXC Data Beats) counter.

The following table lists the options available in the **Scenarios** pane.

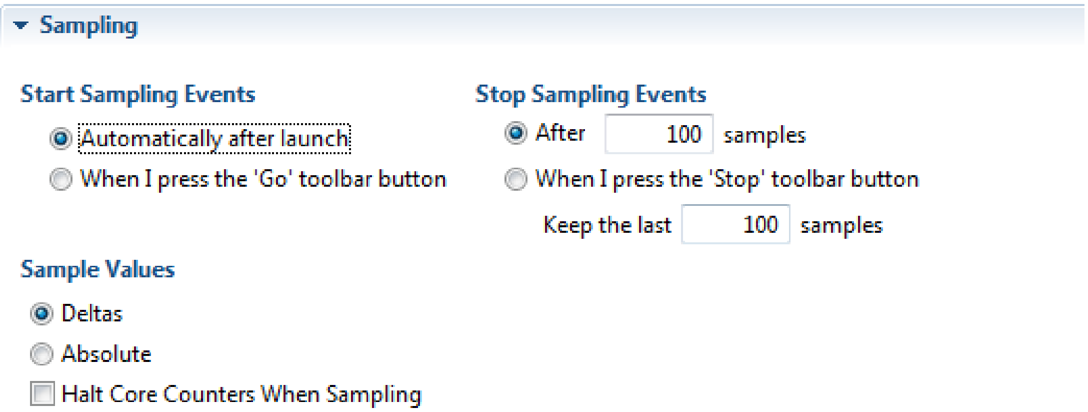
Table 6-2. Configuration View - Scenarios Pane Options

Option	Name	Description
➕	Add a Stock Scenario	Lets you select different stock scenarios. A stock scenario is the standard scenario available on the tools. Refer Selecting Scenarios for more details.
✖	Delete a Stock Scenario	Removes the selected scenario from the list.
⇧	Move Scenarios Up	Moves scenarios up in the order of execution. This option is only available when multiple scenarios are added. The scenarios run in the order they appear in the Scenarios view. All scenarios are independent of each other.
⇩	Move Scenarios Down	Moves scenarios down in the order of execution. This option is only available when multiple scenarios are added. The scenarios run in the order they appear in the Scenarios view. All scenarios are independent of each other.
⚙	Show/Hide Advanced Options	When clicked to show advanced options, adds the **Configure Counters** column to the right of the **Description** column. The **Configure Counters** column displays the checkboxes against each scenario. If checked, the hardware is configured to measure the selected scenario when the profiling session starts. If unchecked, no hardware configuration happens, that is the previous hardware settings remain active and the current values of the counters are read.

6.1.2.3 Sampling Pane

The **Sampling** pane is used to configure the events sampling or collection settings.

The following figure shows an example of Sampling pane.



Sampling

Start Sampling Events

● Automatically after launch

○ When I press the 'Go' toolbar button

Stop Sampling Events

● After 100 samples

○ When I press the 'Stop' toolbar button

Keep the last 100 samples

Sample Values

● Deltas

○ Absolute

☐ Halt Core Counters When Sampling

Figure 6-5. Configuration View - Sampling Pane

The table below lists the options available in the **Sampling** pane of the **Configuration** view.

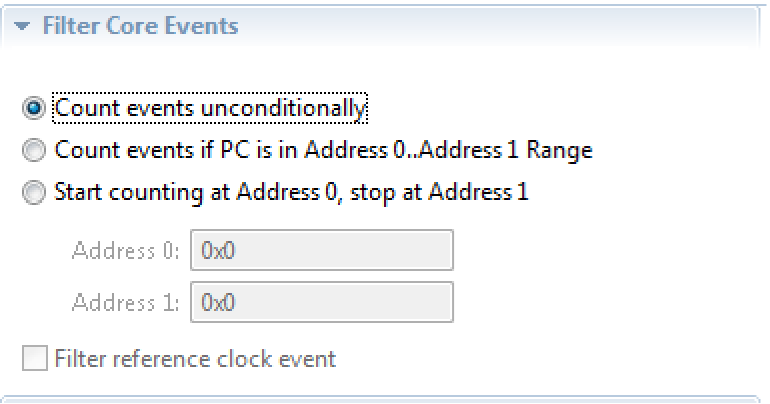
Table 6-3. Options Available in Sampling Pane

Group	Option	Description
Start Sampling Events		Lets you specify the settings for starting the event collection.
	Automatically after launch	If selected, starts collecting events automatically when **Profile** button is clicked. This button is used to launch profiling to measure scenarios. For details, refer Running Scenarios.
	When I press the `Go' toolbar button	If selected, waits for you to click the **Go** button after **Profile** button is clicked. The **Go** button is present in the Performance Analysis Session View.
Stop Sampling Events		Lets you specify the settings for stopping the event collection.
	After *<number>* samples	Stops event collection or scenario measurement after collecting a specified number of samples.
	When I press the `Stop' toolbar button. Keep the last *<number>* samples	Stops event collection after you click the **Stop** button in the Performance Analysis Session View. The number of last collected samples that will be displayed need to be specified in the textbox.
Sample Values		Lets you collect delta or absolute values from the target.

6.1.2.4 Filter Core Events pane

The **Filter Core Events** pane is used to limit the data collection between two addresses.

For example, if you want the branch miss ration event generated only for a specific part of the code.


Filter Core Events

● Count events unconditionally

○ Count events if PC is in Address 0..Address 1 Range

○ Start counting at Address 0, stop at Address 1

Address 0: 0x0

Address 1: 0x0

□ Filter reference clock event

Figure 6-6. Configuration view - Filter Core Events pane

The table below lists the options available in the **Filter Core Events** pane of the **Configuration** view.

Table 6-4. Options available in Filter Core Events pane

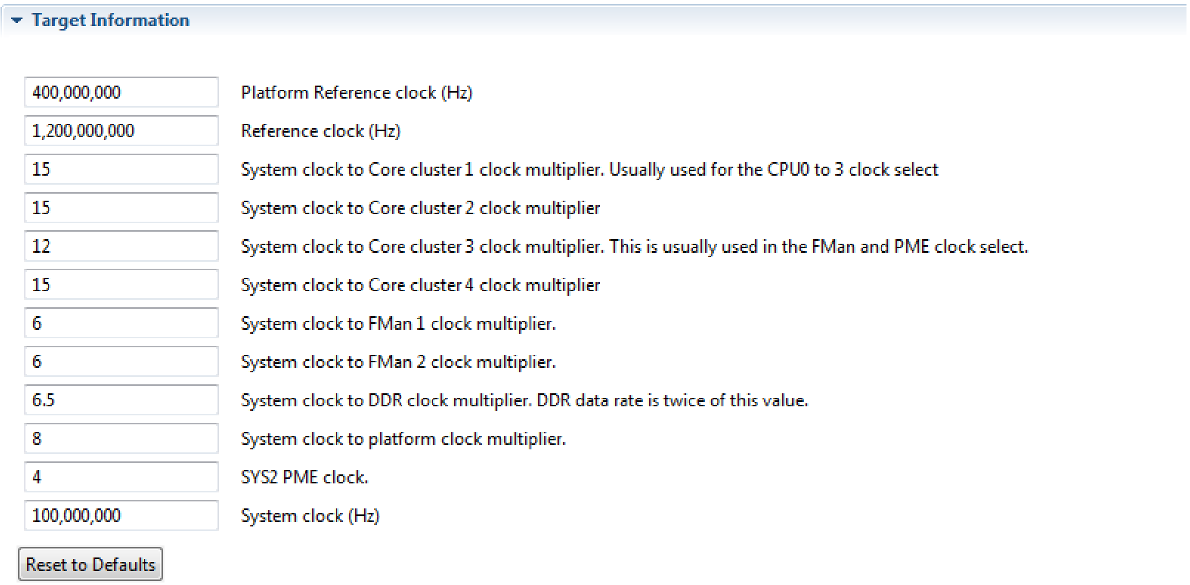
Option	Description
Count events unconditionally	Collects events without specifying any range in the code.
Count events if PC is in Address 0..Address 1 Range	Counts events within a specified address range only.
Start counting at Address 0, stop at Address 1	Starts counting when address 0 is hit and stops at address 1.
Filter reference clock event	Allows you to filter the measurements on reference clock. This option becomes active only when you define a filter using any one of the above options, and if selected, applies that filter to the reference clock measurement.

6.1.2.5 Target Information Pane

The **Target Information** pane allows you to set the clock frequencies and multipliers on the target board.

These frequencies are used to compute complex metrics that imply reporting the number of events to a certain time frame, for example, computing the DDR traffic. The accuracy of the results depends on the correctness of the frequency values you specify.

The default values are displayed in the respective textboxes, you can modify them according to the speed on a particular target board.



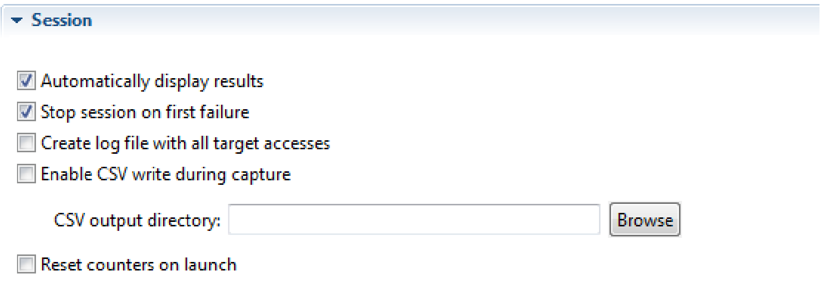
Target Information

400,000,000	Platform Reference clock (Hz)
1,200,000,000	Reference clock (Hz)
15	System clock to Core cluster1 clock multiplier. Usually used for the CPU0 to 3 clock select
15	System clock to Core cluster2 clock multiplier
12	System clock to Core cluster3 clock multiplier. This is usually used in the FMan and PME clock select.
15	System clock to Core cluster4 clock multiplier
6	System clock to FMan 1 clock multiplier.
6	System clock to FMan 2 clock multiplier.
6.5	System clock to DDR clock multiplier. DDR data rate is twice of this value.
8	System clock to platform clock multiplier.
4	SYS2 PME clock.
100,000,000	System clock (Hz)

Figure 6-7. Configuration View - Target Information Pane

6.1.2.6 Session pane

The **Session** pane lets you select the settings to be applied while running the scenario measurement session.



Session

☑ Automatically display results
☑ Stop session on first failure
□ Create log file with all target accesses
□ Enable CSV write during capture

CSV output directory:

□ Reset counters on launch

Figure 6-8. Configuration view - Session pane

The table below lists the options available in the **Session** pane of the **Configuration** view.

Table 6-5. Options available in Session pane

Option	Description
Automatically display all results	If selected, displays the generated data automatically after data measurement or when collection stops.
Stop session on first failure	If selected, stops collection of data when a failure is detected, for example, the application unable to configure some counters for measurement.

Table continues on the next page...

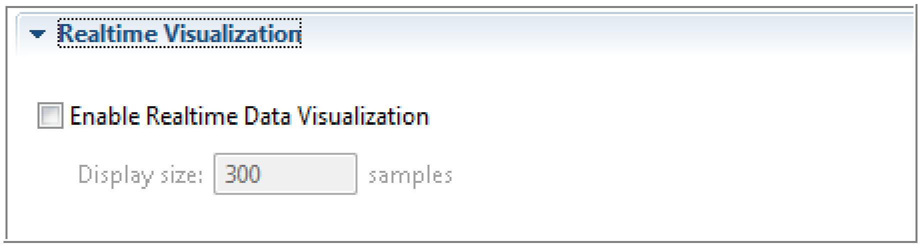
Table 6-5. Options available in Session pane (continued)

Option	Description
Create log file with all target accesses	If selected, creates a log file which might be useful in case of a debug failure. You can view the log file in the **Target Access Log** tab, which appears (at the bottom of the results screen) after running a collection.
Enable CSV write during capture	If selected, allows you to save the data in CSV output file. You can select the CSV output directory using the **Browse** button.
Reset counters on launch	If selected, resets counters on the target when launching a new measurement session.

6.1.2.7 Realtime Visualization

The **Realtime Visualization** pane allows you to view the real time data of specified number of samples.

The following figure shown an example of Realtime Visualization pane.



Realtime Visualization

☐ Enable Realtime Data Visualization

Display size: 300 samples

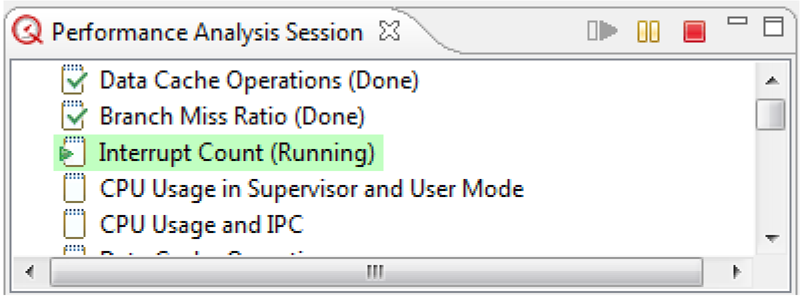
Figure 6-9. Configuration View - Realtime Visualization Pane

Check **Enable Realtime Data Visualization** checkbox to edit the number of samples to be displayed.

6.1.3 Performance Analysis Session View

The **Performance Analysis Session** view displays the status of all the events that are being generated when collection starts.

Once an event is generated successfully, the status is shown as **Done**. For pending events, the status is shown as **Running**.



Performance Analysis Session

Data Cache Operations (Done)

Branch Miss Ratio (Done)

Interrupt Count (Running)

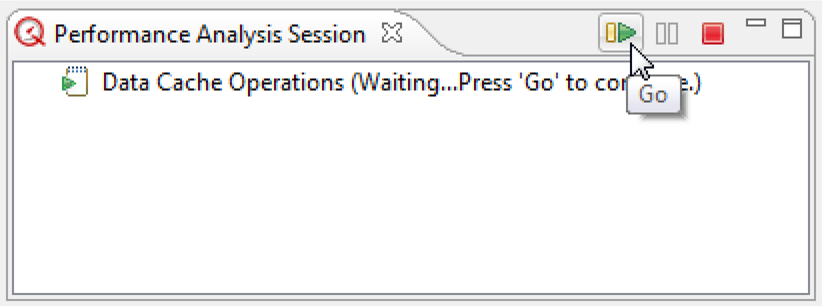
CPU Usage in Supervisor and User Mode

CPU Usage and IPC

Figure 6-10. Performance Analysis Session View

The buttons available in the **Performance Analysis Session** view are:

- **Go** - Click this button to start data collection if you have selected the **When I press the `Go' toolbar button** option in the Sampling Pane, else the data collection starts automatically after pressing the **Profile** button.



Performance Analysis Session

Data Cache Operations (Waiting...Press 'Go' to co

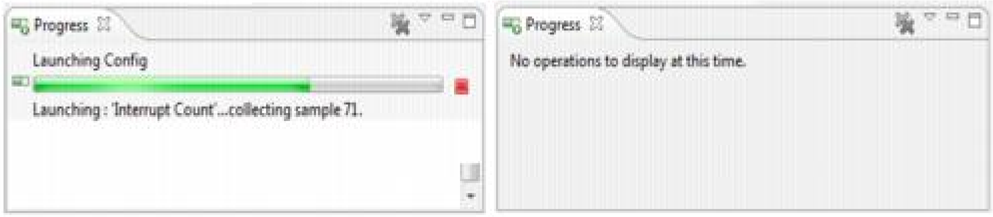
Figure 6-11. Go Button to Start Data Collection

- **Pause** - Click this button to pause data collection.
- **Stop** - Click this button to stop data collection.

6.1.4 Progress View

The **Progress** view of the **Performance Analysis** perspective displays the progress of the launching configuration.

The progress bar continues as events generate one by one. After complete generation, it displays the status as `No operations to display at this time'.



Launching Config

Launching : 'Interrupt Count'...collecting sample 71.

No operations to display at this time.

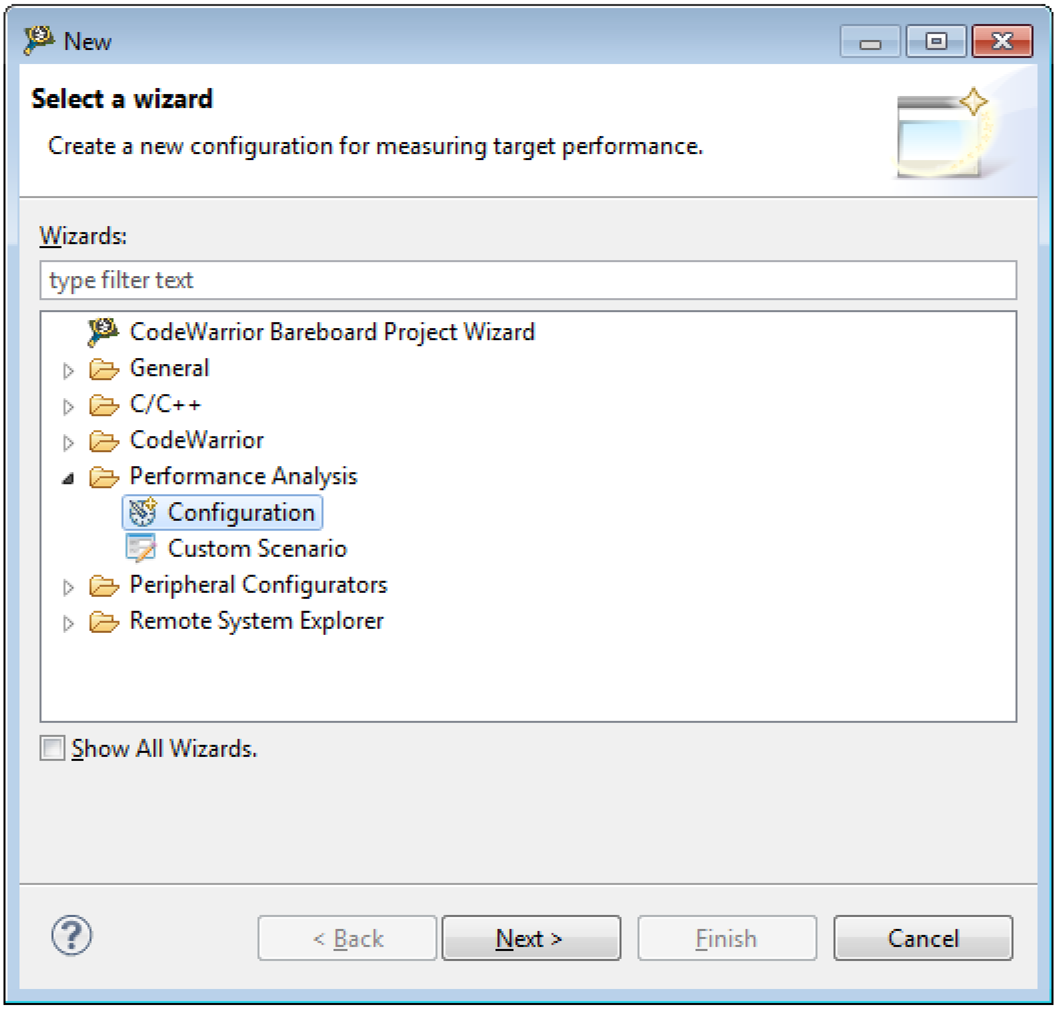
Figure 6-12. Progress View

6.2 Creating New Configuration

This section explains how to create a new configuration for Performance Analysis.

Perform the following steps to create a new configuration:

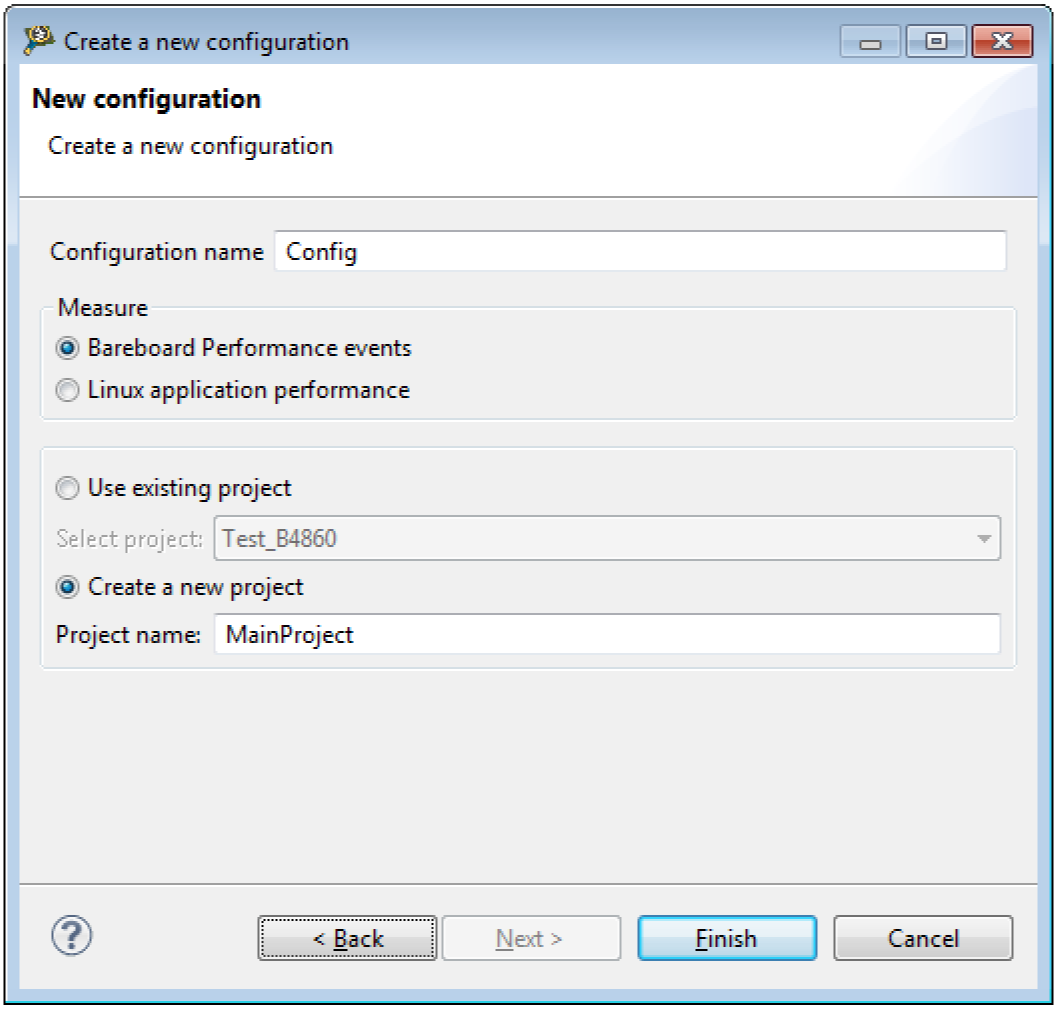
1. Select **Window > Open Perspective > Other > Performance Analysis** from the CodeWarrior IDE menu bar to open the **Performance Analysis** perspective.
2. Select **File > New > Other** to open the **New** wizard.



Create a new configuration for measuring target performance.

Figure 6-13. New Wizard

3. Select **Performance Analysis > Configuration** to open the **New Configuration** screen. Alternatively, click the icon from the main menu bar to get the **New Configuration** screen directly.



Create a new configuration

New configuration

Create a new configuration

Configuration name: Config

Measure

● Bareboard Performance events

○ Linux application performance

○ Use existing project

Select project: Test_B4860

● Create a new project

Project name: MainProject

Figure 6-14. New Configuration Screen

4. Type a name for the configuration in the **Configuration name** textbox.
5. Select **Bareboard Performance events** or **Linux application performance** depending upon the requirements.
6. Type a name for the project in the **Project name** textbox. This is the project in which your Performance Analysis configuration is defined. You can create the Performance Analysis configuration either in the existing project or in the new project. If you are creating the Performance Analysis project for the first time, the **Use existing project** option is disabled.
7. Click **Finish**.

   The Performance Analysis project along with its configuration is created and appears in the **Performance Analysis** view.

6.3 Connecting to Board

You need to select a connection to the board after creating a Performance Analysis project.

You can select a connection to the board by manually configuring the connection.

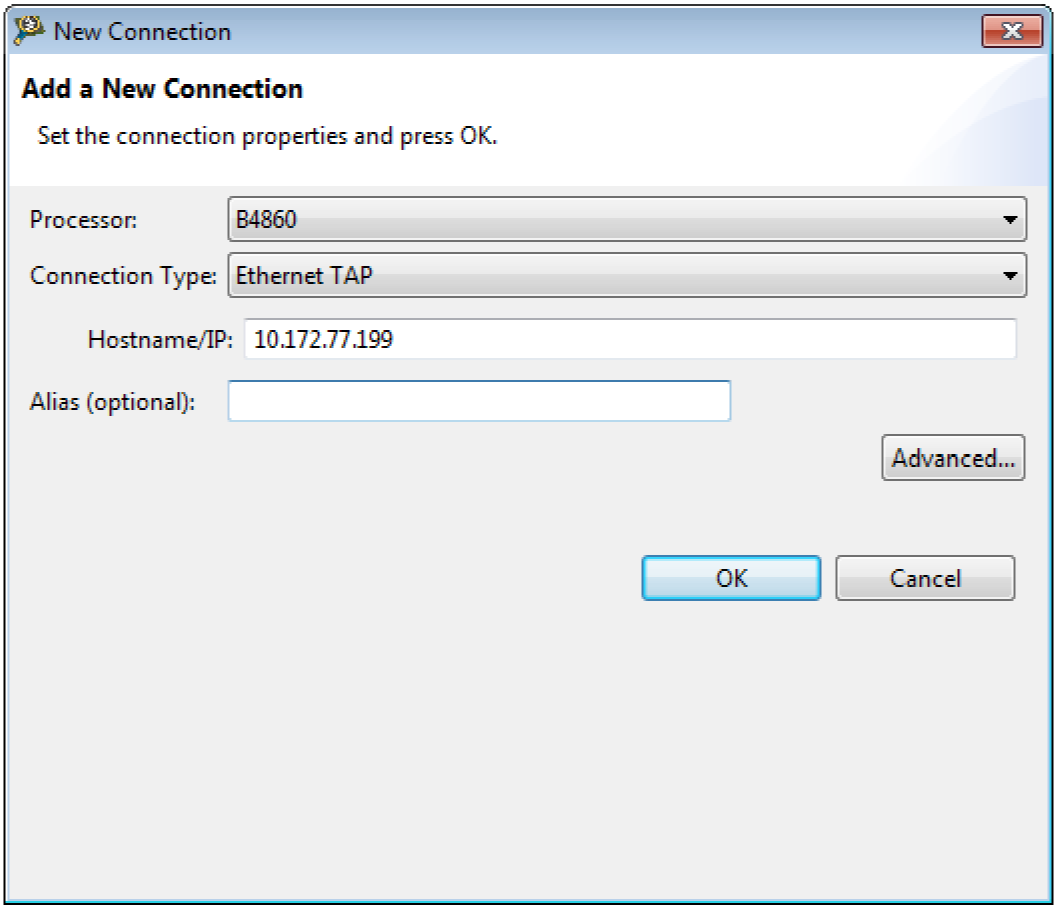
- Adding Connection Manually

6.3.1 Adding Connection Manually

This section explains the steps required to manually configure the connection with the board.

Follow the steps to manually add a connection to the board:

1. Click the **New Connection** icon ➕ to open the **Add a New Connection** screen.
2. Select the processor and connection type in the respective fields.
3. Provide hostname or IP address depending on the connection type selected.



New Connection

Add a New Connection

Set the connection properties and press OK.

Processor: B4860

Connection Type: Ethernet TAP

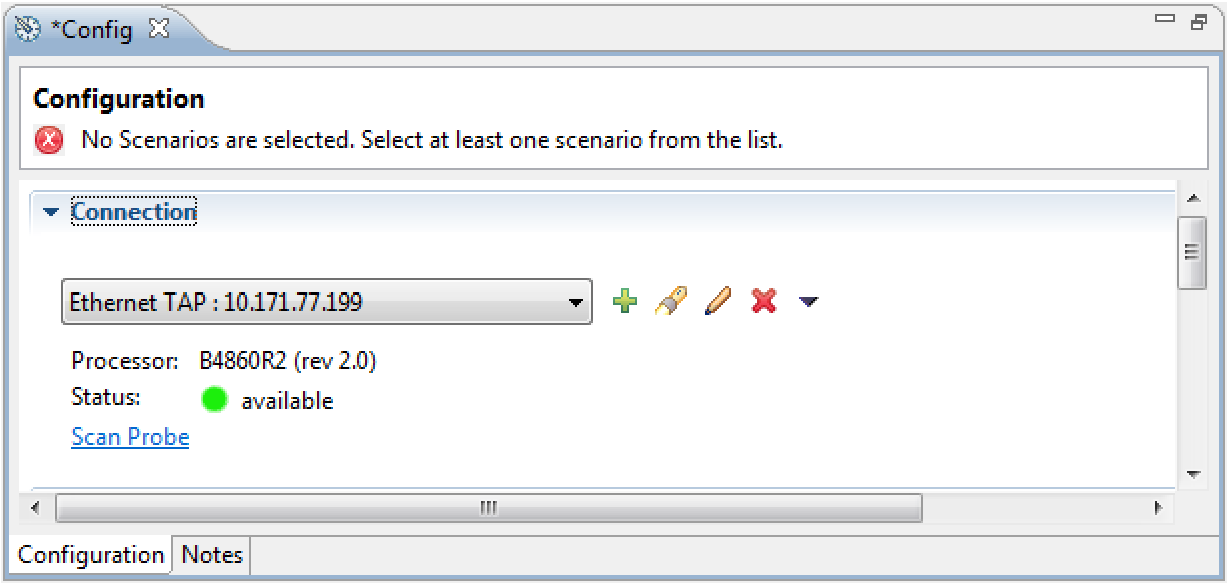
Hostname/IP: 10.172.77.199

Alias (optional):

Figure 6-15. Add a New Connection Screen

4. Specify an alias for the connection in the **Alias** textbox. This field is optional.
5. Click **OK** to save the settings.

   The connection is set and selected in the **Configuration** view.



*Config

Configuration

No Scenarios are selected. Select at least one scenario from the list.

Connection

Ethernet TAP : 10.171.77.199

Processor: B4860R2 (rev 2.0)

Status: available

Figure 6-16. Connection Added

NOTE

If the connection displays unknown status, click the **Scan Probe** hyperlink to scan the probe connected to the processor.

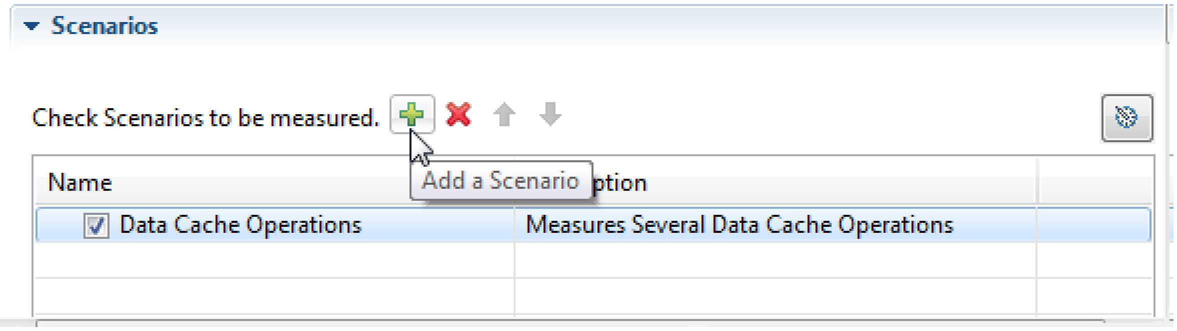
6. Click **File > Save** to save your configuration.

6.4 Selecting Scenarios

This section explains the steps required to select and add a scenario to the Performance Analysis configuration.

Perform the following steps to select and add a scenario:

1. Click **Add a Scenario** in the **Scenarios** section of the **Configuration** view.

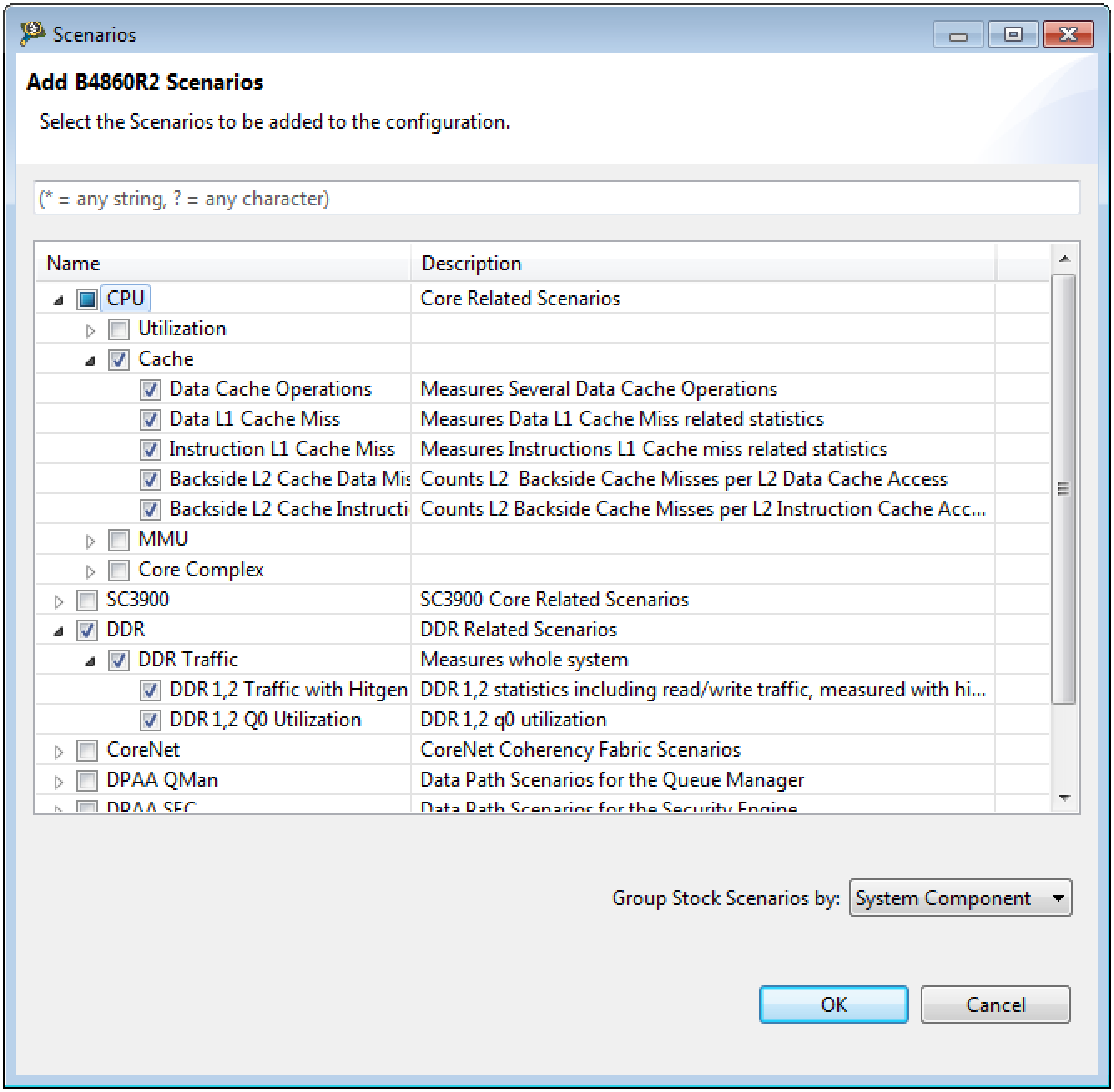


Name	Description
☑ Data Cache Operations	Measures Several Data Cache Operations

Figure 6-17. Add a Scenario

The **Add <Processor> Scenarios** screen appears in which only those scenarios that are applicable to the selected processor are displayed.

2. Select the subsystem that you want to analyze.



Add B4860R2 Scenarios

Select the Scenarios to be added to the configuration.

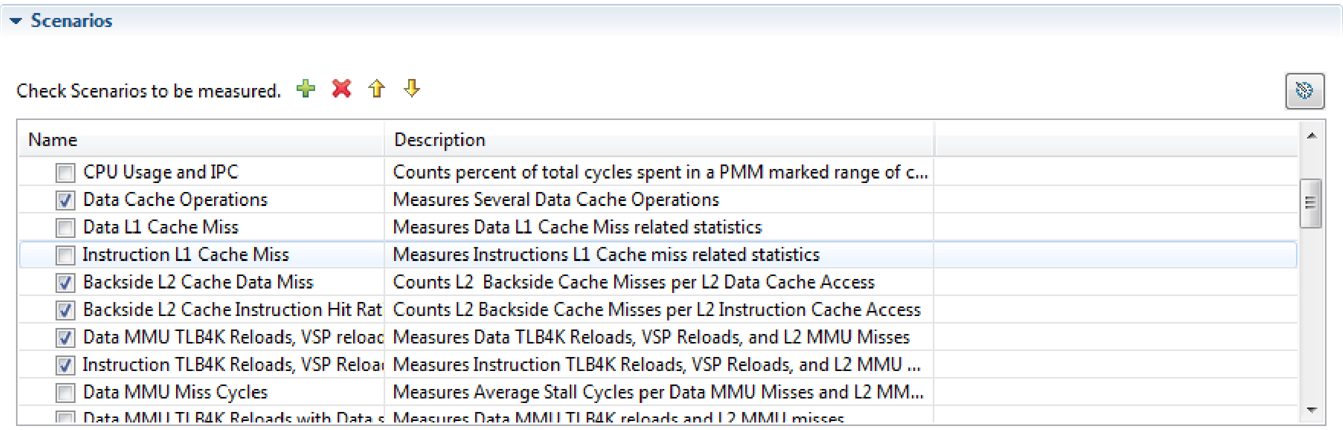
(* = any string, ? = any character)

Name	Description
◢ ■ CPU	Core Related Scenarios
▷ □ Utilization	
◢ ☑ Cache	
☑ Data Cache Operations	Measures Several Data Cache Operations
☑ Data L1 Cache Miss	Measures Data L1 Cache Miss related statistics
☑ Instruction L1 Cache Miss	Measures Instructions L1 Cache miss related statistics
☑ Backside L2 Cache Data Mis	Counts L2 Backside Cache Misses per L2 Data Cache Access
☑ Backside L2 Cache Instructi	Counts L2 Backside Cache Misses per L2 Instruction Cache Acc...
▷ □ MMU	
▷ □ Core Complex	
▷ □ SC3900	SC3900 Core Related Scenarios
◢ ☑ DDR	DDR Related Scenarios
◢ ☑ DDR Traffic	Measures whole system
☑ DDR 1,2 Traffic with Hitgen	DDR 1,2 statistics including read/write traffic, measured with hi...
☑ DDR 1,2 Q0 Utilization	DDR 1,2 q0 utilization
▷ □ CoreNet	CoreNet Coherency Fabric Scenarios
▷ □ DPAA QMan	Data Path Scenarios for the Queue Manager
▷ □ DPAA SEC	Data Path Scenarios for the Security Engine

Group Stock Scenarios by: System Component

Figure 6-18. Add <Processor> Scenarios Screen

3. Click **OK** to add the selected scenarios to the configuration.



Scenarios

Check Scenarios to be measured.

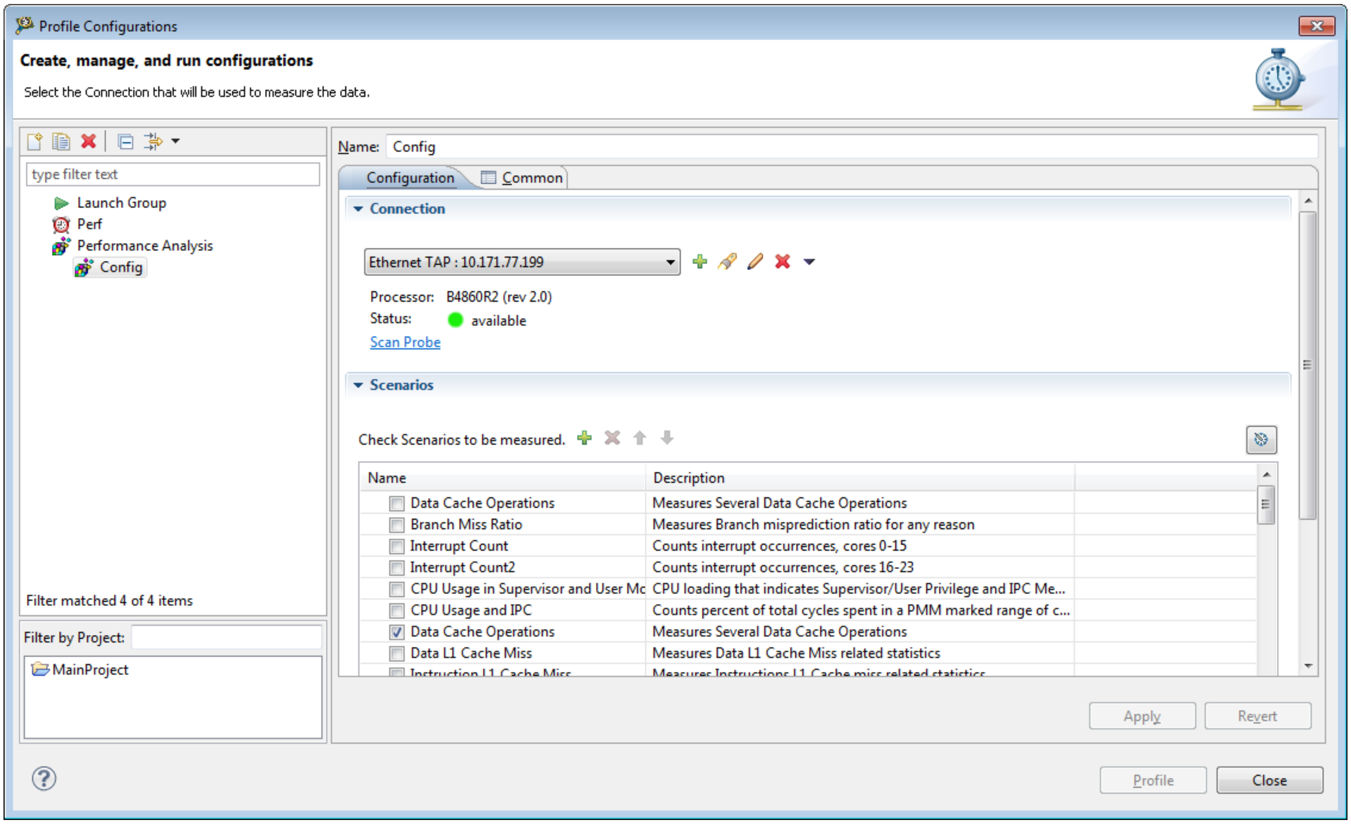
	Name	Description
☐	CPU Usage and IPC	Counts percent of total cycles spent in a PMM marked range of c...
☑	Data Cache Operations	Measures Several Data Cache Operations
☐	Data L1 Cache Miss	Measures Data L1 Cache Miss related statistics
☐	Instruction L1 Cache Miss	Measures Instructions L1 Cache miss related statistics
☑	Backside L2 Cache Data Miss	Counts L2 Backside Cache Misses per L2 Data Cache Access
☑	Backside L2 Cache Instruction Hit Rat	Counts L2 Backside Cache Misses per L2 Instruction Cache Access
☑	Data MMU TLB4K Reloads, VSP reloac	Measures Data TLB4K Reloads, VSP Reloads, and L2 MMU Misses
☑	Instruction TLB4K Reloads, VSP Reloa	Measures Instruction TLB4K Reloads, VSP Reloads, and L2 MMU ...
☐	Data MMU Miss Cycles	Measures Average Stall Cycles per Data MMU Misses and L2 MM...
☐	Data MMU TLB4K Reloads with Data s	Measures Data MMU TLB4K reloads and L2 MMU misses

Figure 6-19. Scenarios Added to Configuration

4. Click **File > Save** to save your configuration.

You can also modify your configuration settings such as configuring board connection or adding scenarios using the **Profile Configurations** dialog box. The **Profile Configurations** dialog box appears on selecting **Run > Profile Configurations** from the CodeWarrior IDE menu bar.

Select your configuration on left side, specify the required settings, and click **Apply** to save the settings.



Profile Configurations

Create, manage, and run configurations

Select the Connection that will be used to measure the data.

type filter text

- Launch Group
- Perf
- Performance Analysis
  - Config

Filter matched 4 of 4 items

Filter by Project:

MainProject

Name: Config

Configuration | Common

Connection

Ethernet TAP : 10.171.77.199

Processor: B4860R2 (rev 2.0)

Status: ● available

Scan Probe

Scenarios

Check Scenarios to be measured.

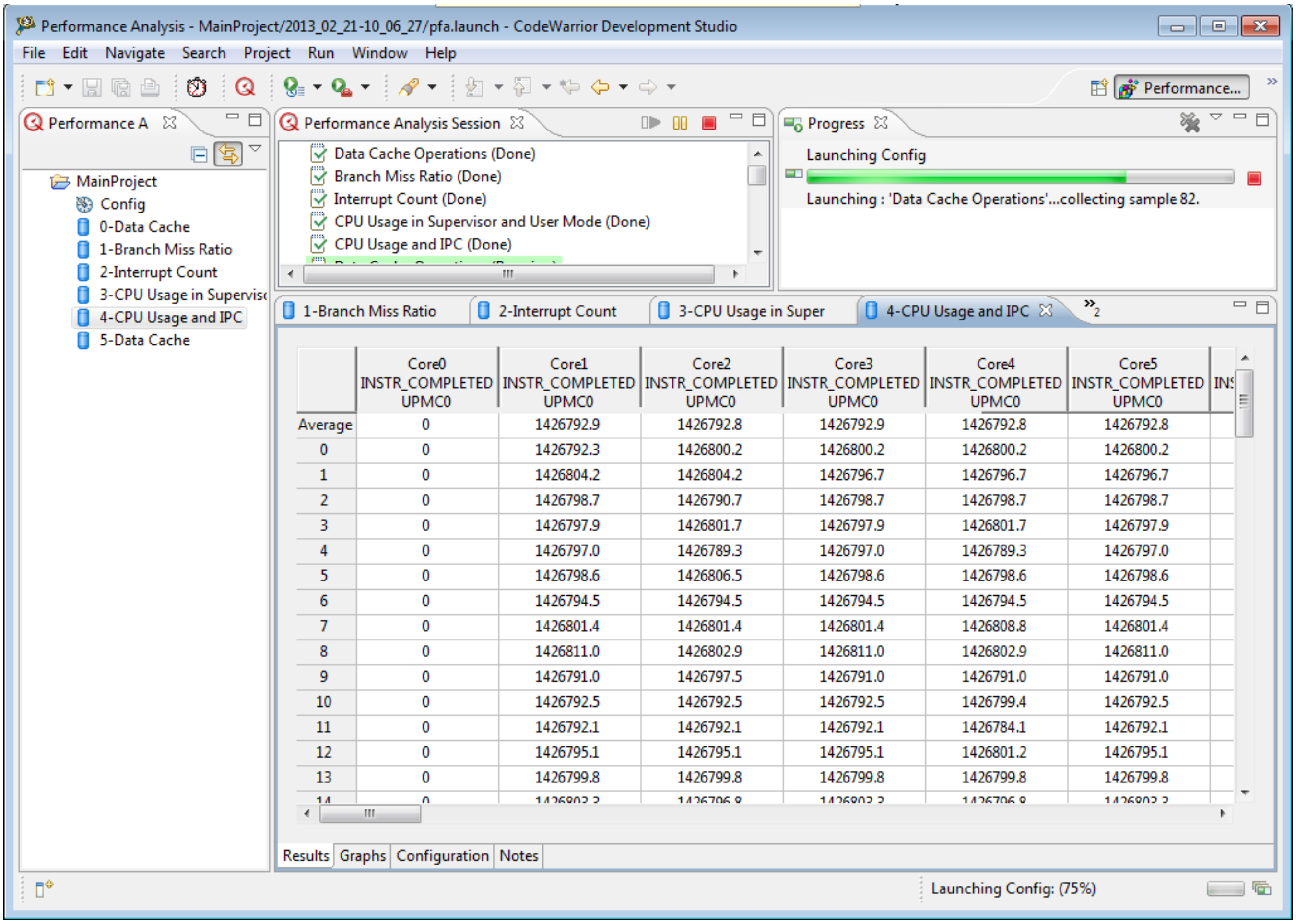
	Name	Description
☐	Data Cache Operations	Measures Several Data Cache Operations
☐	Branch Miss Ratio	Measures Branch misprediction ratio for any reason
☐	Interrupt Count	Counts interrupt occurrences, cores 0-15
☐	Interrupt Count2	Counts interrupt occurrences, cores 16-23
☐	CPU Usage in Supervisor and User Mc	CPU loading that indicates Supervisor/User Privilege and IPC Me...
☐	CPU Usage and IPC	Counts percent of total cycles spent in a PMM marked range of c...
☑	Data Cache Operations	Measures Several Data Cache Operations
☐	Data L1 Cache Miss	Measures Data L1 Cache Miss related statistics
☐	Instruction L1 Cache Miss	Measures Instructions L1 Cache miss related statistics

Figure 6-20. Profile Configuration Dialog Box

6.5 Running Scenarios

This section explains about running the scenerios and veiwing the result in the Performance Analysis view.

To run the selected scenarios, select **Run > Profile** from the CodeWarrior IDE menu bar to launch profiling. You can also click the **Profile** button in the main toolbar of the CodeWarrior IDE. Alternatively, you can click **Profile** in the **Profile Configurations** dialog box, which appears on selecting **Run > Profile Configurations** from the CodeWarrior IDE menu bar.



Data Cache Operations (Done)
Branch Miss Ratio (Done)
Interrupt Count (Done)
CPU Usage in Supervisor and User Mode (Done)
CPU Usage and IPC (Done)

Launching Config

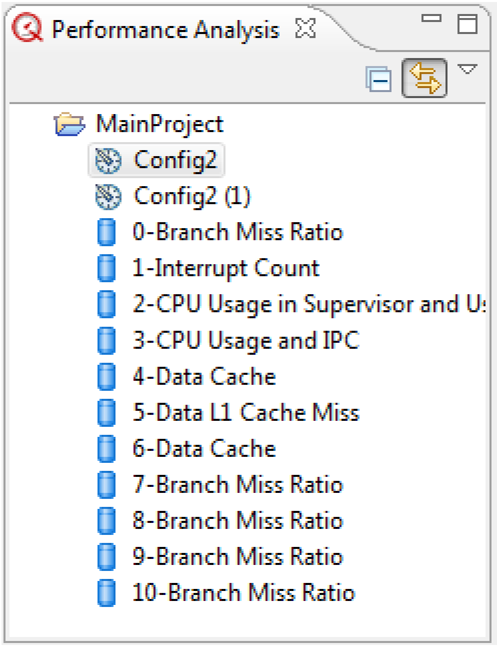
Launching : 'Data Cache Operations'...collecting sample 82.

	Core0 INSTR_COMPLETED UPMC0	Core1 INSTR_COMPLETED UPMC0	Core2 INSTR_COMPLETED UPMC0	Core3 INSTR_COMPLETED UPMC0	Core4 INSTR_COMPLETED UPMC0	Core5 INSTR_COMPLETED UPMC0	INS
Average	0	1426792.9	1426792.8	1426792.9	1426792.8	1426792.8	
0	0	1426792.3	1426800.2	1426800.2	1426800.2	1426800.2	
1	0	1426804.2	1426804.2	1426796.7	1426796.7	1426796.7	
2	0	1426798.7	1426790.7	1426798.7	1426798.7	1426798.7	
3	0	1426797.9	1426801.7	1426797.9	1426801.7	1426797.9	
4	0	1426797.0	1426789.3	1426797.0	1426789.3	1426797.0	
5	0	1426798.6	1426806.5	1426798.6	1426798.6	1426798.6	
6	0	1426794.5	1426794.5	1426794.5	1426794.5	1426794.5	
7	0	1426801.4	1426801.4	1426801.4	1426808.8	1426801.4	
8	0	1426811.0	1426802.9	1426811.0	1426802.9	1426811.0	
9	0	1426791.0	1426797.5	1426791.0	1426791.0	1426791.0	
10	0	1426792.5	1426792.5	1426792.5	1426799.4	1426792.5	
11	0	1426792.1	1426792.1	1426792.1	1426784.1	1426792.1	
12	0	1426795.1	1426795.1	1426795.1	1426801.2	1426795.1	
13	0	1426799.8	1426799.8	1426799.8	1426799.8	1426799.8	
14	0	1426803.3	1426796.8	1426803.3	1426796.8	1426803.3	

Launching Config: (75%)

Figure 6-21. Profiling/Measurement in Progress

The **Progress View** provides a summary of the operations being carried out. Once the measurements are complete, the results are collected as part of the project and appear in the project folder in the **Performance Analysis** view.



Performance Analysis

MainProject

Config2

Config2 (1)

0-Branch Miss Ratio

1-Interrupt Count

2-CPU Usage in Supervisor and U:

3-CPU Usage and IPC

4-Data Cache

5-Data L1 Cache Miss

6-Data Cache

7-Branch Miss Ratio

8-Branch Miss Ratio

9-Branch Miss Ratio

10-Branch Miss Ratio

Figure 6-22. Measurement Results

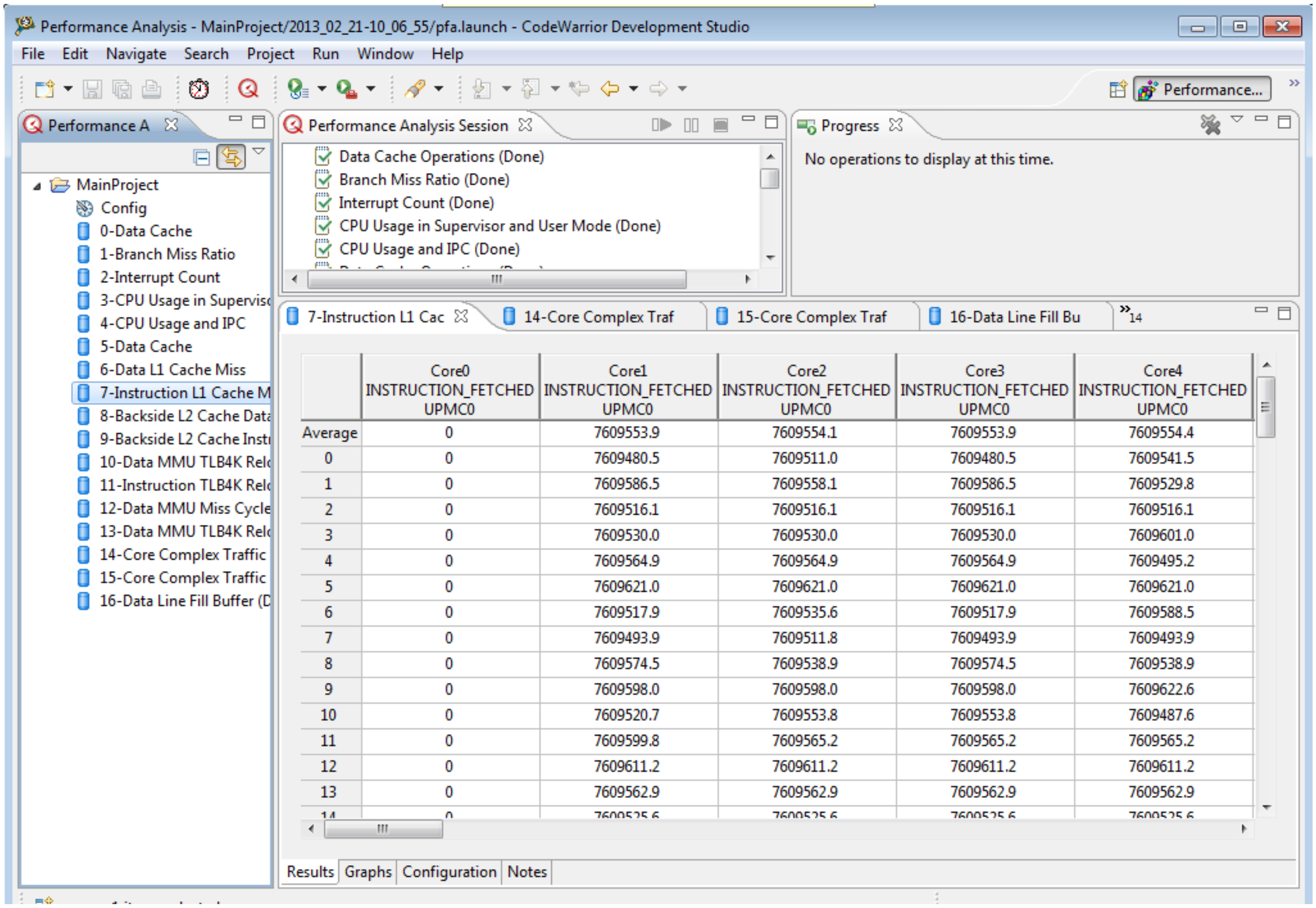
6.6 Viewing Results

You can view the scenario measurement results either in tabular or graphical format.

To view the results, double-click the collections associated with your project in the **Performance Analysis** view. The generated data appears displaying a summary of the metrics and scenarios collected in a tabular format as shown in the following figure.

NOTE

If you have kept the **Automatically display all results** checkbox checked in the **Session** pane of the **Configuration** view, the results will get open automatically while data is collecting.



	Core0 INSTRUCTION_FETCHED UPMC0	Core1 INSTRUCTION_FETCHED UPMC0	Core2 INSTRUCTION_FETCHED UPMC0	Core3 INSTRUCTION_FETCHED UPMC0	Core4 INSTRUCTION_FETCHED UPMC0
Average	0	7609553.9	7609554.1	7609553.9	7609554.4
0	0	7609480.5	7609511.0	7609480.5	7609541.5
1	0	7609586.5	7609558.1	7609586.5	7609529.8
2	0	7609516.1	7609516.1	7609516.1	7609516.1
3	0	7609530.0	7609530.0	7609530.0	7609601.0
4	0	7609564.9	7609564.9	7609564.9	7609495.2
5	0	7609621.0	7609621.0	7609621.0	7609621.0
6	0	7609517.9	7609535.6	7609517.9	7609588.5
7	0	7609493.9	7609511.8	7609493.9	7609493.9
8	0	7609574.5	7609538.9	7609574.5	7609538.9
9	0	7609598.0	7609598.0	7609598.0	7609622.6
10	0	7609520.7	7609553.8	7609553.8	7609487.6
11	0	7609599.8	7609565.2	7609565.2	7609565.2
12	0	7609611.2	7609611.2	7609611.2	7609611.2
13	0	7609562.9	7609562.9	7609562.9	7609562.9
14	0	7609525.6	7609525.6	7609525.6	7609525.6

Figure 6-23. Data Collection Results

At the bottom of the table view, there are four tabs:

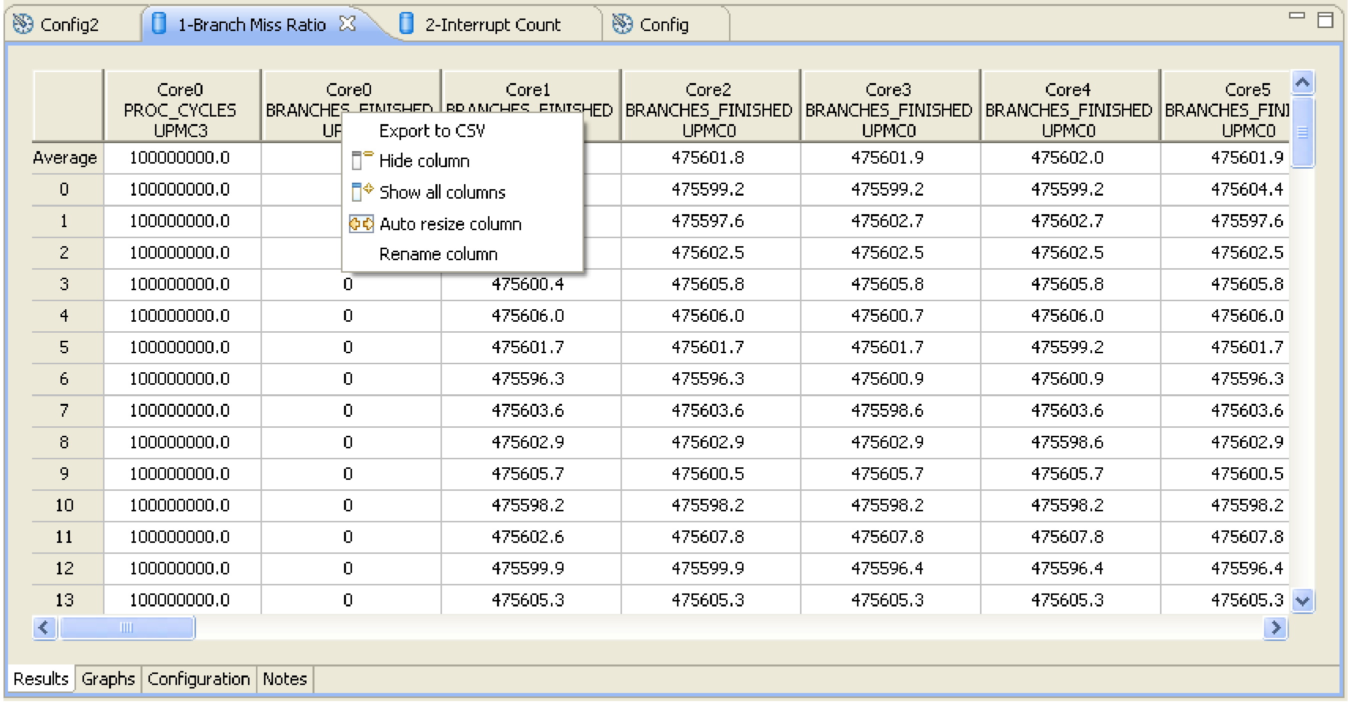
- **Results** - Displays the default view showing the event data generated from the Performance Analysis tool. For details, refer Tabular Format.
- **Graphs** - Displays a graphical view of the data. For details, refer Graphical Format.
- **Configuration** - Displays a summary of the connection to the target.
- **Notes** - Allows you to write your own notes corresponding to the data results. The notes that you write get automatically saved which can be viewed later when you open the corresponding data results.
- **Delete** - Deletes the selected result.
- **Rename** - Lets you rename the selected result.
- **Import** - Lets you import the selected result into another project.
- **Export** - Lets you export results from another project.

6.6.1 Tabular Format

The tabular format of the results generated from the Performance Analysis tool displays the number of samples in rows and the number of events in columns.

There is one event per core generated in each column.

If you want to view event data of a particular column only, you can hide other columns. The columns hidden in the **Results** view are visible in the **Graphs** view.



	Core0 PROC_CYCLES UPMC3	Core0 BRANCHES_FINISHED UP	Core1 BRANCHES_FINISHED	Core2 BRANCHES_FINISHED UPMC0	Core3 BRANCHES_FINISHED UPMC0	Core4 BRANCHES_FINISHED UPMC0	Core5 BRANCHES_FINI UPMC0
Average	100000000.0			475601.8	475601.9	475602.0	475601.9
0	100000000.0			475599.2	475599.2	475599.2	475604.4
1	100000000.0			475597.6	475602.7	475602.7	475597.6
2	100000000.0			475602.5	475602.5	475602.5	475602.5
3	100000000.0	0	475600.4	475605.8	475605.8	475605.8	475605.8
4	100000000.0	0	475606.0	475606.0	475600.7	475606.0	475606.0
5	100000000.0	0	475601.7	475601.7	475601.7	475599.2	475601.7
6	100000000.0	0	475596.3	475596.3	475600.9	475600.9	475596.3
7	100000000.0	0	475603.6	475603.6	475598.6	475603.6	475603.6
8	100000000.0	0	475602.9	475602.9	475602.9	475598.6	475602.9
9	100000000.0	0	475605.7	475600.5	475605.7	475605.7	475600.5
10	100000000.0	0	475598.2	475598.2	475598.2	475598.2	475598.2
11	100000000.0	0	475602.6	475607.8	475607.8	475607.8	475607.8
12	100000000.0	0	475599.9	475599.9	475596.4	475596.4	475596.4
13	100000000.0	0	475605.3	475605.3	475605.3	475605.3	475605.3

Figure 6-24. Data Collection Results in Tabular Form

Right-click a column header to view the various options available:

- **Export to CSV** - Allows you to export the selected column into a CSV file. Click this option to open the **Export to CSV** dialog box. Specify the location and name of the file and click **Save**.
- **Hide column** - Allows you to hide the selected column. To hide multiple columns, hold the control key and click through the columns.
- **Show all columns** - Displays all columns in the table.
- **Auto resize column** - Resizes all displayed columns according to table width.
- **Rename column** - Allows you to rename the selected column.

6.6.2 Graphical Format

The graphical format allows you to view the graphical representation of the results generated from the Performance Analysis tool.

Click the **Graphs** tab to view the graphical form of the collected data.

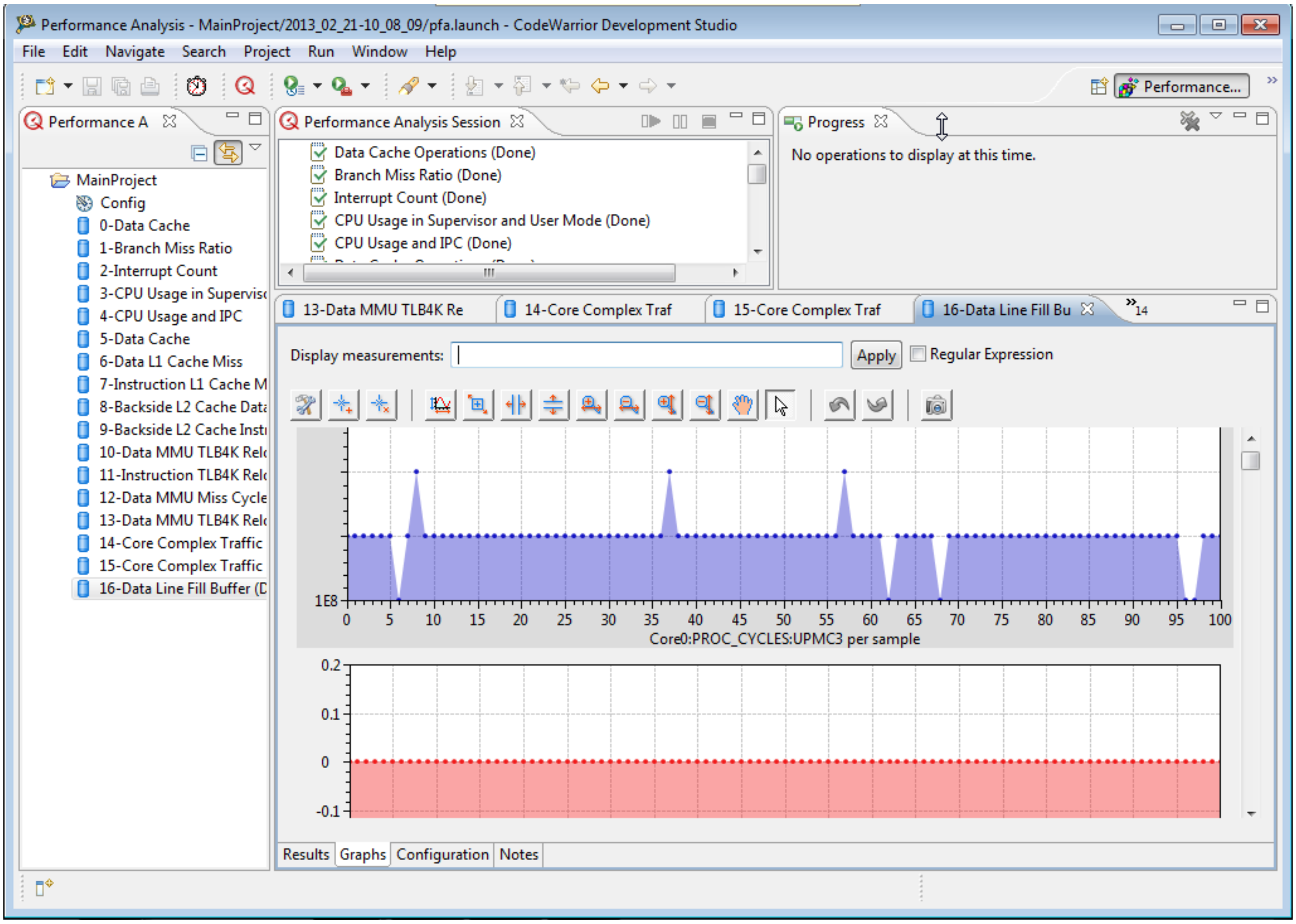


Figure 6-25. Data Collection Results in Graphical Format

A graph is generated for each event per core. You can use the scroll bar to view all the generated graphs for a particular collection. You can perform various tasks on the graphs, such as auto scaling, zooming-in/out, and panning. The table below lists the description of each option available on the graph.

Table 6-6. Options Available in Graphical Format of Collected Results

Option	Name	Description
	Display Measurements	Allows you to specify strings to filter graphs according to their titles. For example, to display graphs with title BTB_MISS_RATIO (of Branch Miss Ratio event), specify the string BTB_MISS_RATIO in the text box, and click **Apply**. Only the graphs with

Table continues on the next page...

Table 6-6. Options Available in Graphical Format of Collected Results (continued)

Option	Name	Description
		BTB_MISS_RATIO as title will be displayed. If you want a specific string to be matched, uncheck the **Regular Expression** checkbox. If you want to match a particular string, anchor the string with * on both sides. For example, **core6** (with **Regular Expressions** unchecked) will display all results corresponding to *core6* only. If you want to match a single character, anchor it with *?* on both sides. For example, *?z?*.
	Configure Settings	Allows you to configure the settings of a selected graph. Click to display the **Configure Graph Settings** dialog box, and specify the graph settings (such as title color, legend, title font), axes settings (such as title, color, grid format), and traces settings (such as name, trace type, color, line width).
	Add Annotation	Allows you to add annotations to a selected graph. Click to display the **Add Annotation** dialog box and specify the settings according to requirements.
	Remove Annotation	Lets you remove the added annotations from a selected graph.
	Perform Auto scale	Lets you adjust the x-axis and y-axis according to the range of the data in the selected graph.
	Rubberband Zoom	Lets you select a rectangular area in the graph to zoom in.
	Horizontal Zoom	Lets you specify the exact range horizontally in the graph to zoom in.
	Vertical Zoom	Lets you specify the exact range vertically in the graph to zoom in.
	Zoom In Horizontally	Zooms in horizontally in the area you clicked.
	Zoom Out Horizontally	Zooms out horizontally in the area you clicked.
	Zoom In Vertically	Zooms in vertically in the area you clicked.
	Zoom Out Vertically	Zooms out vertically in the area you clicked.
	Panning	Lets you move the selected graph to view a specific area of the graph.
	Undo Panning	Lets you undo the panning action on the graph.
	Redo Panning	Lets you redo the panning action on the graph.

Table continues on the next page...

Table 6-6. Options Available in Graphical Format of Collected Results (continued)

Option	Name	Description
📷	Save Snapshot to PNG FIle	Allows you to capture image of the selected graphs in the PNG format.

6.7 Custom scenarios

The Performance Analysis tool allows you to select predefined analysis configurations called Stock Scenarios.

A stock scenario configures events that are to be measured and predefined metrics to be calculated on the collected data.

As you become more familiar with the system that you are measuring, the ability to create you own scenarios becomes crucial. The Performance Analysis tool lets you create a custom scenario in order to minutely narrow or extend the analysis.

This topic contains the following sub-topics.

- Defining custom scenarios
- Editing custom scenarios

6.7.1 Defining custom scenarios

This section explains the steps to define a custom scenerios.

Perform the following steps to define a custom scenerio.

1. Choose **File > New > Other** to open the **New** wizard.
2. Choose **Performance Analysis > Custom Scenario** and click **Next**.

   The **New Custom Scenario** screen appears.

3. Specify the **Processor**, **Name**, and **Description** for the scenario in the respective fields.
4. Click **Next** and select an existing scenario to be used as a starting point. By default, scenarios are grouped by system component; you can choose to group the scenarios by measurement type using the **Group By** pop-up menu.
5. Click **Finish** to complete the creation of a custom scenario.

The **New** wizard closes.

6.7.2 Editing custom scenarios

After defining a custom scenario using the **New** wizard, a node appears for **Custom Scenarios** in the **Performance Analysis** view. Expand this node and select the custom scenario. Double-click the scenario to open the **Custom Scenario Editor**.The **Custom Scenario Editor** allows you to modify the custom scenarios that you have defined according to your requirements.

You can add events, counters and metrics to the custom scenario using Custom Scenario Editor.

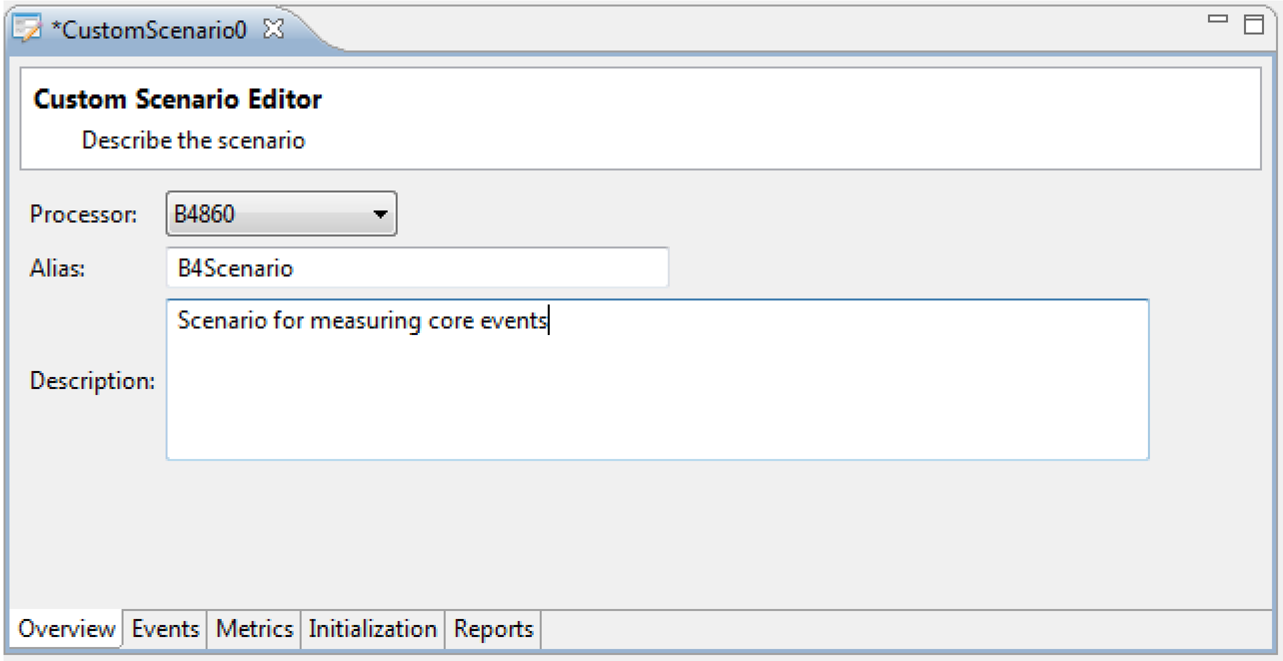
To modify a custom scenario, use the following tabs of the **Custom Scenario Editor**.

- Overview
- Events
- Metrics
- Initialization
- Reports

6.7.2.1 Overview

The **Overview** tab of the **Custom Scenario Editor** provides a brief summary of the scenario you are editing.

It replicates the information that was entered during the creation of the custom scenario. You can edit the processor to be used for the scenario, provide an alias name to the scenario, or modify the description of the custom scenario using the respective fields.


*CustomScenario0

Custom Scenario Editor

Describe the scenario

Processor: B4860

Alias: B4Scenario

Description: Scenario for measuring core events

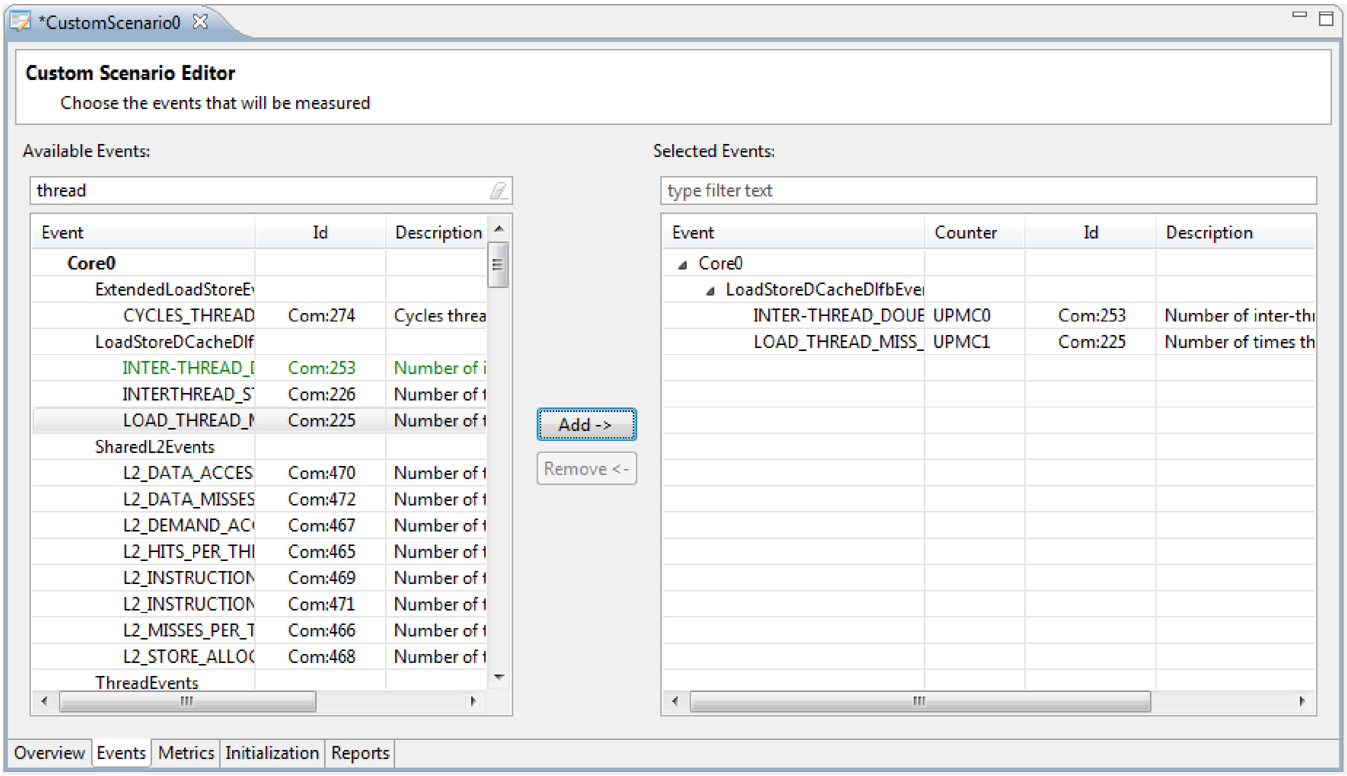
Figure 6-26. Custom Scenario Editor - Overview tab

6.7.2.2 Events

The **Events** tab of the **Custom Scenario Editor** lists all of the device's components that have events available for configuration, in the **Available Events** pane.

There can be hundred of these events, so to make it easier to locate an event of interest, this tab provides a search text box where you can enter a search string. For example, in the figure below, as the word *thread* is entered, the tool has displayed all events that include the word *thread* in their description.

Once you locate an event of interest, select it and click the **Add** button. The selected event will be added to the list of **Selected Events** pane on the right side. The Performance Analysis tool automatically handles the allocation of the event to a given counter and manages the available counters as well. As soon as all counters have been assigned, any other events listed will be grayed out. If you try to add an event from the grayed out list, the tool displays an error message indicating that there are no more counters available to assign.



Custom Scenario Editor

Choose the events that will be measured

Available Events: thread

Event	Id	Description
Core0		
ExtendedLoadStoreE		
CYCLES_THREAD	Com:274	Cycles threa
LoadStoreDCacheDIf		
INTER-THREAD_I	Com:253	Number of i
INTERTHREAD_S	Com:226	Number of t
LOAD_THREAD_I	Com:225	Number of t
SharedL2Events		
L2_DATA_ACCES	Com:470	Number of t
L2_DATA_MISSES	Com:472	Number of t
L2_DEMAND_AC	Com:467	Number of t
L2_HITS_PER_THI	Com:465	Number of t
L2_INSTRUCTION	Com:469	Number of t
L2_INSTRUCTION	Com:471	Number of t
L2_MISSES_PER_T	Com:466	Number of t
L2_STORE_ALLOC	Com:468	Number of t
ThreadEvents		

Selected Events: type filter text

Event	Counter	Id	Description
Core0			
LoadStoreDCacheDIfbEve			
INTER-THREAD_DOUE	UPMC0	Com:253	Number of inter-thi
LOAD_THREAD_MISS_	UPMC1	Com:225	Number of times th

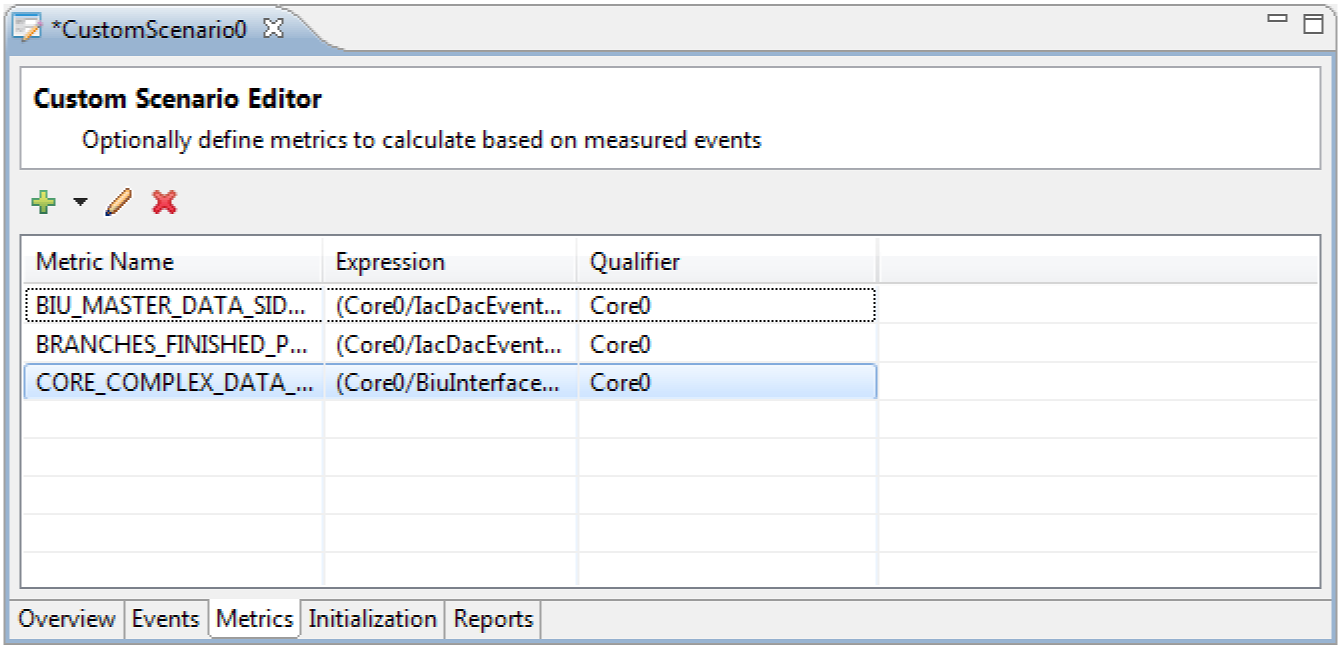
Figure 6-27. Custom Scenario Editor - Events tab

6.7.2.3 Metrics

The **Metrics** tab of the **Custom Scenario Editor** lets you define (optionally) metrics (mathematical operations) of the measured events.

The **Metrics** tab provides a toolbar, which allows you:

- Add a metric
- Edit an existing metric that is selected in the table
- Remove an existing metric that is selected in the table



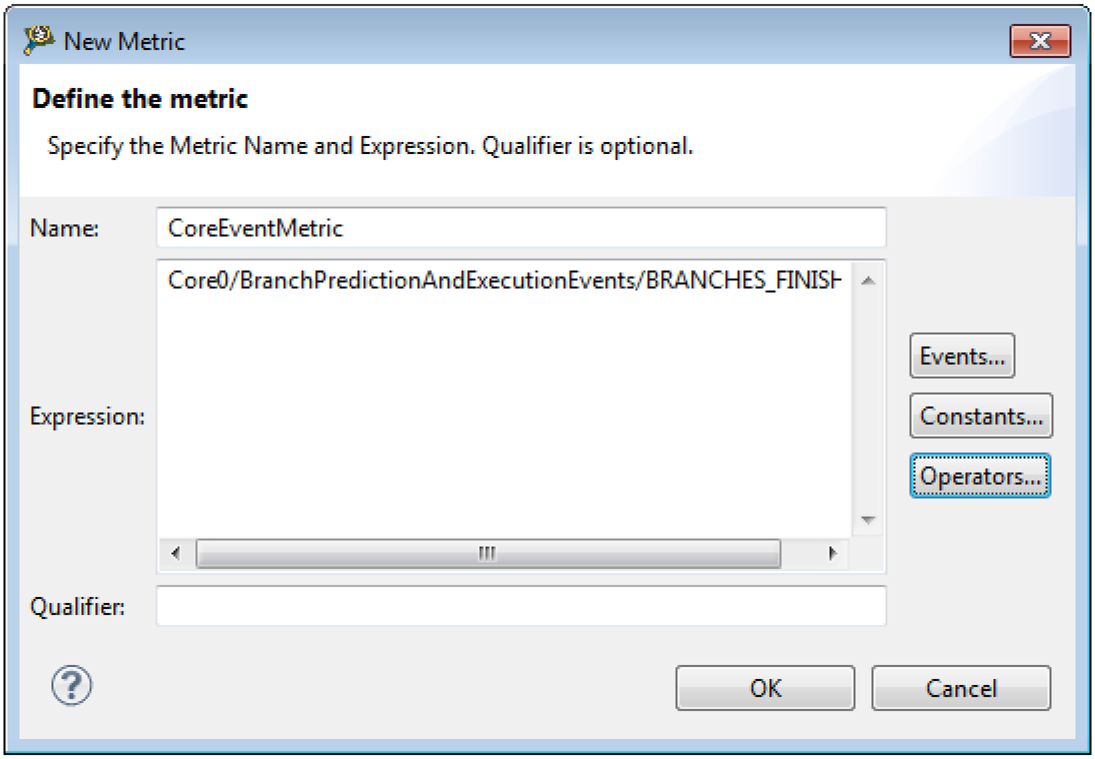
Custom Scenario Editor

Optionally define metrics to calculate based on measured events

Metric Name	Expression	Qualifier
BIU_MASTER_DATA_SID...	(Core0/IacDacEvent...	Core0
BRANCHES_FINISHED_P...	(Core0/IacDacEvent...	Core0
CORE_COMPLEX_DATA_...	(Core0/BiuInterface...	Core0

Figure 6-28. Custom Scenario Editor - Metrics tab

To add a metric to a custom scenario, click the **Add Metric** button in the toolbar to open the **New Metric** dialog. The **Add Metric** button provides a pop-up menu containing two options: **Add New Metric** and **Add Stock Metrics**.



Define the metric

Specify the Metric Name and Expression. Qualifier is optional.

Name: CoreEventMetric

Expression: Core0/BranchPredictionAndExecutionEvents/BRANCHES_FINISH

Qualifier:

Figure 6-29. Add new metric

Specify the **Name**, **Expression**, and **Qualifier** for the metric using the respective fields. The **Expression** represents a combination of event names and mathematical operands. The **Events** button allows you to select the necessary events for a metric. You can also select a constant to be used within the metric expression using the **Constants** button. As metrics are operations on events, you will also need to choose an appropriate operator. Click the **Operators** button to view the list of C language operator supported on the metric expressions. Click **OK** to complete the definition of the metric. The **Qualifier** represents a tag or a label. It helps to define filters based on an attribute/property.

Click the **Add Stock Metrics** option to open the **Stock Metrics** dialog, which allows you to select a metric from a list of predefined metrics to be added to a custom scenario. You can add a stock metric by selecting the checkbox corresponding to the metric. To enable a stock metric, right-click it and choose the **Add Events to Enable Metric** option.

Some of the metrics in the **Stock Metrics** dialog are disabled due to missing events. The Performance Analysis tool automatically handles the allocation of the event to a given counter and manages the available counters as well. As soon as all counters have been assigned, any other events listed will be grayed out. If you try to add an event from the grayed out list, the tool displays an error message indicating that there are no more counters available to assign.

6.7.2.4 Initialization

Use the **Initialization** tab in situations where you may require advanced target initialization beyond the configuration that enables the event-counter pairs.

The **Initialization** tab of the **Custom Scenario Editor** allows you to enter a python code or specify a python script directly. Select the **Enter Python Code** option to directly write the python code in the available text editor. Select the **Specify Python Script File** option to search the workspace or file system for previously saved python scripts.

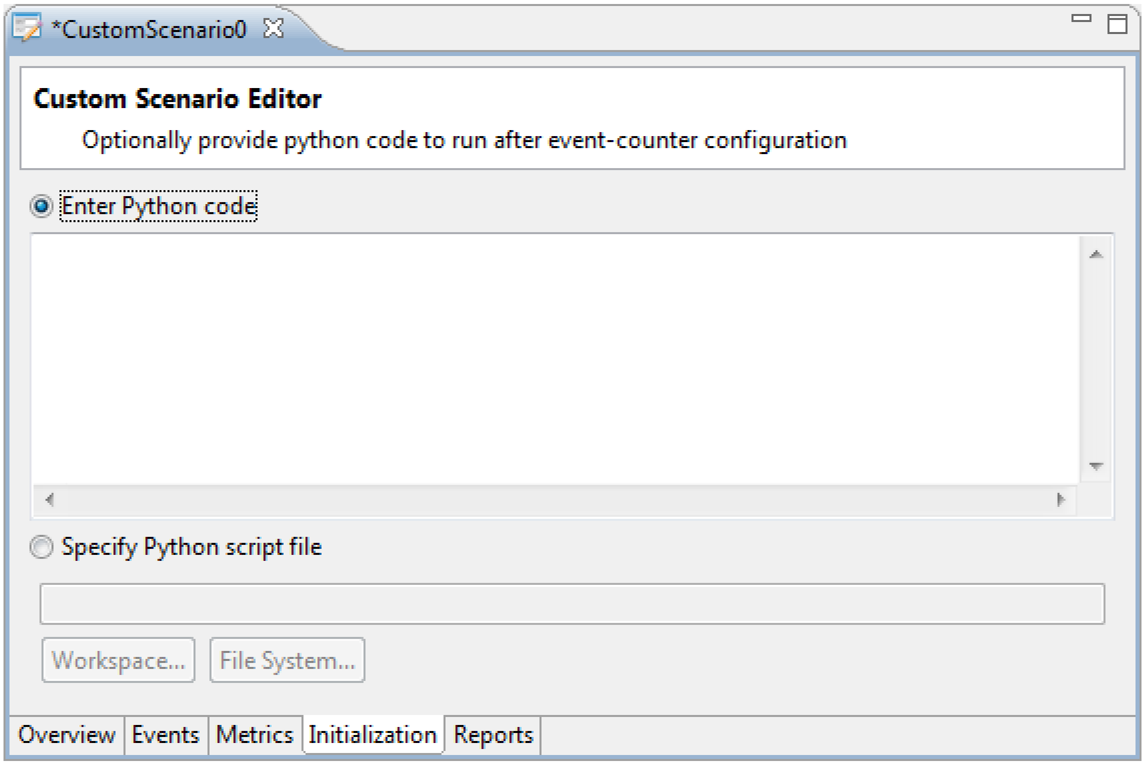
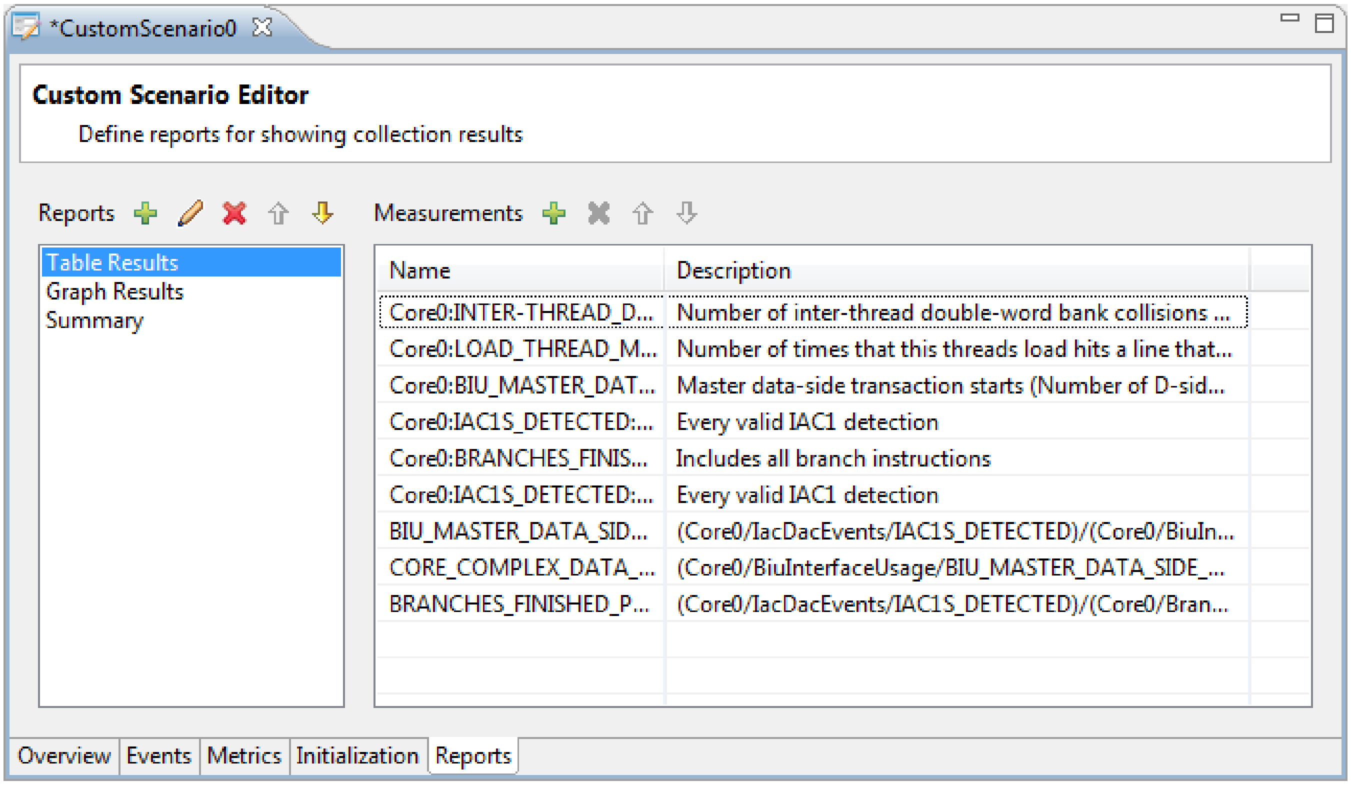


Figure 6-30. Custom Scenario Editor - Initialization tab

6.7.2.5 Reports

The **Reports** tab of the **Custom Scenario Editor** displays the reports of the event and metric measurements that you have added to the custom scenario.



Custom Scenario Editor

Define reports for showing collection results

Table Results
Graph Results
Summary

Name	Description
Core0:INTER-THREAD_D...	Number of inter-thread double-word bank collisions ...
Core0:LOAD_THREAD_M...	Number of times that this threads load hits a line that...
Core0:BIU_MASTER_DAT...	Master data-side transaction starts (Number of D-sid...
Core0:IAC1S_DETECTED:...	Every valid IAC1 detection
Core0:BRANCHES_FINIS...	Includes all branch instructions
Core0:IAC1S_DETECTED:...	Every valid IAC1 detection
BIU_MASTER_DATA_SID...	(Core0/IacDacEvents/IAC1S_DETECTED)/(Core0/BiuIn...
CORE_COMPLEX_DATA_...	(Core0/BiuInterfaceUsage/BIU_MASTER_DATA_SIDE_...
BRANCHES_FINISHED_P...	(Core0/IacDacEvents/IAC1S_DETECTED)/(Core0/Bran...

Figure 6-31. Custom Scenario Editor - Reports tab

The toolbar next to the **Reports** pane provides the following buttons.

- **New Report** - Click to open the **New Report** dialog and create a new report. Specify the report **Type**, **Name**, and **Description** in the respective fields. Select the **Automatically add new events and metrics** checkbox to automatically include in the report the new measurements that you add to the custom scenario.
- **Edit Report** - Click to open the **Edit Report** dialog to modify the selected report.
- **Delete Report** - Click to delete the selected report.
- **Move Report Up** - Click to move the selected report up in the list.
- **Move Report Down** - Click to move the selected report down in the list.

The toolbar next to the **Measurements** pane provides the following buttons.

- **Add Measurement** - Click to open the **Add Measurement** dialog. This dialog lets you select the events and metrics that you want to add to the report. This dialog displays only those events and metrics that have not been added yet to the report.
- **Remove Measurement** - Click to delete the selected measurement.
- **Move Measurement Up** - Click to move the selected measurement up in the list.
- **Move Measurement Down** - Click to move the selected measurement down in the list.

Chapter 7
Launching Scripts

The Profiling and Analysis tools help you launch scripts written in Python language. These scripts allow you to collect and export trace data automatically using remote launch. This feature is integrated with CodeWarrior IDE scripting. The scripts are launched in remote launch using the Jython console. The remote launch feature of CodeWarrior allows launch configurations to be executed remotely. For more details on API of remote launch, refer to the Remote Launch API Appendix.

In this chapter:

- Run Sample Python Script
- Collect Trace Using Jython

7.1 Run Sample Python Script

The steps in this topic demonstrate the procedure of running a sample Python script from Jython console, that is running the script that you might have written for collecting trace data.

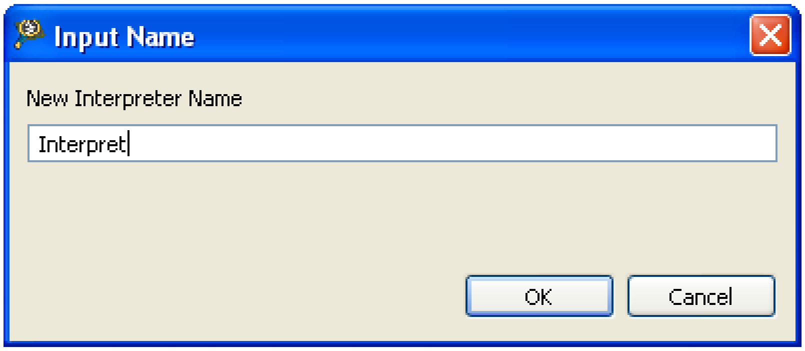
Perform the following steps:

1. Create a new B4860 project with the name remote-launch.
2. Replace the source code of `b4860_main.c` with your own code.
3. Save and build your project.
4. Update the project name, launch config name and start/stop tracepoints line numbers in the `TestHwTracepointsProgTrace.py` script according to your project that is created, `TestHwTracepointsProgTrace.py` file is located at: `<CW installation directory>\eclipse\plugins\com.freescale.sa.sc.scripting_*\remote\scripting_sa_sc`.
5. Select **Window > Show View > Other > Debug > Remote Launch** to open the **Remote Launch** view.

6. Click the **Enable Remote Launch** button displayed on the top right of the view to enable remote launching.

   The color of the button changes to red.

7. Select **Window > Show View > Other > Debug > Jython Consoles** to open the **Jython Consoles** view.
8. Open the **View** menu in the **Jython Consoles** view, and select the **New Interpreter** option.



New Interpreter Name

Interpret

Figure 7-1. Select New Interpreter Option in Jython Consoles View

The **Input Name** dialog box appears.

9. Specify a name for the new interpreter and click **OK**.
10. In the Jython console command prompt (">>>"), input code in Python syntax. You may access any classes in your test scripts, for example, `TestHwTracepointsProgTrace.py`, by inputting the following commands in the Jython command prompt: `>>> from scripting_sa_sc import TestHwTracepointsProgTrace>>> RemoteLaunchUtils.executeLaunchScript(TestHwTracepointsProgTrace())`

NOTE

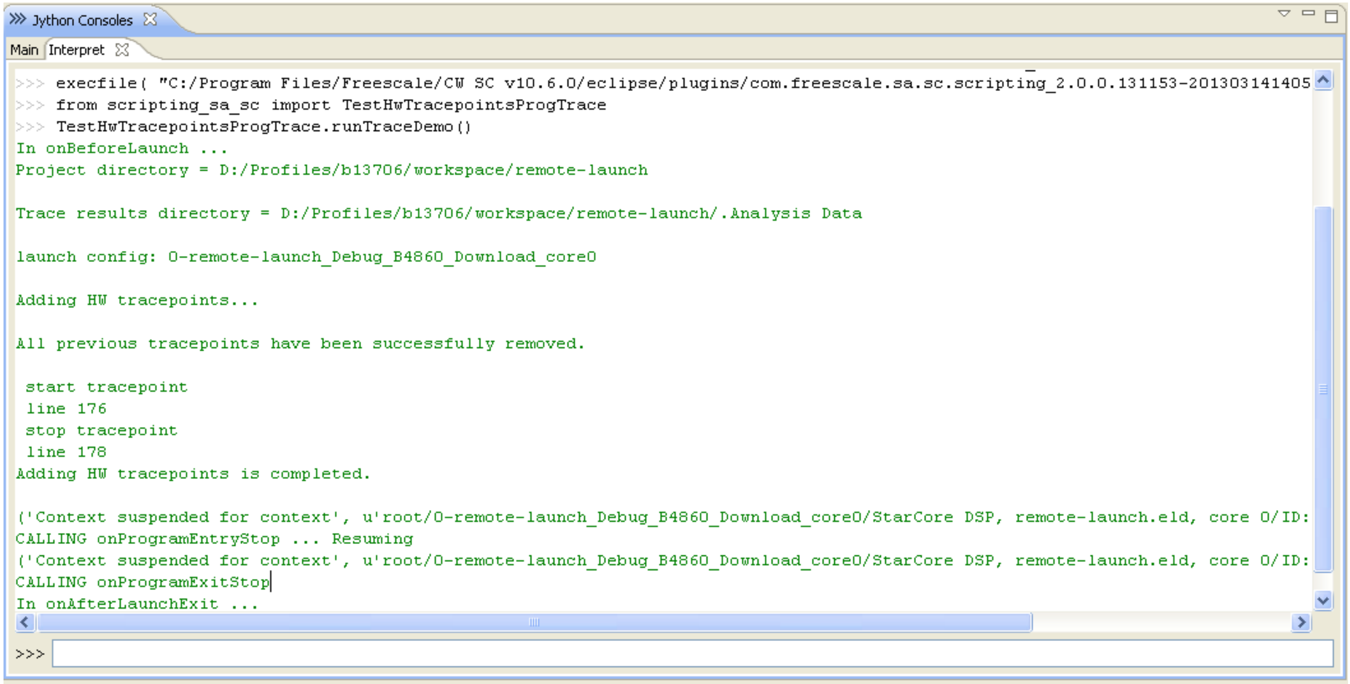
The advantage of using Jython is that besides Python modules, it can also access Java modules in the similar manner.

This is how you run a sample Python script from Jython console.

7.2 Collect Trace Using Jython

This section demonstrates the procedure of launching configuration and collecting both manual and automatic trace on the B4860 target.

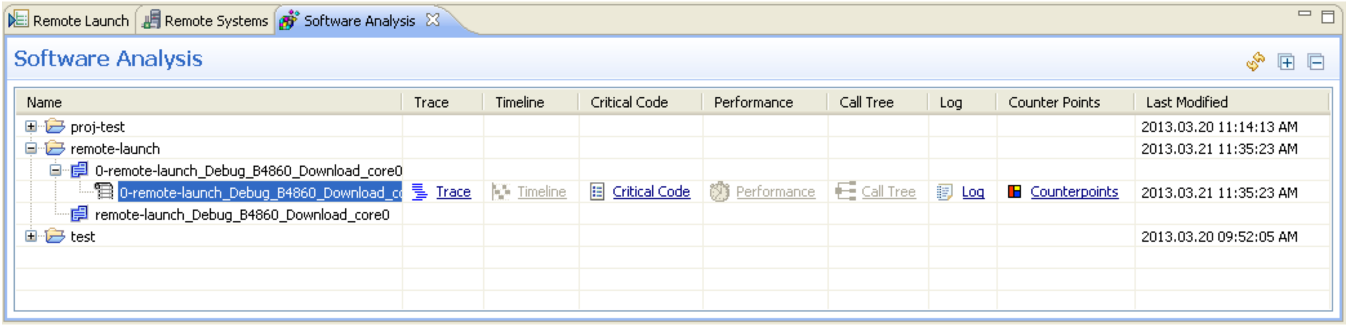
To collect trace using remote launch through Jython console, perform the following steps.



```
>>> execfile( "C:/Program Files/Freescale/CW SC v10.6.0/eclipse/plugins/com.freescale.sa.sc.scripting_2.0.0.131153-201303141405
>>> from scripting_sa_sc import TestHwTracepointsProgTrace
>>> TestHwTracepointsProgTrace.runTraceDemo()
In onBeforeLaunch ...
Project directory = D:/Profiles/b13706/workspace/remote-launch

Trace results directory = D:/Profiles/b13706/workspace/remote-launch/.Analysis Data

launch config: 0-remote-launch_Debug_B4860_Download_core0

Adding HW tracepoints...

All previous tracepoints have been successfully removed.

 start tracepoint
 line 176
 stop tracepoint
 line 178
Adding HW tracepoints is completed.

('Context suspended for context', u'root/0-remote-launch_Debug_B4860_Download_core0/StarCore DSP, remote-launch.eld, core 0/ID:
CALLING onProgramEntryStop ... Resuming
('Context suspended for context', u'root/0-remote-launch_Debug_B4860_Download_core0/StarCore DSP, remote-launch.eld, core 0/ID:
CALLING onProgramExitStop
In onAfterLaunchExit ...
>>>
```

Figure 7-2. Running Commands at Jython Command Prompt for Collecting Trace

The trace collection starts and the **Software Analysis** view appears with the trace results. Open the **Software Analysis** view and check the collected results.

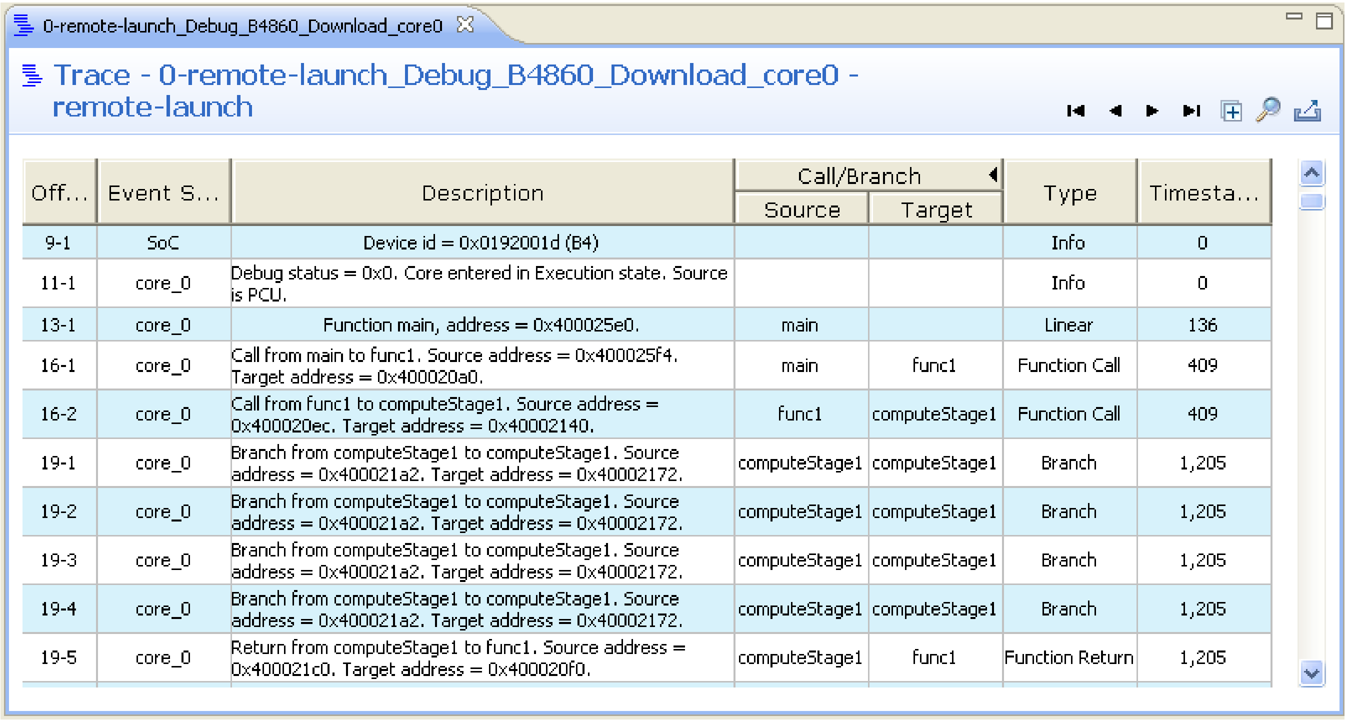


Software Analysis

Name	Trace	Timeline	Critical Code	Performance	Call Tree	Log	Counter Points	Last Modified
proj-test								2013.03.20 11:14:13 AM
remote-launch								2013.03.21 11:35:23 AM
0-remote-launch_Debug_B4860_Download_core0								
0-remote-launch_Debug_B4860_Download_c	Trace	Timeline	Critical Code	Performance	Call Tree	Log	Counterpoints	2013.03.21 11:35:23 AM
remote-launch_Debug_B4860_Download_core0								
test								2013.03.20 09:52:05 AM

Figure 7-3. Software Analysis View Containing Trace Results Collected Using Remote Launch

Click **Trace** and check the collected results.



Trace - 0-remote-launch_Debug_B4860_Download_core0 - remote-launch

Off...	Event S...	Description	Call/Branch		Type	Timesta...
			Source	Target		
9-1	SoC	Device id = 0x0192001d (B4)			Info	0
11-1	core_0	Debug status = 0x0. Core entered in Execution state. Source is PCU.			Info	0
13-1	core_0	Function main, address = 0x400025e0.	main		Linear	136
16-1	core_0	Call from main to func1. Source address = 0x400025f4. Target address = 0x400020a0.	main	func1	Function Call	409
16-2	core_0	Call from func1 to computeStage1. Source address = 0x400020ec. Target address = 0x40002140.	func1	computeStage1	Function Call	409
19-1	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,205
19-2	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,205
19-3	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,205
19-4	core_0	Branch from computeStage1 to computeStage1. Source address = 0x400021a2. Target address = 0x40002172.	computeStage1	computeStage1	Branch	1,205
19-5	core_0	Return from computeStage1 to func1. Source address = 0x400021c0. Target address = 0x400020f0.	computeStage1	func1	Function Return	1,205

Figure 7-4. Trace Results Collected Using Remote Launch

NOTE

The sample scripts used for collecting continuous and automatic trace on the B4860 targets are located at: `<CW Installation Directory>\eclipse\plugins \com.freescale.sa.sc.scripting_*\remote\scripting_sa_sc`.

This is how you collect trace using the Jython console.

Chapter 8
Remote Launch API

This appendix documents following classes and interface for the remote launch API.

Classes

- com::freescale::sa::scripting::api::AnalysisActions
- com::freescale::sa::scripting::api::AnalysisConfig
- com::freescale::sa::scripting::api::AnalysisErrors
- com::freescale::sa::scripting::api::AnalysisFactory
- com::freescale::sa::scripting::api::AnalysisResults
- com::freescale::sa::sc::scripting::api::ScAnalysisConfig
- com::freescale::sa::sc::scripting::api::ScHwAnalysisActions
- com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig
- com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig
- com::freescale::sa::scripting::api::TracePoint

Interface

- com.freescale.sa.scripting.IAnalysisConfigFactory

8.1 com::freescale::sa::scripting::api::AnalysisActions

Refer below.

8.1.1 Public Member Functions

Lists the public member functions of the **com::freescale::sa::scripting::api::AnalysisActions** class.

Following public members are available:

- AnalysisActions

- startTrace
- stopTrace

Comprehensive description of each of the above listed functions is as follows:

8.1.1.1 AnalysisActions

Constructor.

```
com.freescale.sa.scripting.api.AnalysisActions.AnalysisActions(AnalysisConfig config)
```

Parameters

```
config
```

The AnalysisConfig-inherited instance, which carries the information needed to identify the running launch.

8.1.1.2 startTrace

Start trace collection.

```
synchronized void com.freescale.sa.scripting.api.AnalysisActions.startTrace()
```

8.1.1.3 stopTrace

Stop trace collection.

```
synchronized void com.freescale.sa.scripting.api.AnalysisActions.stopTrace()
```

8.1.2 Public Static Member Functions

List the public static member functions of the **com::freescale::sa::scripting::api::AnalysisActions** class.

Following public static members are available:

- startTrace
- stopTrace

Comprehensive description of each of the above listed functions is as follows:

8.1.2.1 startTrace

Starts trace collection.

This method is intended for a static use, where an analysis configuration is not available.

```
static synchronized void com.freescale.sa.scripting.api.AnalysisActions.startTrace(String
launchConfigName)
```

Parameters

```
launchConfigName
```

The name of the running launch configuration, on which the action is triggered.

8.1.2.2 stopTrace

Stop trace collection.

This method is intended for a static use, where an analysis configuration is not available.

```
static synchronized void com.freescale.sa.scripting.api.AnalysisActions.stopTrace(String
launchConfigName)
```

Parameters

```
launchConfigName
```

The name of the running launch configuration, on which the action is triggered.

8.2 com::freescale::sa::scripting::api::AnalysisConfig

Refer below.

8.2.1 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::scripting::api::AnalysisConfig** class.

Following protected members are available:

- AnalysisConfig
- setProject
- load
- save
- getWorkingCopy
- getLaunchConfiguration

Comprehensive description of each of the above listed functions is as follows:

8.2.1.1 AnalysisConfig

```
com.freescale.sa.scripting.api.AnalysisConfig.AnalysisConfig(String projectName)
```

8.2.1.2 setProject

```
void com.freescale.sa.scripting.api.AnalysisConfig.setProject(ILaunchConfiguration
launchConfig)
```

8.2.1.3 load

```
void com.freescale.sa.scripting.api.AnalysisConfig.load()
```

8.2.1.4 save

Saves current configuration.

```
void com.freescale.sa.scripting.api.AnalysisConfig.save()
```

8.2.1.5 getWorkingCopy

Get the working copy of the active launch configuration.

```
ILaunchConfigurationWorkingCopy
com.freescale.sa.scripting.api.AnalysisConfig.getWorkingCopy()
```

Returns

The current launch configuration working copy.

8.2.1.6 getLaunchConfiguration

```
ILaunchConfiguration com.freescale.sa.scripting.api.AnalysisConfig.getLaunchConfiguration()
```

8.2.2 Public Member Functions

Lists the public member functions of the **com::freescale::sa::scripting::api::AnalysisConfig** class.

Following public member functions are available:

- getOutputFolder
- addSrcLineTracePoint
- addAddressTracePoint
- removeTracePoint
- getTracePoints
- setIgnoreAllTracePoints
- getIgnoreAllTracePoints
- removeAllTracePoints
- removeLineCounterpoint
- removeAddrCounterpoint
- setLineCounterpointEnabled
- setAddrCounterpointEnabled
- getAttribute
- setAttribute

8.2.2.1 getOutputFolder

Returns the SA output folder attribute.

```
String com.freescale.sa.scripting.api.AnalysisConfig.getOutputFolder()
```

Returns

The output folder path.

8.2.2.2 addSrcLineTracePoint

Adds a tracepoint identified by a source file and a line number to a project.

```
TracePoint com.freescale.sa.scripting.api.AnalysisConfig.addSrcLineTracePoint(String
sourceFilePath, int lineNo, TracePoint.Action action, TracePoint.Type type)
```

Parameters

`sourceFilePath`

Path towards source file of tracepoint.

`lineNo`

Line Number of tracepoint.

`action`

Action of tracepoint, see for possible values. TracePoint.Action

`type`

Type of tracepoint, see for possible values. TracePoint.Action

Returns

Return the resulted . TracePoint

8.2.2.3 addAddressTracePoint

Adds an tracepoint idenfified by an address to a project.

```
TracePoint com.freescale.sa.scripting.api.AnalysisConfig.addAddressTracePoint(String
sourceFilePath, String address, TracePoint.Action action, TracePoint.Type type, int lineNo)
```

Parameters

`sourceFilePath`

Path towards source file where marker for tracepoint should be displayed. Set it to empty string and no marker will be displayed for this tracepoint in source file.

`address`

Address of tracepoint.

`action`

Action of tracepoint, see for possible values. TracePoint.Action

`type`

Type of tracepoint, see for possible values. TracePoint.Action

`lineNo`

Line number from source file where marker for tracepoint should be displayed. Set it to 0 and no marker will be displayed for this tracepoint in source file.

`coreException`

Returns

Return the resulted . TracePoint

8.2.2.4 removeTracePoint

Remove tp tracepoint from the current project.

```
int com.freescale.sa.scripting.api.AnalysisConfig.removeTracePoint(TracePoint tp)
```

Parameters

`tp`

Tracepoint to be removed.

Returns

AnalysisErrors.STATUS_OK if the operation was successful, or AnalysisErrors.STATUS_ERROR if it fails.

8.2.2.5 getTracePoints

Get all tracepoints from the current project.

```
Vector com.freescale.sa.scripting.api.AnalysisConfig.getTracePoints()
```

Returns

All tracepoints from the current project

8.2.2.6 setIgnoreAllTracePoints

Set all tracepoints to be ignored or not.

```
void com.freescale.sa.scripting.api.AnalysisConfig.setIgnoreAllTracePoints(boolean ignoreAll)
```

Parameters

`ignoreAll`

True for ignore all tracepoints, false otherwise.

8.2.2.7 getIgnoreAllTracePoints

Get ignore all tracepoints status.

```
boolean com.freescale.sa.scripting.api.AnalysisConfig.getIgnoreAllTracePoints()
```

Returns

True if all tracepoints are currently ignored, false otherwise.

8.2.2.8 removeAllTracePoints

Remove all tracepoints from the current project.

```
int com.freescale.sa.scripting.api.AnalysisConfig.removeAllTracePoints()
```

Returns

AnalysisErrors.STATUS_OK if the operation was successful, or AnalysisErrors.STATUS_ERROR if it fails.

8.2.2.9 removeLineCounterpoint

Remove a counterpoint from a C source file line number.

```
void com.freescale.sa.scripting.api.AnalysisConfig.removeLineCounterpoint(int lineNumber,
String sourceFilePath)
```

Parameters

`lineNumber`

The line number the counterpoint to be added to.

`sourceFilePath`

The path to the source file, where the counterpoint will be added.

`CoreException`

8.2.2.10 removeAddrCounterpoint

Remove an address counterpoint.

```
void com.freescale.sa.scripting.api.AnalysisConfig.removeAddrCounterpoint(String hexAddr)
```

Parameters

`hexAddr`

The hex address at which this counterpoint is set.

`CoreException`

8.2.2.11 setLineCounterpointEnabled

Enable or disable a line counterpoint.

```
void com.freescale.sa.scripting.api.AnalysisConfig.setLineCounterpointEnabled(int
lineNumber, String sourceFilePath, boolean enabled)
```

Parameters

`lineNumber`

The line number the counterpoint needs to be set up at.

`sourceFilePath`

The path to the C source file.

`enabled`

Set counterpoint enabled or disabled.

`CoreException`

8.2.2.12 setAddrCounterpointEnabled

Enable or disable an address counterpoint.

```
void com.freescale.sa.scripting.api.AnalysisConfig.setAddrCounterpointEnabled(boolean
enabled, String hexAddr)
```

Parameters

`enabled`

Set counterpoint enabled or disabled.

`hexAddr`

The HEX address the counterpoint needs to be set up at.

`CoreException`

8.2.2.13 getAttribute

Get the value of a SA general configuration attribute.

```
Object com.freescale.sa.scripting.api.AnalysisConfig.getAttribute(Object attributeType)
```

Parameters

`attributeType`

The attribute from which the value will be retrieved. The attributes are exposed by the AnalysisConfigAttributes enum.

Returns

The specific attribute value.

8.2.2.14 setAttribute

Set one of the SA general config attributes.

```
void com.freescale.sa.scripting.api.AnalysisConfig.setAttribute(ObjectattributeType, Object
arg)
```

Parameters

```
attributeType
```

One of the AnalysisConfigAttributes enum attributes (SET_AUTO_TRACE_COLLECTION).

```
arg
```

The value the attribute will take.

8.2.3 Public Static Member Functions

Lists the public static member functions of the **com::freescale::sa::scripting::api::AnalysisConfig** cl;ass.

Following public static member functions are available:

- removeAllTracepointsByType

8.2.3.1 removeAllTracepointsByType

Remove all tracepoints of the same kind.

```
static void
com.freescale.sa.scripting.api.AnalysisConfig.removeAllTracepointsByType(TracePoint.Type
tpType)
```

Parameters

```
tpType
```

Tracepoint type.

```
CoreException
```

8.3 com::freescale::sa::scripting::api::AnalysisErrors

Refer below.

8.3.1 Public Static Member Functions

Lists the public static member functions of the **com::freescale::sa::scripting::api::AnalysisErrors** class.

Following public static member functions are available:

- log

8.3.1.1 log

Log error.

```
static void com.freescale.sa.scripting.api.AnalysisErrors.log(Throwable e)
```

Parameters

```
e
```

A throwable object.

8.4 com::freescale::sa::scripting::api::AnalysisFactory

Refer below.

8.4.1 Public Static Member Functions

Lists the public static member functions of the **com::freescale::sa::scripting::api::AnalysisFactory** class.

Following public static members are available:

- getAnalysisConfig
- getAnalysisConfigFromLaunch
- getAnalysisResults
- getAnalysisResults

- getAnalysisResults
- getAnalysisResults

8.4.1.1 getAnalysisConfig

Get which configures the SA configuration attributes. AnalysisConfig

```
static AnalysisConfig
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisConfig(String projectName)
```

Parameters

```
projectName
```

The project name.

Returns

An object. AnalysisConfig

8.4.1.2 getAnalysisConfigFromLaunch

Get which configures the SA configuration attributes. AnalysisConfig

This method has a more focused use case, where the user has more than one launch configuration and he needs to specifically select one for the launch script.

```
static AnalysisConfig
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisConfigFromLaunch(String
launchConfigName)
```

Parameters

```
launchConfigName
```

The launch configuration name, as it is described only by the base name without the file's extension.

Returns

An object. AnalysisConfig

8.4.1.3 getAnalysisResults

Get which helps in exporting trace information. AnalysisResults

```
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, int
timeout)
```

Parameters

`projectName`

The name of the project.

`timeout`

Maximum allowed time for the results to become available.

Returns

An instance. AnalysisResults

8.4.1.4 getAnalysisResults

Get which helps in exporting trace information. AnalysisResults

This method returns an AnalysisResult instance with a default 10 second wait for results availability.

```
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName)
```

Parameters

`projectName`

The name of the project.

Returns

An instance. AnalysisResults

8.4.1.5 getAnalysisResults

Get which helps in exporting trace information. AnalysisResults

```
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, String
baseFileName, int timeout)
```

Parameters

`projectName`

The name of the project.

`baseFileName`

Base name, without extension, of the result set

`timeout`

Base name, without extension, of the result set

Returns

An instance. AnalysisResults

8.4.1.6 getAnalysisResults

Get which helps in exporting trace information. AnalysisResults

This method returns an AnalysisResult instance with a default 10 second wait for results availability.

```
static AnalysisResults
com.freescale.sa.scripting.api.AnalysisFactory.getAnalysisResults(String projectName, String
baseFileName)
```

Parameters

`projectName`

The name of the project.

`baseFileName`

Base name, without extension, of the result set

Returns

An instance. AnalysisResults

8.5 com::freescale::sa::scripting::api::AnalysisResults

Refer below.

8.5.1 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::scripting::api::AnalysisResults** class.

Following protected members are available:

- AnalysisResults
- AnalysisResults
- exportTraceData

Comprehensive description of each of the above listed functions is as follows:

8.5.1.1 AnalysisResults

```
com.freescale.sa.scripting.api.AnalysisResults.AnalysisResults(String projectName, String
baseName, int timeoutSeconds)
```

Parameters

`projectName`

The name of the current project.

`baseName`

Base name, without extension, of the result set

`stimeoutSeconds`

Maximum allowed time for the results to become available, in seconds.

8.5.1.2 AnalysisResults

Overloaded constructor which uses the default waiting time for the target file (i.e.config file) file.

```
com.freescale.sa.scripting.api.AnalysisResults.AnalysisResults(String projectName, String
baseFileName)
```

Parameters

```
projectName
```

The name of the current project.

```
baseName
```

Base name, without extension, of the result set

8.5.1.3 exportTraceData

```
void com.freescale.sa.scripting.api.AnalysisResults.exportTraceData(String filePath)
```

8.5.2 Public Member Functions

Lists the public member functions of the **com::freescale::sa::scripting::api::AnalysisResults** class.

Following public members are available:

- getTraceRecordsNo
- exportResults
- exportFunctionResults

Comprehensive description of each of the above listed functions is as follows:

8.5.2.1 getTraceRecordsNo

```
long com.freescale.sa.scripting.api.AnalysisResults.getTraceRecordsNo()
```

8.5.2.2 exportResults

Export to a data format one of the available data types.

The data types that can be exported with this method can take only one argument.

```
com.freescale.sa.scripting.api.AnalysisResults.exportResults(ExportDataType dataType, String
filePath)
```

Parameters

```
dataType
```

the data type as one of the attributes from ExportDataType enum.

```
filePath
```

the path to the desired .csv output file.

8.5.2.3 exportFunctionResults

Export to a data format one of the available data types.

The data types that can be exported with this method should take two arguments.

```
com.freescale.sa.scripting.api.AnalysisResults.exportFunctionResults(ExportDataType
dataType, String filePath, Object arg)
```

Parameters

```
dataType
```

the data type as one of the attributes from ExportDataType enum.

```
filePath
```

the path to the desired .csv output file.

```
arg
```

Defines the SHOW_CODE_TYPE for the CRITICAL_CODE_HTML attribute or the name of the analysed C function (one of the source file's functions) for CRITICAL_CODE_FUNCTION_CSV and HIERARCHICAL_FUNCTION_CSV.

8.5.3 Public Static Member Functions

Lists the public static member functions of the **com::freescale::sa::scripting::api::AnalysisResults** class.

Following public ststic members are available:

- logMessages

Comprehensive description of each of the above listed functions is as follows:

8.5.3.1 logMessages

Method used for temporary logging.

Afterwards, the logging will be done by the RemoteLaunch API.

```
static void com.freescale.sa.scripting.api.AnalysisResults.logMessages(String message)
```

Parameters

`message`

The message to be logged.

8.6 com::freescale::sa::sc::scripting::api::ScAnalysisConfig

Refer below.

8.6.1 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::sc::scripting::api::ScAnalysisConfig** class.

Following protected members are available:

- ScAnalysisConfig
- ScAnalysisConfig

8.6.1.1 ScAnalysisConfig

```
com.freescale.sa.sc.scripting.api.ScAnalysisConfig.ScAnalysisConfig(ILaunchConfiguration
launchConfig)
```

8.6.1.2 ScAnalysisConfig

```
  com.freescale.sa.sc.scripting.api.ScAnalysisConfig.ScAnalysisConfig(St
  ring projectName)
```

8.6.2 Public Member Functions

Lists the public member functions of the **com::freescale::sa::sc::scripting::api::ScAnalysisConfig** class.

Following public members are available:

- getAttribute
- setAttribute

Comprehensive description of each of the above listed functions is as follows:

8.6.2.1 getAttribute

Get the value of a specific SA StarCore configuration attribute.

```
Object com.freescale.sa.sc.scripting.api.ScAnalysisConfig.getAttribute(Object attributeType)
```

Parameters

```
attributeType
```

The attribute from which the value will be retrieved. The list of attributes are exposed by the ScAnalysisConfigAttributes enum.

Returns

The specific attribute value or null in case of attribute not found.

8.6.2.2 setAttribute

Set one of the SA StarCore general config attributes.

```
void
com.freescale.sa.sc.scripting.api.ScAnalysisConfig.setAttribute(Object attributeType, Object
arg)
```

Parameters

```
attributeType
```

One of the AnalysisConfigAttributes enum attributes.

```
arg
```

The value the attribute will take.

8.7 com::freescale::sa::sc::scripting::api::ScHwAnalysisActions

Refer below.

8.7.1 Public Member Functions

Lists the public member functions of the **com::freescale::sa::sc::scripting::api::ScHwAnalysisActions** class.

Following public member functions are available:

- ScHwAnalysisActions
- uploadTrace

Comprehensive description of each of the above listed functions is as follows:

8.7.1.1 ScHwAnalysisActions

It enables user defined events trace.

```
com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.ScHwAnalysisActions(AnalysisConfig
config)
```

8.7.1.2 uploadTrace

Trigger trace upload.

This should be triggered only in attach mode or user code, when a suspend is hit, after tracing was enabled.

```
synchronized void com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.uploadTrace()
```

8.7.2 Public Static Member Functions

Lists the public static member functions of the **com::freescale::sa::sc::scripting::api::ScHwAnalysisActions** class.

Following public static members are available:

- uploadTrace

Comprehensive description of each of the above listed functions is as follows:

8.7.2.1 uploadTrace

This method is intended for a static use, where an analysis configuration is not available. This should be triggered only in attach mode or user code, when a suspend is hit, after tracing was enabled.

```
static synchronized void
com.freescale.sa.sc.scripting.api.ScHwAnalysisActions.uploadTrace(String launchConfigName)
```

Parameters

`launchConfigName`

The name of the running launch configuration, on which the action is triggered.

8.8 com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig

Refer below.

8.8.1 Public Member Functions

Lists the public member functions of the **com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig** class.

Following public member functions are available:

- ScHwAnalysisConfig

- ScHwAnalysisConfig
- setAttribute
- getAttribute

8.8.1.1 ScHwAnalysisConfig

```
com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.ScHwAnalysisConfig(ILaunchConfiguration
launchConfig)
```

8.8.1.2 ScHwAnalysisConfig

```
com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.ScHwAnalysisConfig(String projectName)
```

8.8.1.3 setAttribute

Set one of the SA StarCore hardware config attributes. If called from python interpreter, there's no need to mention the enum name (e.g. ScHwAnalysisConfigAttributes.USER_CODE, just USER_CODE). The enum objects are automatically added as they are to the namespace of the running project, via a config.py file, found in com.freescale.sa.sc.scripting/remote.

```
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setAttribute(Object attributeType,
Object arg)
```

Parameters

```
attributeType
```

One of the ScHwAnalysisConfigAttributes enum attributes (TRACE_SCENARIO, OWNERSHIP_TRACE, USER_DEFINED_EVENTS, TRACE_BUFFER_START_ADDRESS, DATA_ADDR_TRACE, PROBE_BUFFER_SIZE, TRIAD_A_EVENT, TRIAD_B_EVENT, TRACE_COLLECTION_MODE, USER_CODE, DDR_BUFFER_START_ADDRESS, DDR_BUFFER_SIZE, SET_LCF_SETTINGS_FROM_FILE, BANDWIDTH, COLLECTION_PORT_TYPE).

Parameters

```
arg
```

The value the attribute will take. In case, the attribute to be set is ambigous, consult com.freescale.sa.sc.scripting/remote/config.py, where the scripting namespace is set. There can be found possible values for multiple-choice arguments(e.g. setTriadAEvent(..) or setTraceCollectionMode(..)).

8.8.1.4 getAttribute

Get the value of a specific SA StarCore hardware config attribute. If called from python interpreter, there's no need to mention the enum name (e.g. ScHwAnalysisConfigAttributes.USER_CODE, just USER_CODE). The enum objects are automatically added as they are to the namespace of the running project, via a config.py file, found in com.freescale.sa.sc.scripting/remote.

```
Object com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.getAttribute (Object
attributeType)
```

Parameters

```
attributeType
```

One of the ScHwAnalysisConfigAttributes enum attributes (TRACE_SCENARIO, OWNERSHIP_TRACE, USER_DEFINED_EVENTS, TRACE_BUFFER_START_ADDRESS, DATA_ADDR_TRACE, PROBE_BUFFER_SIZE, TRIAD_A_EVENT, TRIAD_B_EVENT, TRACE_COLLECTION_MODE, USER_CODE, DDR_BUFFER_START_ADDRESS, DDR_BUFFER_SIZE, SET_LCF_SETTINGS_FROM_FILE, BANDWIDTH, COLLECTION_PORT_TYPE).

Returns

the specific attribute value.

8.8.2 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::sc::scripting::api::ScHwAnalysisConfig** class.

Following protected members are available:

- setUserDefinedEventsTrace
- isUserDefinedEventsTrace
- setOwnershipTrace

- isOwnershipTrace
- setTraceScenario
- getTraceScenario
- load

Comprehensive description of each of the above listed functions is as follows:

8.8.2.1 setUserDefinedEventsTrace

It enables user defined events trace.

```
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setUserDefinedEventsTrace(boolean
userDefinedEvents)
```

Parameters

```
userDefinedEvents
```

Option to enable or not user defined events trace.

8.8.2.2 isUserDefinedEventsTrace

It returns the enable state of user defined events trace.

```
boolean com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.isUserDefinedEventsTrace()
```

8.8.2.3 setOwnershipTrace

It enables ownership trace.

```
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setOwnershipTrace(boolean
ownershipTrace)
```

Parameters

```
ownershipTrace
```

Option to enable or not ownership trace.

8.8.2.4 isOwnershipTrace

It returns the enable state of ownership trace.

```
boolean com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.isOwnershipTrace()
```

8.8.2.5 setTraceScenario

It sets the trace scenario by index.

```
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.setTraceScenario(TraceScenarioCode
scenarioCode)
```

Parameters

`scenarioCode`

The TraceScenarioCode used to identify the trace scenario. It is based on one of the scenario codes provided by the com.freescale.sa.sc.launch.ui# ScHwTargetSettingsManager# TraceScenarioCode enum. The members of this enum are: PROFILING_CACHE_EVENTS, PROFILING_DATA_TRACE, PROFILING_CLOCK_CYCLES, PROFILING_ADVANCED, PROGRAM_TRACE, COVERAGE, NONE

8.8.2.6 getTraceScenario

It gets index of trace scenario set as default.

```
TraceScenarioCode com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.getTraceScenario()
```

Returns

The trace scenario based on one of the scenario codes provided by the com.freescale.sa.sc.launch.ui# ScHwTargetSettingsManager# TraceScenarioCode enum. The members of this enum are: PROFILING_CACHE_EVENTS, PROFILING_DATA_TRACE, PROFILING_CLOCK_CYCLES, PROFILING_ADVANCED, PROGRAM_TRACE, COVERAGE, NONE

8.8.2.7 load

Loads configuration using a launch config.

Creates a trace config file in the process (if required).

```
void com.freescale.sa.sc.scripting.api.ScHwAnalysisConfig.load(ILaunchConfiguration
launchConfig)
```

Parameters

```
launchConfig
```

name of the launch configuration.

```
CoreException
```

8.9 com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig

Refer below.

8.9.1 Public Member Functions

Lists the public member functions of the **com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig** class.

Following public members are available:

- ScSimAnalysisConfig
- ScSimAnalysisConfig
- setAttribute
- getAttribute

Comprehensive description of each of the above listed functions is as follows:

8.9.1.1 ScSimAnalysisConfig

```
com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.ScSimAnalysisConfig(ILaunchConfiguratio
n launchConfig)
```

8.9.1.2 ScSimAnalysisConfig

```
com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.ScSimAnalysisConfig(String projectName)
```

8.9.1.3 setAttribute

Set one of the SA StarCore simulator config attributes.

```
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setAttribute(Object
attributeType, Object arg)
```

Parameters

`attributeType`

One of the ScSimAnalysisConfigAttributes enum attributes (COM_PORT_NUMBER, LOGGING, OUTPUT_FOLDER).

`arg`

The value the attribute will take.

8.9.1.4 getAttribute

Get the value of a specific SA StarCore simulator config attribute.

```
Object com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.getAttribute(Object
attributeType)
```

Parameters

`attributeType`

The attribute from which the value will be retrieved.

Returns

the specific attribute value.

8.9.2 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::sc::scripting::api::ScSimAnalysisConfig** class.

Following protected members are available:

- load
- setCommPortNumber
- getCommPortNumber
- setOutputFolder
- enableLogging
- isLoggingEnabled

Comprehensive description of each of the above listed functions is as follows:

8.9.2.1 load

Loads configuration using a launch config.

Creates a trace config file in the process (if required).

```
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.load(ILaunchConfiguration
launchConfig)
```

Parameters

```
launchConfig
```

name of the launch configuration.

```
CoreException
```

8.9.2.2 setCommPortNumber

Sets the communication port.

```
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setCommPortNumber(int
commPortNumber)
```

Parameters

```
commPortNumber
```

8.9.2.3 getCommPortNumber

```
int com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.getCommPortNumber()
```

Returns

The stored communication port number

8.9.2.4 setOutputFolder

Sets the results output folder.

```
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.setOutputFolder(String
outputFolder)
```

Parameters

`outputFolder`

the path to the new results folder

8.9.2.5 enableLogging

Enables logging.

```
void com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.enableLogging(boolean enableLog)
```

Parameters

`enableLog`

Enable/disable logging.

8.9.2.6 isLoggingEnabled

Returns the current state of logging state.

```
boolean com.freescale.sa.sc.scripting.api.ScSimAnalysisConfig.isLoggingEnabled()
```

Returns

true/false based on logging state.

8.10 com::freescale::sa::scripting::api::TracePoint

Refer below.

8.10.1 Protected Member Functions

Lists the protected member functions of the **com::freescale::sa::scripting::api::TracePoint** class.

Following protected members are available:

- TracePoint
- getITracepoint

Comprehensive description of each of the above listed functions is as follows:

8.10.1.1 TracePoint

```
com.freescale.sa.scripting.api.TracePoint.TracePoint(ITracepoint _tp)
```

8.10.1.2 getITracepoint

```
ITracepoint com.freescale.sa.scripting.api.TracePoint.getITracepoint()
```

8.10.2 Public Member Functions

Lists the public member functions of the **com::freescale::sa::scripting::api::TracePoint** class.

Following public members are available:

- setEnabled
- getEnabled
- getAction
- getType
- getAddress
- getFileName
- getLineNumber

Comprehensive description of each of the above listed functions is as follows:

8.10.2.1 setEnabled

Set enable state of the tracepoint.

```
void com.freescale.sa.scripting.api.TracePoint.setEnabled(boolean en)
```

Parameters

```
en
```

Enable if true, disable if false.

8.10.2.2 getEnabled

Get enable state of the tracepoint.

```
boolean com.freescale.sa.scripting.api.TracePoint.getEnabled()
```

Returns

True if tracepoint is enabled, false otherwise.

8.10.2.3 getAction

```
Action com.freescale.sa.scripting.api.TracePoint.getAction()
```

Returns

The Action associated with the Tracepoint.

8.10.2.4 getType

```
Type com.freescale.sa.scripting.api.TracePoint.getType()
```

Returns

The Type of the tracepoint.

8.10.2.5 getAddress

Get address of tracepoint.

```
String com.freescale.sa.scripting.api.TracePoint.getAddress()
```

Returns

The address of the tracepoint.

8.10.2.6 getFileName

Get source file name of the tracepoint.

```
String com.freescale.sa.scripting.api.TracePoint.getFileName()
```

Returns

The source file name for the tracepoint.

8.10.2.7 getLineNumber

Get line number of tracepoint.

```
int com.freescale.sa.scripting.api.TracePoint.getLineNumber()
```

Returns

The line number of the tracepoint.

8.11 com.freescale.sa.scripting.IAnalysisConfigFactory

Refer below.

8.11.1 Public Member Functions

Lists the public member functions of the **com.freescale.sa.scripting.IAnalysisConfigFactory** interface.

Following public static member functions are available:

- createAnalysisConfigFromArchive
- createAnalysisConfigFromLaunch

8.11.1.1 createAnalysisConfigFromArchive

Creates an AnalysisConfig object from a trace config file.

```
AnalysisConfig
com.freescale.sa.scripting.IAnalysisConfigFactory.createAnalysisConfigFromArchive(String
configPath)
```

Parameters

`configPath`

The path of the configuration file archive.

Returns

An object. AnalysisConfig

8.11.1.2 createAnalysisConfigFromLaunch

Creates an AnalysisConfig object for a launch config name. May create a trace config file in the process (if needed).

```
AnalysisConfig
com.freescale.sa.scripting.IAnalysisConfigFactory.createAnalysisConfigFromLaunch(String
launchConfigName)
```

Parameters

`launchConfigName`

The launch configuration name.

Returns

An object. AnalysisConfig

Index

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, CodeWarrior, QorIQ, StarCore are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. QorIQ Qonverge is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.