CodeWarrior Development Studio for Power Architecture Processors Tracing and Analysis Tools User Guide

Document Number: CWPASWAUG
Rev. 10.5.1, 01/2016

Contents

Section number **Title** **Page**

Chapter 1
Introduction

Chapter 2
Trace

Chapter 3
Tracepoints

Chapter 4
Performance Analysis

Chapter 1
Introduction

The CodeWarrior for Power Architecture® V10.x Tracing and Performance Analysis tool is used to analyze and collect data of an application. You can use this collected data to identify the bottlenecks, such as slow execution of routines or heavily-used routines within the application.

This manual describes how the CodeWarrior for Power Architecture V10.x Tracing and Performance Analysis tool can be used for the P4080, P4040, P5020/21, P5040, P5021, P2040/41, P5010, P3041, B4860, B4420, B4460, G4860, T4240, T4160, T2080, T1023, T1024, T1040, and T1042 devices. The current implementation of this tool is at engineering level. This chapter presents an overview of this manual and introduces you to the Tracing and Performance Analysis tools.

This chapter describes:

1.1 Limitations

This section describes the limitations of the following tools:

1.1.1 Performance analysis

The Performance Analysis tools have the following limitations:

- Limited integration with CodeWarrior for Power Architecture V10.x tool chain.

- Does not share the same launch configuration as the debugger.
- Connection configuration to the target is done manually by editing a text file. There is no GUI support for this at this time. However, there is a connection configuration option to extract the configuration from the current application.
- The tools need to stop the core once during its initialization. This is because some core registers cannot be accessed while the device is running. Once this is done, for future runs in the same session with the same configuration, the target system runs without further interference.
- The tools need to stop the core whenever core changes the events that are required to be measured by the core.
- The tools does not support analysis of performance events collected in the Nexus stream.
- The graphical presentation of data is limited in its ability to manipulate displayed data. Scales, labels, colors, and other items cannot be changed.
- The user needs to launch CCS manually before starting the Performance Analysis tools. CCS must be running while the Performance Analysis tools are being used.
- The master reference clock set as core 0; PMC3 is hard coded.

1.1.2 Trace

The tracing tools have the following limitations:

- No hypervisor support.
- No support for dynamically loaded code, self-modifying code, and code relocated at runtime.
- No support for Linux User Space Application Trace collection.

1.2 Overview

Each chapter of this manual describes a different area of Tracing and Performance Analysis tools.

The table below describes each chapter in the manual.

Table 1-1. Manual Contents

Chapter	Description
Introduction	Introduces Tracing and Performance Analysis tool and provides limitations of Performance Analysis and Trace (this chapter)

Table continues on the next page...

Table 1-1. Manual Contents (continued)

Chapter	Description
Trace	Describes the trace collection process and how to use trace data for debugging
Tracepoints	Describes how to start and stop trace collection at specific instruction addresses
Performance Analysis	Describes how to configure, collect, and analyze scenarios in the QorIQ processors as well as in the Bx and Tx devices

1.3 Accompanying documentation

The Documentation page describes the documentation included in this version of CodeWarrior Development Studio for Power Architecture Processors. You can access the Documentation page by:

- a shortcut link on the Desktop that the installer creates by default
- opening START_HERE.html in <*CWInstallDir*>*PA**Help* folder

Chapter 2
Trace

Tracing is a technique that obtains diagnostic and performance information about a program's execution. This information is typically used for debugging purposes, and additionally, depending on the type and detail of information contained in a trace log, to diagnose common problems with the software. The trace log includes information about the trace source, type of event, description of the event and time stamp value.

This chapter describes:

- Configuration
- Collecting data
- Viewing data
- Preparing Aurora interface for collecting Aurora Nexus trace
- Importing trace data file
- Multicore tracing

2.1 Configuration

In order to collect trace, you need a project to define your application, a launch configuration to set up a debug connection to the target, and a trace configuration to define the type of trace data you would like to collect.

This topic describes the following sub-topics.

- Creating new project
- Configuring trace

2.1.1 Creating new project

Create a new CodeWarrior bareboard project or open an existing project before you start collecting the trace data.

To create a new bareboard project:

1. Start the CodeWarrior IDE.
2. From the IDE menu bar, choose **File > New > CodeWarrior Bareboard Project Wizard**.

   The **Create a CodeWarrior Bareboard Project** page appears.

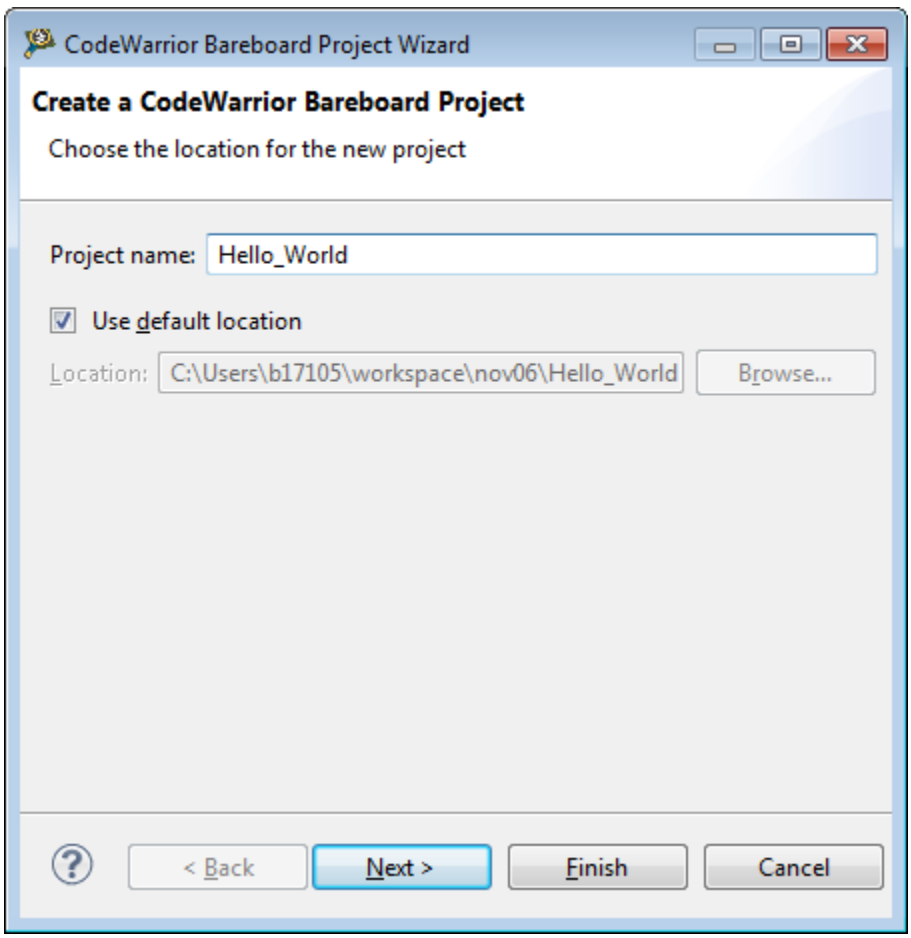


Figure 2-1. CodeWarrior Bareboard Project Wizard

3. Enter a name for the new project in the **Project Name** text box.
4. The **Location** text box shows the default workspace location. To change this location, deselect the **Use default location** checkbox and click **Browse** to choose a new location.
5. Use subsequent dialog to specify a new location.
6. Click **OK**.

   The dialog returns you to the **Create a CodeWarrior Bareboard Project** page, which now shows the new location.

7. Click **Next**.

   The **Processor** page appears.

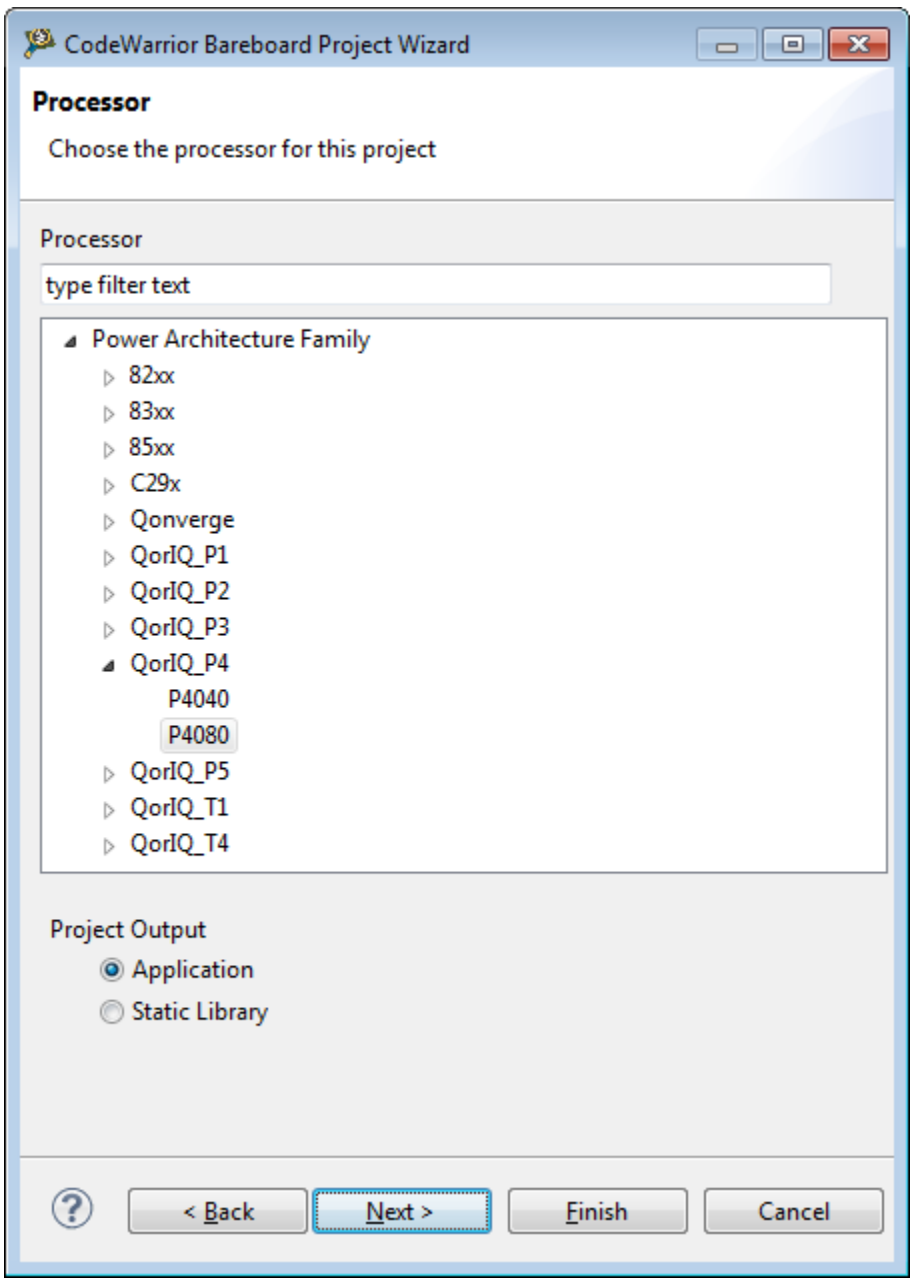


Figure 2-2. Processor page

8. Expand the **QorIQ_P4** tree control and select **P4080**.

NOTE

In this chapter, P4080 has been taken as an example. The settings may differ based upon the selection of devices in the **Processor** page.

9. Click **Next**.

The **Debug Target Settings** page appears.

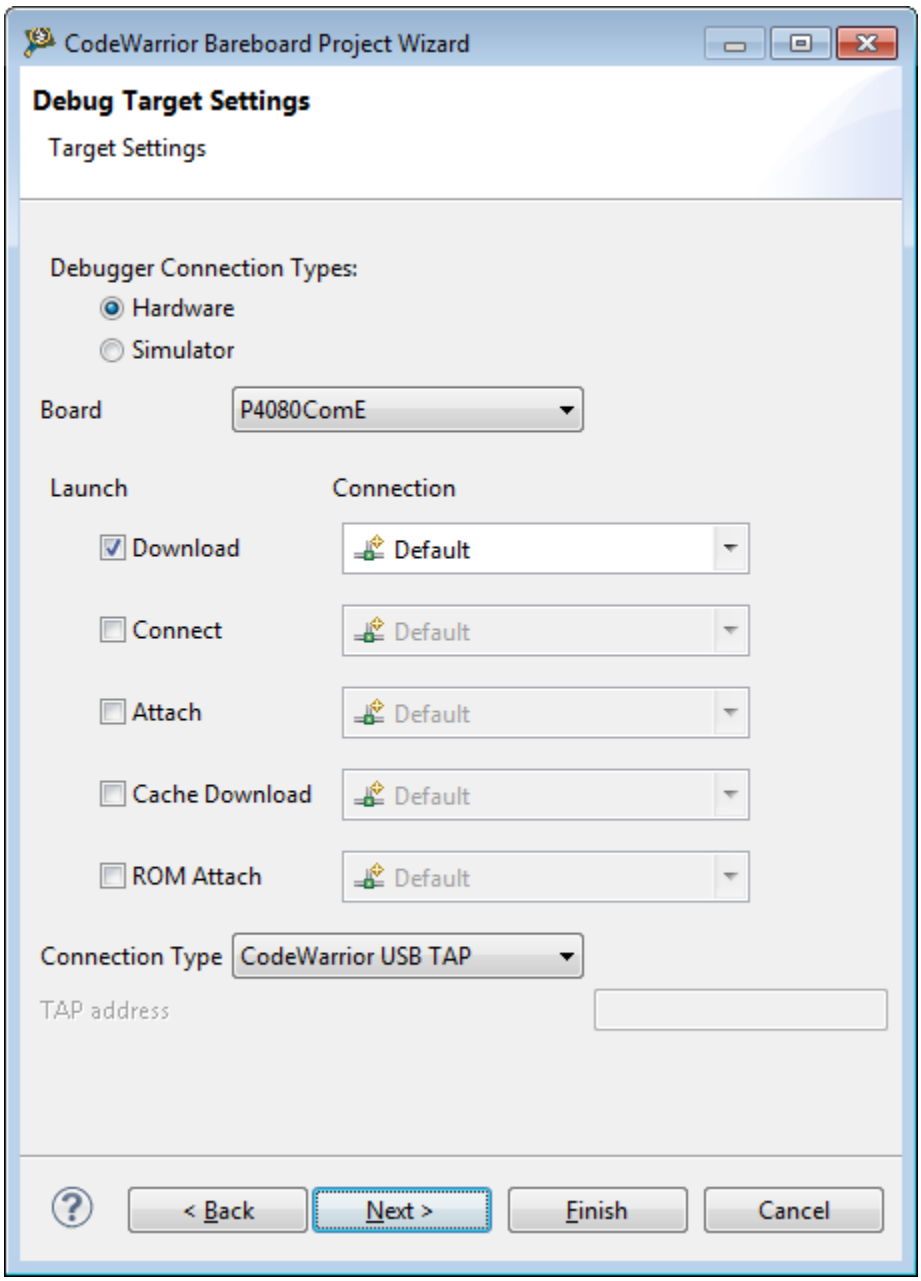


Figure 2-3. Debug Target Settings page

10. From the **Board** pop-up menu, choose **P4080DS**.
11. From the **Connection Type** pop-up menu, choose the TAP that is connected to the target.

NOTE

The **Gigabit TAP + Trace** option corresponds to a Gigabit TAP that includes an Aurora daughter card, which allows you to collect Nexus trace in a real-time non-intrusive fashion from the high-speed serial trace port (the Aurora

interface). To know more about collecting Nexus trace using Aurora, see Preparing Aurora interface for collecting Aurora Nexus trace.

The Gigabit TAP devices that include an Aurora daughter card have three ways to send debug commands to the target, depending on the configuration of the P4080 board.

Table 2-1. Gigabit TAP + Trace connection type options

Option	Description
JTAG over JTAG cable	Sends JTAG commands over the JTAG cable
JTAG over Aurora cable	Sends JTAG commands over the Aurora cable

NOTE

The settings of the **Gigabit TAP + Trace** debug connection should match the jumper settings on the target board.

12. Enter the TAP address in the **TAP address** text box.
13. Click **Next**.

    The **Build Settings** page appears.

14. Click **Next**.

    The **Configurations** page appears.

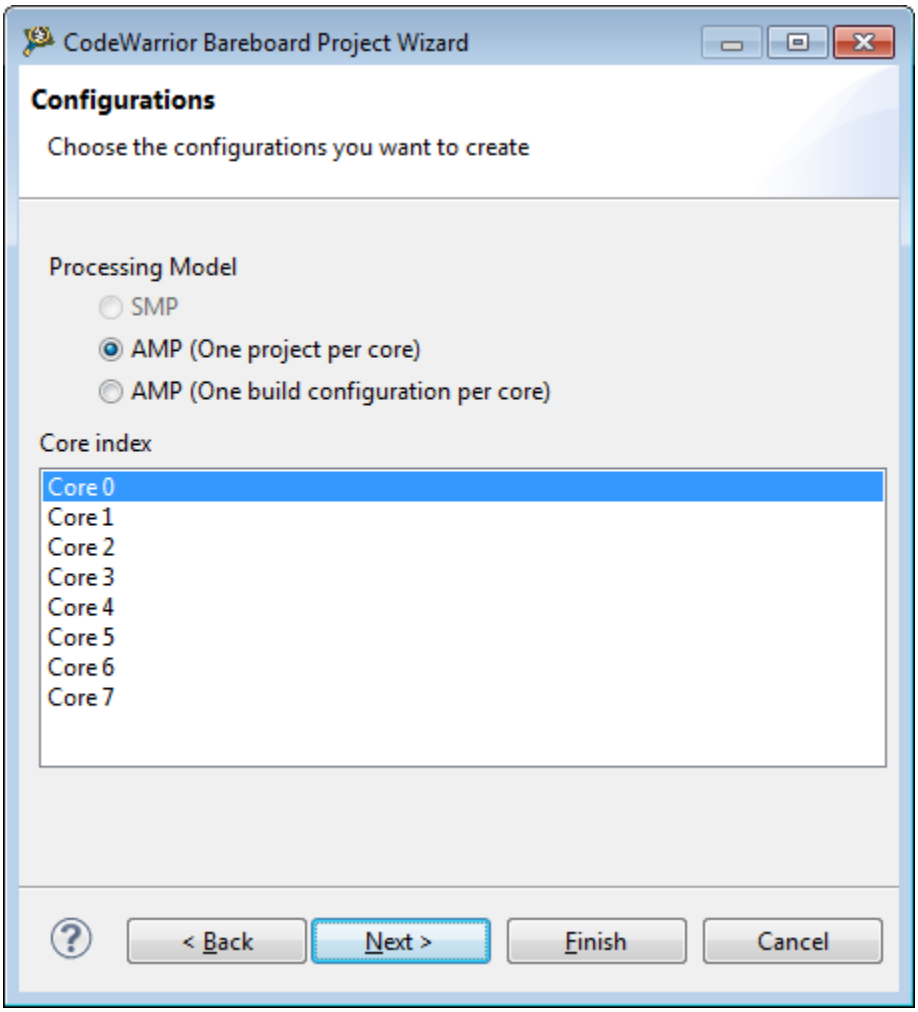


Figure 2-4. Configurations page

15. From the **Processing Model** group, select **AMP (One project per core)**.
16. From the **Core Index** list, select **Core 0**.

NOTE

For the bareboard applications, launch the application on **Core 0** in order to configure the system. In addition, you can launch other applications on **Core 1** - **Core 7**, using the same target initialization file.

17. Click **Next**.

    The **Trace Configuration** page appears.

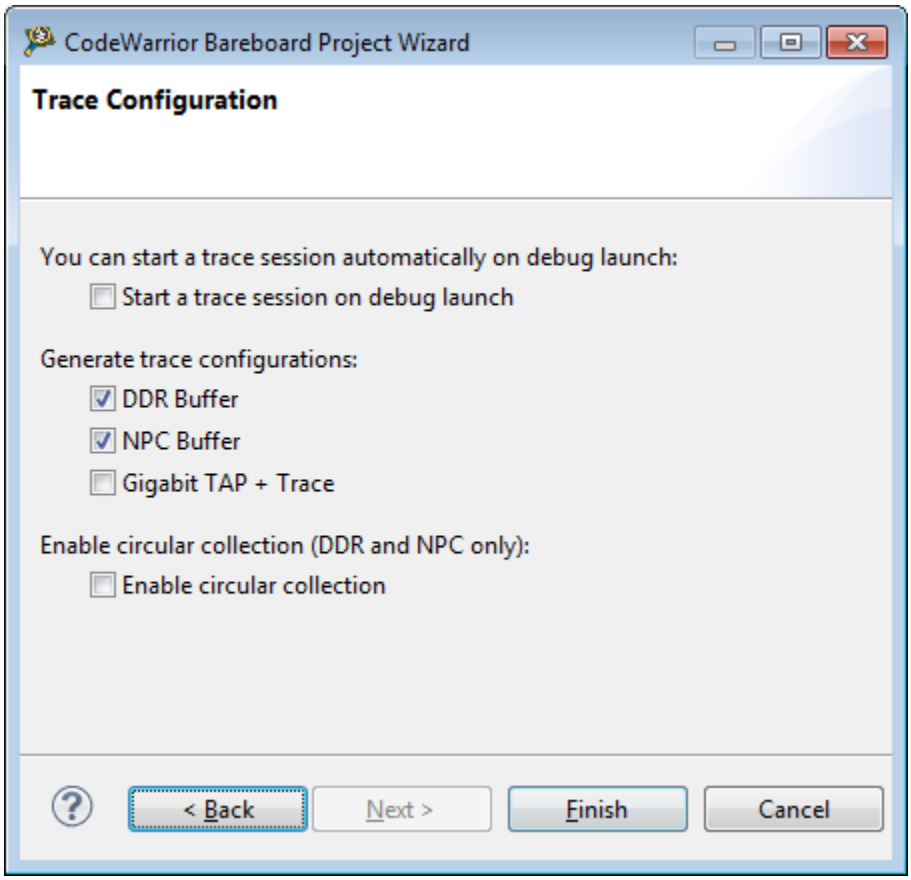


Figure 2-5. Trace Configuration page

18. To start trace session, select the **Start a trace session on debug launch** checkbox.
19. From **Generate trace configurations for the following types of trace sources** group, choose the desired option.

Table 2-2. Trace collection source options

Option	Description
DDR Buffer	This collection method sends trace to a Double Data Rate (DDR) memory buffer.
NPC Buffer	This collection method sends trace data to a small dedicated trace buffer.
Gigabit TAP + Trace	This collection method collects trace data on a Gigabit TAP+Trace debug connection.
Enable circular collection	Turns on circular collection for DDR and NPC-based trace. In the Circular Collection mode, the tracing continues even after the buffer is full overwriting the old data with the new data.

20. Click **Finish**.

    The IDE creates a project according to your specifications.

21. From the IDE menu bar, choose **Window>Show View>CodeWarrior Projects**.

The **CodeWarriorProjects** view appears and displays the project you just created.

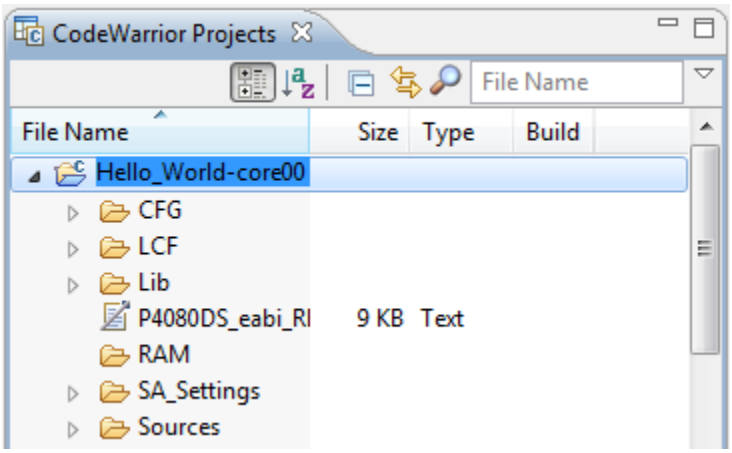


Figure 2-6. CodeWarrior Projects view

22. From the IDE menu bar, choose **Project > Build Project**.

The **Build Project** dialog appears; the build tools generate an executable program.

NOTE

For more information on creating a CodeWarrior project, see the *CodeWarrior Development Studio for Power Architecture Processors Targeting Manual* (document CWPADBGUG) in the folder: <*CWInstallDir*>*PA**Help**PDF*, where <*CWInstallDir*> is the directory where CodeWarrior for Power Architecture V10.x is installed.

2.1.2 Configuring trace

You need to define the trace configuration before debugging the application for trace collection.

NOTE

This section documents debugger features that are specific to Trace. For more information on other debugger features of CodeWarrior for Power Architecture Processors product, see the *CodeWarrior Development Studio for Power Architecture Processors Targeting Manual (document CWPADBGUG)* in the folder: <*CWInstallDir*>*PA**Help**PDF*, where <*CWInstallDir*> is the directory where CodeWarrior for Power Architecture V10.x is installed.

To define a trace configuration:

1. From the IDE menu bar, choose **Run > Debug Configurations**.

   The **Debug Configurations** dialog appears.

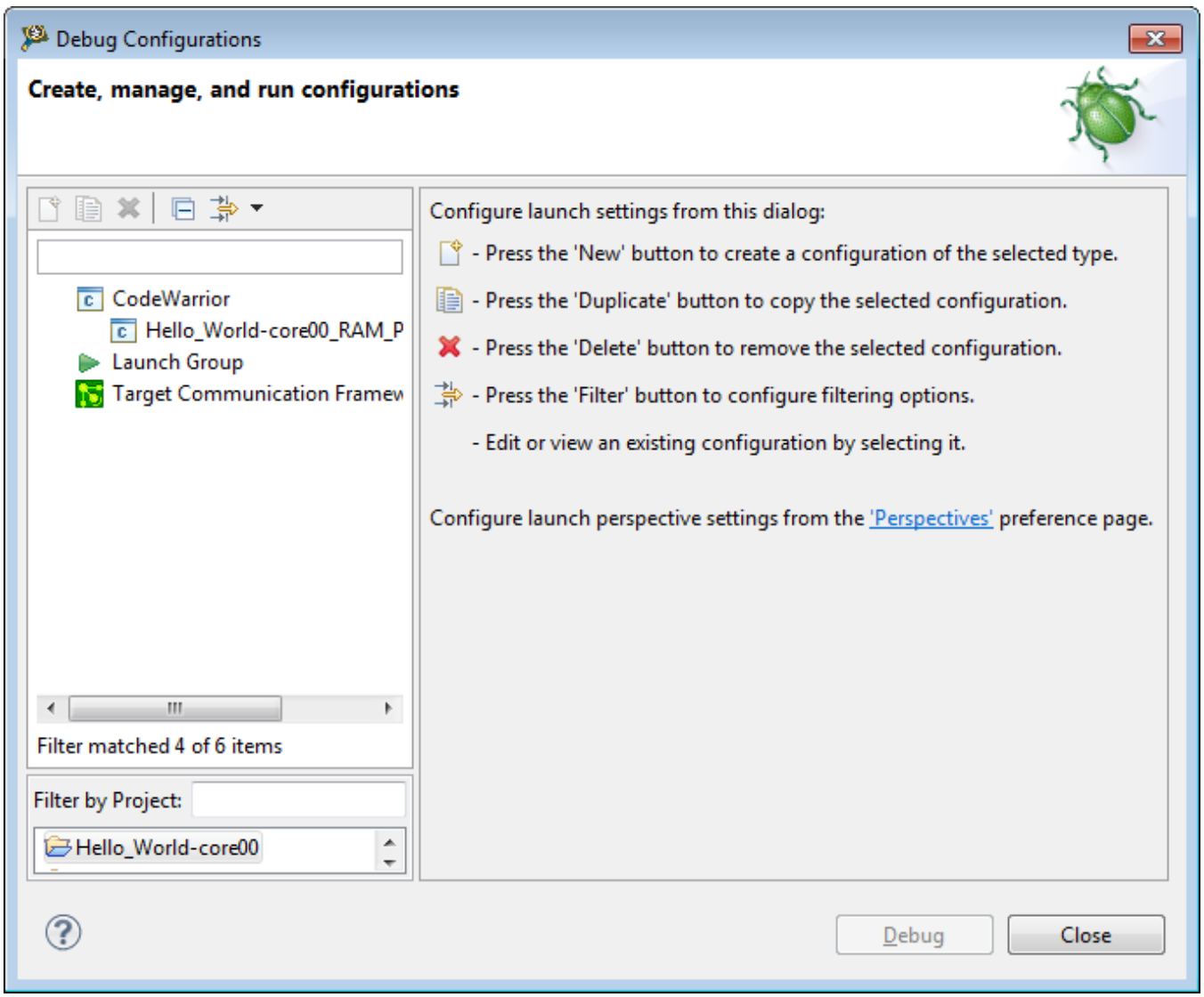


Figure 2-7. Debug Configurations dialog

2. In the left pane of this dialog, expand the **CodeWarrior** tree control.
3. Select the launch configuration corresponding to your project. For example, **Hello_World-core0_RAM_P4080_Download**.

   A set of tabbed configuration panels appears in the right pane of the dialog.

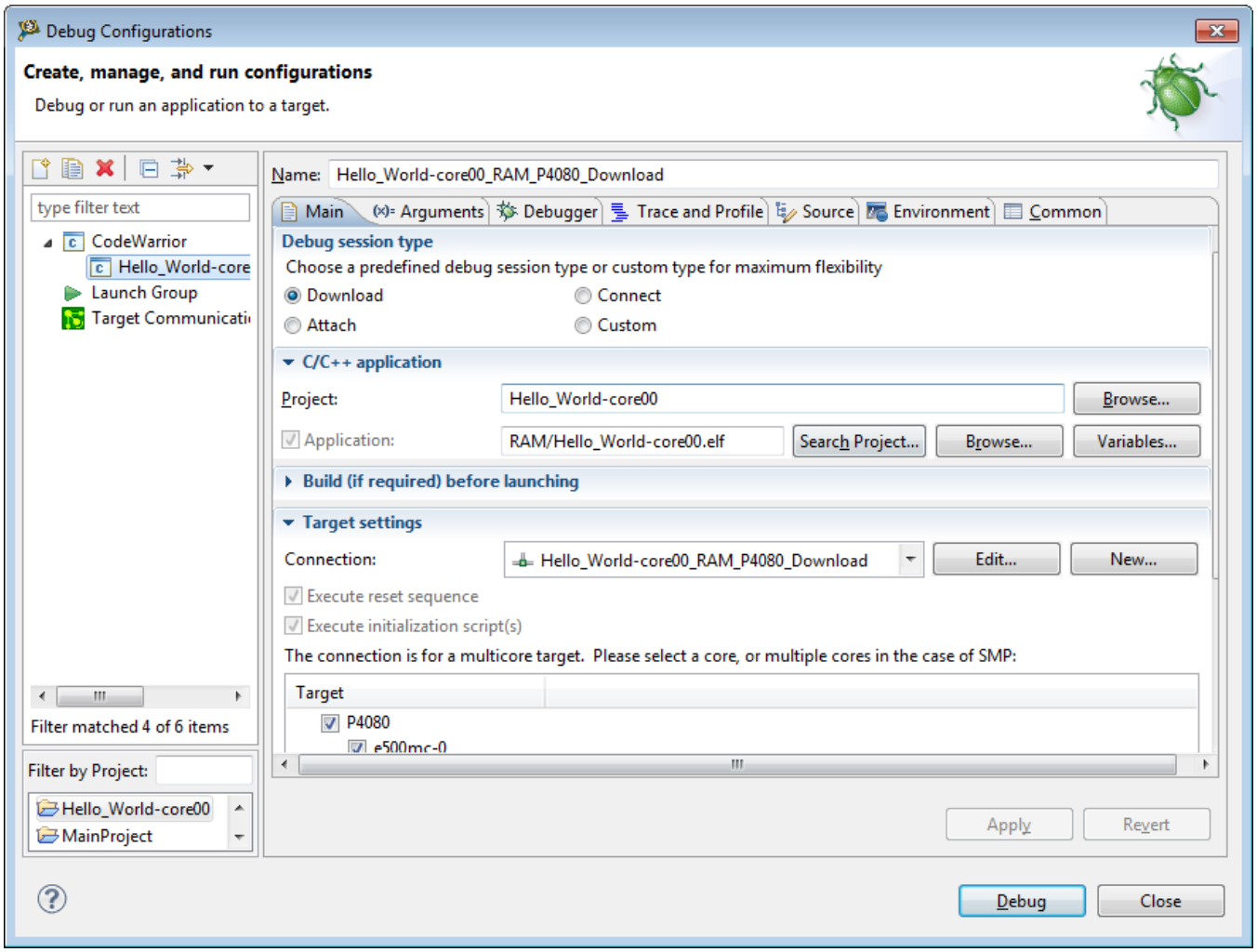


Figure 2-8. Debug Configurations dialog - Create, manage, and run configurations

4. Click the **Trace and Profile** tab.

The trace configuration options appear, as shown in the figure below.

NOTE

Trace is not supported for all processors. Also, trace may be licensed. If an unsupported processor is selected in the RSE Target Configuration, or if trace is licensed and you do not have a valid license, the trace configuration options in the **Trace and Profile** tab appear disabled.

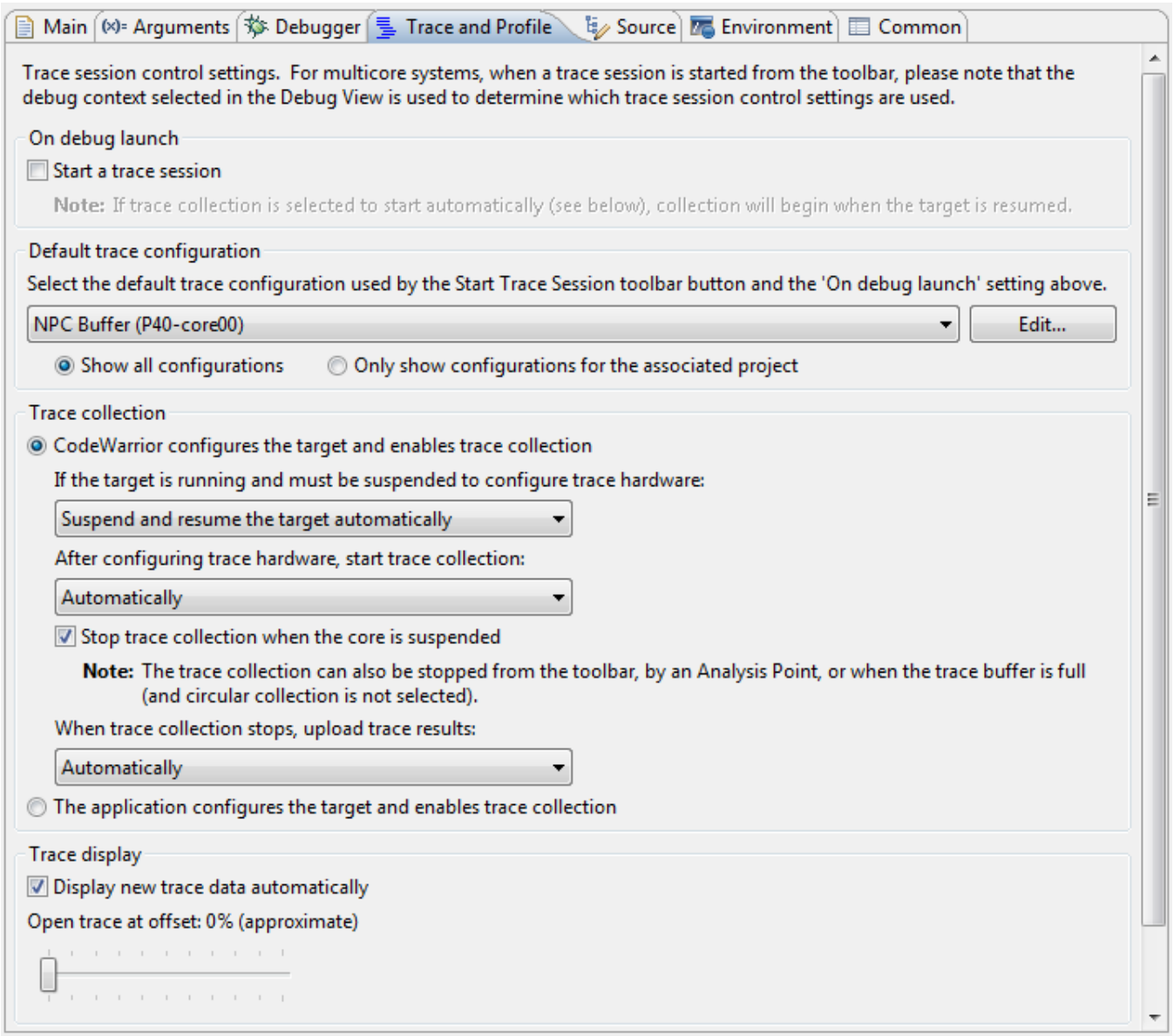


Figure 2-9. Trace and Profile tab

5. Select the **Start a trace session** checkbox for the trace session to start immediately on debug launch.

NOTE

If you have selected the **Start a trace session on debug launch** checkbox in Figure 2-5 while creating a new bareboard project, you will see the **Start a trace session** checkbox selected by default in the **Trace and Profile** tab.

6. From the **Default trace Configuration** pop-up menu, choose the desired default configuration. It also provides two options:

- **Show all configurations** - displays all configurations in the Default trace Configuration pop-up menu.
- **Only show configuration for the associated project** - displays those configurations that are related to the selected project.

7. From the **Trace Collection** group, select one of the following options for collecting trace data.

Table 2-3. Trace collection options with description

Option	Sub-option	Description
CodeWarrior configures the target and enables trace collection		Lets you configure and control trace collection for the trace session started from CodeWarrior.
	If the target is running and must be suspended to configure trace hardware	• **Suspend and resume the target Automatically** - Suspends the target automatically for trace configuration. If this option is selected, and you choose to configure trace while the target is running, the target suspends immediately while trace configuration is applied, and then resumes automatically. • **Wait until the target is resumed manually** - Lets you suspend and resume the target for trace configuration manually. If this option is selected, and you choose to configure trace while the target is running, the configuration is changed, but not applied to the target until the target is suspended and resumed.
	After configuring trace hardware, start trace collection	• **Automatically** - Starts trace collection immediately after you press resume. • **Manually from the toolbar or by an Analysis Point** - Lets you start trace session and configure trace hardware with trace collection turned off. Trace collection will be turned on later by clicking the **Start Collection** button in the toolbar or by

Table continues on the next page...

Table 2-3. Trace collection options with description (continued)

Option	Sub-option	Description
		executing code at an analysis point or tracepoint. For details on the toolbar, see Figure 2-33 and Table 2-20.
	Stop trace collection when the core is suspended	If selected, stops trace collection automatically when the target is suspended. If you do not select this checkbox, you can stop trace collection either using the **Stop Trace Collection** button from the toolbar, or by using an analysis point or tracepoint. If you do not choose any of these options, the trace collection will stop after buffer is full or when application finishes execution. For details on the toolbar, see Figure 2-33 and Table 2-20.
	When trace collection stops, upload trace results	• **Automatically** - Saves data to the `Trace.dat` file automatically after collection completes or is stopped. • **Manually from the toolbar** - Lets you use the toolbar to save the trace data manually to the `Trace.dat` file. **NOTE:** For details on the toolbar, see Table 2-20.
The application configures the target and enables trace collection		Lets you start collecting trace data for the trace session using an application other than CodeWarrior. That is, the application where you make your own configuration of trace registers in the trace hardware explicitly and CodeWarrior is not involved. In this case, all the above options are disabled. This feature is useful in Attach configurations to collect trace. For example, on an Attach configuration, when something goes wrong on a target running an application or your application throws an exception,

Table 2-3. Trace collection options with description

Option	Sub-option	Description
		you can start the trace session and get the trace data collected until the exception.

8. Perform the following settings in the **Trace Display** group.
   a. Select the **Display new trace data automatically** checkbox to display the newly uploaded trace data automatically.
   b. In the **Open trace at offset** option, adjust the slider to select an approximate memory location for opening the collected trace data. This setting is stored with the trace data, and thus will always control where the particular data is opened. The slider is approximate and provides selections from 0 to 100% in 10% increments. This option is normally set to 0%, but will be set to 100% if you select the **Enable circular collection** checkbox in the **Trace Configurations** page of the **New Power Architecture** wizard.

This topic contains the following sub-topics:

- Modifying selected trace configuration
- Configuring trace for e6500 core

2.1.2.1 Modifying selected trace configuration

You can create, manage, and edit configuration for a trace session to collect trace on particular trace settings.

To modify the selected configuration, click **Edit** in the **Trace and Profile** tab of the **Debug Configurations** dialog. The **Trace Configurations** dialog appears, as shown in the following figure. You can use this dialog to create, delete, rename, and modify trace configuration.

NOTE

The trace configuration options change according to the trace collection sources you select from the **Trace** page.

NOTE

Steps 1-13 are optional; you may use these steps to modify the trace configurations.

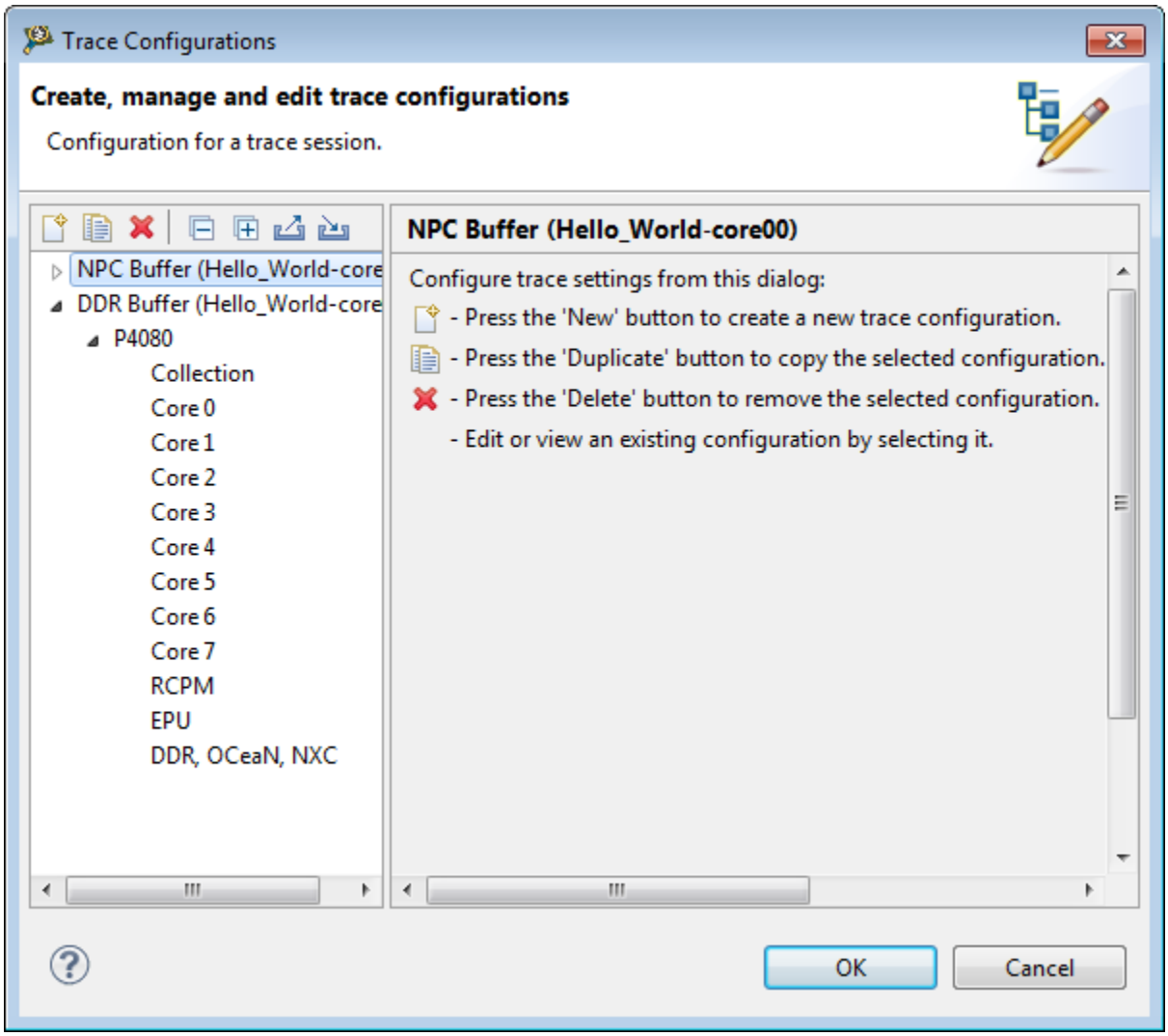


Figure 2-10. Trace Configurations dialog

1. In the left pane of this dialog, expand the appropriate tree control. For example, **DDR Buffer (Hello_World-core0)**.

   The **P4080** tree control appears in the left pane.

   NOTE

   The tree control appears according to the target device you select: **P4080**, **P4040**, or **P5020**.

2. Expand the **P4080** tree control and choose **Collection**.

   The **Collection** configuration options appear in the right pane of the dialog. These options let you modify how and where the trace data is collected.

3. From the **Collection Method** pop-up menu, choose one of the following:
   - **DDR Buffer** - For information on the options that appear for DDR Buffer collection method, see the table and figure below.

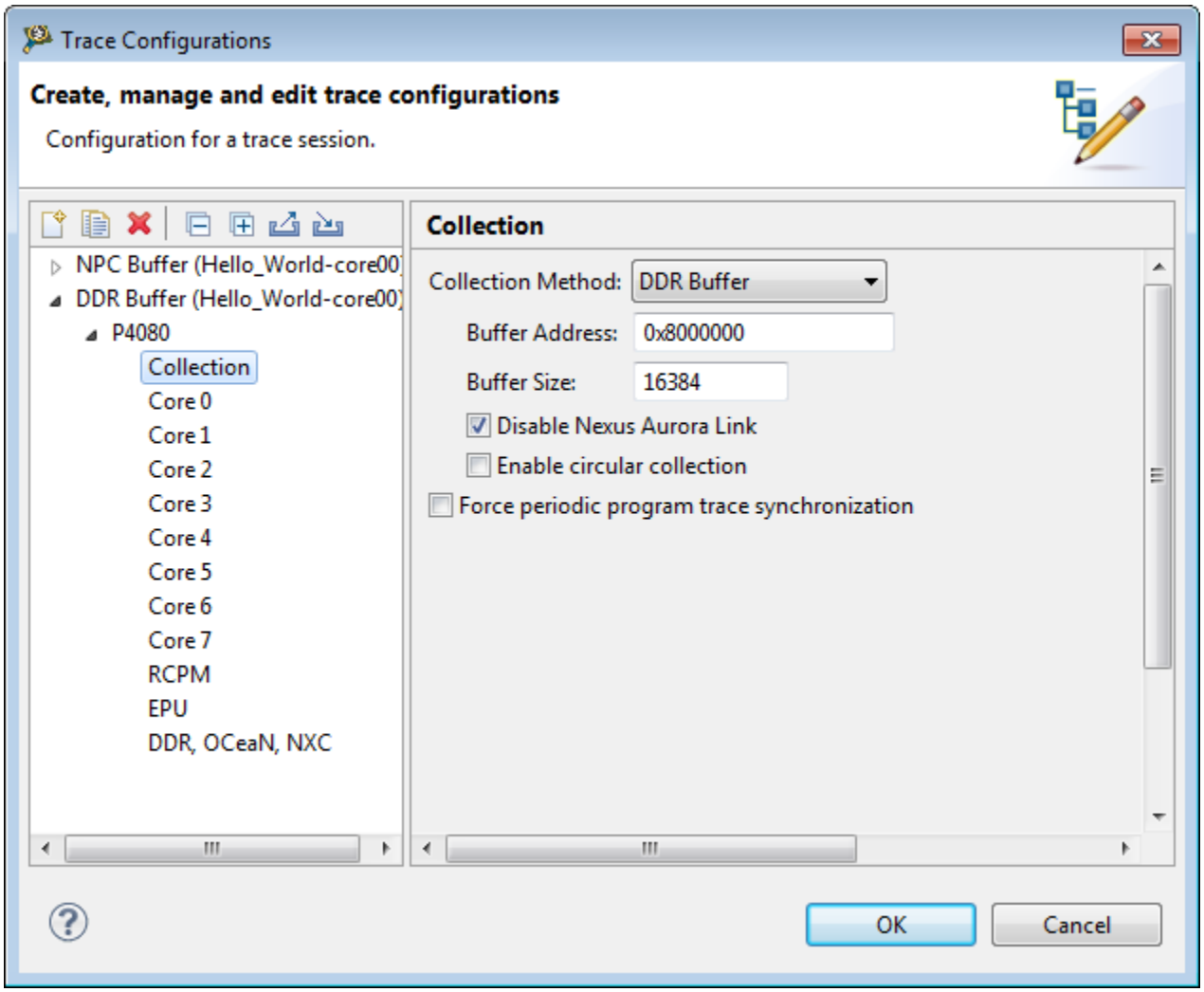


Figure 2-11. DDR buffer configuration

Table 2-4. DDR buffer configuration options

Option	Description
Buffer Address	Enter the memory address from where the buffer begins collecting trace.
Buffer Size	Enter the size of the collection buffer.
Disable Nexus Aurora Link	Select to disable the Nexus Aurora Link (NAL) layer.
Enable circular collection	Select to allow the buffer to contain the most recent *n* bytes of trace data. The circular buffer overwrites the old data with new in case the buffer is full. **NOTE:** Circular collection is useful when the events of interest may occur at some unknown point in the program during execution. In this situation, an application can be configured to set up a circular buffer

Table continues on the next page...

Table 2-4. DDR buffer configuration options (continued)

Option	Description
	and to stop tracing when the event of interest occurs, or soon thereafter. The buffer then contains the last n events to occur before the point of interest. Circular buffer collection is available for DDR or NPC based collection buffers. Tracing continues until you explicitly terminate the debug session using the stop trace button in the debug view, or cancel the trace collection task from the **Progress** view.
Force periodic program trace synchronization	Select to generate a synchronization trace message periodically. This is useful in situations when the branch buffer is overrun by many branches between naturally occurring synchronization messages. However, the buffered branch information will be discarded leaving a sampled view of code execution rather than a complete branch-by-branch trace of execution. **NOTE**: Use this setting for Linux kernel tracing. This option is available only when the advanced properties of the trace configuration editor are selected.

- **NPC Buffer** - The options for NPC are same as DDR. For information on the options that appear for NPC Buffer collection method, see the above table.

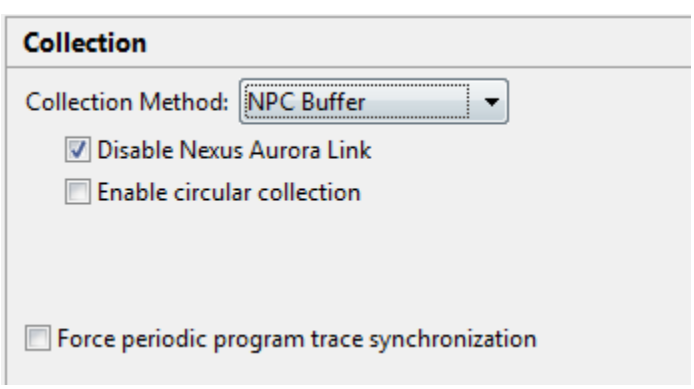


Figure 2-12. NPC buffer configuration

- **Gigabit TAP + Trace** - For information on the options that appear for Gigabit TAP + Trace collection method, see the table above.

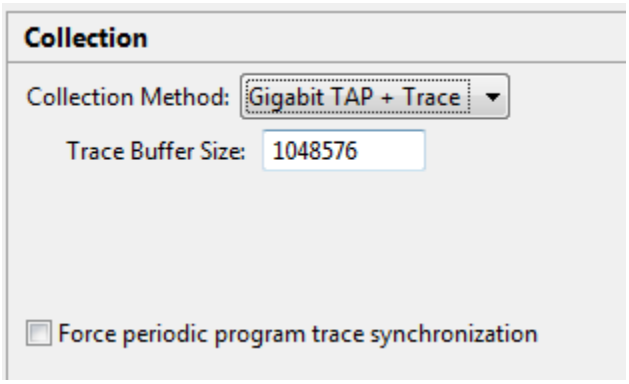


Figure 2-13. Gigabit TAP + Trace configuration

4. In the left pane of the **Trace Configurations** dialog, choose a core from the **P4080** tree control. For example, **Core 0**.

   A set of tabbed configuration panels appears in the right pane of the dialog with the **Program Trace** tab in front, as shown below. These configuration options control the trace produced in the processor cores.

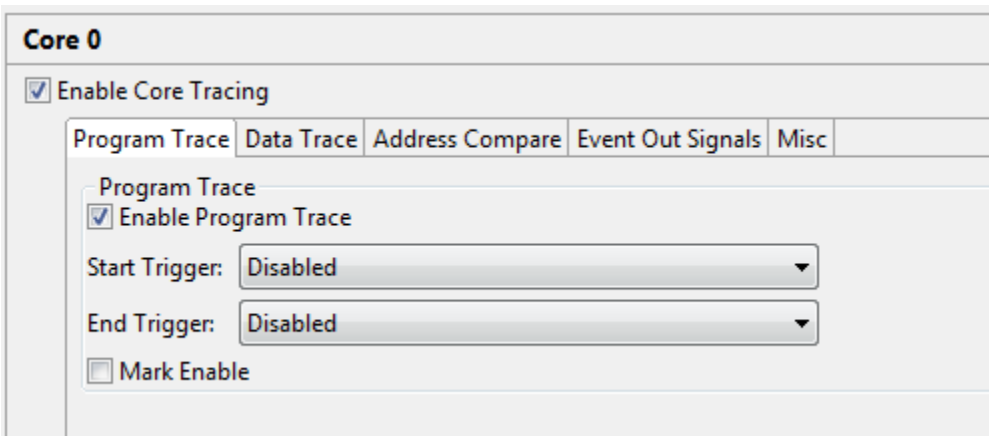


Figure 2-14. Trace Configurations dialog - Program Trace tab

Table 2-5. Program Trace options

Option	Description
Enable Core Tracing	Select to produce the trace output for **Core 0**.
Enable Program Trace	Select to turn on the tracing of program flow for this core.
Start Trigger	Specify an event that turns on the program trace.
End Trigger	Specify an event that turns off the program trace.
Mark Enable	Select to generate Nexus Program Trace selectively only for processes that set MSR[PMM]. Program trace will be masked or disabled for all other processes.

NOTE

The e500mc core program trace output can also be triggered by the watchpoint events. For example, watchpoint 1: IAC1 (Instruction Address Comparator 1) event, watchpoint 2: IAC2 event, or watchpoint 3: interrupt taken.

5. Click **Data Trace** tab.

   The data trace configuration panel appears.

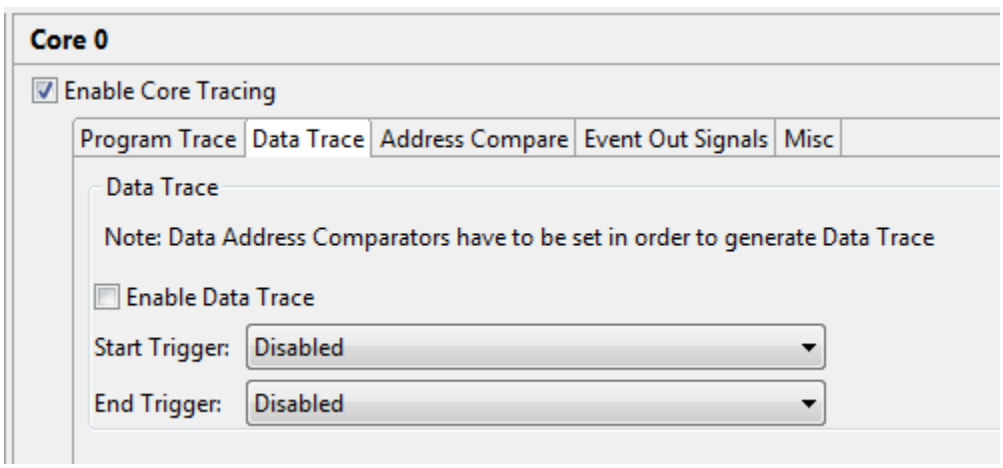


Figure 2-15. Trace Configurations dialog - Data Trace tab

Table 2-6. Data Trace options

Option	Description
Enable Data Trace	Select to turn on the tracing of selected memory reads and writes by this core.
Start Trigger	Specify an event that enables the data trace.
End Trigger	Specify an event that disables the data trace.

NOTE

Data trace will only occur in the range(s) specified by the Data Address Compare settings explained in Table 2-8. If no range is specified, no trace will occur.

NOTE

The e500mc core data trace output can also be triggered by the watchpoint events. For example, watchpoint 1: IAC1 event, watchpoint 2: IAC2 event, or watchpoint 3: interrupt taken.

6. Click **Address Compare** tab.

   The address compare configuration panel appears.

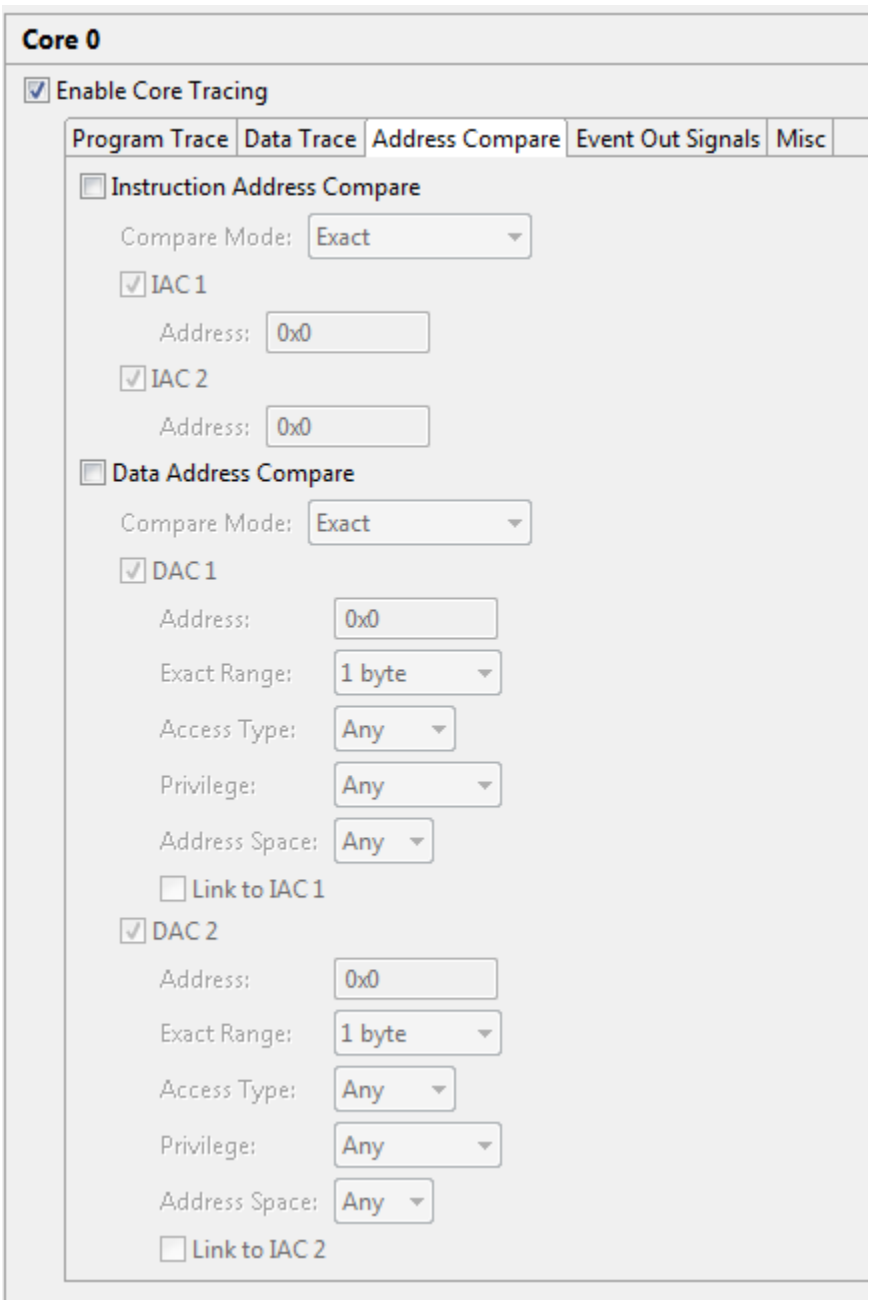


Figure 2-16. Trace Configurations dialog - Address Compare tab

7. Select the appropriate checkbox to enable the **Instruction Address Compare** or **Data Address Compare** options. For more information on these options, see the tables below.

Table 2-7. Instruction Address Compare options

Option	Suboption	Description
Compare Mode		Select the type of instruction address comparison to be made.
	Exact	IAC 1 debug conditions occur only if the address of the instruction fetch is equal to the value specified in IAC 1. IAC 2 debug conditions occur only if the address of the instruction fetch is equal to the value specified in IAC 2.
	BitMatch	IAC 1 debug conditions occur only if the address of the instruction fetch, ANDed with the contents of IAC 2 is equal to the contents of IAC 1, also ANDed with the contents of IAC 2. IAC 2 debug conditions do not occur.
	Inclusive Range	IAC 1 debug conditions occur only if the address of the instruction fetch is greater than or equal to the value specified in IAC 1 and less than the value specified in IAC 2. IAC 2 debug conditions do not occur.
	Exclusive Range	IAC 1 debug conditions occur only if the address of the instruction fetch is less than the value specified in IAC 1 or is greater than or equal to the value specified in IAC 2. IAC 2 debug conditions do not occur.
IAC 1		Select to enable instruction address 1 comparator.
	Address	Enter the address to set the value of instruction address comparator 1.
IAC 2		Select to enable instruction address 2 comparator.
	Address	Enter the address to set the value of instruction address comparator 2.

Table 2-8. Data Address Compare options

Option	Suboption	Description
Compare Mode		Select the type of data address comparison to be made.
	Exact	DAC 1 debug conditions occur only if the address of the instruction fetch is in the range specified in the DAC 1 **Address** and **Range** fields. DAC 2 debug conditions occur only if the address of the instruction fetch is in the range specified in the DAC 2 **Address** and **Range** fields.
	BitMatch	DAC 1 debug conditions occur only if the address of the instruction fetch, ANDed with the DAC 2 **Address** setting is equal to the DAC 1 **Address** setting, also ANDed with the DAC 2 **Address** setting. DAC 2 debug conditions do not occur.
	Inclusive Range	DAC 1 debug conditions can occur only if the address of the instruction fetch is greater than or equal to the value specified in DAC 1 and less than the value specified in DAC 2. DAC 2 debug conditions do not occur.
	Exclusive Range	DAC 1 debug conditions occur only if the address of the instruction fetch is less than the value specified in DAC 1 or is greater than or equal to the value specified in DAC 2. DAC 2 debug conditions do not occur.
DAC 1		Select to enable DAC 1.
	Address	Enter the address to set the value of data address comparator 1.
	Exact Range	Select the range of addresses to match in Exact mode.
	Access Type	Select an option to let a data store or data load trigger the data address 1 comparator. **NOTE:** For e500 core, only **Stores** are traced. There is no trace if you select **Loads**.

Table continues on the next page...

Table 2-8. Data Address Compare options (continued)

Option	Suboption	Description
	Privilege	Specify the privilege level to enable the DAC 1 debug condition.
	Address Space	Specify the memory address space to enable the DAC 1 debug condition.
	Link to IAC 1	Select to link DAC 1 with IAC 1. When linked to IAC 1, the DAC 1 debug event is qualified based on whether the instruction also generated an IAC 1 debug condition.
DAC 2		Select to enable data address comparator 2.
	Address	Enter the address to set the value of data address comparator 2.
	Range	Select the range of addresses to match in Exact mode.
	Access Type	Select an option to let a data store or data load trigger the data address 2 comparator.
	Privilege	Specify the privilege level to enable the DAC 2 debug condition.
	Address Space	Specify the memory address space to enable the DAC 2 debug condition.
	Link to IAC 2	When linked to IAC 2, the DAC 2 debug event is qualified based on whether the instruction also generated an IAC 2 debug condition.

8. Click the **Event Out Signals** tab.

   The event out signals configuration panel appears.

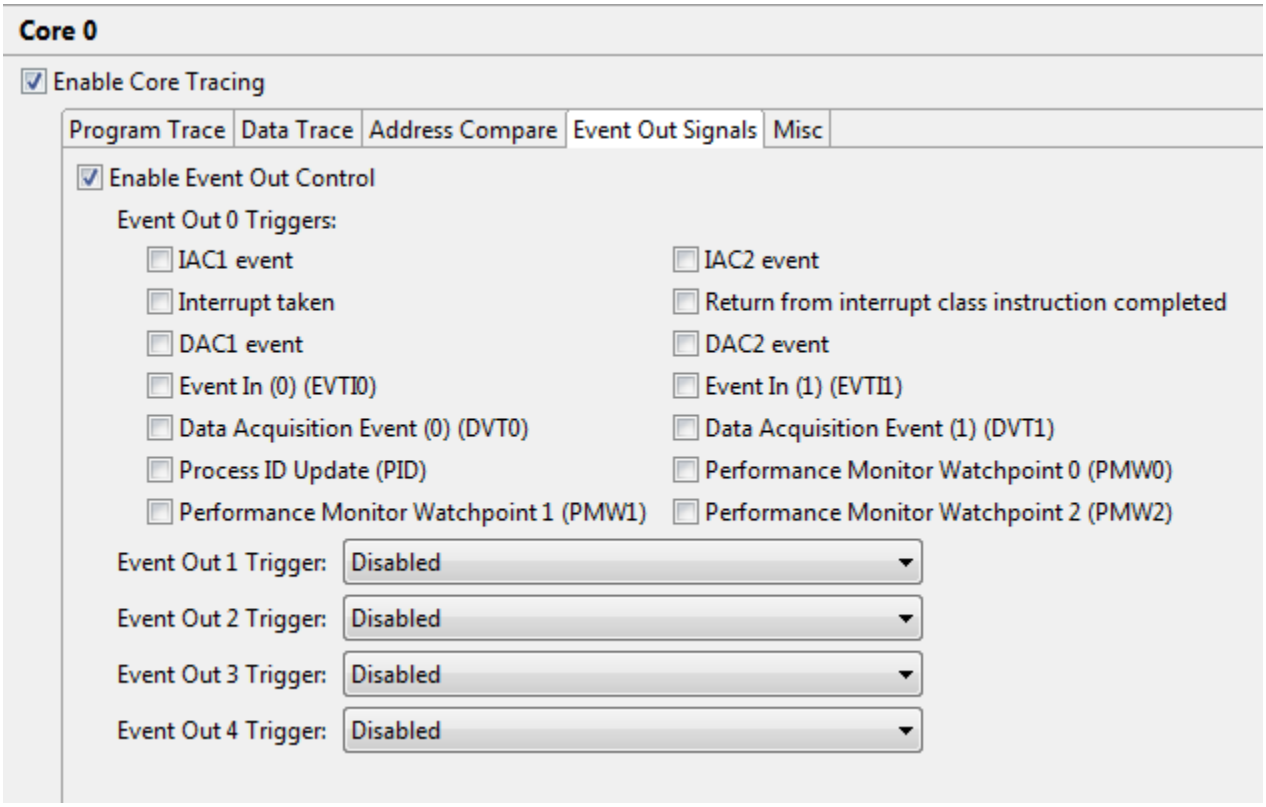


Figure 2-17. Trace Configurations dialog - Event Out Signals tab

a. Enter the debugger event values in the respective text boxes.

Table 2-9. Event Out Signals fields

Field	Description
Event Out 0 Triggers	Determines which debug event(s) triggers Event Out 0. You can select the debug event(s) from the available checkboxes.
Event Out 1 Trigger	Determines which debug event triggers Event Out 1.
Event Out 2 Trigger	Determines which debug event triggers Event Out 2.
Event Out 3 Trigger	Determines which debug event triggers Event Out 3.
Event Out 4 Trigger	Determines which debug event triggers Event Out 4.

NOTE

The events 1-4 can also be triggered by the watchpoint events. For example, watchpoint 1: IAC1 event, watchpoint 2: IAC2 event, or watchpoint 3: interrupt taken. However, event 0 can be triggered if any of the

watchpoints selected by the event 0 mask occurs, that is the selected events are OR-ed.

b. Click **Apply** to apply the settings.

9. Click the **Misc** tab.

The miscellaneous configuration panel appears.

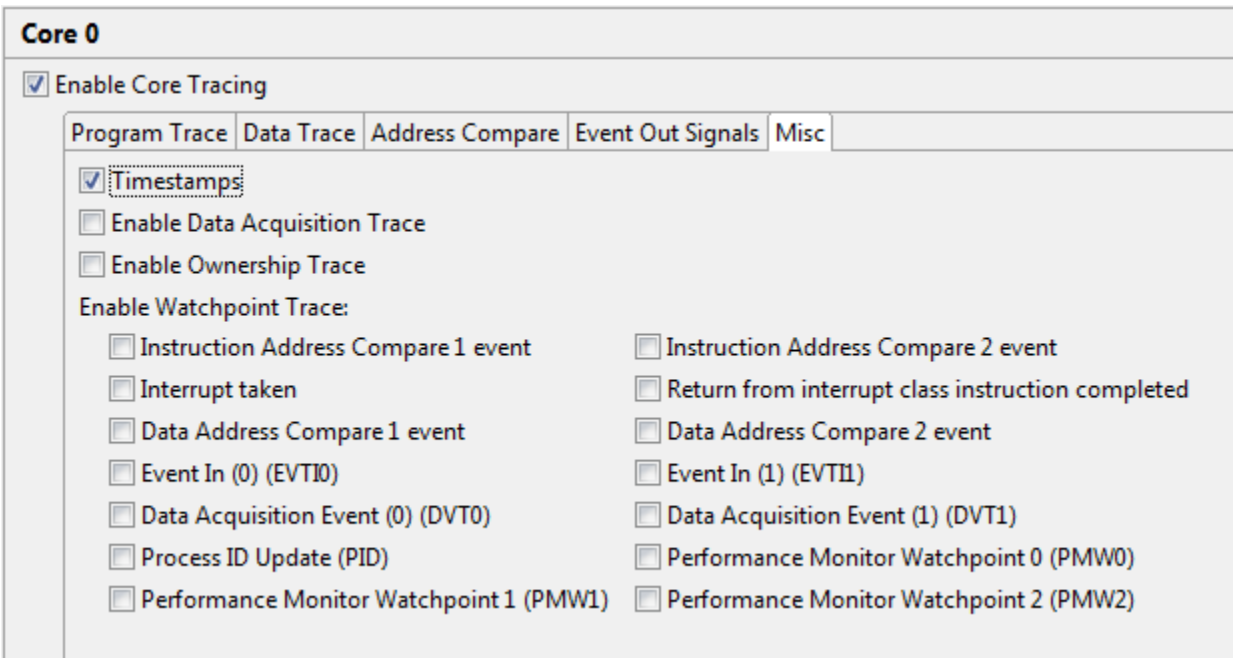


Figure 2-18. Trace Configurations dialog - Misc tab

Table 2-10. Misc fields

Field	Description
Timestamps	Select to configure trace messages to include timestamps.
Enable Data Acquisition Trace	Select to generate Data Acquisition messages.
Enable Ownership Trace	Select to generate ownership messages.
Enable Watchpoint Trace	Controls which watchpoint events are enabled to produce Watchpoint Trace Messages. You can select the watchpoint events from the available checkboxes. These watchpoint events, when triggered, could optionally create a Nexus watchpoint message.

NOTE

The e500mc core watchpoint trace output can also be triggered if any of the selected watchpoints by this mask occurs (the selected events are OR-ed).

10. In the left pane of the **Trace Configurations** dialog, choose **RCPM** from the **P4080** tree control. RCPM passes signals between the cores and the rest of the chip.

    The RCPM configurations page appears in the right pane of the dialog.

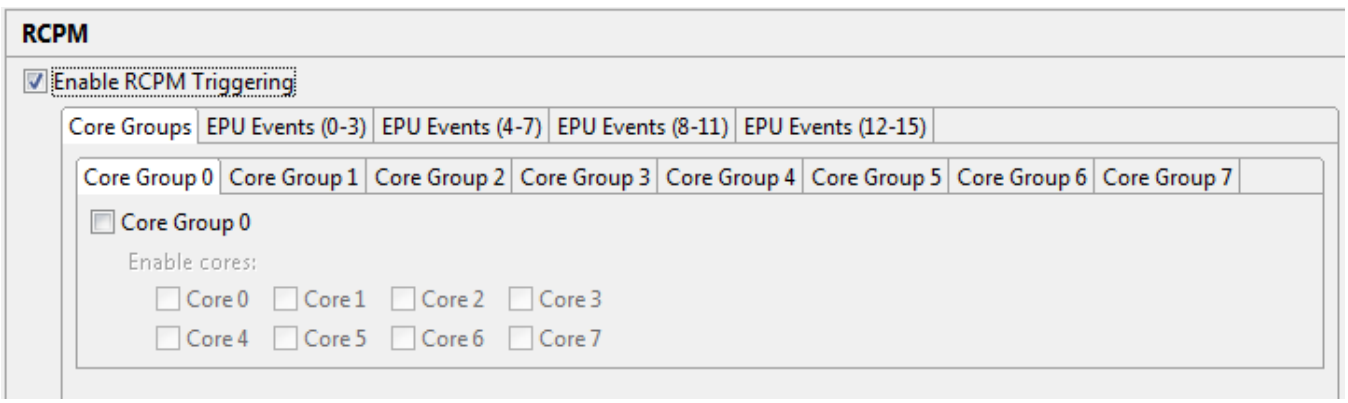


Figure 2-19. RCPM Configuration page

a. To enable RCPM triggering, select **Enable RCPM Triggering**.
b. Click the **Core Groups** tab.

   The core group configuration options appear. You can use these options to create groups of cores that can be signaled together.

c. Select the checkboxes for the required core corresponding to a core group.
d. Click **Apply** to save the settings.
e. Click the **EPU Events (0-3)**, **EPU Events (4-7)**, **EPU Events (8-11)**, **EPU Events (12-15)** tabs.

   The EPU event configuration options appear. These options enable EPU events to trigger actions in the cores. EPU processes the events that can be sent to different units on the chip, including the RCPM.

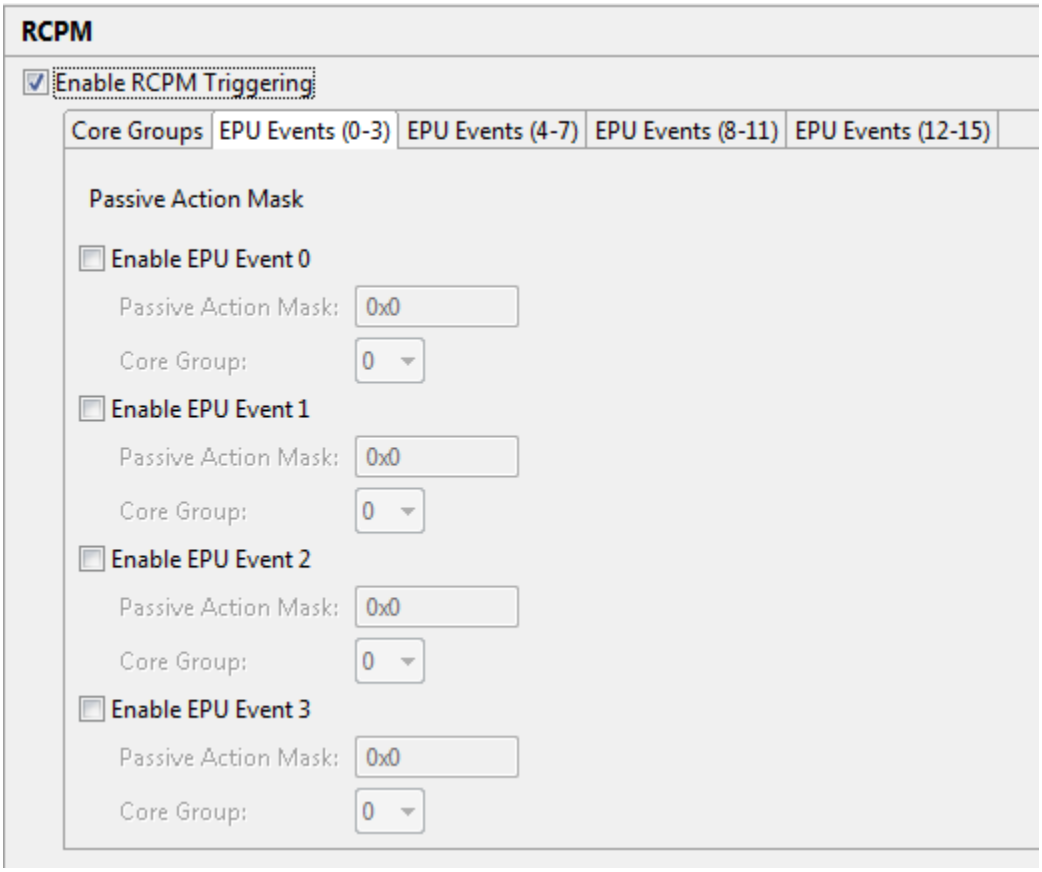


Figure 2-20. EPU Events settings

The following table describes the EPU Events configuration options.

Table 2-11. EPU Events options

Option	Description
Enable EPU Event 0 - 15	Select to turn on the event for triggering from the corresponding Sequencing and Combining Unit (SCU) in the EPU. When an EPU event is selected, the corresponding **Passive Action Mask** and **Core Group** options become available.
Passive Action Mask	Specify the actions that will occur in the cores while the trigger is generated.
Core Group	Select the core group that the EPU event affects.

NOTE

A core group action control register is provided for each event generated by the EPU.

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

11. In the left pane of the **Trace Configurations** dialog, choose **EPU** from the **P4080** tree control.

    The EPU configuration page appears in the right pane of the dialog.

    a. To enable EPU events, select **Enable EPU Events**.

       You can configure EPU counters (32 in number) as well as program each Sequencing and Combining Unit (SCU) to trigger an EPU event.

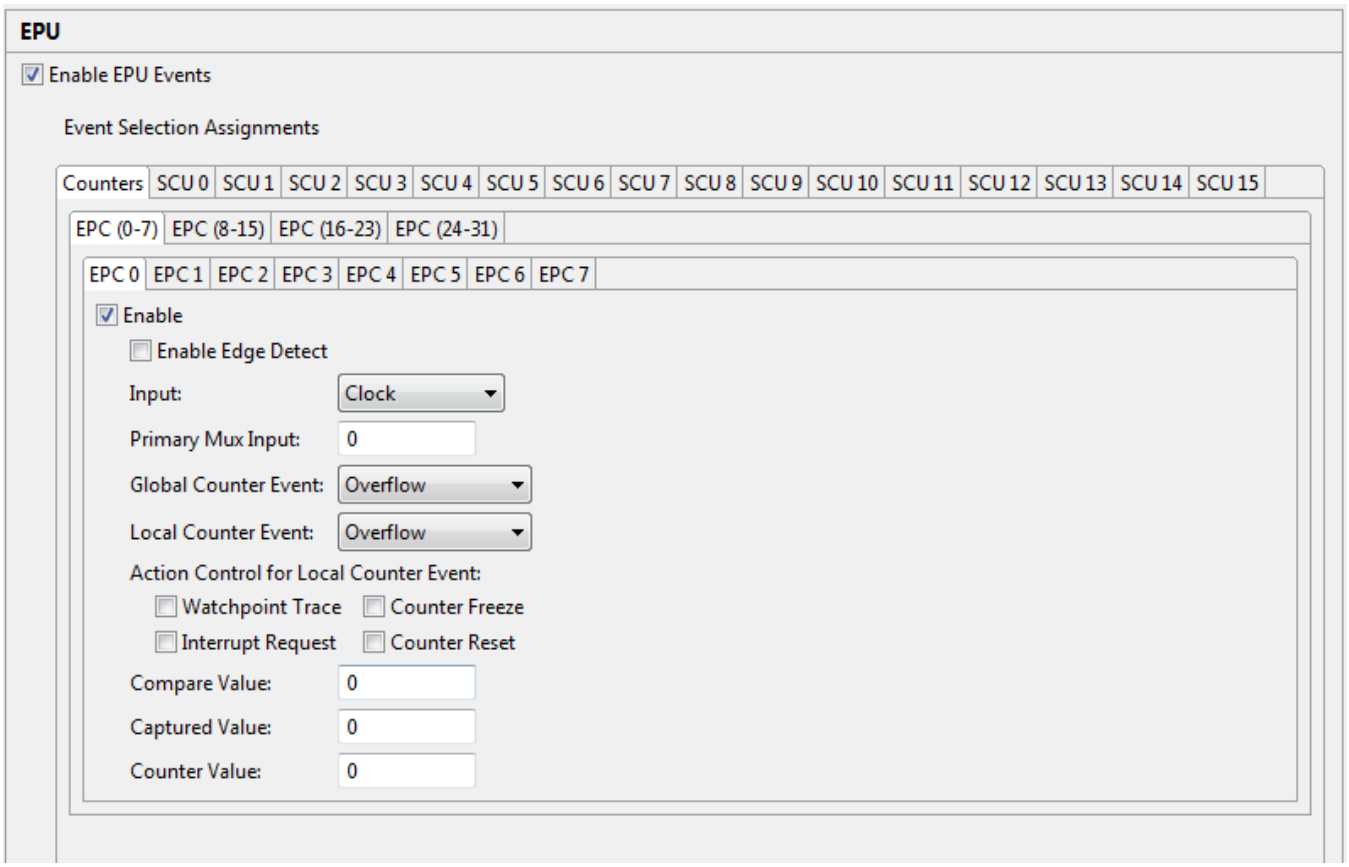


Figure 2-21. EPU Configuration page

The table below describes the EPU counter configuration options. These configuration options remain valid for **EPU 0** - **EPU 31**.

Table 2-12. EPU Counter configuration options

Option	Description
Enable	Select to turn on the EPU counter unit.
Enable Edge Detect	Select for the EPU counter to be triggered only for one clock cycle, when the input changes to high.

Table continues on the next page...

Table 2-12. EPU Counter configuration options (continued)

Option	Description
Input	Select to select the input for the counter. Following options are available: • **Clock** - Free running. Used only with EDE = 0. No count with EDE = 1 = Edge-detected counter input. • **SCU** - SCU event (SCU events are direct-mapped to counters) • **Next Counter** - Event from the next counter • **Primary Mux** - Counter-specific event selected from primary muxes.
Primary Mux Input	Choose one of 64 possible signals to be used as an input. The input can differ between the counters.
Global Counter Event	Select the counter condition that needs to be used as an event to be sent to the EPU Mux Unit (Counter Muxes and SCU Muxes).
Local Counter Event	Select the counter condition to be used as an event for local action (local counter event). The selected event causes actions according to the option(s) selected in **Action Control for Local Counter Event**.
Action Control for Local Counter Event	Select the action control for local counter event from the following options. Reset is higher priority than freeze if they are both set. • **Watchpoint Trace** - Local counter event to be sent out in the watchpoint trace message to the debug port if watchpoint trace control is not set. • **Counter Freeze** - Local counter event freezes the counter. • **Interrupt Request** - Local counter event signals an interrupt request. • **Counter Reset** - Local counter event resets the counter.
Compare Value	Specify the EPU counter compare register. It holds 16-bit values that are compared against corresponding counter to produce counter events.
Captured Value	Specify the EPU counter capture register. It holds sampled values of the counter.
Counter Value	Specify the EPU counter. The 16-bit counter that may be used to delay events, count distances, count latencies, or count events.

The table below describes the SCU configuration options. These configuration options remain valid for **SCU 0 - SCU 15**.

Table 2-13. SCU configuration options

Option	Description
Enable SCU	Select to turn on the SCU unit.
Enable Edge Detect	Select for the event to be triggered only for one clock cycle, when the input changes to high.
Invert Output	Select to invert the event logic at the output.
Enable Trace	Select to turn on the SCU to generate trace messages.
Enable Input 0 - 3	Select to turn on the SCU input for processing. When an SCU input option is selected, the corresponding **Input Source**, **Necessary**, and **Invert Input** options become available.
Input Source	Enter the signal to use for the SCU input.
Necessary	Select to make the input mandatory. To create the trigger logic, the mandatory inputs are ANDed together and then ORed with the other inputs, which are called sufficient inputs.
Invert Input	Select to invert the input before it enters the logic.

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

12. In the left pane of the **Trace Configurations** dialog, choose **DDR, OCeaN, NXC** from the **P4080** tree control.

    The DDR, OCeaN, NXC configuration page appears in the right pane of the dialog. NXC collects and filters trace from DDR and Ocean.

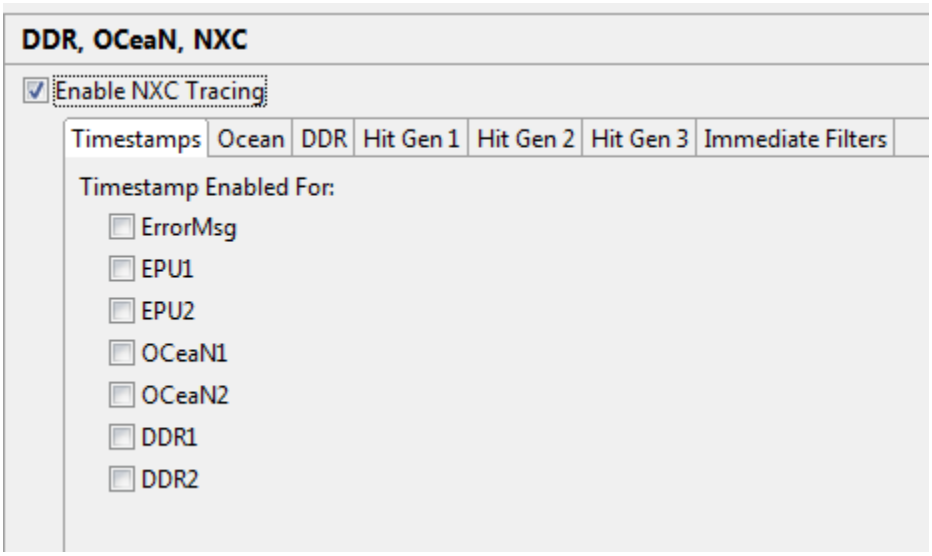


Figure 2-22. DDR, OCeaN, NXC configuration page

a. To allow the debug client to send trace through NXC, select the **Enable NXC Tracing** checkbox.

   The debug client filters the trace data before sending it to the collector.

b. In the **Timestamps** tab, enable timestamp for the required events.
c. Click the **Ocean** tab.

   The Ocean configurations options appear in the right pane of the dialog. The Ocean configuration allows you to produce trace from the on-chip network. The Ocean configuration options correspond to the bits in OCDICR0 (Ocean Debug Control register). Ocean tracing enables Nexus Trace of traffic on the Ocean high-speed (Serial Rapid IO/PCI Express) peripheral I/O interface.

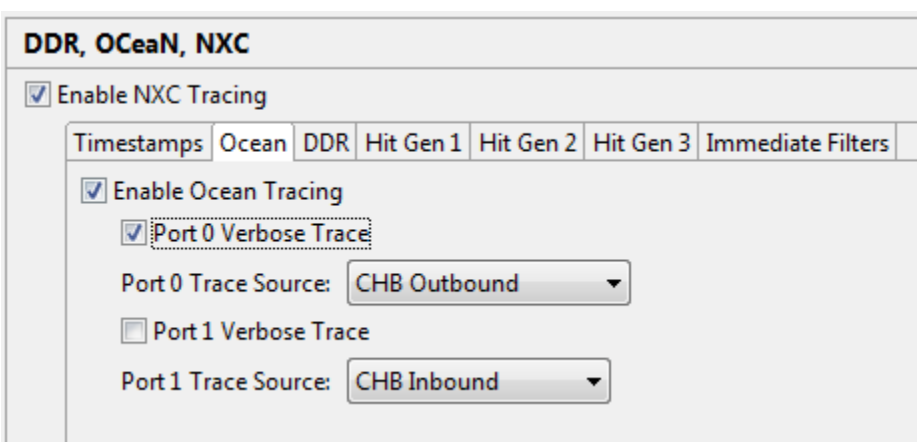


Figure 2-23. Ocean configuration page

d. Configure the Ocean options.

The following table describes the Ocean tracing configuration options.

Table 2-14. Ocean configuration options

Option	Description
Enable Ocean Tracing	Select to turn on Ocean tracing.
Port 0 Verbose Trace	Select to generate verbose trace messages for trace port0.
Port 0 Trace Source	Available when **Port 0 Verbose Trace** is selected. Select the message source from port 0.
Port 1 Verbose Trace	Select to generate verbose trace messages for trace port1.
Port 1 Trace Source	Available when **Port 1 Verbose Trace** is selected. Select the message source from port 1.

e. Click the **DDR** tab.

The DDR configuration page appears.

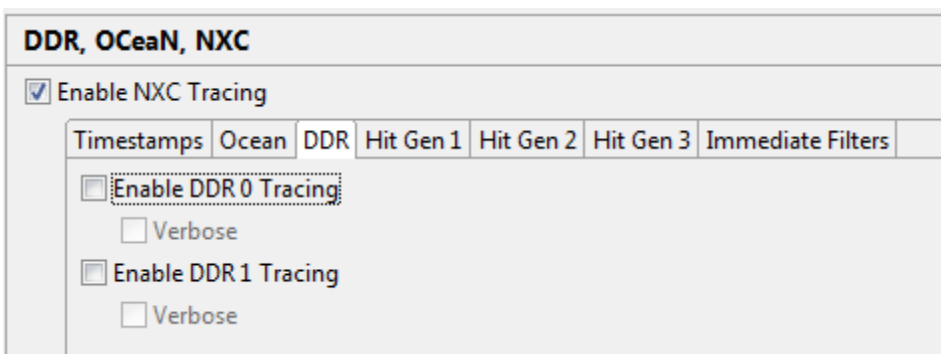


Figure 2-24. DDR configuration page

f. Select the desired DDR trace options.

Table 2-15. DDR configuration options

Option	Description
Enable DDR 0 Tracing	Turns on tracing for DDR 0.
Verbose	Generates verbose trace messages for DDR 0.
Enable DDR 1 Tracing	Turns on tracing for DDR 1.
Verbose	Generates verbose trace messages for DDR 1.

g. Choose any one or all of the **Hit Gen 1**, **Hit Gen 2**, **and Hit Gen 3** tabs to make comparisons and generate signals that are used for filtering.

The respective tab opens with the comparator configurations.

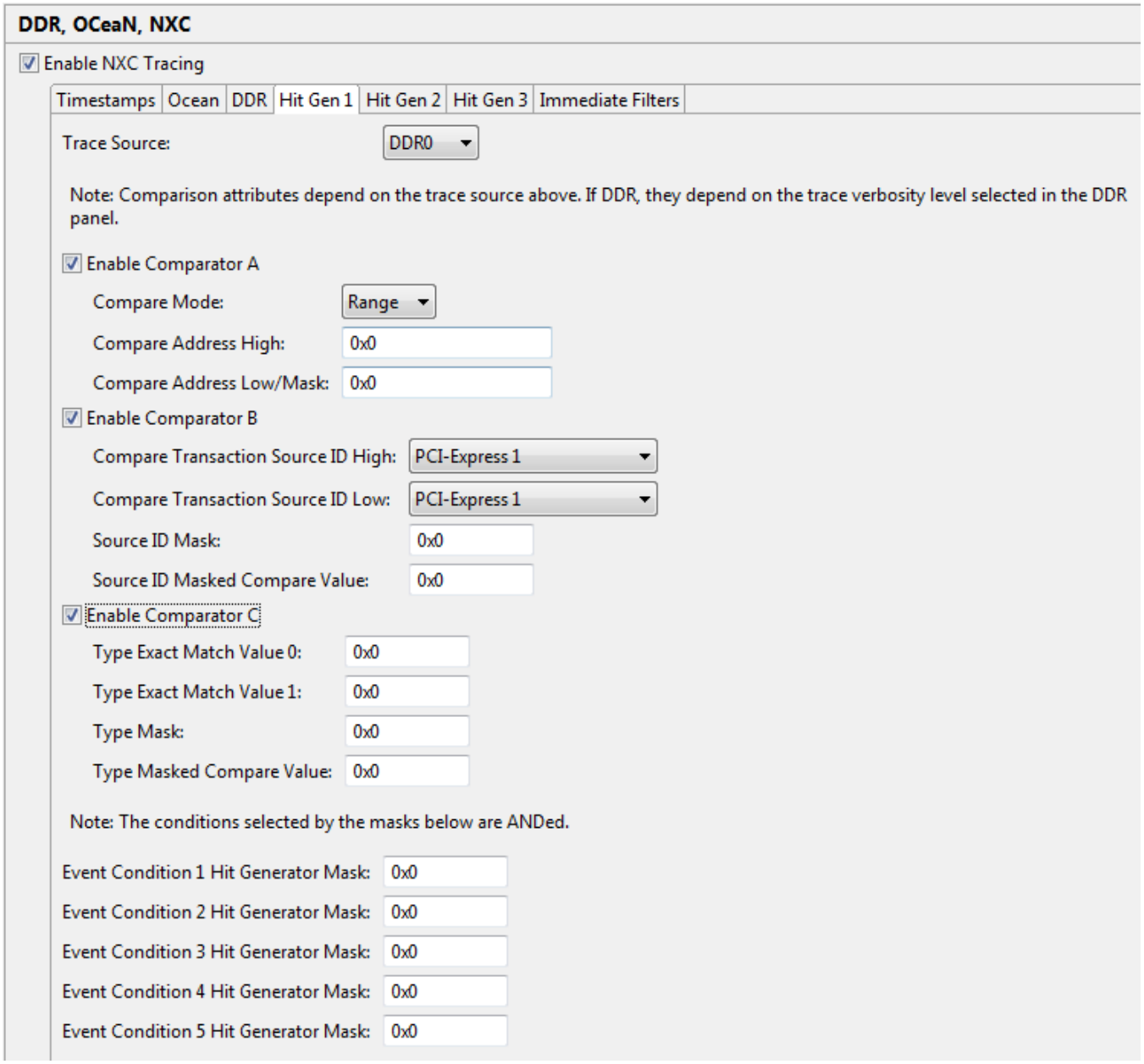


Figure 2-25. Hit Gen 1 tab

h. Use the **Trace Source** list box to select a trace source as the input for the hit generator.

i. Select all or any one of **Comparator A**, **Comparator B**, and **Comparator C** to compare the desired values. For more information on the options for all the comparators, see the following tables.

Table 2-16. Comparator A options

Option	Description
Compare Mode	Select the comparison logic:

Table continues on the next page...

Table 2-16. Comparator A options (continued)

Option	Description
	• Range - If selected, a match occurs when the transaction address falls within the values of **Compare Address High** text box and **Compare Address Low/Mask** text box. • Mask - If selected, a match occurs when the transaction address after being ANDed with the value in **Compare Address Low/Mask** is equal to the value in **Compare Address High** text box.
Compare Address High	Enter an address for comparison. The address is a range or a value depending on the comparison mode.
Compare Address Low/Mask	Enter an address for comparison. The address is a range or a mask depending on the comparison mode.

Table 2-17. Comparator B options

Option	Description
Compare Source ID High	Enter the highest value of the source ID range for comparison.
Compare Source ID Low	Enter the lowest value of the source ID range for comparison.
Source ID Mask	Enter the mask for source ID masked comparison.
Source ID Masked Compare Value	Enter the value for source ID masked comparison.

Table 2-18. Comparator C options

Option	Description
Type Exact Match Value 0	Enter a transaction type value for exact matching comparison.
Type Exact Match Value 1	Enter a transaction type value for exact matching comparison.
Type Mask	Enter a mask for transaction type masked comparison.
Type Masked Compare Value	Enter a value for transaction type masked comparison.

Table continues on the next page...

Table 2-18. Comparator C options (continued)

Option	Description
Event Condition <x> Hit Generator	Enter the mask values for the hit generators. These values select the hit signals that are ANDed to generate events. Each hit generator can trigger up to five events.

NOTE

The transaction attributes used for comparison depend on the trace source (DDR or Ocean) and on the trace verbosity level selected in the DDR panel as well, provided the trace source is one of the DDR memory controllers.

j. Click **Apply** to save the settings.
k. Click **Immediate Filters** tab.

The **Immediate Filters** configuration options appear. Use the options to turn on filtering of trace messages for each NXC trace source. Each debug client that sends trace through the NXC can have its trace data filtered. The immediate filters, when turned on for a client, require that a trace message meets the filter requirements before the message is passed on to the collector.

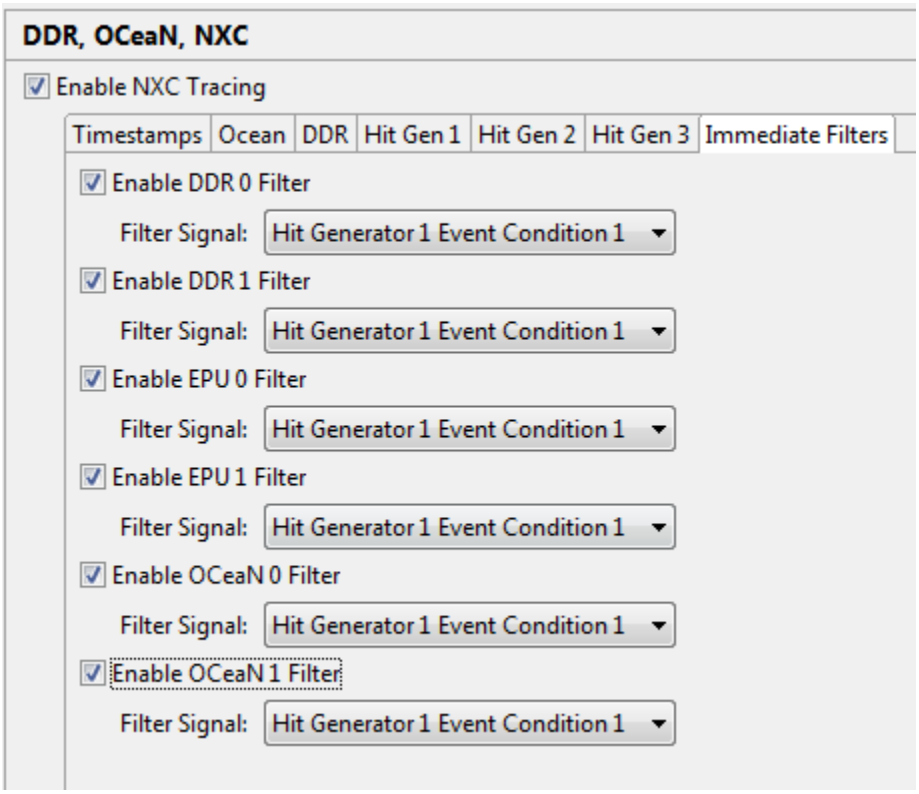


Figure 2-26. Immediate Filters tab

l. To turn on filters for a client, select from the available filters.

   The trace messages that meet the filtering criteria are passed to the collector.

m. Use the respective **Filter Signal** list box to specify the event condition to be met in order to pass a trace message.
n. Click **Apply** to save the settings.

13. Click **OK**.

    The **Trace Configurations** dialog closes and the **Debug Configurations** dialog appears.

2.1.2.2 Configuring trace for e6500 core

You can configure basic, intermediate, and advanced level trace parameters for the e6500 core to collect trace on this core.

The **Trace and Profile** tab used for configuring trace and profile parameters for the e6500 core is shown below. The e6500 core is supported by B4420, B4460, B4860, G4860, T1040, T1042, T2080, T4160, and T4240 targets.

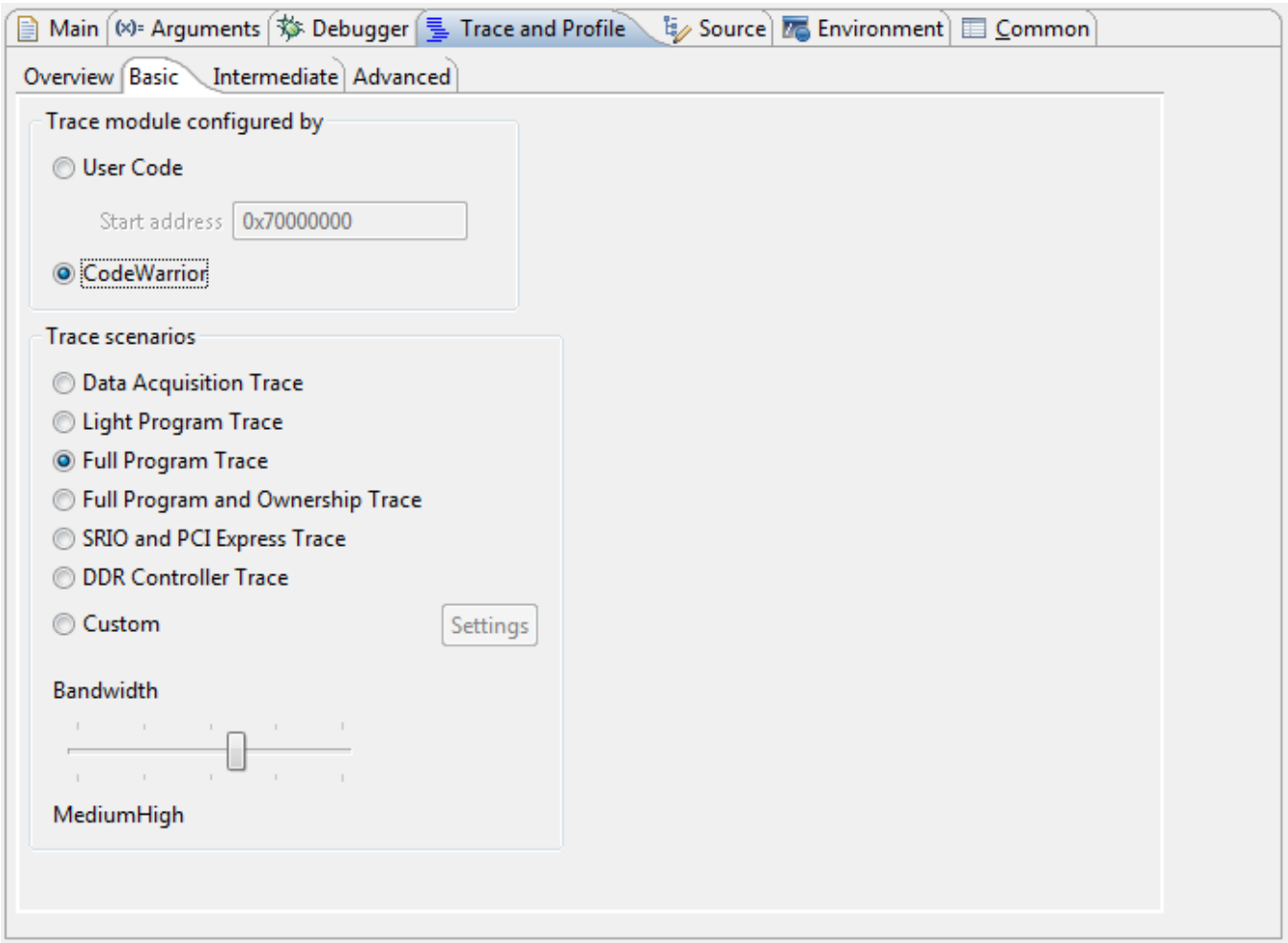


Figure 2-27. Trace and Profile tab for e6500 core

The table below describes the options of the **Trace and Profile** tab for the e6500 core.

Table 2-19. Trace and Profile tab options for e6500 core with description

Option	Sub-option	Description
Overview Tab		Displays the architecture of trace showing the graphical integration of trace and profile configuration of StarCore SC3900 and PA e6500 cores depending on the target (B4420, B4460, B4860, G4860, T1040, T1042, T2080, T4160, and T4240) selected. **NOTE:** B-series targets have e6500 and SC3900 cores and T-series targets have only e6500 core.
Basic Tab		
Trace module configured by		
	User Code	Lets you upload trace into CodeWarrior when trace session is stared from user code or tools other than CodeWarrior.

Table continues on the next page...

Table 2-19. Trace and Profile tab options for e6500 core with description (continued)

Option	Sub-option	Description
		That is, the application where you make your own configuration of trace registers in the trace hardware explicitly and CodeWarrior is not involved. The **Start address** is used only when DDR is used as trace buffer due to a hardware limitation which does not allow to read the start address from registers. The characteristics of this feature are: • Selecting the **User Code** option disables all other trace options and lets you automatically switch to manual trace collection. • You need to use the **Start Trace Collection** button in the **Debug** view to start trace collection in the User Code mode. • The User Code feature works in conjunction with the the **Upload Trace** button. The **Upload Trace** button becomes available in the **Debug** view after debugging the application with the **User Code** option selected. This button helps you get the collected trace in the respective viewers. For details on **Upload Trace** button, see the **Upload Trace** option described in Table 2-20. • The User Code feature is useful on Attach debug configurations when the program is already running on the target, and you want to attach to the program without changing its settings. • The Attach debug mode is used in conjunction with the User Code feature. When you select the Attach configuration for debugging, the trace mode settings automatically switch to User Code. Therefore, you can collect trace in the Attach debug mode using the **Upload Trace** button only. To know how to collect trace using the User Code feature on Attach configuration, see Collecting trace on Attach configuration. The process of collecting trace in User Code mode in Attach or Download configuration is same.
	CodeWarrior	Lets you upload trace when trace session is stared from CodeWarrior.

Table continues on the next page...

Table 2-19. Trace and Profile tab options for e6500 core with description (continued)

Option	Sub-option	Description
Trace Scenarios		Allows you to select predefined trace scenarios.
	Data Acquisition Trace	Provides a convenient and flexible mechanism for the debugger to observe the architectural state of the machine through software instrumentation in either Internal Debug Mode (IDM) or External Debug Mode (EDM). **NOTE:** IDM provides access to debug capabilities from resident software running on the device through memory-mapped registers. EDM provides full control and visibility from an off-chip external debugger with or without the need of a resident software debug agent.
	Light Program Trace	Traces only subroutine/interrupt call/return instructions.
	Full Program Trace	Traces each change of flow instruction.
	Full Program and Ownership Trace	Combines full program trace and ownership trace. 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.
	SRIO and PCI Express Trace	Monitors transactions of Serial RapidIO (SRIO) and PCI Express interfaces.
	DDR Controller Trace	Lets you collect Double Data Rate Controller (DDRC) trace.
	Custom	Allows you to customize predefined trace scenarios. Select this option to enable the **Settings** button. Click the **Settings** button to display the **Customize Trace Scenario** dialog.
Bandwidth		Indicates how many messages are routed in a trace stream depending on the selected trace scenario to collect trace. Each default trace scenario has a specific bandwidth. You can configure the predefined trace scenarios using the slider.
Intermediate Tab		
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.
Location	NPC Buffer	Saves trace data in NPC internal buffer.

Table continues on the next page...

Table 2-19. Trace and Profile tab options for e6500 core with description (continued)

Option	Sub-option	Description
	Gigabit TAP + Trace	Saves trace data collected on the Gigabit TAP probe in the Aurora buffer.
	Probe buffer size (bytes)	Specifies the memory size of the Aurora probe for trace collection. The Aurora probe has a dedicated memory of 4 GB for trace collection.
	Magenta	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 trace data is saved.
	Buffer size	Specifies the DDR memory size of the region used as trace buffer.
Trace Control Settings		Allows you to specify trace configuration settings.
	Automatically (When debug session starts)	Starts the trace collection process automatically when the debug session launches.
	Manually (Using debug toolbar trace buttons)	Lets you control trace collection manually. The trace collection process will not start until it is explicitly turned on. Use the Start/Stop trace collection buttons available in the action bar of the **Debug** view to control trace collection. You can collect trace within a specific section of the program (by using debugger breakpoints).
SRIO and PCI Express Trace Configuration		
	Port #0/Port #1	Emits trace messages about PCI and SRIO transactions.
	Source	Is the module that initiates the PCI and SRIO transaction.
	Save Bandwidth	Saves bandwidth by filtering out data messages. The trace will continue to monitor transactions (request/response messages).
DDR Controller Trace Configuration		
	Enable DDRC0/DDRC1/DDRC2 Trace	Allows you to enable three DDR controllers that can be traced separately.
	Mode	Lets you select one of the following trace modes: • Verbose - provides maximum information in trace as allowed by the protocol. • Terse1/2/3 - provides less trace information as compared to Verbose mode. This mode saves bandwidth as less data means

Table 2-19. Trace and Profile tab options for e6500 core with description

Option	Sub-option	Description
		smaller packets which results in less load on the bus. Each Terse mode provides less trace information than another. The trace information provides in Terse1 mode > Terse2 mode > Terse3 mode. Depending on how much trace can be generated, you can choose to get more or less information along with the granularity.

NOTE

Full Program Trace and **Light Program Trace** are mutually exclusive. Data trace is used in its inclusive range mode, and it is currently tracing store operations only. Therefore, the **Data Trace** option lets you configure the **Start Address** and **Stop Address** options which can be given as hexadecimal values.

NOTE

There is a hardware limitation with B4 rev 1 due to which you encounter an issue while viewing collected trace data in case the **Light Program Trace** option is selected. In the **Trace** viewer, you see decoding errors or some invalid portions of trace such as "Branch and link instruction occurrence" and "Branch from ? to ?" messages. This happens because the on-the-fly decoder attempts to synchronize with a sync message (TCODE 29) that was sent without a Nexus Link Stack invalidation. Currently, it is not possible to ensure proper synchronization, because the only useful message, after which the Nexus Link Stack is invalid, is TCODE 9 and happens to be sent only at the beginning, which is not enough for on-the-fly decoding. The workaround for this problem is opening results that fully decode the trace data, for example, **Timeline**, **Critical Code**, **Call Tree**. After this, clicking on the **Trace** hyperlink will result in accessing the decoded trace data that does not need to rely on sync messages in random positions. The issue is fixed in B4 rev 2.

This topic contains the following sub-topics:

- Customizing trace scenarios
- Collecting trace on Attach configuration

2.1.2.2.1 Customizing trace scenarios

The **Customize Trace Scenario** dialog allows you to customize trace scenarios to collect trace on the e6500 core

by combining different trace scenarios.

The **Customize Trace Scenario** dialog appears on clicking the **Settings** button in the **Basic** tab of the **Trace and Profile** page.

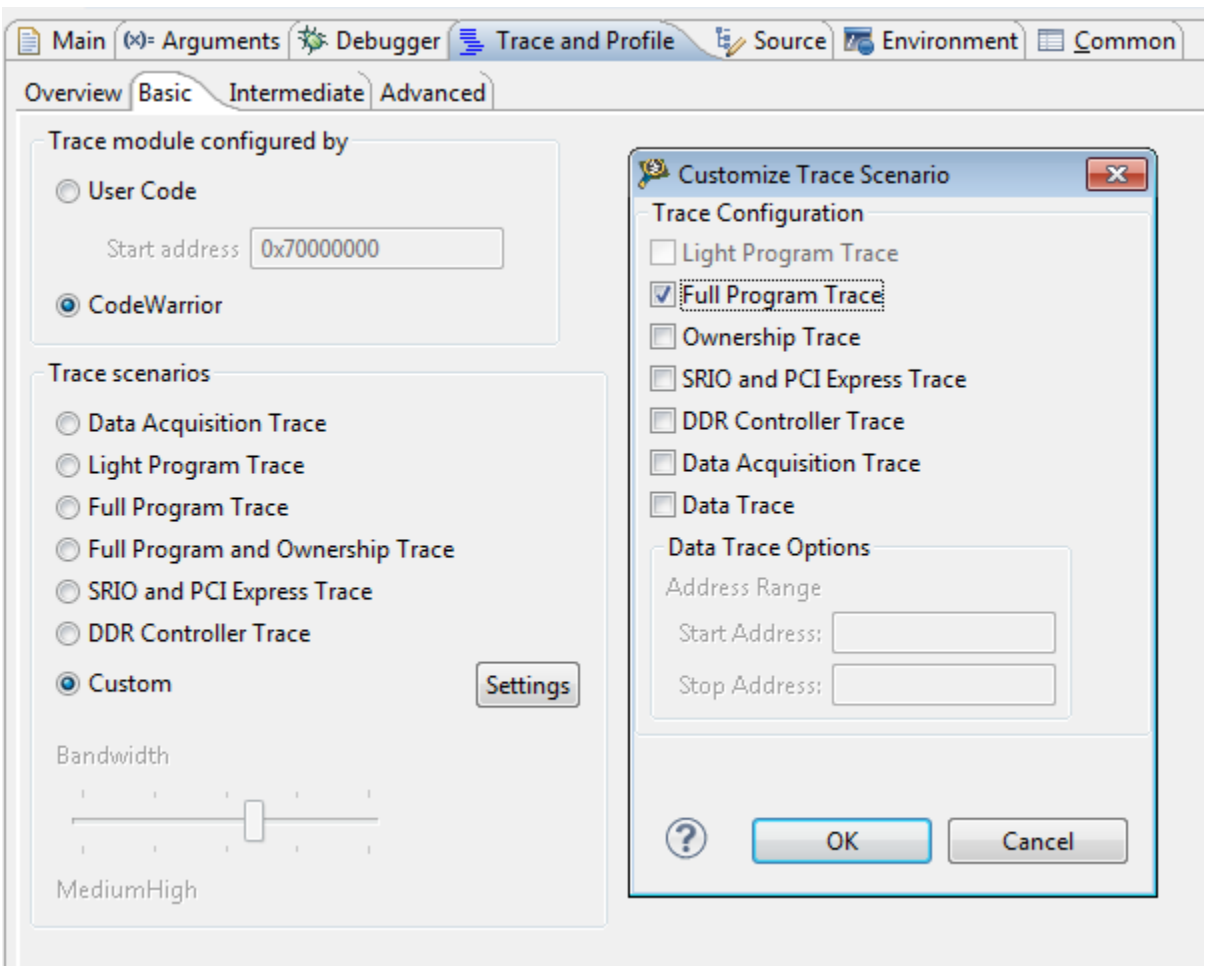


Figure 2-28. Customize Trace Scenario dialog

The **Customize Trace Scenario** dialog displays the following options:

- Full Program Trace - Lets you trace each change of flow instruction.
- Ownership Trace - Lets you trace information on process ID by tracking the active operating system.
- SRIO and PCI Express Trace - Lets you trace transactions of SRIO and PCI Express interfaces, which are used to transfer large amounts of data between processors.

- DDR Controller Trace - Lets you collect Double Data Rate Controller (DDRC) trace. The **DDR Controller Trace** option traces read and write transactions.
- Data Acquisition Trace - Lets you analyse the architectural state of the machine through software instrumentation in either IDM or EDM mode.
- Data Trace - Lets you collect trace of selected memory reads and writes. Data trace is collected on the memory addresses. You can specify the **Start Address** and **Stop Address** of the memory between which you want the data trace to be collected. The **Data Trace** option traces only write transactions.

2.1.2.2.2 Collecting trace on Attach configuration

The Attach configuration is supported on the e6500 core (supported by B4420, B4460, B4860, G4860, T1040, T1042, T2080, T4160, and T4240 targets) for trace collection. The Attach launch configuration is useful when you want to connect to a running target without resetting the target or downloading a different application on it. For more information on Attach launch configuration, see the *CodeWarrior Development Studio for Power Architecture Processors Targeting Manual (document CWPADBGUG)* in the folder: *<CWInstallDir>\PA\Help\PDF*.

To configure your application for trace collection (on the e6500 core) using the Attach launch configuration:

1. Create a project with the Attach launch configuration using the **CodeWarrior Bareboard Project Wizard**.
   a. From the IDE menu bar, choose **File > New > CodeWarrior Bareboard Project Wizard**.
   b. Enter a name for the new project in the **Project Name** text box.
   c. Click **Next** and select the required Bx or Tx processor in the **Processors** page.
   d. Click **Next** and select the **Attach** checkbox in the **Debug Target Settings** page. Choose the **Default** option to create an Attach configuration based on default parameters.

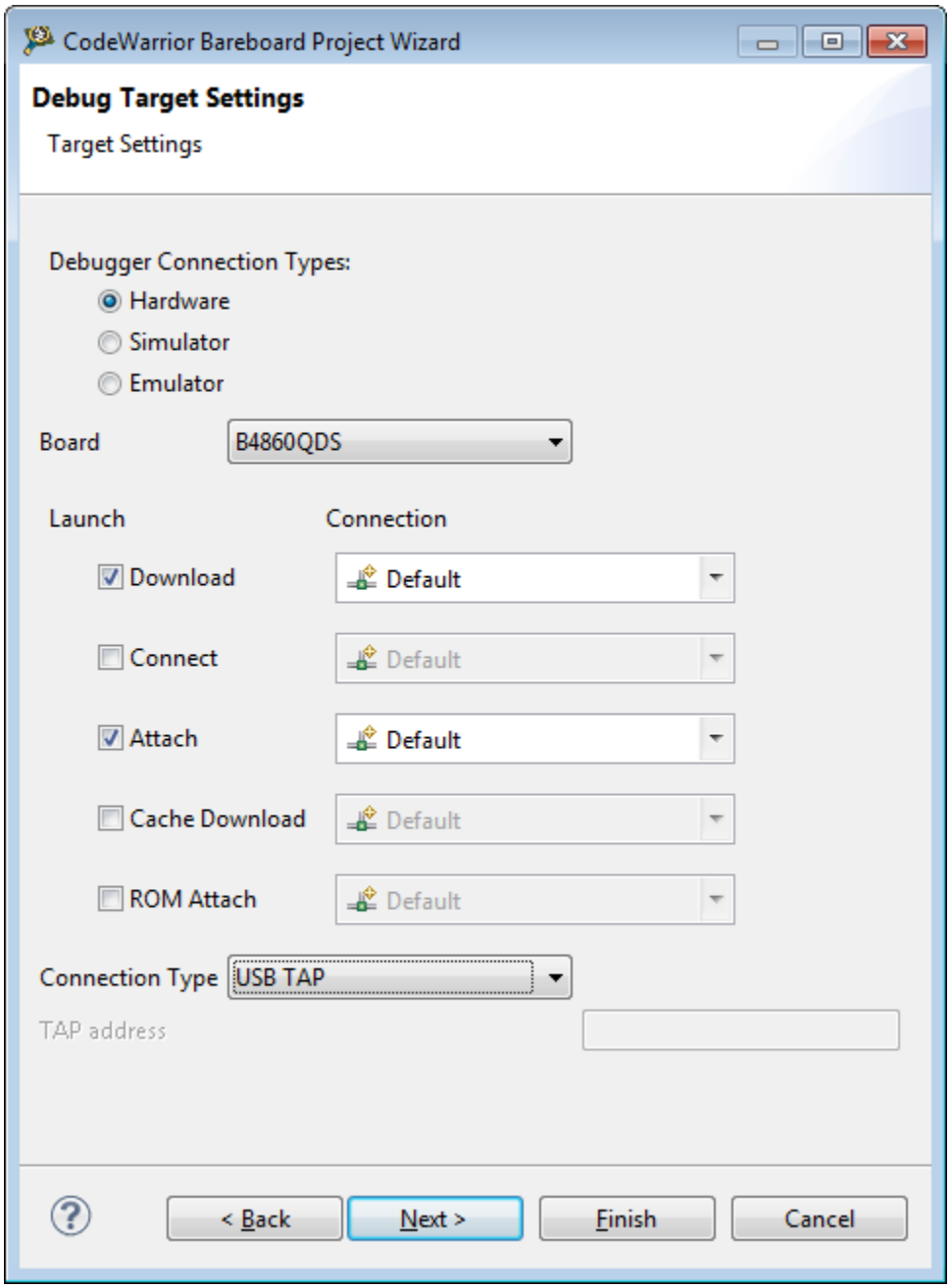


Figure 2-29. Creating Attach launch configuration in Debug Target Settings page

e. Select the required **Connection Type** and click **Next**.
f. Follow the remaining steps of the wizard and click **Finish**.

2. Build the project.
3. Choose **Run > Debug Configurations** to open the **Debug Configurations** dialog.
4. Expand **CodeWarrior** in the left pane of the dialog and select the Attach launch configuration corresponding to your project.

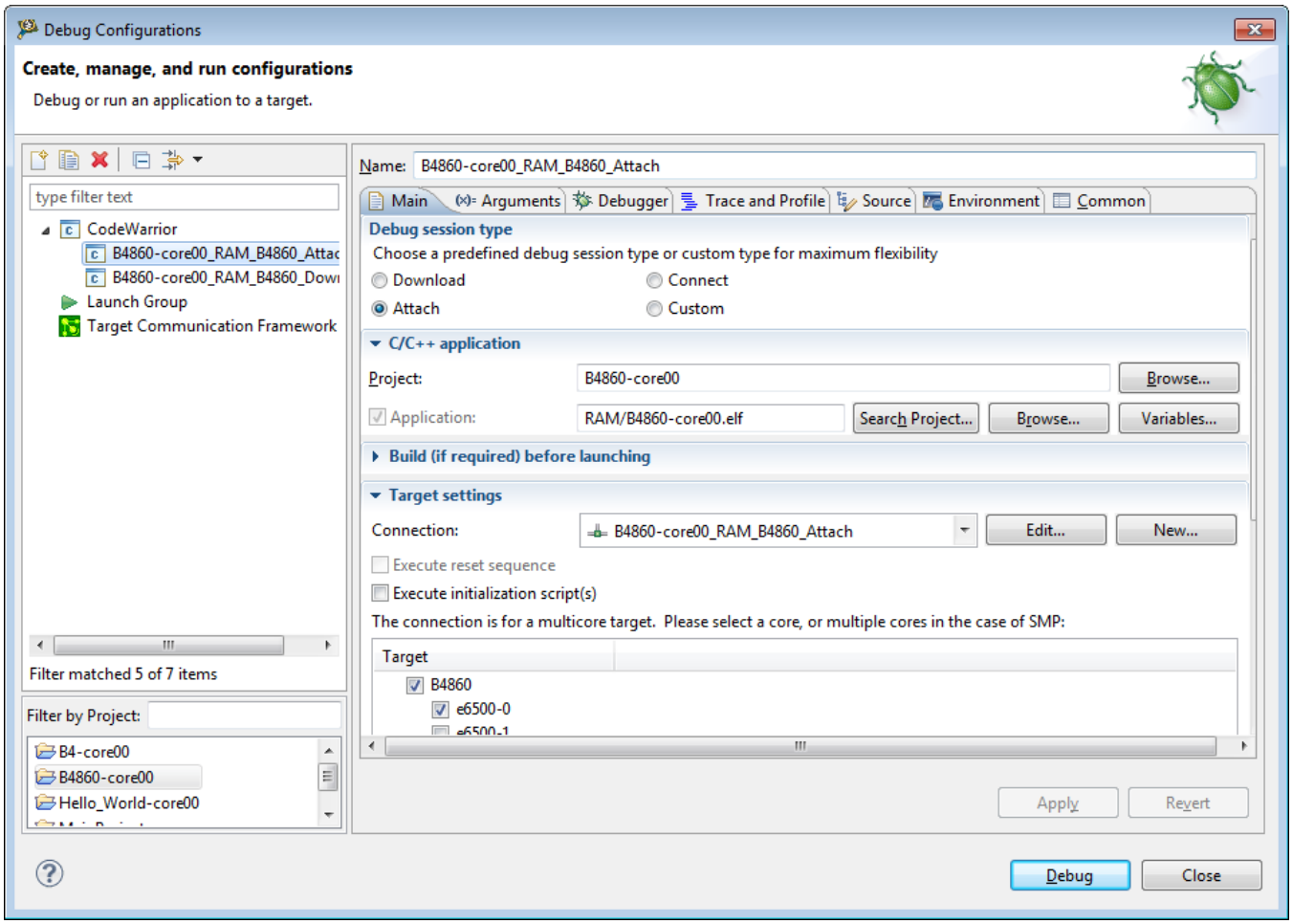


Figure 2-30. Selecting Attach launch configuration

NOTE

The Attach configuration assumes that code is already running on the board and therefore does not run a target initialization file. Therefore, when attach launches are used, ensure that the target is already running.

5. In the **Main** tab of the dialog, make sure that **Attach** is selected in the **Debug Session Type** group.
6. Open the **Trace and Profile** tab.

   The **User Code** mode is automatically selected in the **Basic** tab. All other trace options are disabled. In the **Intermediate** tab, the **Trace control settings** is selected as **Manually**.

7. Click **Debug**.
8. Click the **Start Trace Collection** button, in the **Debug** view toolbar, to turn on the tracing session.
9. Click **Resume** to start collecting trace data.

10. Click **Suspend** after sometime to stop trace collection.
11. Click the **UploadTrace** button to get the collected trace.
12. Open the **Trace** viewer to view the collected data.

NOTE

The Attach configuration for the e6500 core is used with **User Code** trace mode and **Upload Trace** button only. For details, see these options described in Table 2-19.

NOTE

The steps of collecting trace using the Attach configuration for P devices is same as e6500 core. The basic difference is that you need to select the **The application configures the target and enables trace collection** option in the **Trace and Profile** tab for P devices to switch to the Attach debug mode. For details, see Table 2-3.

2.2 Collecting data

Now that a project is created and a launch configuration and trace configuration defined, you can start a debug session and start collecting trace data.

You can perform trace collection tasks using the buttons available in the **Debug** view.

To collect trace data:

1. In the **Debug Configurations** dialog, click **Debug**.

   The IDE switches to the **Debug** perspective; the debugger downloads your program to the target development board and halts execution at the first statement of `main()`.

   In the editor view (center-left of perspective), the program counter icon on the marker bar points to the next statement to be executed.

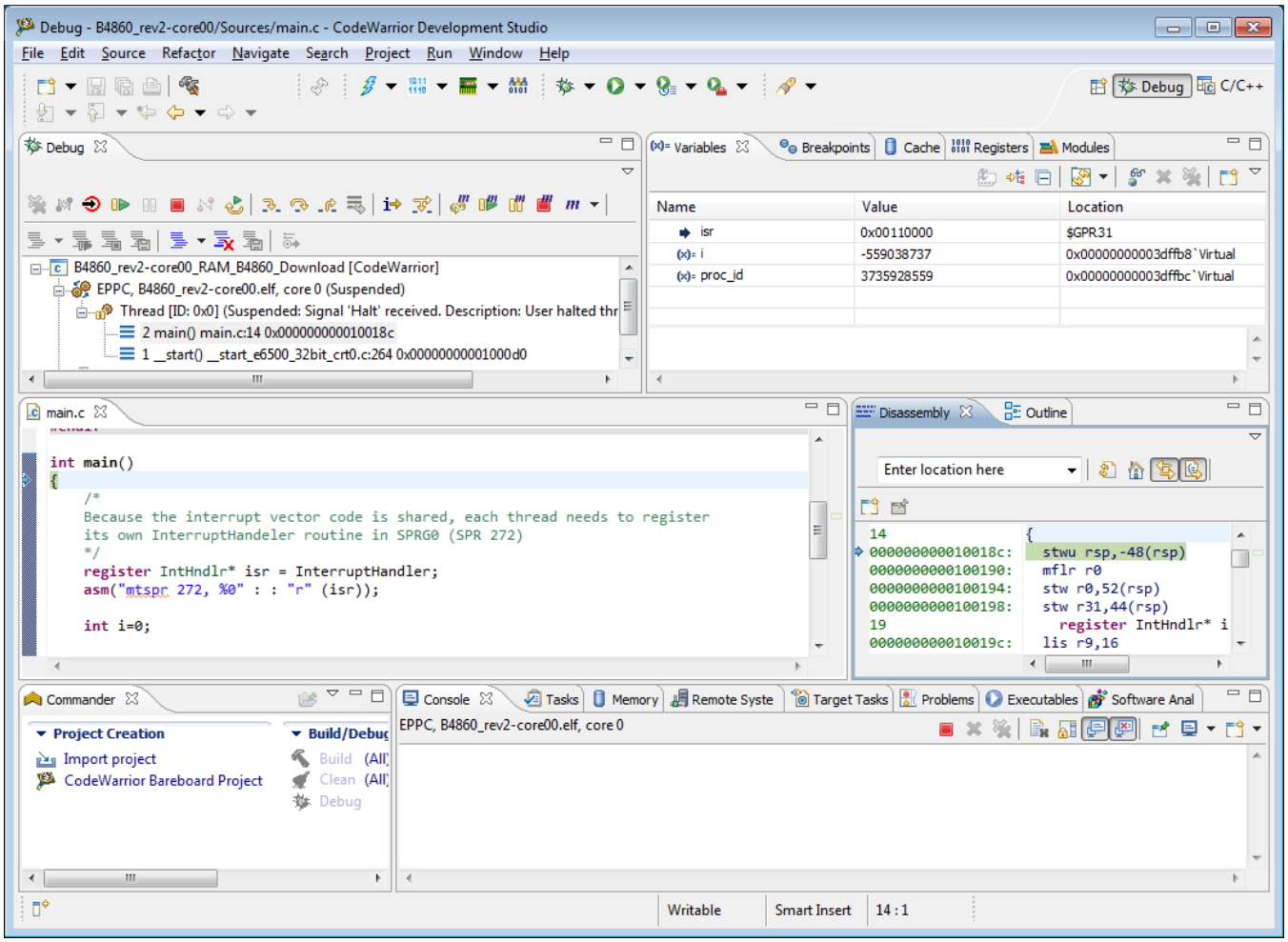


Figure 2-31. Debug perspective

2. In the **Debug** view, click **Resume** .

   The data collection starts. The debugger executes all statements, the program writes the strings in the **Console** view and then enters an infinite loop.

3. Let the application execute for several seconds and click **Suspend** . This step is required for B and T processors only. For P processors, trace starts collecting and appears automatically in the **Trace** viewer after clicking **Resume**.

   Once tracing starts, the progress bar in the **Progress** view (choose **Window > Show View > Other > General > Progress**) indicates the collection status and buffer contents based on the information reported by the probe or target. The progress bar disappears after trace collection completes.

NOTE

Some combinations of probe and target may not result in a useful percent full indication.

Collecting data

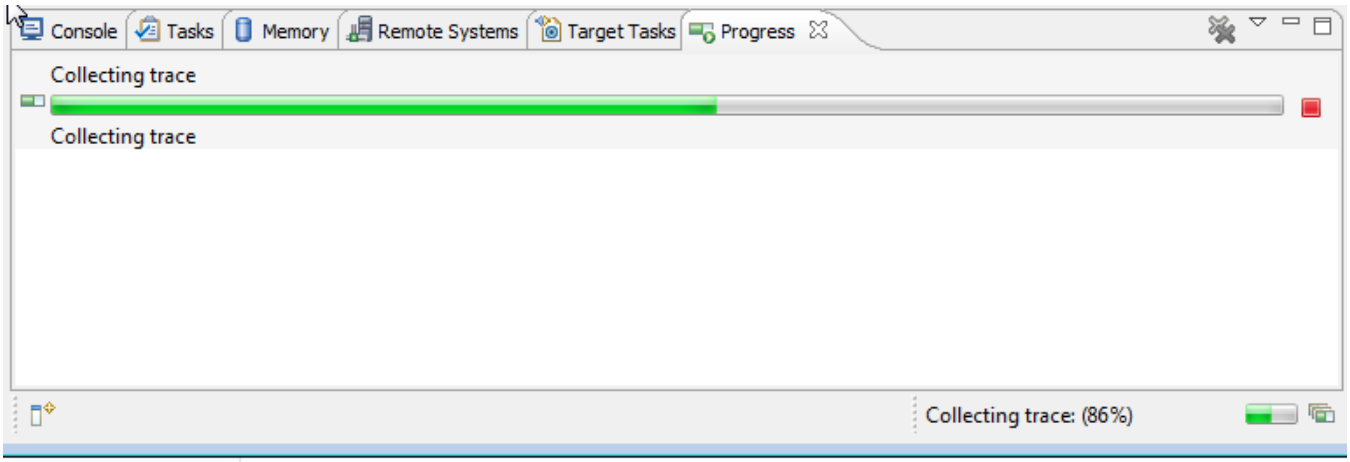


Figure 2-32. Trace collection status indication

The trace data collects at: <*UserWorkspace*>*Hello_World**AnalysisData*, where *Hello_World* is the name of the project folder you created and <*UserWorkspace*> is your working directory.

In previous versions of the Power Architecture 10.x product, tracing was configured only at the launch time, collected only when the program first resumed, and saved only when the debug session was terminated. Tracing could only be stopped by canceling the collection task, or by suspending the debug session.

The new GUI buttons and menus make these actions available to the user at any appropriate time for any selected debug target.

The trace controls are available on the action bar in the Debug perspective as shown in the following figure separately for P series as well as Bx/Tx devices. These trace commands apply to the selected debug targets in the **Debug** view. They are available only when the selected target is in an appropriate state.

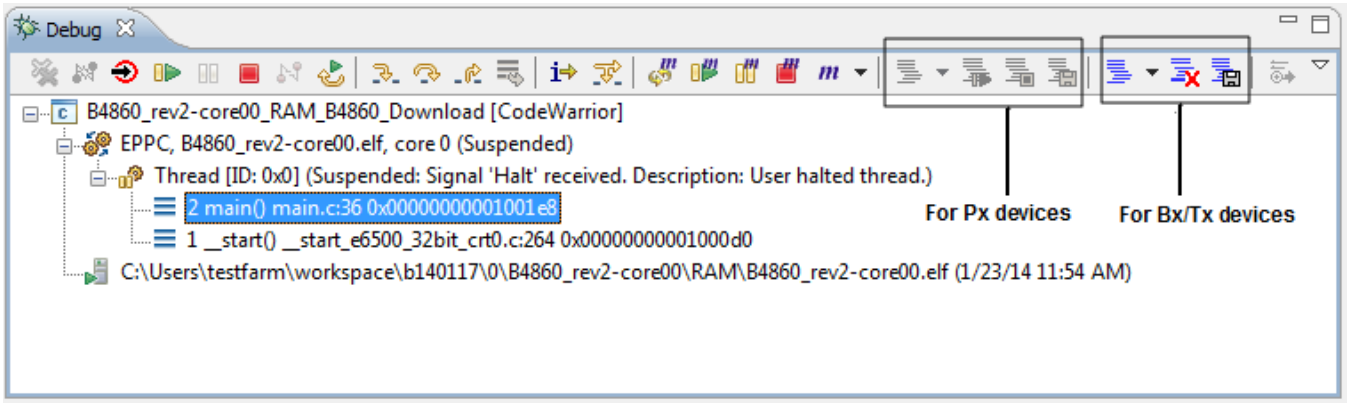


Figure 2-33. Debug view toolbar

Table 2-20. Trace controls

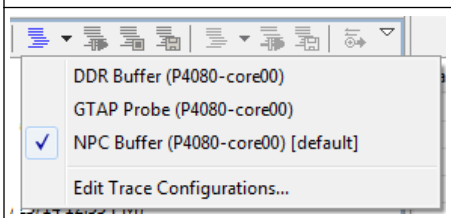
Controls	Name	Description
	Start Trace Session/Configure Trace	The **Start Trace Session** button allows you to launch a trace session for P series processors. Clicking on the left side of the button launches a new trace session using the default trace configuration. The button also has a pop-up menu containing a list of the available trace configurations. For Bx/Tx devices, the **Configure Trace** button is used. Clicking the button opens the **Configure Trace** dialog to let you change the current configuration. The pop-up menu is not available.
	Start Trace Session	Provides immediate access to the trace configurations associated with the current launch configuration as well as access to the **Trace Configurations** dialog. The available profiles can only be applied while the target is connected but not collecting trace currently. The dialog is available at any time. An active session is marked with a checkbox, while the default trace configuration is marked with *[default]*. Clicking on one of the configurations launches a new trace session, replacing an existing one if it exists. **NOTE:** When applying a new trace configuration, all data from the previous configuration (including all collected trace) will be lost.
	Start Trace Collection	Turns on the tracing session for collecting trace data. This button is available when no trace collection or trace upload is ongoing.

Table continues on the next page...

Table 2-20. Trace controls (continued)

Controls	Name	Description
	Stop Trace Collection (for P series processors)	Stops an ongoing trace collection. This button does not cancel the trace, and if you have selected the automatic upload and display of trace configured in your launch configuration, then whatever trace has been collected will be uploaded and displayed. This button is available only when trace is being collected. However, the button appears disabled in case you are collecting program trace where buffer is small and fills instantaneously. The button is also disabled if you have selected **The application configures the target and enables trace collection** option in the **Trace and Profile** tab.
	Stop Trace Collection (for Bx/Tx devices)	Stops an ongoing trace collection. However, you can view whatever data has been collected by clicking the **Upload Trace** button.
	Upload Trace	Uploads/saves the collected trace data to a file on the host system. It allows you to manually upload the trace data from the target. The Upload Trace feature provides a better control over the trace collection process as it allows you to upload whatever trace data is in the buffer at any time when the target is suspended. After uploading, the button becomes disabled. For P series processors, this button is available if **The application configures the target and enables trace collection** option is selected, and it depends on when you want to upload trace data relative to your application's execution. This feature is useful in Attach configurations to collect trace. For example, on an Attach configuration, when something goes wrong on a target running an application or your application throws an exception, you can start the trace session and click the **Upload Trace** button to get the trace data collected until the exception. For Bx/Tx devices, the Attach configuration uses the **User Code** feature to collect trace. For details on the **User Code** feature, see Table 2-19.

NOTE

The button behavior is affected by the settings in the **Trace and Profile** tab of the launch configuration associated with the currently selected target. As a result, some buttons may not ever be available for that target.

2.3 Viewing data

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

This topic describes the following sub-topics.

- Software Analysis view
- Finding and filtering trace data
- Exporting trace data
- Configuring time unit
- Excluding symbols from data results

2.3.1 Software Analysis view

The **Software Analysis** view provides access to various types of data results collected on an application.

The **Software Analysis** view appears automatically if you have chosen to display the trace results automatically. You can manually open the **Software Analysis** view by choosing **Window > Show View > Software Analysis** in the CodeWarrior IDE menu bar.

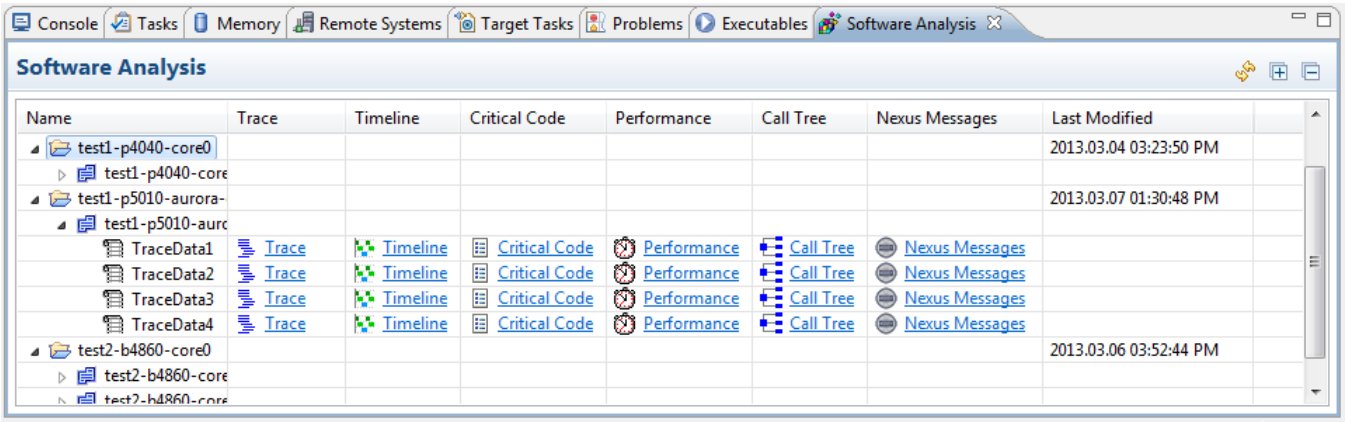


Figure 2-34. Software Analysis view

The **Software Analysis** view toolbar provides the following actions.

Table 2-21. Software Analysis - Toolbar options

Option	Description
	Refreshes the displayed data.
	Expands all the nodes.
	Collapses all the nodes.
	Lets you import an external format of trace into your project.

The **Software Analysis** view shortcut menu, which appears on right-click, provides the following actions.

Table 2-22. Software Analysis - Shortcut menu commands

Option	Description
Refresh	Refreshes the displayed data.
Expand All	Expands all the nodes.
Collapse All	Collapses all the nodes.
Import Data From Txt	Lets you import an external format of trace into your project.
Copy Cell	Copies the currently selected cell. The name of the data source is copied to the clipboard.
Copy Line	Copies the complete line of the data source.
Save Results	Saves the trace results.
Delete Results	Deletes all the saved results.

The **Software Analysis** view provides hyperlinks to the following results:

- Trace

- Timeline
- Critical Code
- Performance data
- Call Tree
- Nexus messages
- Configure

2.3.1.1 Trace

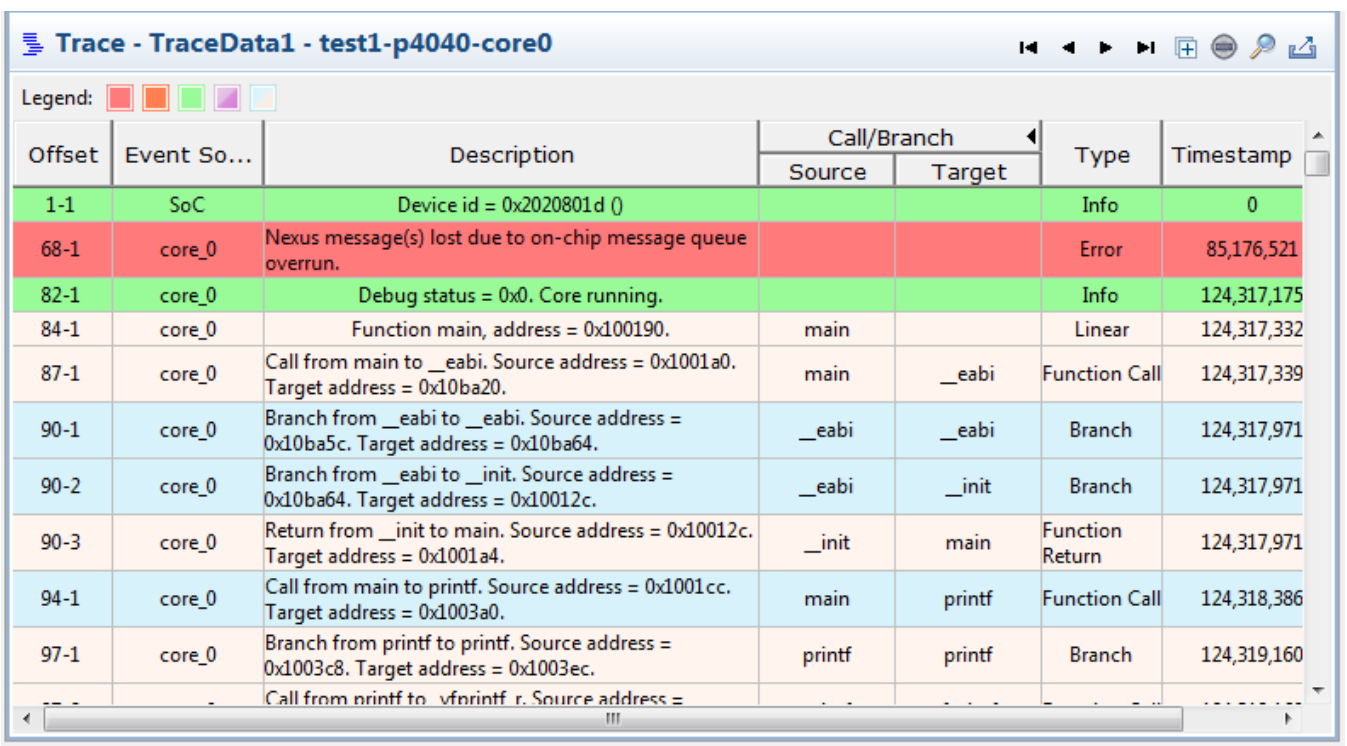
The **Trace** viewer displays the decoded trace event data generated during the data collection phase.

To view trace data:

1. In the **Software Analysis** view, expand the project name.

   All trace collections performed for the project are displayed.

2. Click the **Trace** hyperlink, and view the trace data in the **Trace** viewer.



Offset	Event So...	Description	Call/Branch Source	Call/Branch Target	Type	Timestamp
1-1	SoC	Device id = 0x2020801d ()			Info	0
68-1	core_0	Nexus message(s) lost due to on-chip message queue overrun.			Error	85,176,521
82-1	core_0	Debug status = 0x0. Core running.			Info	124,317,175
84-1	core_0	Function main, address = 0x100190.	main		Linear	124,317,332
87-1	core_0	Call from main to __eabi. Source address = 0x1001a0. Target address = 0x10ba20.	main	__eabi	Function Call	124,317,339
90-1	core_0	Branch from __eabi to __eabi. Source address = 0x10ba5c. Target address = 0x10ba64.	__eabi	__eabi	Branch	124,317,971
90-2	core_0	Branch from __eabi to __init. Source address = 0x10ba64. Target address = 0x10012c.	__eabi	__init	Branch	124,317,971
90-3	core_0	Return from __init to main. Source address = 0x10012c. Target address = 0x1001a4.	__init	main	Function Return	124,317,971
94-1	core_0	Call from main to printf. Source address = 0x1001cc. Target address = 0x1003a0.	main	printf	Function Call	124,318,386
97-1	core_0	Branch from printf to printf. Source address = 0x1003c8. Target address = 0x1003ec.	printf	printf	Branch	124,319,160
		Call from printf to _vfprintf_r. Source address =				

Figure 2-35. Trace viewer

The data in **Trace** viewer is organized in a way to ease the evaluation of tracing information and navigation through the events in the sequence they were logged. This data can be very complex, and the size of the decoded data can be very large, up to approximately 40 GB. The **Trace** viewer is constrained by the size of the decoded data. Currently, the SAE can iterate over a maximum of 2^32 items of raw Nexus trace. Each item of undecoded trace can be associated with zero, one, or multiple decoded trace events.

The **Trace** viewer displays the source code and corresponding linear instructions, or non-executed branch instructions, which have occurred since the last program trace event of the same trace source. The source of the trace event is displayed in the **Source** sub-column, and the destination of the change of flow instruction is displayed in the **Target** sub-column of **Call/Branch** column.

The **Trace** viewer displays the collected trace data in a tabular form. You can move the columns to the left or right of another column by dragging and dropping.

The following table describes the **Trace** viewer fields.

Table 2-23. Trace viewer fields

Field	Description
Offset	The first number in the offset specifies the Nexus Message from which the trace event has been decoded. The second number specifies the number of the decoded event, since each Nexus Message can generate 0 to n decoded events. If Nexus Messages are shown, they appear with a second offset value of 0. Any decoded events always appear beginning with a second offset value of 1.
Event Source	Specifies the trace source that produced the event, such as the Nexus client and trace probe device.
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	Specifies the type of event that has occurred, such as Info, Branch, Error, Function Return and Function Call.
Timestamp	Specifies the timestamp value that is expressed as clock ticks. Depending on the configuration of the trace source, this can be relative to the event's trace source or relative to all events logged from all sources.

The **Trace** viewer provides a color convention in the form of **Legend** to highlight trace events and source changes. The colors available are:

- Red - Highlights error events reported in the trace data.

- Orange - Highlights interrupt jumps reported in both program and profiling trace data.
- Green - Highlights info events reported in the trace data.
- Purple (Light and Dark) - Highlights 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.

You can perform the following actions from the **Trace** viewer:

- Click the navigation buttons available to move between the trace data.
  - Click  to go to the beginning of trace data
  - Click  to go to the previous trace entry
  - Click  to go to the next trace entry
  - Click  to go to the end of trace data
- Click the **Expand All** button  to display the source as well as assembly code of the instructions in the trace viewer.
- Click the **Collapse All** button  to display the assembly code only.
- Click the **Show Nexus Messages** button  to display the Nexus messages in the **Trace** viewer.
- Click  to search for particular events or categories of events within the current trace data set and filter the trace data based on certain criteria.
- Click  to export trace data to a CSV (Comma Separated Values) file.

2.3.1.1.1 Customizing trace viewer

You can customize or modify the appearance or display of the trace results on the **Trace** viewer.

Right-click on a column to open a shortcut menu that lets you perform the following actions:

- Hide column - Allows you to hide column(s). To hide a column on the **Trace** viewer, select that column, right-click and choose **Hide column** from the shortcut menu. To display it back, choose **Show all columns** from the shortcut menu. To hide multiple columns, select the columns with Control key pressed, right-click and choose **Hide column** from the shortcut menu.
- Group selected columns- Allows you to group multiple columns into one. Grouping columns allows you to perform a set of operations to a number of columns grouped

together in one go. To group columns, select the columns with Control key pressed. Right-click and choose the **Group selected columns** option. The **Create Column Group** dialog appears.

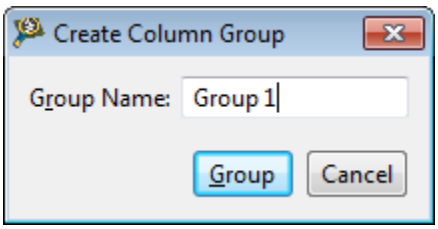
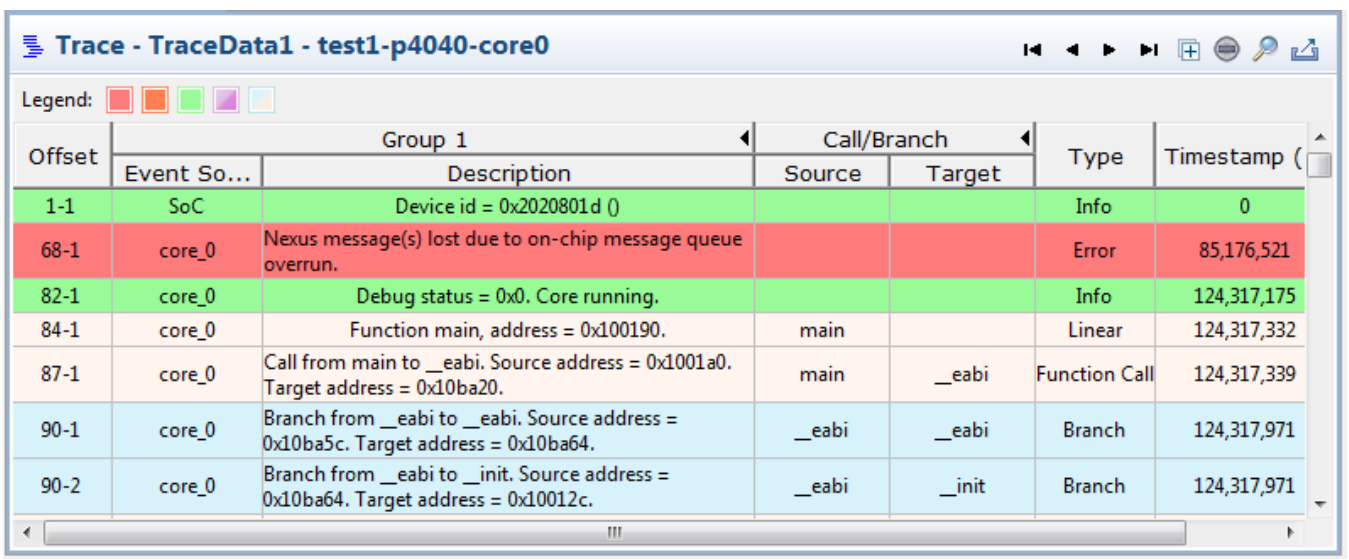


Figure 2-36. Create Column Group

Enter a name for the group in the **Group Name** text box and click **Group**. The following figure shows the **Event Source** and **Description** columns grouped together.



Offset	Group 1		Call/Branch		Type	Timestamp (
	Event So...	Description	Source	Target		
1-1	SoC	Device id = 0x2020801d ()			Info	0
68-1	core_0	Nexus message(s) lost due to on-chip message queue overrun.			Error	85,176,521
82-1	core_0	Debug status = 0x0. Core running.			Info	124,317,175
84-1	core_0	Function main, address = 0x100190.	main		Linear	124,317,332
87-1	core_0	Call from main to __eabi. Source address = 0x1001a0. Target address = 0x10ba20.	main	__eabi	Function Call	124,317,339
90-1	core_0	Branch from __eabi to __eabi. Source address = 0x10ba5c. Target address = 0x10ba64.	__eabi	__eabi	Branch	124,317,971
90-2	core_0	Branch from __eabi to __init. Source address = 0x10ba64. Target address = 0x10012c.	__eabi	__init	Branch	124,317,971

Figure 2-37. Columns grouping

- Ungroup selected columns - To ungroup the columns, select the grouped columns, right-click and choose **Ungroup selected columns** from the shortcut menu.
- Choose columns - You can set the columns that you want to display in the **Trace** viewer. Right-click on any column and choose the **Choose columns** option. The **Column Chooser** dialog appears.

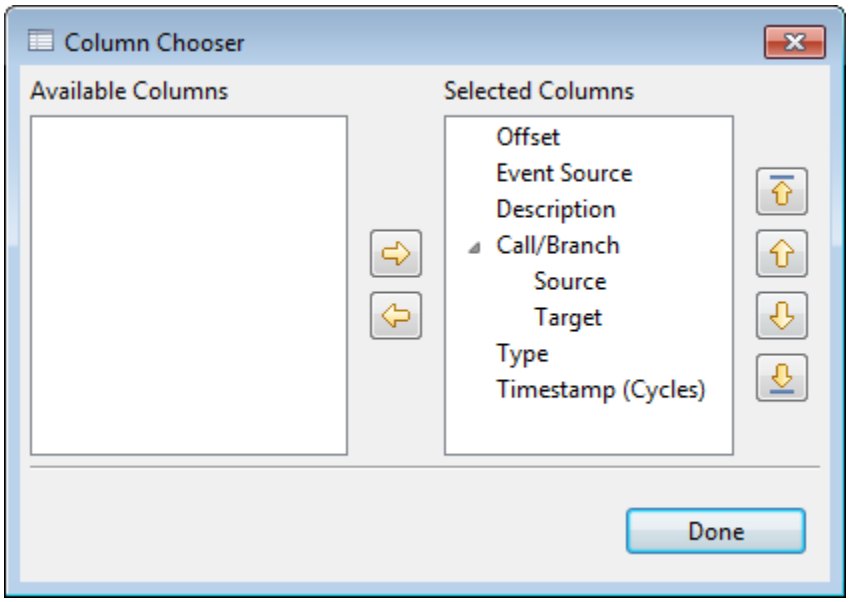


Figure 2-38. Column Chooser

Select the column in the **Selected Columns** section, and use the **up** and **down** arrow buttons to position the column up or down according to your choice. Use the **left** and **right** arrow buttons to move the columns to **Available Columns** from **Selected Columns**. The columns moved to the **Available Columns** section are not shown in the **Trace** viewer. Click **Done** to save the settings.

- Rename column - Select the column, right-click and choose the **Rename** column option. The **Rename Column** dialog appears. Enter a new name for the column in the **Rename** text box and click **OK**.

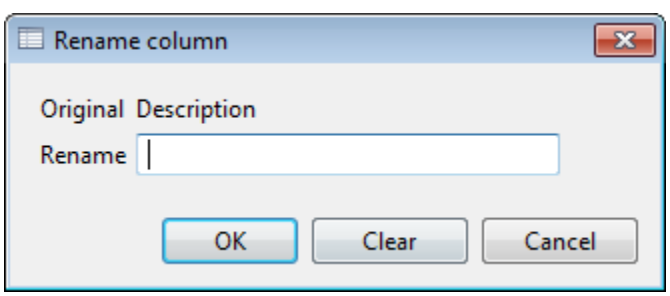


Figure 2-39. Rename column

2.3.1.2 Timeline

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.

To view timeline data:

1. In the **Software Analysis** view, expand the project name.

   The data source is listed under the project name.

2. Click the **Timeline** hyperlink.

   The **Timeline** viewer appears.

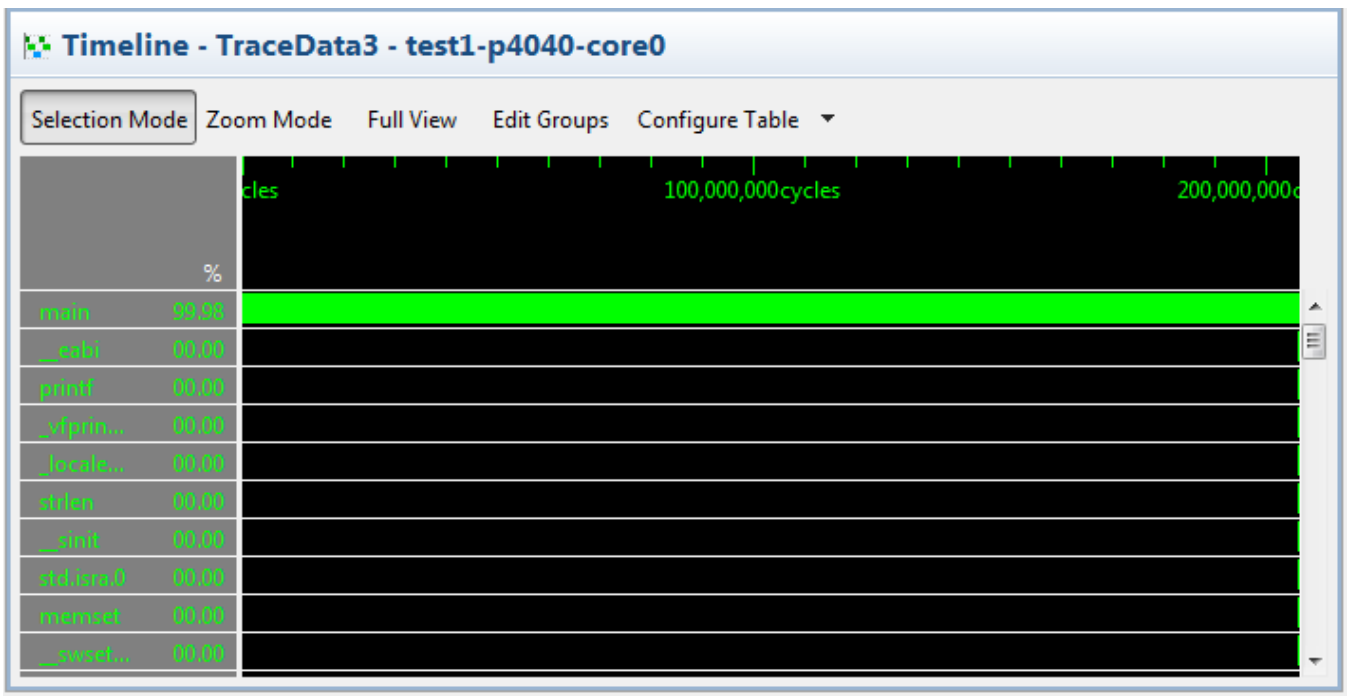


Figure 2-40. Timeline viewer

The **Timeline** viewer shows a timeline graph in which the functions appear on y-axis and the number of cycles appear on x-axis. The green-colored bars show the time and cycles that the function takes.

The **Timeline** viewer also displays the following buttons:

- Selection Mode
- Zoom Mode
- Full View
- Edit Groups
- Configure Time Unit

2.3.1.2.1 Selection Mode

The **Selection Mode** allows you to mark points in the function bars in the timeline graph to measure the difference of cycles between those points.

To mark a point in the bar:

1. Click **Selection Mode**.
2. Click on the bar where you want to mark the point.

   A yellow vertical line appears displaying the number of cycles at that point.

3. Right-click 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.

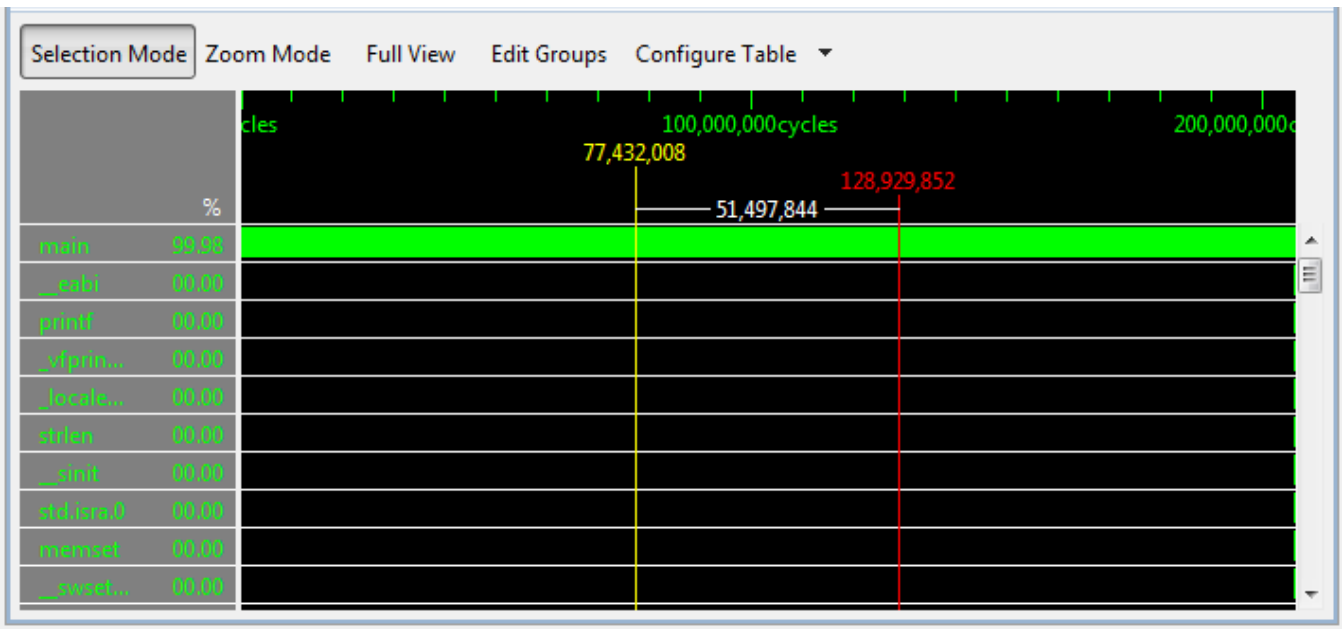


Figure 2-41. Selection mode to measure difference of cycles between functions

You might view a difference in the time cycles displayed in the **Timeline** and the **Critical Code** viewer. The difference is caused by the events in the functions (in your source code) that have no new timestamp. For timeline, any instruction that has no timestamp information is considered to take one CPU cycle.

2.3.1.2.2 Zoom Mode

The **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 in the timeline graph. You can also move the mouse wheel up and down to zoom-in and zoom-out.

2.3.1.2.3 Full View

The **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.

2.3.1.2.4 Edit Groups

The **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, click **Edit Groups**. The **Edit Groups** dialog appears.

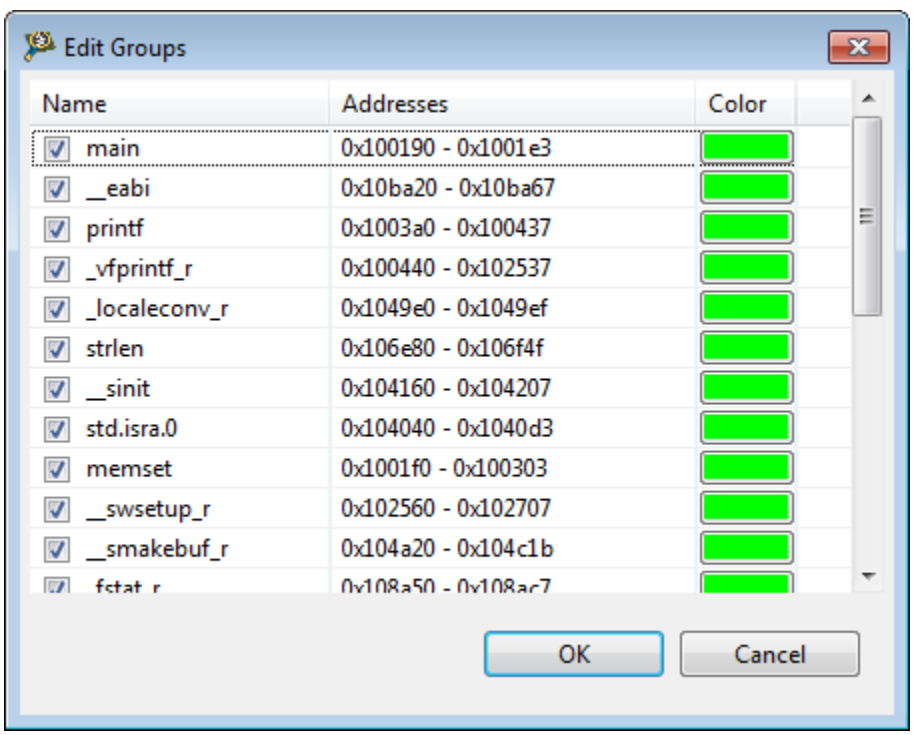


Figure 2-42. Edit Groups dialog

You can perform the following operations in the **Edit Groups** dialog.

- Add/Remove function
- Edit address range of function
- Change color
- Add/Remove group
- Merge groups/functions

2.3.1.2.4.1 Add/Remove function

Right-click the function name in the **Name** column and choose **Insert Function,** or press Control-F to add a function. Choose **Delete Selected** from the shortcut menu to delete the function from the graph. You can disable a function from the graph by deselecting the corresponding checkbox in the **Name** column. Select it again to include it in the graph.

2.3.1.2.4.2 Edit address range of function

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

   The cell becomes editable.

3. Specify an address range for the group/function in the cell.

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

2.3.1.2.4.3 Change color

You can change the color of a function displayed as a horizontal bar in the timeline graph. Click the **Color** column of the corresponding function, and select the color of your choice from the **Color** window that appears.

2.3.1.2.4.4 Add/Remove group

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.

To add a group:

1. Right-click in the row, in the **Edit Groups** dialog, where you want to insert a group, and choose **Insert Group** from the shortcut menu. Alternatively, press the Control-G key.

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

2. Double-click the *new* group cell.

The cell becomes editable.

3. Enter a name for the group, for example, *MyGroup*.
4. Double-click the cell of the corresponding **Addresses** column, and edit the address range according to requirements, for example, `0x105a84 - 0x106e07`.
5. Change the color of the group.

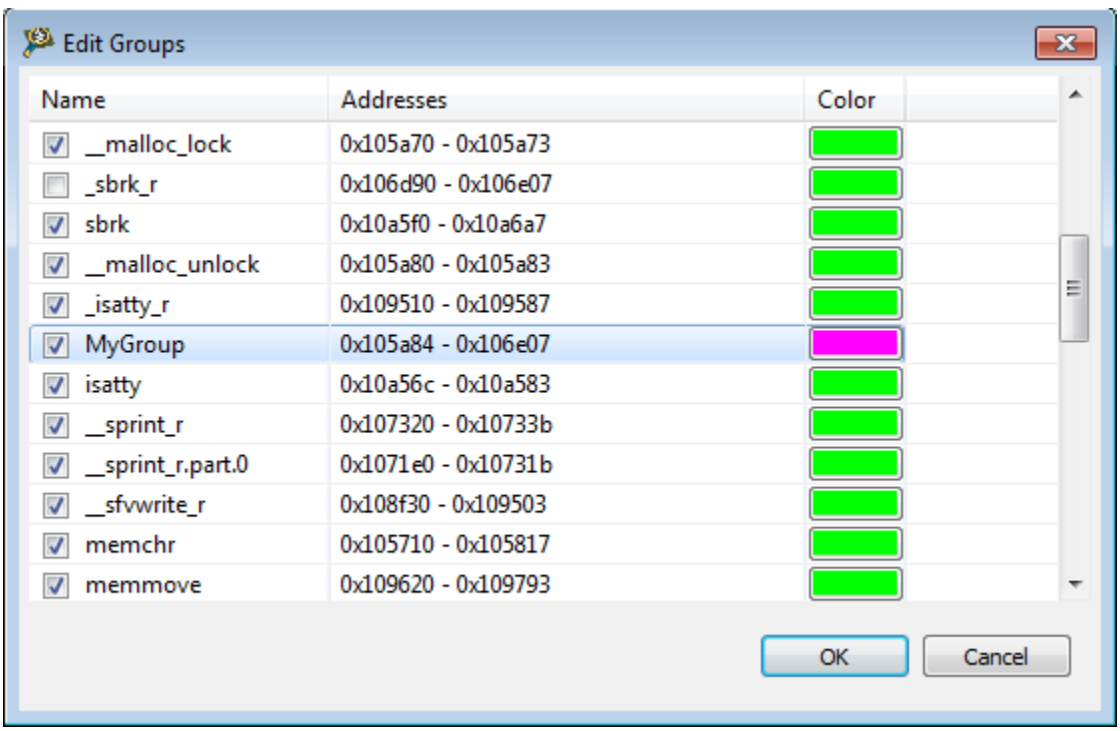


Figure 2-43. Edit Groups dialog after editing address range and color of group

6. Click **OK**.

The *MyGroup* group is added to the timeline.

NOTE

In case the address range you entered overlaps with the address range of any existing function, you get the **Overlapped Groups** message after clicking **OK**. You can disable the functions that are overlapped with the address range of the added group.

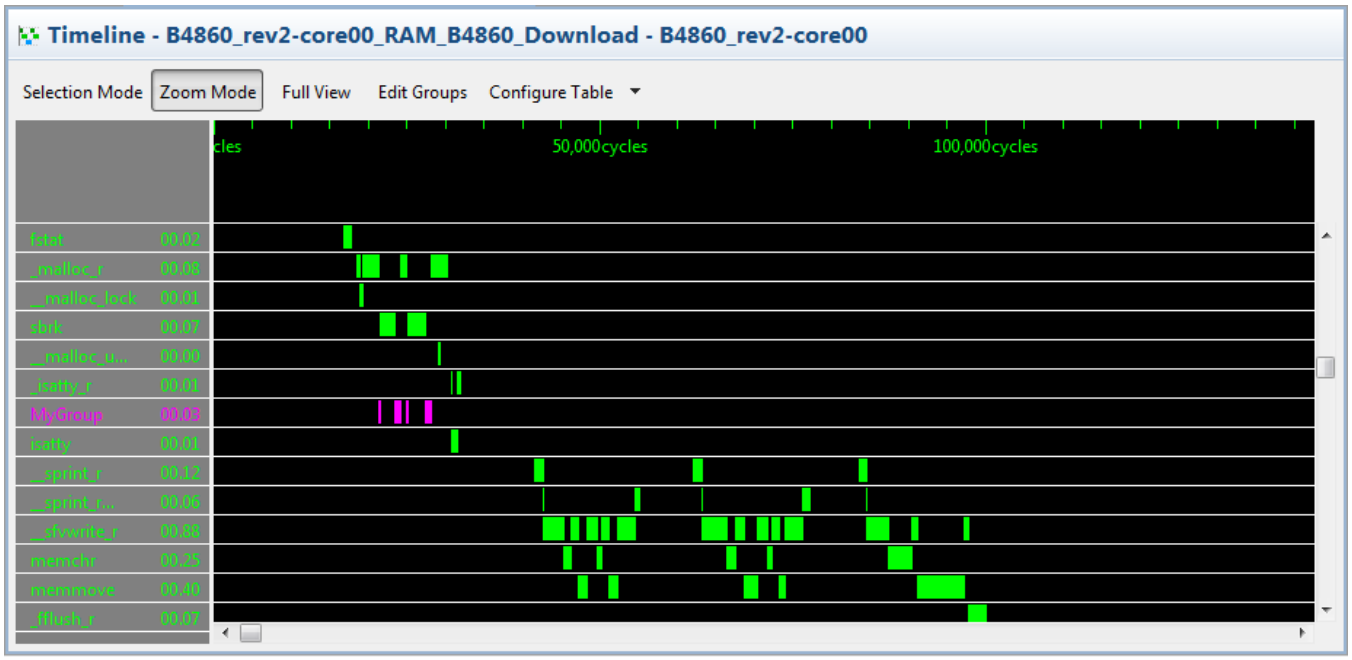


Figure 2-44. Timeline after adding group

To delete a group, select it, right-click in the **Edit Groups** dialog, and choose **Delete Selected** from the shortcut menu. You can also remove a group from the graph by deselecting the corresponding checkbox in the **Name** column. Select it again to include it in the graph.

2.3.1.2.4.5 Merge groups/functions

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

   Both the functions/groups merge into a single function/group that covers both address ranges.
3. Click **OK**.

Merging is useful in case there are many functions and you do not want to view trace of each and every function. You cannot undo this operation, that is you cannot separate the merged functions/groups. To view the original trace data, reopen the **Trace** viewer.

2.3.1.2.5 Configure Time Unit

The **Configure Time Unit** option allows you to set CPU frequency and convert the clock cycles into real time in milliseconds, microseconds, or nanoseconds.

See Configuring time unit for details.

2.3.1.3 Critical Code

The **Critical Code** viewer allows you to analyze the flat profile of your application either at a functional level or at a file level.

To view critical code data:

1. In the **Software Analysis** view, expand the project name.

   All trace collections performed for the project appear.

2. Click the **Critical Code** hyperlink.

   The **Critical Code** viewer appears.

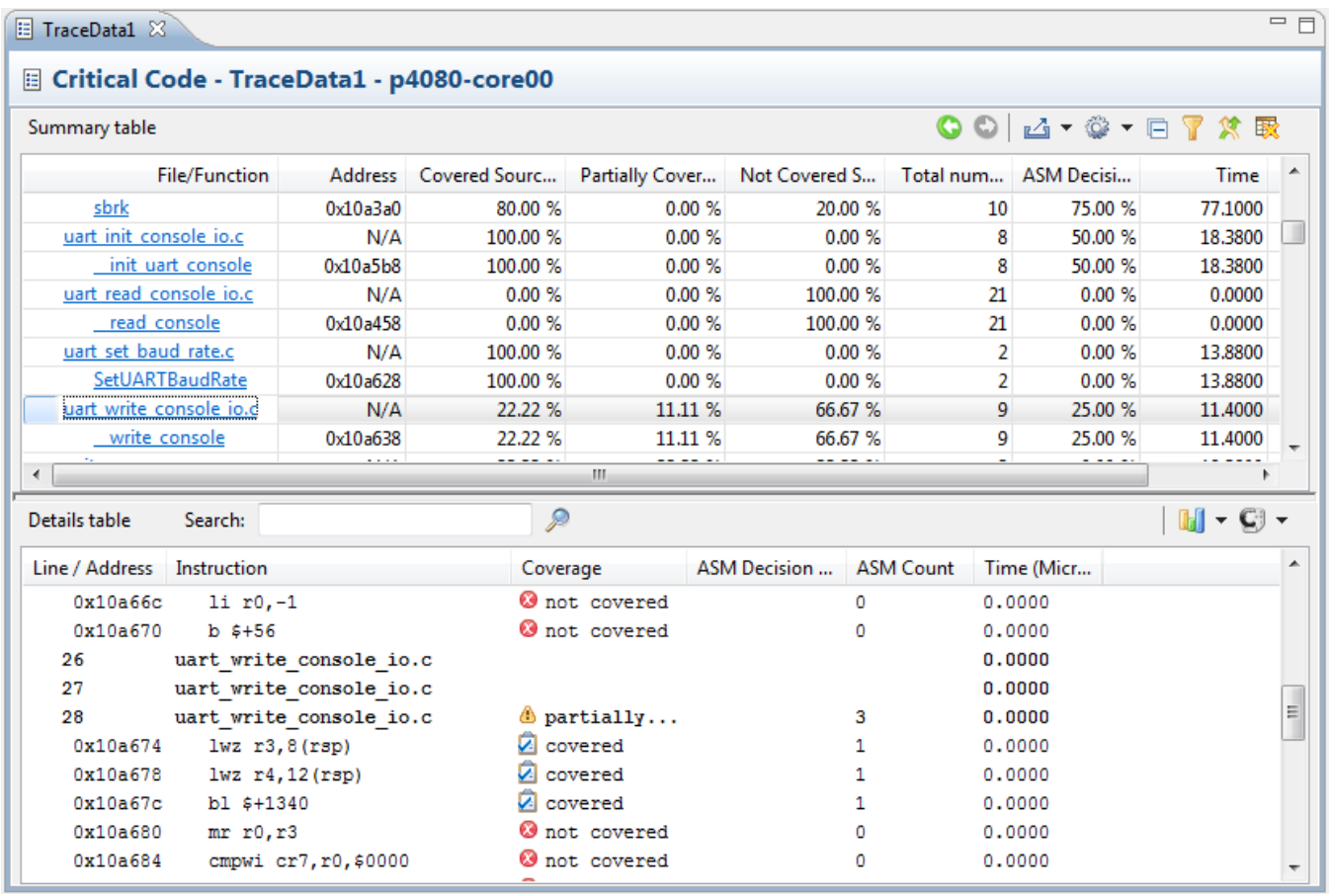


Figure 2-45. Critical Code viewer

The **Critical Code** viewer divides the critical code data into two tabular views: **Summary Table** and **Details Table**.

This topic contains the following sub-topics:

- Summary table
- Details table

2.3.1.3.1 Summary table

The **Summary** table of the **Critical Code** viewer displays the summary of the functions executed in the application.

The **Summary** table provides tree and flat view structures to display critical code data.

The tree view is the default structure in which the functions are grouped by source file. You can expand or collapse in the column to view the functions contained in the corresponding source file. In a flat view structure, all functions are displayed individually. You can switch between tree view or flat view by right-clicking on a column name of the **Summary** table and selecting the **Switch to tree/flat view** option.

The **Summary** table contains the fields as described in the table below. You can switch to ASM instructions statistics or source lines statistics alternatively using the button  available on the toolbar on the right side of the **Summary** table. The columns are movable; you can drag and drop the columns to move them according to your requirements. The table below shows the metrics for ASM level coverage with assembly instructions coverage percentage and total number of assembly instructions per function/module. Clicking the button  to switch to source lines statistics shows metrics for source line coverage with number of source lines covered, not covered, partially covered, and total number of source lines per function/module.

Table 2-24. Summary table - Description of ASM instructions statistics

Name	Description
File/Function	Displays the name of the function that has executed.
Address	Displays the start address of the function.
Covered ASM %	Displays the percentage of number of assembly instructions executed from the total number of assembly instructions per function or per source file.
Not Covered ASM %	Displays the percentage of number of assembly instructions not executed from the total number of assembly instructions per function or per source file.
Total ASM instructions	Displays the total number of assembly instructions per function and per source file.
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.

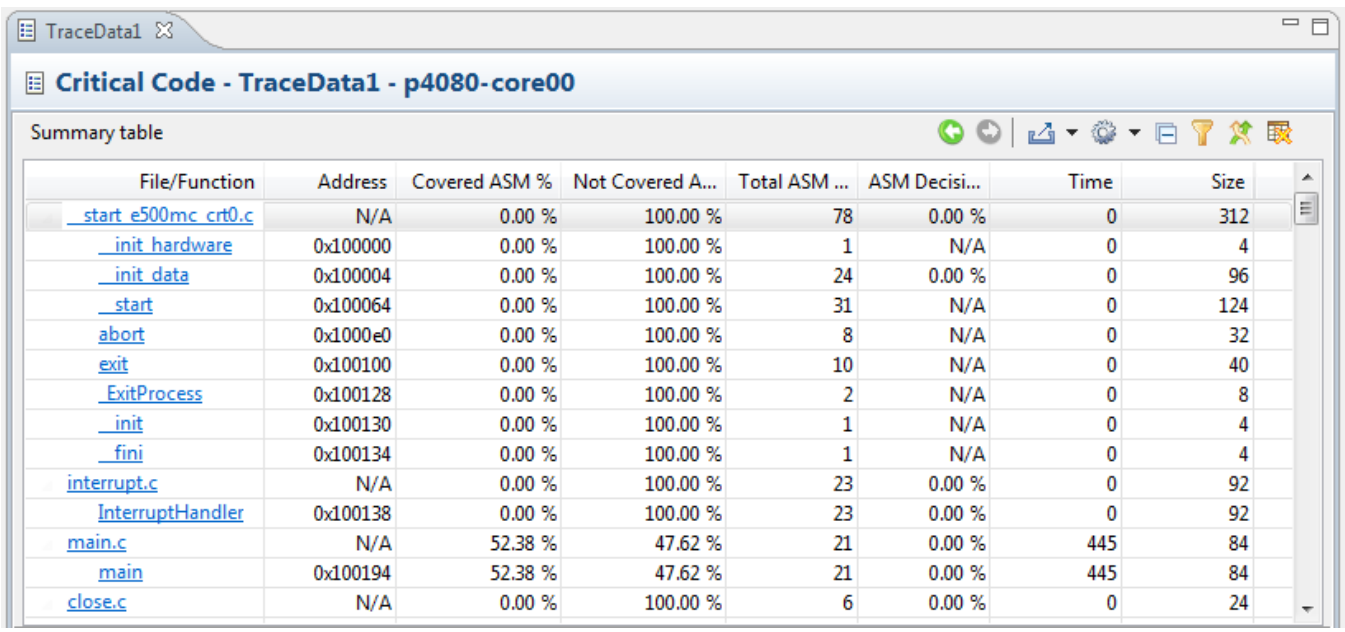
Table continues on the next page...

Table 2-24. Summary table - Description of ASM instructions statistics (continued)

Name	Description
Time	Displays the total number of clock cycles that the function takes.
Size	Displays the number of bytes required by 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.



File/Function	Address	Covered ASM %	Not Covered A...	Total ASM ...	ASM Decisi...	Time	Size
__start_e500mc_crt0.c	N/A	0.00 %	100.00 %	78	0.00 %	0	312
__init_hardware	0x100000	0.00 %	100.00 %	1	N/A	0	4
__init_data	0x100004	0.00 %	100.00 %	24	0.00 %	0	96
__start	0x100064	0.00 %	100.00 %	31	N/A	0	124
abort	0x1000e0	0.00 %	100.00 %	8	N/A	0	32
exit	0x100100	0.00 %	100.00 %	10	N/A	0	40
_ExitProcess	0x100128	0.00 %	100.00 %	2	N/A	0	8
__init	0x100130	0.00 %	100.00 %	1	N/A	0	4
__fini	0x100134	0.00 %	100.00 %	1	N/A	0	4
interrupt.c	N/A	0.00 %	100.00 %	23	0.00 %	0	92
InterruptHandler	0x100138	0.00 %	100.00 %	23	0.00 %	0	92
main.c	N/A	52.38 %	47.62 %	21	0.00 %	445	84
main	0x100194	52.38 %	47.62 %	21	0.00 %	445	84
close.c	N/A	0.00 %	100.00 %	6	0.00 %	0	24

Figure 2-46. Summary table of Critical Code viewer

Click the column header to sort the critical code data by that column. However, you can sort the critical code data only in the flat view structure. The table below lists the buttons available in the statistics view of the **Critical Code** tab.

Table 2-25. Buttons available in Summary table of Critical Code viewer

Name	Button	Description
Previous function		Lets you view the details of the previous function that was selected in the **Summary** table 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 **Summary** table.

Table continues on the next page...

Table 2-25. Buttons available in Summary table of Critical Code viewer (continued)

Name	Button	Description
		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 critical code data in a CSV or html format. Click the button to choose between **Export to CSV** or **Export to HTML** options. The **Export to CSV** option lets you export data of both **Summary** and **Details** tables. The exported html file contains the statistics for all the source files/functions from the **Summary** table along with the statistics of source, assembly or mixed instructions.
Configure table		Lets you show and hide column(s) of the critical code data. Click the button and select the appropriate option to show/hide columns of the **Summary**/**Details** table. The **Drag and drop to order columns** dialog appears in which you can select/deselect 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 expand or collapse all files in the **Summary** table.
Filter files		Allows you to choose the list of files to be displayed in the **Summary** table.
Switch to executable source lines statistics/Switch to ASM instructions statistics		Lets you switch between source lines or ASM instructions to be displayed in the **Summary** table.
Exclude symbols		Lets you exclude statistics libraries or symbols from the critical code data. Click this button to select/specify a function or library you want to exclude from the critical code data. After excluding the selected function, library, or symbol, the statistics will be recomputed and reloaded in the **Critical Code** data viewer.

2.3.1.3.2 Details table

The **Details** table of the **Critical Code** viewer displays the statistics for all the instructions (source and disassembly) executed in a particular source file.

Click a hyperlinked file/function in the **Summary** table of the **Critical Code** viewer to view the corresponding statistics for the instructions executed in that file/function. For example, the statistics of the `strlen()` function are shown in the figure below.

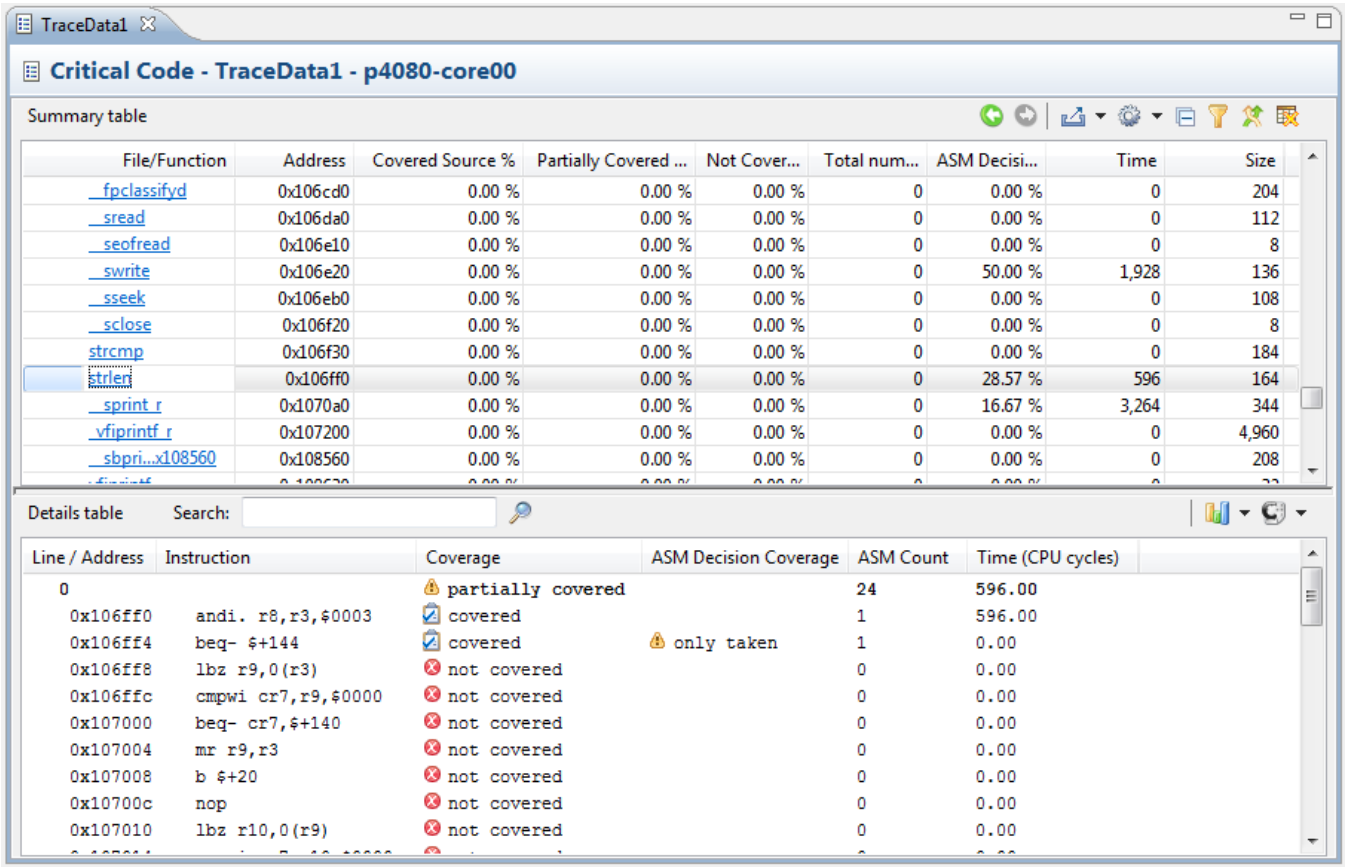


Figure 2-47. Details table

The table below describes the fields of the **Details** table.

Table 2-26. Details table - Description of fields

Name	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 source files, displays if the instructions were covered, not covered, or partially covered.
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 number of clock cycles that each instruction in the function takes.

You can perform the following actions on the **Details** table.

- Search  - Lets you search for a particular text in the **Details** table. In the **Search** text box, type the data that you want to search and click the **Search** button. The first instance of the data is selected in the statistics view. Click the button again or press the *Enter* key to view the next instances of the data.
- Graphics  - Lets you display the histograms in two colors for the **ASM Count** and **Time** columns in the bottom view of the critical code data. Click the button and select the **Assembly/Source > ASM Count or Assembly/Source > Time** option to display histograms in the **ASM Count** or **Time** column. The colors in these columns differentiate source code with the assembly code.
- Show code  - Lets you display the assembly, source or mixed code in the statistics of the critical code data.

2.3.1.4 Performance data

The **Performance** viewer displays the metric and invocation information for each function that executes in the application.

To view performance data:

1. In the **Software Analysis** view, expand the project name.

   All trace collections performed for the project are displayed.

2. Click the **Performance** hyperlink.

   The **Performance** viewer appears.

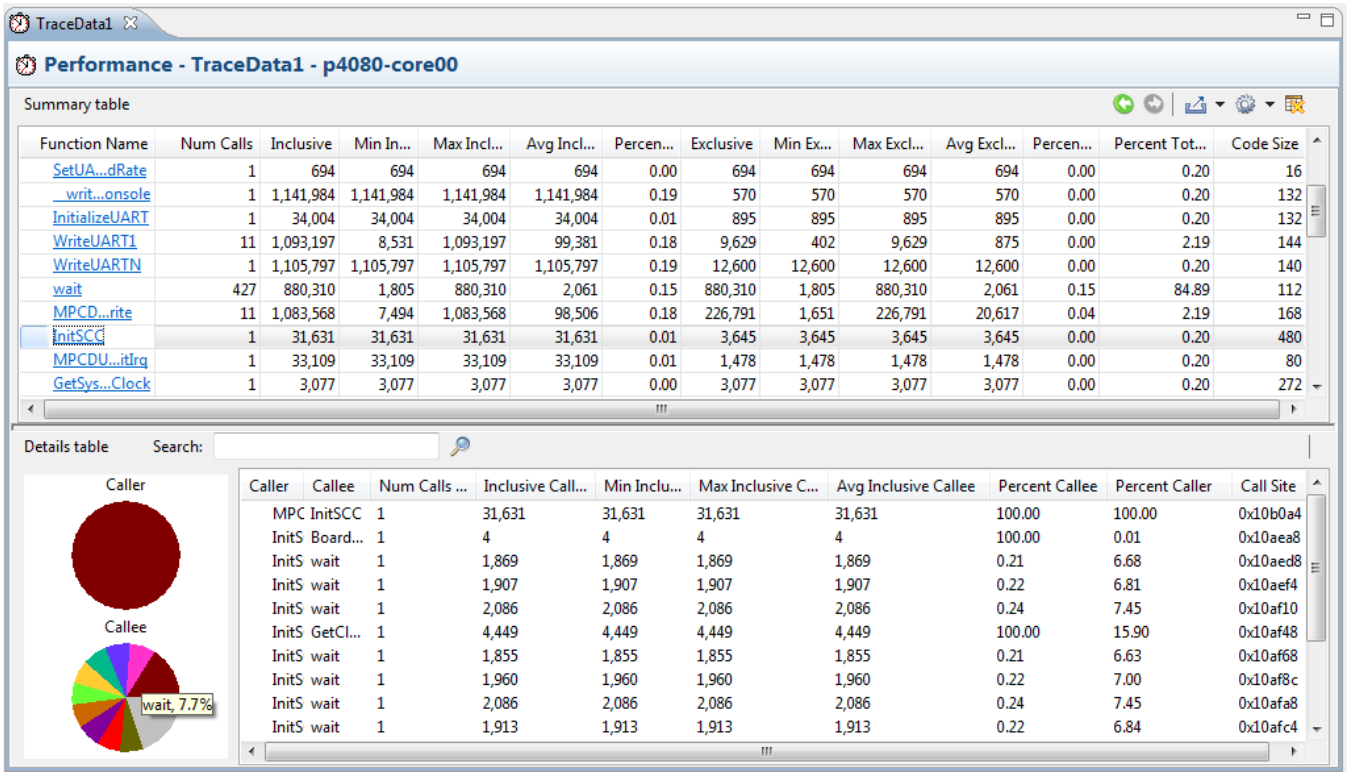


Figure 2-48. Performance viewer

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 table below describes the fields of the **Summary** table.
- The bottom view or the **Details** table presents call pair data for the function selected in the **Summary** table. The **Details** table displays call pair relationships for the selected function, that is which function called which 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 next table below describes the fields of the **Details** table. You cannot sort the columns of this table.

Table 2-27. Field description of Summary table

Name	Description
Function Name	Name of the function that has executed.
Num Calls	Number of times the function has executed.

Table continues on the next page...

Table 2-27. Field description of Summary table (continued)

Name	Description
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.

Table 2-28. Field description of Details table

Name	Description
Caller	Name of the calling function.
Callee	Name of the function that is called by the calling function.
Num Calls Callee	Number of times the caller called the callee.
Inclusive Callee	Cumulative metric count during execution time spent from function entry to exit.
Min Inclusive Callee	Minimum metric count during execution time spent from function entry to exit.
Max Inclusive Callee	Maximum metric count during execution time spent from function entry to exit.
Avg Inclusive Callee	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.

You can move the columns to the left or right of another column by dragging and dropping. You can perform the Export and Configure table actions on the performance data similar to critical code data. You can also view the previous and next functions of the performance data using the icons available in the lower section of the **Performance** viewer. For details on these icons, see Table 2-25.

2.3.1.5 Call Tree

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

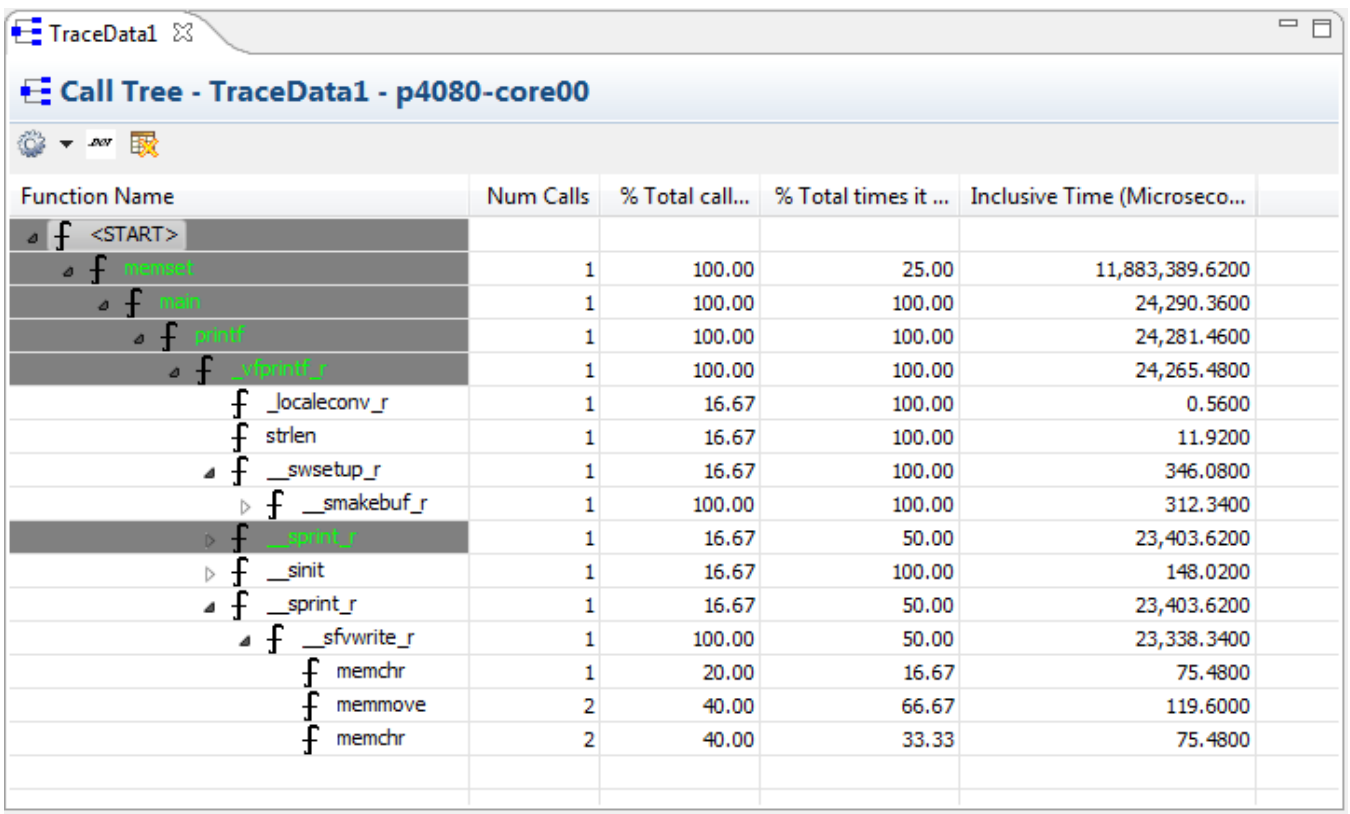
To view call tree data:

1. In the **Software Analysis** view, expand the project name.

   All trace collections performed for the project are displayed.

2. Click the **Call Tree** hyperlink.

   The **Call Tree** viewer appears.



Function Name	Num Calls	% Total call...	% Total times it ...	Inclusive Time (Microseco...
<START>				
memset	1	100.00	25.00	11,883,389.6200
main	1	100.00	100.00	24,290.3600
printf	1	100.00	100.00	24,281.4600
_vfprintf_r	1	100.00	100.00	24,265.4800
_localeconv_r	1	16.67	100.00	0.5600
strlen	1	16.67	100.00	11.9200
__swsetup_r	1	16.67	100.00	346.0800
__smakebuf_r	1	100.00	100.00	312.3400
__sprint_r	1	16.67	50.00	23,403.6200
__sinit	1	16.67	100.00	148.0200
__sprint_r	1	16.67	50.00	23,403.6200
__sfvwrite_r	1	100.00	50.00	23,338.3400
memchr	1	20.00	16.67	75.4800
memmove	2	40.00	66.67	119.6000
memchr	2	40.00	33.33	75.4800

Figure 2-49. Call Tree viewer

In the **Call Tree** viewer, START is the root of the tree. You can click "+" 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 the node to view the source code.

The table below describe the fields of **Call Tree** data. The columns are movable; you can move the columns to the left or right of another column by dragging and dropping.

Table 2-29. Call Tree viewer fields

Name	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 (Microseconds; 50.0 MHz)	Cumulative count during execution time spent from function entry to exit.

You can perform the following actions using the buttons available on the toolbar of the **Call Tree** viewer:

- **Configure Table** - Sets CPU frequency and set Inclusive Time displayed in the **Call Tree** viewer in cycles, milliseconds, microseconds, and nanoseconds. Click the **Configure Table** button  to configure time unit and set CPU frequency.
- **Export to dot** - Exports call tree data in the *.dot* format using the button  .
- **Exclude Symbols** - Lets you exclude statistics libraries or symbols from the call tree data. Click the button  to select or 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.

2.3.1.6 Nexus messages

For QorIQ, trace is stored in the form of Nexus messages. Normally, these messages are translated to higher-level trace events that are displayed in the **Trace** viewer. The Nexus messages themselves can also be seen in the **Nexus Messages** view.

This topic describes the following sub-topics.

- Nexus Messages view
- Nexus messages in Trace viewer

2.3.1.6.1 Nexus Messages view

Click the **Nexus Messages** hyperlink in the **Software Analysis** view to open the **Nexus Messages** view.

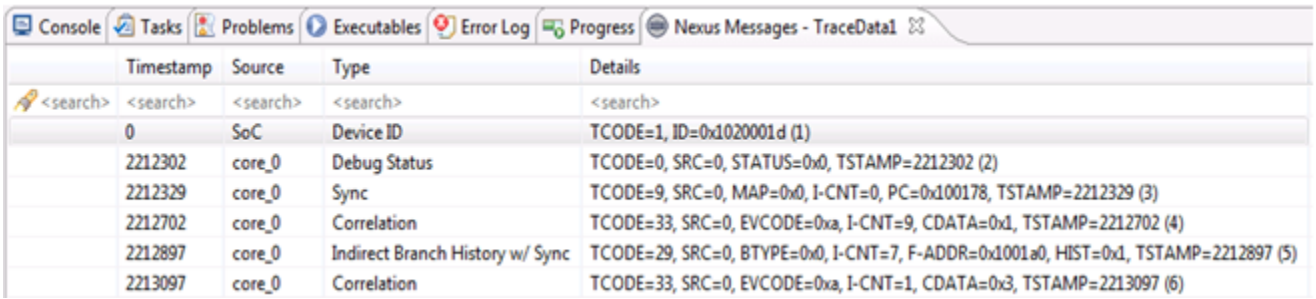


Figure 2-50. Nexus Messages view

The **Nexus Messages** view contains the following fields.

Table 2-30. Nexus Messages view fields description

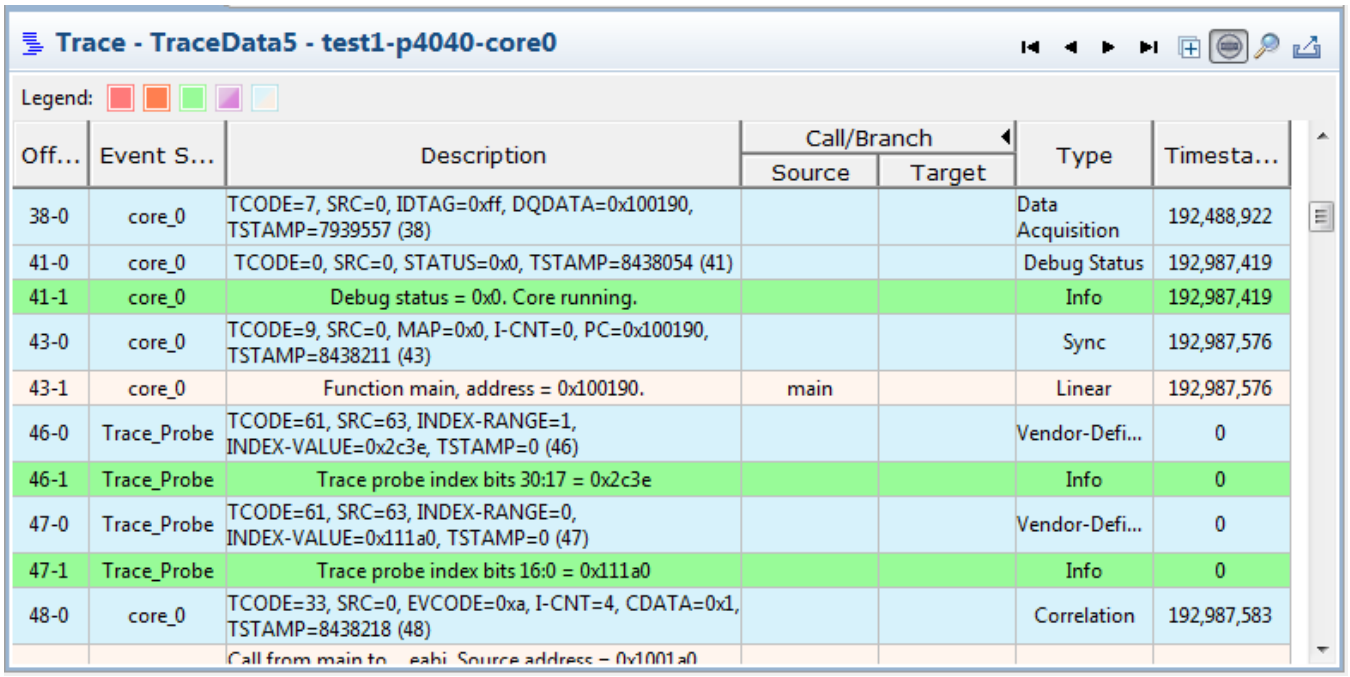
Name	Description
Timestamp	Specifies the timestamp value that is expressed as clock ticks.
Source	Specifies the trace source that produced the event.
Type	Specifies the Nexus Message type as encoded in the TCODE.
Details	Displays detailed information of the Nexus fields for each Nexus Message type. It includes a Nexus Message index in parenthesis for easy correlation with the **Trace** viewer.

NOTE

You can also open the **Nexus Messages** view by choosing **Window > Show View > Other > Software Analysis > Nexus Messages**.

2.3.1.6.2 Nexus messages in Trace viewer

To display Nexus messages in the **Trace** viewer, click the **Show Nexus Messages** button available in the toolbar of the **Trace** viewer. The figure below shows the **Trace** viewer displaying Nexus messages.



Off...	Event S...	Description	Call/Branch Source	Call/Branch Target	Type	Timesta...
38-0	core_0	TCODE=7, SRC=0, IDTAG=0xff, DQDATA=0x100190, TSTAMP=7939557 (38)			Data Acquisition	192,488,922
41-0	core_0	TCODE=0, SRC=0, STATUS=0x0, TSTAMP=8438054 (41)			Debug Status	192,987,419
41-1	core_0	Debug status = 0x0. Core running.			Info	192,987,419
43-0	core_0	TCODE=9, SRC=0, MAP=0x0, I-CNT=0, PC=0x100190, TSTAMP=8438211 (43)			Sync	192,987,576
43-1	core_0	Function main, address = 0x100190.	main		Linear	192,987,576
46-0	Trace_Probe	TCODE=61, SRC=63, INDEX-RANGE=1, INDEX-VALUE=0x2c3e, TSTAMP=0 (46)			Vendor-Defi...	0
46-1	Trace_Probe	Trace probe index bits 30:17 = 0x2c3e			Info	0
47-0	Trace_Probe	TCODE=61, SRC=63, INDEX-RANGE=0, INDEX-VALUE=0x111a0, TSTAMP=0 (47)			Vendor-Defi...	0
47-1	Trace_Probe	Trace probe index bits 16:0 = 0x111a0			Info	0
48-0	core_0	TCODE=33, SRC=0, EVCODE=0xa, I-CNT=4, CDATA=0x1, TSTAMP=8438218 (48)			Correlation	192,987,583
		Call from main to __eabi, Source address = 0x1001a0				

Figure 2-51. Trace viewer displaying Nexus messages

The indices in the **Offset** column are shown in pairs. The left number corresponds to the Nexus Message number. For the right number, 0 is used for the actual Nexus Message, while 1- n are used for the trace events derived from the Nexus Message. The rest of the fields are same as the fields of the **Trace** viewer.

To hide the Nexus messages, click the **Show Nexus Messages** button again.

NOTE

Nexus messages are available in the **Trace** Viewer only until the Nexus messages have been fully translated to higher-level trace events. Afterwards, Nexus messages can only be viewed in the the **Nexus Messages** View.

2.3.1.7 Configure

The **Configure** hyperlink allows you to configure the results collected in the **Critical Code**, **Performance**, and **Call Tree** viewers.

This feature allows you to filter symbols or full libraries from the data results before they are computed.

Click the hyperlink to open the **Configure results** dialog. For details on the **Configure results** dialog, see Excluding symbols from data results.

2.3.2 Finding and filtering trace data

You can find and filter the trace data displayed in the **Trace** viewer.

Click the button to find a particular set of events that define a set of criteria for filtering desired events within the current trace data set.

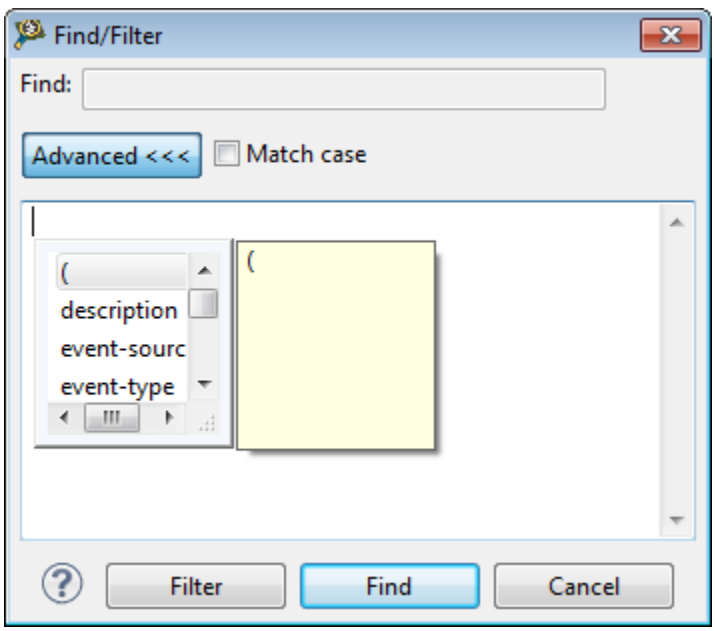


Figure 2-52. Find and Filter dialog

To find events in trace data:

1. On the **Trace** viewer toolbar, click the Find and Filter button to open the **Find/ Filter** window and enter the text you want to find in the **Find** textbox.
2. Click the **Advanced** button if you wish to specify a search criteria for your entry.
3. Double-click on one of the fields in the list box for defining your search criteria and add the entry to the textbox next to **Advanced** button.
4. Select an operator from the list box. For numerical fields, standard equivalency operators are provided (for example, =, >, !=, depending upon the field). For text fields, **Contains**, **Doesn-t-contain**, **Doesn-t-equal**, and **Equals** operators are provided:
   - Contains - tests that the character string entered is contained in the field value.
   - Doesn-t-contain - tests that the character string entered is not contained in the field value.

- Doesn-t-equal - tests that the character string entered does not exactly match the string in the field value.
- Equals - tests that the character string entered exactly matches the string in the field value.

5. Select the value for comparison by typing in the **Value** field.
6. If you want to add more fields to the search expression, define another entry.
7. Repeat the steps to add additional fields to the search expression.
8. Click **FIND** or press **F3**. The focus in the data will move to the first event that matches the search expression.
9. Press **F3** on your keyboard to locate the next event that matches the search expression.

   A search for events matching the specified criteria is initiated. Every click on **Find** will move the cursor to the next event searched in the **Trace** viewer according to the criteria specified.

The search starting point is determined by the currently selected cell in the **Trace** viewer and the search direction. If no cell is currently selected then the search starts at the beginning of the trace data if the search direction is forward, or at the end of the trace data if the search direction is backward.

If a cell is currently selected then the search begins in the row immediately after the selected row if the search direction is forward, or in the row immediately preceding the selected row if the search direction is backward.

NOTE

Similar to filtering, you can add appropriate logical operators to the compare expression while specifying a criteria for locating a particular set of data.

To filter events:

1. Perform steps 2-7 of Finding and filtering trace data to specify the criteria for filtering a particular data. For example, to filter the data where *Type* is equal to *Info*.

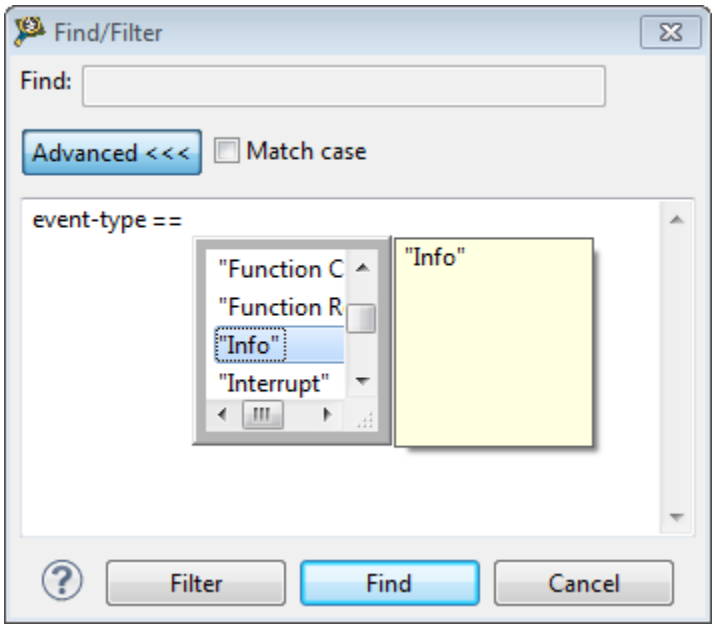
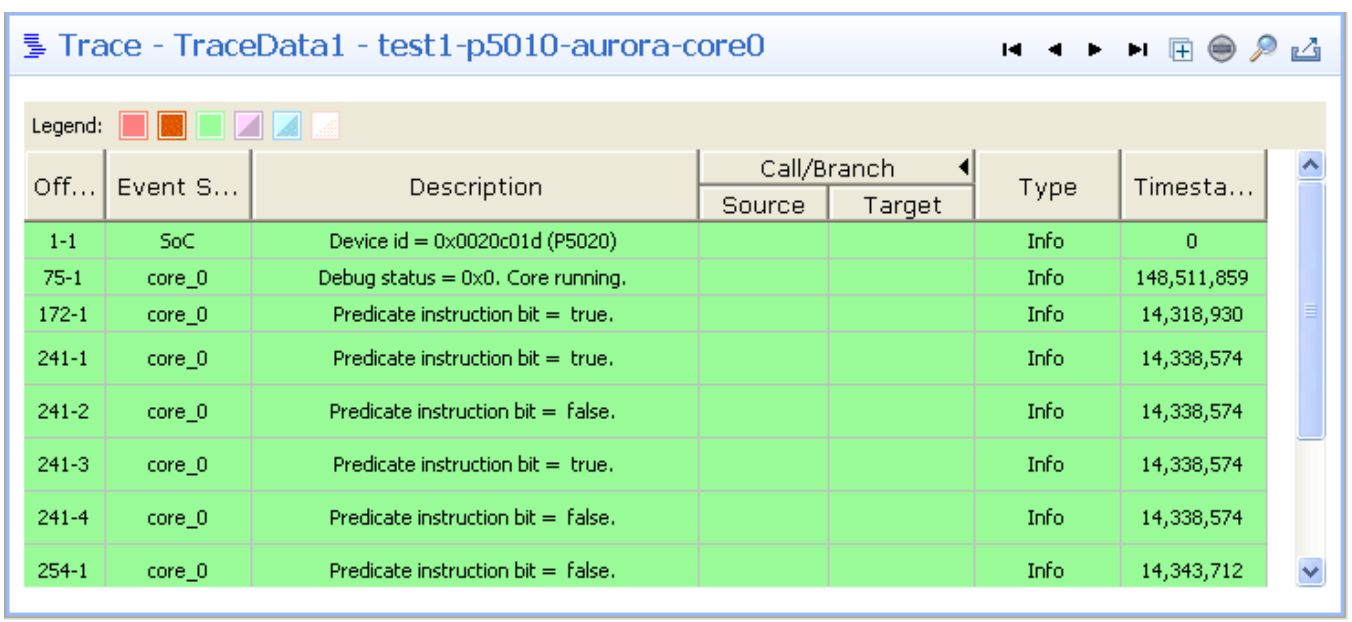


Figure 2-53. Specifying criteria for filtering

2. Click **Filter**.

   The data will be filtered to show only the data that matches the filter expression.



Off...	Event S...	Description	Call/Branch Source	Call/Branch Target	Type	Timesta...
1-1	SoC	Device id = 0x0020c01d (P5020)			Info	0
75-1	core_0	Debug status = 0x0. Core running.			Info	148,511,859
172-1	core_0	Predicate instruction bit = true.			Info	14,318,930
241-1	core_0	Predicate instruction bit = true.			Info	14,338,574
241-2	core_0	Predicate instruction bit = false.			Info	14,338,574
241-3	core_0	Predicate instruction bit = true.			Info	14,338,574
241-4	core_0	Predicate instruction bit = false.			Info	14,338,574
254-1	core_0	Predicate instruction bit = false.			Info	14,343,712

Figure 2-54. Filtered data results

2.3.3 Exporting trace data

You can export the collected trace data to a CSV file using the **Exporting Trace Data** dialog.

The **Exporting Trace Data** dialog appears on clicking the  button. The **Exporting Trace Data** dialog lets you define a set of criteria for exporting desired trace data to a CSV file.

The CSV file format is supported for import to common spreadsheet applications. It contains the event names and values of the collected samples.

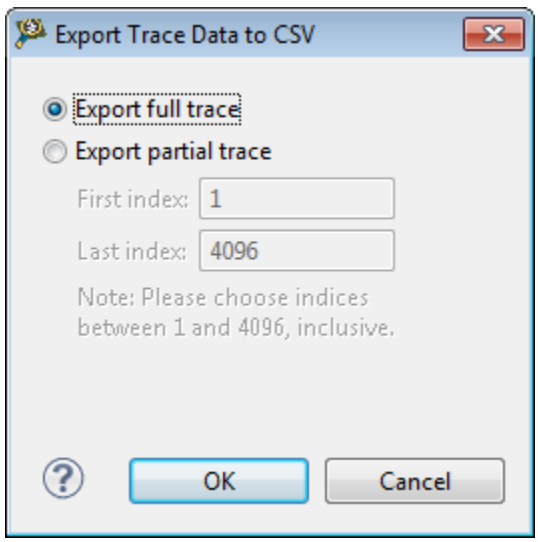


Figure 2-55. Exporting trace data

To export trace data to a CSV file:

1. Select one of the following options in the **Export Trace Data to CSV** dialog
   - Export Full Trace - Allows you to export the entire trace data to the CSV file.
   - Export Partial Trace - Allows you to export the desired trace data to the CSV file depending on the indices selected in the **First index** and **Last index** text boxes. The trace data falling between the **First index** and **Last index** will be exported to the CSV file.
2. Click **OK**.

   The **Export Trace Data to CSV** dialog appears.
3. Browse to the location where you want to save the trace data and click **Save**.

2.3.4 Configuring time unit

The **Configure TimeUnit** option allows you to set CPU frequency and convert the clock cycles into real time in milliseconds, microseconds, or nanoseconds.

Click this button and choose the **Configure Time Unit** option. To convert the clock cycles into milliseconds, microseconds, or nanoseconds, select the corresponding option.

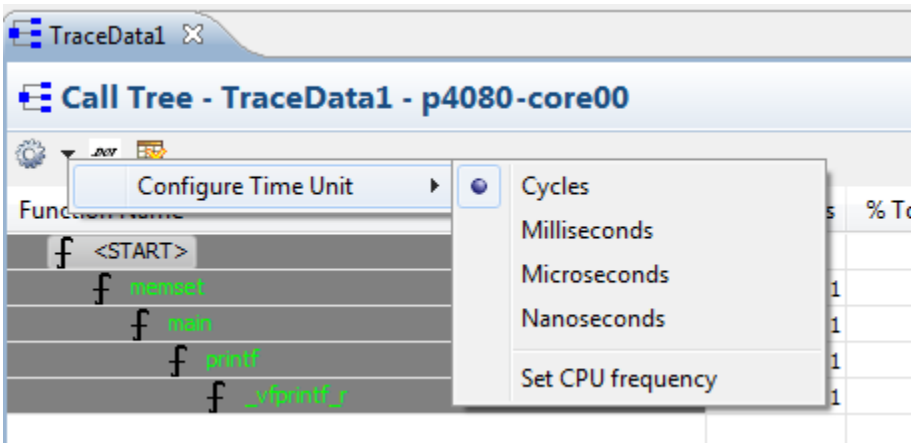


Figure 2-56. Configuring Time Unit

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

NOTE

The **Configure Time Unit** option is available in **Critical Code Data**, **Performance**, and **Call Tree** viewers. All the viewers share the same time and frequency that you set for a particular viewer. For example, if you set time in microseconds for **Critical Code Data** viewer, all other viewers will display time in microseconds.

2.3.5 Excluding symbols from data results

The Exclude symbols feature allows you to configure the statistics before or after the data is processed in the **Critical Code**, **Performance**, and **Call Tree** viewers.

You can configure the statistics using the **Configure** hyperlink available in the **Software Analysis** view or using the **Exclude Symbols** button available on the toolbar of the **Critical Code**, **Performance**, and **Call Tree** viewers.

To configure the statistics before data results are computed, click the **Configure** hyperlink in the **Software Analysis** view. To configure the statistics after data results are computed and then reload the **Critical Code**, **Performance**, and **Call Tree** viewers with the recomputed statistics, click the **Exclude Symbols** button in the toolbar.

Perform the following steps to configure the statistics before results are computed or processed.

1. Open the **Software Analysis** view after collecting the data results.
2. Click the **Configure** hyperlink available under the **Config Results** column.

   The **Configure results** dialog appears.

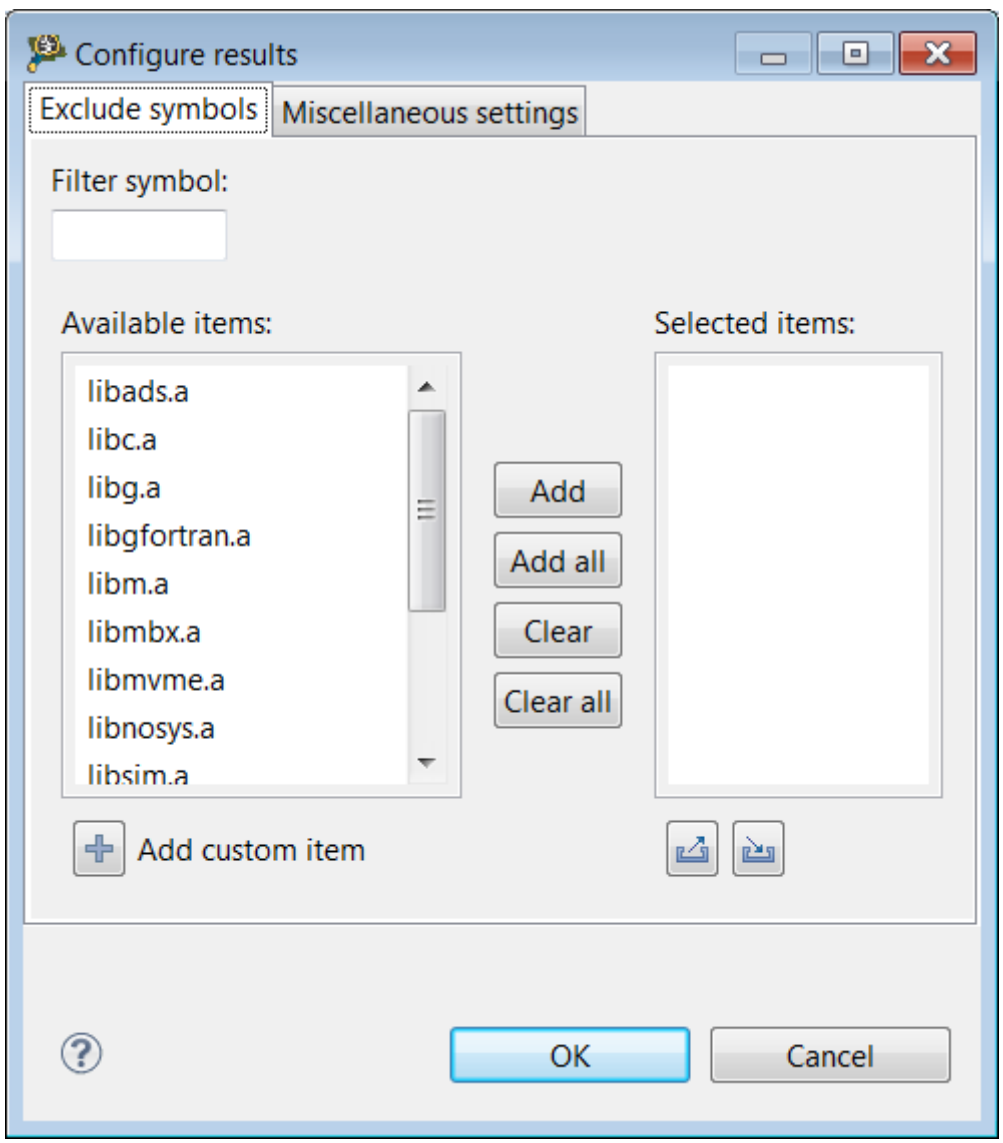


Figure 2-57. Configure results dialog - Exclude symbols tab

3. From the **Exclude symbols** tab, select the libraries from the **Available items** list and click **Add**.

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

4. Click **OK**.

   The configuration is saved with the list of all symbols and libraries that need to be excluded from statistics.

5. Open the **Critical Code**, **Performance**, and **Call Tree** data viewers.

   The viewers will display results excluding the selected libraries and symbols.

Perform the following steps to configure the statistics after results are computed or processed in the **Critical Code**, **Performance**, or **Call Tree** viewer.

1. Click the **Exclude Symbols** button in the opened viewer.

   The **Configure results** dialog appears.

2. From the **Exclude symbols** tab, select the existing configuration or modify the configuration by adding or deleting the libraries or symbols from the **Selected items** list.
3. Click **OK** to save the settings.

   The configuration is applied on the data, that is the symbols and libraries that are present in the **Selected items** list will be excluded from the data results. The old statistics will be deleted and the new statistics will be computed based on the configuration, and the 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 figures show an example that the `printf` symbol is excluded from the computed data. If the filtered symbol is called from a non-filtered function, it will appear dimmed in the **Details** table.

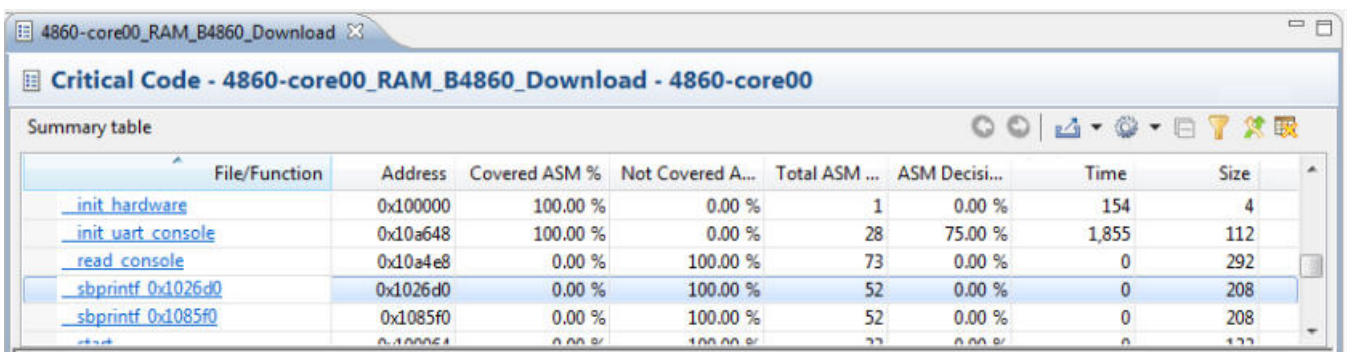


Figure 2-58. Critical Code viewer with excluded symbols

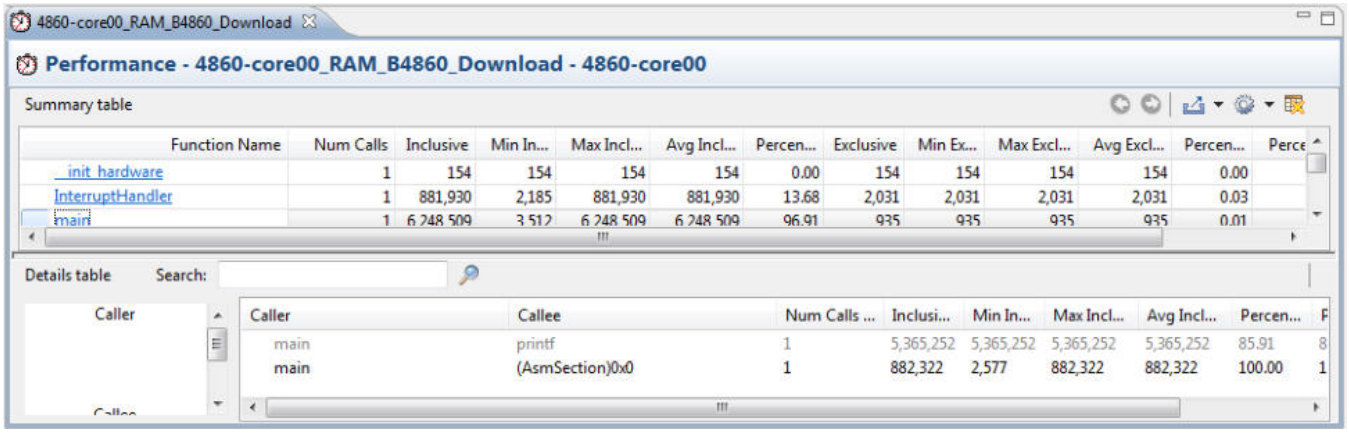


Figure 2-59. Performance viewer with dimmed filtered results

NOTE

Applying the configuration in one result viewer (Critical Code, Performance or Call Tree) will reload the new recomputed results automatically in all other opened viewers.

You can also perform the following actions using the **Exclude symbols** tab in **Configure results** dialog.

Table 2-31. Actions available in Exclude Symbols dialog

Name	Description
Filter symbol	Acts as a real time filter that lets you search a symbol in a library from the **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	Lets you add a customized symbol or an entire library (*.elb) to the **Available Items** list. Click this button to open the **Add Custom Item** dialog, and browse the desired symbol or library on your machine. The custom item that you select gets added to the **Available Items** list, and you can exclude this custom item from the data results and recompute them in the respective viewers. Duplicate custom items are not allowed to enter the **Available Items** list.
Export item list to a file	Lets you export the current configuration in a file. The file is saved with the *.sym* extension in the **Analysis Results** folder of the current project location. You can then apply the same settings of this configuration into another project by importing it.
Import item list from file	Lets you import or load an existing configuration file on your machine.

NOTE

If the **Configure results** dialog is opened for a new project, the latest saved configuration is loaded into the **Available Items** and **Selected Items** lists as a default configuration. You can either use this configuration or modify it to exclude the desired symbols and libraries.

When scrolling through unprocessed data in the **Trace** viewer, the timestamp values increase from the beginning of the trace, and at some point of time, the value resets. Also, the timestamp value does not increase beyond 16 million (approximately). You can use the **Configure results** dialog to control the timestamps overflow 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 values 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 timestamp values increase till approximately 16 million and reset at some point.

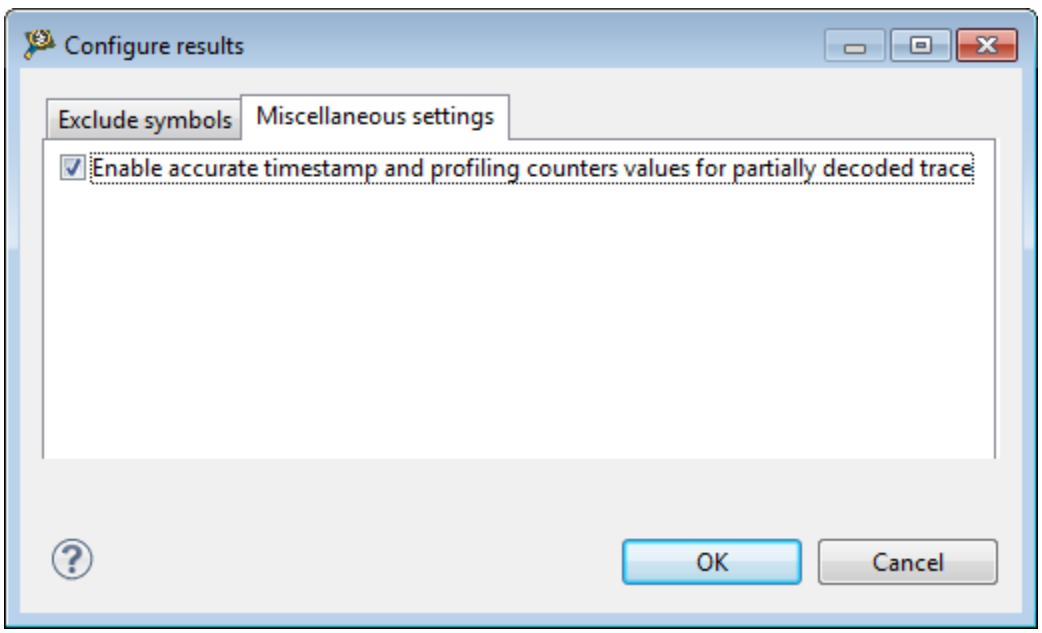


Figure 2-60. Configure results dialog - Miscellaneous settings tab

2.4 Preparing Aurora interface for collecting Aurora Nexus trace

If you want to collect Nexus trace from the target via 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 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.

This topic contains the following sub-topics:

2.4.1 Enabling Aurora

You can enable Aurora on the following boards using Reset Configuration Word (RCW).

- B4860QDS
- T4240/T4160QDS
- P4080DS
- P5010DS
- P5020/21DS
- P5040DS
- P3041DS
- P5040RDB
- P5020RDB
- P5021RDB
- P5010RDB
- P2040/41RDB

NOTE

T4240RDB, T4160RDB, and P4080 PCI Express (PCIe) boards do not support Aurora.

The RCW and switch configuration for these boards is as follows:

- On all the supported boards, the Aurora debug lanes must be configured to run at 2.5 Gbit/s.
- For P5010/20DS, both the `SW5[1:2]` switches must be off (00) to select a 100MHz SERDES reference clock. Also, the `SRDS_DIV_B1` for the I-J lanes (RCW bit 143) must be on in the used RCW to enable the clock divider.
- For P5040DS and P5021DS boards, `SW2.5 (LANE_8_SEL)` must be 0 and `SW11.6 (LANE_9_SEL)` must be 1 for Aurora. Also, `SRDS_PRTCL` mush have the Debug I-J lanes set for debug.
- 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.
- For P3041DS boards, only one Aurora receive/transmit lane is used as opposed to all the other supported platforms that use two lanes. However, this is done in CodeWarrior, by default, in the debug connection settings. Also, unlike all other P-

series boards, lanes I-J are not used for debugging Aurora on P3041DS board. Lane A in Bank 2 is used to support the Aurora debug connector for P3041board.

- For P5040RDB, P5010RDB, P5020RDB and P5021RDB boards, `SW8.8` (`SW_LEGACY_POD_B`) must be 1 to connect the JTAG port to the Aurora connector. Also, `SRDS_PRTCL` must have the Debug I-J lanes set for debug, and the `SRDS_DIV_B1` for the I-J lanes (RCW bit 143) must be 1 to enable the clock divider.
- For P2040/41RDB boards, `SW2.8` must be `1` to select Aurora and `SW[5:6] = 00` to select 100MHz reference clock for SerDes bank 2. Also, `SRDS_PRTCL` must have lane A (SerDes, bank 2) set for Debug, which can be done using one of the following configurations: `SRDS_PRTCL_14` or `SRDS_PRTCL_1C`.

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

NOTE

The B4060 board or any other target that does not have Power Architecture cores is not supported for collecting trace over Aurora, since there is no core to execute u-boot.

2.4.2 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 various targets as follows.

- On QorIQ processor P series
- On T4240/T4160/T2080QDS target
- On B4860QDS target

2.4.2.1 On QorIQ processor P series

For the following QorIQ Processor P series targets, the TCL script that is used to train Aurora on the target is available in the CodeWarrior installation folder at: `PA/PA_Support/Initialization_Files/Aurora/train_aurora.tcl`

- P4080DS
- P5010DS
- P5020DS
- P5021DS
- P5040DS

- P3041DS
- P5040RDB
- P5020/10 RDB
- P5021RDB
- P2040/41RDB

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

There are different ways to execute the Aurora training script on the QorIQ Processor P series targets depending on the following scenarios.

This topic contains the following sub-topics:

- Setting Up debugger launch - General approach
- Downloading and debugging application using CodeWarrior
- Attaching to and debugging target using CodeWarrior
- Executing training from debugger console
- Initializing Aurora trace outside CodeWarrior debugger

2.4.2.1.1 Setting Up debugger launch - General approach

You can set up a debugger launch configuration to execute **train_aurora** before executing normal debug initialization script to collect trace via Aurora on the Aurora probe during the debug session.

Set the `Initialize target script` field of the `Hardware or Simulator Target` specification to `train_aurora.tcl` so that **train_aurora** is called immediately before the normal initialization file is executed when a connection is established from CodeWarrior.

The routine **train_aurora** can also be called directly from the **Debugger Shell** after a connection has been established.

2.4.2.1.2 Downloading and debugging application using CodeWarrior

If you are downloading and debugging the application using CodeWarrior, execute the TCL script using the following steps:

1. Copy the TCL training script to your project directory hierarchy in the folder `<Project folder>/CFG`. Rename this file to avoid confusion with other versions needed for other use cases. For example, rename it from `train_aurora.tcl` to `train_aurora_download.tcl`.
2. Edit this file to call the **train_aurora** routine, and then execute the appropriate initialization script (present in the folder `<Project folder>/CFG`) from the training script copied in step 1. For example, if P4080 ComE is the target device, and you want to debug without cache, a call to the training script and then a call to the

`P4080ComE_init_core.tcl` initialization script should be added as follows at the very bottom of the training script file:

```
train_aurora

source [file dirname [info script]]/P4080ComE_init_core.tcl
```

3. Modify your target configuration to point to the training script you just edited.
   a. Open the **Debug Configurations** dialog and select the required configuration.
   b. Click **Edit** in the **Main** tab for the connection setting.
   c. Choose the appropriate target (or create a new one) in the **Target** pop-up menu.
   d. Edit the new target settings by clicking **Edit**.
   e. Specify the training script edited in step 2 in the `Initialize target script` for each core.
   f. Click **OK**.

   Now, whenever a launch configuration is debugged that uses the target modified to execute the new `Initialize target script`, Aurora will be initialized and ready for trace collection.

If the Aurora probe fails training on a clean P2040/41RDB configuration, you can apply the following workaround to get the probe trained:

1. Start a debug session using JTAG and select the `train_aurora.tcl` init file in the connection settings. Make sure that you select the correct jtag config file.
2. After the board stops at the start of main file, open the **Debugger Shell** window by choosing **Window > Show View > Debugger Shell**, and run the following command (similar to the `reset_to_user` command run from CCS console):

   ```
   protocol ccs::reset_to_user
   ```

3. Reset the probe from telnet and wait for the CCS connection to re-establish.
4. Terminate the current session from CodeWarrior and debug again.

   You can now collect trace.

2.4.2.1.3 Attaching to and debugging target using CodeWarrior

If you are attaching to and debugging the target using CodeWarrior, execute the TCL script using the following steps:

1. Copy the TCL training script to your project directory hierarchy in the folder `<Project folder>/CFG`. Rename this file to avoid confusion with other versions needed for other use cases. For example, rename it from `train_aurora.tcl` to `train_aurora_attach.tcl`.

2. Edit this file to call the **train_aurora** routine by adding the following line at the bottom of the script file:

```
train_aurora
```

3. Modify your target configuration to point to the training script you just edited.
   a. Open the **Debug Configurations** dialog and select the required configuration.
   b. Click **Edit** in the **Main** tab for the connection setting.
   c. Choose the appropriate target (or create a new one) in the **Target** pop-up menu.
   d. Edit the new target settings by clicking **Edit**.
   e. Specify the training script edited in step 2 in the `Initialize target script` for each core.
   f. Click **OK**.

   Now, whenever a launch configuration is debugged that uses the target modified to execute the new `Initialize target script`, Aurora will be initialized and ready for trace collection.

2.4.2.1.4 Executing training from debugger console

You can execute the training script from the CodeWarrior **Debugger Shell** window (**Window > Show View > Debugger Shell**). Type the following command in the window to source the training script:

```
source "path/train_aurora.tcl"
```

To execute the training script, call **train_aurora**.

2.4.2.1.5 Initializing Aurora trace outside CodeWarrior debugger

You can run the training script from CCS when CodeWarrior is not being used for debugging. You need to open a CCS console, source the `train_aurora.tcl` file, and then execute the training by calling **train_aurora**.

2.4.2.2 On T4240/T4160/T2080QDS target

For T4240/T4160/T2080QDS target, the TCL script that is used to train Aurora on the target is available in the CodeWarrior installation folder at:

```
PA/PA_Support/Initialization_Files/Aurora/train_aurora_t4.tcl
```

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

To train Aurora on the T4240/T4160/T2080QDS target:

1. Ensure that the RCW on board is having Aurora enabled (for example, SERDES port #4 for T4240/T4160 and port #2 for T2080) and the speed is set to 2.5 Gbit/s.

   NOTE

   Use the custom JTAG file to override the board RCW and enable Aurora interface for trace collection. Use it as reference to build your own RCW if you need some other functionality on board that is not enabled by this RCW.

2. Ensure that the following DIP switches on the board are configured for Aurora lanes to be routed to Aurora connector on the board:

   To train Aurora on the T4240/4160QDS target:

   - `SW4.7=0 SW4.8=0` (`SW_SD4=00 -> 100MHz` reference clock)
   - `SW9.2=0 SW9.3=0` (`SW_SD4_MX_CTL=00-> SD4 Lanes 4-7` muxed to SATA/Aurora)

   To train Aurora on the T2080QDS target:

   - `SW4.[5:6]=11`[Default] `SW4.[7:8]=11`[Default] (SERDES2 Clock1 and SERDES2 Clock2 -> 100 MHz)
   - `SW8.[4:5]=11` (SERDES2 Protocol muxed to Aurora)
   - RCW bit 180 must be set. This is for setting the clock divider to configure Aurora to run at 2.5 GHz.

3. Run the TCL script from the **CCS** console window, the **Debugger Shell** window, or an initialization file.
   - From the **CCS** console window:
     1. Make sure you have a proper Command Converter started and configured.
     2. In console, type: `source {$path}/train_aurora_t4.tcl`, where `$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 {$path}/train_aurora_t4.tcl`, where `$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.
   - From initialization file:
     1. Copy `train_aurora_t4.tcl` script to your project in the **CFG** folder.
     2. Add the following line to the start of your initialization file:

```
source [file dirname [info script]]/train_aurora_t4.tcl
```

3. Add a call to `train_aurora_dbg` procedure in the `init_platform` procedure:

```
train_aurora_dbg
```

2.4.2.3 On B4860QDS target

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

```
PA/PA_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 Gbit/s. 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 2.5 Gbit/s.

     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, see *CodeWarrior Common Features Guide.pdf* and *Targeting_PA_Processors.pdf* in the folder: *<CWInstallDir>\PA\Help\PDF*, where *<CWInstallDir>* is the directory where CodeWarrior for Power Architecture V10.x is installed. You can also see the cheatsheet *Burning U-Boot to Flash* available from **Help > Cheat Sheets > CodeWarrior for Power Architecture V10.x**.
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 deselected 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.

2.5 Importing trace data file

This feature allows you to import trace data from an existing project into another project and view the data in the **Trace** viewer.

To import trace data:

1. From the IDE menu bar, choose **File > Import**.

   The **Import** wizard appears.

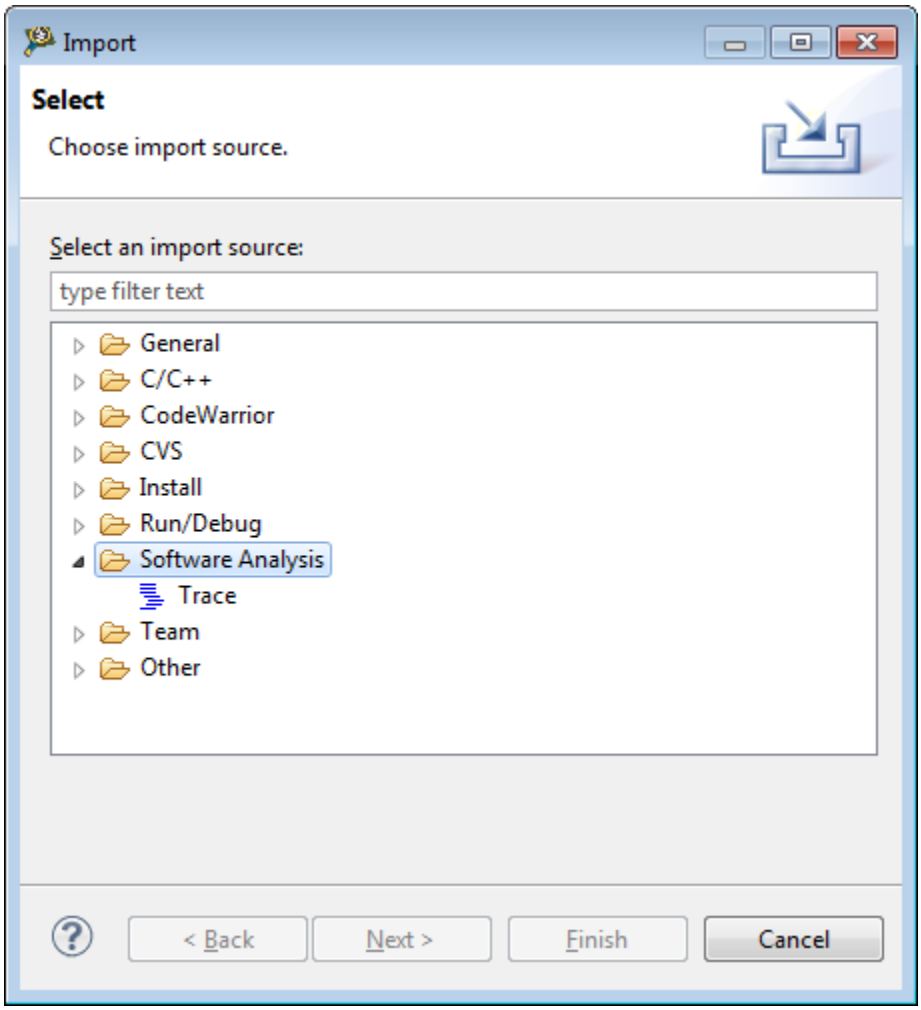


Figure 2-61. Import Wizard

2. Expand the **Software Analysis** tree node and choose **Trace**.
3. Click **Next**.

   The **Import Trace** page appears.

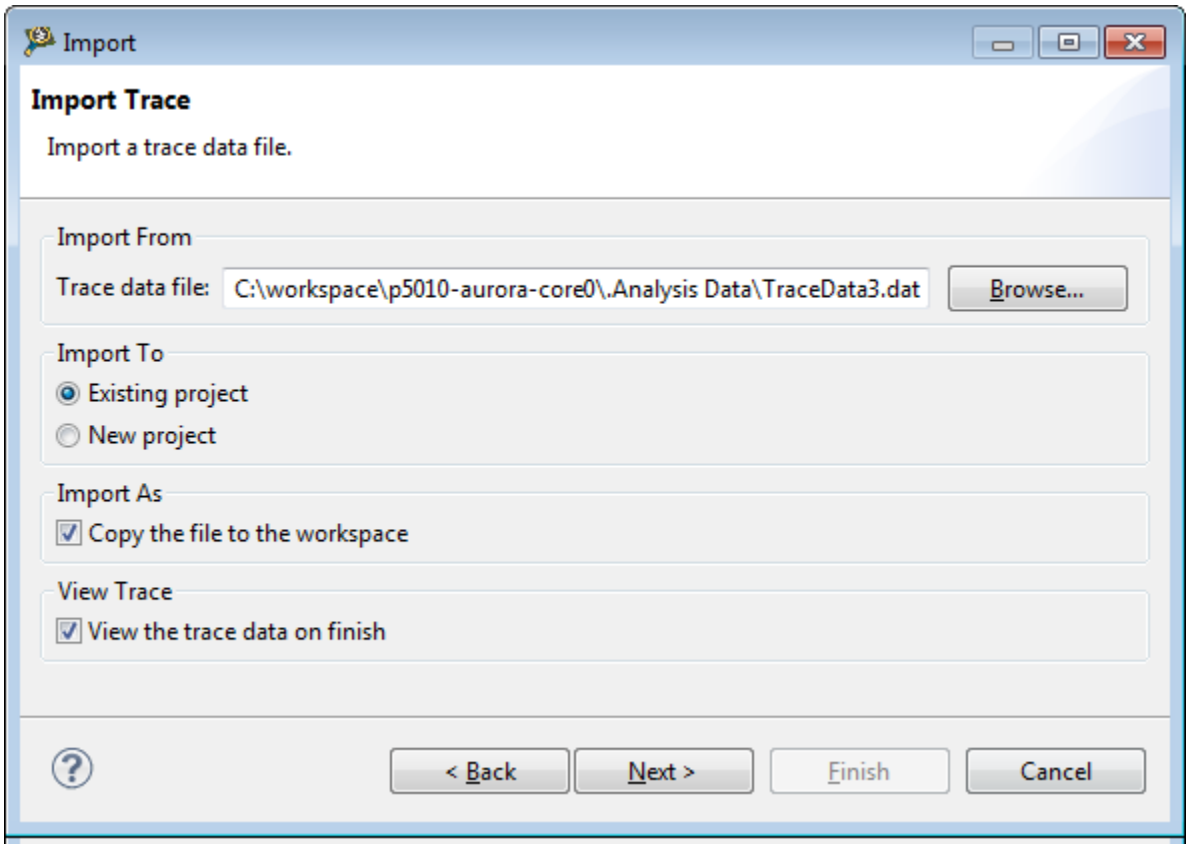


Figure 2-62. Import Trace page

4. Click **Browse** next to the **Trace data file** field to select the trace data file to be imported. The trace data file is located in the `.Analysis Data` folder of the project in the workspace.
5. Select one of the following options:
   - **Existing project**: Imports the trace data file to an existing project present in your current workspace. Selecting this option opens the **Select an Existing Project** page, as shown in the following figure, where you can select the project you wish to import the trace data file to.
   - **New project**: Imports the trace data file to a new project.
6. Select the **Copy the file to the workspace** checkbox if you want to maintain a copy of the trace data file in your current workspace.
7. Select the **View the trace data on finish** checkbox to open the **Trace** viewer automatically after clicking the **Finish** button.
8. Click **Next**.

   The **Import Trace to Existing Project** page appears.

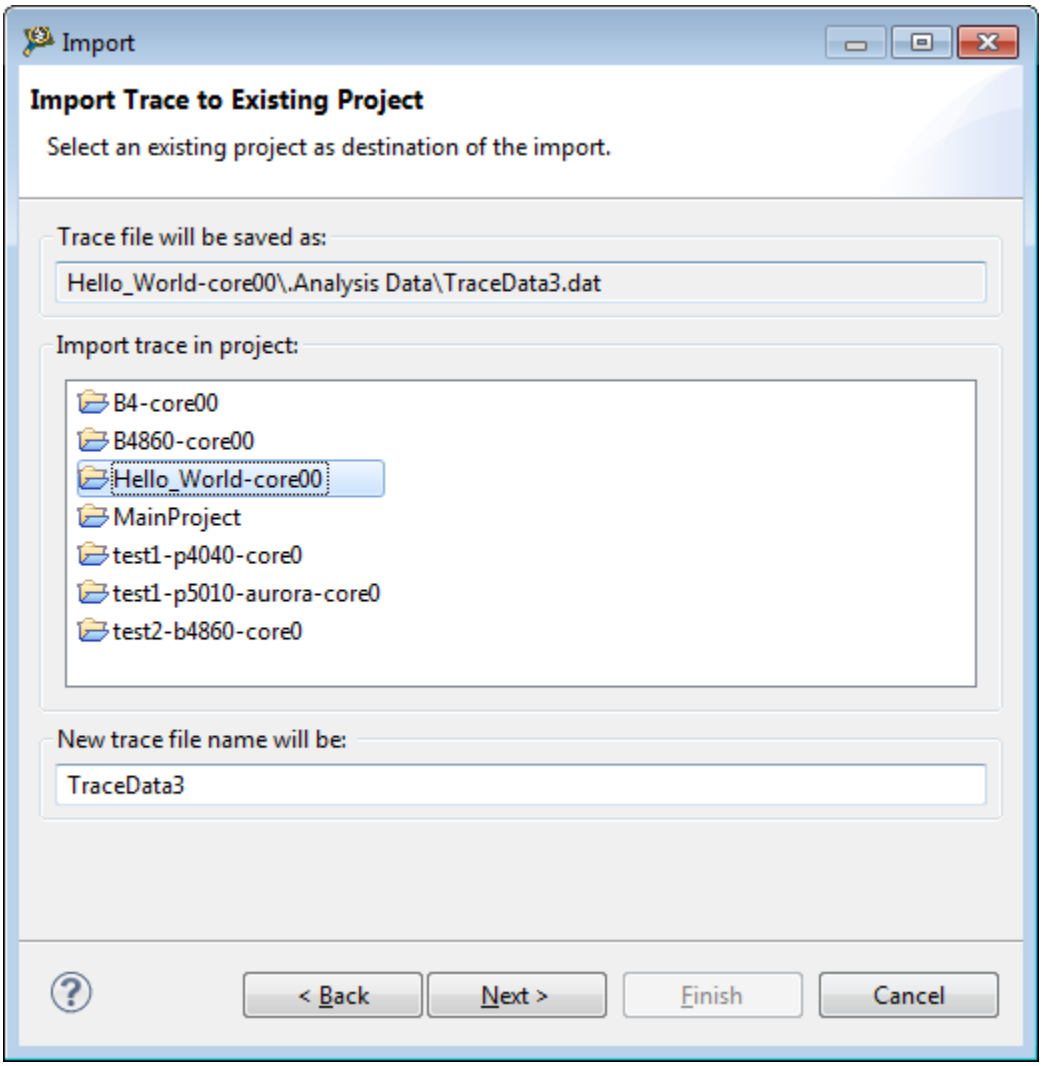


Figure 2-63. Import Trace to Existing Project page

9. Select the project in which you want to import the trace data, in the **Import trace in project** group.
10. Enter a new file name for the trace data file in the **New trace file name will be** text box. This field is optional.
11. Click **Next**.

    The **Import Trace Configuration** page appears.

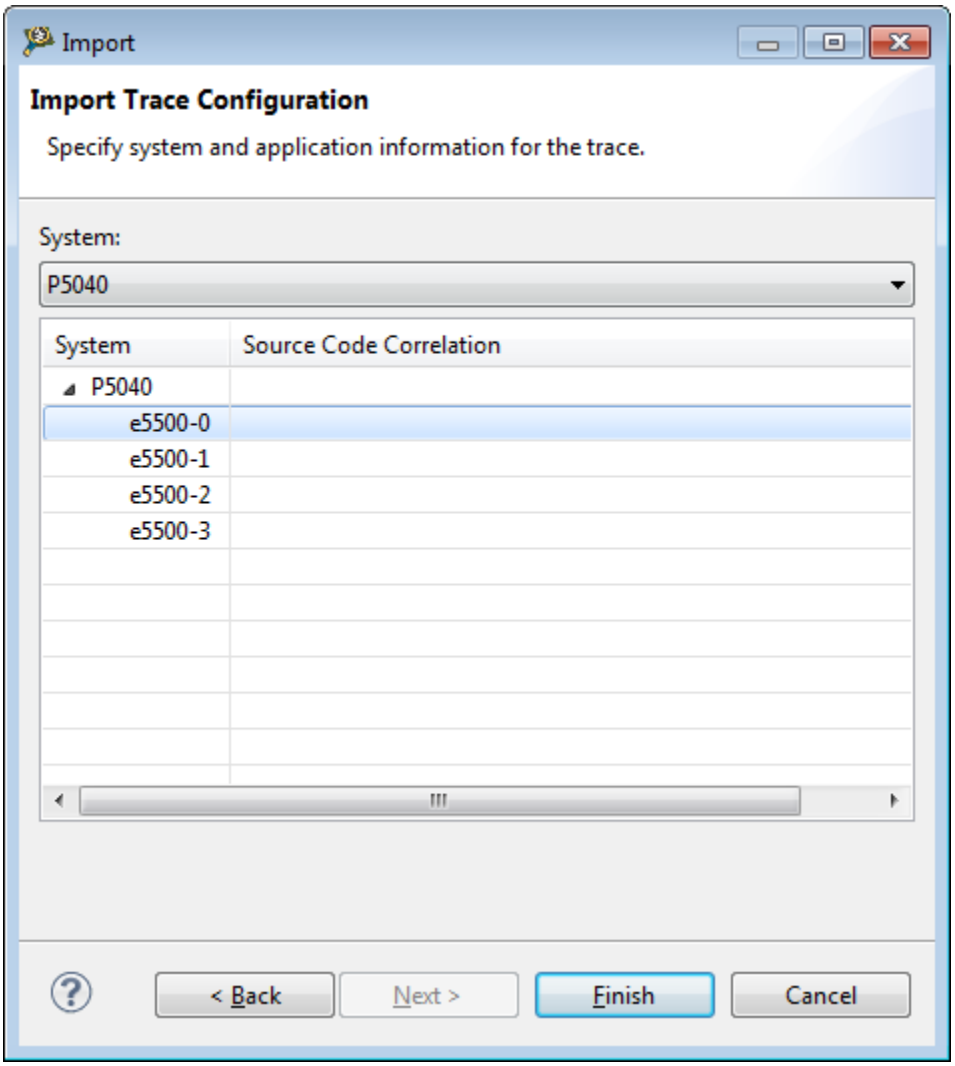


Figure 2-64. Import Trace Configuration page

12. Choose the desired hardware for the trace from the **System** pop-up menu.
13. Click **Finish**.

    The trace data file is imported to the `.Analysis Data` folder in your project and the **Trace** viewer appears with the imported trace data.

NOTE

Select the project folder in your workspace to access the `.Analysis Data` folder.

2.6 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 target hardware, create, and build a hardware 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:

2.6.1 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.

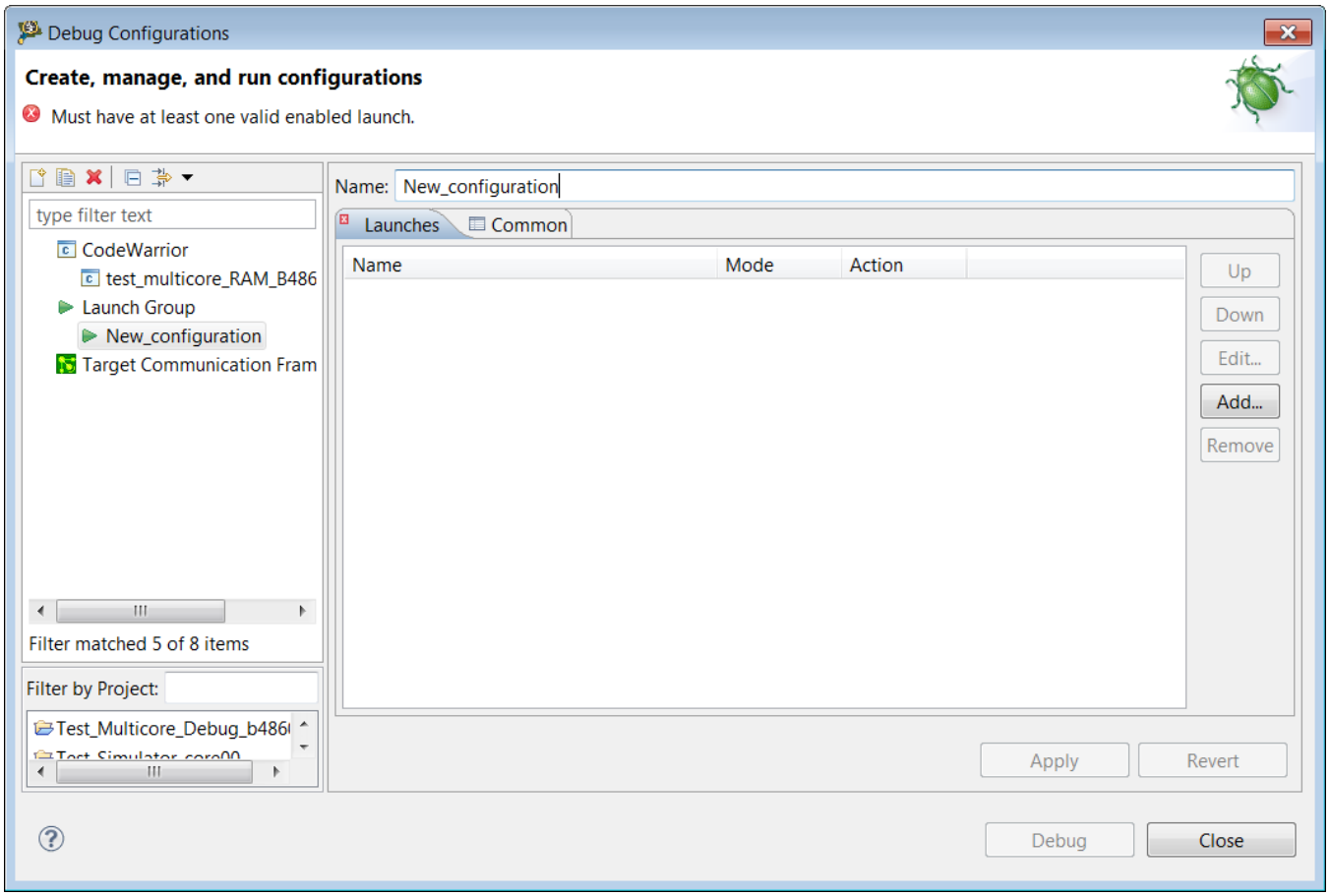


Figure 2-65. Creating new launch group dialog box

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 cores with shift key pressed.

5. Click **OK**.

   The core gets added to the new launch group.

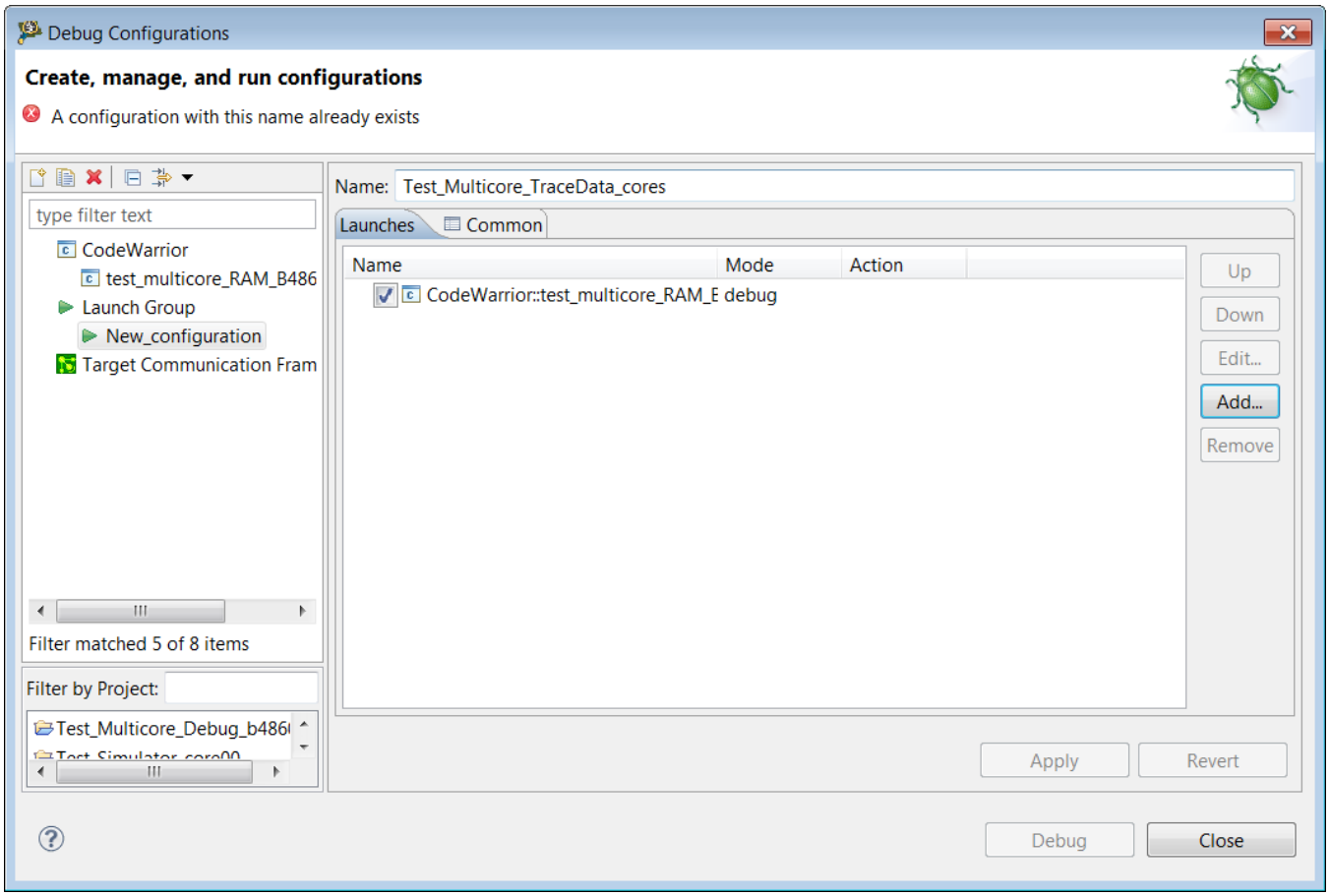


Figure 2-66. Cores added to new launch group

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()`.

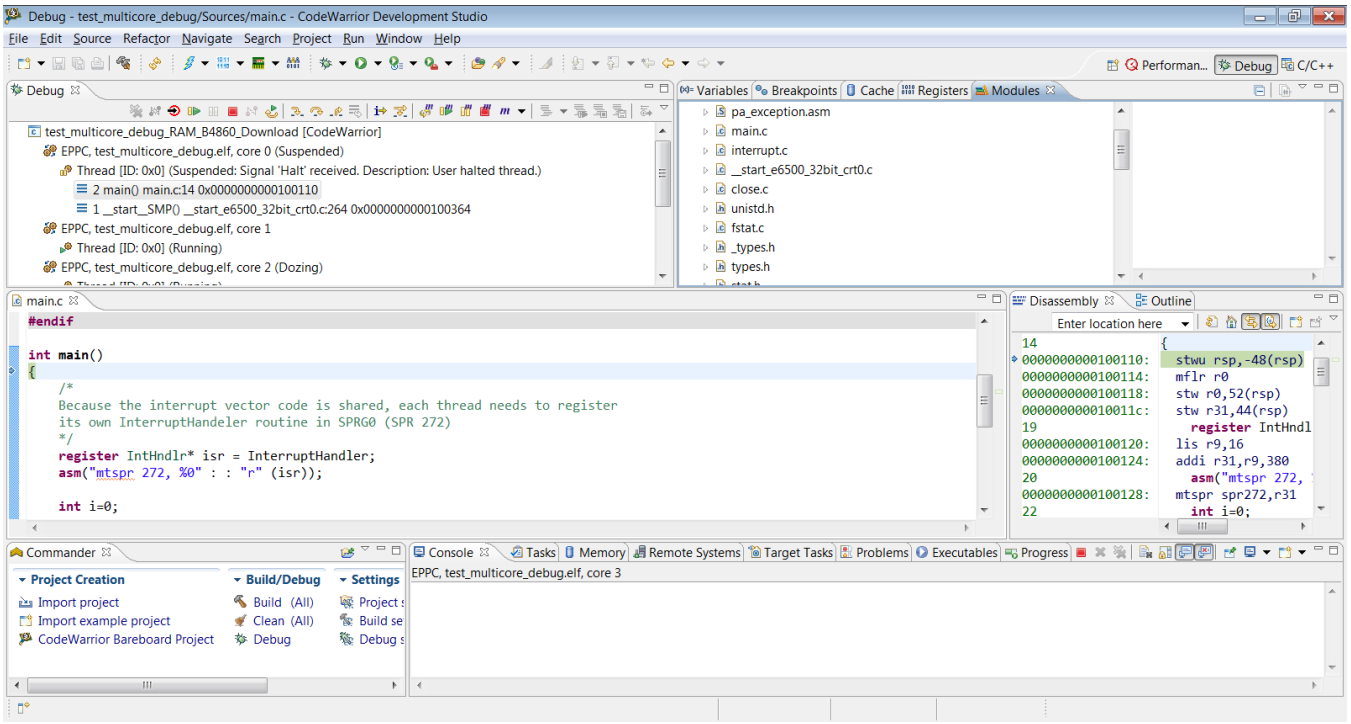


Figure 2-67. Debug perspective window

2.6.2 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 cores, click **Multicore Terminate** .

2.6.3 Viewing multicore trace data

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

In the Software Analysis view, the results are the leafs of a tree, where platform config name is the root and the parent of results is the data stream name.

The following figure shows an example of Software Analysis view:

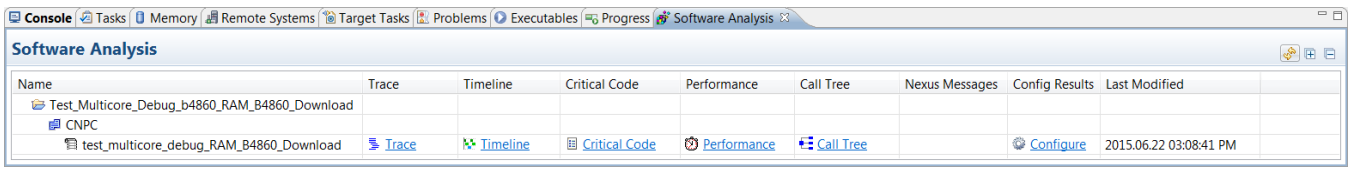


Figure 2-68. Software Analysis view

Click on the Timeline link to view the timeline data.

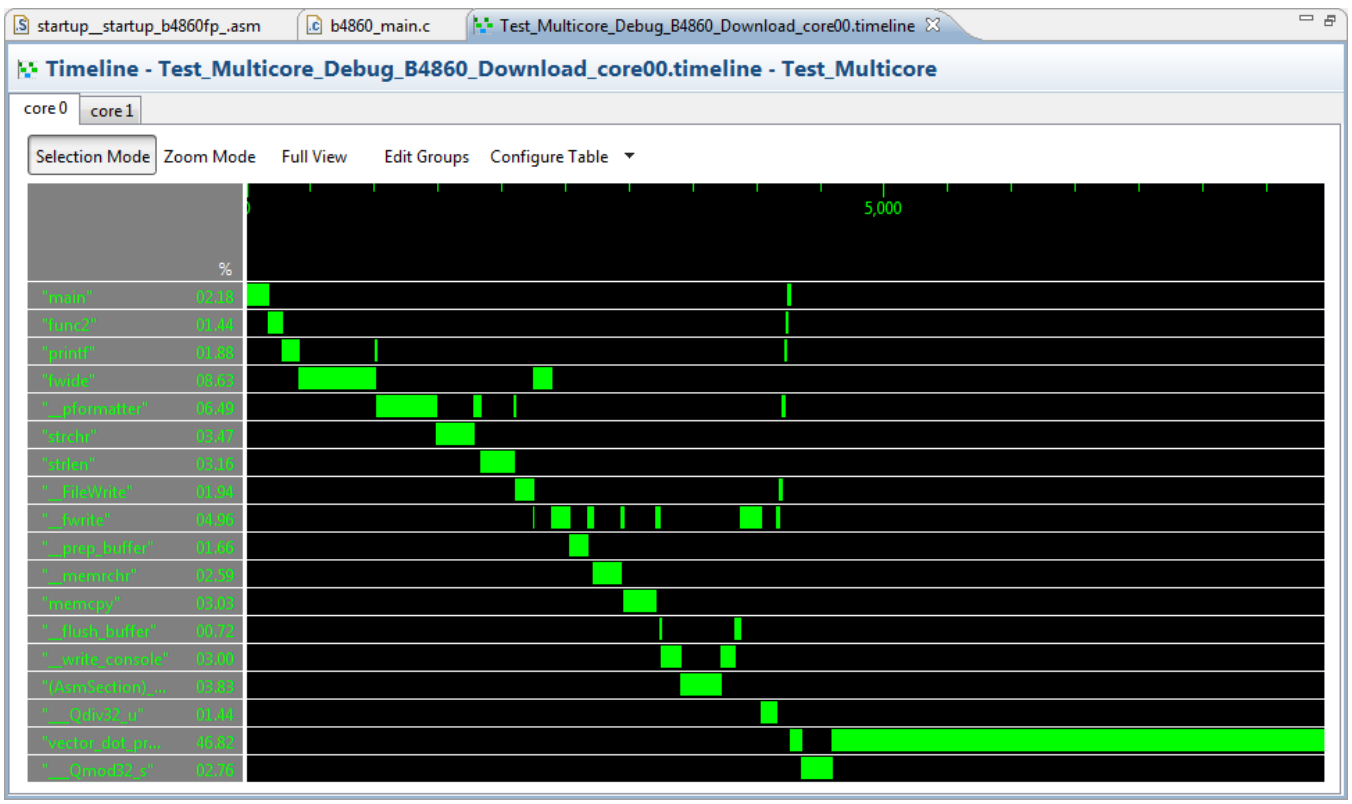


Figure 2-69. Timeline view

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

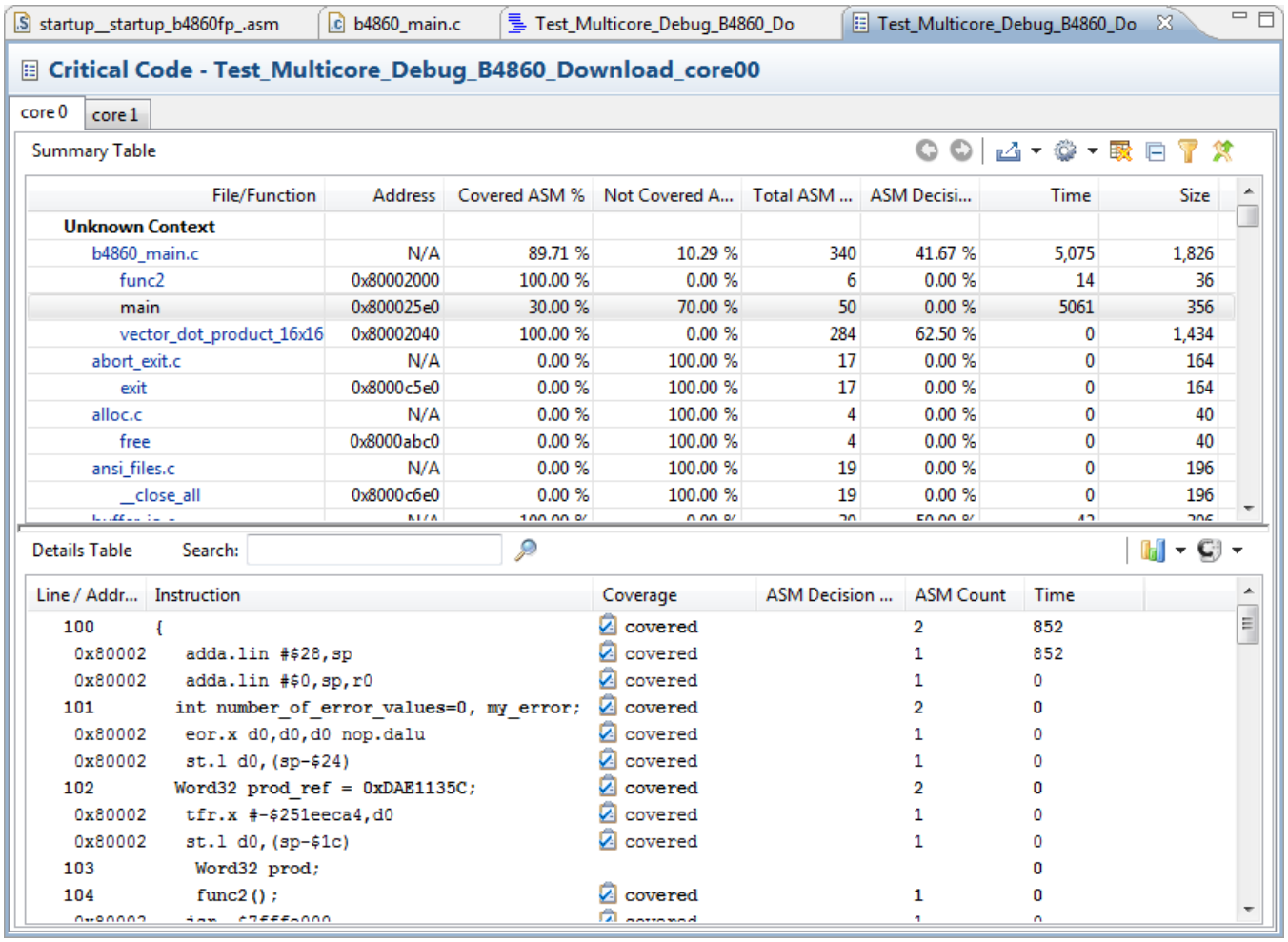


Figure 2-70. Critical Code view

Click on the Performance link to view the Performance data.

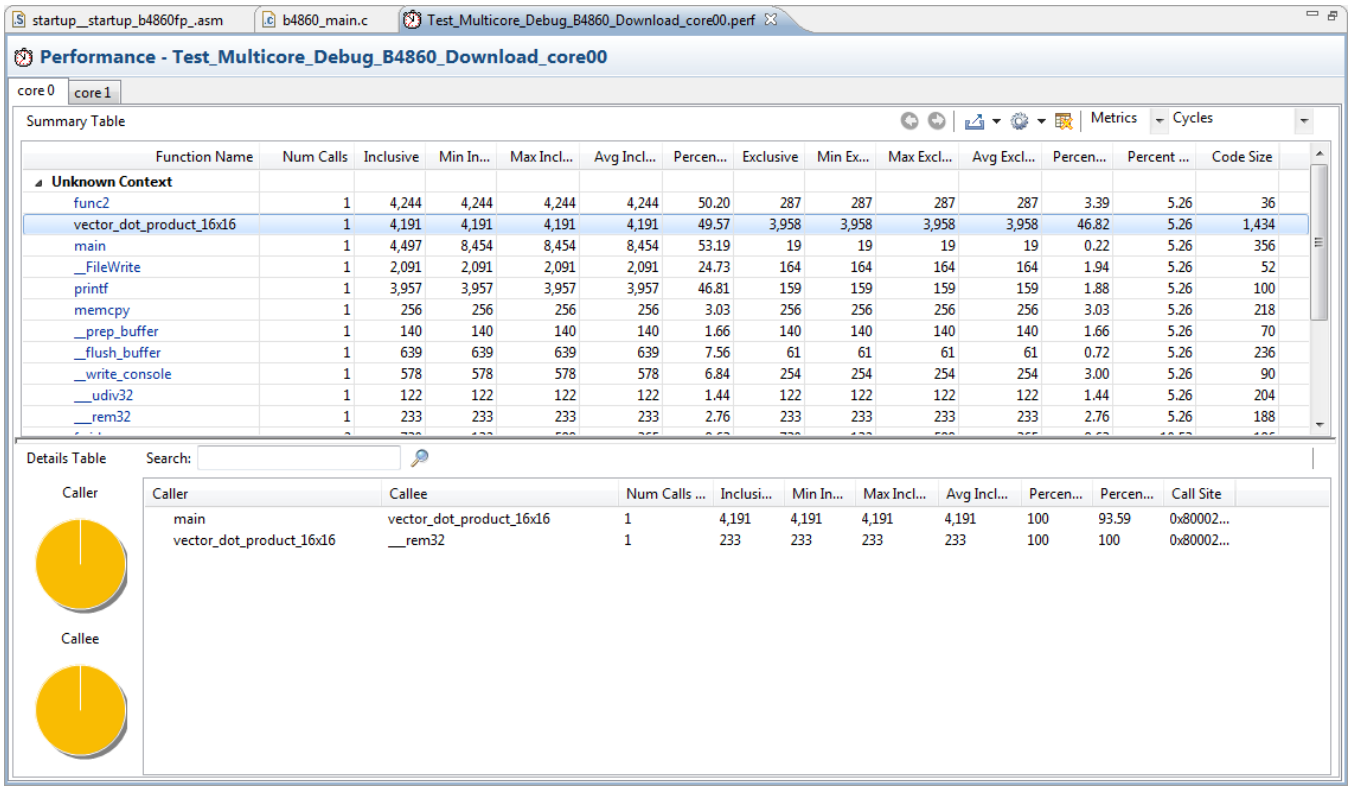


Figure 2-71. Performance view

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.

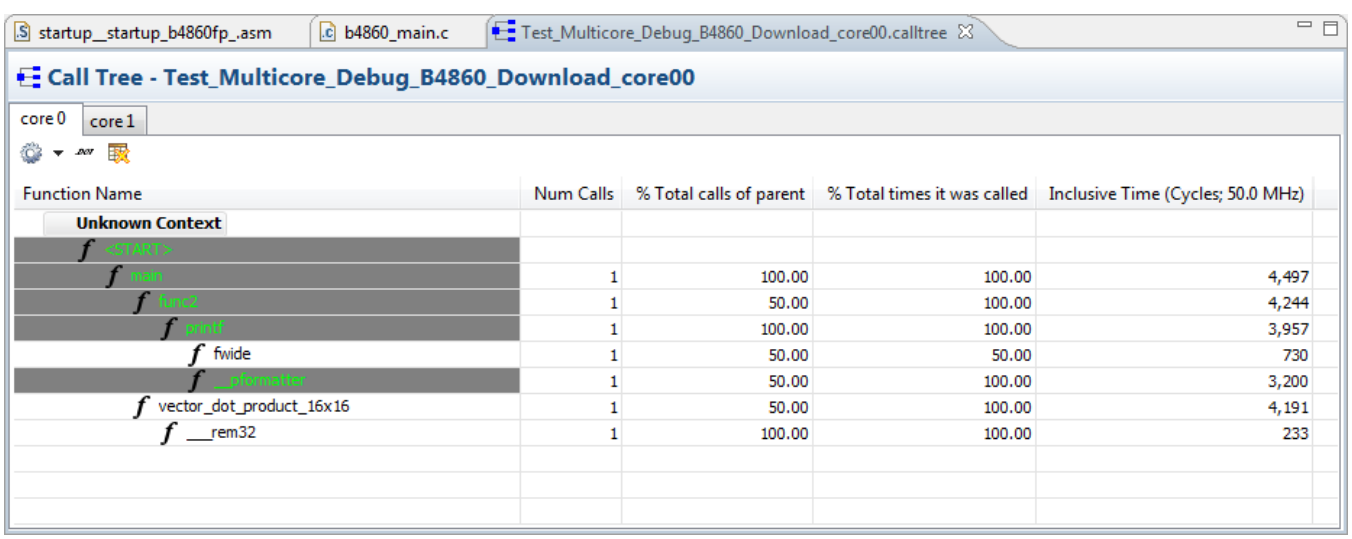


Figure 2-72. Call Tree view

2.6.4 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.
   - **Existing project**: Imports the trace data file to an existing project present in your current workspace. Selecting this option opens the **Select an Existing Project** page, where you can select the project you wish to import the trace data file to.
   - **New project**: Imports the trace data file to a new 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. Click **Finish**.
13. The trace viewer will display the imported multicore data.

2.6.5 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.
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.

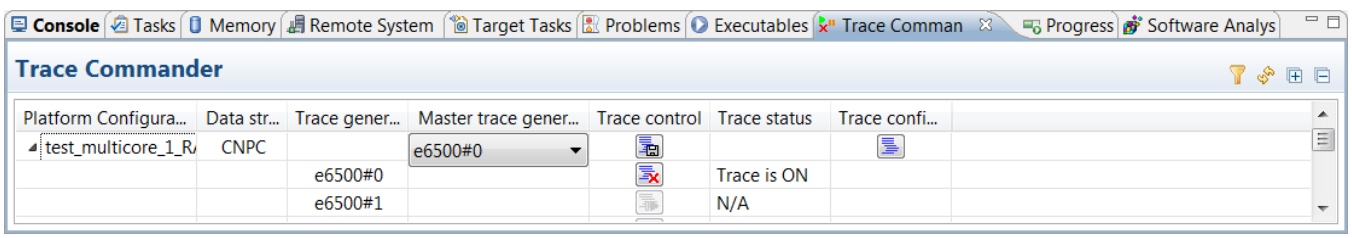


Figure 2-73. Trace Commander View

NOTE

When you click on Trace Commander option, a dialog box is displayed "*Multicore trace control settings should be handled manually. Automatic upload of multicore trace does not guarantee that trace will be collected correctly*". Click OK to display the Trace Commander view.

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

Table 2-32. 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.

Chapter 3
Tracepoints

Setting tracepoints limits tracing to certain blocks of a program instead of tracing everything and filtering the unnecessary parts later. This decreases the time required to upload and process. Additionally, setting tracepoints reduces the storage space requirements.

Based on the available resources, a tracepoint may be a hardware or a software tracepoint which can be determined at runtime. For each type of tracepoint, a different menu item is created.

- Hardware tracepoint - Uses hardware resources to control the trace collection, so no modification is required inside the project. You can set only two hardware tracepoints. If you try to install a third tracepoint, the application provides you with the options to remove an existing tracepoint or cancel the action.
- Software tracepoint - Provides a way to increase the number of tracepoints that can be used in an application using interrupts. You can set multiple software tracepoints. The hardware resources required for tracepoints on the Power Architecture platform are limited.

NOTE

At present, the P4080DS development board supports only software tracepoints, and the targets supported by the e6500 core support only hardware tracepoints.

This chapter explains the following topics:

3.1 Setting software tracepoints

You can set points for trace to start and stop in the source code. 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 solution uses a runtime patching mechanism. However, to reconfigure the software tracepoints, the running application needs to be re-compiled. You may combine tracing and debugging easily.

Each tracepoint is defined by the:

- virtual address where the trace should be started or stopped
- type of the tracepoint (software; action: start or stop)

The tracepoints can be set either in the Debug or C/C++ perspective.

Before setting tracepoints, you need to modify your project to use the software tracepoints. The following steps demonstrate the procedure of modifying a project to use the software tracepoints in a sample source code.

1. Navigate to the **CodeWarrior Projects** view.
2. Expand the CodeWarrior project tree control for which you want to set the tracepoints.
3. Modify `main.c`.
   a. From the project tree structure, expand the **Source** tree control.
   b. Double-click the source code file `main.c`.

      The source code file opens in the default editor area.

   c. Replace the contents of `main.c` with the contents of the following listing.
      Listing: Sample source code for setting software tracepoints

```
#include <stdio.h>
int main()

{

  int i=0;

//currently software trace points can't be used on change-of-flow
instructions

  i++;


  i += 100;


  unsigned long proc_id;

  asm ("mfpir %0" : "=r" (proc_id));
```

```
  printf("Core%d: Software trace points test!\r\n", proc_id>>5);

  i++;

  while (1) { printf("Core%d: Welcome to CodeWarrior!\r\n",
proc_id>>5); } // loop forever

}
```

d. Save the file.

4. Add `sa_tphandle.c` to the project.
   a. Right-click the `Source` folder in the **CodeWarrior Projects** view.
   b. Choose **Add Files** from the shortcut menu, and navigate to the location *<CWInstallDir>\PA\cdde\sasdk\samples\cw\sa_software_tracepoints\source* where *<CWInstallDir>* is the directory where CodeWarrior for Power Architecture V10.x is installed
   c. Select `sa_tphandle.c` in the **Open** dialog and click **Open**.

      The **File Operation** dialog appears.

   d. Select the **Copy files** option and click **OK**.

      The `sa_tphandle.c` file is added to the project.

5. Open the `pa_exception.asm` file present in the `Source` folder in the **Editor** view.
6. Overwrite `pa_exception.asm` with `pa_exception.asm` located at *<CWInstallDir>\PA\cdde\sasdk\samples\cw\sa_software_tracepoints\source*, and save the file.
7. Edit the linker control file.
   a. Expand the **LCF** folder of your project, and open the `gcc-eabi_core0.lcf` file in the **Editor** view.
   b. Add the `_swtp_addr` section to the file as follows.

```
_swtp_addr = 0x00100000;/*reserve the memory needed */

_swtp_end = 0x0010ffff;/*software tracepoints table*/

.....

.....

.sw_hashtable 0x00100000 :

{

*(.sw_hashtable)

} = 0x0010ffff

.....

.....
```

The added section is highlighted in the following figure.

c. Save the file.

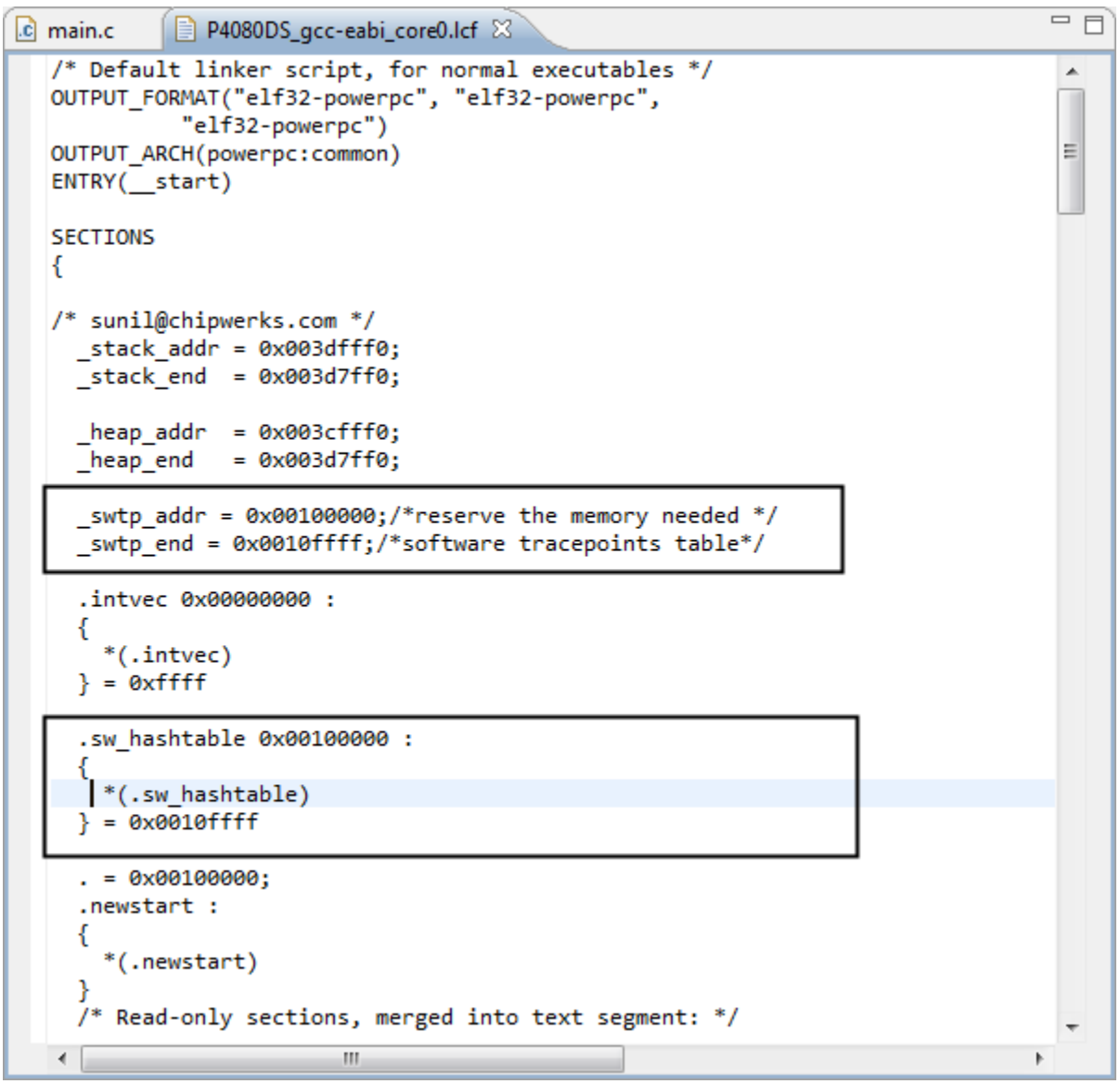


Figure 3-1. Adding _swtp_addr section to linker file

8. Update the `sa_tphandle.c` file with the number of software tracepoints used.
   a. Open `sa_tphandle.c` in the **Editor** view.
   b. Update the macro `TRACEPOINTS_NUMBER` with the number of software tracepoints used. For example, if you are using two tracepoints, modify the macro as follows:

```
#define TRACEPOINTS_NUMBER 0x2//number of trace points used by project
```

   c. Save the file.
9. Right-click on your project in the **CodeWarrior Projects** view and choose **Properties**.

The **Properties** dialog appears.

10. Select the **C/C++ Build** node and click the **References** tab.
11. Deselect the **Build referenced projects and build configurations.....** checkbox.
12. Click **Apply** and close the dialog.
13. Build your project.
14. Set tracepoints in the source code.
    a. Navigate to the **Editor** view and select the statement `i++;` (after `int i = 0;`).
    b. Right-click on the marker bar, and choose **Toggle Trace Start Point > Software Trace Point** from the shortcut menu.

       The start tracepoint icon appears on the marker bar.

       NOTE

       Avoid setting tracepoints on the lines containing only comments, parenthesis, and variable declaration with no value. Do not set software tracepoints on a call, return or branch.

    c. Select the statement `i++;` (after `printf()`) and right-click on the marker bar.
    d. Choose **Toggle Trace Stop Point > Software Trace Point** from the shortcut menu.

       The stop tracepoint icon appears on the marker bar.

       NOTE

       Before setting the tracepoints in the Debug perspective, ensure that the project is debugged and the source code file is open in the **Editor** view of the Debug perspective. For information on creating and debugging a CodeWarrior project, see the Configuration and Collecting data topics.

       NOTE

       To disable the start or stop tracepoints, navigate to the corresponding source code line, right-click on the marker bar and from the shortcut menu, choose **Toggle Trace Start Point** or **Toggle Trace Stop Point**.

15. Add Debug Configuration Control and Status Register (DCSR) address space to your project. This is necessary if you are using the non-cacheable launch configuration.
    a. Open the **Debug Configurations** dialog and select your project in the left pane.
    b. Click **Edit** in the **Main** tab.
    c. Click **Edit** in the **Properties** dialog.

d. Select the **Initialize target script** column against core 0 in the **Initialization** tab, as shown in the following figure.

   The **Target Initialization File** dialog appears.

e. Click **File System** and browse to the location *<CWInstallDir\PA\cdde\sasdk\samples\cw\sa_software_tracepoints\CFG* in the **Open** dialog.
f. Select the `P4080DS_init_DCSR_core0.cfg` file and click **Open**.
g. Click **OK** in the **Target Initialization File** dialog.

   The file gets added to the **Initialize target script** column.

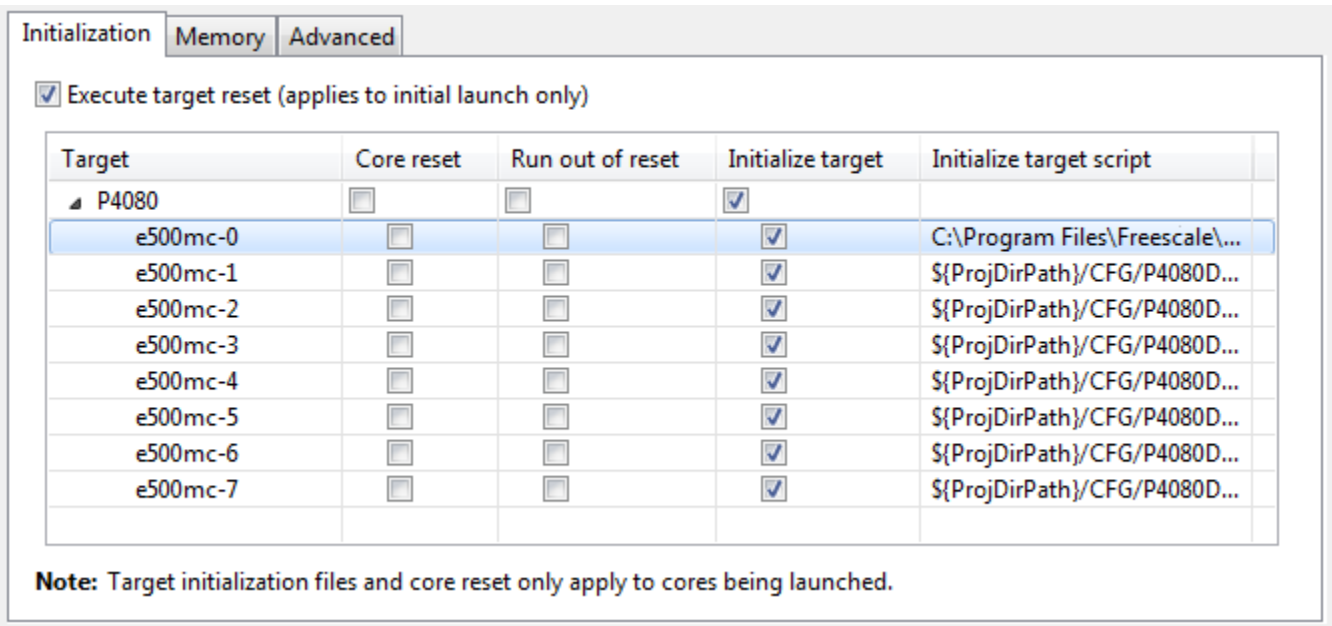


Figure 3-2. Adding Initialize target script

h. Click **OK** to close the **Properties** dialog.

16. Turn on tracing in the **Debug Configurations** dialog.
17. Click **Debug**.
18. Click **Resume**.

    The CodeWarrior starts collecting trace when a start tracepoint occurs and stops collecting trace when a stop tracepoint occurs.

19. Click **Suspend** to suspend the application.

NOTE

CodeWarrior can handle any number of start and stop tracepoints, which can be reached any number of times during the program execution.

3.2 Setting hardware tracepoints

Hardware tracepoints are supported only on the e6500 core, and you can set hardware tracepoints to collect **Full Program** and **Light Program** trace only. Also, you cannot modify the trace configuration of the hardware tracepoints during debug session. A message box appears prompting you to restart the debug session in case you want to apply the changes.

You can set the hardware tracepoints with the following scenarios:

- If you set only one stop hardware tracepoint, the trace data starts collecting from the beginning of the application and stops when execution reaches the stop hardware tracepoint.
- If you set one start and one stop hardware tracepoint, the trace data will be collected between these tracepoints.
- If you set one start hardware tracepoint, the trace data starts collection from that tracepoint and stops when the debug session ends.

To set hardware tracepoints:

1. Open `main.c` in the **Editor** view, and select the statement at which you want to set the start hardware tracepoint.
2. Right-click on the marker bar, and choose **Toggle Trace Start Point > Hardware Trace Point** from the shortcut menu.

   The start tracepoint icon appears on the marker bar.

3. Select the statement at which you want to set the stop hardware tracepoint.
4. Choose **Toggle Trace Stop Point > Hardware Trace Point** from the shortcut menu.

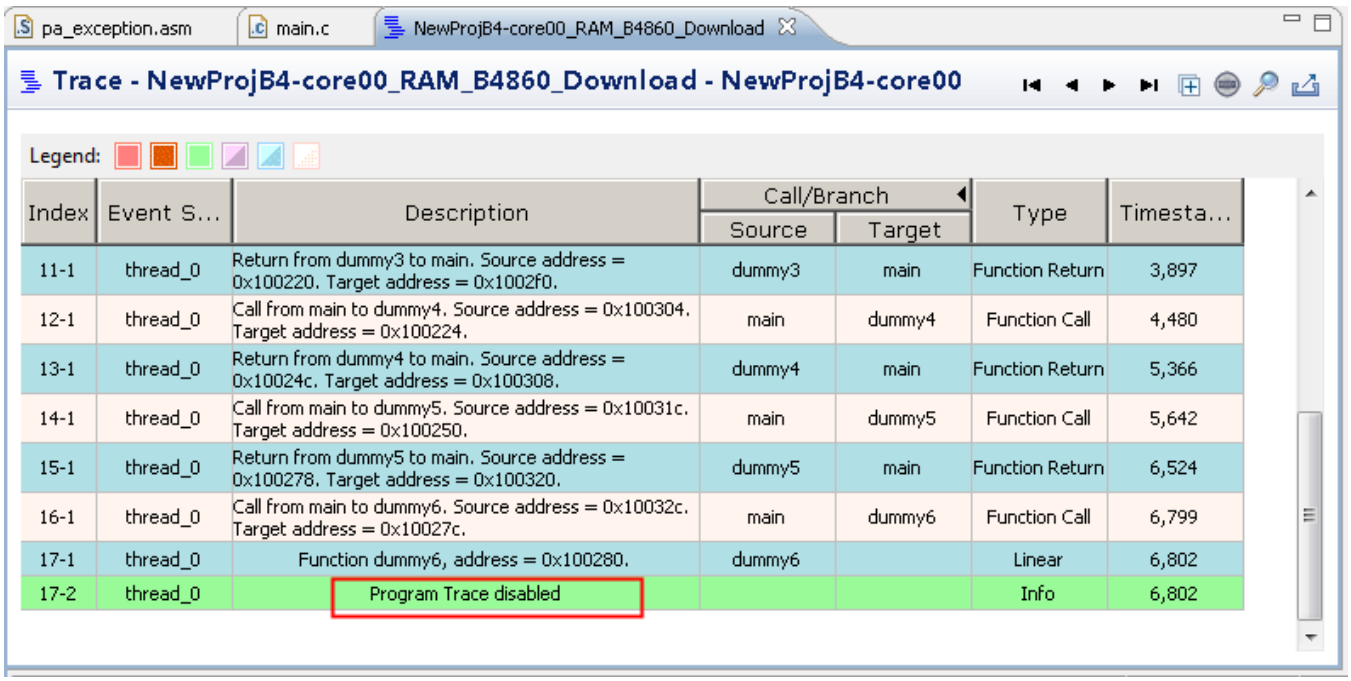
   The stop tracepoint icon appears on the marker bar.

5. Open the **Debug Configurations** dialog and select the **Trace and Profile** tab.
6. Select the required trace scenario, **Full Program Trace** or **Light Program Trace**.
7. Select the **Intermediate** tab, and select **Automatically** in the **Trace Control Settings** group.
8. Specify other trace settings in the **Intermediate** tab as required.
9. Click **Apply** and then **Debug**.
10. Click **Resume**.

    The CodeWarrior starts collecting trace when a start tracepoint occurs and stops collecting trace when a stop tracepoint occurs.

11. Click **Suspend** to suspend the application.
12. Open **Trace** viewer to view the trace results.

The **Trace** viewer displays the occurrence of stop hardware tracepoint using the "Program Trace Disabled" message in the **Description** column.



Index	Event S...	Description	Call/Branch Source	Call/Branch Target	Type	Timesta...
11-1	thread_0	Return from dummy3 to main. Source address = 0x100220. Target address = 0x1002f0.	dummy3	main	Function Return	3,897
12-1	thread_0	Call from main to dummy4. Source address = 0x100304. Target address = 0x100224.	main	dummy4	Function Call	4,480
13-1	thread_0	Return from dummy4 to main. Source address = 0x10024c. Target address = 0x100308.	dummy4	main	Function Return	5,366
14-1	thread_0	Call from main to dummy5. Source address = 0x10031c. Target address = 0x100250.	main	dummy5	Function Call	5,642
15-1	thread_0	Return from dummy5 to main. Source address = 0x100278. Target address = 0x100320.	dummy5	main	Function Return	6,524
16-1	thread_0	Call from main to dummy6. Source address = 0x10032c. Target address = 0x10027c.	main	dummy6	Function Call	6,799
17-1	thread_0	Function dummy6, address = 0x100280.	dummy6		Linear	6,802
17-2	thread_0	Program Trace disabled			Info	6,802

Figure 3-3. Trace viewer displaying occurrence of stop hardware tracepoint

3.3 Viewing analysispoints

You can view the list of all the existing analysispoints, that is tracepoints, in the **Analysispoints** view.

The **Analysispoints** view displays the attributes of the analysispoints set in source code or/and assembly code. You can turn on, turn off, or remove a analysispoint in this view.

To view analysispoints:

1. From the IDE menu bar, choose **Window > Show View > Other**.

   The **Show View** dialog appears.

2. Expand the **Software Analysis** tree control and choose **Analysispoints**.

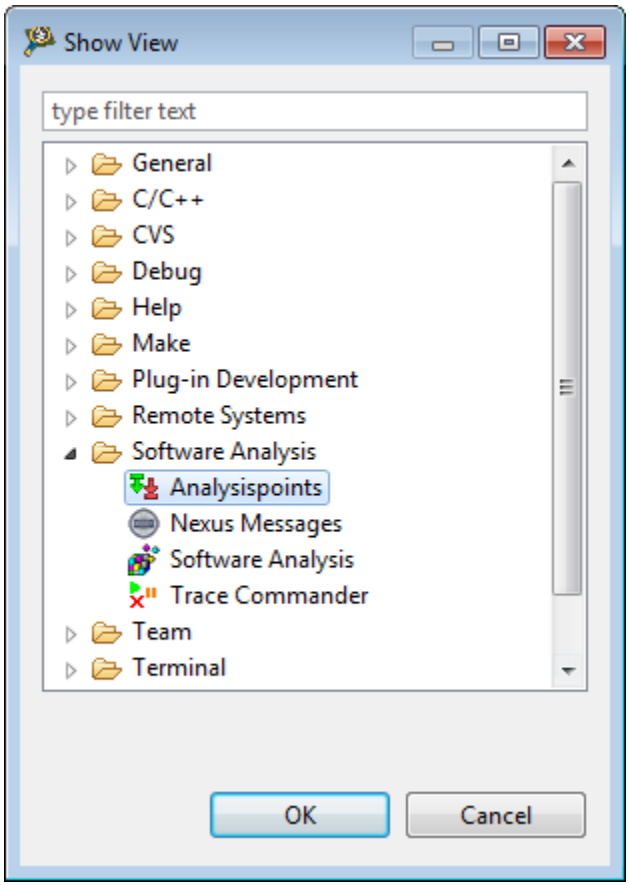


Figure 3-4. Show View dialog - Software Analysis options

3. Click **OK**.

   The **Analysispoints** view appears with a list of existing analysispoints.

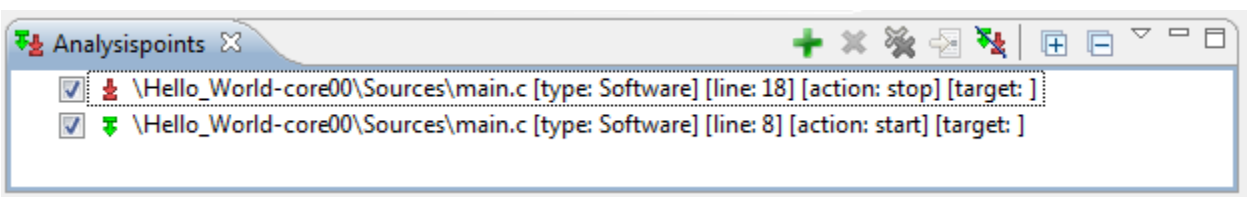


Figure 3-5. Analysispoints view

4. Select or deselect the checkbox next to the analysispoint if you want to turn on or turn off the associated analysispoint.

3.3.1 Analysispoints view

The Analysispoints view displays the attributes of the analysispoints (or tracepoints) set in source code or/and assembly code.

The **Analysispoints** view displays the following attributes of the analysispoints set from the source code:

- Path the file where analysispoint is set
- Type of the analysispoint, that is software or hardware
- Line number where analysispoint is set
- Action of the analysispoint, that is start or stop

The **Analysispoints** view displays the following attributes of the analysispoints set from the **Disassembly** view:

- Path of the file where analysispoint is set
- Type of the analysispoint, that is software or hardware
- Address where analysispoint is set
- Line number where analysispoint is set
- Action of the analysispoint, that is start or stop

You can use the **Analysispoints** view to perform the following actions:

- View full path of Analysispoint attribute
- Group analysispoints
- Define working set
- Add new analysispoint
- Enable/Disable analysispoints
- Navigate to analysispoint line
- Remove analysispoints
- Shortcut menu

3.3.1.1 View full path of Analysispoint attribute

You can view the complete path of the files of the analysispoint attribute in the Analysispoints view.

To view the complete path of the analysispoint attribute, click the pop-up menu in the **Analysispoints** view and choose **Show Full Paths**. Select the command again to hide the complete path.

3.3.1.2 Group analysispoints

You can group the attributes of the analysispoints by analysispoint type, files, projects, or working sets.

To group the attributes, click the pop-up menu in the **Analysispoints** view, click **Group By** and then choose the necessary option. The following figure shows the analysispoint attributes in the **Analysispoints** view which are grouped by projects. You can ungroup the attributes by choosing **Group By > Analysispoints**.

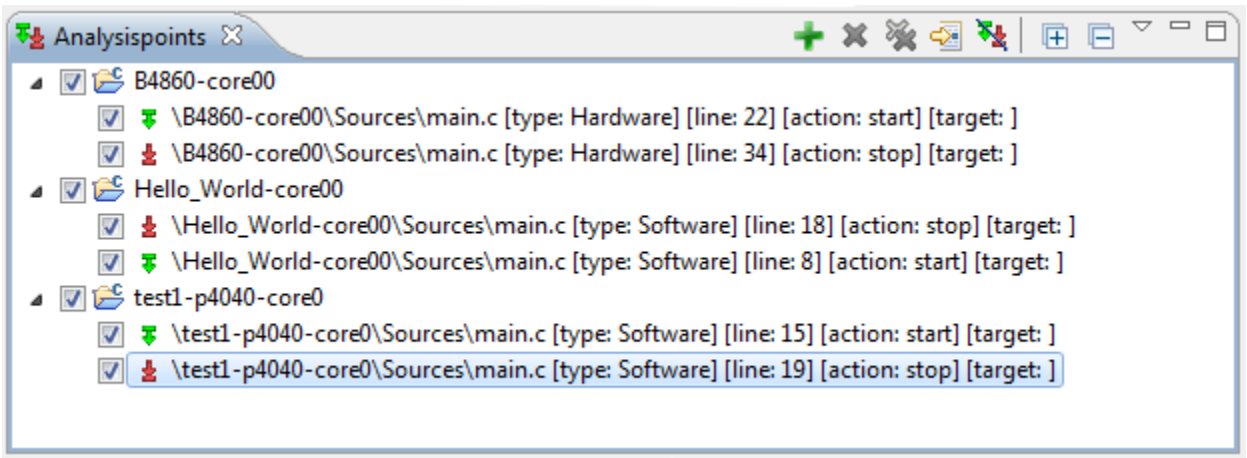


Figure 3-6. Analysispoints view - Group by projects

You can also specify nested groupings for the **Analysispoints** view. Click the pop-up menu and choose **Group By > Advanced**. The **Group Analysis Points** dialog appears. Move the **Analysispoint Types** and **Projects** groups from the **Available Groups** section to the **Selected Groups** section using the **Add** button. You can use the **Move Up** and **Move Down** buttons to customize the nesting of groups, that is to move a particular group up or down according to your choice.

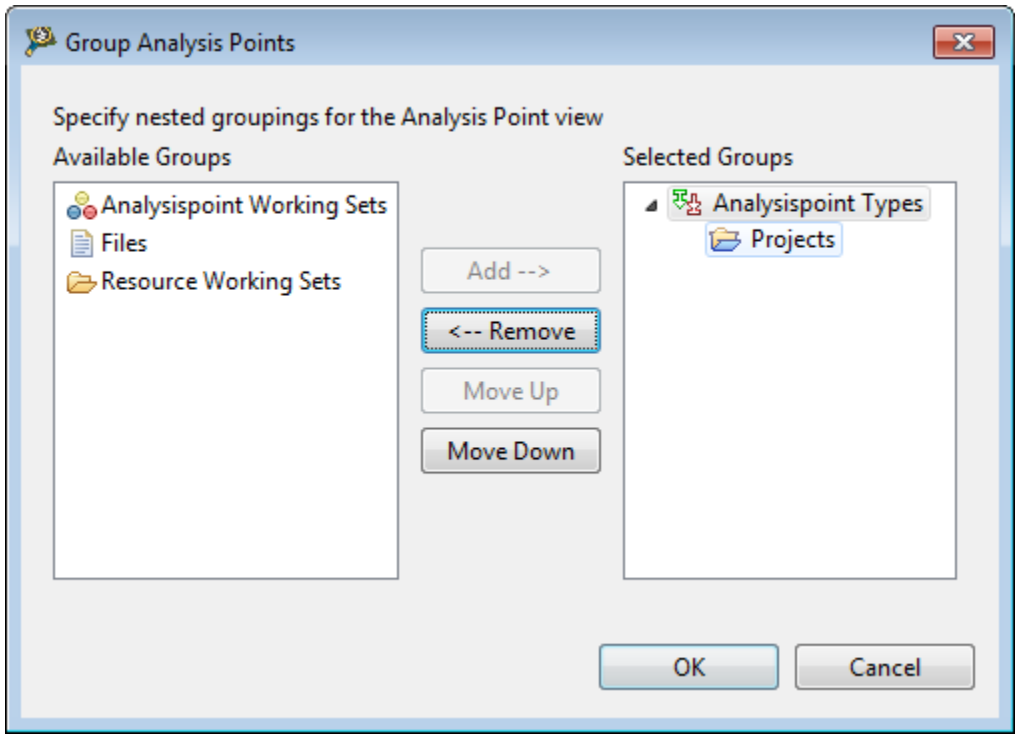


Figure 3-7. Group Analysis Points dialog

Click **OK** to view the nested grouped analysispoints in the **Analysispoints** view.

3.3.1.3 Define working set

Working sets allow you to create named groups for the analysispoints.

You can define working sets for the selected analysispoints and then group the analysispoints by working sets. To create a working set:

1. Click the pop-up menu in the **Analysispoints** view and choose **Working Sets**.

   The **Select Working Set** dialog appears.

2. Click **New** to create a new working set.

   The **New Working Set** dialog appears.

3. Enter a name for the working set in the **Working Set Name** textbox.
4. Select the analysispoints in the **Working Sets** list that you want to add to the new working set. For example, you can select analysispoints of core 0 for one working set and analysispoints of core 1 for another working set.
5. Click **Finish**.

   The new working set appears in the **Select Working Set** dialog.

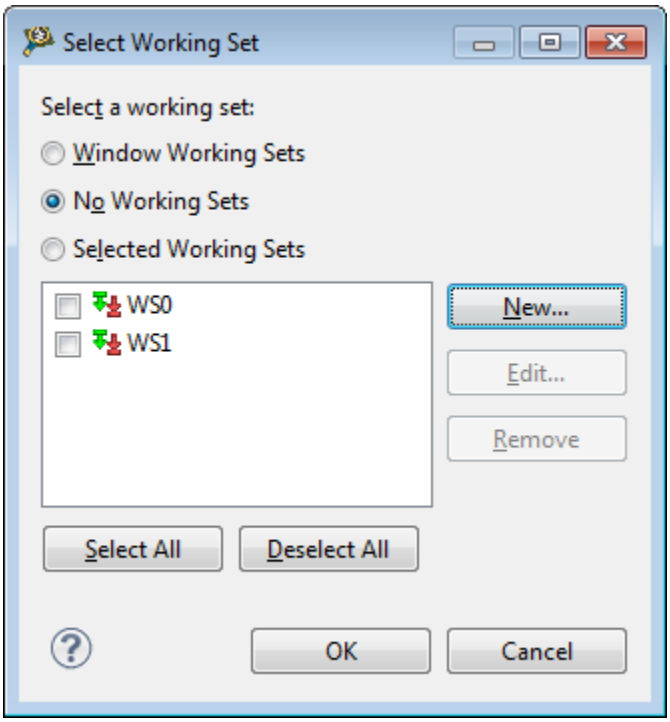


Figure 3-8. Select Working Set dialog

6. Click **OK** to close the dialog.

To group analysispoints by working sets, click the pop-up menu in the **Analysispoints** view and choose **Group By > Analysispoint Working Set**. The analysispoints grouped by the working sets appear in the **Analysispoints** view.

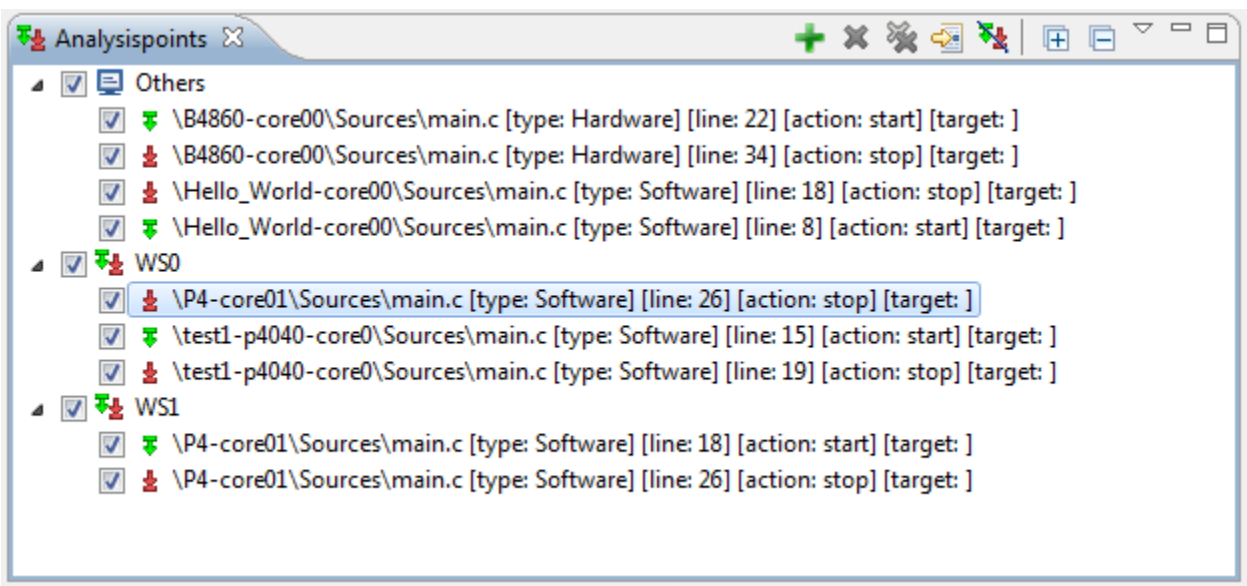


Figure 3-9. Analysispoints grouped by working sets

You can also set a working set as the default one so that whenever you add any stop or start analysispoint to your application, it gets automatically added to the default working set. In this way, working sets help you work on a particular set of analysispoints at a particular time. Click the pop-up menu in the **Analysispoints** view and choose **Select Default Working Set**. It opens the **Select Default Working Set** dialog where you can set one of the available analysispoints working sets as the default one. Alternatively, right-click on the working set, and choose **ToggleDefault.label** from the shortcut menu to set the working set as default.

You can deselect the working set which you set as default, choosing **Deselect Default Working Set** from the pop-up menu or choosing the **ToggleDefault.label** option from the shortcut menu.

3.3.1.4 Add new analysispoint

You can add an analysispoint on the address of an instruction using the **Analysispoints** view.

To add an address analysispoint:

1. Click the **Add a new Analysispoint** button, in the **Analysispoints** view, to display the **Add new analysispoint** dialog.

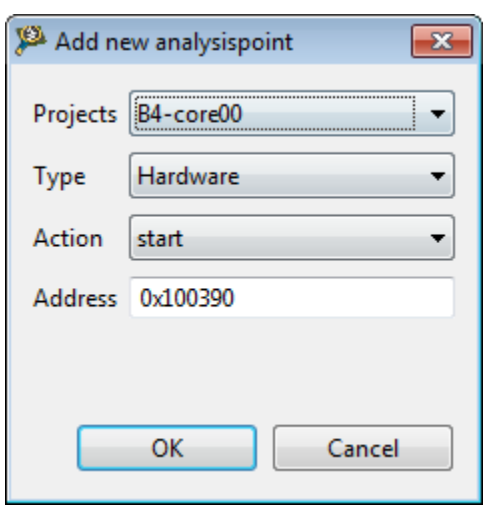


Figure 3-10. Add new analysispoint dialog

2. Choose the required project from the **Projects** pop-up menu.
3. Choose the type of the analysispoint from the **Type** pop-up menu.
4. Choose the action of the analysispoint, that is whether start or stop, from the **Action** pop-up menu.
5. Enter the address where you want to set the analysispoint in the **Address** text box.

6. Click **OK**.

   The analysispoint is set and appears in the **Analysispoints** view.

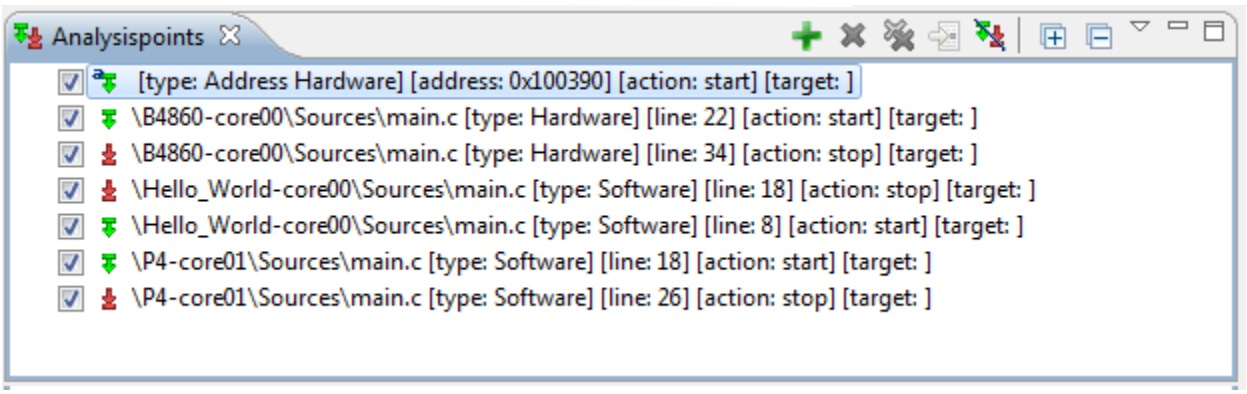


Figure 3-11. Address analysispoint set in Analysispoints view

3.3.1.5 Enable/Disable analysispoints

You can enable/disable an analysispoint from **Analysispoints** view by selecting/deselecting it.

The deselected attribute indicates the disabled analysispoint. A disabled analysispoint will have no effect during the collection of trace data.

You can also use the **Ignore all Analysispoints** icon to disable all the analysispoints without manually selecting them. Click **Ignore AllAnalysispoints** again to enable the analysispoints.

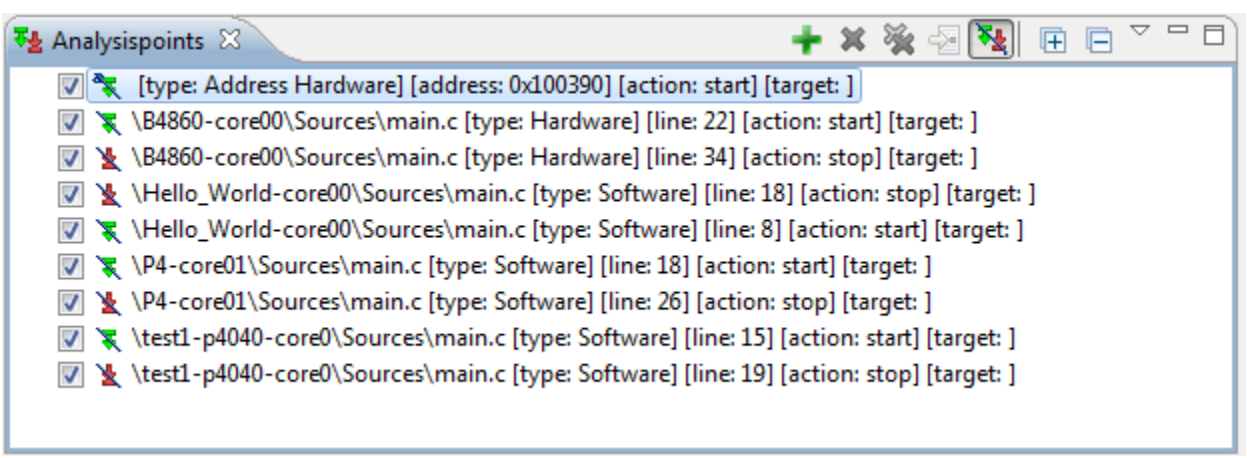


Figure 3-12. Disabling analysispoints from Analysispoints view

3.3.1.6 Navigate to analysispoint line

The **Analysispoints** view shows where start and stop analysispoints are set in the editor area or the **Disassembly** view.

It helps you navigate to the source code or assembly code where the analysispoint is set. If you want to go to the specific line of the source code where the analysispoint is set, select the analysispoint attribute in the **Analysispoints** view and click the **Go To File for Analysispoints** icon. This option will navigate you to that line.

3.3.1.7 Remove analysispoints

You can remove a particular analysispoint or all analysispoints from an application using the **Analysispoints** view.

To remove an analysispoint, select the analysispoint attribute and click the **Remove Selected Analysispoint** icon. To remove all the analysispoint, click the **Remove All Analysispoints** icon.

3.3.1.8 Shortcut menu

Right-click in the **Analysispoints** view or on the analysispoint to display a shortcut menu, which allows you to perform the following actions:

- Go to File - Navigates you to the specific line of the source file where you have set the selected analysispoint.
- Enable/Disable - Allows you to enable/disable analysispoints. A disabled analysispoint will have no effect during the collection of trace data.
- Remove - Removes the selected analysispoint.
- Remove All - Removes all analysispoints that are displayed in the **Analysispoints** view.
- Select All - Selects all analysispoints in the **Analysispoints** view.
- Copy - Copies the selected analysispoint on the clipboard.
- Set Cores - Lets you select the cores for which you want to set analysispoints. To choose the cores to set a analysispoint, select the analysispoint and right-click. Choose **Set cores** from the shortcut menu. **The Select Cores** dialog appears. By default, all cores are selected in the **Select Cores** dialog if the application is not in the Debug mode. For a selected analysispoint, you can select/deselect the required core to associate it with the analysispoint. If the application is in the Debug mode then depending on the thread selected in the **Debug** view, the **Select Cores** dialog will have the corresponding core selected for the analysispoint.

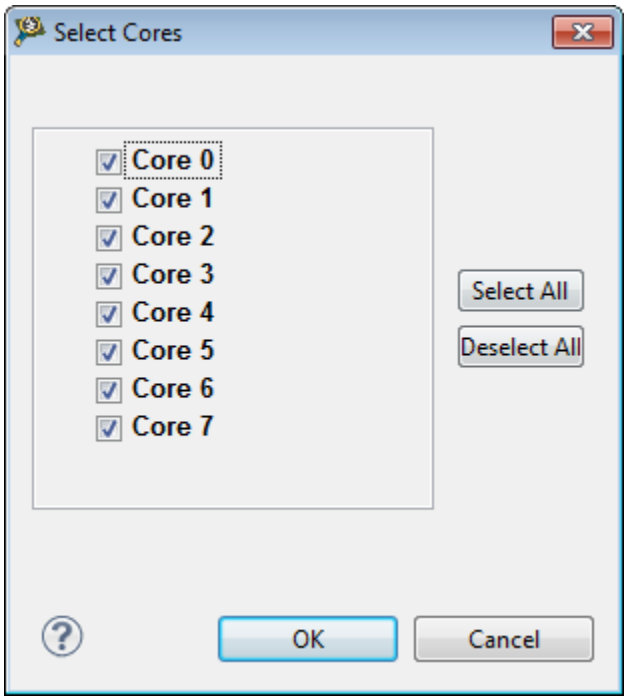


Figure 3-13. Select Cores dialog

NOTE

The default settings mark all the cores selected.

The following table describes the **Select Cores** dialog options.

Table 3-1. Select Cores options

Option	Description
Core 0 - Core 7	Select or deselect the desired checkboxes to install the analysispoint on the selected cores.
Select All	Click to select all the checkboxes. This installs the analysispoint on all cores.
Deselect All	Click to deselect all the checkboxes. The analysispoint is not installed on any core.

- Export Analysispoints - Lets you save the analysispoints in a `.apt` file at a desired location. Choose this option to display the **Export Analysis Points** dialog. Select the analysispoint(s) that you want to save, and click **Browse** to navigate to a location where you want to save the analysispoint(s), in the **Save Analysis Points As** dialog. Specify a filename in which the analysispoints will be exported. The full path will appear in the **Destination** text box. Click **Finish**.

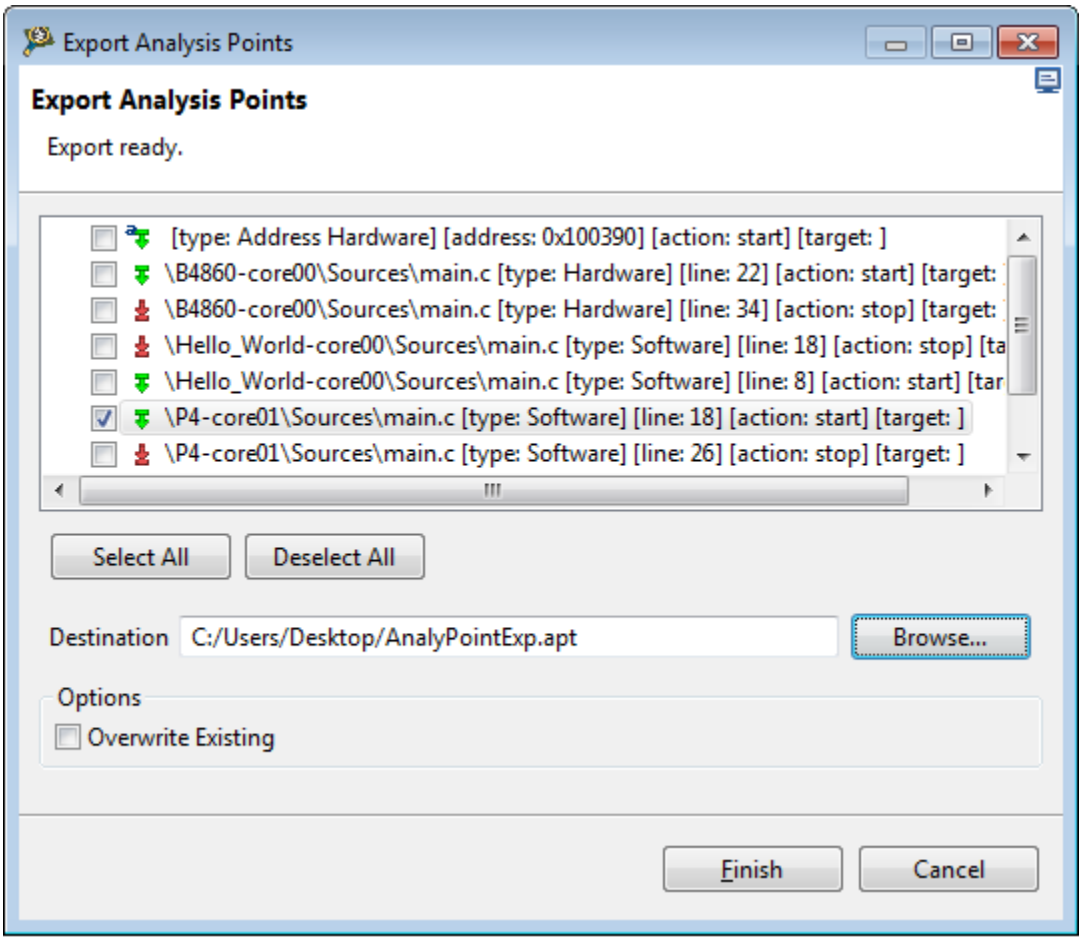


Figure 3-14. Export Analysis Points dialog

- Import Analysispoints - Lets you import analysispoints from another location in the **Analysispoints** view. Choose this option to display the **Import Analysis Points** dialog. Click **Browse** to locate the `.apt` file where you saved the analysispoints. Click **Finish**. The analysispoints will be imported in the **Analysispoints** view.

Chapter 4
Performance Analysis

The Performance Analysis tool allows you to configure and analyze scenarios in the QorIQ processors: P4080, P4040, P2040/41, P3041, P5010, P5040, P5020/21, B4860, G4860, B4460, B4420, T4240, and T4160. 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.

This chapter explains the following topics:

4.1 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
- Analysis Sessions view
- Progress view

4.1.1 Performance Analysis view

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

A project contains configuration information and the data that is collected on that configuration as shown in the following figure. See Creating new configuration to understand how to define a Performance Analysis configuration.

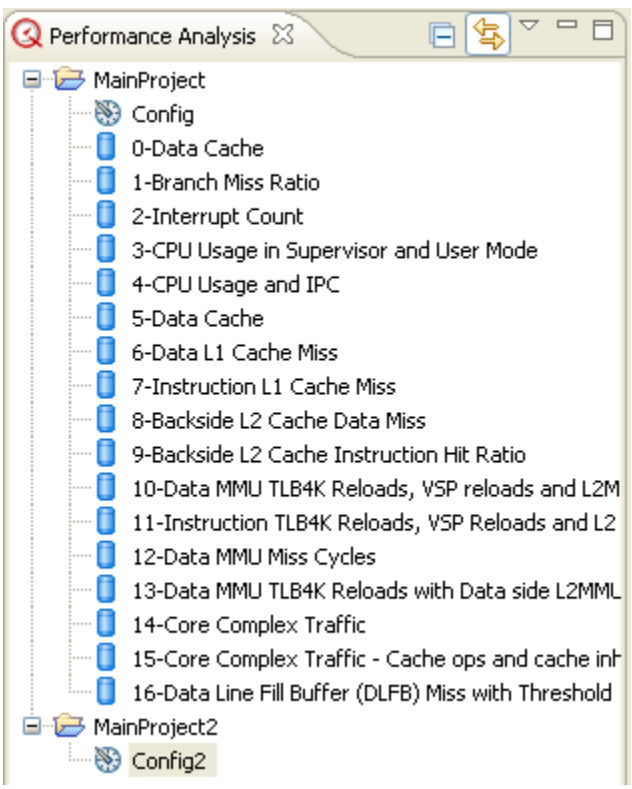


Figure 4-1. Performance Analysis view

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 choose **Import Projects** from the shortcut 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 choose **Export Projects** from the shortcut menu. In the **Export** wizard that appears, select the destination type and the directory where you want to export the project. Click **Finish**.

You can import a custom scenario into a performance analysis configuration. A custom scenario can be created by combining the events/metrics. The default (stock) scenarios are predefined and cannot be changed, but you can modify a custom scenario and import it into your configuration. To import a custom scenario, right-click the project and choose the **Import Custom Scenarios** option. The **Import Custom Scenarios** dialog appears. Browse through the directory where the custom scenario folder is available, and select the custom scenario. Click **OK** to close the dialog and get the custom scenario imported into your configuration.

4.1.2 Configuration view

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

This view appears after you define a new configuration for Performance Analysis. See Creating new configuration to understand how to define a new Performance Analysis configuration.

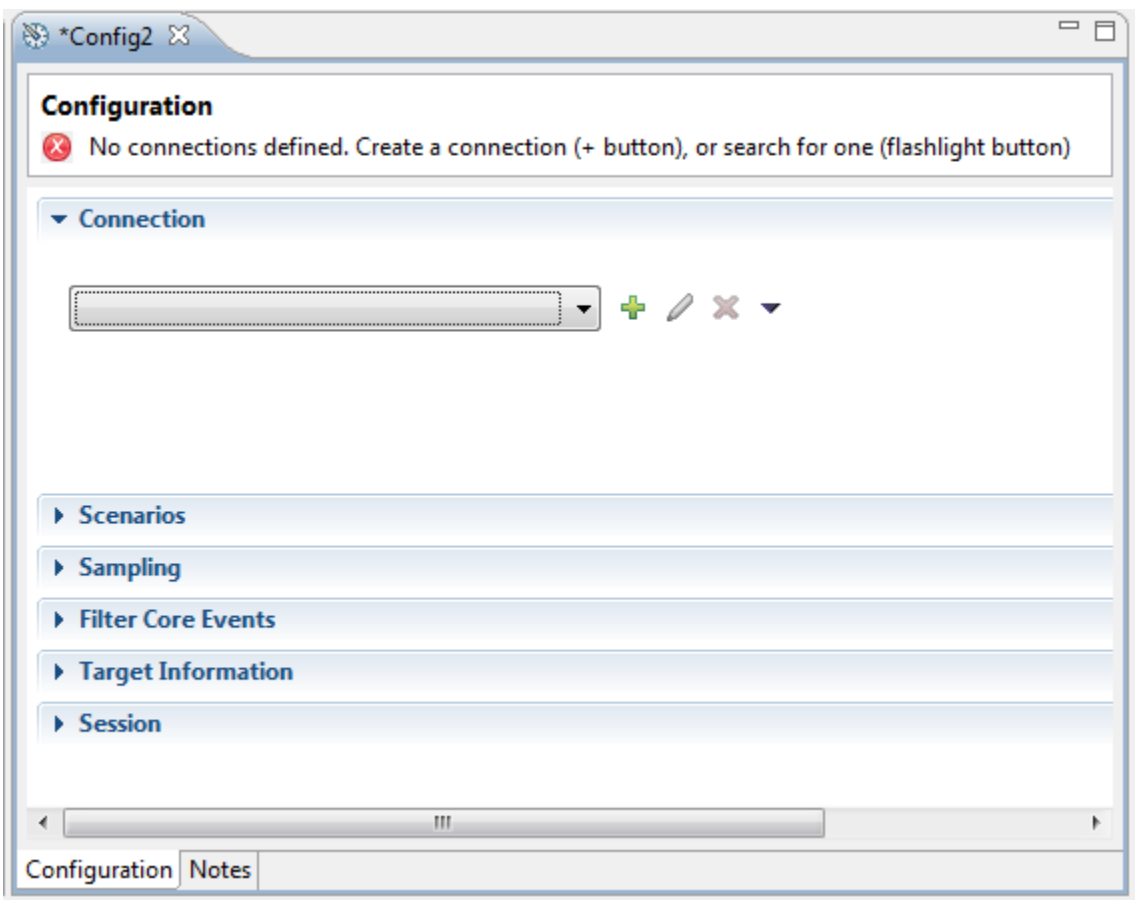


Figure 4-2. Configuration view

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

- Connection pane
- Scenarios pane
- Sampling pane
- Filter Core Events pane
- Target Information pane
- Session pane

4.1.2.1 Connection pane

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

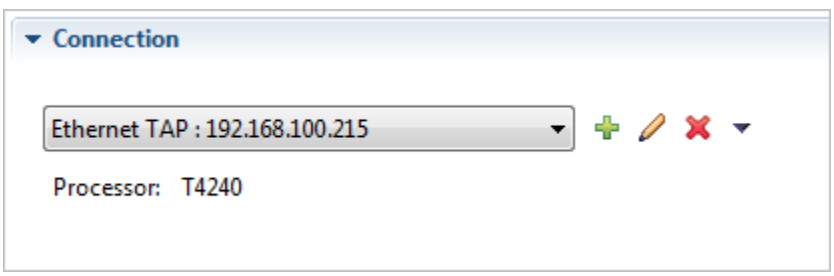


Figure 4-3. Configuration view - Connection pane

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

Table 4-1. Configuration view - Connection pane options

Option	Name	Description
	Pop-up 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. See Connecting to board 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 choose an option to display the selected connection by IP address, host name or alias.

Table continues on the next page...

Table 4-1. 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.

4.1.2.2 Scenarios pane

The **Scenarios** pane lets you define the scenarios that will be measured on a Performance Analysis project.

A scenario contains the following:

- Events - Hardware occurrences that can be counted to any event in the processor, including the output of other counters in a feedback loop.
- Counters - Counters that can be connected to any event in the 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.

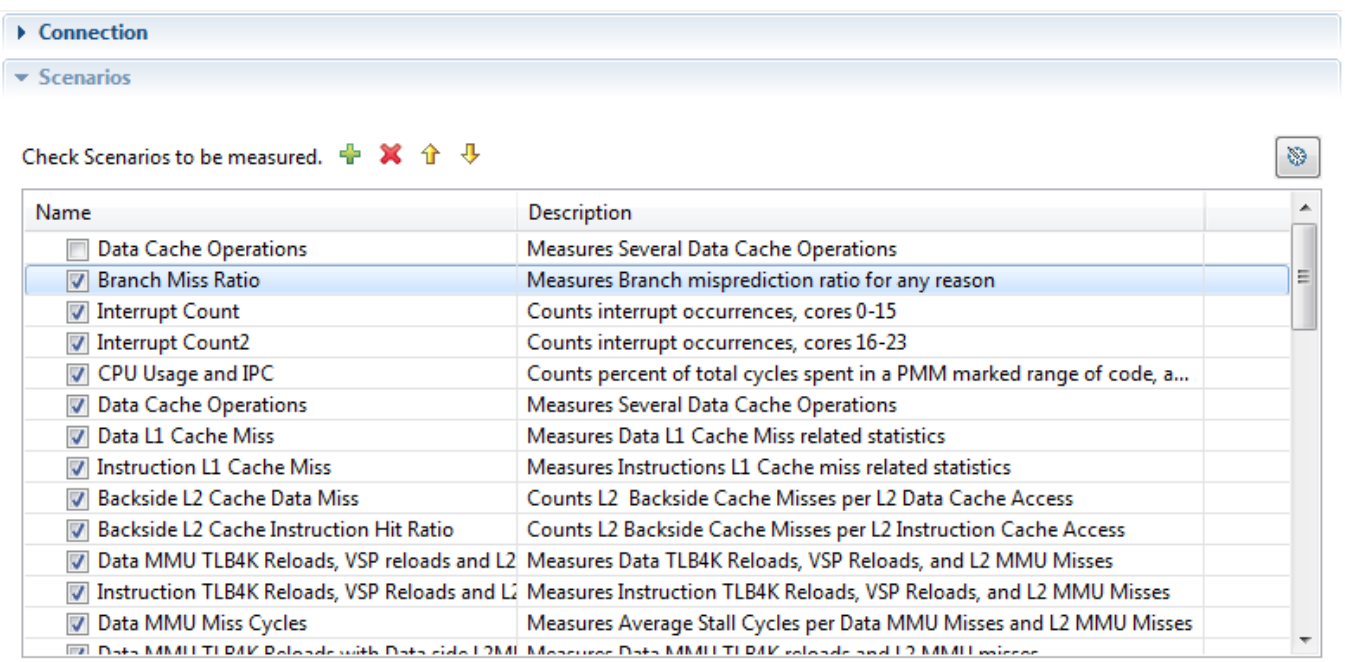


Figure 4-4. Configuration view - Scenarios pane

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

Table 4-2. 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. See 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 selected, the hardware is configured to measure the selected scenario when the profiling session starts. If deselected, no hardware configuration happens, that is the previous hardware settings remain active and the current values of the counters are read.

4.1.2.3 Sampling pane

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

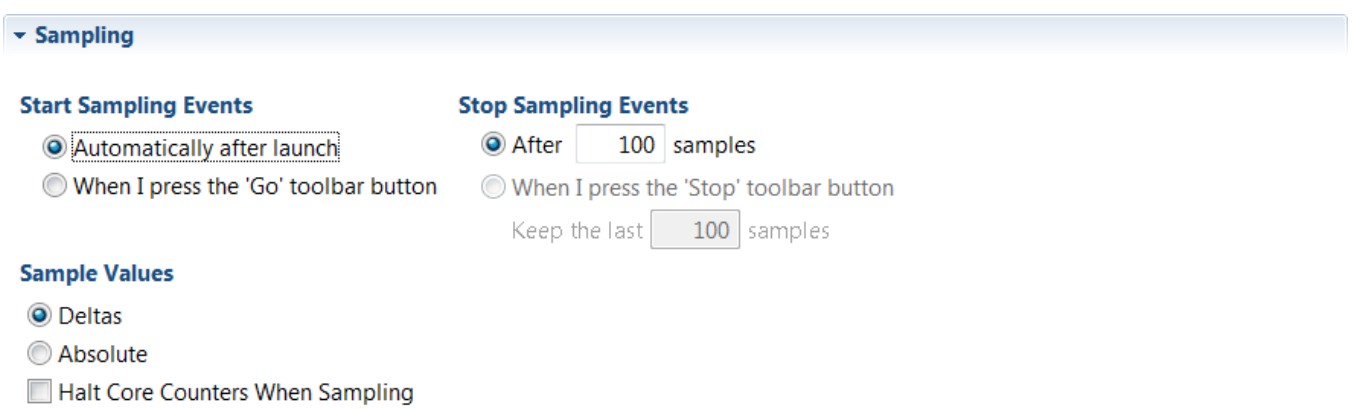


Figure 4-5. Configuration view - Sampling pane

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

Table 4-3. 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.
	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 **Analysis Sessions** 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 **Analysis Sessions** 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.

4.1.2.4 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.

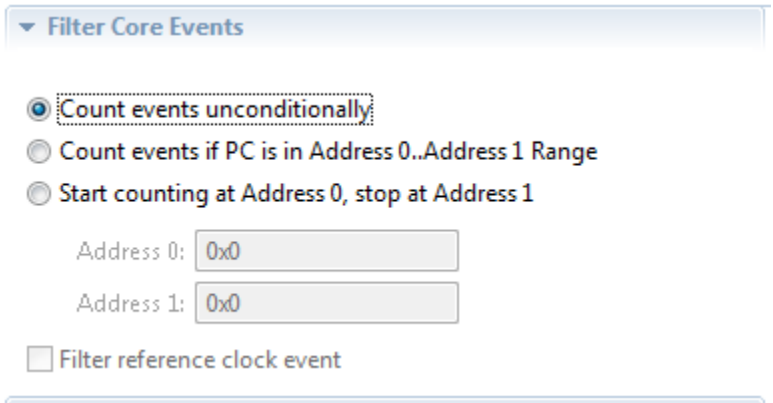


Figure 4-6. Configuration view - Filter Core Events pane

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

Table 4-4. 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.

4.1.2.5 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.

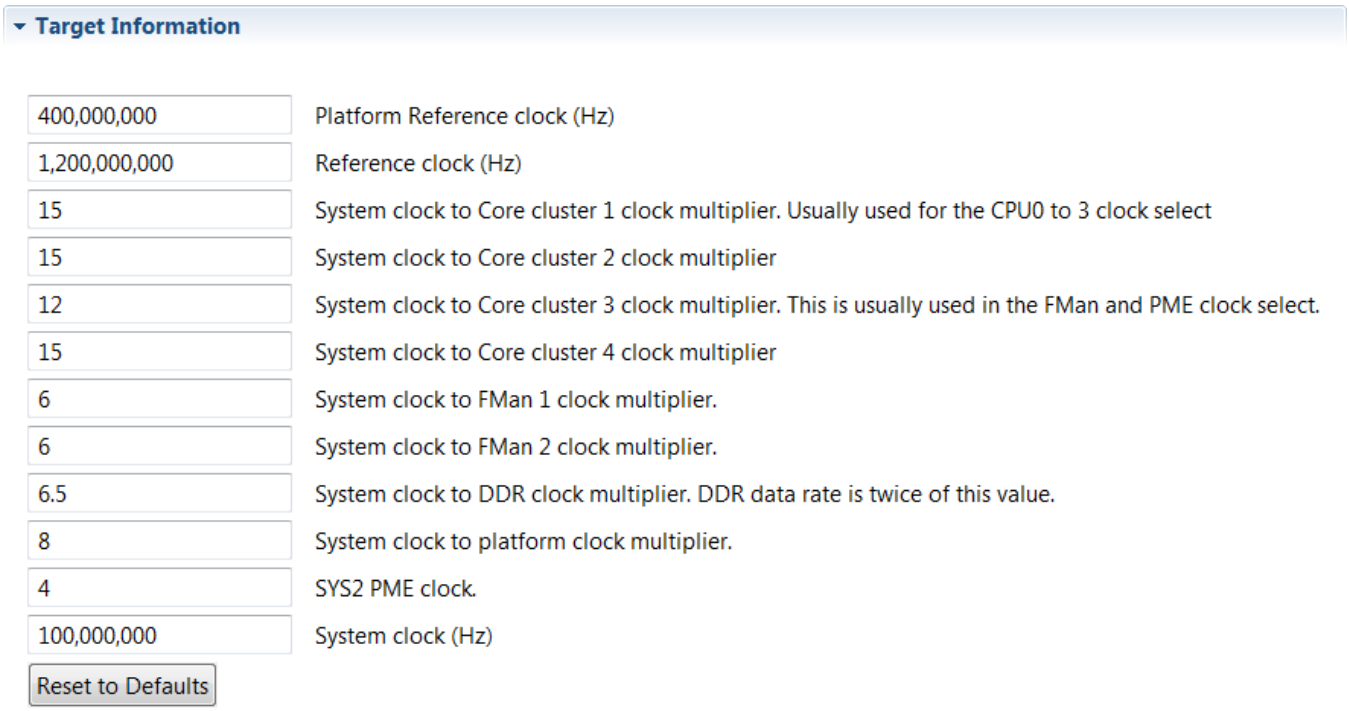


Figure 4-7. Configuration view - Target Information pane

4.1.2.6 Session pane

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

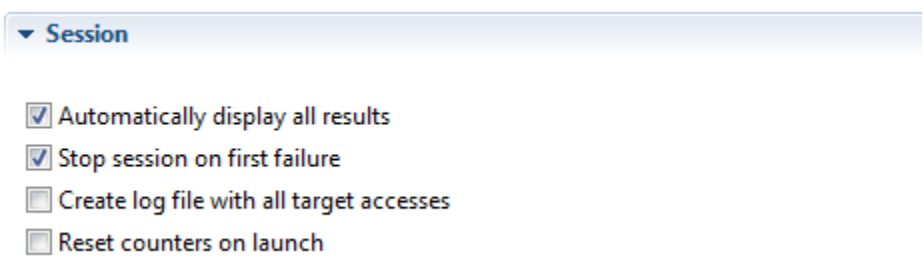


Figure 4-8. Configuration view - Session pane

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

Table 4-5. 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.
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.
Reset counters on launch	If selected, resets counters on the target when launching a new measurement session.

4.1.3 Analysis Sessions view

The **Analysis Sessions** 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**.

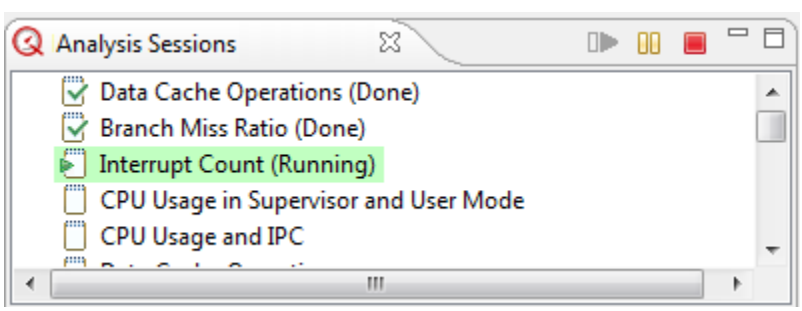


Figure 4-9. Analysis Sessions view

The buttons available in the **Analysis Sessions** 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.

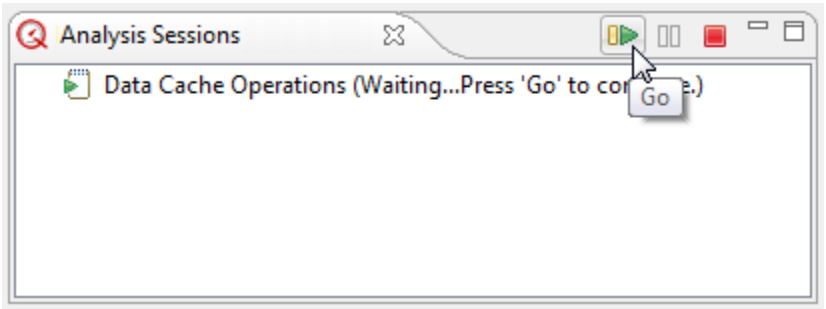


Figure 4-10. Go Button to start data collection

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

4.1.4 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, the status appears as `No operations to display at this time'.

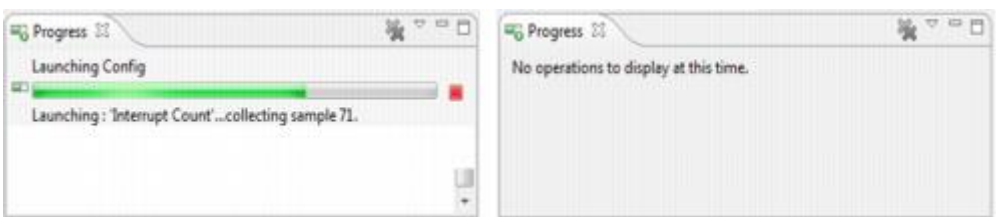


Figure 4-11. Progress view

4.2 Creating new configuration

To create a new configuration for Performance Analysis:

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

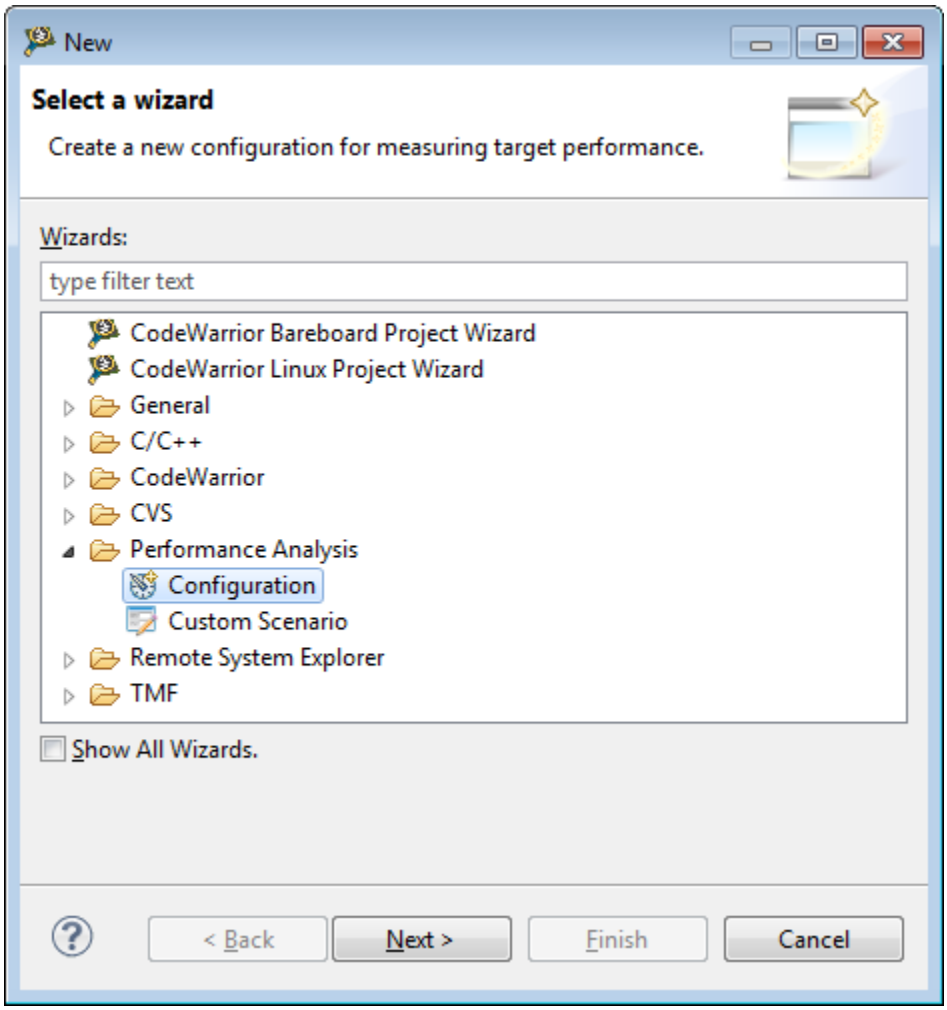


Figure 4-12. New Wizard

3. Choose **Performance Analysis > Configuration** to open the **New Configuration** screen.

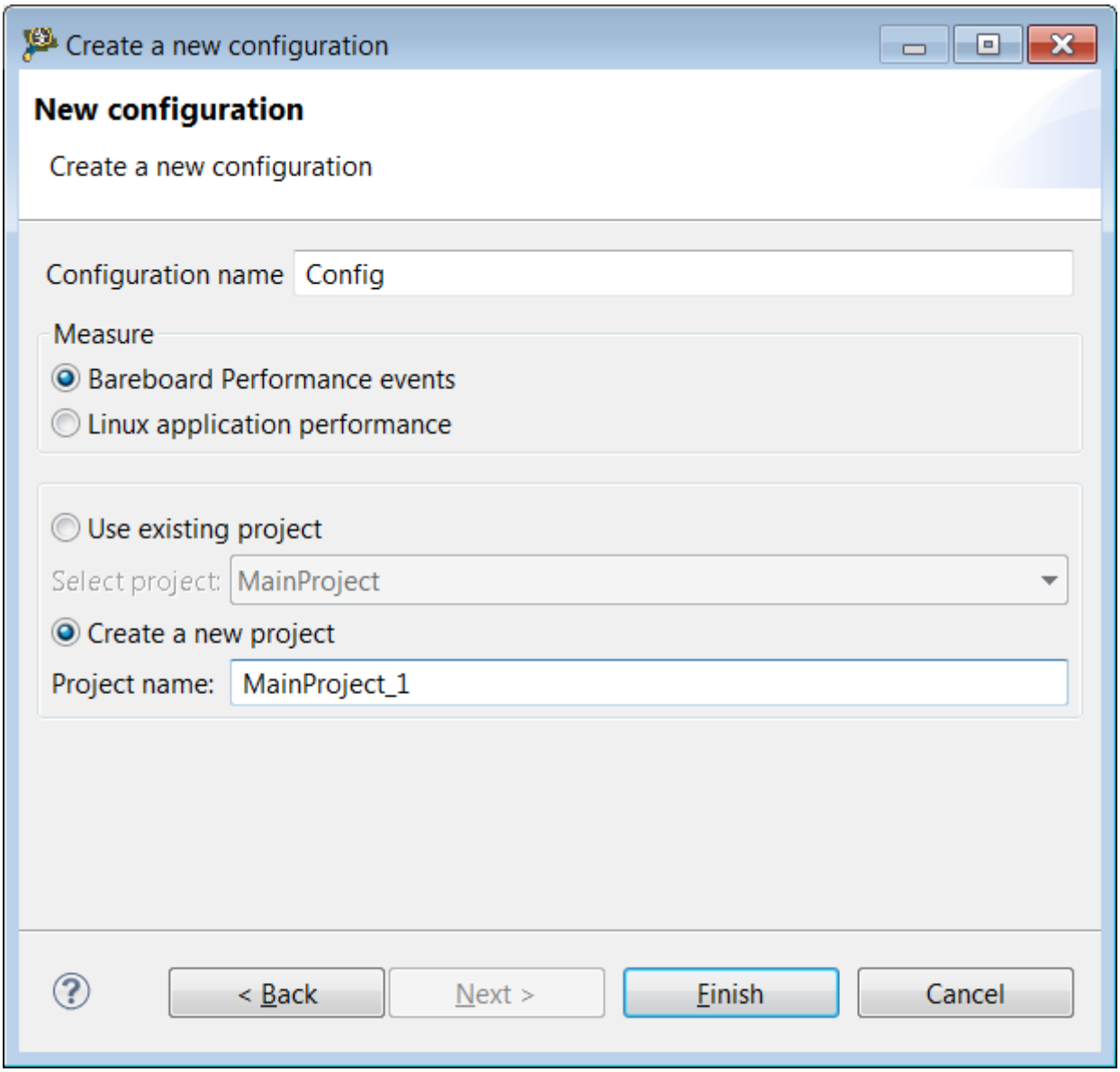


Figure 4-13. New Configuration screen

NOTE

You can also open the **New Configuration** screen directly by choosing **File > New Configuration** from the CodeWarrior IDE menu bar.

4. Enter a name for the configuration in the **Configuration name** textbox.
5. Choose **Bareboard Performance events** for measuring profiling events in bareboard applications or choose **Linux application performance** for Linux performance, under **Measure** field.
6. Enter 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.

4.3 Connecting to board

After creating a Performance Analysis project, you need to select a connection to the board.

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

To manually add a connection to the board:

1. Click the **New Connection** icon  in the **Connection** pane 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.

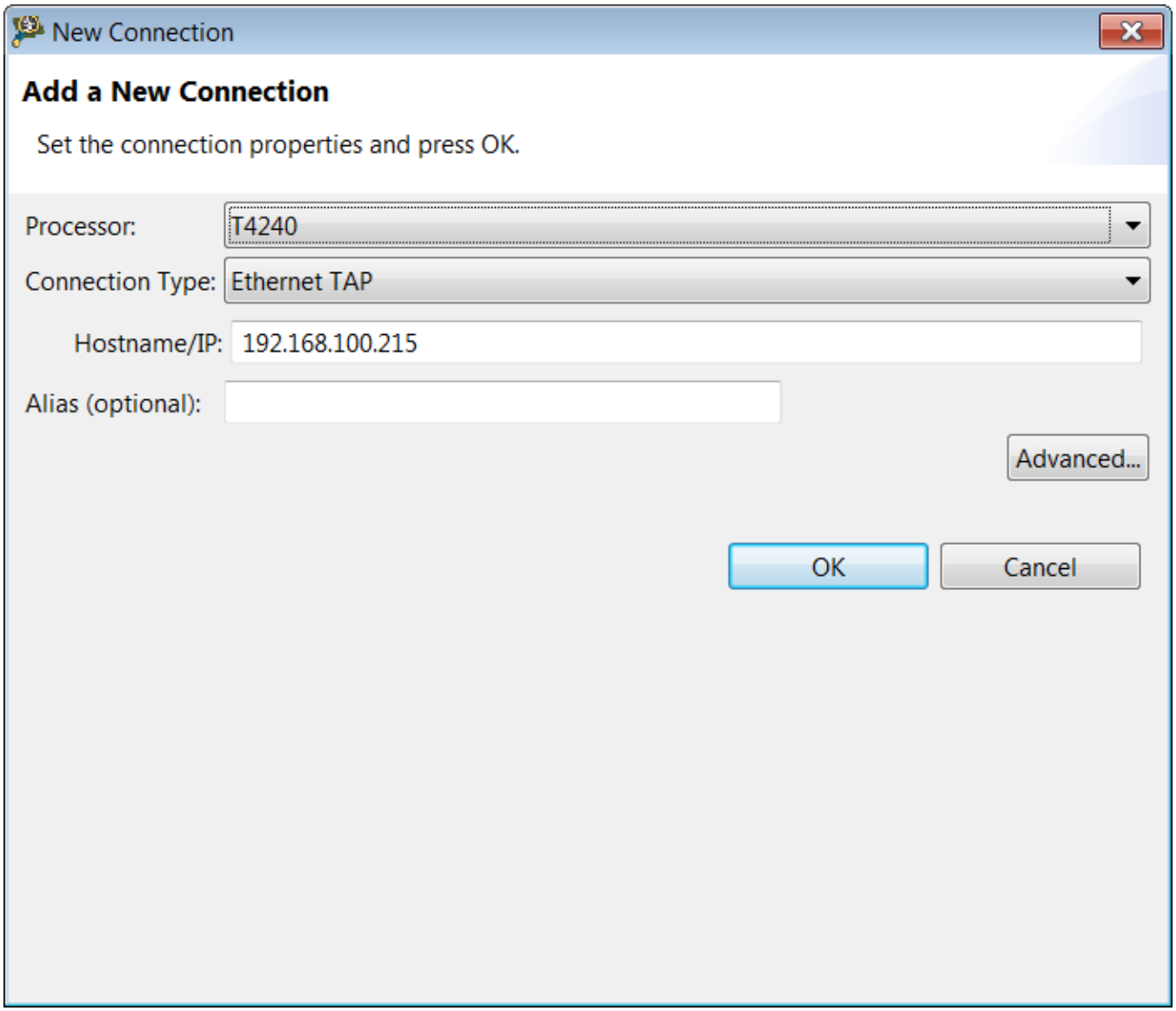


Figure 4-14. Add a New Connection screen

4. Specify an alias for the connection in the **Alias** textbox. This field is optional.
5. For updating **Command Converter Server (CCS)**, **JTAG settings (CCS)**, and **CDDE (agent)** internal parameter values, click **Advanced** button.
6. Click **OK** to save the settings.

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

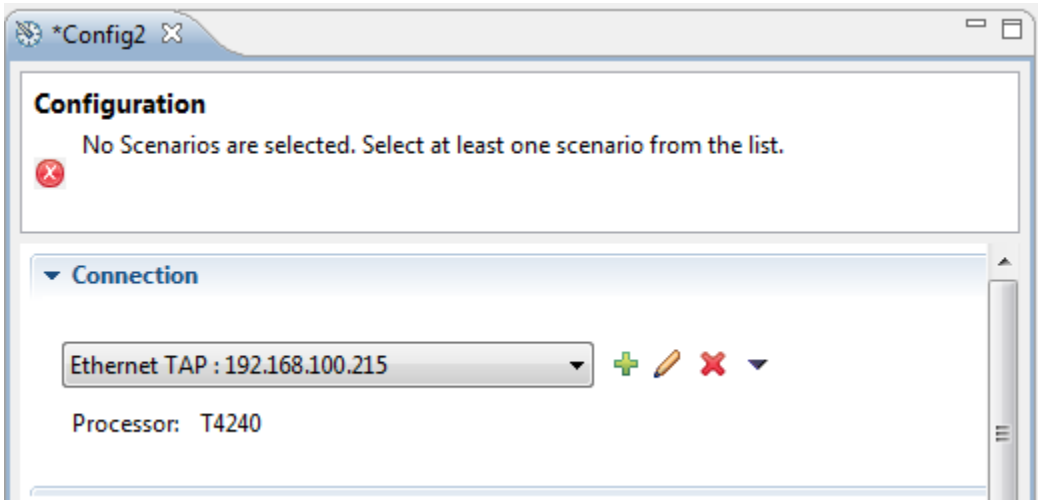


Figure 4-15. Connection added

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

4.4 Selecting scenarios

You can add scenarios to be measured to the Performance Analysis configuration.

To select a scenario to add it to the Performance Analysis configuration:

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

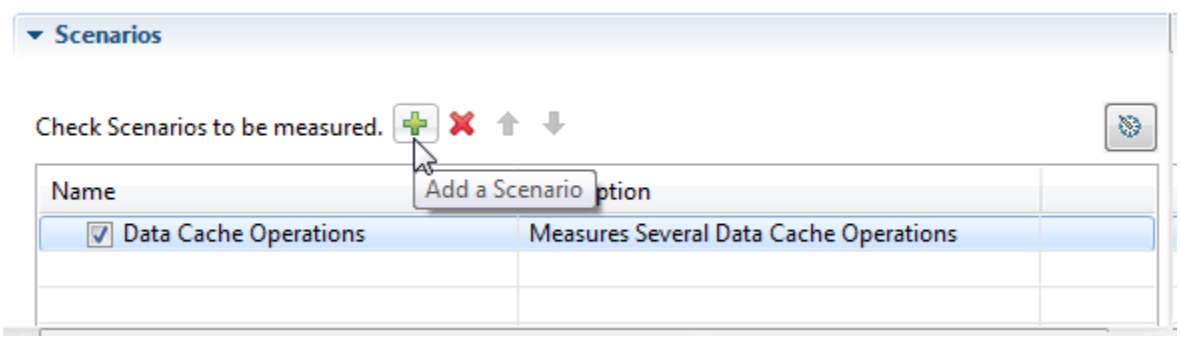


Figure 4-16. Add a Scenario

   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.

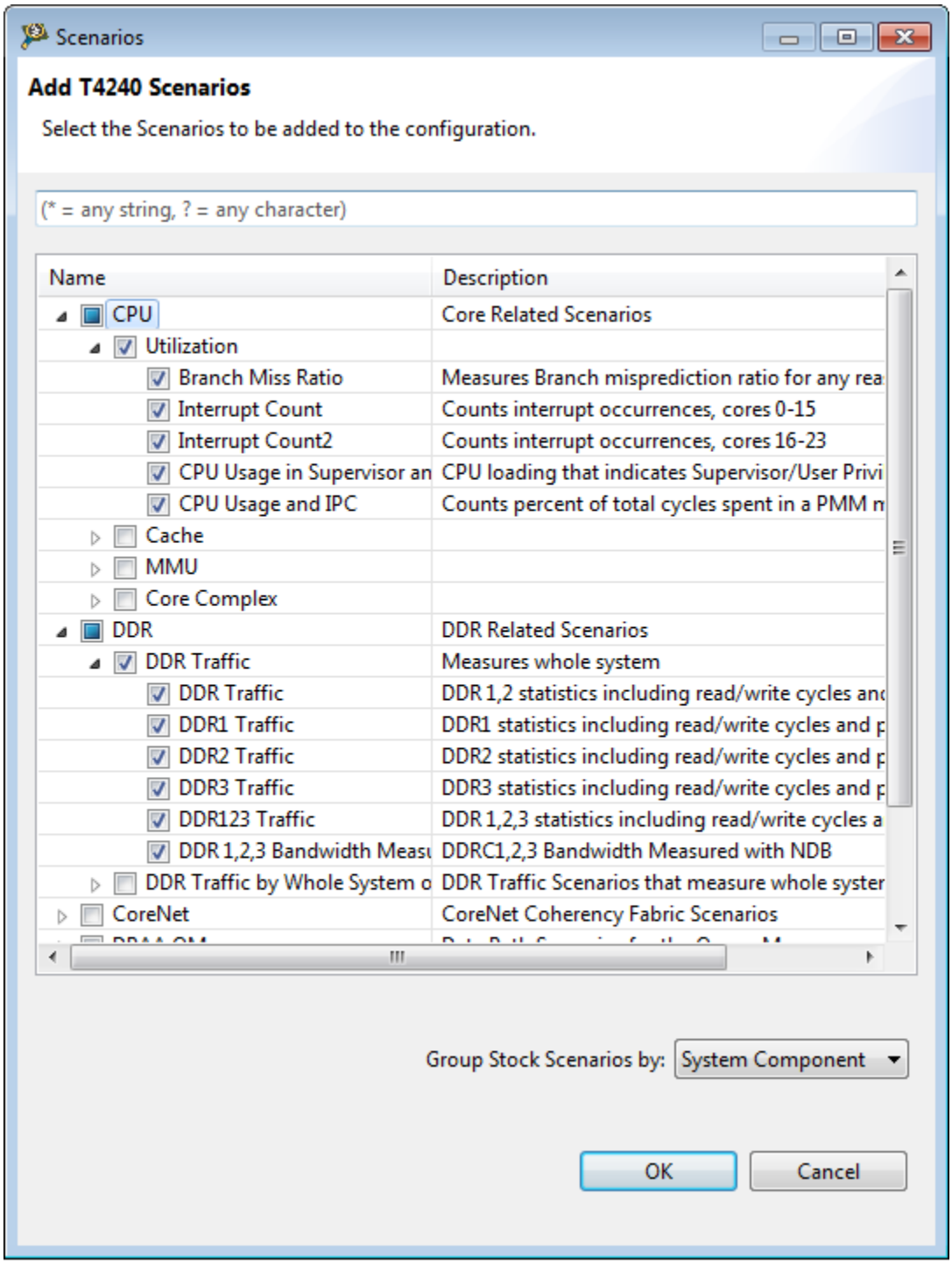


Figure 4-17. Add <Processor> Scenarios screen

NOTE

For Bx devices, 2 DDR Controllers are supported to measure DDR bandwidth with NDB (NXC Data Beats) counter. For Tx devices, 3 DDR Controllers are supported to measure DDR bandwidth with NDB counter. To measure DDR bandwidth on B4860 or the platforms that

use less than 3 cores, it is recommended to use the DDR bandwidth measured with NDB scenario. When a platform executes on more than 3 cores, the DDR traffic with Page Miss and Collision Analysis scenario can be used to measure DDR qualification traffic.

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

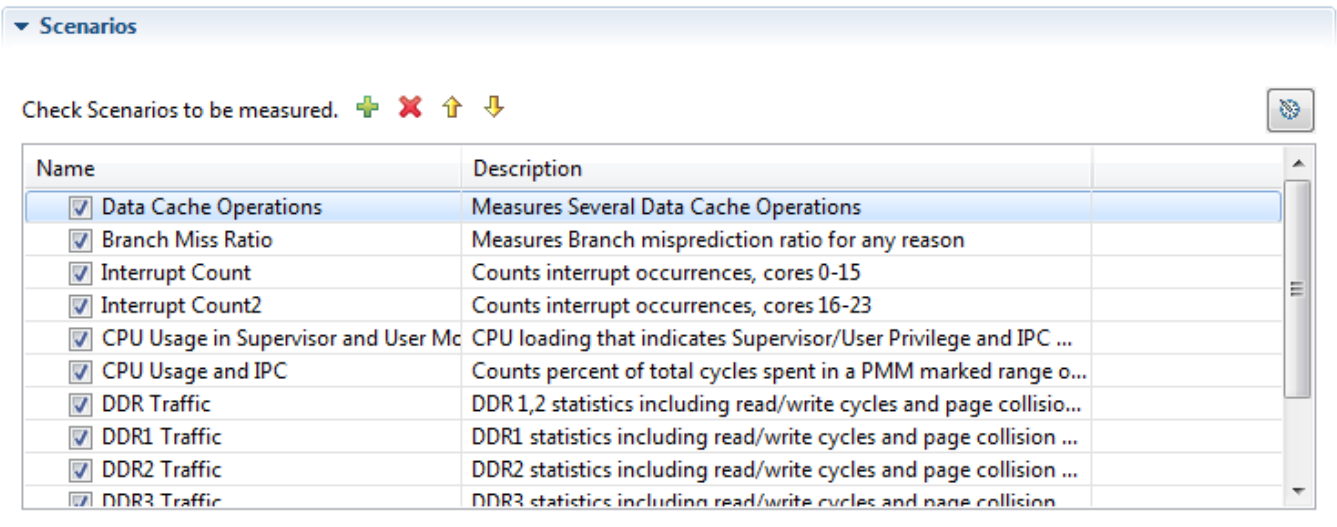


Figure 4-18. Scenarios added to configuration

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. The **Profile Configurations** dialog appears on choosing **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.

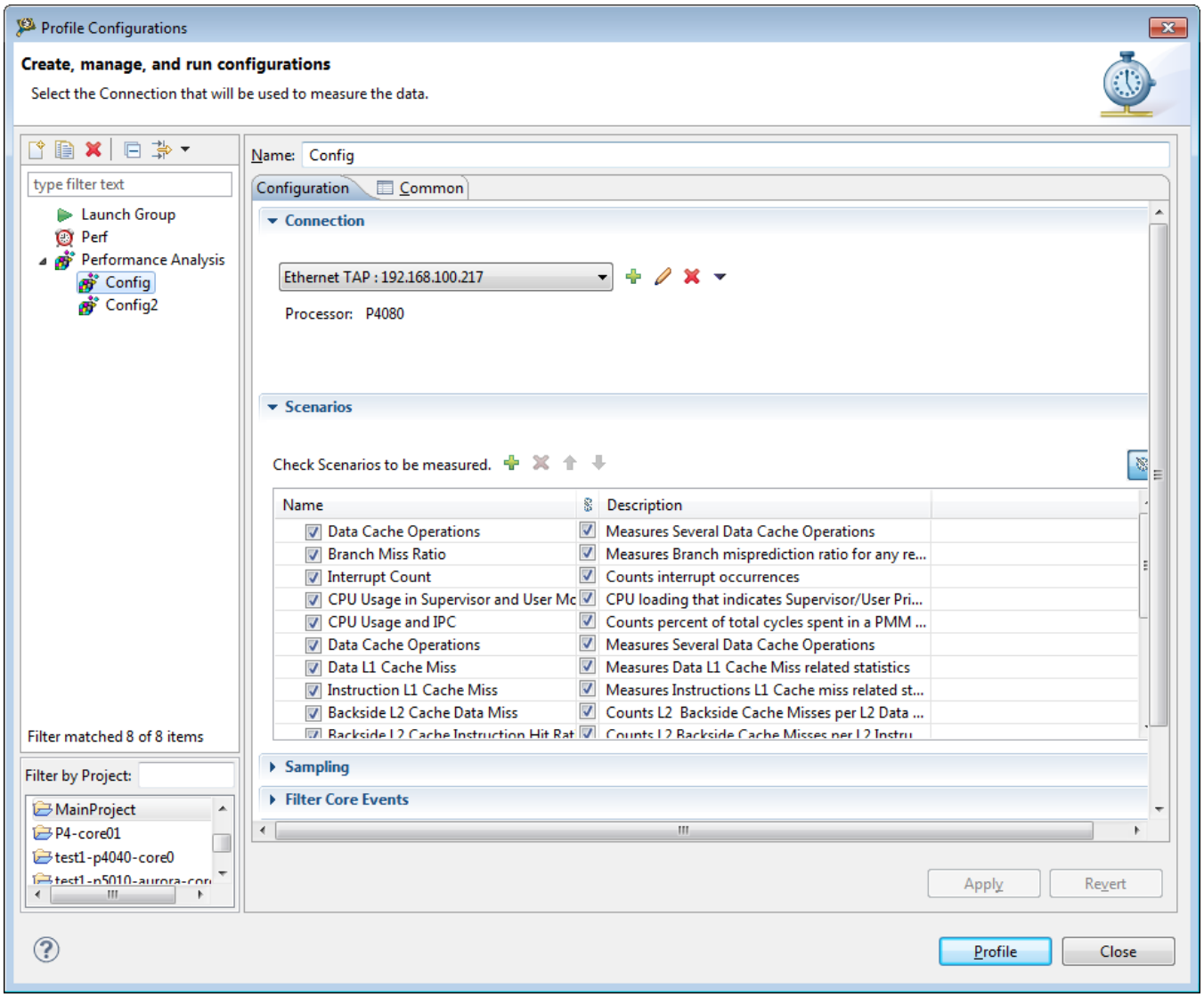


Figure 4-19. Profile Configurations dialog

4.5 Running scenarios

After adding scenarios to a Performance Analysis configuration, you need to run them for measurement.

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

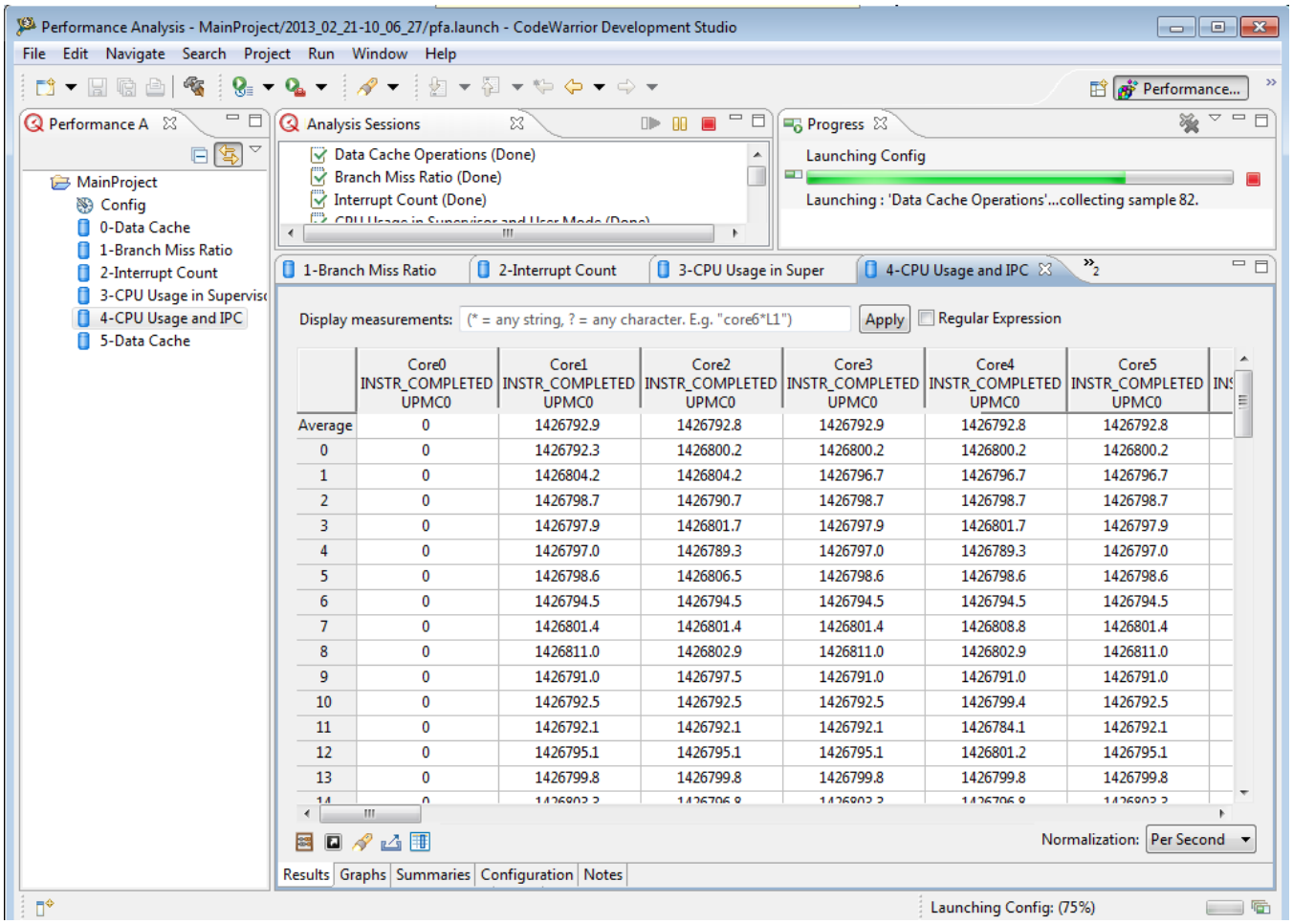


Figure 4-20. Profiling/Measurement in progress

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.

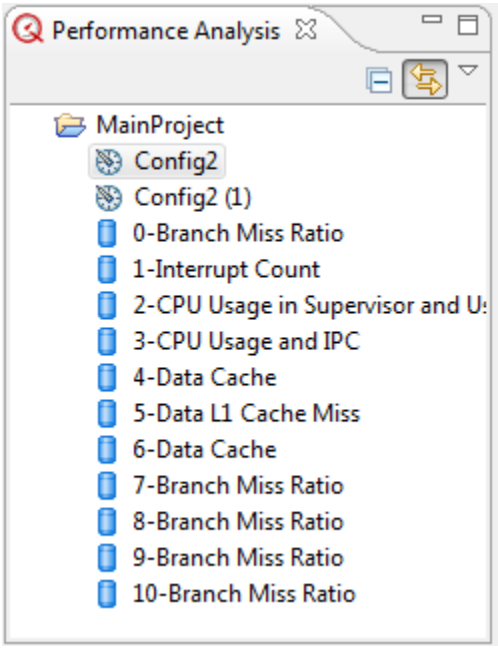


Figure 4-21. Measurement results

4.6 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 (Figure 4-21). 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 selected in the **Session** pane of the **Configuration** view, the results will get open automatically while data is collecting.

NOTE

If you are working on the Windows platform, do not keep too many scenario measurements results opened in their respective viewers, because Windows has a limited number of user handles, which are used for the SWT components. When this limit is reached, UI elements do not display or work properly. The workaround for this problem is to close the unused editors and tabs in the **Performance Analysis** perspective.

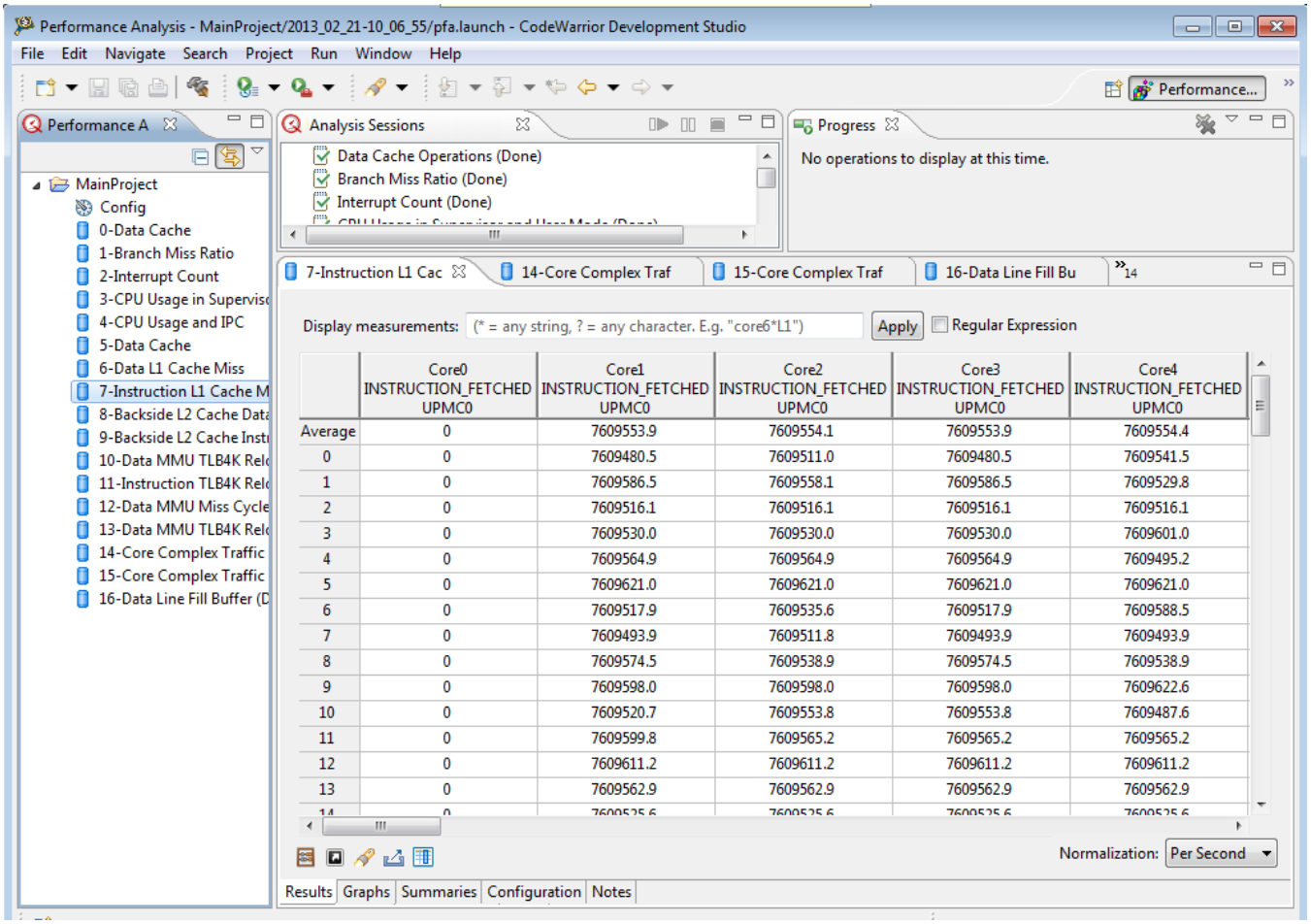


Figure 4-22. Data collection results

You can specify strings to filter scenario results according to their titles using the **Display measurements** textbox. For example, to display results 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 results with BTB_MISS_RATIO as title will be displayed. If you want a specific string to be matched, deselect 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** deselected) 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?*.

The table view displays a toolbar providing various options, such as viewing metrics definitions, searching the table to look for desired results, and exporting results to CSV format. The table below lists the options available in the toolbar of the table view.

Table 4-6. Options available in toolbar of collected results

Option	Name	Description
	View Metric Definitions	Lets you view definitions of all metrics that appear in the results report. Click the button to open the **Report Metric Definitions** dialog. You can also use the dialog to search for a particular metric definition. This option is also available in the **Graphs** and **Summaries** tab.
	Aliases	Lets you define alternate names to be used for the metrics in the results report. Click the button to open the **Aliases** dialog, and provide a name in the **Alias** column. This option is also available in the **Graphs** and **Summaries** tab.
	Search Table	Lets you find and search particular results in the report. Click the button to open the **Find** dialog, and specify search options according to requirements. This option is available only in the **Results** tab.
	Export to CSV	Lets you export the results report to a CSV file. Click the button to open the **Export to CSV** dialog. You can choose the results based on the selected normalization. In addition, you can filter the events and metrics that you want to export. This option is available only in the **Results** tab.
	Choose Measurements	Lets you display the measurements on the table view according to choice. Click the button to open the **Measurement Chooser** dialog and select the checkboxes corresponding to the measurement you want to display. You can also specify the order in which you want the measurements to appear by using the **Move Up**/**Move Down** buttons. This option is also available in the **Graphs** and **Summaries** tab.

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

- **Results** - Displays the default view showing the event data generated from the Performance Analysis tool.
- **Graphs** - Displays a graphical view of the data.
- **Summaries** - Displays the maximum, minimum, and average value of the statistics measurements computed for each metric.
- **Configuration** - Displays read-only view of the configuration settings related to connection , sampling, scenarios, filter core events, which you provided to generate results from the Performance Analysis tool.

- **Target Access Log** - Displays the log file of the measurement after running the collection. The tab appears only if the **Create log file with all target accesses** checkbox is selected in the **Session** pane.
- **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.

Right-click a particular data result in the **Performance Analysis** view to get a shortcut menu displaying the following commands:

- **Normalization** - Provides three options to view results. Normalization is also available at the bottom of the table view on the right side.
  - **None** - Displays the number of events without further processing.
  - **Per Cycle** - Displays the number of events occurring per cycle.
  - **Per Second** - Displays the number of events occurring per second. To compute normalization per second, the system clock set in the **Target Information** pane is used.
- **Delete** - Deletes the selected result.
- **Rename** - Lets you rename the selected result.
- **Import Projects** - Lets you import the selected result into another project.
- **Export Projects** - Lets you export results from another project.
- **Import Custom Scenarios** - Lets you import customized scenarios into your configuration. A custom scenario can be created by combining the events/metrics. The default (stock) scenarios are predefined and cannot be changed, but you can modify a custom scenario and import it into a performance analysis configuration. Clicking this option opens the **Import Custom Scenarios** dialog. Browse through the directory where the custom scenario folder is available, and select the custom scenario. Click **OK** to close the dialog and get the custom scenario imported into your configuration.
- **Show in Explorer** - Lets you view scenario results in Windows Explorer.

This topic describes the following sub-topics.

- Tabular format
- Graphical format

4.6.1 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.

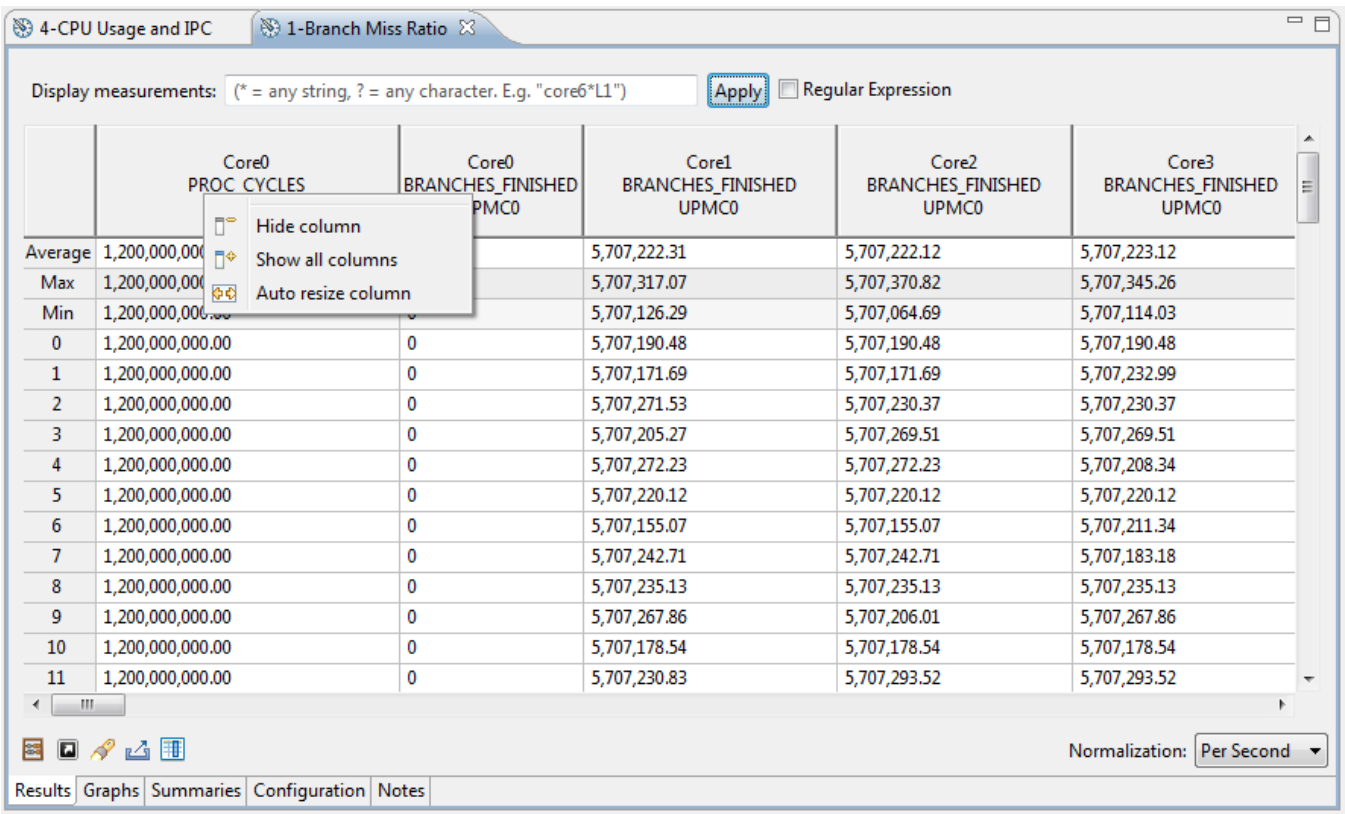


Figure 4-23. Data collection results in tabular form

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

- **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.

4.6.2 Graphical format

The graphical format of the results generated from the Performance Analysis tool displays measurements in a graph for each event per core.

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

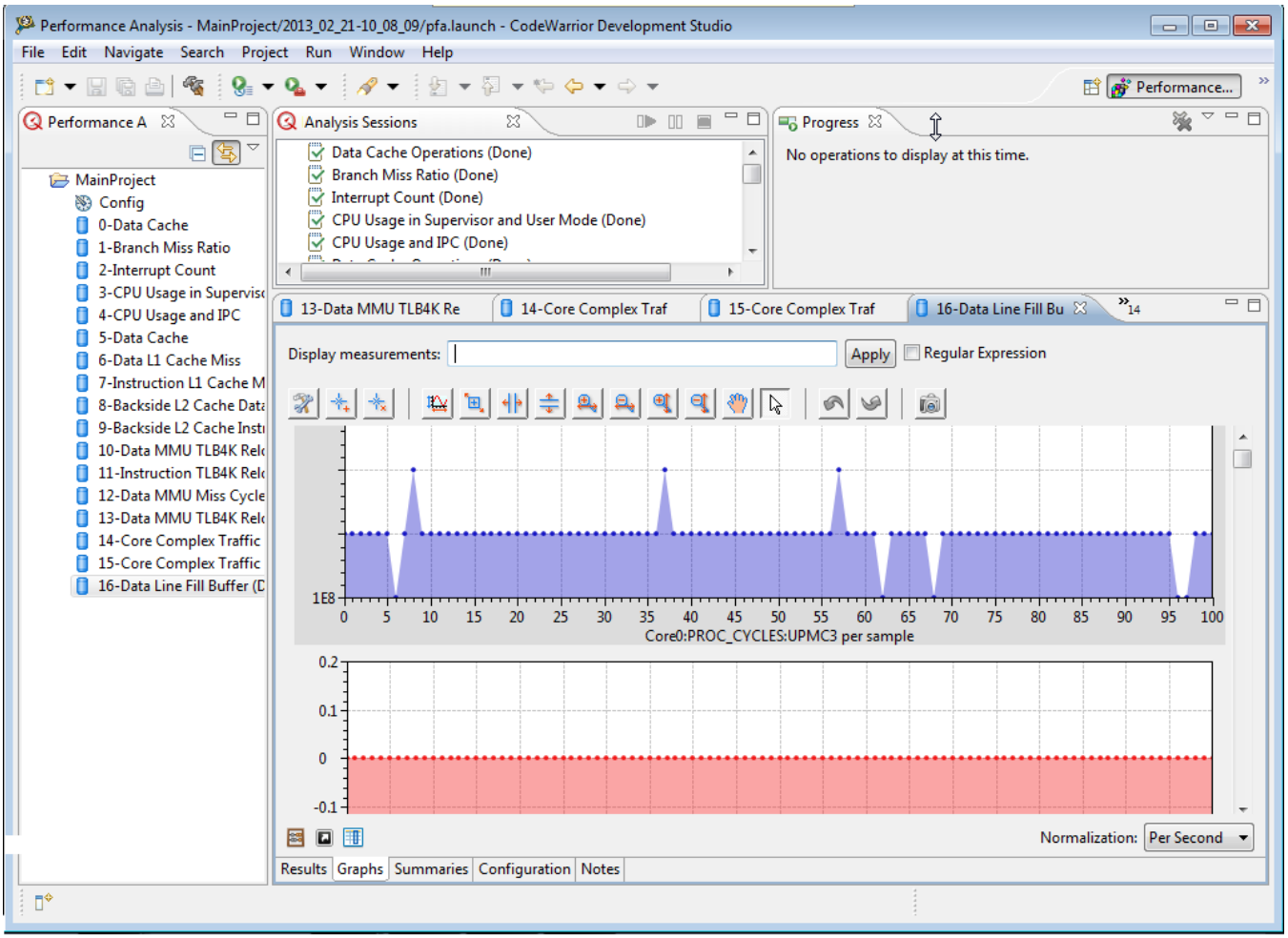


Figure 4-24. Data collection results in graphical format

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 4-7. 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 BTB_MISS_RATIO as title will be displayed. If you want a specific string to be matched, deselect 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** deselected) 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?*.

Table continues on the next page...

Table 4-7. Options Available in Graphical Format of Collected Results (continued)

Option	Name	Description
	Configure Settings	Allows you to configure the settings of a selected graph. Click to display the **Configure Graph Settings** dialog, 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 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.
	Save Snapshot to PNG FIle	Allows you to capture image of the selected graphs in the PNG format.

The graphical view also displays a toolbar providing various actions that can be performed on the graph. See Table 4-6 for detailed description on these toolbar options.

4.7 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.

4.7.1 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.

4.7.2 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

4.7.2.1 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.

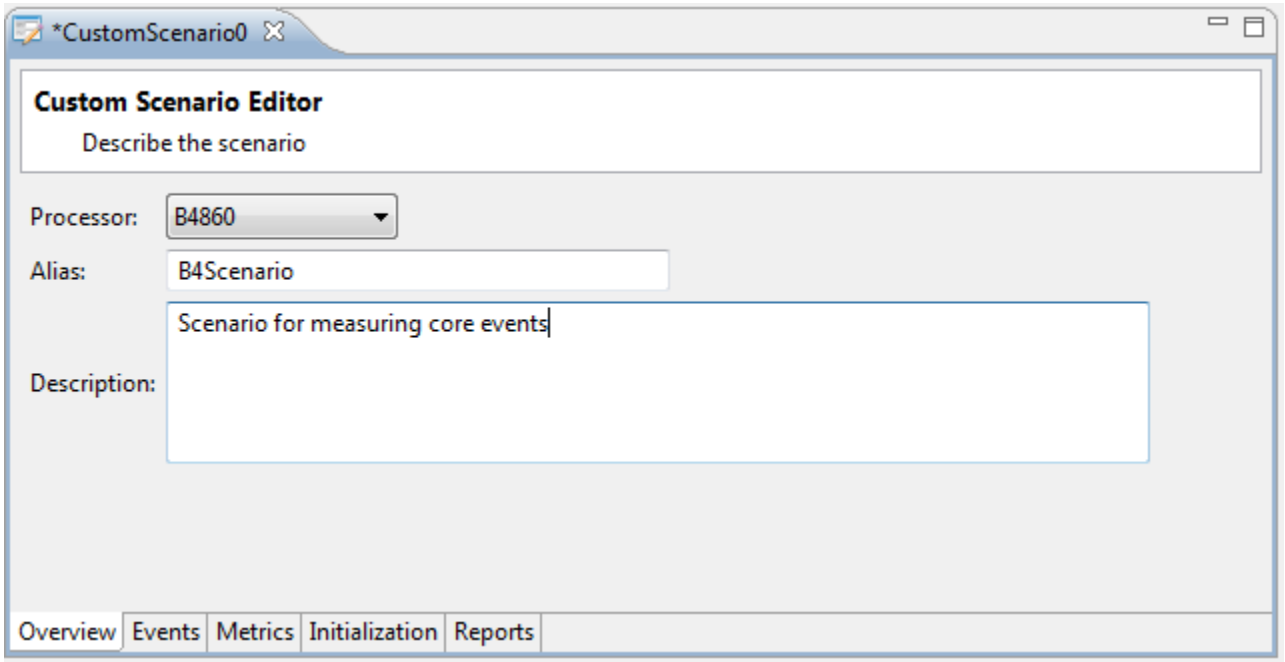


Figure 4-25. Custom Scenario Editor - Overview tab

4.7.2.2 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.

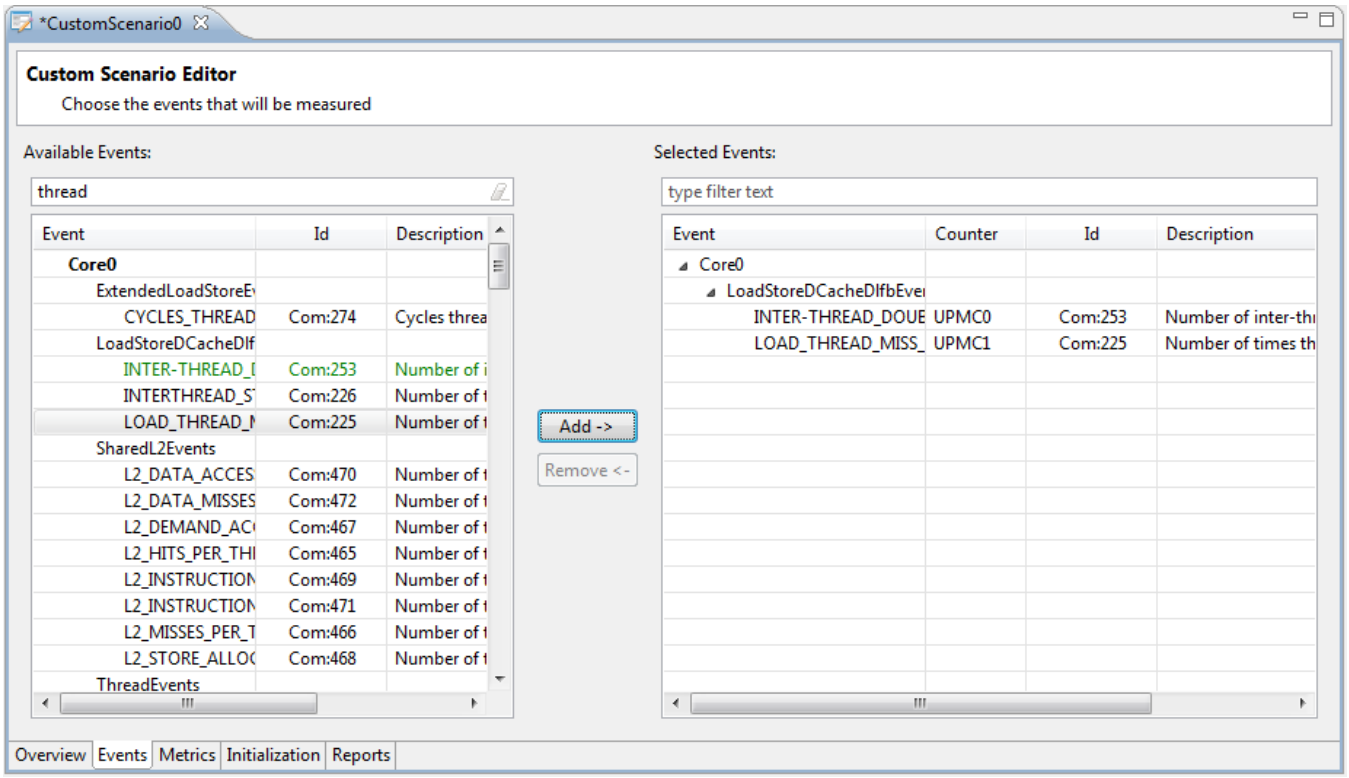


Figure 4-26. Custom Scenario Editor - Events tab

4.7.2.3 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

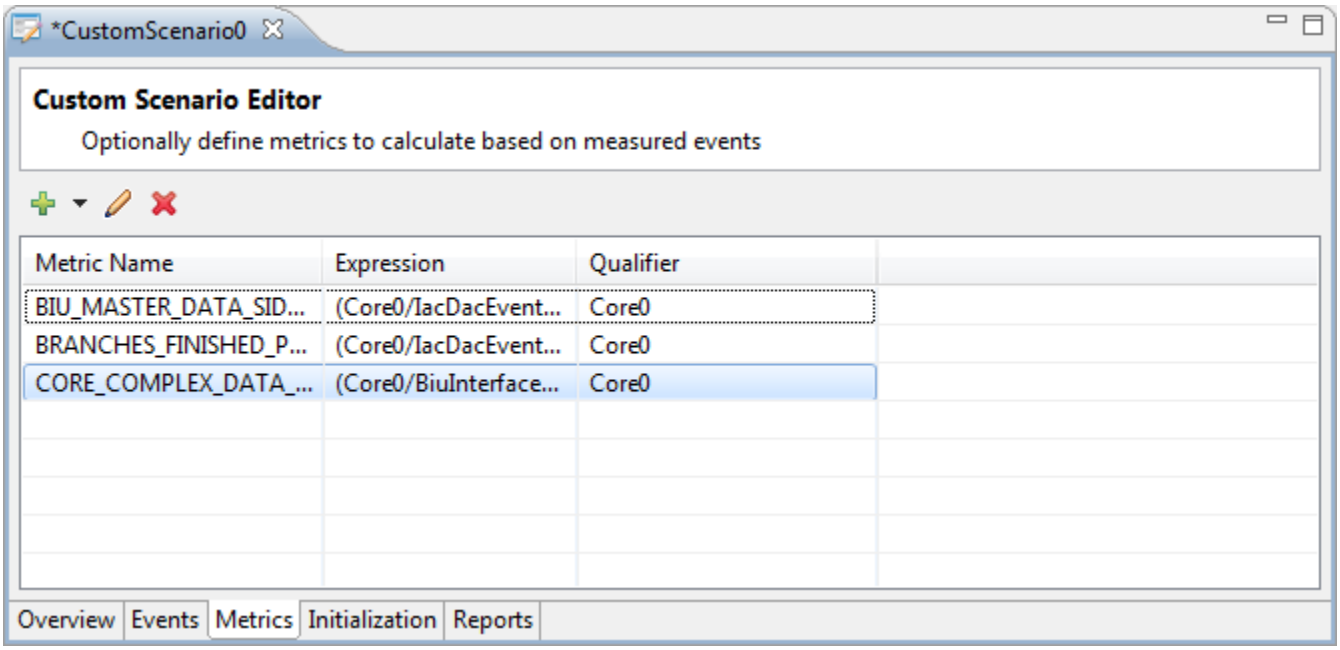


Figure 4-27. Custom Scenario Editor - Metrics tab

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**.

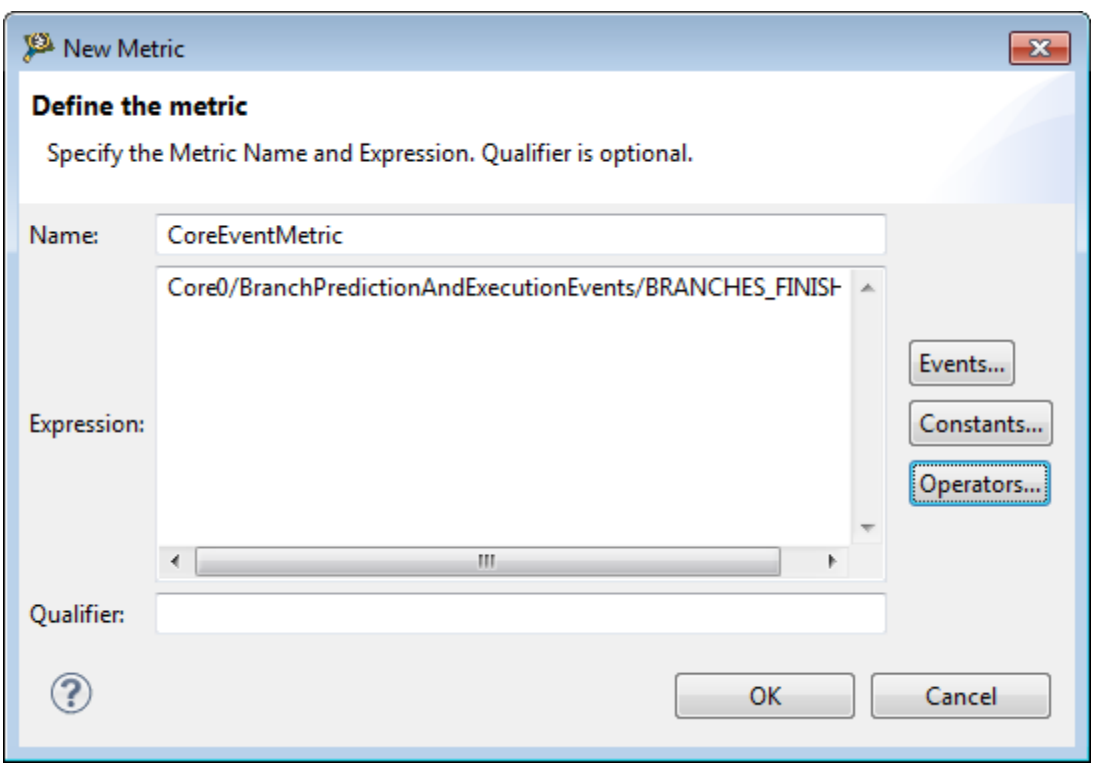


Figure 4-28. Add new metric

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.

4.7.2.4 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.

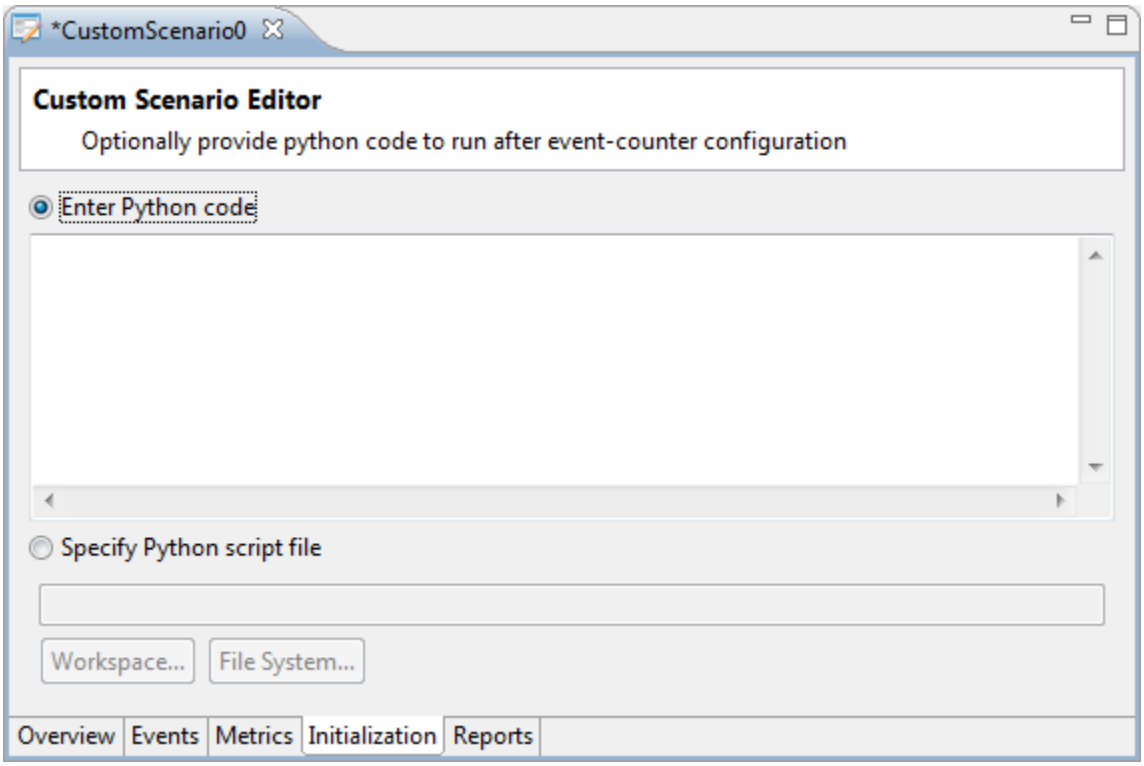


Figure 4-29. Custom Scenario Editor - Initialization tab

4.7.2.5 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.

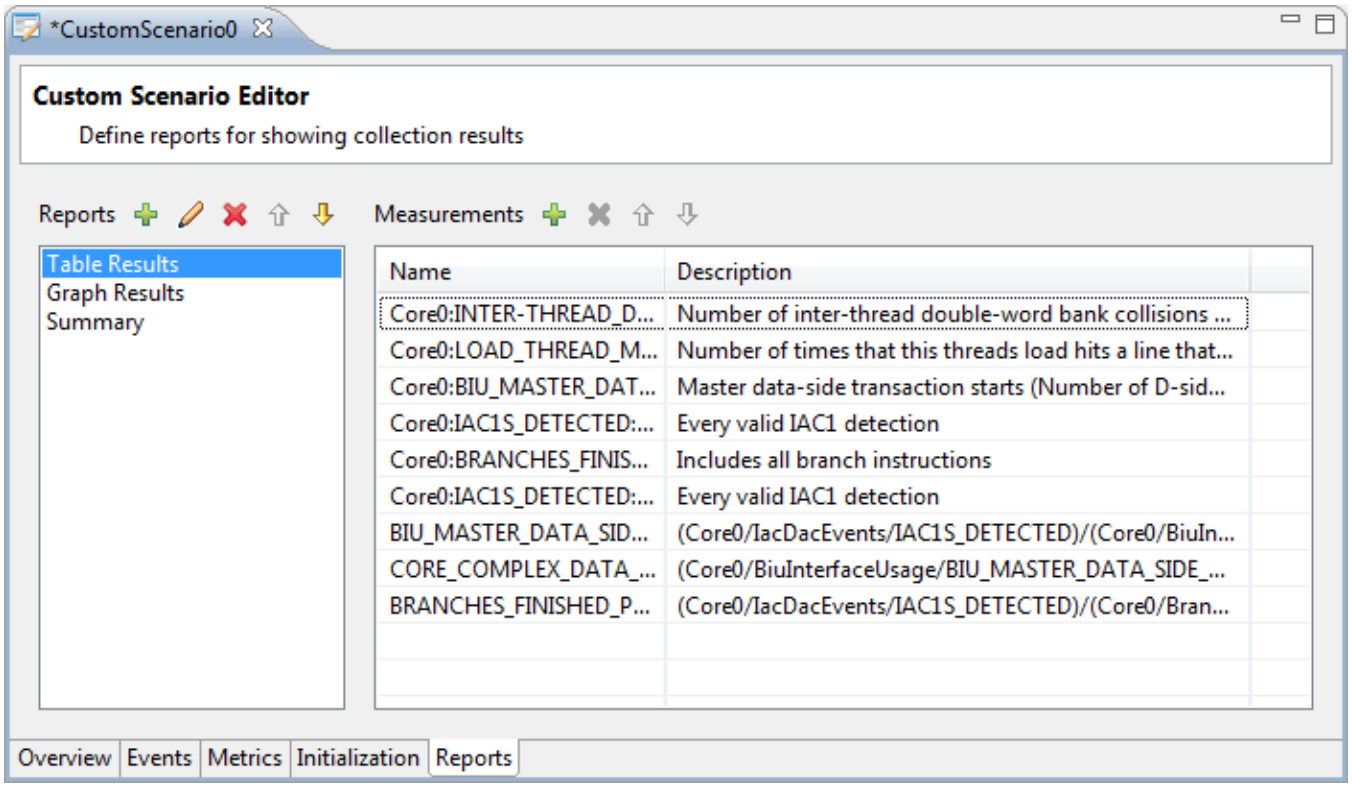


Figure 4-30. Custom Scenario Editor - Reports tab

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.

Index

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number CWPASWAUG
Revision 10.5.1, 01/2016