NXP Semiconductors
Reference Manual

Document Number: CWARMv8TAUG
Rev. 11.3.0, 12/2017

CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA, Tracing and Analysis User Guide

Contents

Chapter 1 Introduction

This manual explains how to use the CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA product.

This chapter presents an overview of the manual. The topics in this chapter are:

- About this Manual - Describes the contents of this manual
- Accompanying Documentation - Describes supplementary CodeWarrior documentation, third-party documentation, and references.

1.1 About this manual

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

The following table lists the contents of this manual.

Table 1. Manual contents

Chapter	Description
Introduction	This chapter.
Collect Trace Data on page 4	Explains how to use the CodeWarrior for ARMv8 to collect trace data.
Trace Commander View on page 16	Explains how to manage and view the trace data.
View Trace Data on page 21	Explains how to view various types of data trace collected on an application.
Collect and View Linux satrace Data on page 45	Explains how to collect and view satrace with and without using CodeWarrior.
Linux Kernel and User Applications Debug Print Tool on page 56	Explains how Debug Print Tool works. The tool is independent of CodeWarrior and does not require a debug session.

1.2 Accompanying documentation

The Documentation page describes the documentation included in this version of CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA.

You can access the Documentation page by:

- Opening START_HERE.html from *<CWInstallDir>**CW_ARMv8**ARMv8**Help* folder or select **Help > Documentation** from the IDE's menu bar.
- To view the online help for the CodeWarrior tools, select **Help > Help Contents** from the IDE's menu bar.

Chapter 2 Collect Trace Data

This chapter guides you on how to use the CodeWarrior for ARMv8 to collect trace.

It also explains the steps to configure, collect, store, and visualize trace from a target. This chapter includes the following sections:

- Process for collecting data on page 4
- Creating a new project on page 4
- Configuring target on page 6
- Configuring Debug Launcher on page 8

2.1 Process for collecting data

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

To collect trace data, perform the following steps in sequence:

1. Create and configure a project
2. Set up the debugger Target Connection Configuration (TCC) to collect the analysis data from the hardware
3. Run the application on the target to collect trace data

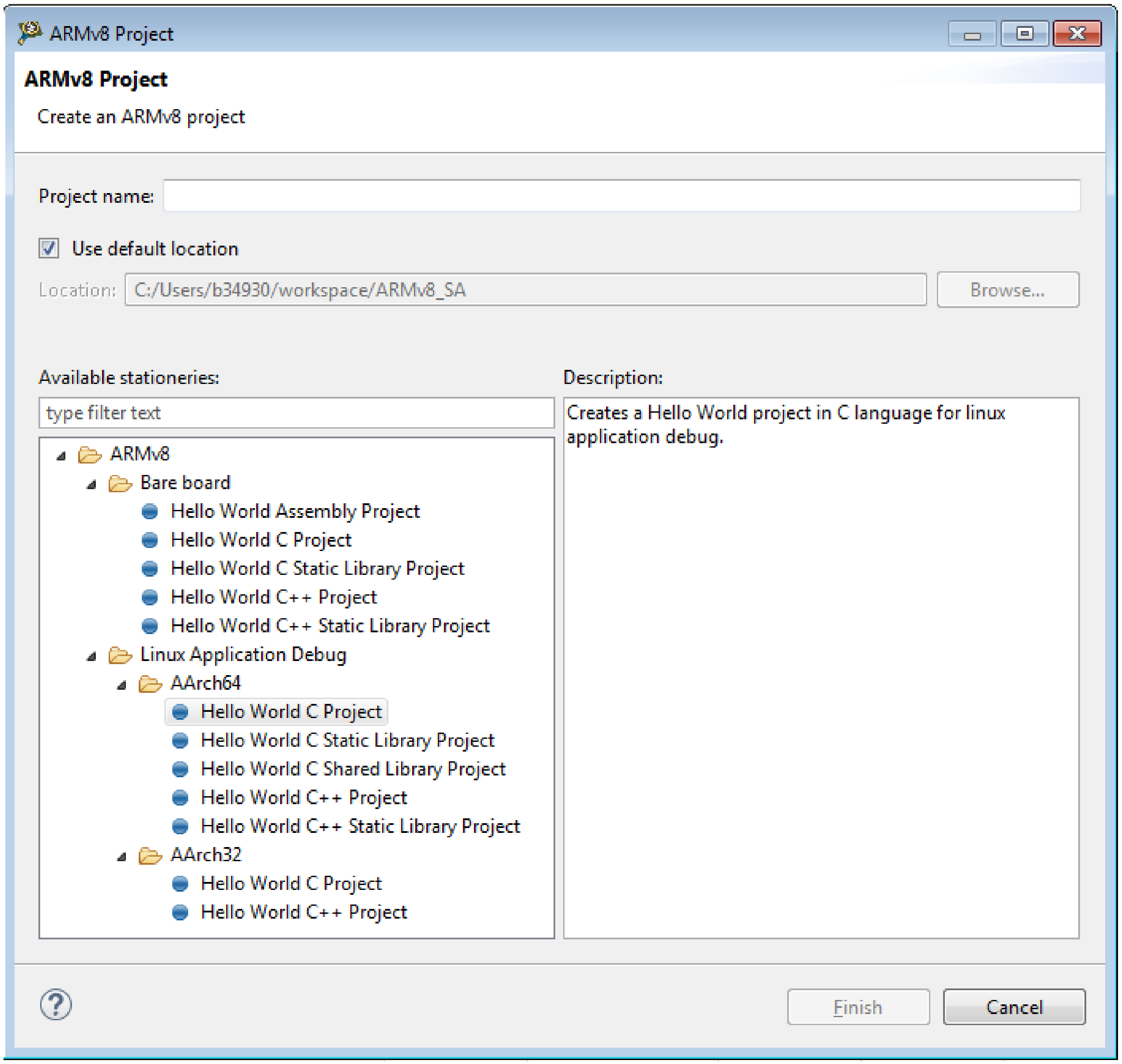
2.2 Creating a new project

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

You can use the CodeWarrior Bareboard Project Wizard to create new projects for tracing. The CodeWarrior IDE is a project-oriented interface. You must create a new project or open an existing project before using the Analysis tools. To create a new project, perform the following steps:

1. Select **File > New > ARMv8 Stationery**.

   The ARMv8 Project page appears.


ARMv8 Project

ARMv8 Project

Create an ARMv8 project

Project name:

[✓] Use default location

Location: C:/Users/b34930/workspace/ARMv8_SA

Available stationeries:

- ARMv8
  - Bare board
    - Hello World Assembly Project
    - Hello World C Project
    - Hello World C Static Library Project
    - Hello World C++ Project
    - Hello World C++ Static Library Project
  - Linux Application Debug
    - AArch64
      - Hello World C Project
      - Hello World C Static Library Project
      - Hello World C Shared Library Project
      - Hello World C++ Project
      - Hello World C++ Static Library Project
    - AArch32
      - Hello World C Project
      - Hello World C++ Project

Description: Creates a Hello World project in C language for linux application debug.

Figure 1. Create an ARMv8 Project page

2. On the **ARMv8 Project** page, enter the name of your project and also specify the location of the project.
3. Choose the ARMv8 project type. Select **Hello World C Project**.
4. Click **Finish**.

In CodeWarrior IDE, the project is created in the **Project Explorer** view.

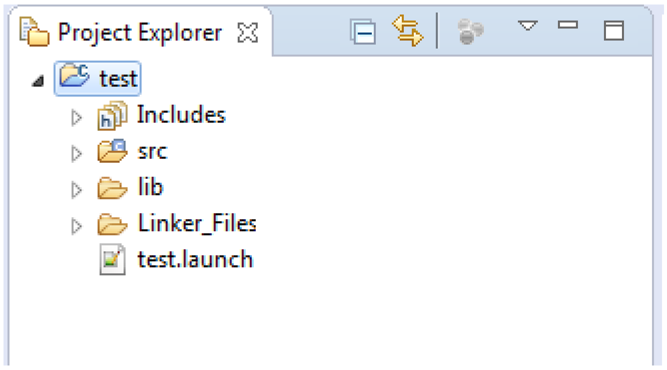


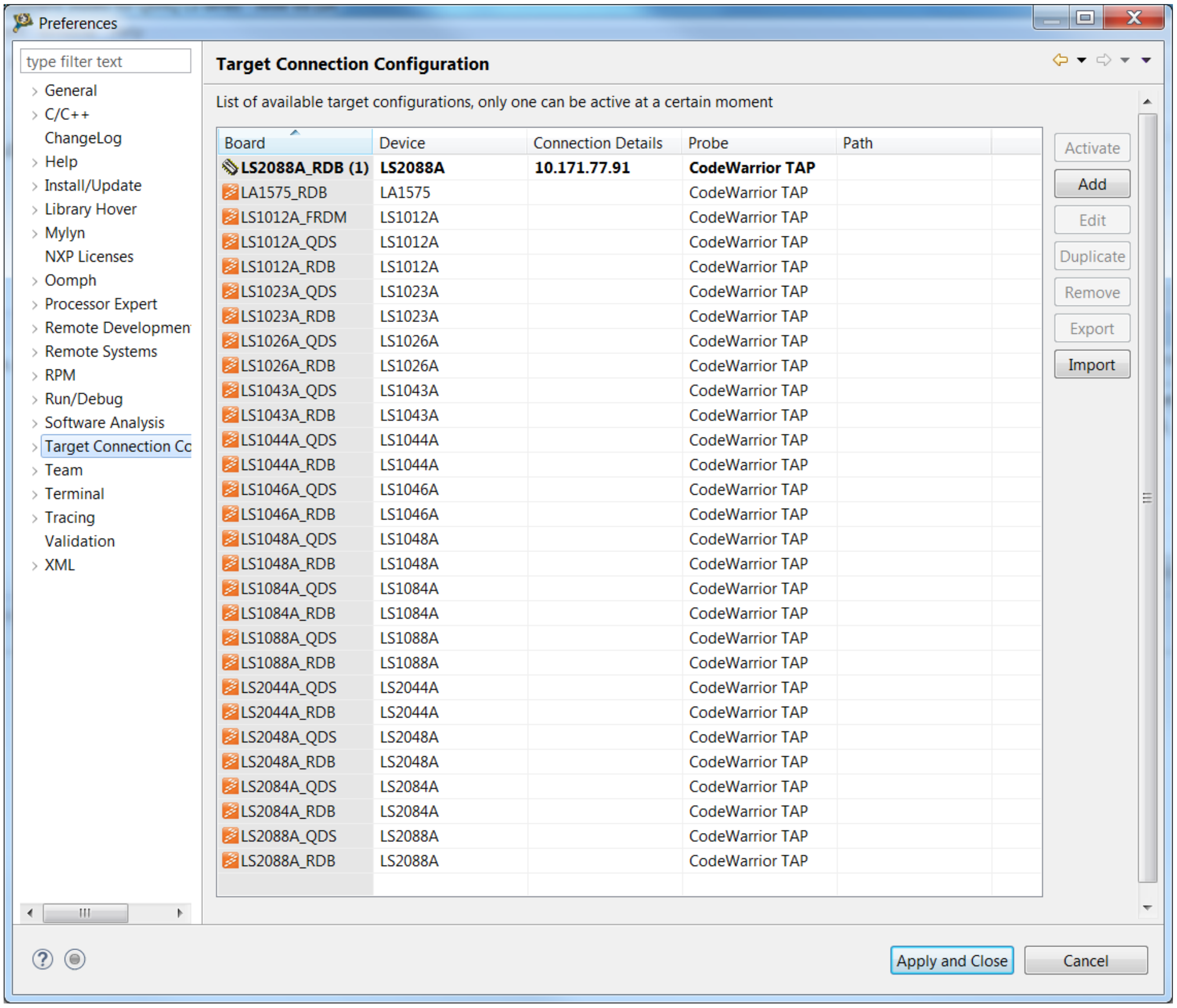
Figure 2. Project Explorer view

2.3 Configuring target

Target Connection Configuration (TCC) feature allows you to configure the probe and the target hardware.

Before debugging your application, you need to configure the **Target Connection Configuration (TCC)**. You can view all existing configurations, manage, and set the active configuration using the **Target Connection manager**. To access the Target Connection manager, select **Window > Preferences > Target Connection Configuration**. You will be able to see the Target Connection manager in the right panel of the **Preferences** window. Besides the possibility to configure the target connection through the preferences, you can access the same capabilities available in the Target Connection view. To access the Target Connections view, select **Window > Show View > Other > Debug >Target Connections**.

The Target Connections view lists the available target configurations.



Target Connection Configuration

List of available target configurations, only one can be active at a certain moment

Board	Device	Connection Details	Probe	Path
LS2088A_RDB (1)	**LS2088A**	**10.171.77.91**	**CodeWarrior TAP**	
LA1575_RDB	LA1575		CodeWarrior TAP	
LS1012A_FRDM	LS1012A		CodeWarrior TAP	
LS1012A_QDS	LS1012A		CodeWarrior TAP	
LS1012A_RDB	LS1012A		CodeWarrior TAP	
LS1023A_QDS	LS1023A		CodeWarrior TAP	
LS1023A_RDB	LS1023A		CodeWarrior TAP	
LS1026A_QDS	LS1026A		CodeWarrior TAP	
LS1026A_RDB	LS1026A		CodeWarrior TAP	
LS1043A_QDS	LS1043A		CodeWarrior TAP	
LS1043A_RDB	LS1043A		CodeWarrior TAP	
LS1044A_QDS	LS1044A		CodeWarrior TAP	
LS1044A_RDB	LS1044A		CodeWarrior TAP	
LS1046A_QDS	LS1046A		CodeWarrior TAP	
LS1046A_RDB	LS1046A		CodeWarrior TAP	
LS1048A_QDS	LS1048A		CodeWarrior TAP	
LS1048A_RDB	LS1048A		CodeWarrior TAP	
LS1084A_QDS	LS1084A		CodeWarrior TAP	
LS1084A_RDB	LS1084A		CodeWarrior TAP	
LS1088A_QDS	LS1088A		CodeWarrior TAP	
LS1088A_RDB	LS1088A		CodeWarrior TAP	
LS2044A_QDS	LS2044A		CodeWarrior TAP	
LS2044A_RDB	LS2044A		CodeWarrior TAP	
LS2048A_QDS	LS2048A		CodeWarrior TAP	
LS2048A_RDB	LS2048A		CodeWarrior TAP	
LS2084A_QDS	LS2084A		CodeWarrior TAP	
LS2084A_RDB	LS2084A		CodeWarrior TAP	
LS2088A_QDS	LS2088A		CodeWarrior TAP	
LS2088A_RDB	LS2088A		CodeWarrior TAP	

Figure 3. Target Connections view

To configure the target configuration in Target Connection Configurator, you need to select the debugged processor and the probe:

1. Choose the Target Connection based on the debugged core from the launch configuration file. Select any target from the **Target Configurations** list.
2. Click **Duplicate** to duplicate the predefined configuration. You can edit the duplicated configuration.

   The **Target Connection Configurator** window appears.
3. Select a connection type such as CodeWarrior TAP to connect to the target.
4. Specify the probe configuration details by selecting the Connection type, Hostname/IP, and Serial number for USB connection. Select the **Preserve Probe Configuration** option to hide all the CWTAP configurations. In this case, specify only the CCS server to access the CWTAP

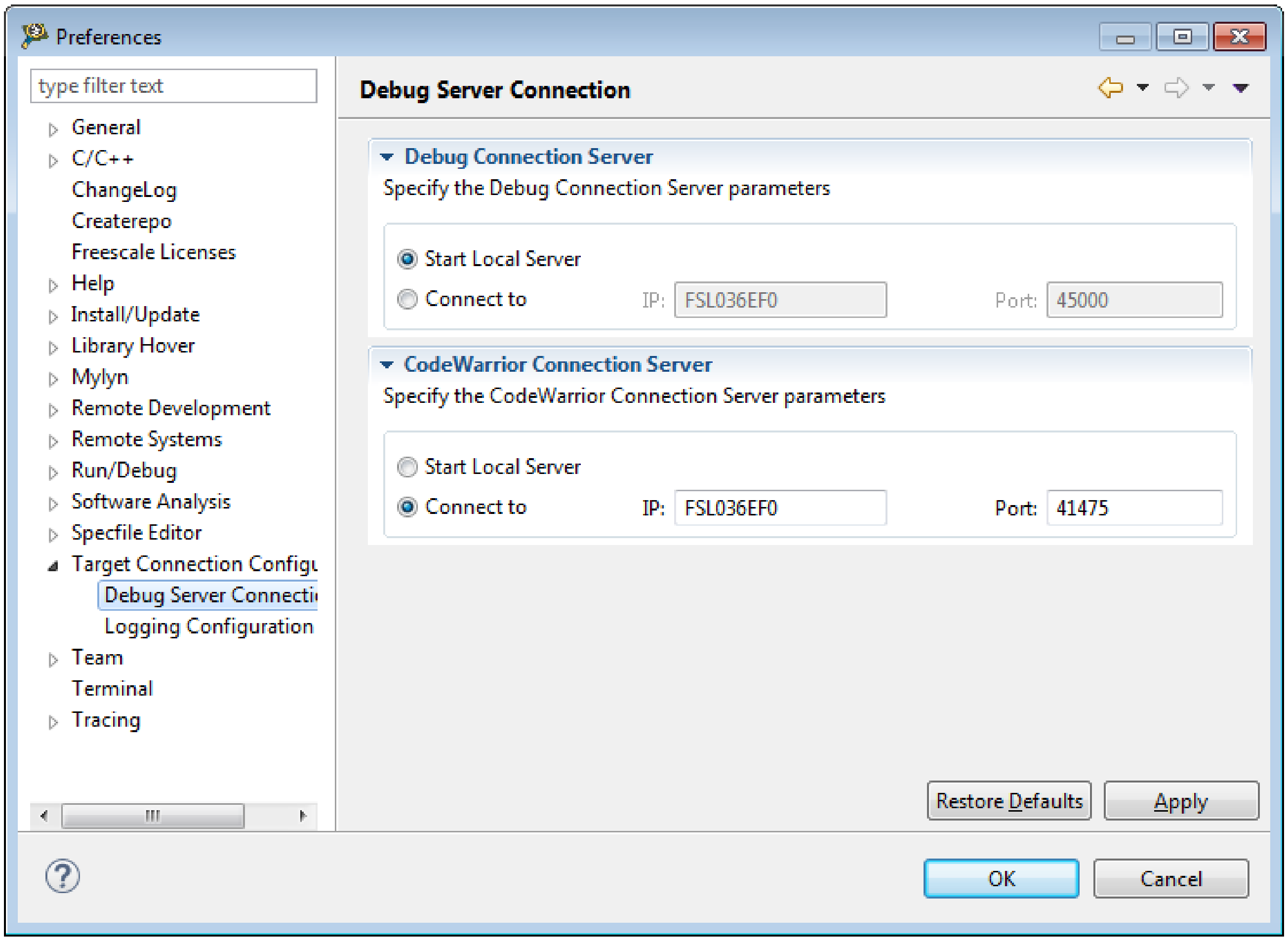
NOTE

The probes are available depending on the selected processor.

Each target connection configuration allows the user to select the type of connection to use with GTA: a local server or a remote connection to an already set up GTA server.

Start local server: In order to use a local GTA instance, the user must correctly specify the path to the GTA executable. TCC lets you start and stop an instance of the local GTA from the Target Connections Configurator dialog using the Start/Stop buttons in the Start local server area. In the case a GTA instance is already opened, TCC does not open a new instance but reuses the existing GTA. A scan is performed to determine if a GTA process is already running locally, which is transparent to the user and can be stopped on demand. TCC can start and stop an instance of the local GTA on user demand.

Connect to debug server: User can specify the server address and IP of an already running debug server. This is used for debugging and configuring the target. The GTA server can be configured from **Window** > **Preferences** > **Target Connection Configuration** > **Debug Server Connection**. You will be able to see the Debug Server Connection in the right panel of the **Preferences** window.



Debug Server Connection

Debug Connection Server

Specify the Debug Connection Server parameters

(● Start Local Server) ○ Connect to IP: FSL036EF0 Port: 45000

CodeWarrior Connection Server

Specify the CodeWarrior Connection Server parameters

○ Start Local Server (● Connect to) IP: FSL036EF0 Port: 41475

Figure 4. Debug Server Connection

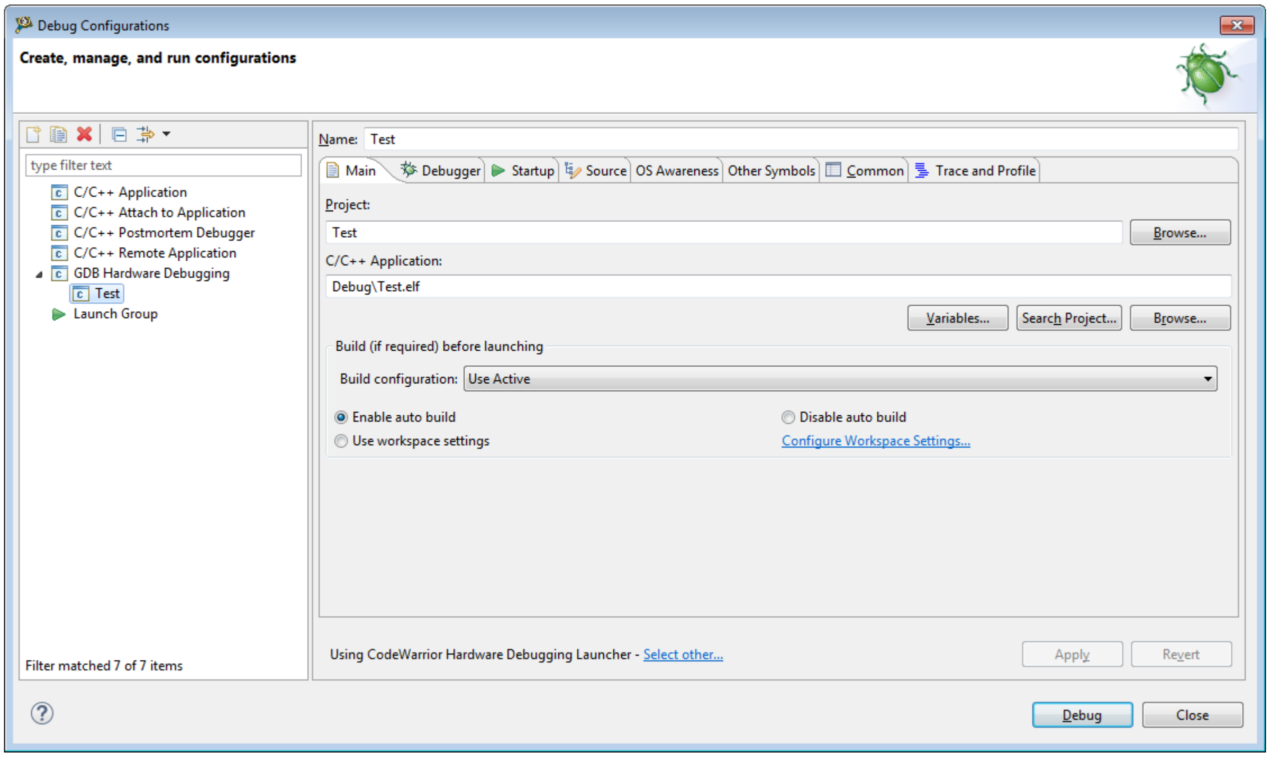
2.4 Configuring Debug Launcher

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

To define a trace configuration:

1. Right-click the selected project in the Project Explorer view and select **Build Project** from the context menu.
2. In the Project Explorer view, right-click on the project and select **Debug as** > **Debug Configurations** from the context menu.

   The **Debug Configurations** dialog appears.



Name: Test

Project: Test

C/C++ Application: Debug\Test.elf

Build (if required) before launching

Build configuration: Use Active

[✓ Enable auto build] Disable auto build

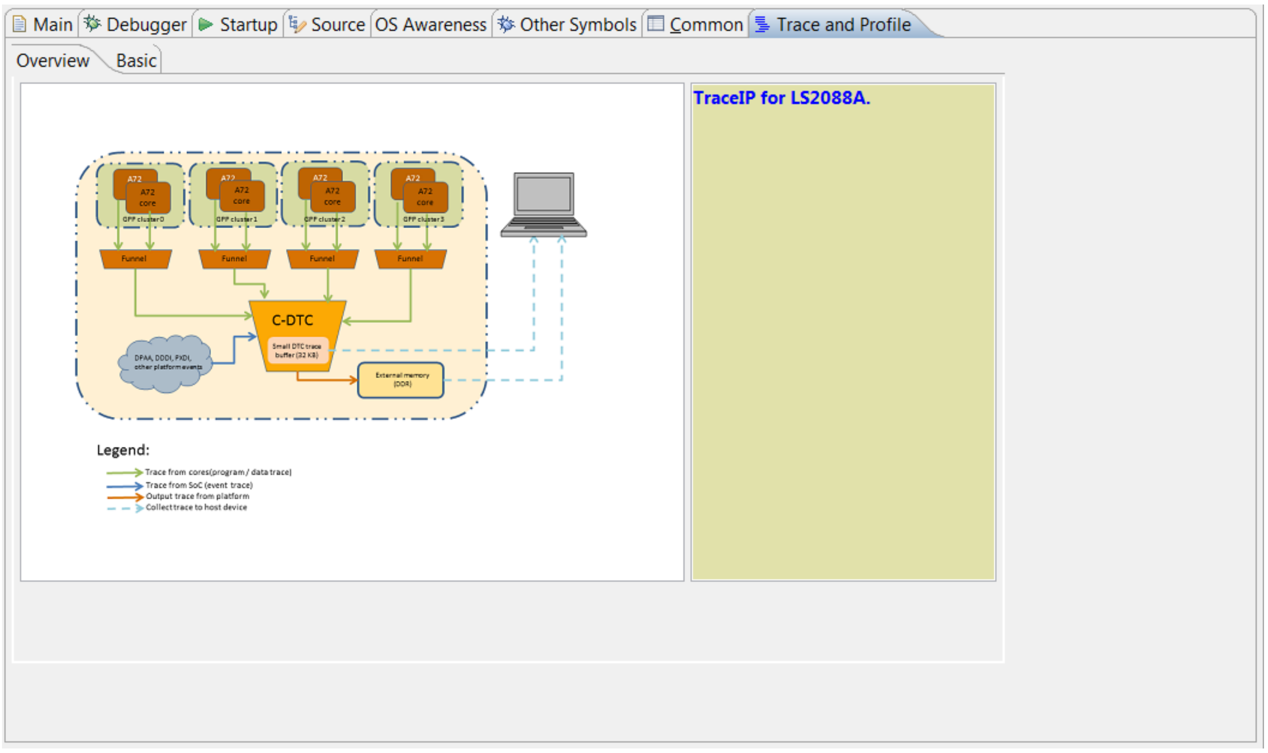
Use workspace settings

Using CodeWarrior Hardware Debugging Launcher - Select other...

Figure 5. Debug Configurations Dialog Box - Main Page

3. Expand the **GDB Hardware Debugging** group in the tree structure on the left, and select the launch configuration corresponding to the project you are using for example, Test.
4. Select *Test* in the **Project** field available in the **Main** tab of the **Debug Configurations** dialog box.
5. Click the **Trace and Profile** tab. This tab has the **Overview** and **Basic** page.

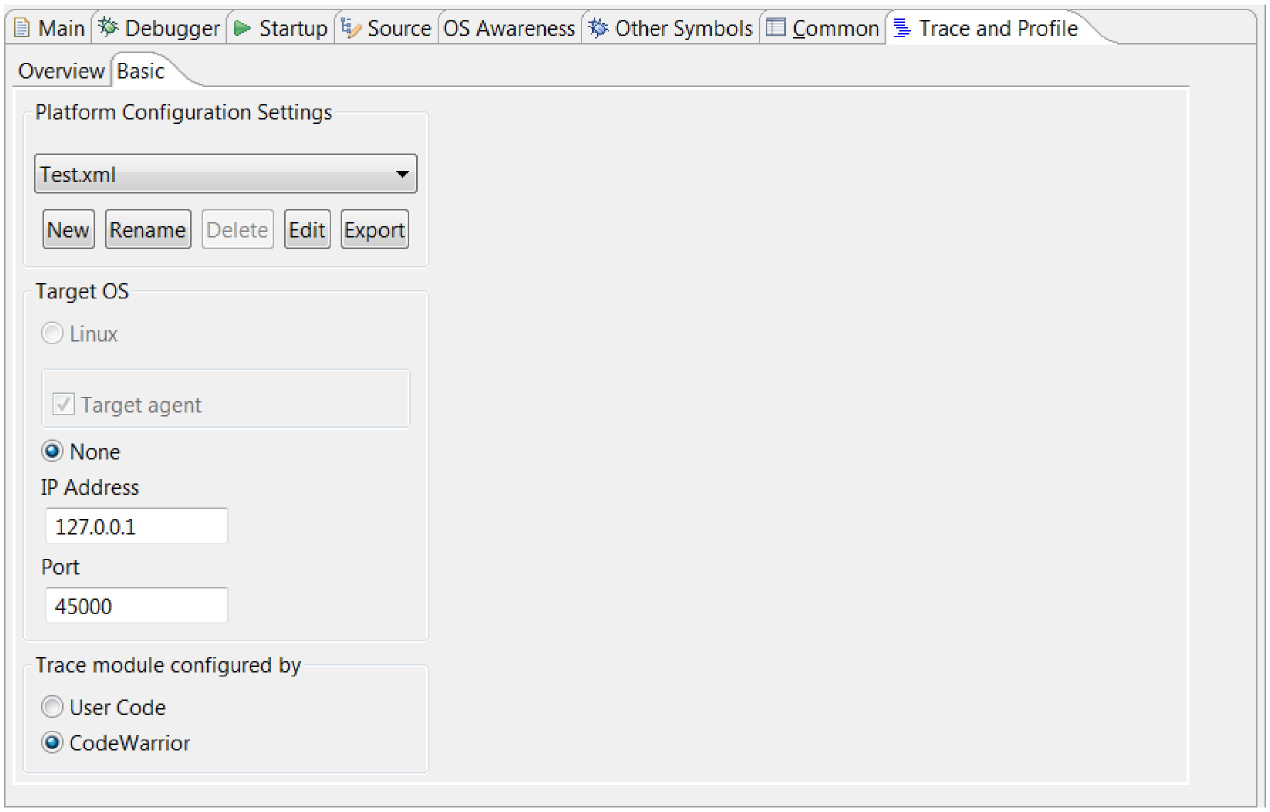
   The **Overview** page displays the flow diagram for collecting trace.


Main
Debugger
Startup
Source
OS Awareness
Other Symbols
Common
Trace and Profile
Overview
Basic
A72
A72 core
GPP cluster 0
GPP cluster 1
GPP cluster 2
GPP cluster 3
Funnel
C-DTC
Small DTC trace buffer (32 KB)
DPAA, DDDI, PXDI, other platform events
External memory (DDR)
Legend:
Trace from cores(program / data trace)
Trace from SoC (event trace)
Output trace from platform
Collect trace to host device

TraceIP for LS2088A.

Figure 6. Trace and Profile tab - Overview page

6. In the **Basic** page, select **Test.xml** file from **Platform Configuration Settings**.



Platform Configuration Settings

Test.xml

Target OS

Linux

Target agent

None

IP Address

127.0.0.1

Port

45000

Trace module configured by

User Code

CodeWarrior

Figure 7. Trace and Profile tab - Basic page

7. Specify the **IP address** and **Port** value to match the GTA Port value from Debug Server Connection. For more details, see Configuring target on page 6.
8. Select **CodeWarrior** option from the **Trace module configured by** list.
9. Click **Apply** to save the configuration.

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

This section includes the following topics:

- Configuring platform configuration file on page 11
- Display target accesses on page 14

2.4.1 Configuring platform configuration file

This section explains the steps to configure the platform configuration file.

Perform the following steps to configure the platform configuration file to collect trace data:

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

   The **Debug Configuration** dialog appears.
2. Click the **Trace and Profile** tab.
3. In the **Basic** page, select a platform configuration file.

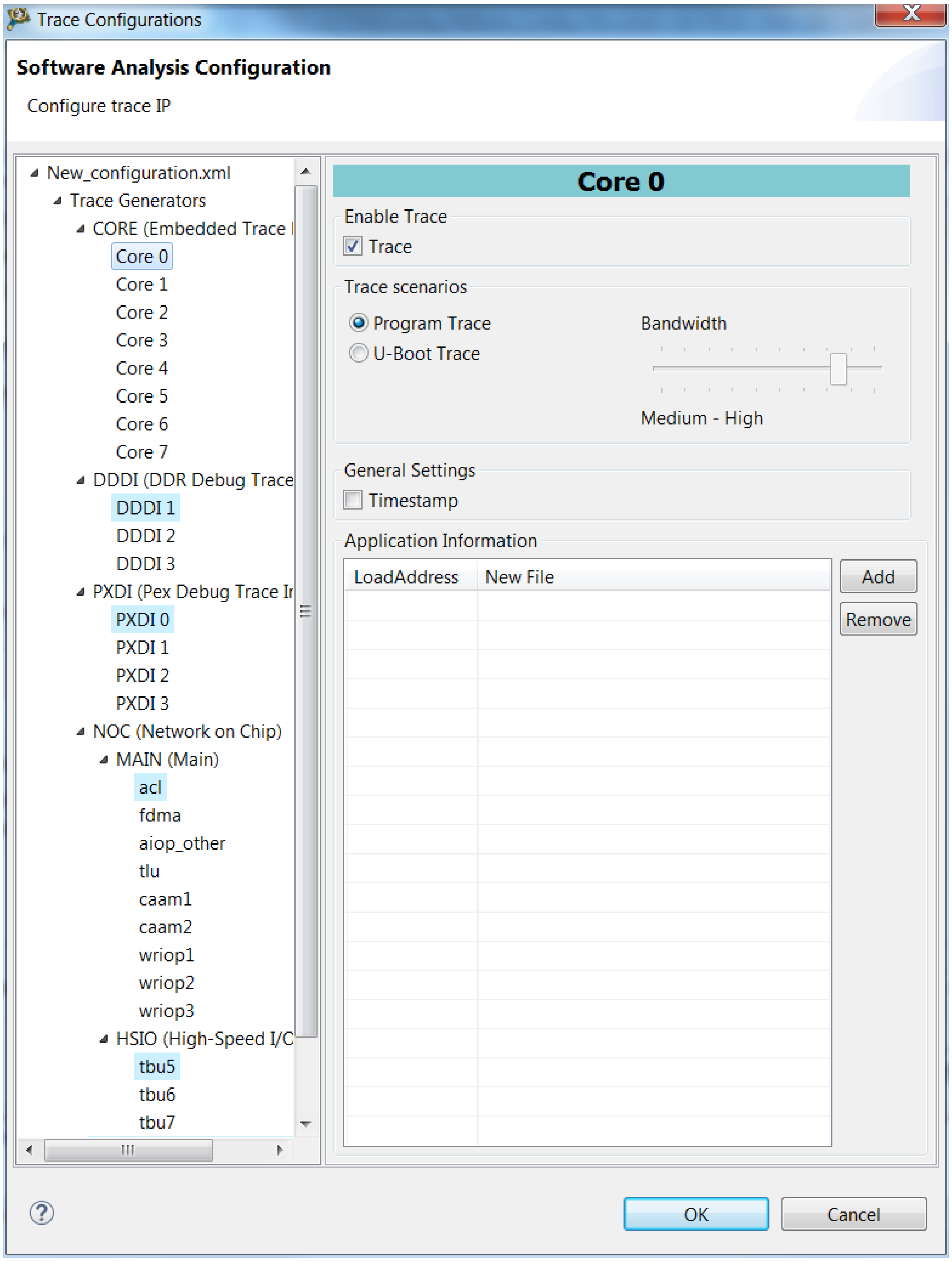
   The buttons in the Platform Configuration settings are:

Table 2. Platform Configuration settings buttons

Button	Description
New	Create a new configuration file. Select the processor as LS2088A or LS2048 in the **New Platform Configuration** dialog box.
Rename	Rename the selected configuration file.
Delete	Delete the selected configuration file.
Edit	Edit the configuration from the selected file.
Export	Export the selected configuration file.

4. Click **Edit** to configure an existing platform configuration file.

   The **Trace Configurations** dialog appears.


Trace Configurations

Software Analysis Configuration

Configure trace IP

- New_configuration.xml
  - Trace Generators
    - CORE (Embedded Trace
      - Core 0
      - Core 1
      - Core 2
      - Core 3
      - Core 4
      - Core 5
      - Core 6
      - Core 7
    - DDDI (DDR Debug Trace
      - DDDI 1
      - DDDI 2
      - DDDI 3
    - PXDI (Pex Debug Trace Ir
      - PXDI 0
      - PXDI 1
      - PXDI 2
      - PXDI 3
    - NOC (Network on Chip)
      - MAIN (Main)
        - acl
        - fdma
        - aiop_other
        - tlu
        - caam1
        - caam2
        - wriop1
        - wriop2
        - wriop3
      - HSIO (High-Speed I/C
        - tbu5
        - tbu6
        - tbu7

Core 0

Enable Trace

[✓] Trace

Trace scenarios

(●) Program Trace
( ) U-Boot Trace

Bandwidth: Medium - High

General Settings

[ ] Timestamp

Application Information

LoadAddress	New File

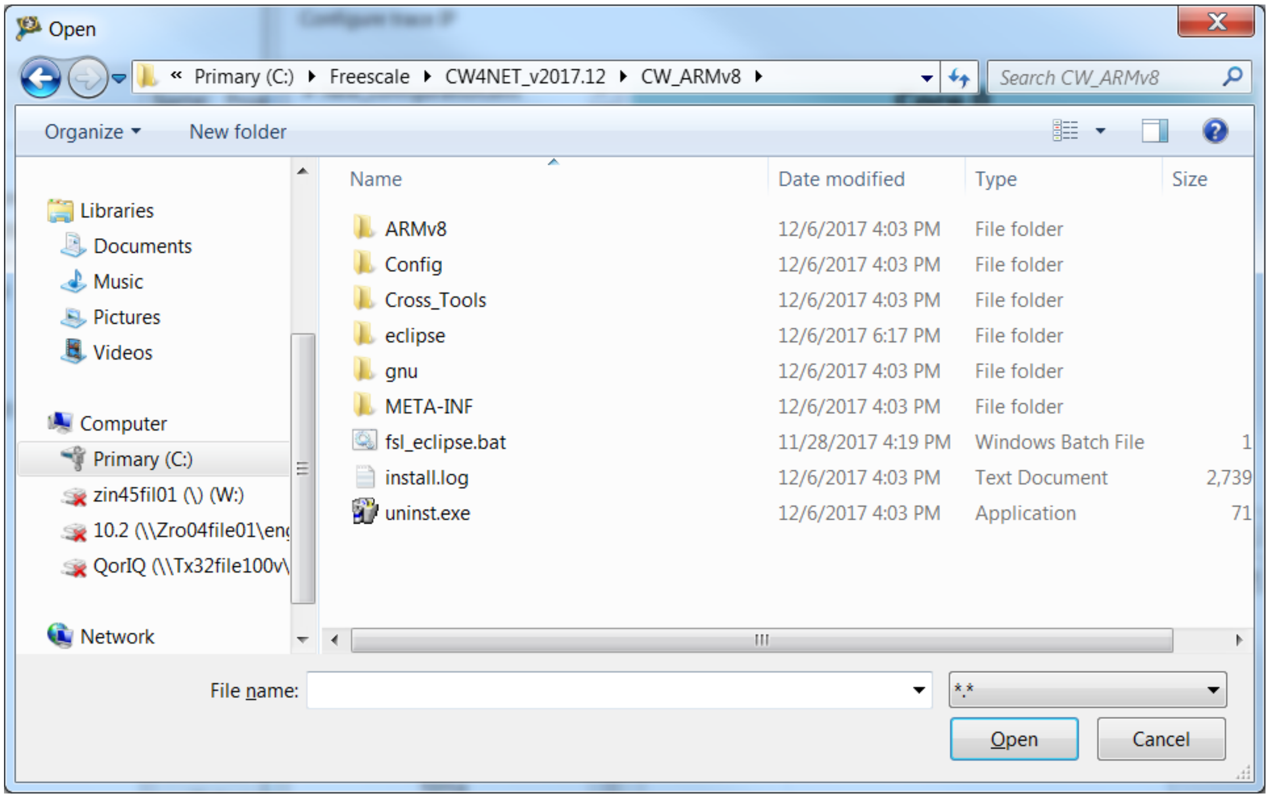
Figure 8. Trace Configurations window

5. Select a trace generator type group from the **Trace Generators** available in the left panel, for example select Core 0.

   The respective settings for Core 0 trace generator will appear.
6. Select the desired trace scenario from the **Trace Scenarios** group. You can select any of the below listed scenerio:
   - Program Trace - Inspects the program execution flow.
   - U-Boot Trace - Monitors the primary boot loader used to package the instructions to boot the device's operating system kernel.
7. Select **Timestamp** checkbox in the **General Settings** panel to enable timestamp. Timestamping is useful for correlating multiple trace sources. Timestamping is performed by the insertion of timestamp packets into the trace streams. It displays the value of platform global timestamp generator (64-bit wide).
8. Click **Add** to include the executable ELF file from the associated project. This ELF file will be used to interpret the collected trace data and provide the symbolic-debug information.

   A row gets added in the **Application Information** area.
9. Click the Ellipsis button.

   The browse for executable file dialog box appears.



Name	Date modified	Type	Size
ARMv8	12/6/2017 4:03 PM	File folder	
Config	12/6/2017 4:03 PM	File folder	
Cross_Tools	12/6/2017 4:03 PM	File folder	
eclipse	12/6/2017 6:17 PM	File folder	
gnu	12/6/2017 4:03 PM	File folder	
META-INF	12/6/2017 4:03 PM	File folder	
fsl_eclipse.bat	11/28/2017 4:19 PM	Windows Batch File	1
install.log	12/6/2017 4:03 PM	Text Document	2,739
uninst.exe	12/6/2017 4:03 PM	Application	71

Figure 9. Select the executable file window

10. Click **Open**.

    The executable file is added.
11. Click **OK**.
12. Click **Export** button in the **Basic** page > **Platform Configuration Settings** area if you want to export your platform configuration file to xml.

    The **Export platform Configuration to xml** dialog appears.

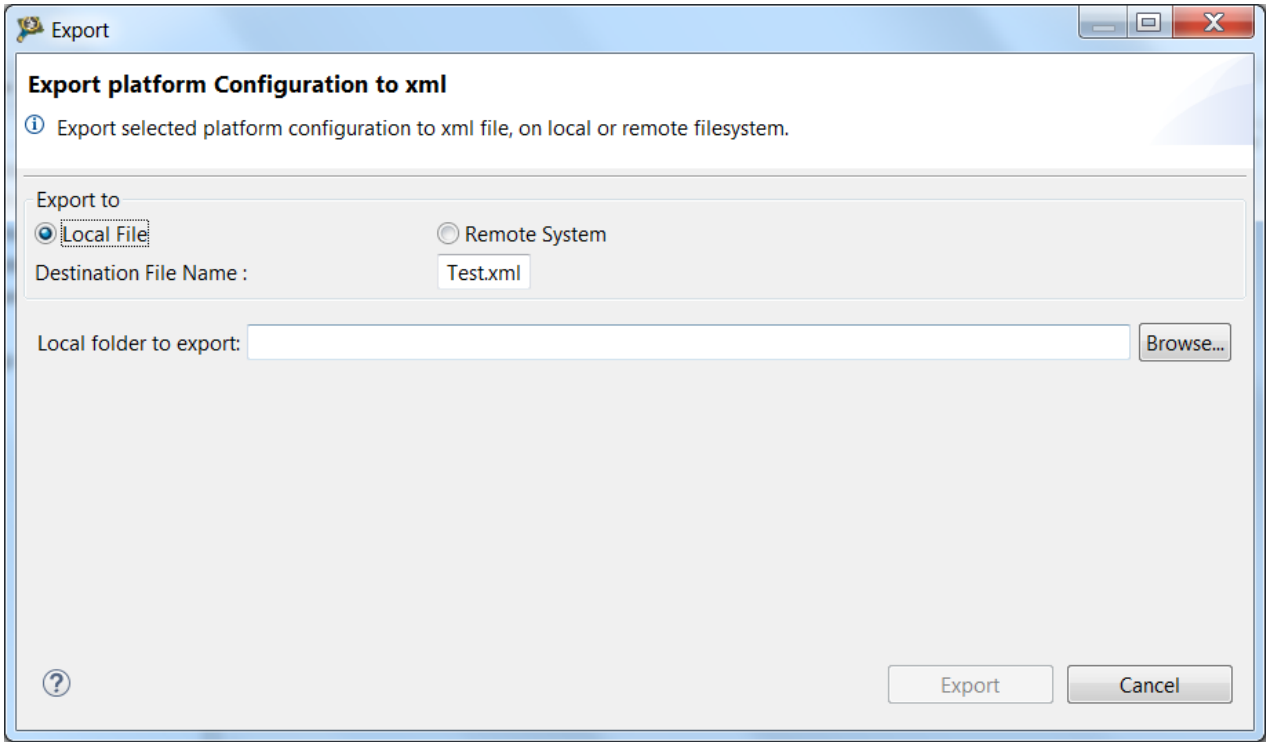

Export
Export platform Configuration to xml
Export selected platform configuration to xml file, on local or remote filesystem.
Export to
Local File
Remote System
Destination File Name :
Test.xml
Local folder to export:
Browse...
Export
Cancel

Figure 10. Export platform Configuration to xml

13. Browse to the location where you want to save the exported file and click **Export**.

2.4.2 Display target accesses

The Target Agent (TA) from **Basic** tab, specifies the way in which the tool connects to the target.

Currently, there is the GTA protocol available in CodeWarrior and LTA Logger which is a server application available in *<CWInstallDir>/ARMv8/sa_ls/bin/ls2.lta.log.server(.exe)*. This can be executed from a console and it displays the target accesses made by the configurator, at stdout. This is an off-line tool that displays which settings are made, which registers are written while configuring trace.

From the **Basic** tab, the IP Address represents the address of the machine where the server is running and the port used to communicate between the server and the configurator.

If a communication protocol is available (the *ls2.lta.log.server* is running), the TC2 Configurator accesses the target and performs the writes to the configuration registers.

Example of stdout for TC2 Configurator:

```
write_mem 0x700084fb0 data=0xc5acce55 length=0x4
write_mem 0x700084fb0 data=0xc5acce55 length=0x4
write_mem 0x700004000 data=0x00000001 length=0x4
write_mem 0x700004fb0 data=0xc5acce55 length=0x4
write_mem 0x700004000 data=0x80000060 length=0x4
read_mem  0x700004020 data=0x80000060 length=0x4
write_mem 0x700004020 data=0x00000000 length=0x4
write_mem 0x70000403c data=0x70000000 length=0x4
write_mem 0x7000040ac data=0x90000000 length=0x4
write_mem 0x7000040b0 data=0x00000000 length=0x4
write_mem 0x7000040b8 data=0x00000040 length=0x4
write_mem 0x7000040b4 data=0x93000000 length=0x4
write_mem 0x700048fb0 data=0xc5acce55 length=0x4
write_mem 0x700048000 data=0x00000403 length=0x4
write_mem 0x700048004 data=0x00000008 length=0x4
write_mem 0x701040fb0 data=0xc5acce55 length=0x4
write_mem 0x701040300 data=0x00000000 length=0x4
write_mem 0x701040004 data=0x00000000 length=0x4
write_mem 0x701040010 data=0x00000040 length=0x4
write_mem 0x701040020 data=0x00000000 length=0x4
write_mem 0x701040024 data=0x00000000 length=0x4
write_mem 0x701040040 data=0x00000001 length=0x4
write_mem 0x701040034 data=0x0000000a length=0x4
write_mem 0x701040030 data=0x00000000 length=0x4
write_mem 0x701040080 data=0x00000201 length=0x4
write_mem 0x701040084 data=0x00000000 length=0x4
write_mem 0x701040088 data=0x00000000 length=0x4
write_mem 0x701040004 data=0x00000001 length=0x4
write_mem 0x70000cfb0 data=0xc5acce55 length=0x4
write_mem 0x70000c000 data=0x00000407 length=0x4
write_mem 0x70000c004 data=0x00000088 length=0x4
write_mem 0x70000dfb0 data=0xc5acce55 length=0x4
write_mem 0x70000d000 data=0x00000403 length=0x4
write_mem 0x70000d004 data=0x00000008 length=0x4
write_mem 0x700024fb0 data=0xc5acce55 length=0x4
write_mem 0x700024004 data=0x00000200 length=0x4
write_mem 0x700024004 data=0x00000000 length=0x4
write_mem 0x700028000 data=0x00000000 length=0x4
read_mem  0x700024000 data=0x00000000 length=0x4
```

Each line describes a target access that the TC2 Configurator is performing including the Physical Address, the register size (length) and the actual data is written as an unsigned integer value (data).

Chapter 3 Trace Commander View

This section explains how to use the Trace Commander feature of CodeWarrior for ARMv8.

This section has following topics:

- Overview on page 16
- Configuring and collecting trace using Trace Commander view on page 17

3.1 Overview

Trace Commander feature uploads the trace and configures the modules included in the target architecture.

For example, for ARMv8 LS2088A, the modules are ETM4.0 cores (ETM - Embedded Trace Macrocell 4.0), Big Funnel (that makes the link between the active trace generator (active core) and the central DTC module), PXDIn (n from 0 to 3) modules, PXDI Funnel and SoC Funnel (makes the link to central DTC module), DDDIn (n form 1 to 3) modules, NXC, HSIO Funnel, Main NoC and HSIO NoC observers (each of these two observers has a number of trace generators probe) and STM. It also configures the C-DTC collector.

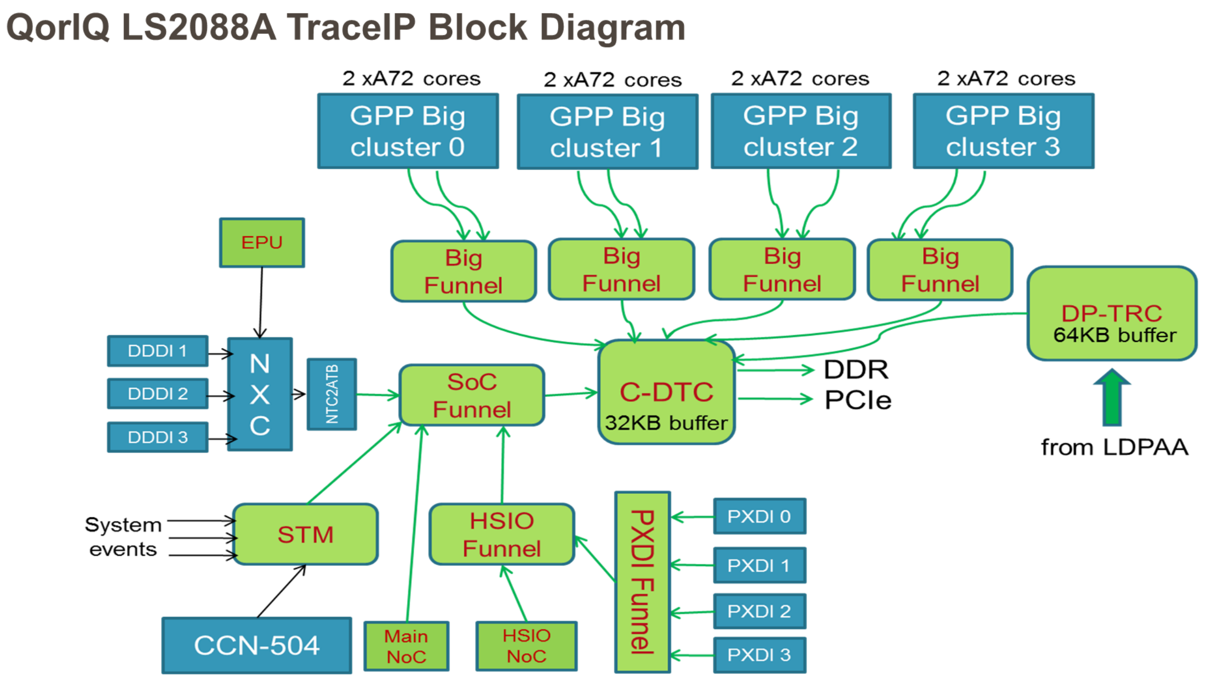


Figure 11. Core Trace path

The C-DTC stores and forwards trace data using a dedicated RAM buffer. This reduces trace loss by absorbing spikes in trace data. The destination of the raw trace is either the internal trace buffer of the C-DTC module or a user defined buffer in DDR. Trace Commander is deployed as an Eclipse Editor view.

3.2 Configuring and collecting trace using Trace Commander view

Trace Commander view is used to manage the trace collection data. The view is used to perform actions such as, start, stop trace on different trace generators, manual upload or trace configuration for data trace collection.

Trace Commander is based on an xml platform configuration file, which is responsible with configuring all the modules included in the target architecture. Hence, the Trace Commander tool is used to ease the task of configuring and uploading trace.

Trace Commander displays all modules from a platform configuration file. Perform the following steps in order to collect multi core trace data using Trace Commander:

1. Select **Window > Show View > Other > Software Analysis > Trace Commander**.

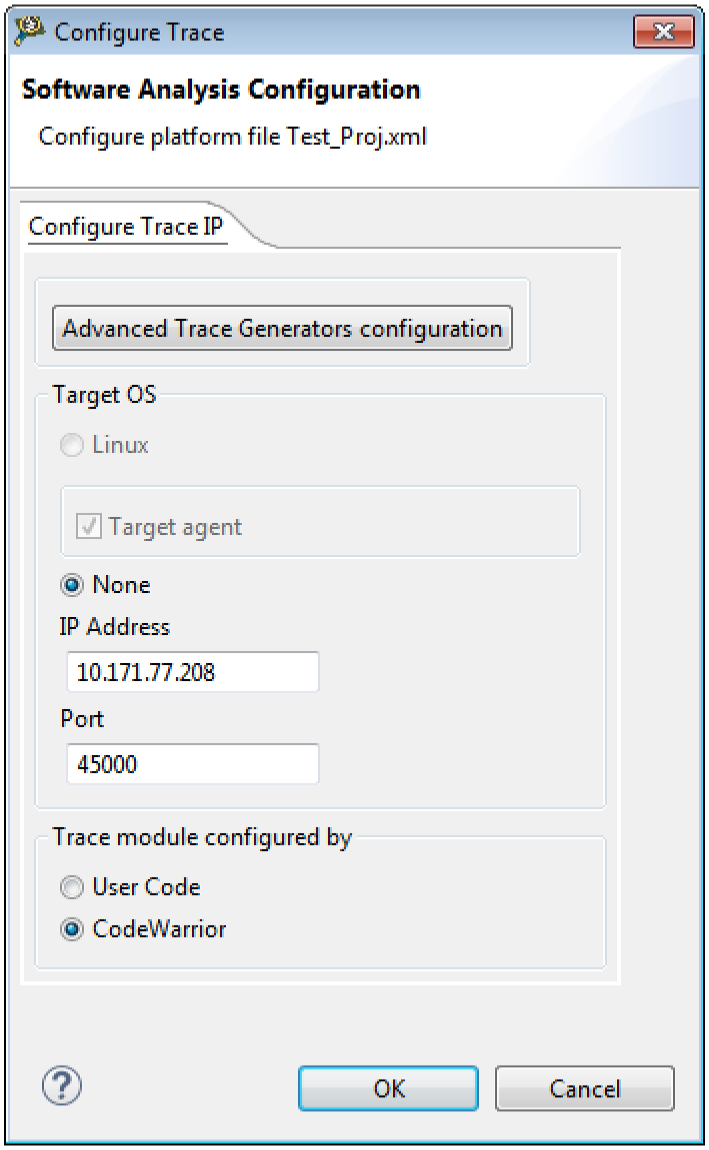
   The **Trace Commander**view is displayed.

2. Choose the **platform configuration file**, for example select **test.xml**.

   The Trace Commander view displays all trace generators and trace buffers available in the selected platform file. The colors suggest their state, available or not for collecting trace: green if the trace generator is enabled and grey if trace generator is disabled. For trace buffers, the green color determines the trace collector.

3. Double click on one of the **trace generators** or **trace buffers** to configure it. You can also use **Config** button.

   The **Trace Configuration** window is displayed.



Configure Trace

Software Analysis Configuration

Configure platform file Test_Proj.xml

Configure Trace IP

Advanced Trace Generators configuration

Target OS

( ) Linux

[✓] Target agent

(●) None

IP Address: 10.171.77.208

Port: 45000

Trace module configured by

( ) User Code

(●) CodeWarrior

OK | Cancel

Figure 12. Trace Configuration window

4. Click one of the **trace buffer** to collect trace, for example select **DTC**. They are exclusive; you can collect trace in one buffer per running.
5. Click **Connect** button.

   While starting the session, the target is configured using the selected platform file. This will be attached to the active debug session, **GTA** (see Configuring Debug Launcher on page 8). The configuration is applied on the whole platform. Now the trace buffer selection and configuration becomes unavailable, also the trace generators configuration.

   On stop session, the trace stream is interrupted and the file gets inactive. Hence, the trace buffer selection and configuration and trace generators configuration becomes available.
6. Click the **trace generator** or a **trace generators group** to **start** or **stop** trace collection on the selected module. For example, click **PXDI group** to stop all PXDI trace collection and click on **STM** trace to enable it. The Trace Commander view will update each trace generator state.

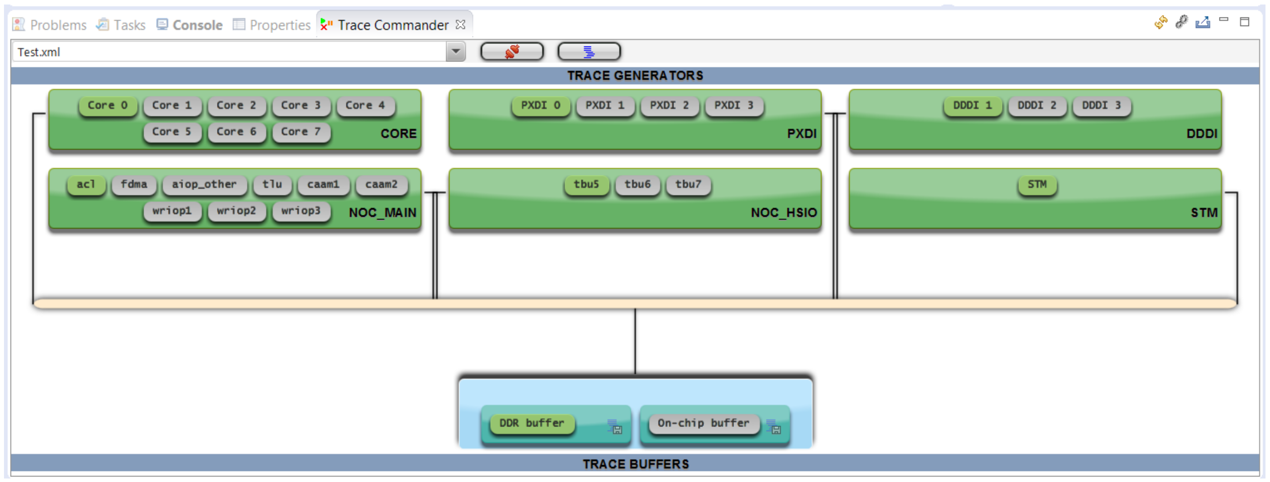


Figure 13. Trace Commander view - Start/Stop trace generators

7. Click **upload**, available on the trace buffer, to save the data trace collected. For example, on **DTC**, click **upload button**. Data trace can be saved only on connection with the active debug session, GTA or LTA.

NOTE

It is recommended to suspend the generator core or the target while collecting core trace.

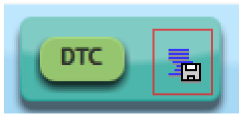


Figure 14. Selected trace buffer and upload button

Trace data is saved in ***.AnalysisData*** folder from the application's workspace.

The toolbar options from the Trace Commander view allows you to perform:

- Refresh : refreshes the view .
- Settings : opens the **Platform Configuration Settings** window. This lists all available platform configuration files from `<workspace dir>/.metadata/.plugins/com.freescale.sa/platformConfig`. You can add, remove, or duplicate a platform configuration file.

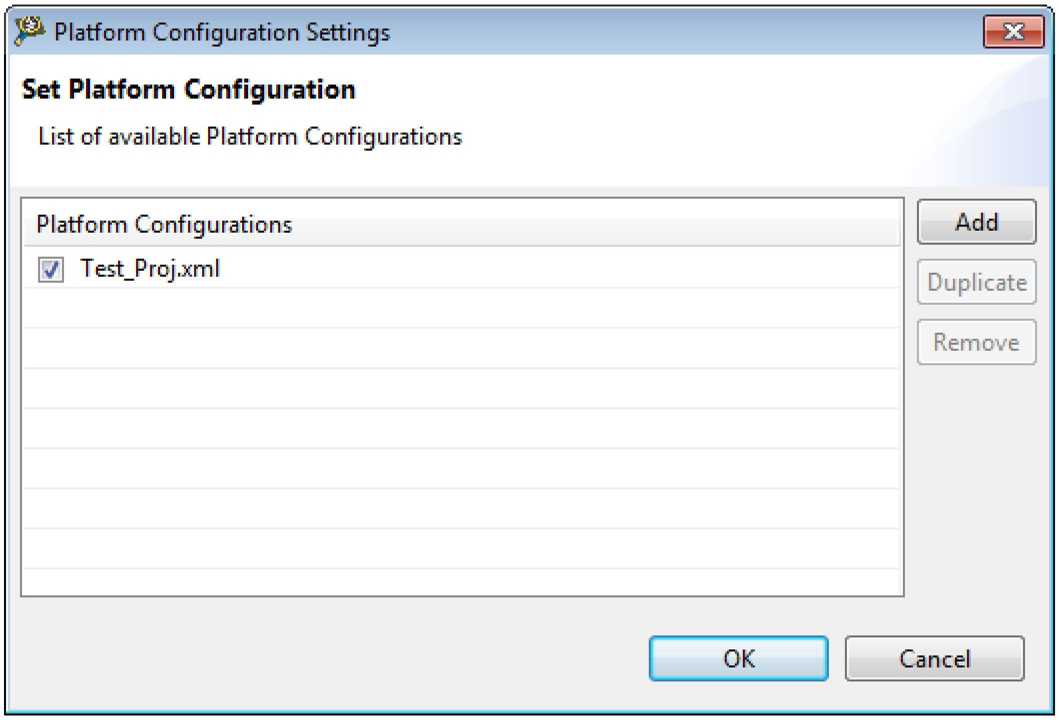

Platform Configuration Settings
Set Platform Configuration
List of available Platform Configurations
Platform Configurations
Test_Proj.xml
Add
Duplicate
Remove
OK
Cancel

Figure 15. Platform Configuration window

- Export : exports the current displayed configuration file.

Chapter 4 View Trace Data

This chapter describes how to display the collected trace data and how to view the decoded trace data.

This chapter includes:

- Analysis Results view on page 21
- Import trace data on page 39

4.1 Analysis Results view

The Analysis Results view shows and manages collected traces.

The main features of the Analysis Results view are: organizing logic of trace and profiling results, enable/disable links towards profiling viewers, save/rename/delete results, expand all/collapse all results, and refreshing the view.

From the menu bar, select **Window > Show View > Other > Software Analysis > Analysis Results** to view Analysis Results view.

NOTE

You can also press *Alt+Shift+Q, Q* to open the Other Views dialog box and select **Software Analysis --> Analysis Results** view.

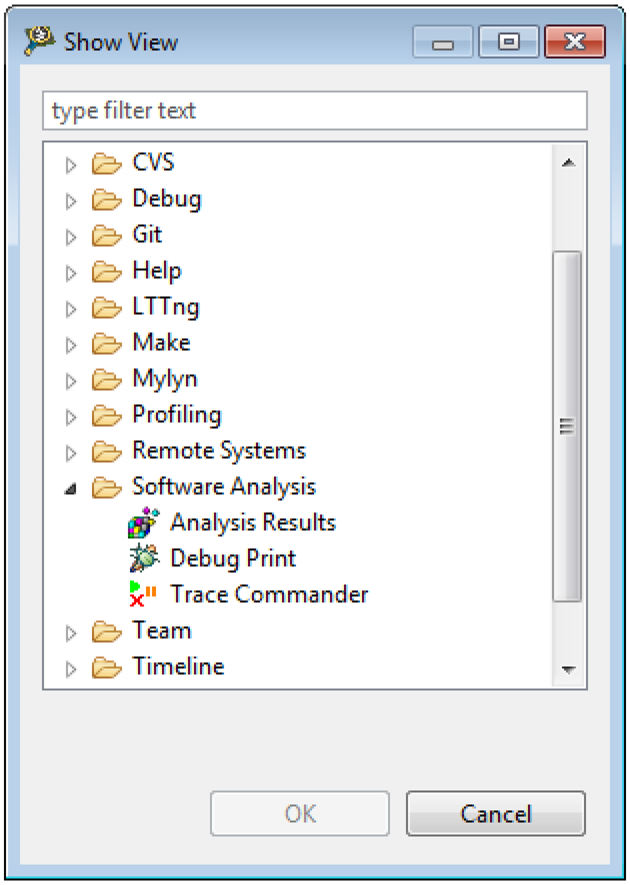
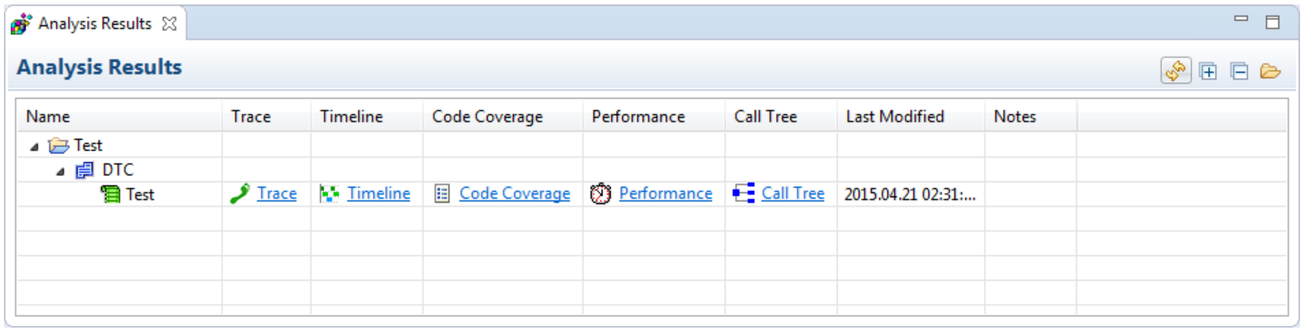


Figure 16. Show View dialog box

The Analysis Results view pops up and gets focus after successfully collecting a trace. It does not refresh automatically. Click the Refresh button in the Analysis Results view to update the links to view the profiling trace data. The following figure shows the Analysis Results view.



Name	Trace	Timeline	Code Coverage	Performance	Call Tree	Last Modified	Notes
Test							
DTC							
Test	Trace	Timeline	Code Coverage	Performance	Call Tree	2015.04.21 02:31:...	

Figure 17. Analysis Results view

The data in results table is organized (from top to bottom) in the following manner:

- Platform Configuration: Name of the platform configuration file used to collect the trace result.
- Data streams: The data source from where the trace was collected. For example, it can be DDR or DTC.
- Result sets: Lists the collected trace data.

The Analysis Results view allows to perform the following actions:

Table 3. Analysis Results Toolbar

Button	Description
Refresh	Refreshes the view and rescans all output folders for trace and analysis results
Expand all	Expands all nodes in the tree
Collapse all	Collapses all nodes in the tree
Select custom results folder	Allows to select a custom folder to scan for trace results

Apart from the above described toolbar actions, the context menu also allows to perform the following actions for a selected trace result:

- **Rename**: To rename the selected trace result
- **Save**: To save the selected trace result. The save function creates a copy of the selected trace results with an index appended to the name. By default when a new trace is collected it receives the name of the project from where it was collected. If there is already a trace with that name, it is overwritten.
- **Delete**: To delete the selected trace result.

The Analysis Results view can also show the results from more folders. Each platform configuration available in `.metadata/.plugins/com.freescale.sa/platformConfig/` folder is checked for the path specified towards the set of results. All sets of trace and profiling results found in those paths will appear in the Analysis Results view. All the trace results available in `workspace/.AnalysisData` are displayed in Analysis Results view.

To configure the view to display trace data of another custom results folder that does not belong to any platform configuration inside the workspace:

1. Click **Select custom results folder** option available on the Analysis Results view toolbar.

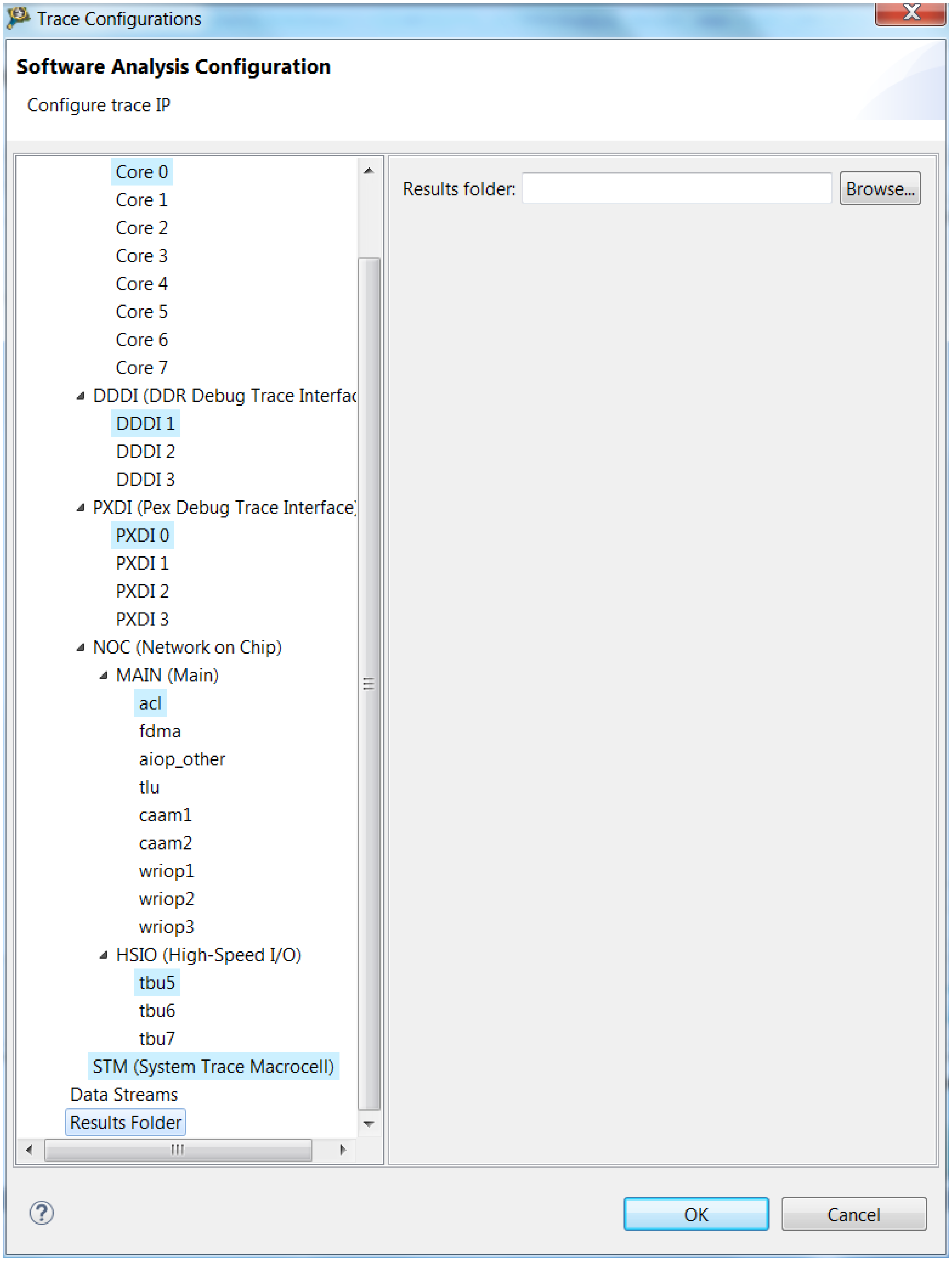
   The **Custom results folder** dialog gets displayed.
2. Click **Add**.

The **SA Results Folder Selection** dialog opens.

3. Browse the location of the required custom folder.
4. Click **Ok**.

   The path to the custom folder for which the trace needs to be collected gets added under the **Path** field.

To configure the results folder for a platform configuration, specify the **Results folder** location available in the **Trace Configurations** dialog box as shown in the below figure. The collected trace using the Platform configuration will be saved in the specified folder.



Trace Configurations

Software Analysis Configuration

Configure trace IP

- Core 0
- Core 1
- Core 2
- Core 3
- Core 4
- Core 5
- Core 6
- Core 7
- DDDI (DDR Debug Trace Interfac
  - DDDI 1
  - DDDI 2
  - DDDI 3
- PXDI (Pex Debug Trace Interface)
  - PXDI 0
  - PXDI 1
  - PXDI 2
  - PXDI 3
- NOC (Network on Chip)
  - MAIN (Main)
    - acl
    - fdma
    - aiop_other
    - tlu
    - caam1
    - caam2
    - wriop1
    - wriop2
    - wriop3
  - HSIO (High-Speed I/O)
    - tbu5
    - tbu6
    - tbu7
- STM (System Trace Macrocell)
- Data Streams
- Results Folder

Results folder:

Figure 18. Trace configuration dialog box

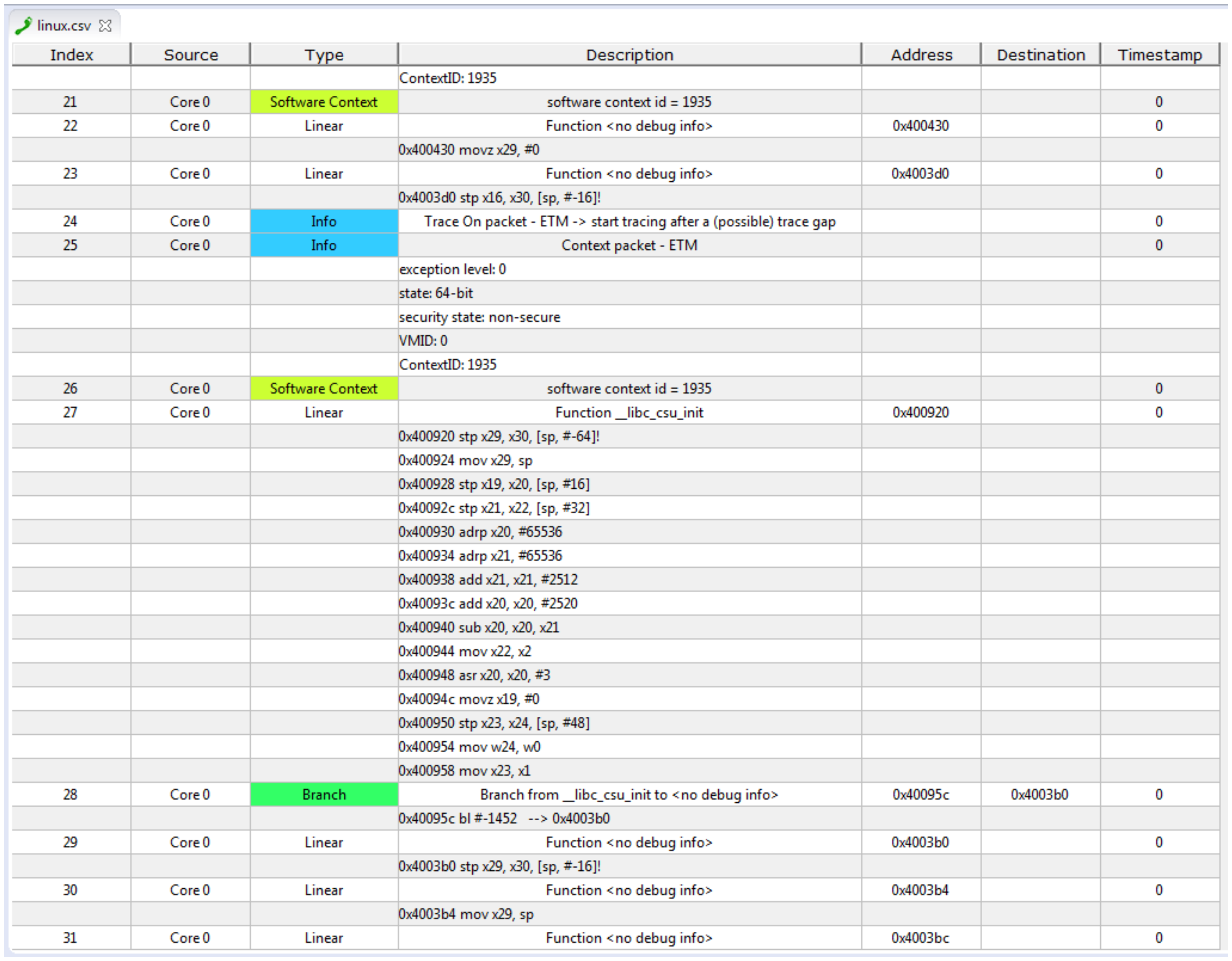
To view the collected trace data, click on the links of profiling results from the Analysis Results View:

- Viewing Trace data on page 25
- Viewing Performance data on page 26
- Viewing Timeline data on page 29
- Viewing Call Tree data on page 33
- Viewing Code Coverage data on page 35

4.1.1 Viewing Trace data

Click the **Trace** link from the Analysis Results view to display the trace data in the **Trace** viewer.

The following figure shows the fully decoded trace data.



linux.csv

Index	Source	Type	Description	Address	Destination	Timestamp
			ContextID: 1935			
21	Core 0	Software Context	software context id = 1935			0
22	Core 0	Linear	Function <no debug info>	0x400430		0
			0x400430 movz x29, #0			
23	Core 0	Linear	Function <no debug info>	0x4003d0		0
			0x4003d0 stp x16, x30, [sp, #-16]!			
24	Core 0	Info	Trace On packet - ETM -> start tracing after a (possible) trace gap			0
25	Core 0	Info	Context packet - ETM			0
			exception level: 0			
			state: 64-bit			
			security state: non-secure			
			VMID: 0			
			ContextID: 1935			
26	Core 0	Software Context	software context id = 1935			0
27	Core 0	Linear	Function __libc_csu_init	0x400920		0
			0x400920 stp x29, x30, [sp, #-64]!			
			0x400924 mov x29, sp			
			0x400928 stp x19, x20, [sp, #16]			
			0x40092c stp x21, x22, [sp, #32]			
			0x400930 adrp x20, #65536			
			0x400934 adrp x21, #65536			
			0x400938 add x21, x21, #2512			
			0x40093c add x20, x20, #2520			
			0x400940 sub x20, x20, x21			
			0x400944 mov x22, x2			
			0x400948 asr x20, x20, #3			
			0x40094c movz x19, #0			
			0x400950 stp x23, x24, [sp, #48]			
			0x400954 mov w24, w0			
			0x400958 mov x23, x1			
28	Core 0	Branch	Branch from __libc_csu_init to <no debug info>	0x40095c	0x4003b0	0
			0x40095c bl #-1452 --> 0x4003b0			
29	Core 0	Linear	Function <no debug info>	0x4003b0		0
			0x4003b0 stp x29, x30, [sp, #-16]!			
30	Core 0	Linear	Function <no debug info>	0x4003b4		0
			0x4003b4 mov x29, sp			
31	Core 0	Linear	Function <no debug info>	0x4003bc		0

Figure 19. Trace Viewer

The below table explains various fields of the trace data:

Table 4. Trace data fields description

Field	Description
Index	Displays the order number of the instructions.

Table continues on the next page...

Table 4. Trace data fields description (continued)

Field	Description
Source	Displays source of trace event.
Type	Displays type of trace event.
Description	Displays description of trace event.
Address	Displays the starting address of the target function.
Destination	Displays the end address of the target function.
Timestamp	Displays the value of platform global timestamp generator (64-bit wide). Time stamping is useful for correlating multiple trace sources.

The following table lists the Trace viewer events and their description:

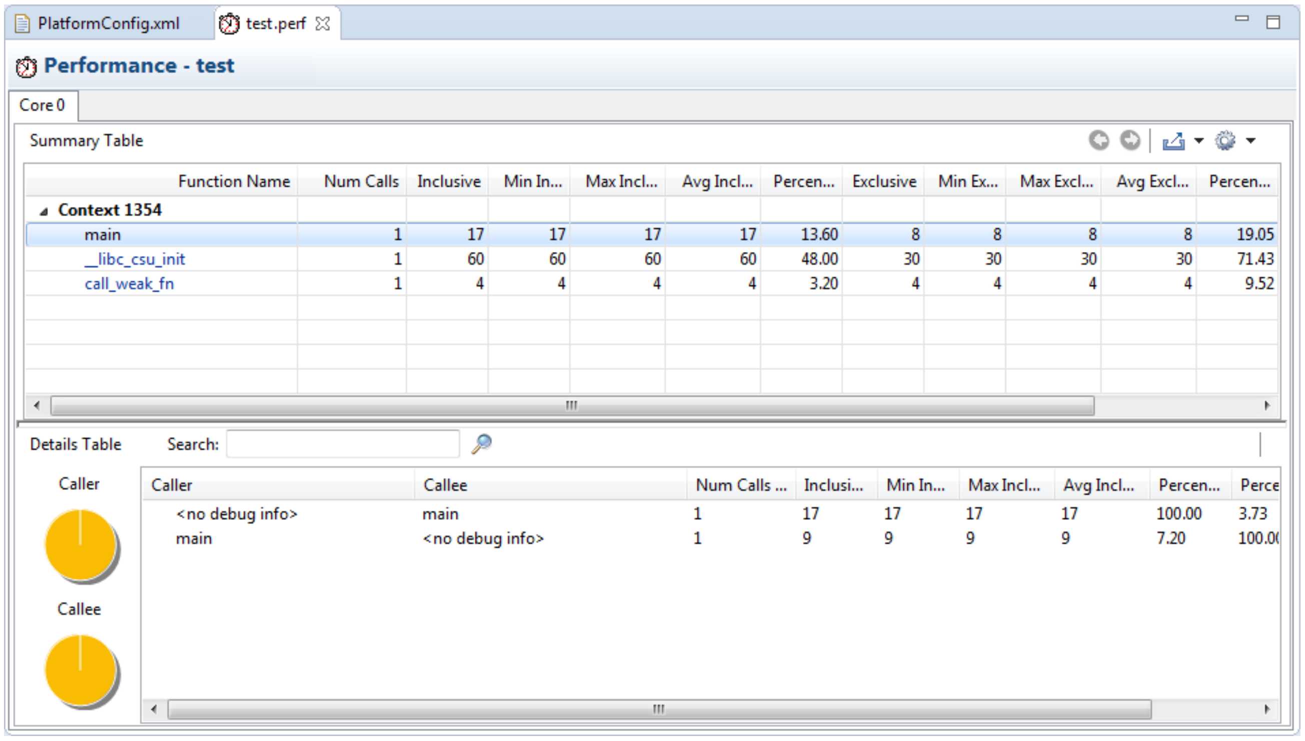
Table 5. Trace viewer events description

Trace events	Description
Branch	It reports a branch instruction from a source function to a destination function.
Linear	It reports a linear instruction.
Interrupt	It is used for interrupt contexts. It can be useful while constructing an accurate call stack.
Return	It reports a return instruction from a source to a destination function.
Data	It reports data address, data value, data size information or read/write transaction type.
Info	It reports the Trace Source's run status. The semantic of the information code is specific to the Trace Source, so it can only be interpreted by a specialized trace consumer.
Error	It reports an error instruction.
Custom	It provides the following information: • Label - A data label that indicates the "channel" with which the data value is associated. • Data - The custom data value.

4.1.2 Viewing Performance data

Click on the **Performance** link available in Analysis Results View to display performance data.

The Performance viewer appears as shown below.



Function Name	Num Calls	Inclusive	Min In...	Max Incl...	Avg Incl...	Percen...	Exclusive	Min Ex...	Max Excl...	Avg Excl...	Percen...
Context 1354											
main	1	17	17	17	17	13.60	8	8	8	8	19.05
__libc_csu_init	1	60	60	60	60	48.00	30	30	30	30	71.43
call_weak_fn	1	4	4	4	4	3.20	4	4	4	4	9.52

Caller	Callee	Num Calls ...	Inclusi...	Min In...	Max Incl...	Avg Incl...	Percen...	Perce
<no debug info>	main	1	17	17	17	17	100.00	3.73
main	<no debug info>	1	9	9	9	9	7.20	100.0

Figure 20. Performance Viewer

The Performance viewer is divided into two views:

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

  The following table explains the fields of the Summary table.

Table 6. Field description of Summary table

Field	Description
Function Name	Name of the function that has executed.
Num Calls	Number of times the function has executed.
Inclusive	Cumulative metric count during execution time spent from function entry to exit.
Min Inclusive	Minimum metric count during execution time spent from function entry to exit.
Max Inclusive	Maximum metric count during execution time spent from function entry to exit.
Avg Inclusive	Average metric count during execution time spent from function entry to exit.
Percent Inclusive	Percentage of total metric count spent from function entry to exit.

Table continues on the next page...

Table 6. Field description of Summary table (continued)

Field	Description
Exclusive	Cumulative metric count during execution time spent within function.
Min Exclusive	Minimum metric count during execution time spent within function.
Max Exclusive	Maximum metric count during execution time spent within function.
Avg Exclusive	Average metric count during execution time spent within function.
Percent Exclusive	Percentage of total metric count spent within function.
Percent Total Calls	Percentage of the calls to the function compared to the total calls.
Code Size	Number of bytes required by each function.

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

  When you double-click in **Details** pane on a call site, it opens (if available) the source file and highlights the call-site line. If the source is not available, you can Locate the file.

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

Table 7. Field description of Details table

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

Table continues on the next page...

Table 7. Field description of Details table (continued)

Field	Description
Percent Caller	Percent of total metric count during the time the selected function is the callee of a specific caller. The data is also shown in the Callee pie chart.
Call Site	Address from where the function was called.

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

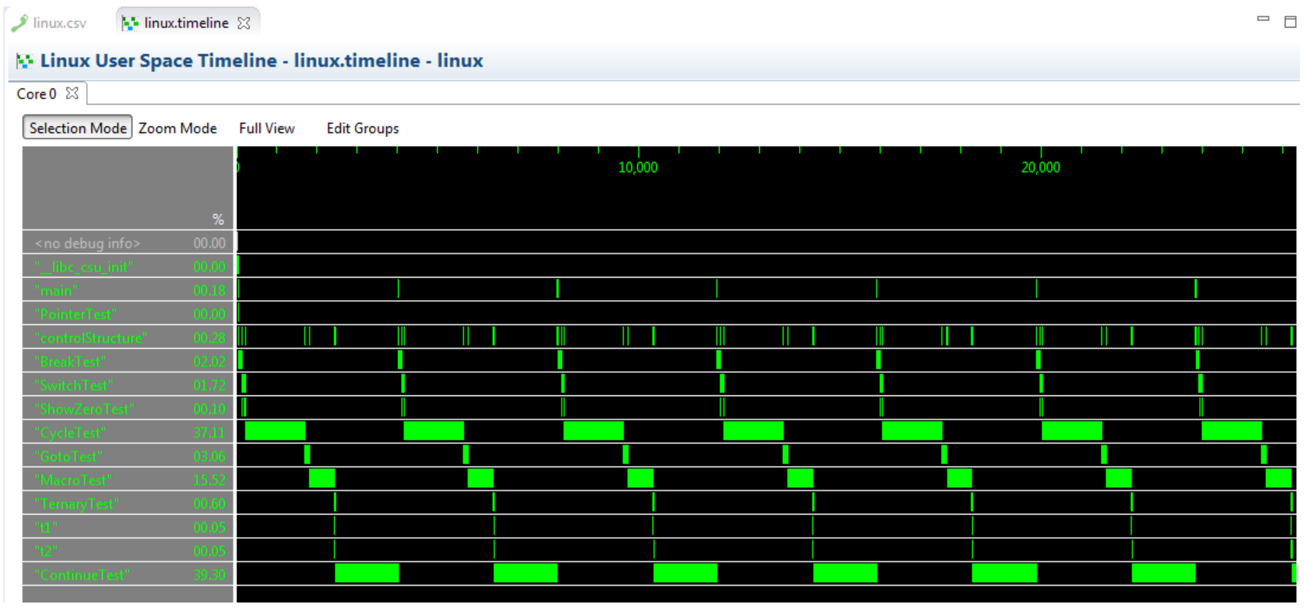
Table 8. Buttons Available in Summary Table of Performance Viewer

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

4.1.3 Viewing Timeline data

Click on the **Timeline** link from Analysis Results view to view timeline data.

The following figure shows the trace data displayed in Timeline viewer.



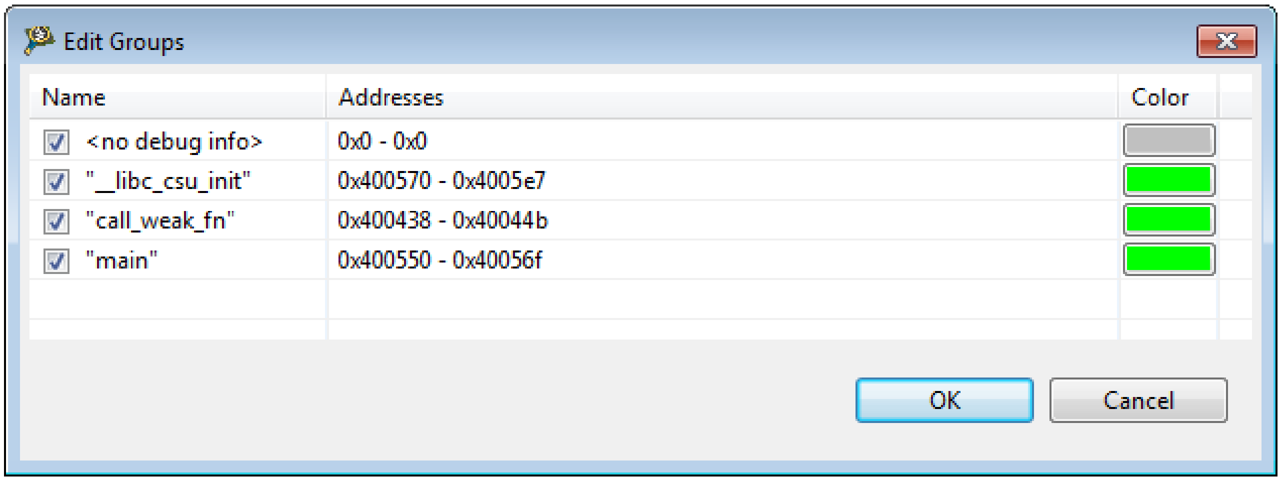
	%
<no debug info>	00.00
"__libc_csu_init"	00.00
"main"	00.18
"PointerTest"	00.00
"controlStructure"	00.28
"BreakTest"	02.02
"SwitchTest"	01.72
"ShowZeroTest"	00.10
"CycleTest"	37.11
"GotoTest"	03.06
"MacroTest"	15.52
"TernaryTest"	00.60
"t1"	00.05
"t2"	00.05
"ContinueTest"	39.30

Figure 21. Timeline Viewer

Timeline viewer has a tabbed interface. A tab for each source (usually a core) is created automatically. Each tab has its independent timeline plot with its own setup of groups and time unit. The timeline viewer can be controlled using the toolbar. The timeline data displays the functions that are executed in the application and the number of cycles each function takes when the application is run. The Timeline viewer shows 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 taken by the function.

The Timeline viewer has the following buttons:

- **Selection Mode**: Allows you to mark points in the function bars to measure the difference of cycles between those points. To mark a point in the bar:
  1. Click **Selection Mode**.
  2. Click on the bar where you want to mark the point.
  3. A yellow vertical line appears displaying the number of cycles at that point.
  4. Right-click on another point in the bar.
  5. A red vertical line appears displaying the number of cycles at that point along with the difference of cycles between two marked points.
- **Zoom Mode**: Allows you to zoom-in and zoom-out in the timeline graph. Zooming can be performed using the mouse wheel (even if this mode is not selected).
- **Full View**: Resets to the default zoom that allows seeing the full timeline
- **Edit Groups**: Allows you to customize the timeline according to the requirements. You can change the default color of the line bars representing the functions to differentiate. You can add/remove a function to/from the timeline. To perform these functions, select Edit Groups. The Edit Groups dialog box appears.



Name	Addresses	Color
✓ <no debug info>	0x0 - 0x0	⬜
✓ "__libc_csu_init"	0x400570 - 0x4005e7	🟩
✓ "call_weak_fn"	0x400438 - 0x40044b	🟩
✓ "main"	0x400550 - 0x40056f	🟩

Figure 22. Edit Groups dialog box

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

- Add or remove function on page 31
- Edit address range of function on page 31
- Change color on page 31
- Add or remove group on page 32
- Merge groups or functions on page 33

4.1.3.1 Add or remove function

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

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

4.1.3.2 Edit address range of function

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

Perform the following steps:

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

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

NOTE

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

4.1.3.3 Change color

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

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

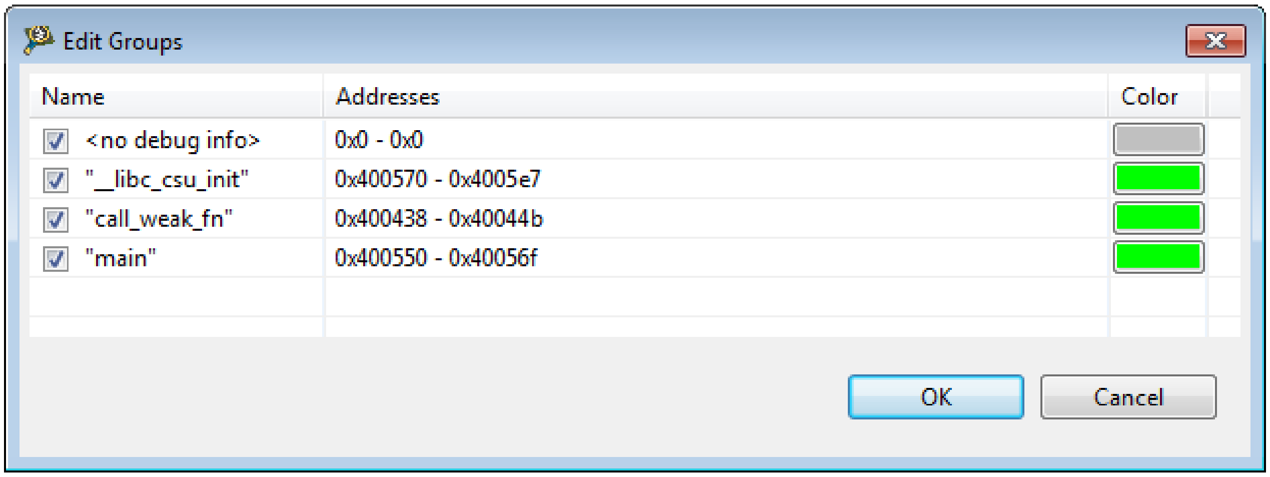
4.1.3.4 Add or remove group

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

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

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

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



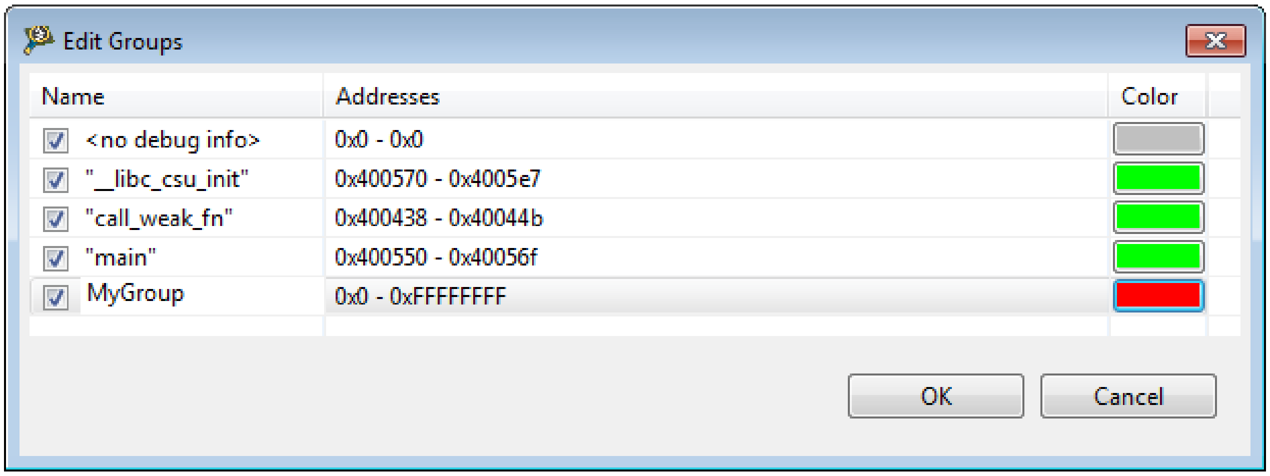
Name	Addresses	Color
✓ <no debug info>	0x0 - 0x0	⬜
✓ "__libc_csu_init"	0x400570 - 0x4005e7	🟩
✓ "call_weak_fn"	0x400438 - 0x40044b	🟩
✓ "main"	0x400550 - 0x40056f	🟩

Figure 23. Adding Group Dialog Box

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

   The cell becomes editable.

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



Name	Addresses	Color
✓ <no debug info>	0x0 - 0x0	⬜
✓ "__libc_csu_init"	0x400570 - 0x4005e7	🟩
✓ "call_weak_fn"	0x400438 - 0x40044b	🟩
✓ "main"	0x400550 - 0x40056f	🟩
✓ MyGroup	0x0 - 0xFFFFFFFF	🟥

Figure 24. After Editing Address Range and Color of Group Screen

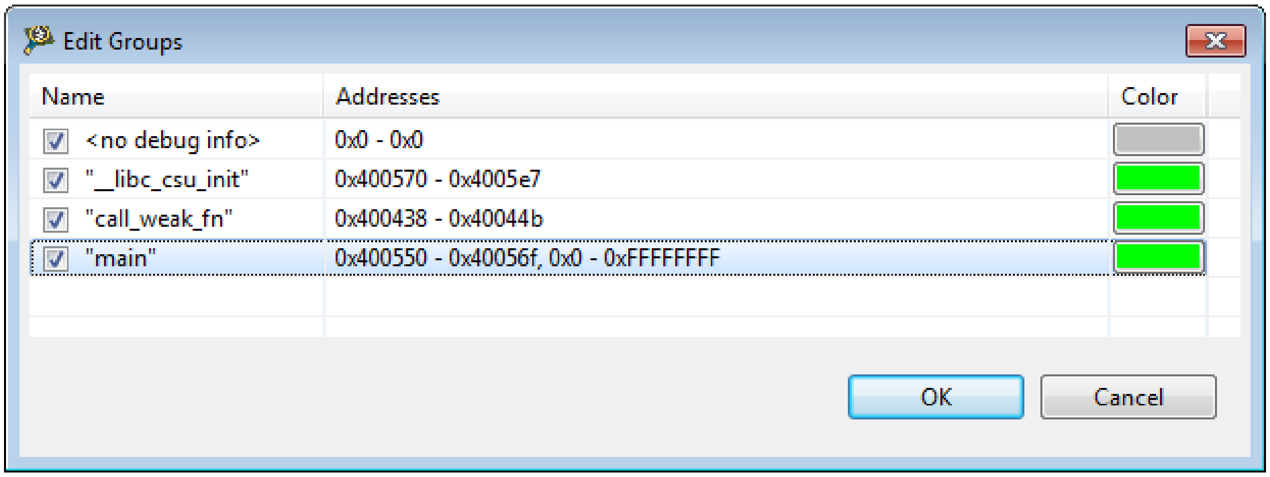
To delete a group, select it, right-click on the **Edit Groups** dialog box, and select the **Delete Selected** option from the context menu. You can also remove a group from the graph by clearing the corresponding checkbox in the **Name** column. Check it again to include it in the graph.

4.1.3.5 Merge groups or functions

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

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

1. In the **Edit Groups** dialog box, select the function or group to be merged.
2. Drag and drop it in the function or group with which you want it to get merged with.
3. Both the functions or groups merge into a single function or group that covers both address ranges, where the function, `main` is merged with the group, `MyGroup`.



Name	Addresses	Color
✓ <no debug info>	0x0 - 0x0	⬜
✓ "__libc_csu_init"	0x400570 - 0x4005e7	🟩
✓ "call_weak_fn"	0x400438 - 0x40044b	🟩
✓ "main"	0x400550 - 0x40056f, 0x0 - 0xFFFFFFFF	🟩

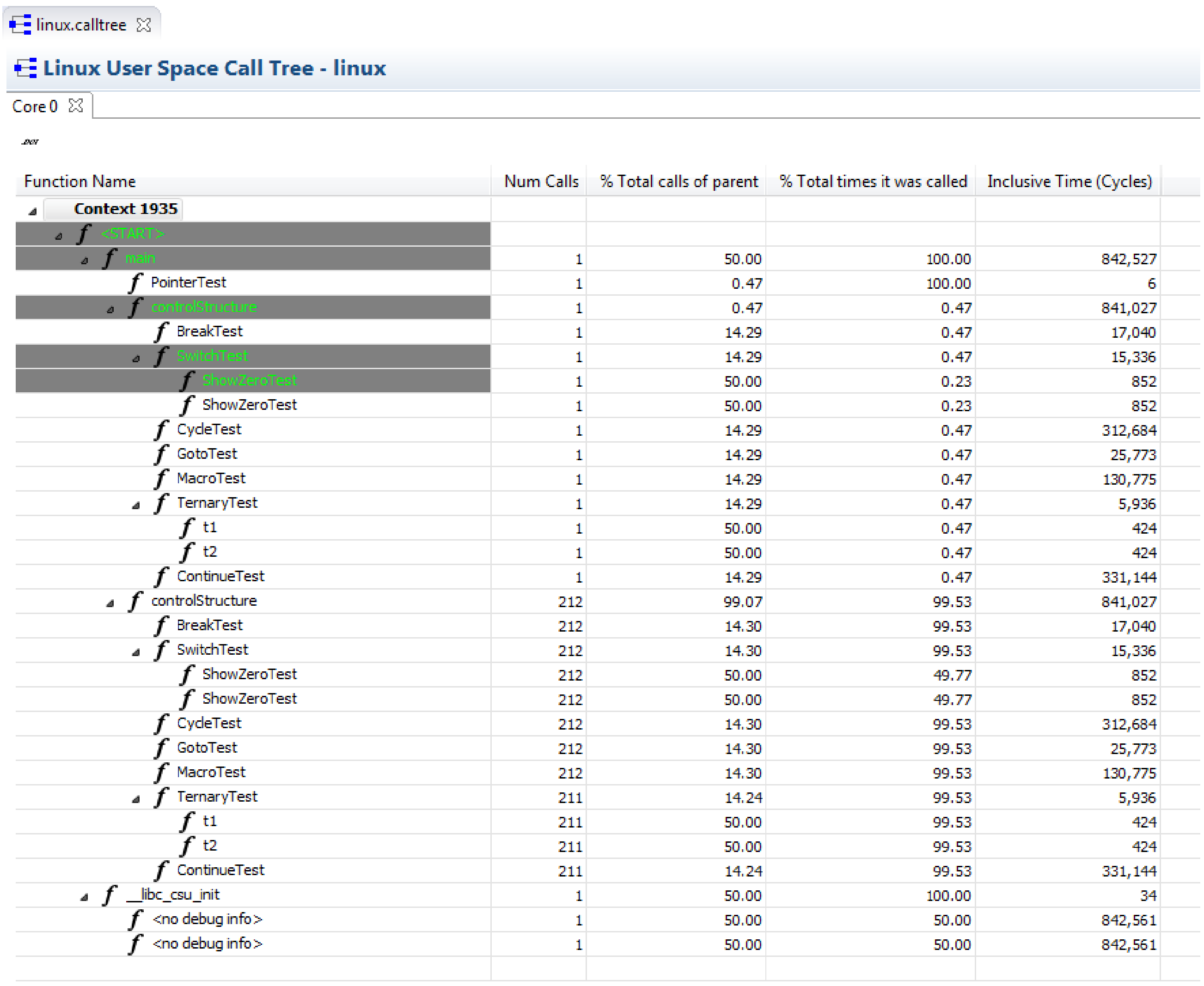
Figure 25. Merging Functions or Groups Screen

4. Click **OK**.

4.1.4 Viewing Call Tree data

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

The following image shows an example of the Call Tree data.



linux.calltree

Linux User Space Call Tree - linux

Core 0

Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles)
Context 1935				
<START>				
main	1	50.00	100.00	842,527
PointerTest	1	0.47	100.00	6
controlStructure	1	0.47	0.47	841,027
BreakTest	1	14.29	0.47	17,040
SwitchTest	1	14.29	0.47	15,336
ShowZeroTest	1	50.00	0.23	852
ShowZeroTest	1	50.00	0.23	852
CycleTest	1	14.29	0.47	312,684
GotoTest	1	14.29	0.47	25,773
MacroTest	1	14.29	0.47	130,775
TernaryTest	1	14.29	0.47	5,936
t1	1	50.00	0.47	424
t2	1	50.00	0.47	424
ContinueTest	1	14.29	0.47	331,144
controlStructure	212	99.07	99.53	841,027
BreakTest	212	14.30	99.53	17,040
SwitchTest	212	14.30	99.53	15,336
ShowZeroTest	212	50.00	49.77	852
ShowZeroTest	212	50.00	49.77	852
CycleTest	212	14.30	99.53	312,684
GotoTest	212	14.30	99.53	25,773
MacroTest	212	14.30	99.53	130,775
TernaryTest	211	14.24	99.53	5,936
t1	211	50.00	99.53	424
t2	211	50.00	99.53	424
ContinueTest	211	14.24	99.53	331,144
__libc_csu_init	1	50.00	100.00	34
<no debug info>	1	50.00	50.00	842,561
<no debug info>	1	50.00	50.00	842,561

Figure 26. Call Tree Viewer

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

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

Table 9. Call Tree fields description table

Field	Description
Function Name	Name of function that has executed.
Num Calls	Number of times function has executed.
% Total calls of Parent	Percent of number of function calls from total number of calls in the application.
% Total times it was called	Percent of number of times a function was called.

Table continues on the next page...

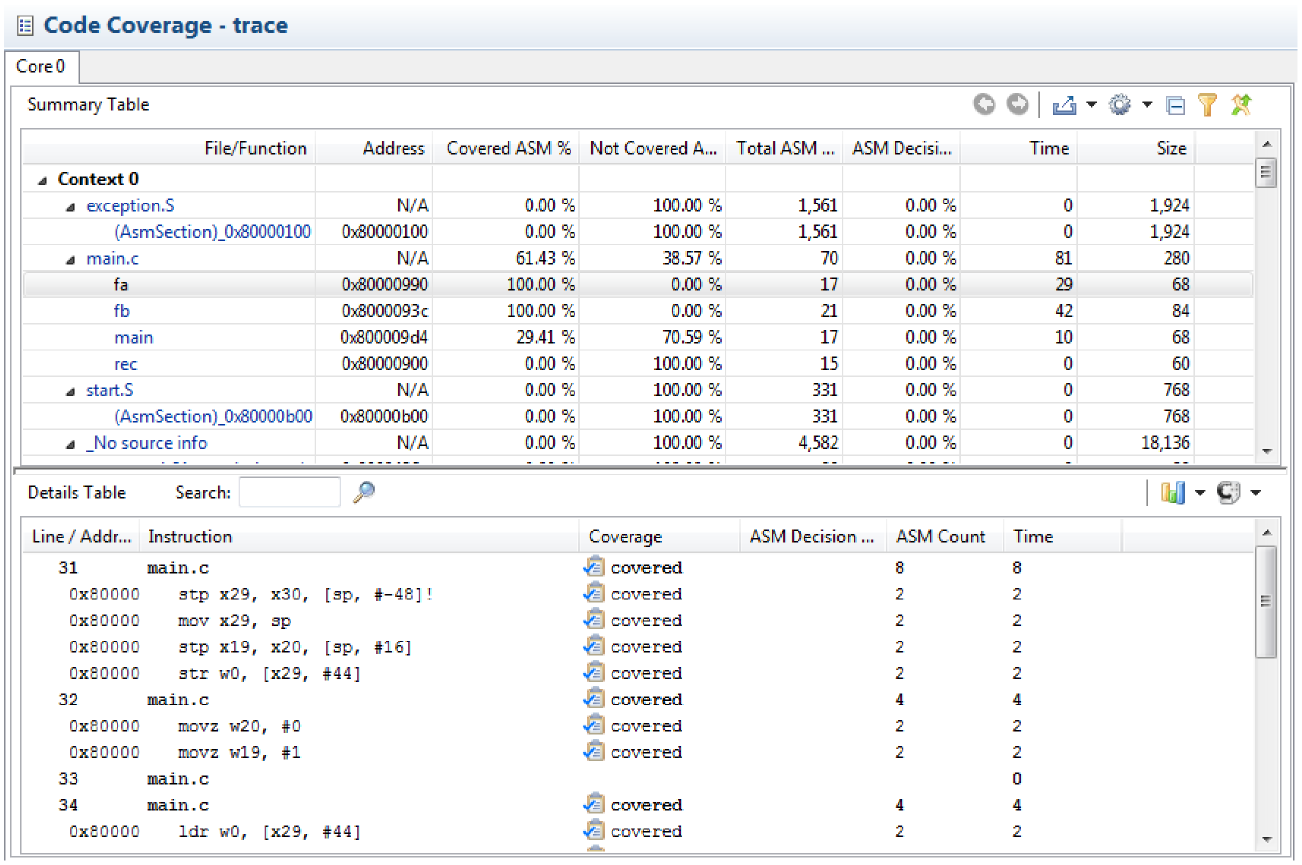
Table 9. Call Tree fields description table (continued)

Field	Description
Inclusive Time	Cumulative count during execution time spent from function entry to exit.

4.1.5 Viewing Code Coverage data

Click the **Code Coverage** link to view to view the code coverage data.

The following diagram shows an example of the Code Coverage data.



Code Coverage - trace

Core 0

Summary Table

File/Function	Address	Covered ASM %	Not Covered A...	Total ASM ...	ASM Decisi...	Time	Size
Context 0							
exception.S	N/A	0.00 %	100.00 %	1,561	0.00 %	0	1,924
(AsmSection)_0x80000100	0x80000100	0.00 %	100.00 %	1,561	0.00 %	0	1,924
main.c	N/A	61.43 %	38.57 %	70	0.00 %	81	280
fa	0x80000990	100.00 %	0.00 %	17	0.00 %	29	68
fb	0x8000093c	100.00 %	0.00 %	21	0.00 %	42	84
main	0x800009d4	29.41 %	70.59 %	17	0.00 %	10	68
rec	0x80000900	0.00 %	100.00 %	15	0.00 %	0	60
start.S	N/A	0.00 %	100.00 %	331	0.00 %	0	768
(AsmSection)_0x80000b00	0x80000b00	0.00 %	100.00 %	331	0.00 %	0	768
_No source info	N/A	0.00 %	100.00 %	4,582	0.00 %	0	18,136

Details Table   Search:

Line / Addr...	Instruction	Coverage	ASM Decision ...	ASM Count	Time
31	main.c	covered		8	8
0x80000	stp x29, x30, [sp, #-48]!	covered		2	2
0x80000	mov x29, sp	covered		2	2
0x80000	stp x19, x20, [sp, #16]	covered		2	2
0x80000	str w0, [x29, #44]	covered		2	2
32	main.c	covered		4	4
0x80000	movz w20, #0	covered		2	2
0x80000	movz w19, #1	covered		2	2
33	main.c				0
34	main.c	covered		4	4
0x80000	ldr w0, [x29, #44]	covered		2	2

Figure 27. Code Coverage viewer

The Code Coverage data displays the summarized data of a function in a tabular form. The columns are movable; you can drag and drop the columns to move them according to your requirements. It displays data into two views; the top view displays the summary of the functions, and the bottom view displays the statistics for all the instructions executed in a particular function. Click on a hyperlinked function in the top view of the Code Coverage viewer to view the corresponding statistics for the instructions executed in that function.

The below table explains the various fields of the Summary table.

Table 10. Code Coverage Summary Table fields description table

Field	Description
File/Function	Displays the name of the function that has executed.

Table continues on the next page...

Table 10. Code Coverage Summary Table fields description table (continued)

Field	Description
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.
Time (Microsecond)	Displays the total number of clock cycles that the function takes
Size	Displays the number of bytes required by each function.

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

Table 11. Code Coverage Details Table fields description table

Field	Description
Line/Address	Displays either the line number for each instruction in the source code or the address for the assembly code.
Instruction	Displays all the instructions executed in the selected function.
Coverage %	For C source lines, displays the percentage of number of assembly instructions executed from the total number of assembly instructions corresponding to the source line. For assembly source lines, it shows if the instructions were executed or not.
ASM Decision Coverage	Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. So if a function has two conditional instructions, one with 100% and another with 50% decision coverage, the decision coverage would be (100 + 50) / 2 = 75%. It is calculated only for assembly instructions and not for C source code.
ASM Count	Displays the number of times each instruction is executed.
Time (CPU Cycles)	Displays the total time taken by each instruction in the function.

NOTE

In the Code Coverage 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.

When you double-click in **Details** table on an instruction, it opens (if available) the source file and highlights the instruction line. If the source is not available, you can Locate the file.

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

- **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.

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 code coverage 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 code coverage data. Click and select the **Configure Columns for Summary Table** option to show/hide columns in the Summary table (top view) or the **Configure Columns for Details table** option to show/hide columns of the Details table (bottom view). The **Drag and drop to order columns** dialog appears in which you can check/uncheck the checkboxes corresponding to the available columns to show/hide them in the **Code Coverage** 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.

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.
- **Graphics** - Lets you display the histograms in two colors for the ASM Count and Time columns in the bottom view of the code coverage 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 Code Coverage data.

4.1.5.1 Export Code Coverage data

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

To export the data, follow the given procedure:

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

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

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

   The **Export to** dialog box appears.
3. Specify the **File name** and browse the location where you want to save the data.
4. Click **Save**.

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

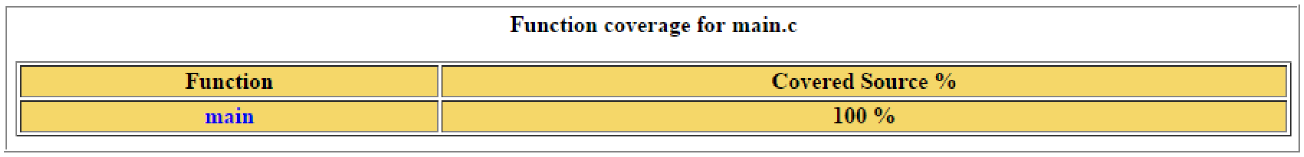
The data exported on the html page is structured in the given format:

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

NOTE

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

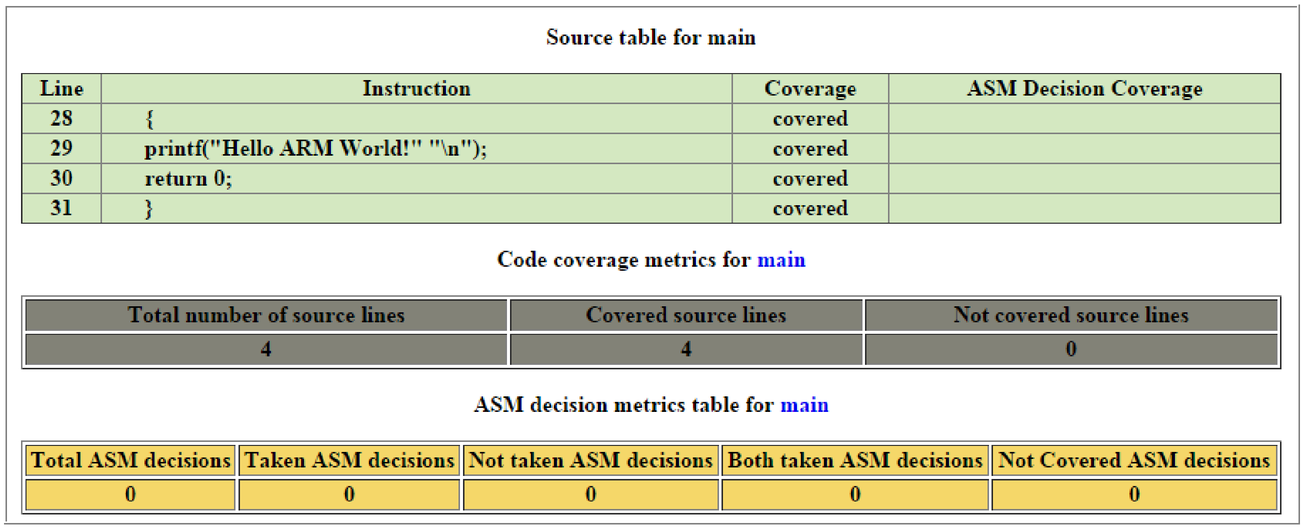
The screens given below shows the data structured in different tables:



Function coverage for main.c

Function	Covered Source %
main	100 %

Figure 28. Function Coverage Table



Source table for main

Line	Instruction	Coverage	ASM Decision Coverage
28	{	covered	
29	printf("Hello ARM World!" "\n");	covered	
30	return 0;	covered	
31	}	covered	

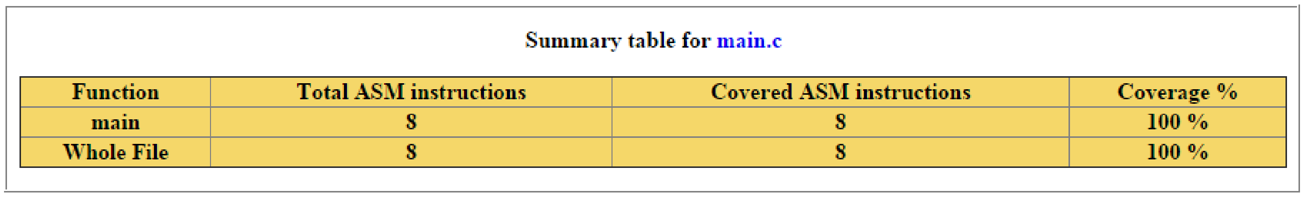
Code coverage metrics for main

Total number of source lines	Covered source lines	Not covered source lines
4	4	0

ASM decision metrics table for main

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

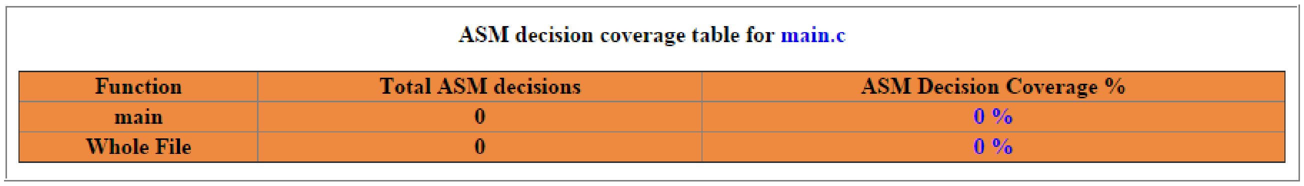
Figure 29. Function Details Table



Summary table for main.c

Function	Total ASM instructions	Covered ASM instructions	Coverage %
main	8	8	100 %
Whole File	8	8	100 %

Figure 30. Summary Table



ASM decision coverage table for main.c

Function	Total ASM decisions	ASM Decision Coverage %
main	0	0 %
Whole File	0	0 %

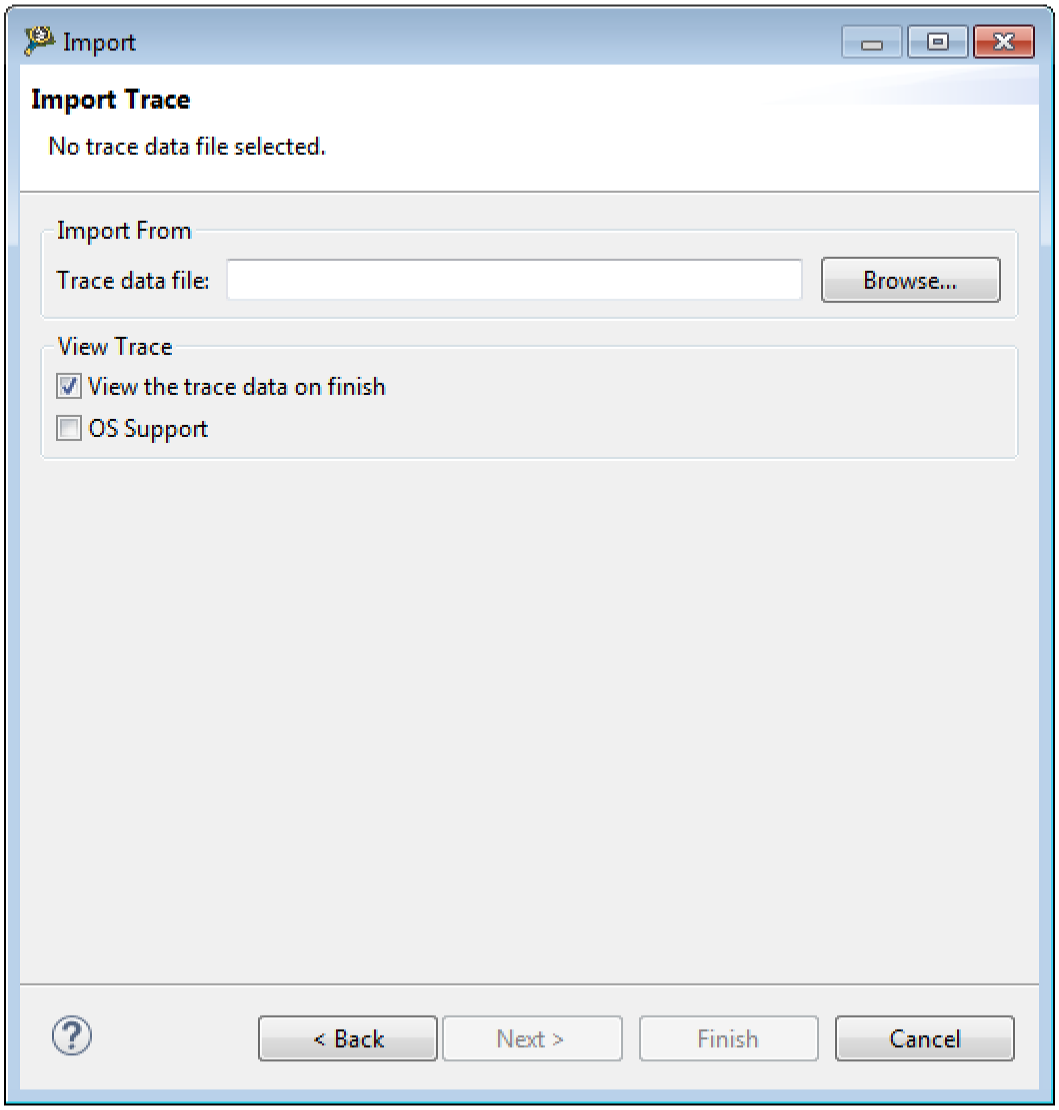
Figure 31. ASM Decision Coverage Table

4.2 Import trace data

This feature allows you to import a trace data file.

To import the trace data, perform the following steps:

1. Select **File > Import** to open the Import wizard.
2. Expand the **Software Analysis** node and select **Trace** option.
3. Click **Next** to display the **Import Trace** page of the Import wizard.



Import Trace

No trace data file selected.

Import From

Trace data file:

View Trace

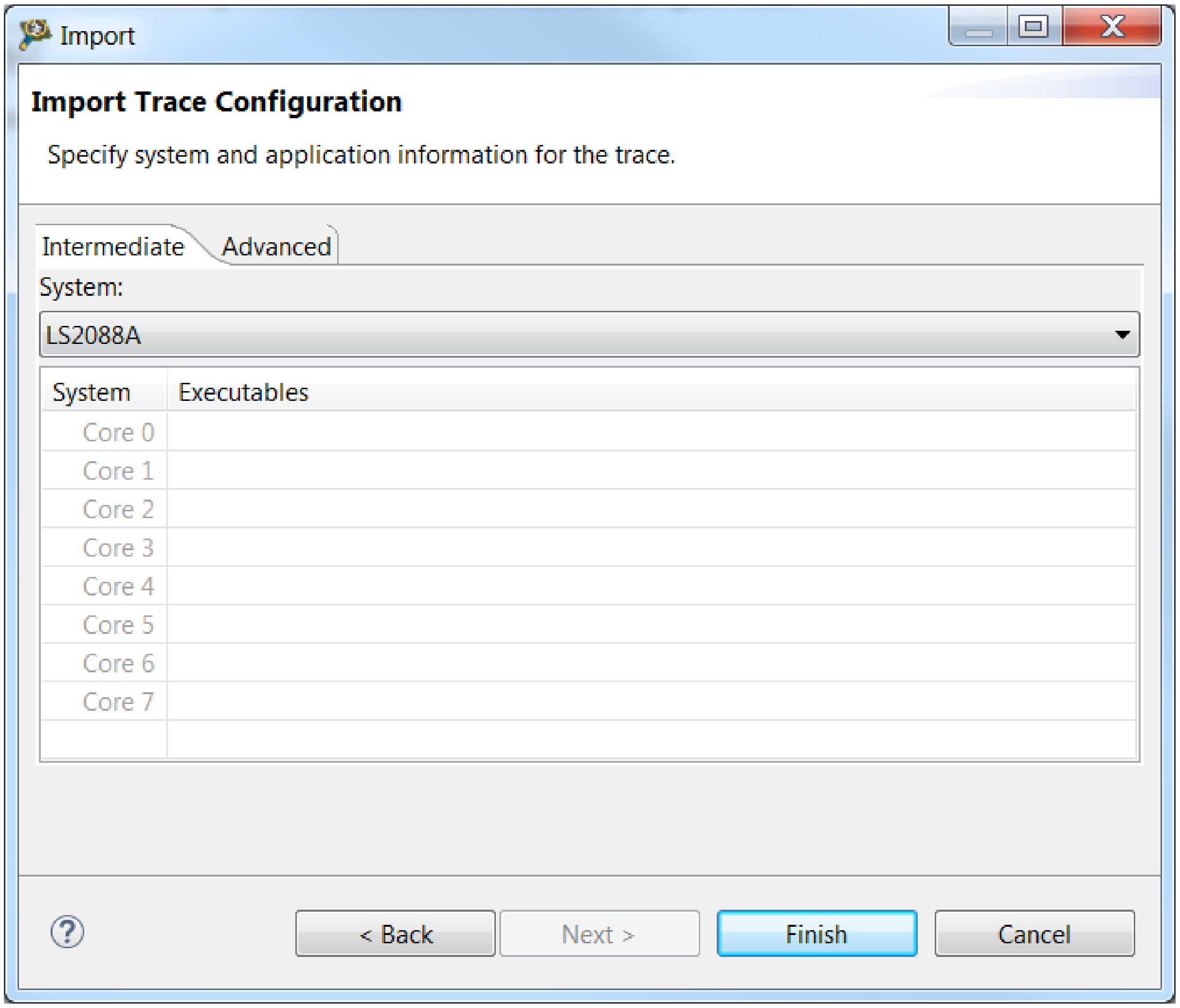
☑ View the trace data on finish

☐ OS Support

Figure 32. Import wizard

4. Click **Browse** to locate the trace data file of the project that you want to import. The files supported are raw trace files `(.dat)` or raw trace archive files `(.cwzsa, .kcwzsa, .scwzsa).`
5. Check **View the trace data on finish** checkbox to view the trace data in Analysis Results view.
6. Select **OS Support** option to import a trace file collected from a Linux trace configuration.
7. Click **Next**.

   The **Import Trace Configuration** page appears.
8. Configure the trace settings on the **Intermediate** tab, depending upon the trace file/trace archive file selected in the **Trace data file** field:
   - Raw trace file (.dat): Allows you to specify the system and application information for which the trace data is imported. Follow the steps to import the trace data of a raw trace file:
     a. Choose an option from the **System** list. This enables the list of executables for each core of the system that the trace was collected.


Import

Import Trace Configuration

Specify system and application information for the trace.

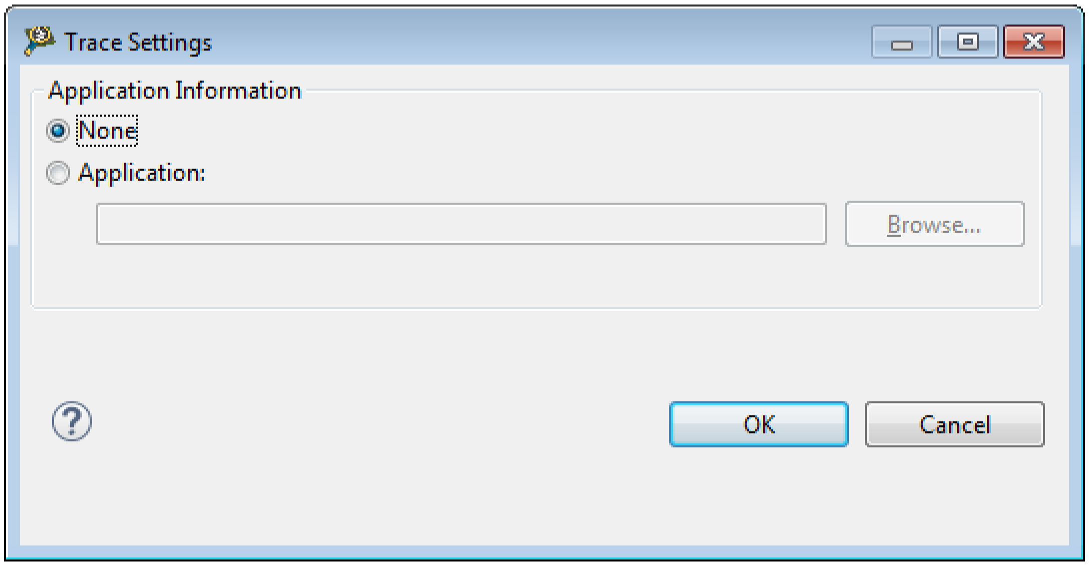
System: LS2088A

System	Executables
Core 0	
Core 1	
Core 2	
Core 3	
Core 4	
Core 5	
Core 6	
Core 7	

Figure 33. Import Trace Configuration - Intermediate tab

b. Click a cell in the **Executables** column to browse the application file for the respective core.

The **Trace Settings** dialog appears.


Trace Settings

Application Information

[✓ None]

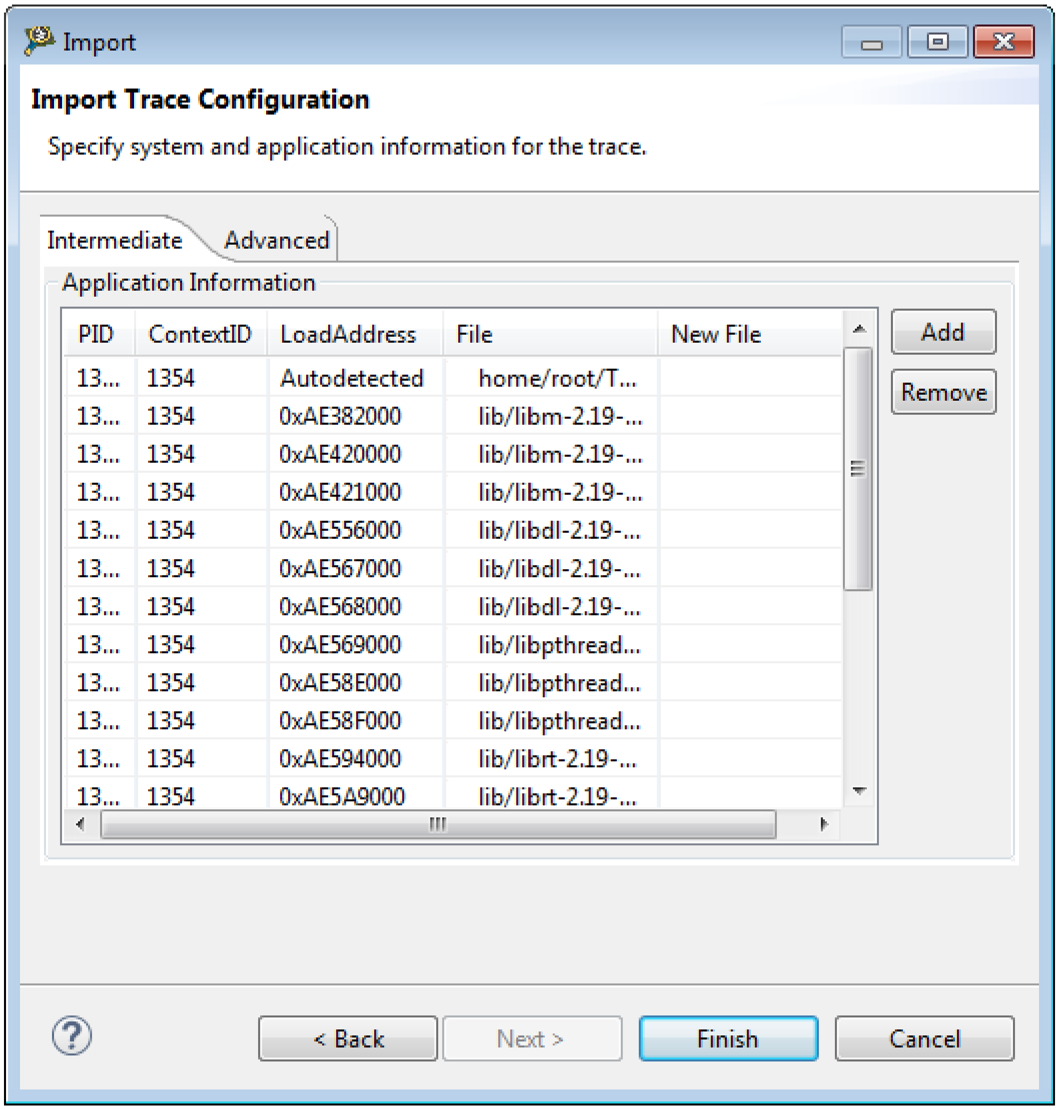
Application:

Figure 34. Trace Settings dialog

c. Browse for the location of the application file and click **OK**.

   The location of the application file will appear in the **Executables** column of the table.

- Trace archive files (.cwzsa, .kcwzsa, .scwzsa): Allows you to configure the PID, Context ID, Load Address, and the location of the executable file. You can also add or remove the executables from the list. You can modify the information in all columns except the File column, also you can add or remove executables. The relocation file contains information about an executable, which is missing from the trace archive. In this case, the wizard displays a red exclamation mark near the file location. Click in the New File column to import the missing executable file from the file system. You can also import from the file system to overwrite the file in the trace archive files. The below figure shows the Application Information available for trace archive files.



Import Trace Configuration

Specify system and application information for the trace.

Application Information

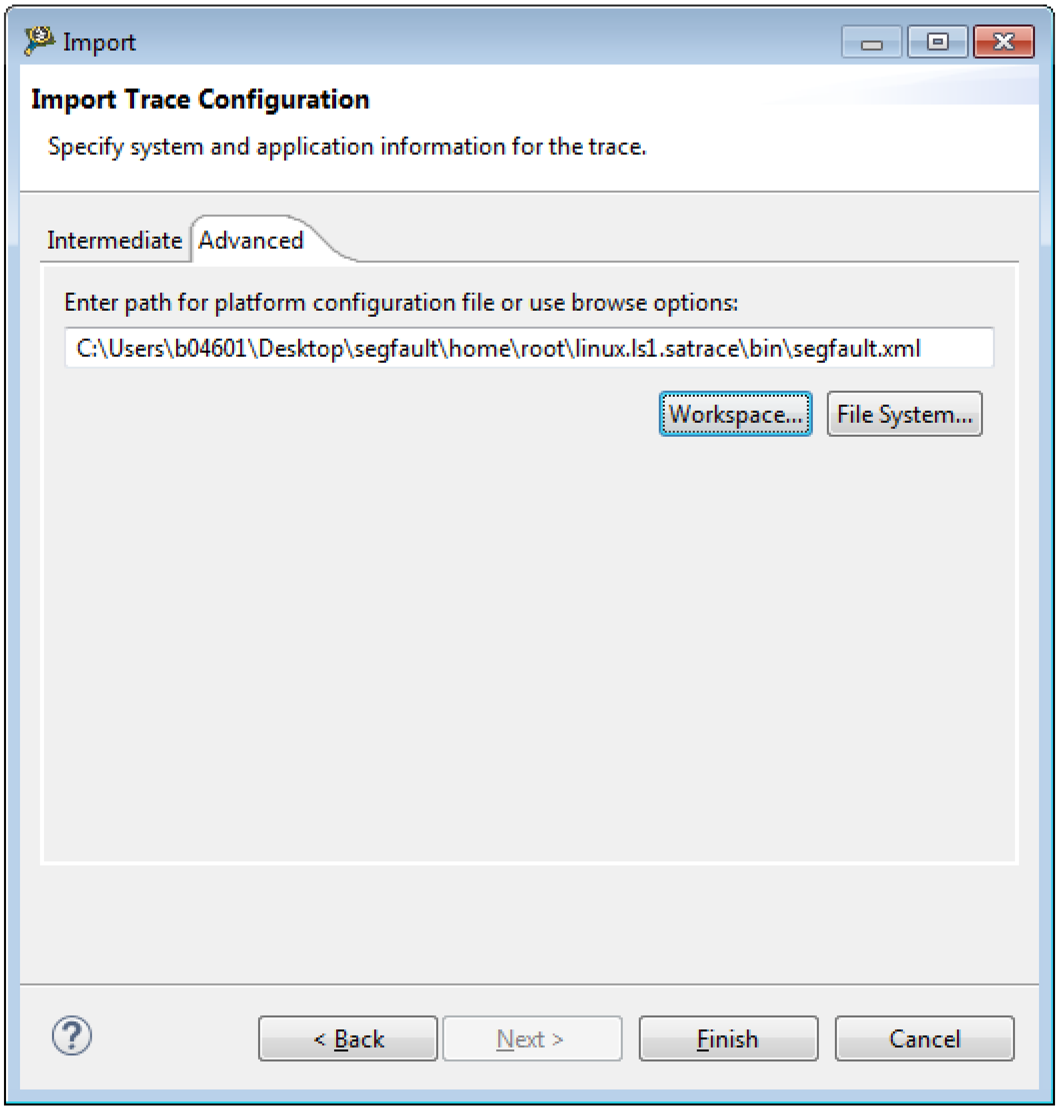
PID	ContextID	LoadAddress	File	New File
13...	1354	Autodetected	home/root/T...	
13...	1354	0xAE382000	lib/libm-2.19-...	
13...	1354	0xAE420000	lib/libm-2.19-...	
13...	1354	0xAE421000	lib/libm-2.19-...	
13...	1354	0xAE556000	lib/libdl-2.19-...	
13...	1354	0xAE567000	lib/libdl-2.19-...	
13...	1354	0xAE568000	lib/libdl-2.19-...	
13...	1354	0xAE569000	lib/libpthread...	
13...	1354	0xAE58E000	lib/libpthread...	
13...	1354	0xAE58F000	lib/libpthread...	
13...	1354	0xAE594000	lib/librt-2.19-...	
13...	1354	0xAE5A9000	lib/librt-2.19-...	

Figure 35. Import Trace Configuration - Intermediate tab

NOTE

If an executable has the Autodetected value for Load Address, it means that the file is not built with position independent code. Hence, the file is loaded at the absolute address. The load address will be read from the file.

9. Click the **Advanced** tab to specify the path of the platform configuration xml file used for collecting trace.



Import

Import Trace Configuration

Specify system and application information for the trace.

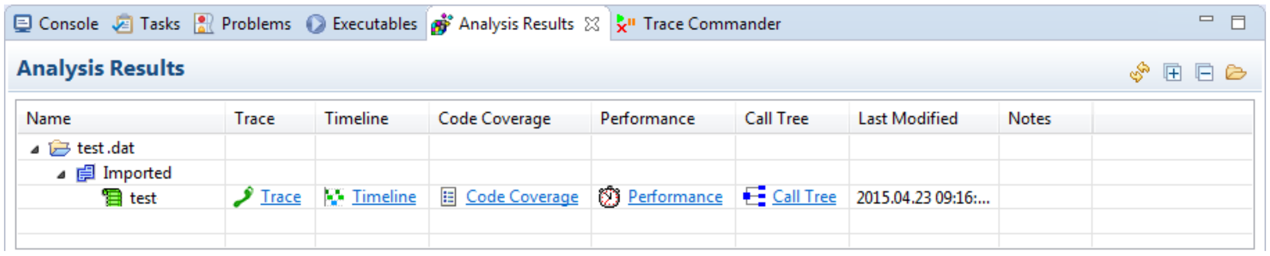
Enter path for platform configuration file or use browse options:

C:\Users\b04601\Desktop\segfault\home\root\linux.ls1.satrace\bin\segfault.xml

Figure 36. Import Trace Configuration - Advanced tab

10. Click **Finish**.

    The data gets displayed in the Analysis Results view. The following figure shows the trace data imported in the Analysis Results view.



Analysis Results

Name	Trace	Timeline	Code Coverage	Performance	Call Tree	Last Modified	Notes
test.dat							
Imported							
test	Trace	Timeline	Code Coverage	Performance	Call Tree	2015.04.23 09:16:...	

Figure 37. Analysis Results view

Click on the links available in the Analysis Results view to display the imported trace data in the respective viewers.

Chapter 5
Collect and View Linux satrace Data

This chapter explains the steps required to collect and view the Linux trace data using satrace utility.

The Linux trace mechanism is independent of CodeWarrior. The satrace is a user space application with hardware trace capabilities with no code instrumentation. It is delivered as a stand-alone component based on an executable and couple of shared libraries. The tool encapsulates the trace configurator and probe. Its main use is to collect the trace of a program without using any type of hardware probe. The satrace mechanism does not provide GUI. This document proposes an integration solution for this command line utility into CodeWarrior and also how to collect trace without using CodeWarrior.

This chapter describes:

- Collecting Linux trace from CodeWarrior using satrace on page 45
- Importing and decoding trace collected using satrace on page 54

5.1 Collecting Linux trace from CodeWarrior using satrace

This section explains how you can collect satrace by establishing RSE connection with the board.

To collect trace using satrace:

1. Start CodeWarrior for ARMv8 and create a CodeWarrior Linux Project.
2. Write the application using the Editor.
3. Build the project to generate an elf file.

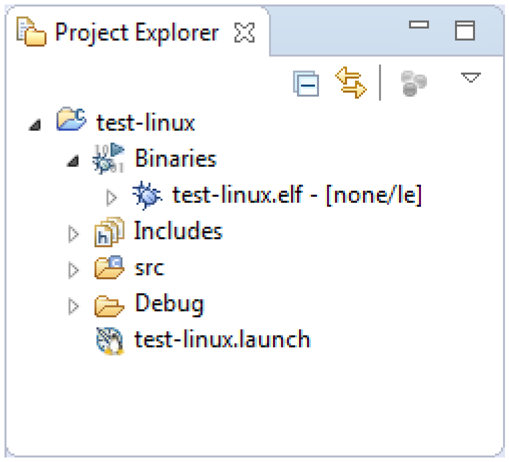


Figure 38. Project Explorer view

4. Create a Linux connection using Remote System Explorer (RSE).
   a. To open a Remote System Explorer view, click **Window > Open Perspective > Other > Remote System Explorer**.

      The Remote Systems view appears.
   b. Click **New connection** available in the Remote Systems view toolbar.

      The New Connection wizard appears.
   c. Expand **General** and select **Linux** option from the list.

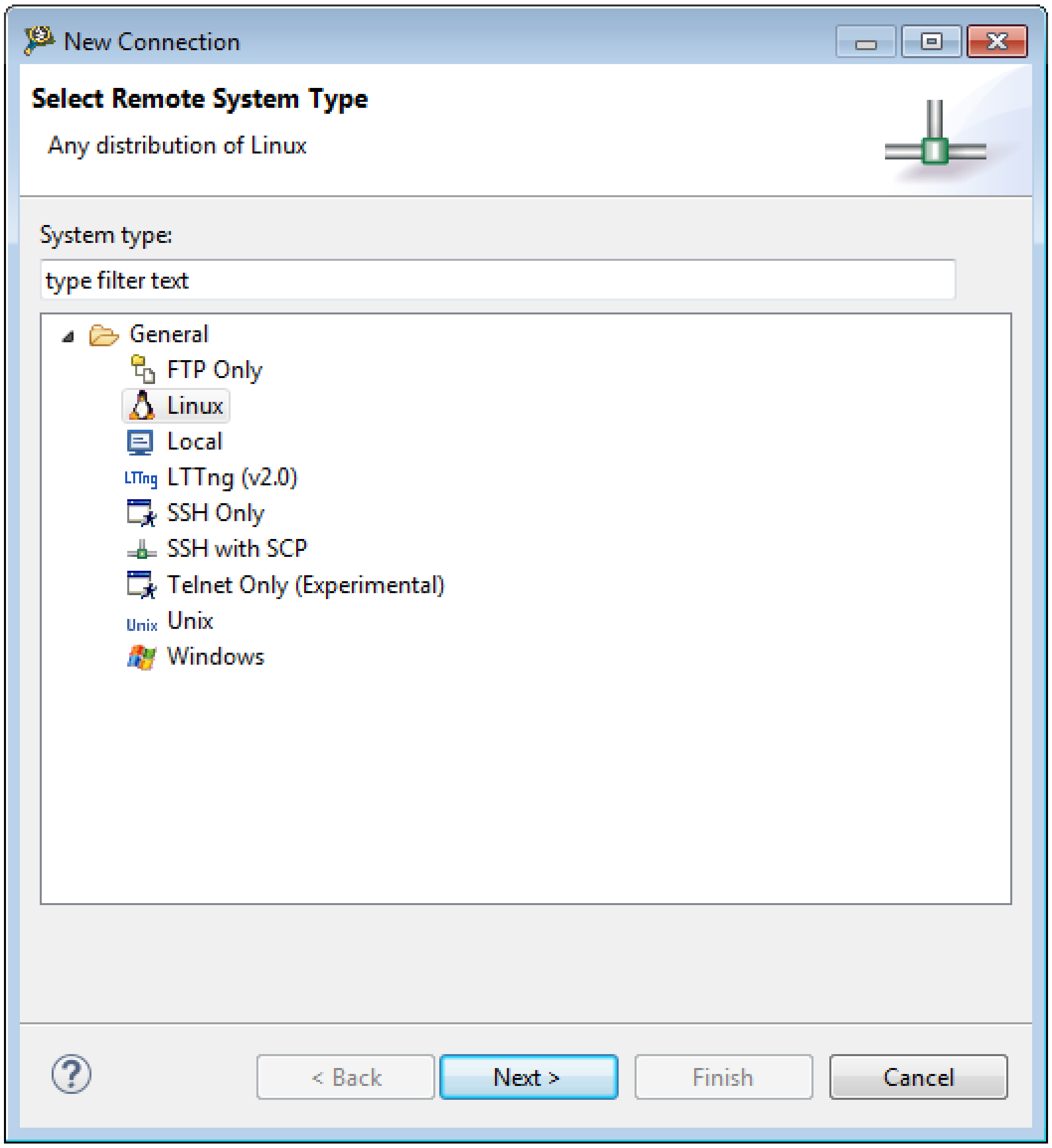
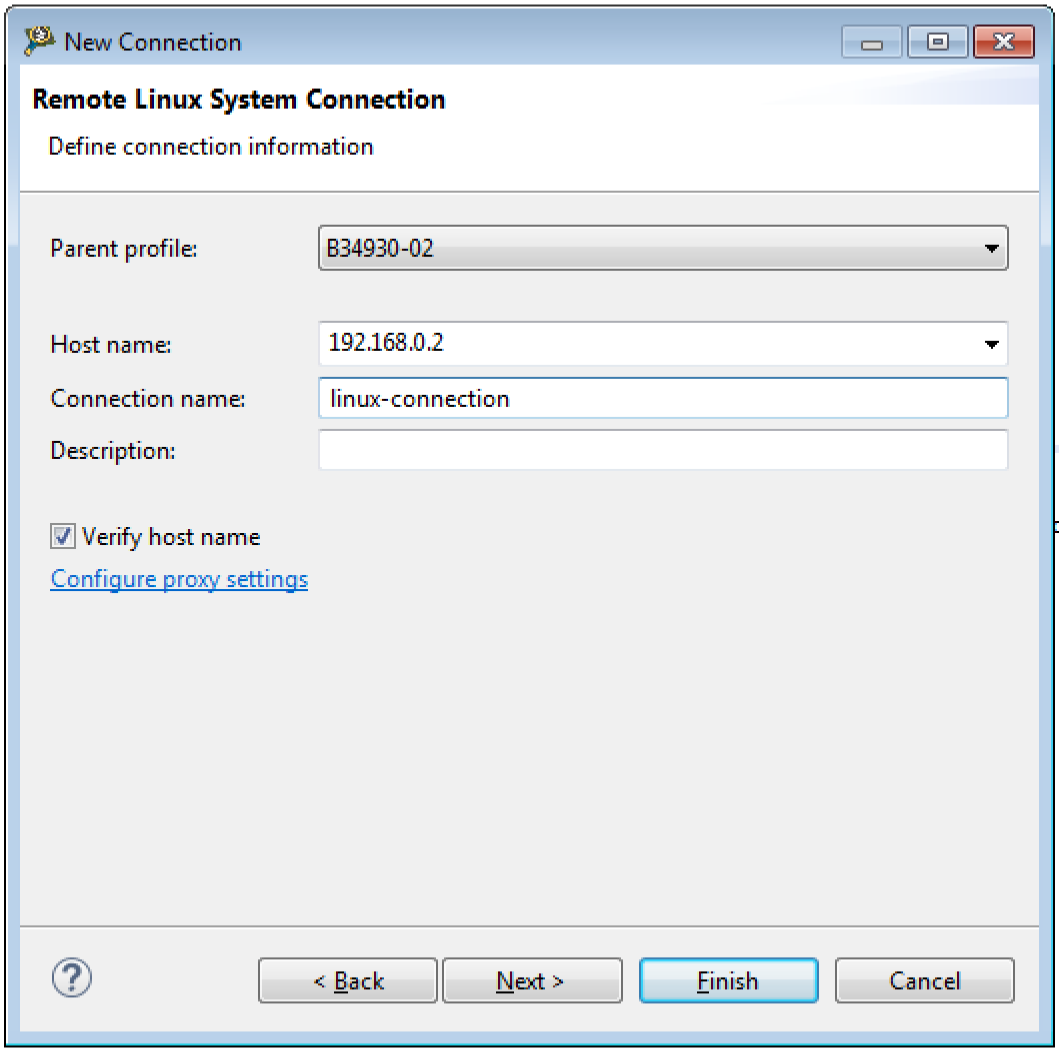


Figure 39. New Connection wizard

d. Click **Next**.

The **Remote Linux System Connection** page appears.



Remote Linux System Connection

Define connection information

Parent profile: B34930-02

Host name: 192.168.0.2

Connection name: linux-connection

Description:

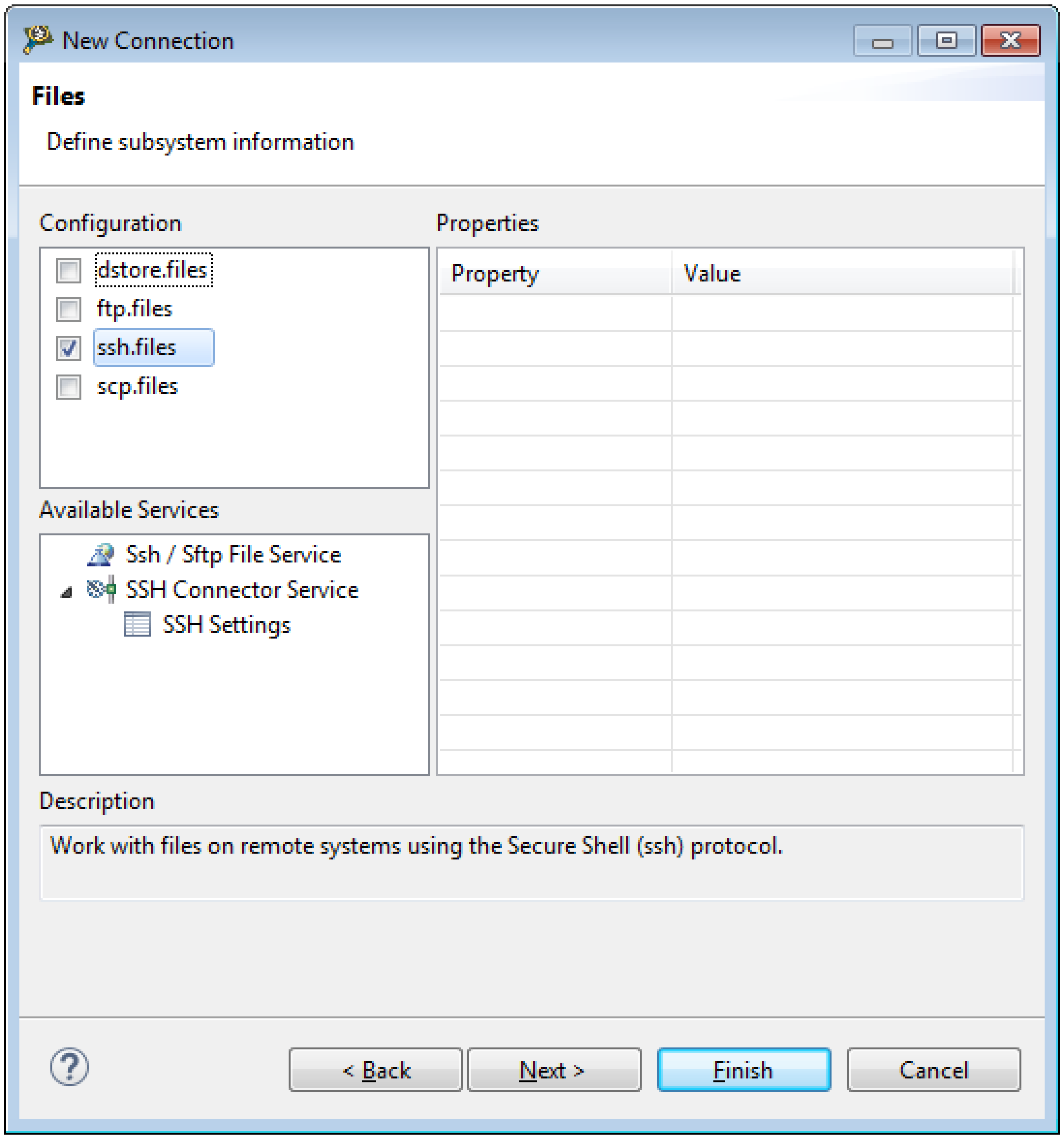
☑ Verify host name

Configure proxy settings

Figure 40. Remote Linux System Connection page

e. Specify the **Host name** and the **Connection name** and click **Next**.

The **Files** page appear.

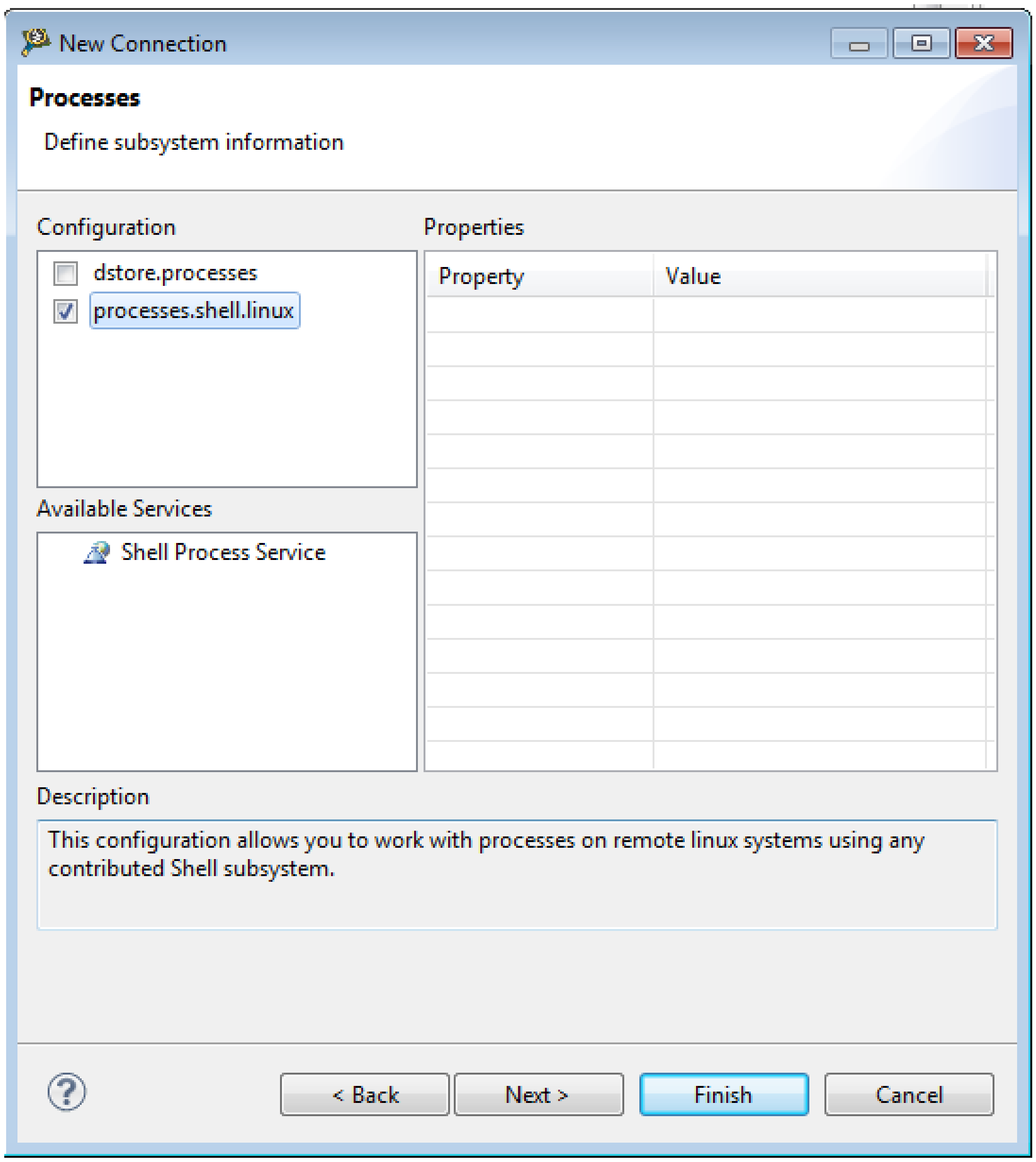


Work with files on remote systems using the Secure Shell (ssh) protocol.

Figure 41. Files page

f. Select the **ssh.files** checkbox and click **Next**.

The **Processes** page appears.


New Connection

Processes

Define subsystem information

Configuration

- [ ] dstore.processes
- [x] processes.shell.linux

Available Services

Shell Process Service

Properties

Property	Value

Description

This configuration allows you to work with processes on remote linux systems using any contributed Shell subsystem.

Figure 42. Processes page

g. Select the **processes.shell.linux** checkbox and click **Next**.

The **Shells** page appears.

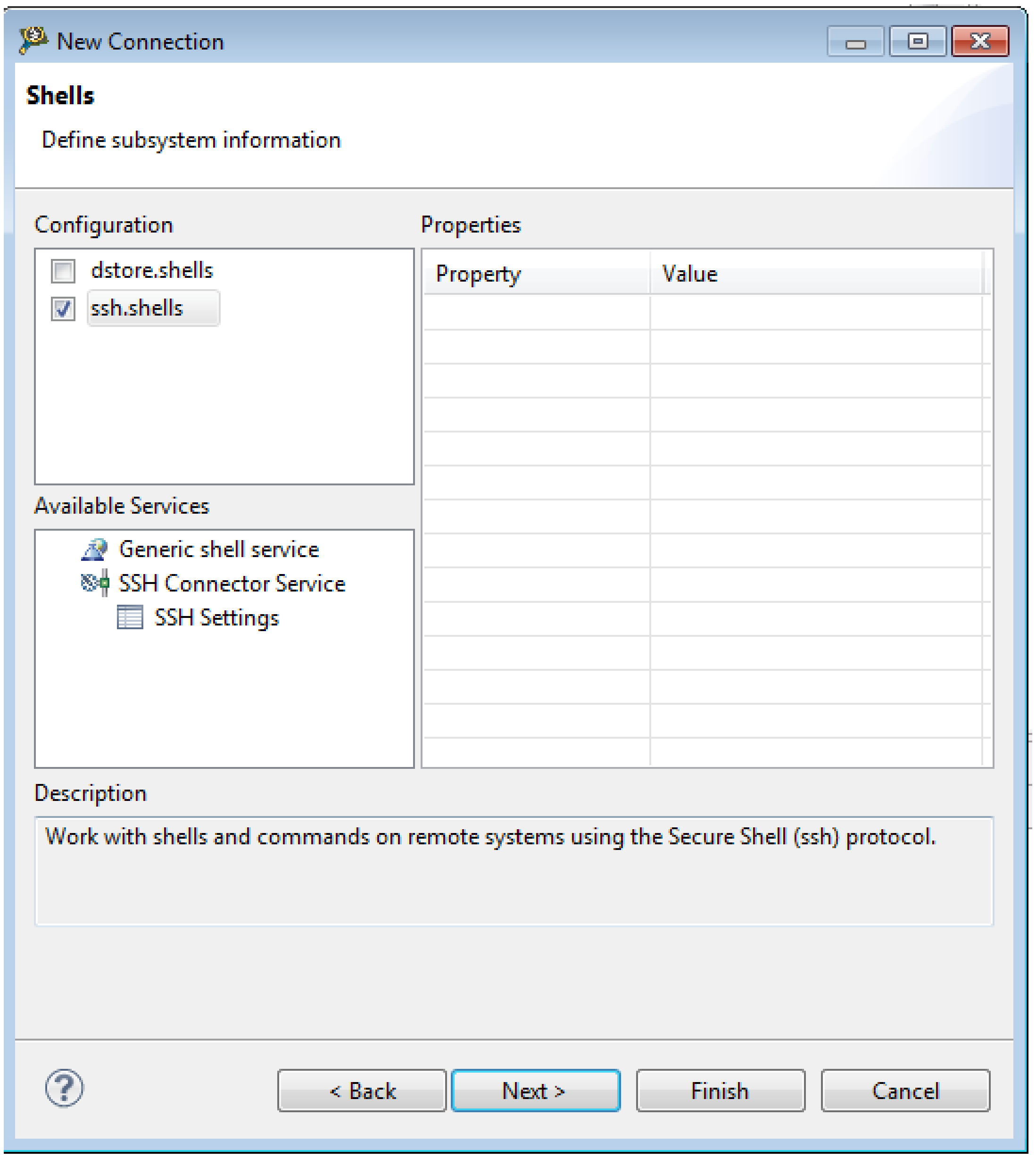
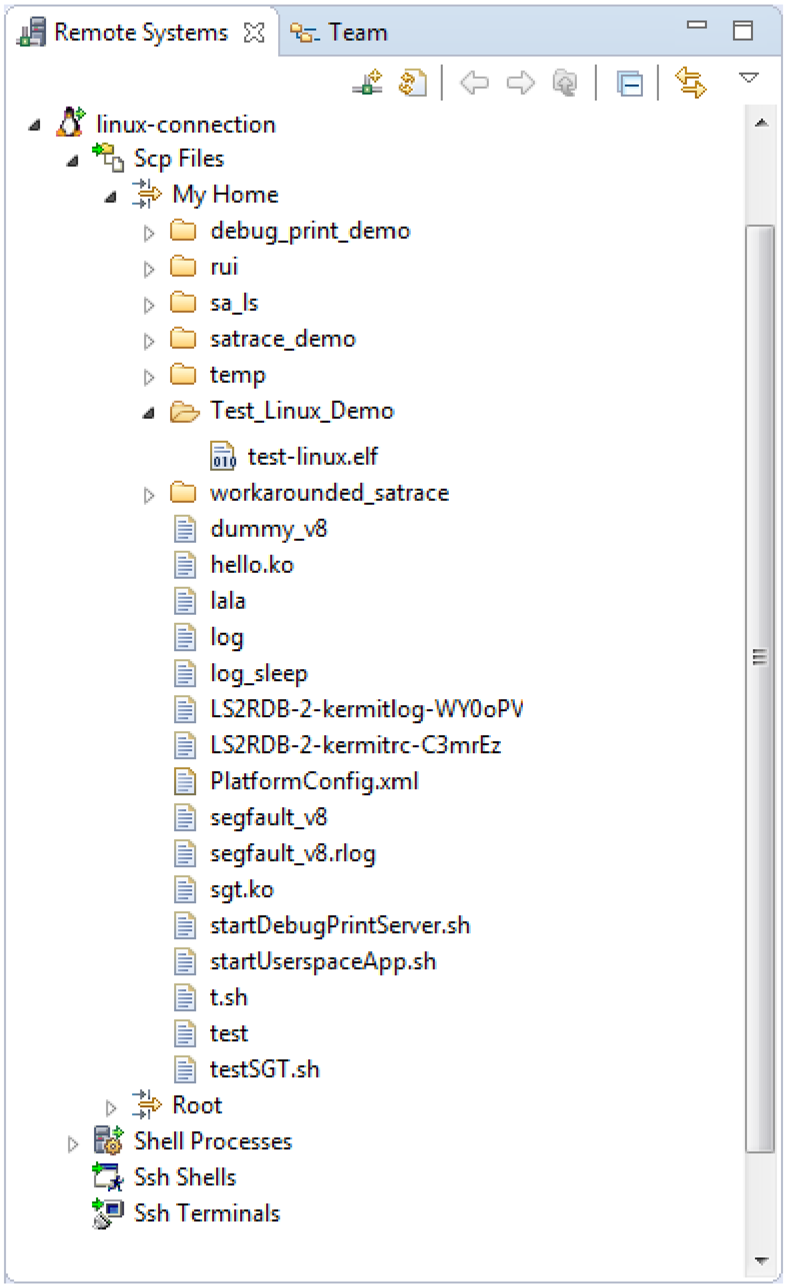


Figure 43. Shells page

h. Select the **ssh.shells** checkbox and click **Finish**.

i. In the **Remote System** view, you can see that the connection with the board has been established. The connection name is *linux-connection*. Now, you can paste the .elf file as shown below.



linux-connection
- Scp Files
  - My Home
    - debug_print_demo
    - rui
    - sa_ls
    - satrace_demo
    - temp
    - Test_Linux_Demo
      - test-linux.elf
    - workarounded_satrace
    - dummy_v8
    - hello.ko
    - lala
    - log
    - log_sleep
    - LS2RDB-2-kermitlog-WY0oPV
    - LS2RDB-2-kermitrc-C3mrEz
    - PlatformConfig.xml
    - segfault_v8
    - segfault_v8.rlog
    - sgt.ko
    - startDebugPrintServer.sh
    - startUserspaceApp.sh
    - t.sh
    - test
    - testSGT.sh
  - Root
- Shell Processes
- Ssh Shells
- Ssh Terminals

Figure 44. ELF file in RSE view

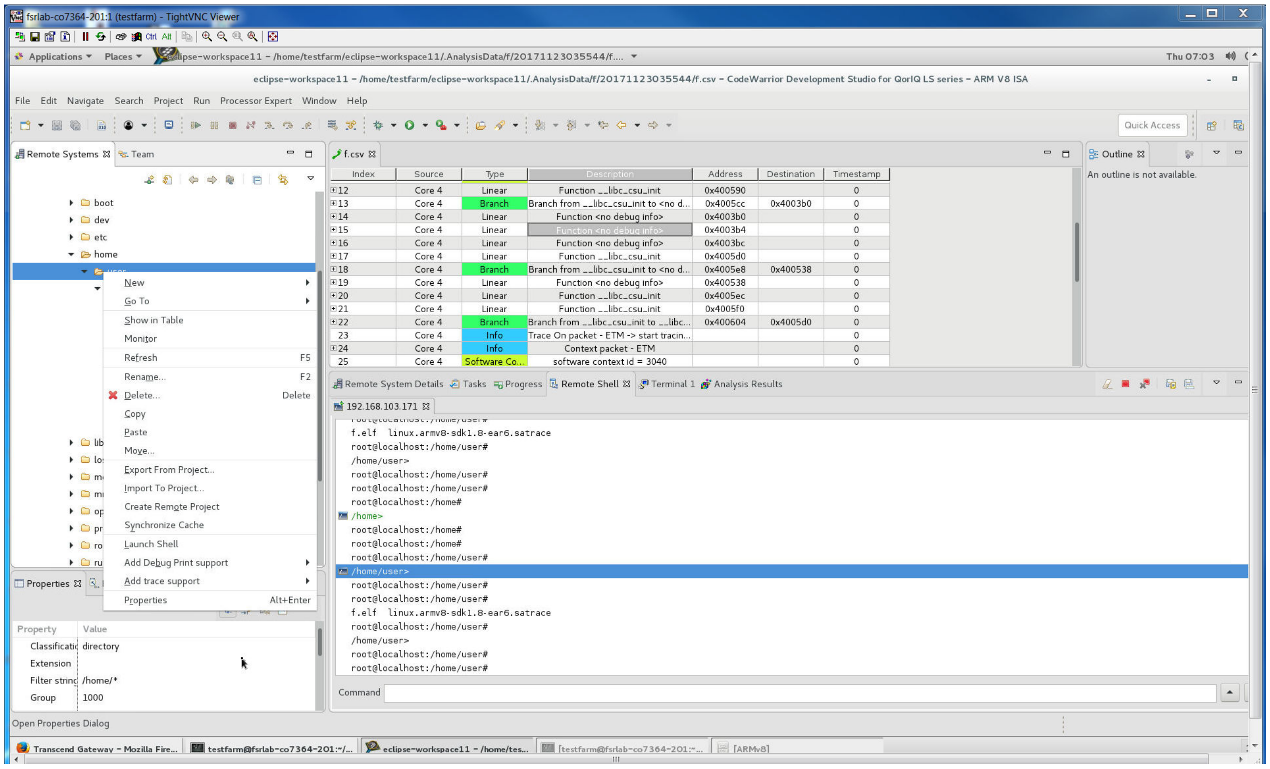
5. Right-click on the parent folder where you have pasted the elf file, select **Add Trace Support** from the context menu to copy the satrace into a directory.

NOTE

Use the apropriate satrace agent from `{CW_INSTALL_DIR}\CW_ARMv8\ARMv8\sa_ls\`:

- `linux.armv8-sdk1.8-ear6.satrace` for linux image with little endian, on all targets, starting with ARMv8 SDK ear6
- `linux.armv8.satrace` for linux image with little endian, on all targets, before ARMv8 SDK ear6
- `linux.armv8-ls1043-be.satrace` for linux image with big endian, on LS1043

6. Right-click on the parent folder to select **Launch Shell** from the context menu.



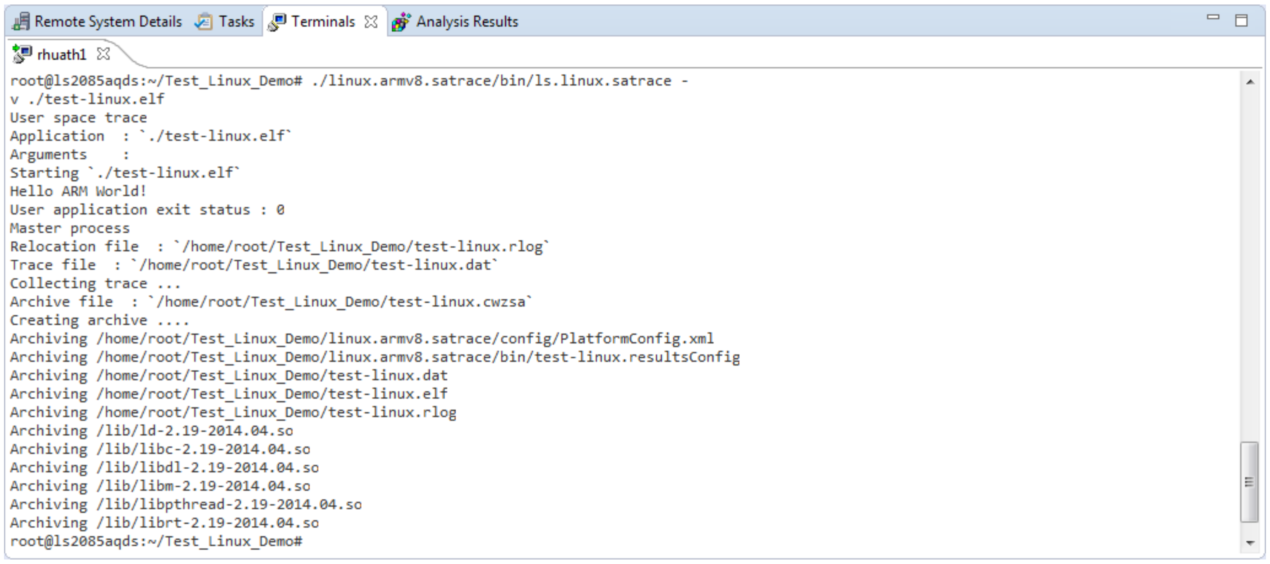
Index	Source	Type	Description	Address	Destination	Timestamp
12	Core 4	Linear	Function __libc_csu_init	0x400590		0
13	Core 4	Branch	Branch from __libc_csu_init to <no d...	0x4005cc	0x4003b0	0
14	Core 4	Linear	Function <no debug info>	0x4003b0		0
15	Core 4	Linear	Function <no debug info>	0x4003b4		0
16	Core 4	Linear	Function <no debug info>	0x4003bc		0
17	Core 4	Linear	Function __libc_csu_init	0x4005d0		0
18	Core 4	Branch	Branch from __libc_csu_init to <no d...	0x4005e8	0x400538	0
19	Core 4	Linear	Function <no debug info>	0x400538		0
20	Core 4	Linear	Function __libc_csu_init	0x4005ec		0
21	Core 4	Linear	Function __libc_csu_init	0x4005f0		0
22	Core 4	Branch	Branch from __libc_csu_init to __libc...	0x400604	0x4005d0	0
23	Core 4	Info	Trace On packet - ETM -> start tracin...			0
24	Core 4	Info	Context packet - ETM			0
25	Core 4	Software Co...	software context id = 3040			0

```
f.elf  linux.armv8-sdk1.8-ear6.satrace
root@localhost:/home/user#
/home/user>
root@localhost:/home/user#
root@localhost:/home/user#
root@localhost:/home#
/home>
root@localhost:/home#
root@localhost:/home#
root@localhost:/home/user#
/home/user>
root@localhost:/home/user#
root@localhost:/home/user#
f.elf  linux.armv8-sdk1.8-ear6.satrace
root@localhost:/home/user#
/home/user>
root@localhost:/home/user#
root@localhost:/home/user#
```

Figure 45. Launch shell

7. In the **Terminal** window, run the satrace using the following command:

```
./linux.armv8.satrace/bin/ls.linux.satrace -v ./test-linux.elf
```



```
root@ls2085aqds:~/Test_Linux_Demo# ./linux.armv8.satrace/bin/ls.linux.satrace -
v ./test-linux.elf
User space trace
Application  : `./test-linux.elf`
Arguments    :
Starting `./test-linux.elf`
Hello ARM World!
User application exit status : 0
Master process
Relocation file  : `/home/root/Test_Linux_Demo/test-linux.rlog`
Trace file  : `/home/root/Test_Linux_Demo/test-linux.dat`
Collecting trace ...
Archive file  : `/home/root/Test_Linux_Demo/test-linux.cwzsa`
Creating archive ....
Archiving /home/root/Test_Linux_Demo/linux.armv8.satrace/config/PlatformConfig.xml
Archiving /home/root/Test_Linux_Demo/linux.armv8.satrace/bin/test-linux.resultsConfig
Archiving /home/root/Test_Linux_Demo/test-linux.dat
Archiving /home/root/Test_Linux_Demo/test-linux.elf
Archiving /home/root/Test_Linux_Demo/test-linux.rlog
Archiving /lib/ld-2.19-2014.04.so
Archiving /lib/libc-2.19-2014.04.so
Archiving /lib/libdl-2.19-2014.04.so
Archiving /lib/libm-2.19-2014.04.so
Archiving /lib/libpthread-2.19-2014.04.so
Archiving /lib/librt-2.19-2014.04.so
root@ls2085aqds:~/Test_Linux_Demo#
```

Figure 46. Terminals Window

8. The tracing starts and collects the data in **.cwzsa* file as shown below.

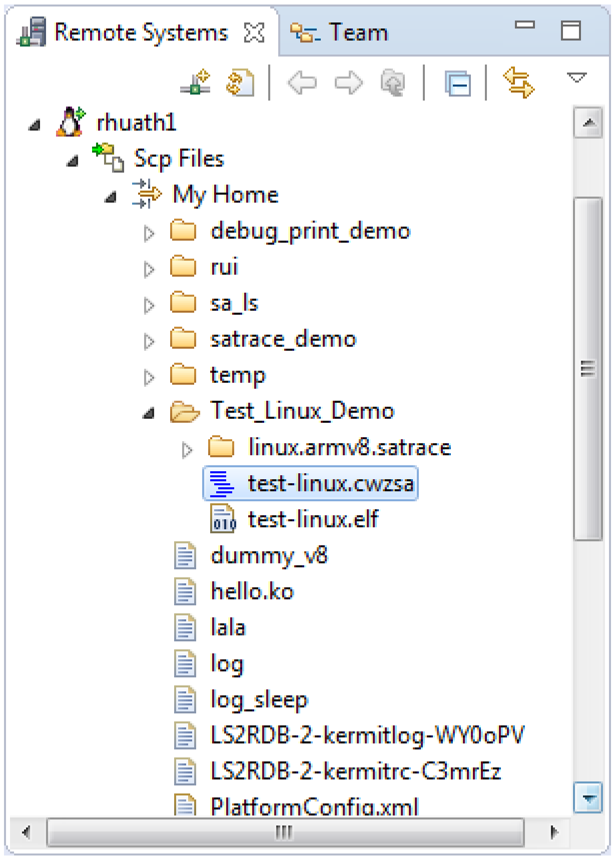
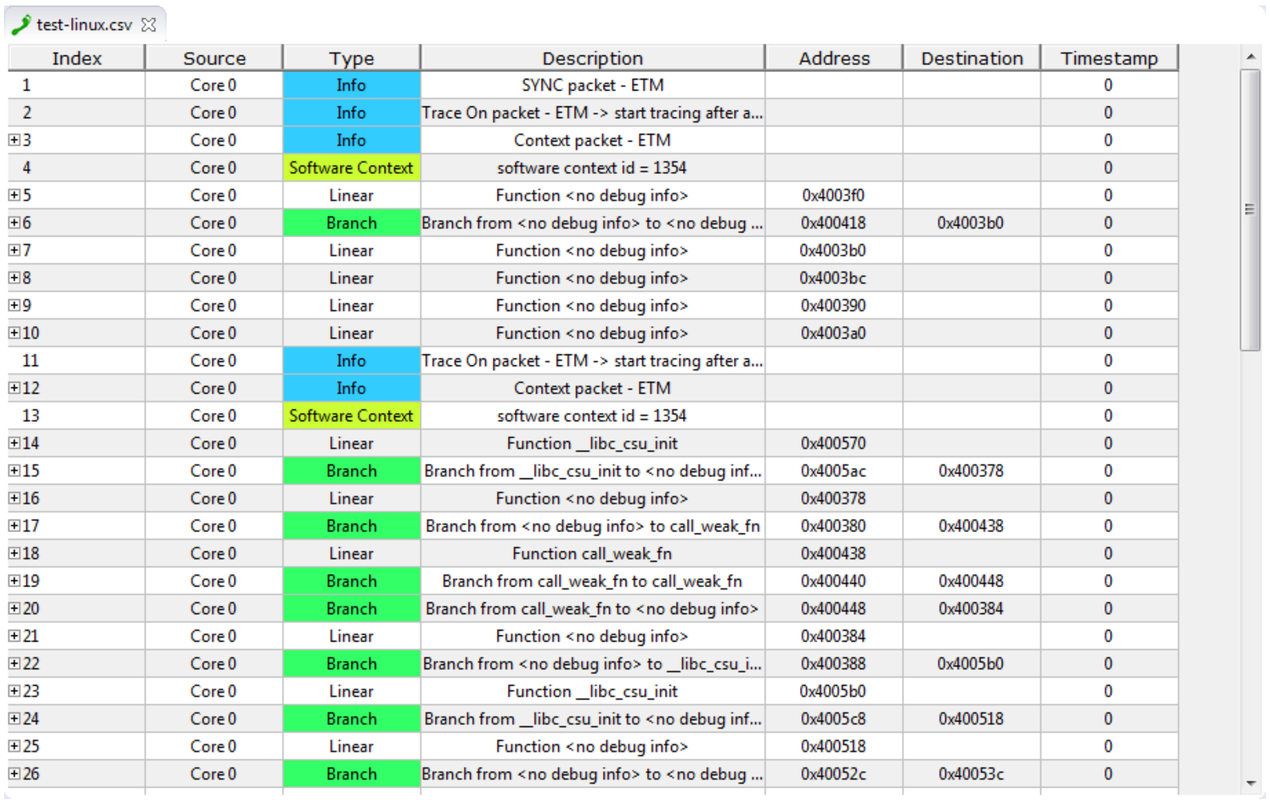


Figure 47. Trace data file

9. Double-click the **.cwzsa* file to trigger an import action. To decode the trace data, open the Analysis Results view and click on Trace link.

   This will decode the trace data and opens the Trace viewer.


test-linux.csv

Index	Source	Type	Description	Address	Destination	Timestamp
1	Core 0	Info	SYNC packet - ETM			0
2	Core 0	Info	Trace On packet - ETM -> start tracing after a...			0
⊞3	Core 0	Info	Context packet - ETM			0
4	Core 0	Software Context	software context id = 1354			0
⊞5	Core 0	Linear	Function <no debug info>	0x4003f0		0
⊞6	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x400418	0x4003b0	0
⊞7	Core 0	Linear	Function <no debug info>	0x4003b0		0
⊞8	Core 0	Linear	Function <no debug info>	0x4003bc		0
⊞9	Core 0	Linear	Function <no debug info>	0x400390		0
⊞10	Core 0	Linear	Function <no debug info>	0x4003a0		0
11	Core 0	Info	Trace On packet - ETM -> start tracing after a...			0
⊞12	Core 0	Info	Context packet - ETM			0
13	Core 0	Software Context	software context id = 1354			0
⊞14	Core 0	Linear	Function __libc_csu_init	0x400570		0
⊞15	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x4005ac	0x400378	0
⊞16	Core 0	Linear	Function <no debug info>	0x400378		0
⊞17	Core 0	Branch	Branch from <no debug info> to call_weak_fn	0x400380	0x400438	0
⊞18	Core 0	Linear	Function call_weak_fn	0x400438		0
⊞19	Core 0	Branch	Branch from call_weak_fn to call_weak_fn	0x400440	0x400448	0
⊞20	Core 0	Branch	Branch from call_weak_fn to <no debug info>	0x400448	0x400384	0
⊞21	Core 0	Linear	Function <no debug info>	0x400384		0
⊞22	Core 0	Branch	Branch from <no debug info> to __libc_csu_i...	0x400388	0x4005b0	0
⊞23	Core 0	Linear	Function __libc_csu_init	0x4005b0		0
⊞24	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x4005c8	0x400518	0
⊞25	Core 0	Linear	Function <no debug info>	0x400518		0
⊞26	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x40052c	0x40053c	0

Figure 48. Trace data file

5.2 Importing and decoding trace collected using satrace

This section explains the steps required to import and decode the trace data using satrace utility.

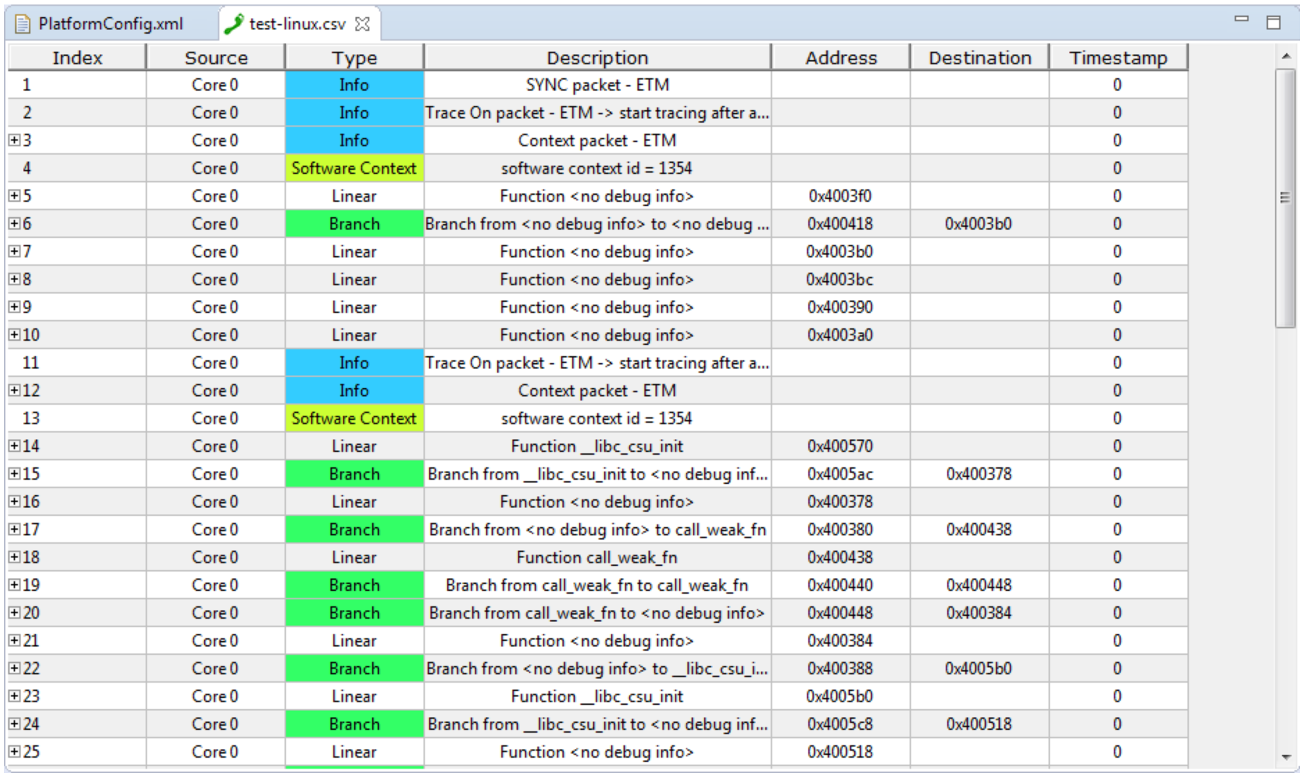
NOTE

Use the apropriate satrace agent from `CW_INSTALL_DIR}\CW_ARMv8\ARMv8\sa_ls\`:

- `linux.armv8-sdk1.8-ear6.satrace` for linux image with little endian, on all targets, starting with ARMv8 SDK ear6
- `linux.armv8.satrace` for linux image with little endian, on all targets, before ARMv8 SDK ear6
- `linux.armv8-ls1043-be.satrace` for linux image with big endian, on LS1043

The Linux trace mechanism is independent of CodeWarrior. Trace data can be collected from LS2088A QDS board without using CodeWarrior. For more details, see an Application Note *AN5129: Linux hardware trace for ARMv8 user space and kernel space applications.*

The standalone tracing tool generates **.cwzsa* file, an archive type that can be imported and fully decoded using ARMv8 decoder or ARMv8 CodeWarrior. The generated archive can be imported and fully decoded in CW ARMv8 after a drag-and-drop action. As a result, the trace viewer is displayed with the decoded trace.



Index	Source	Type	Description	Address	Destination	Timestamp
1	Core 0	Info	SYNC packet - ETM			0
2	Core 0	Info	Trace On packet - ETM -> start tracing after a...			0
⊞3	Core 0	Info	Context packet - ETM			0
4	Core 0	Software Context	software context id = 1354			0
⊞5	Core 0	Linear	Function <no debug info>	0x4003f0		0
⊞6	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x400418	0x4003b0	0
⊞7	Core 0	Linear	Function <no debug info>	0x4003b0		0
⊞8	Core 0	Linear	Function <no debug info>	0x4003bc		0
⊞9	Core 0	Linear	Function <no debug info>	0x400390		0
⊞10	Core 0	Linear	Function <no debug info>	0x4003a0		0
11	Core 0	Info	Trace On packet - ETM -> start tracing after a...			0
⊞12	Core 0	Info	Context packet - ETM			0
13	Core 0	Software Context	software context id = 1354			0
⊞14	Core 0	Linear	Function __libc_csu_init	0x400570		0
⊞15	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x4005ac	0x400378	0
⊞16	Core 0	Linear	Function <no debug info>	0x400378		0
⊞17	Core 0	Branch	Branch from <no debug info> to call_weak_fn	0x400380	0x400438	0
⊞18	Core 0	Linear	Function call_weak_fn	0x400438		0
⊞19	Core 0	Branch	Branch from call_weak_fn to call_weak_fn	0x400440	0x400448	0
⊞20	Core 0	Branch	Branch from call_weak_fn to <no debug info>	0x400448	0x400384	0
⊞21	Core 0	Linear	Function <no debug info>	0x400384		0
⊞22	Core 0	Branch	Branch from <no debug info> to __libc_csu_i...	0x400388	0x4005b0	0
⊞23	Core 0	Linear	Function __libc_csu_init	0x4005b0		0
⊞24	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x4005c8	0x400518	0
⊞25	Core 0	Linear	Function <no debug info>	0x400518		0

Figure 49. Trace data

The trace data is displayed in **Trace** viewer. You will find the profiling data files in the same folder where *segfault.cwzsa* file is present. The profiling data is accessible after decoding through Analysis Results view.

Chapter 6
Linux Kernel and User Applications Debug Print Tool

This chapter explains in detail about the Linux Kernel Debug Print tool.

The Linux Kernel Debug Print tool encapsulates a target server responsible for collecting Kernel Ring Buffer log user space applications messages in the unformatted way and a host which requests periodically the kernel log data from the server and displays it in a view.

This tool's main objective is to provide a user-friendly way of monitoring kernel activity in a CodeWarrior console. It is composed of several modules:

- **Target side**:

  Debug Print server – reads on demand the Kernel Ring Buffer log. It optionally clears the log and sends it to the clients using TCP/IP connection. It collects the redirected standard output from the user space applications.

  Debug Print dynamic library - is responsible for redirection of the user space application's standard output messages to the target server.

- **Host side**:

  Debug Print probe – is the actual client of the Debug Print server; it can be started from the **Debug Print** view. When started, it reads periodically the kernel log data from the server and sends it to the **Debug Print** view to display the kernel log data and other communication messages.

  Debug Print view – displays the log data and other communication messages in a user-friendly manner, also allows you to filter the displayed data on the basis of timestamp, module name/application path and pid, or a custom string contained in each log message.

NOTE

The Arm binaries have been compiled with tool chain *gcc-linaro-aarch64-linux-gnu-4.8.3* and LS2 SDK.

Before working on the Debug Print tool, check that TCP/IP communication is established between the host and the target. Below are the steps that are performed in order to see the functionality of the **Debug Print** tool.

1. Deploy the Software Analysis target binaries on the target using Remote System Explorers view, or an SCP connection, or if you have the target root file system on NFS, you can copy *ls.target.server* and *libls.linux.debugprint.lib.so** to the host location *[NFS_PATH]/home/root*).
2. The debug print target server cross-compiled for Arm is located in CodeWarrior in directory: *<CWInstallDir>/ARMv8/sa_ls/linux.armv8.debugprint/bin*, which needs to be copied on the target (for example, to the home directory), using command `ls.target.server [PORT] [-k]` and requires a single argument; the port number on which clients will listen. If not specified, it will start on the default port 5000. Specify `-k` to keep the kernel buffer unaltered (same as dmesg), with a server processing overhead.

   Start a *ssh* console on the target and then start the server:

   ```
   # ssh root@target_ip_address
   ```

   ```
   # ./ls.target.server
   ```

3. The dynamic library cross-compiled for Arm is located in CodeWarrior directory at: *<CWInstallDir>/ARMv8/sa_ls/linux.armv8.debugprint/lib*, which needs to be copied on the target using the Remote Systems Explorer (RSE) view . This library must be loaded by the shell before the C runtime when you are running the user space applications which need to be monitored by setting the environment variable *LD_PRELOAD*.

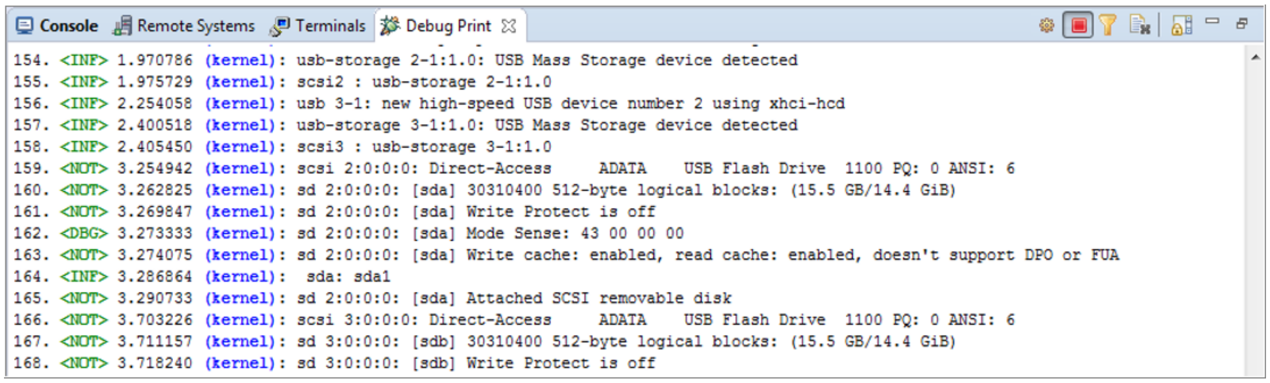
   Preload the debug print library and run the test application:

```
# export LD_PRELOAD=~/libls.linux.debugprint.lib.so

# ./test-arm
```

You will notice next time that the test application will not display any of its standard output messages to the console, but only its standard error messages. The standard output is sent to the target server.

4. On the host machine, open the **Debug Print** view. The Debug Print Probe can be started from the **Debug Print** view and it communicates using TCP/IP connection with the server. When started, it reads periodically the kernel log data from the server and sends it to the **Debug Print** view to display. To start the **Debug Print** view, select **Window > Show View > Other > Software Analysis > Debug Print**. The **Debug Print** view appears.



```
154. <INF> 1.970786 (kernel): usb-storage 2-1:1.0: USB Mass Storage device detected
155. <INF> 1.975729 (kernel): scsi2 : usb-storage 2-1:1.0
156. <INF> 2.254058 (kernel): usb 3-1: new high-speed USB device number 2 using xhci-hcd
157. <INF> 2.400518 (kernel): usb-storage 3-1:1.0: USB Mass Storage device detected
158. <INF> 2.405450 (kernel): scsi3 : usb-storage 3-1:1.0
159. <NOT> 3.254942 (kernel): scsi 2:0:0:0: Direct-Access     ADATA    USB Flash Drive  1100 PQ: 0 ANSI: 6
160. <NOT> 3.262825 (kernel): sd 2:0:0:0: [sda] 30310400 512-byte logical blocks: (15.5 GB/14.4 GiB)
161. <NOT> 3.269847 (kernel): sd 2:0:0:0: [sda] Write Protect is off
162. <DBG> 3.273333 (kernel): sd 2:0:0:0: [sda] Mode Sense: 43 00 00 00
163. <NOT> 3.274075 (kernel): sd 2:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
164. <INF> 3.286864 (kernel):  sda: sda1
165. <NOT> 3.290733 (kernel): sd 2:0:0:0: [sda] Attached SCSI removable disk
166. <NOT> 3.703226 (kernel): scsi 3:0:0:0: Direct-Access     ADATA    USB Flash Drive  1100 PQ: 0 ANSI: 6
167. <NOT> 3.711157 (kernel): sd 3:0:0:0: [sdb] 30310400 512-byte logical blocks: (15.5 GB/14.4 GiB)
168. <NOT> 3.718240 (kernel): sd 3:0:0:0: [sdb] Write Protect is off
```

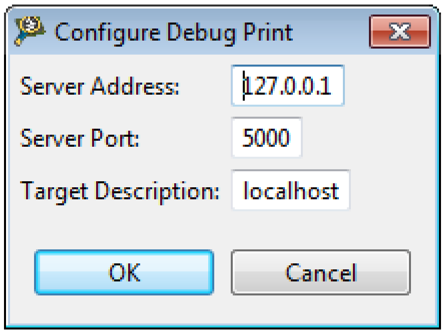
Figure 50. Debug Print view

The table below describes the icons displayed on the Debug Print viewer.

Table 12. Debug Print viewer icons

Icons	Description
Clear All	Removes all text from the view.
▶ ■ Start/Stop	Two-state button used for starting and stopping the Debug Print probe.
Scroll Lock/Unlock	Two-state button used for locking and unlocking the scrollbar. If the scrollbar is unlocked, it would always auto-scroll to the latest Debug Print message.
Configure	Opens a dialog for entering the server address and port.
Create Debug Print Filters	Opens a dialog for configuring what information is to be displayed in the **Debug Print** view (specific to timestamps, module name/application path and pid, other string patterns).

5. Click the **Configure** icon, enter the server address and port (for example, address 192.168.0.2, port 5000 – must be the same as for the server at which the server will listen to client, and the target description).



Configure Debug Print

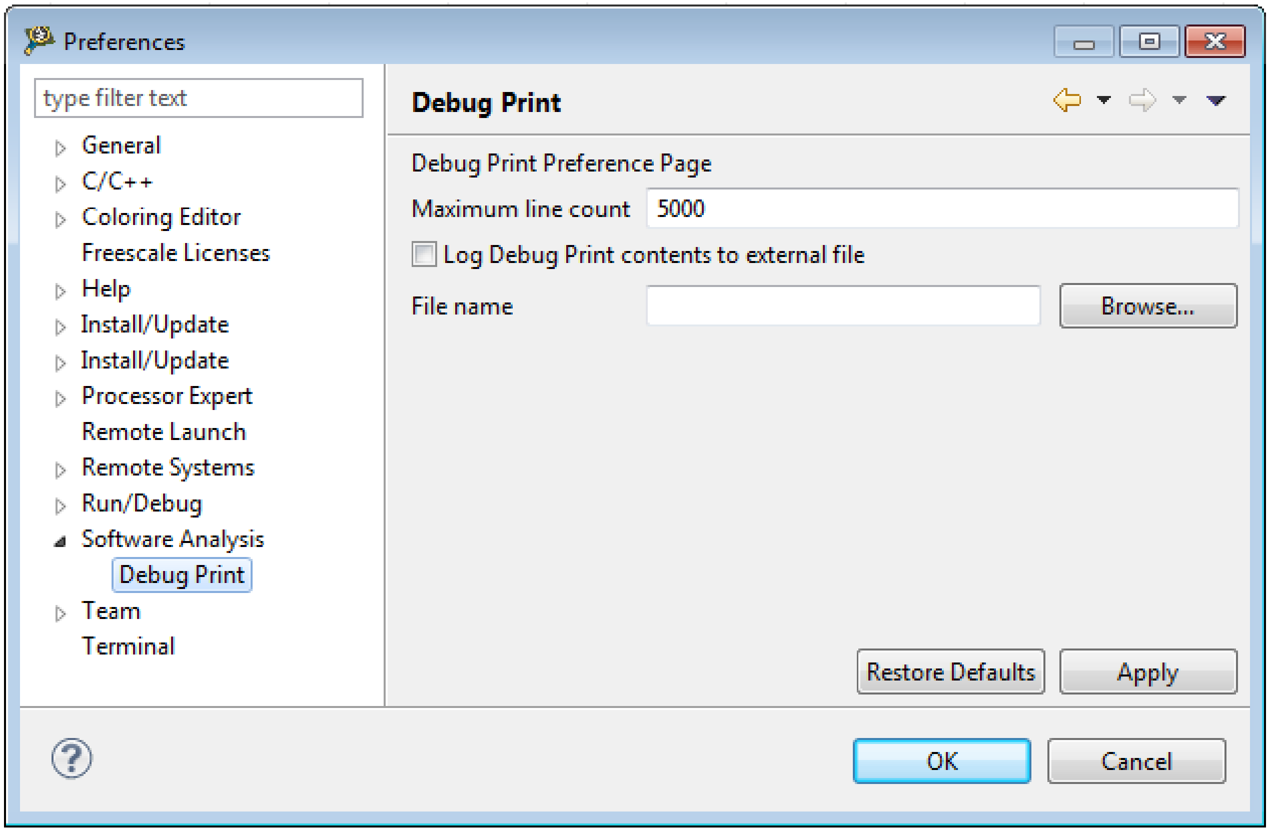
Server Address: 127.0.0.1

Server Port: 5000

Target Description: localhost

Figure 51. Configure Debug Print dialog

There is also a **Preference** page associated to this view, which can be accessed by clicking **Window > Preferences**, expand **Software Analysis** node and then select **Debug Print**.



Debug Print

Debug Print Preference Page

Maximum line count: 5000

Log Debug Print contents to external file: [ ]

File name:

Figure 52. Preferences dialog

The following **Debug Print** settings are available in the **Preference** dialog:

Table 13. Debug Print settings

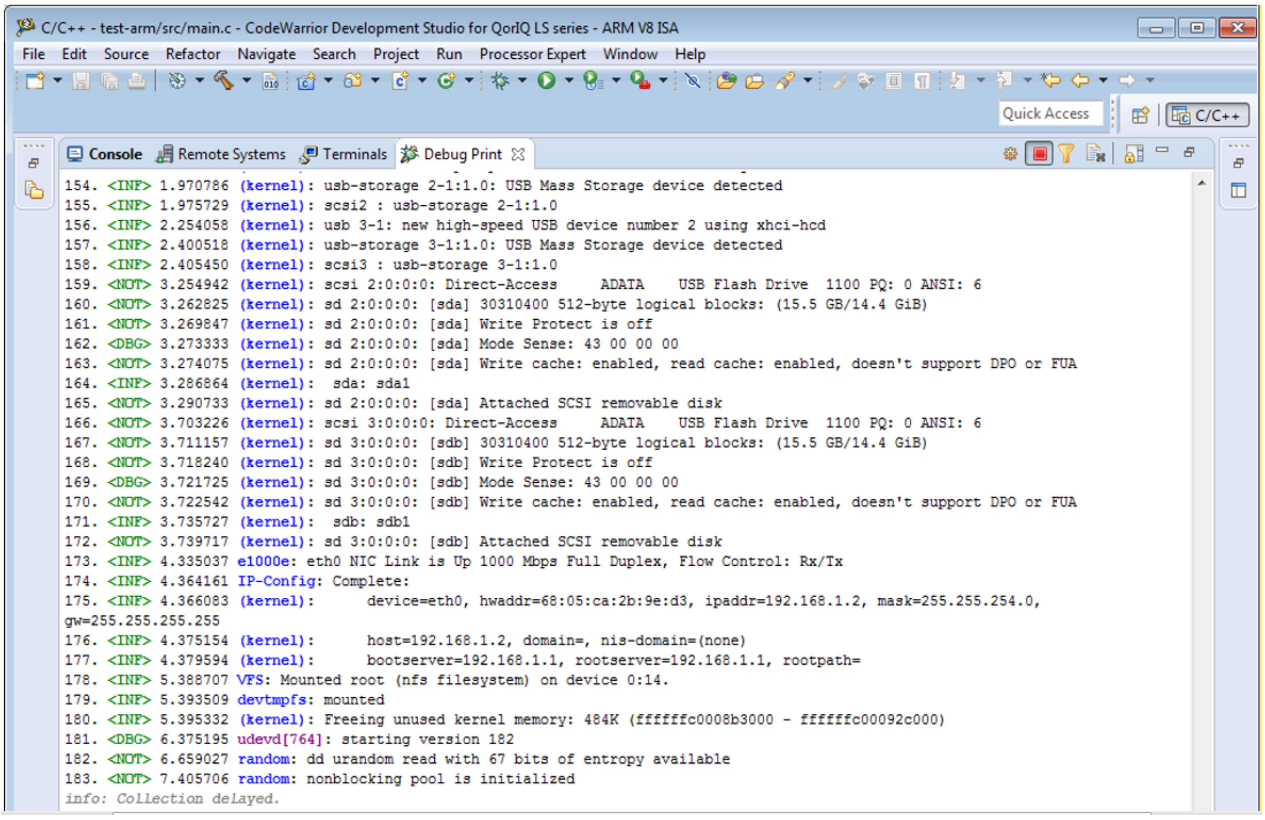
Options	Description
Maximum line count	Limits the number of lines the **Debug Print** view should display. If this limit is exceeded, the old messages are deleted.
Log Debug Print contents to external file	If selected, the messages will be appended to an external file besides displaying them into the **Debug Print** view.

Table continues on the next page...

Table 13. Debug Print settings (continued)

Options	Description
File name	Path for the external log file

6. Click the **Start** icon; you will see the kernel log messages are being populated in the view's text area.


C/C++ - test-arm/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Console
Remote Systems
Terminals
Debug Print

```
154. <INF> 1.970786 (kernel): usb-storage 2-1:1.0: USB Mass Storage device detected
155. <INF> 1.975729 (kernel): scsi2 : usb-storage 2-1:1.0
156. <INF> 2.254058 (kernel): usb 3-1: new high-speed USB device number 2 using xhci-hcd
157. <INF> 2.400518 (kernel): usb-storage 3-1:1.0: USB Mass Storage device detected
158. <INF> 2.405450 (kernel): scsi3 : usb-storage 3-1:1.0
159. <NOT> 3.254942 (kernel): scsi 2:0:0:0: Direct-Access     ADATA    USB Flash Drive  1100 PQ: 0 ANSI: 6
160. <NOT> 3.262825 (kernel): sd 2:0:0:0: [sda] 30310400 512-byte logical blocks: (15.5 GB/14.4 GiB)
161. <NOT> 3.269847 (kernel): sd 2:0:0:0: [sda] Write Protect is off
162. <DBG> 3.273333 (kernel): sd 2:0:0:0: [sda] Mode Sense: 43 00 00 00
163. <NOT> 3.274075 (kernel): sd 2:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
164. <INF> 3.286864 (kernel):  sda: sda1
165. <NOT> 3.290733 (kernel): sd 2:0:0:0: [sda] Attached SCSI removable disk
166. <NOT> 3.703226 (kernel): scsi 3:0:0:0: Direct-Access     ADATA    USB Flash Drive  1100 PQ: 0 ANSI: 6
167. <NOT> 3.711157 (kernel): sd 3:0:0:0: [sdb] 30310400 512-byte logical blocks: (15.5 GB/14.4 GiB)
168. <NOT> 3.718240 (kernel): sd 3:0:0:0: [sdb] Write Protect is off
169. <DBG> 3.721725 (kernel): sd 3:0:0:0: [sdb] Mode Sense: 43 00 00 00
170. <NOT> 3.722542 (kernel): sd 3:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
171. <INF> 3.735727 (kernel):  sdb: sdb1
172. <NOT> 3.739717 (kernel): sd 3:0:0:0: [sdb] Attached SCSI removable disk
173. <INF> 4.335037 e1000e: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: Rx/Tx
174. <INF> 4.364161 IP-Config: Complete:
175. <INF> 4.366083 (kernel):      device=eth0, hwaddr=68:05:ca:2b:9e:d3, ipaddr=192.168.1.2, mask=255.255.254.0,
gw=255.255.255.255
176. <INF> 4.375154 (kernel):      host=192.168.1.2, domain=, nis-domain=(none)
177. <INF> 4.379594 (kernel):      bootserver=192.168.1.1, rootserver=192.168.1.1, rootpath=
178. <INF> 5.388707 VFS: Mounted root (nfs filesystem) on device 0:14.
179. <INF> 5.393509 devtmpfs: mounted
180. <INF> 5.395332 (kernel): Freeing unused kernel memory: 484K (ffffffc0008b3000 - ffffffc00092c000)
181. <DBG> 6.375195 udevd[764]: starting version 182
182. <NOT> 6.659027 random: dd urandom read with 67 bits of entropy available
183. <NOT> 7.405706 random: nonblocking pool is initialized
info: Collection delayed.
```

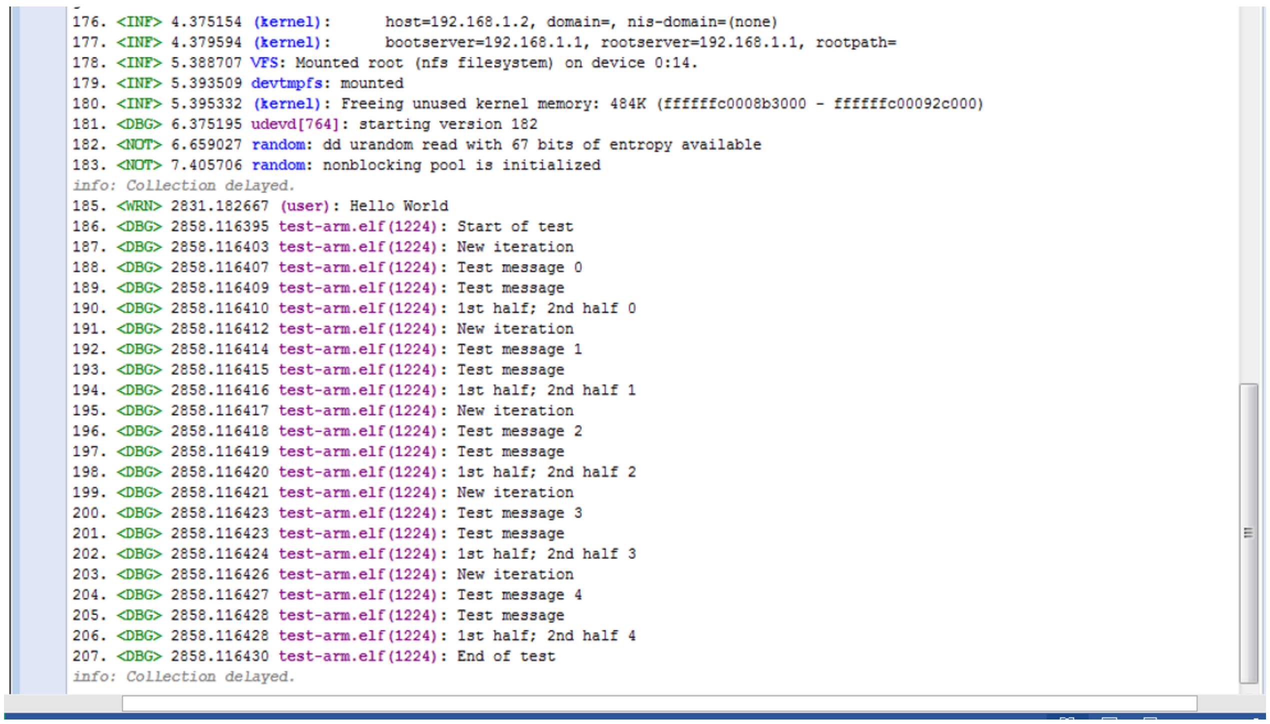
Figure 53. Debug Print view - messages from server

7. Open another console on the target in the same directory, preload the debug print library and run the test application:

```
# export LD_PRELOAD=~/libls.linux.debugprint.lib.so
```

```
# ./test-arm
```

8. You will see the application messages getting appended in the **Debug Print** view:



```
176. <INF> 4.375154 (kernel):      host=192.168.1.2, domain=, nis-domain=(none)
177. <INF> 4.379594 (kernel):      bootserver=192.168.1.1, rootserver=192.168.1.1, rootpath=
178. <INF> 5.388707 VFS: Mounted root (nfs filesystem) on device 0:14.
179. <INF> 5.393509 devtmpfs: mounted
180. <INF> 5.395332 (kernel): Freeing unused kernel memory: 484K (ffffffc0008b3000 - ffffffc00092c000)
181. <DBG> 6.375195 udevd[764]: starting version 182
182. <NOT> 6.659027 random: dd urandom read with 67 bits of entropy available
183. <NOT> 7.405706 random: nonblocking pool is initialized
info: Collection delayed.
185. <WRN> 2831.182667 (user): Hello World
186. <DBG> 2858.116395 test-arm.elf(1224): Start of test
187. <DBG> 2858.116403 test-arm.elf(1224): New iteration
188. <DBG> 2858.116407 test-arm.elf(1224): Test message 0
189. <DBG> 2858.116409 test-arm.elf(1224): Test message
190. <DBG> 2858.116410 test-arm.elf(1224): 1st half; 2nd half 0
191. <DBG> 2858.116412 test-arm.elf(1224): New iteration
192. <DBG> 2858.116414 test-arm.elf(1224): Test message 1
193. <DBG> 2858.116415 test-arm.elf(1224): Test message
194. <DBG> 2858.116416 test-arm.elf(1224): 1st half; 2nd half 1
195. <DBG> 2858.116417 test-arm.elf(1224): New iteration
196. <DBG> 2858.116418 test-arm.elf(1224): Test message 2
197. <DBG> 2858.116419 test-arm.elf(1224): Test message
198. <DBG> 2858.116420 test-arm.elf(1224): 1st half; 2nd half 2
199. <DBG> 2858.116421 test-arm.elf(1224): New iteration
200. <DBG> 2858.116423 test-arm.elf(1224): Test message 3
201. <DBG> 2858.116423 test-arm.elf(1224): Test message
202. <DBG> 2858.116424 test-arm.elf(1224): 1st half; 2nd half 3
203. <DBG> 2858.116426 test-arm.elf(1224): New iteration
204. <DBG> 2858.116427 test-arm.elf(1224): Test message 4
205. <DBG> 2858.116428 test-arm.elf(1224): Test message
206. <DBG> 2858.116428 test-arm.elf(1224): 1st half; 2nd half 4
207. <DBG> 2858.116430 test-arm.elf(1224): End of test
info: Collection delayed.
```

Figure 54. Debug Print view - application messages

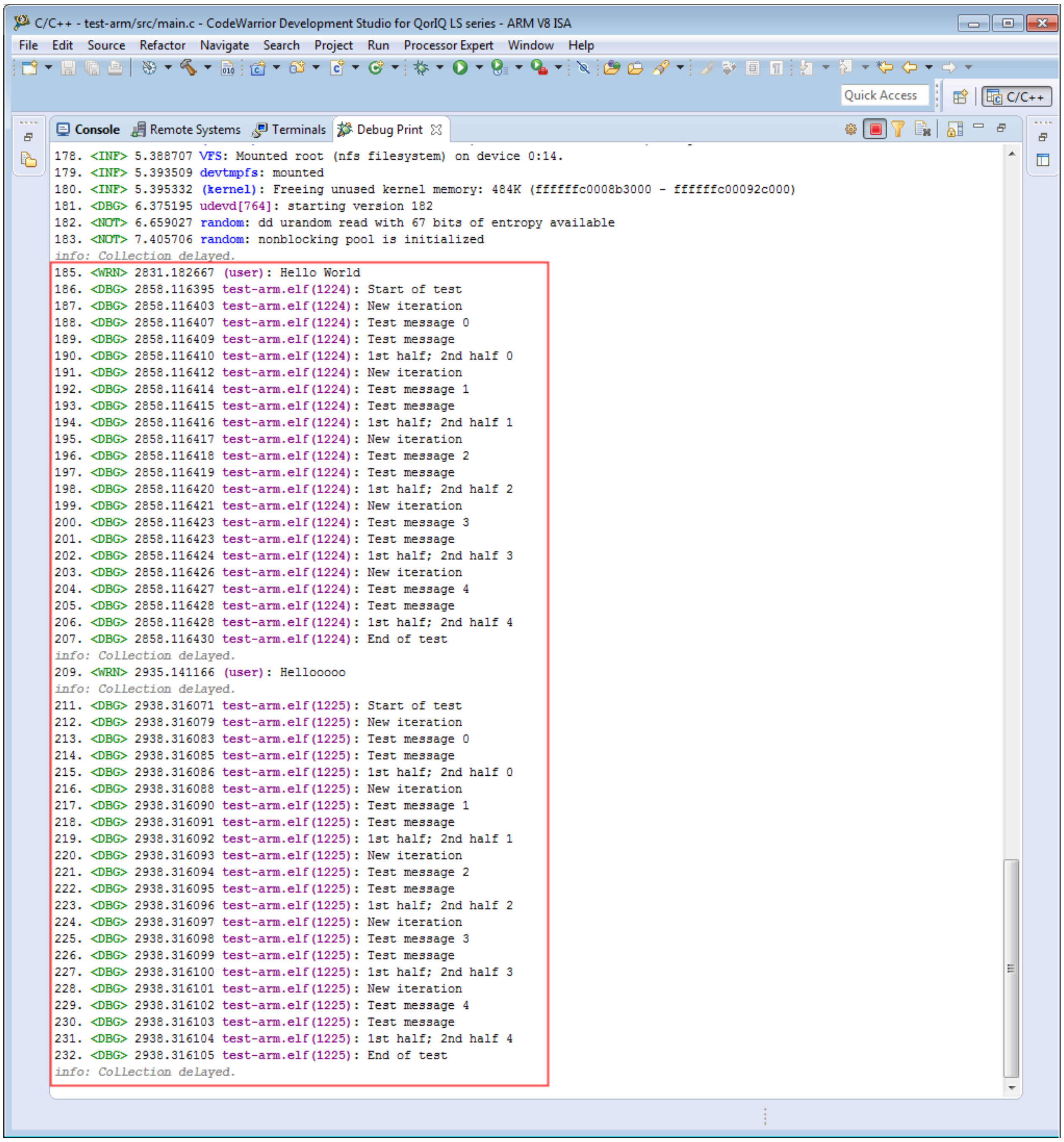
9. To see the real time functionality of the **Debug Print** view, add some more messages to the view, both from kernel and the test application from the same console where the test application was running on the target:

```
# echo Hello World > /dev/kmsg

# ./test-arm

# echo Hellooooooo > /dev/kmsg
```

10. See the new messages displayed in the **Debug Print** text area as you enter them in the target shell:


C/C++ - test-arm/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Console
Remote Systems
Terminals
Debug Print

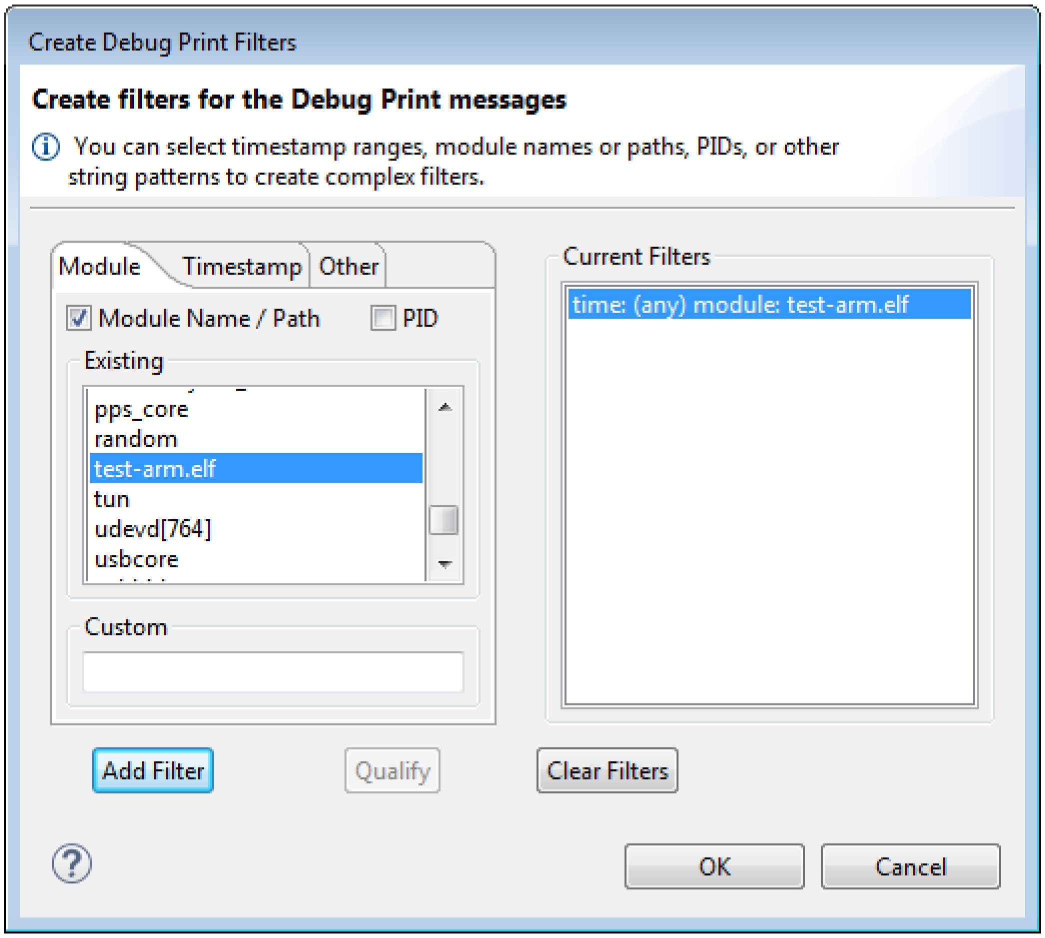
```
178. <INF> 5.388707 VFS: Mounted root (nfs filesystem) on device 0:14.
179. <INF> 5.393509 devtmpfs: mounted
180. <INF> 5.395332 (kernel): Freeing unused kernel memory: 484K (ffffffc0008b3000 - ffffffc00092c000)
181. <DBG> 6.375195 udevd[764]: starting version 182
182. <NOT> 6.659027 random: dd urandom read with 67 bits of entropy available
183. <NOT> 7.405706 random: nonblocking pool is initialized
info: Collection delayed.
185. <WRN> 2831.182667 (user): Hello World
186. <DBG> 2858.116395 test-arm.elf(1224): Start of test
187. <DBG> 2858.116403 test-arm.elf(1224): New iteration
188. <DBG> 2858.116407 test-arm.elf(1224): Test message 0
189. <DBG> 2858.116409 test-arm.elf(1224): Test message
190. <DBG> 2858.116410 test-arm.elf(1224): 1st half; 2nd half 0
191. <DBG> 2858.116412 test-arm.elf(1224): New iteration
192. <DBG> 2858.116414 test-arm.elf(1224): Test message 1
193. <DBG> 2858.116415 test-arm.elf(1224): Test message
194. <DBG> 2858.116416 test-arm.elf(1224): 1st half; 2nd half 1
195. <DBG> 2858.116417 test-arm.elf(1224): New iteration
196. <DBG> 2858.116418 test-arm.elf(1224): Test message 2
197. <DBG> 2858.116419 test-arm.elf(1224): Test message
198. <DBG> 2858.116420 test-arm.elf(1224): 1st half; 2nd half 2
199. <DBG> 2858.116421 test-arm.elf(1224): New iteration
200. <DBG> 2858.116423 test-arm.elf(1224): Test message 3
201. <DBG> 2858.116423 test-arm.elf(1224): Test message
202. <DBG> 2858.116424 test-arm.elf(1224): 1st half; 2nd half 3
203. <DBG> 2858.116426 test-arm.elf(1224): New iteration
204. <DBG> 2858.116427 test-arm.elf(1224): Test message 4
205. <DBG> 2858.116428 test-arm.elf(1224): Test message
206. <DBG> 2858.116428 test-arm.elf(1224): 1st half; 2nd half 4
207. <DBG> 2858.116430 test-arm.elf(1224): End of test
info: Collection delayed.
209. <WRN> 2935.141166 (user): Hellooooo
info: Collection delayed.
211. <DBG> 2938.316071 test-arm.elf(1225): Start of test
212. <DBG> 2938.316079 test-arm.elf(1225): New iteration
213. <DBG> 2938.316083 test-arm.elf(1225): Test message 0
214. <DBG> 2938.316085 test-arm.elf(1225): Test message
215. <DBG> 2938.316086 test-arm.elf(1225): 1st half; 2nd half 0
216. <DBG> 2938.316088 test-arm.elf(1225): New iteration
217. <DBG> 2938.316090 test-arm.elf(1225): Test message 1
218. <DBG> 2938.316091 test-arm.elf(1225): Test message
219. <DBG> 2938.316092 test-arm.elf(1225): 1st half; 2nd half 1
220. <DBG> 2938.316093 test-arm.elf(1225): New iteration
221. <DBG> 2938.316094 test-arm.elf(1225): Test message 2
222. <DBG> 2938.316095 test-arm.elf(1225): Test message
223. <DBG> 2938.316096 test-arm.elf(1225): 1st half; 2nd half 2
224. <DBG> 2938.316097 test-arm.elf(1225): New iteration
225. <DBG> 2938.316098 test-arm.elf(1225): Test message 3
226. <DBG> 2938.316099 test-arm.elf(1225): Test message
227. <DBG> 2938.316100 test-arm.elf(1225): 1st half; 2nd half 3
228. <DBG> 2938.316101 test-arm.elf(1225): New iteration
229. <DBG> 2938.316102 test-arm.elf(1225): Test message 4
230. <DBG> 2938.316103 test-arm.elf(1225): Test message
231. <DBG> 2938.316104 test-arm.elf(1225): 1st half; 2nd half 4
232. <DBG> 2938.316105 test-arm.elf(1225): End of test
info: Collection delayed.
```

Figure 55. Debug Print view - messages from server

6.1 Filtering

This tool has a powerful filtering engine that allows you to see the information that you need. It also allows displaying data filtered by timestamp, module name/application path and PID, or a custom string contained by each log message.

The **Create Debug Print Filters** configuration dialog allows creation of multiple filters, each of them able to match the module name, application path or PID of the messages displayed by the **Debug Print** view. These filters are OR-ed, which means that the view will display all messages which match at least one of the filters.


Create Debug Print Filters

Create filters for the Debug Print messages

You can select timestamp ranges, module names or paths, PIDs, or other string patterns to create complex filters.

Figure 56. Create Debug Print Filters dialog

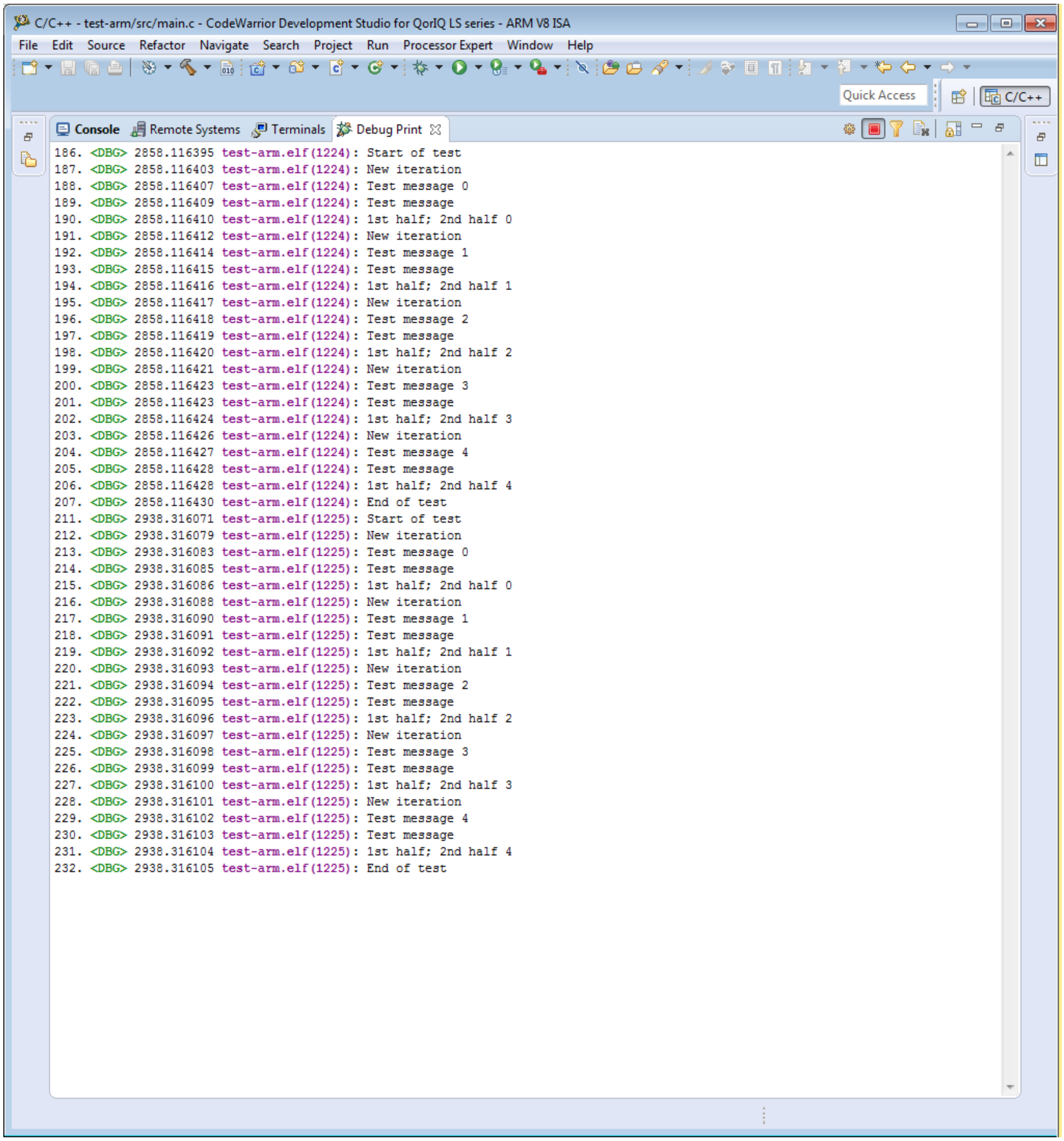
This dialog has three tabs:

- **Module** tab: allows creation of new filters, by selecting from the **Existing** list a module name/application path, PID, or both (if available). Click **Add Filter** to add the filter in the **Current Filters** list. These filters can be qualified with a timestamp range or a string pattern.

  The **Existing** list contains all the module names/application paths/PIDs from the messages already displayed in the **Debug Print** view. When you want to filter messages from a certain module or application that is not started or did not print any messages yet, you can manually enter the module name/path or PID in the **Custom** text box.

  When no module filter is selected, and no global qualification is selected, **(any)** is displayed in the **Current Filters**, which means that no filter is applied (all messages are displayed).

  The following figure shows the messages displayed in the Debug Print view using test-arm.elf filter:

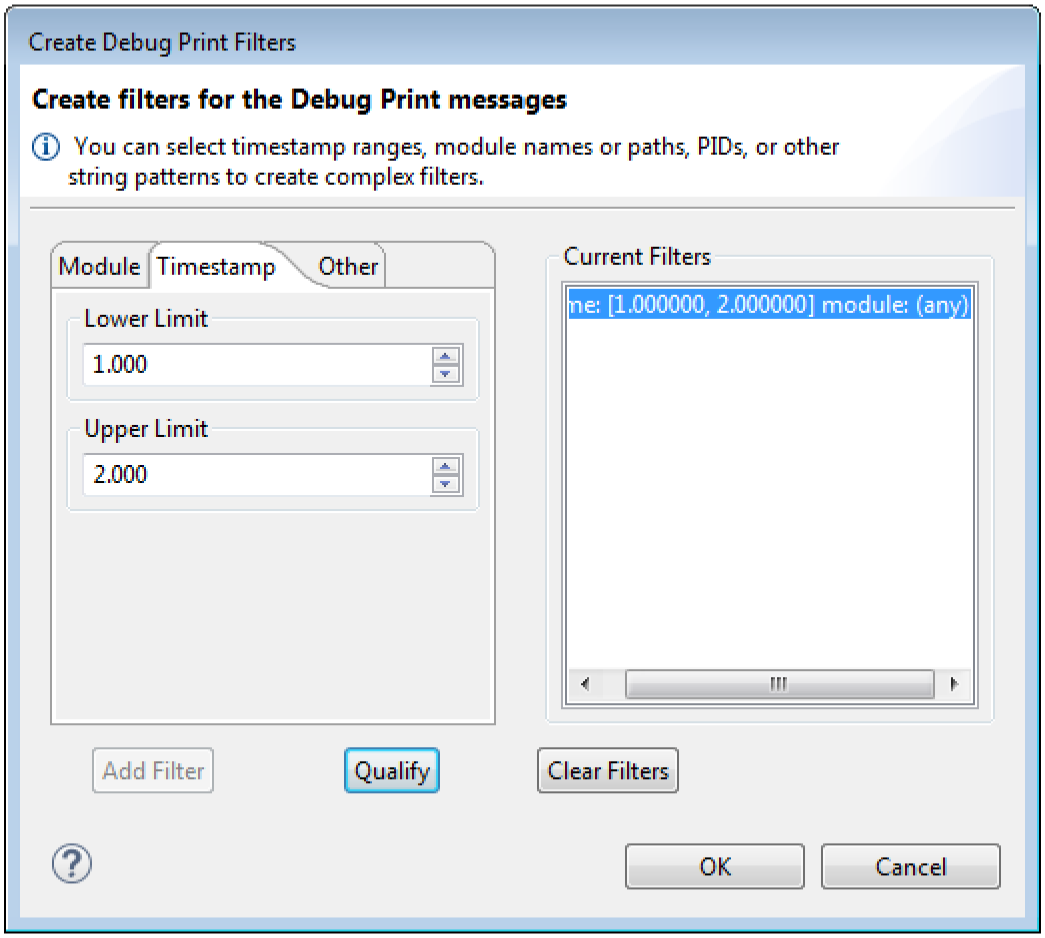

C/C++ - test-arm/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Console
Remote Systems
Terminals
Debug Print

```
186. <DBG> 2858.116395 test-arm.elf(1224): Start of test
187. <DBG> 2858.116403 test-arm.elf(1224): New iteration
188. <DBG> 2858.116407 test-arm.elf(1224): Test message 0
189. <DBG> 2858.116409 test-arm.elf(1224): Test message
190. <DBG> 2858.116410 test-arm.elf(1224): 1st half; 2nd half 0
191. <DBG> 2858.116412 test-arm.elf(1224): New iteration
192. <DBG> 2858.116414 test-arm.elf(1224): Test message 1
193. <DBG> 2858.116415 test-arm.elf(1224): Test message
194. <DBG> 2858.116416 test-arm.elf(1224): 1st half; 2nd half 1
195. <DBG> 2858.116417 test-arm.elf(1224): New iteration
196. <DBG> 2858.116418 test-arm.elf(1224): Test message 2
197. <DBG> 2858.116419 test-arm.elf(1224): Test message
198. <DBG> 2858.116420 test-arm.elf(1224): 1st half; 2nd half 2
199. <DBG> 2858.116421 test-arm.elf(1224): New iteration
200. <DBG> 2858.116423 test-arm.elf(1224): Test message 3
201. <DBG> 2858.116423 test-arm.elf(1224): Test message
202. <DBG> 2858.116424 test-arm.elf(1224): 1st half; 2nd half 3
203. <DBG> 2858.116426 test-arm.elf(1224): New iteration
204. <DBG> 2858.116427 test-arm.elf(1224): Test message 4
205. <DBG> 2858.116428 test-arm.elf(1224): Test message
206. <DBG> 2858.116428 test-arm.elf(1224): 1st half; 2nd half 4
207. <DBG> 2858.116430 test-arm.elf(1224): End of test
211. <DBG> 2938.316071 test-arm.elf(1225): Start of test
212. <DBG> 2938.316079 test-arm.elf(1225): New iteration
213. <DBG> 2938.316083 test-arm.elf(1225): Test message 0
214. <DBG> 2938.316085 test-arm.elf(1225): Test message
215. <DBG> 2938.316086 test-arm.elf(1225): 1st half; 2nd half 0
216. <DBG> 2938.316088 test-arm.elf(1225): New iteration
217. <DBG> 2938.316090 test-arm.elf(1225): Test message 1
218. <DBG> 2938.316091 test-arm.elf(1225): Test message
219. <DBG> 2938.316092 test-arm.elf(1225): 1st half; 2nd half 1
220. <DBG> 2938.316093 test-arm.elf(1225): New iteration
221. <DBG> 2938.316094 test-arm.elf(1225): Test message 2
222. <DBG> 2938.316095 test-arm.elf(1225): Test message
223. <DBG> 2938.316096 test-arm.elf(1225): 1st half; 2nd half 2
224. <DBG> 2938.316097 test-arm.elf(1225): New iteration
225. <DBG> 2938.316098 test-arm.elf(1225): Test message 3
226. <DBG> 2938.316099 test-arm.elf(1225): Test message
227. <DBG> 2938.316100 test-arm.elf(1225): 1st half; 2nd half 3
228. <DBG> 2938.316101 test-arm.elf(1225): New iteration
229. <DBG> 2938.316102 test-arm.elf(1225): Test message 4
230. <DBG> 2938.316103 test-arm.elf(1225): Test message
231. <DBG> 2938.316104 test-arm.elf(1225): 1st half; 2nd half 4
232. <DBG> 2938.316105 test-arm.elf(1225): End of test
```

Figure 57. Debug Print view displays messages with Module filter

- **Timestamp** tab: allows adding timestamp qualification to the existing filters, or a global qualification if no other filter is created (that is a generic filter which applies to all messages, with all module names, paths and PIDs).

  After the user choses the timestamp ranges in the Lower Limit/Upper Limit Spinners, you must click **Qualify** in order to add the timestamp qualification to all existing filters. If no filter exists, a global qualification is performed.


Create Debug Print Filters

Create filters for the Debug Print messages

You can select timestamp ranges, module names or paths, PIDs, or other string patterns to create complex filters.

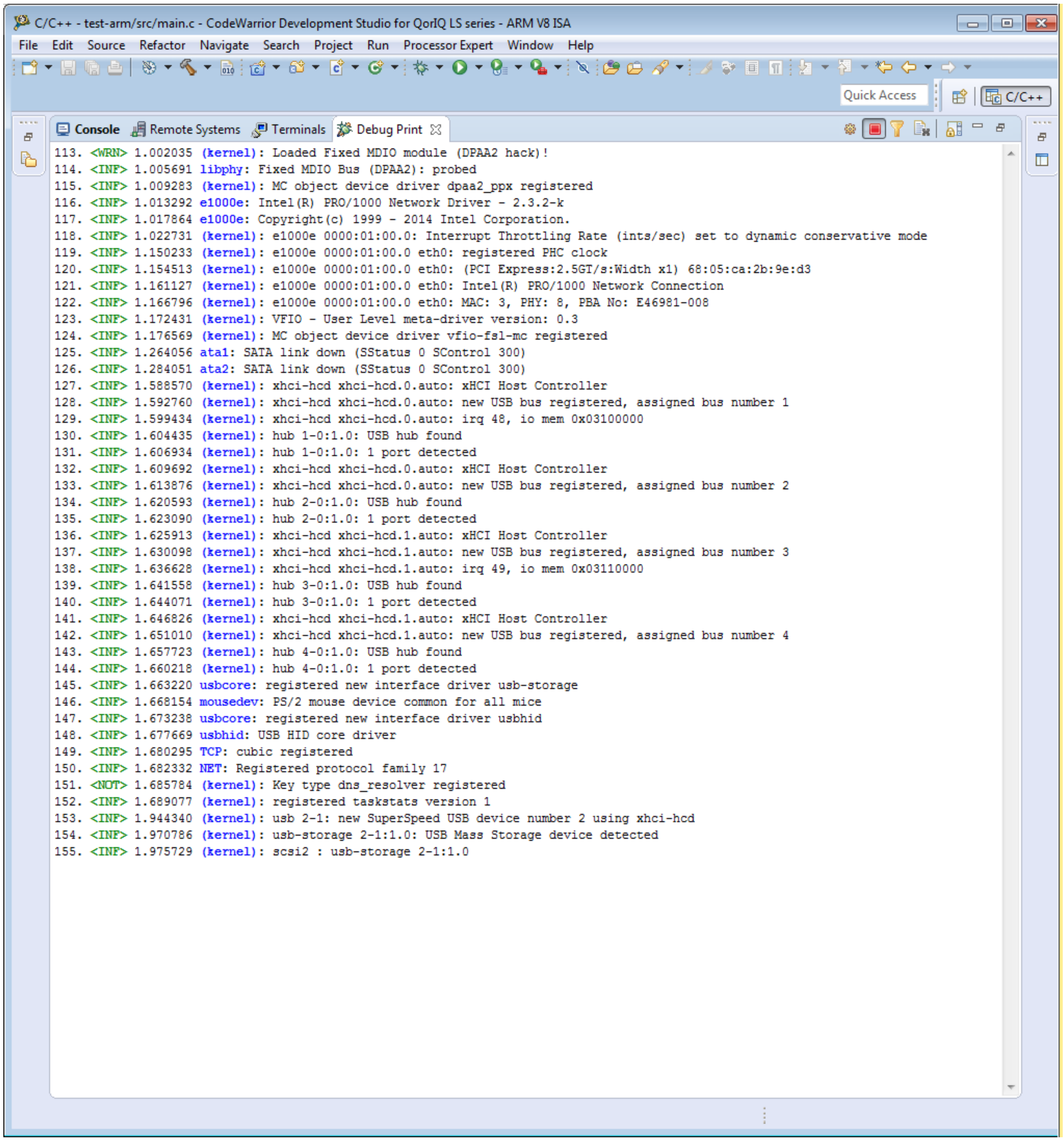
Lower Limit: 1.000

Upper Limit: 2.000

Current Filters: ne: [1.000000, 2.000000] module: (any)

Figure 58. Create Debug Print Filters dialog - Timestamp tab

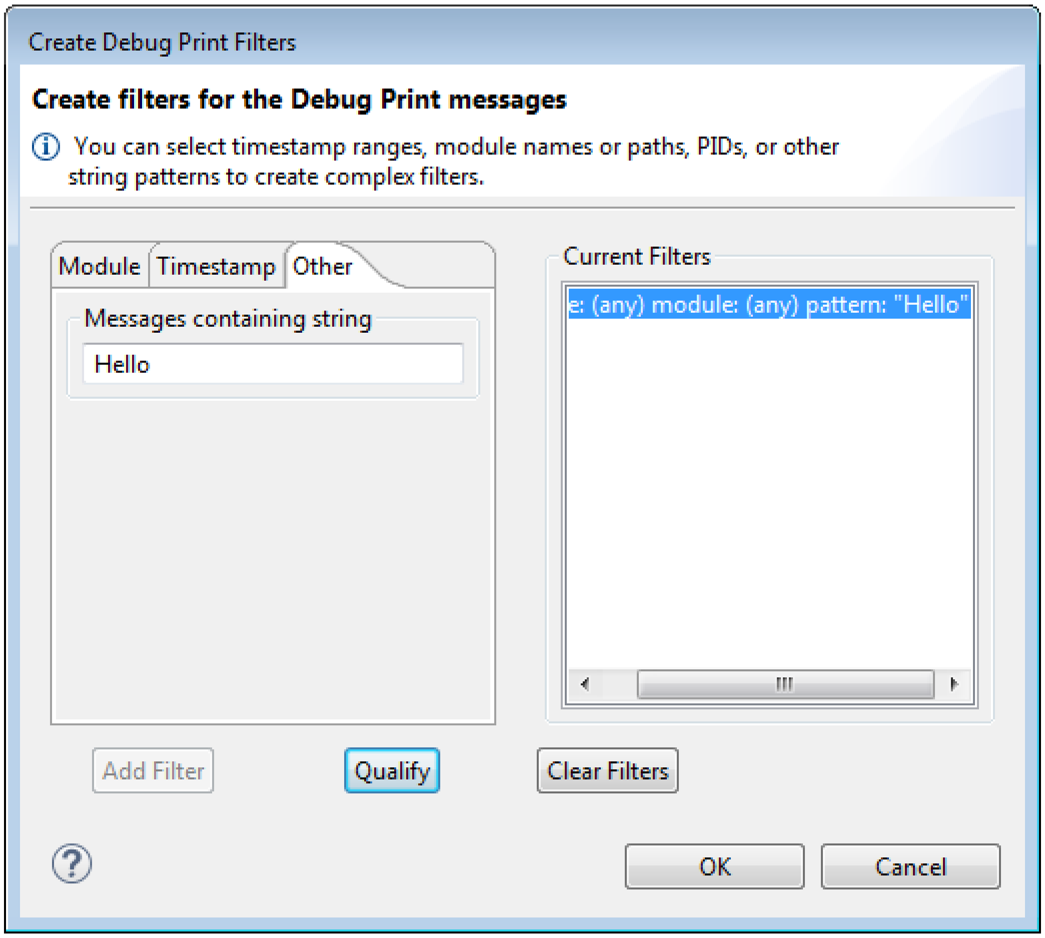
The following figure shows the messages displayed in the Debug Print view using timestamp filter:


C/C++ - test-arm/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Console
Remote Systems
Terminals
Debug Print

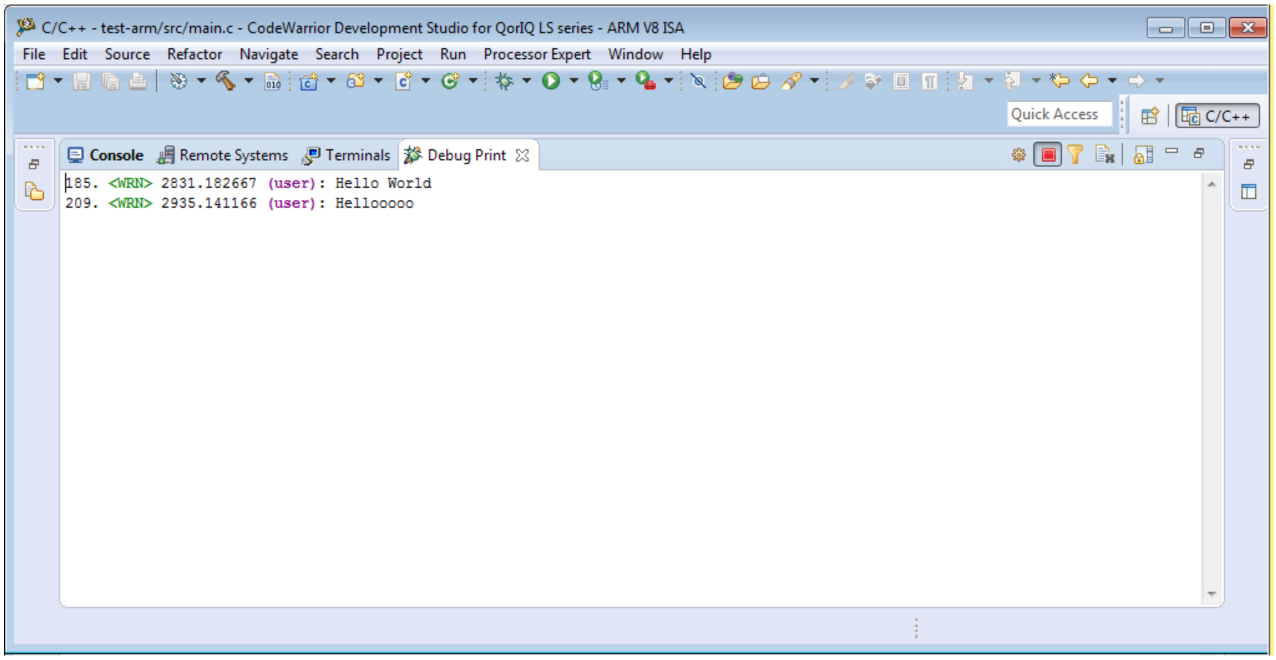
```
113. <WRN> 1.002035 (kernel): Loaded Fixed MDIO module (DPAA2 hack)!
114. <INF> 1.005691 libphy: Fixed MDIO Bus (DPAA2): probed
115. <INF> 1.009283 (kernel): MC object device driver dpaa2_ppx registered
116. <INF> 1.013292 e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
117. <INF> 1.017864 e1000e: Copyright(c) 1999 - 2014 Intel Corporation.
118. <INF> 1.022731 (kernel): e1000e 0000:01:00.0: Interrupt Throttling Rate (ints/sec) set to dynamic conservative mode
119. <INF> 1.150233 (kernel): e1000e 0000:01:00.0 eth0: registered PHC clock
120. <INF> 1.154513 (kernel): e1000e 0000:01:00.0 eth0: (PCI Express:2.5GT/s:Width x1) 68:05:ca:2b:9e:d3
121. <INF> 1.161127 (kernel): e1000e 0000:01:00.0 eth0: Intel(R) PRO/1000 Network Connection
122. <INF> 1.166796 (kernel): e1000e 0000:01:00.0 eth0: MAC: 3, PHY: 8, PBA No: E46981-008
123. <INF> 1.172431 (kernel): VFIO - User Level meta-driver version: 0.3
124. <INF> 1.176569 (kernel): MC object device driver vfio-fsl-mc registered
125. <INF> 1.264056 ata1: SATA link down (SStatus 0 SControl 300)
126. <INF> 1.284051 ata2: SATA link down (SStatus 0 SControl 300)
127. <INF> 1.588570 (kernel): xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
128. <INF> 1.592760 (kernel): xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned bus number 1
129. <INF> 1.599434 (kernel): xhci-hcd xhci-hcd.0.auto: irq 48, io mem 0x03100000
130. <INF> 1.604435 (kernel): hub 1-0:1.0: USB hub found
131. <INF> 1.606934 (kernel): hub 1-0:1.0: 1 port detected
132. <INF> 1.609692 (kernel): xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
133. <INF> 1.613876 (kernel): xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned bus number 2
134. <INF> 1.620593 (kernel): hub 2-0:1.0: USB hub found
135. <INF> 1.623090 (kernel): hub 2-0:1.0: 1 port detected
136. <INF> 1.625913 (kernel): xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
137. <INF> 1.630098 (kernel): xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned bus number 3
138. <INF> 1.636628 (kernel): xhci-hcd xhci-hcd.1.auto: irq 49, io mem 0x03110000
139. <INF> 1.641558 (kernel): hub 3-0:1.0: USB hub found
140. <INF> 1.644071 (kernel): hub 3-0:1.0: 1 port detected
141. <INF> 1.646826 (kernel): xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
142. <INF> 1.651010 (kernel): xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned bus number 4
143. <INF> 1.657723 (kernel): hub 4-0:1.0: USB hub found
144. <INF> 1.660218 (kernel): hub 4-0:1.0: 1 port detected
145. <INF> 1.663220 usbcore: registered new interface driver usb-storage
146. <INF> 1.668154 mousedev: PS/2 mouse device common for all mice
147. <INF> 1.673238 usbcore: registered new interface driver usbhid
148. <INF> 1.677669 usbhid: USB HID core driver
149. <INF> 1.680295 TCP: cubic registered
150. <INF> 1.682332 NET: Registered protocol family 17
151. <NOT> 1.685784 (kernel): Key type dns_resolver registered
152. <INF> 1.689077 (kernel): registered taskstats version 1
153. <INF> 1.944340 (kernel): usb 2-1: new SuperSpeed USB device number 2 using xhci-hcd
154. <INF> 1.970786 (kernel): usb-storage 2-1:1.0: USB Mass Storage device detected
155. <INF> 1.975729 (kernel): scsi2 : usb-storage 2-1:1.0
```

Figure 59. Debug Print view displays messages with Timestamp filter

- **Other** tab: allows adding other type of qualifications to existing filters, or a global qualification if no other filter is created. Currently, the only qualification in this tab is a string pattern which is searched in all the messages (except for timestamps and module names/paths/PIDS). After you input the string pattern, you must click **Qualify** in order to add this qualification to all the existing filters. If no filter exists, a global qualification is performed.


Create Debug Print Filters
Create filters for the Debug Print messages
You can select timestamp ranges, module names or paths, PIDs, or other string patterns to create complex filters.
Module
Timestamp
Other
Messages containing string
Hello
Current Filters
e: (any) module: (any) pattern: "Hello"
Add Filter
Qualify
Clear Filters
OK
Cancel

The following figure shows the messages displayed in the Debug Print view using message strings in the other filter:


C/C++ - test-arm/src/main.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Processor Expert Window Help
Quick Access
C/C++
Console
Remote Systems
Terminals
Debug Print

```
185. <WRN> 2831.182667 (user): Hello World
209. <WRN> 2935.141166 (user): Hellooooo
```

Figure 61. Debug Print view displays messages with Other filter

Figure 60. Create Debug Print Filters dialog - Other tab

Index

CWARMv8TAUG
Rev. 11.3.0
12/2017