Freescale Semiconductor, Inc.
Reference Manual

Document Number: CW_ARMv7_Tracing_User_Guide
10.0.8, 01/2016

CodeWarrior for ARMv7 Tracing and Analysis User Guide

Contents

Chapter 1 Introduction

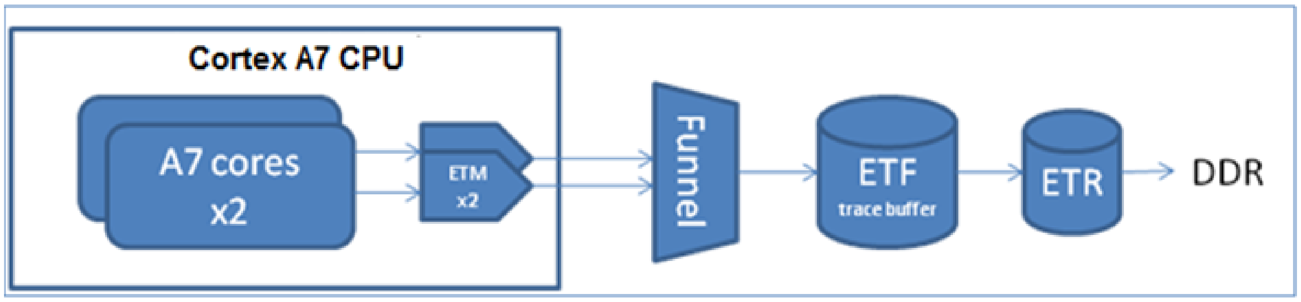
The CodeWarrior for ARMv7® V10.x Tracing and Performance Analysis tool allows you to analyze and collect data of an application.

You can use this collected data to identify the bottlenecks, such as slow execution of routines or heavily-used routines within the application. This visibility can help you understand how your application runs, as well as identify operational problems.

The UI Platform Configurator tool reads the user settings from the input XML file and transforms them into target access memory writes to the configuration registers.

The figure below shows the core trace path for LS1021A within the ARMv7 architecture.

Figure 1: Core trace path for LS1021A



The UI Platform Configurator configures the ARMv7 cores Embedded Trace Macrocell (ETM), Funnel, Embedded Trace FIFO (ETF), and Embedded Trace Router (ETR) modules. The ETF stores and forwards trace data using a dedicated RAM buffer. This reduces trace loss by absorbing spikes in trace data. The ETR redirects trace to the system bus for collection from alternative channels. For ARMv7, it is DDR memory.

The destination of the raw trace is either the internal trace buffer of the ETF module or a user defined buffer in DDR.

This manual describes how the CodeWarrior for ARMv7 V10.x Tracing and Performance Analysis tool can be used for the LS1020A, LS1021A, LS1022A, and LS1024A devices. This chapter presents an overview of this manual and introduces you to the Tracing and Performance Analysis tools.

This chapter contains the following sections:

- Overview on page 5
- Accompanying documentation on page 6

1.1 Overview

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

The table below describes each chapter in the manual.

Table 1: Manual Contents

Chapter	Description
Introduction	Introduces Tracing and Performance Analysis tool (this chapter)
Tracing on page 7	Describes the trace collection process and how to use the trace data for identifying the bottlenecks
Collecting and Viewing Linux Trace on page 31	Describes how you to collect and view satrace with and without using CodeWarrior
Linux Kernel Debug Print Tool on page 45	Describes 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 V7 ISA.

You can access the Documentation page by:

- Opening `START_HERE.html` from the `<CWInstallDir>\CW_ARMv7\ARMv7\Help` folder
- Choosing **Help > Documentation** from the CodeWarrior IDE menu bar

Chapter 2 Tracing

Tracing is a technique that obtains diagnostic and performance information about a program's execution.

This information is typically used for debugging purposes, and additionally, depending on the type and detail information contained in a trace log, to diagnose common problems with the software. The trace log includes information about the trace source, type of event, description of the event and time stamp value.

This chapter contains the following sections:

- Configuring and collecting trace on page 7
- Viewing trace data using Analysis Results view on page 16

2.1 Configuring and collecting trace

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

This section contains the following subsections:

- Creating a new project on page 7
- Configuring trace on page 8
- Collecting trace data on page 13
- Collecting trace data using an Attach configuration on page 14

2.1.1 Creating a new project

You need to create a new CodeWarrior bareboard project or open an existing project before you start collecting the trace data.

To create a new bareboard project:

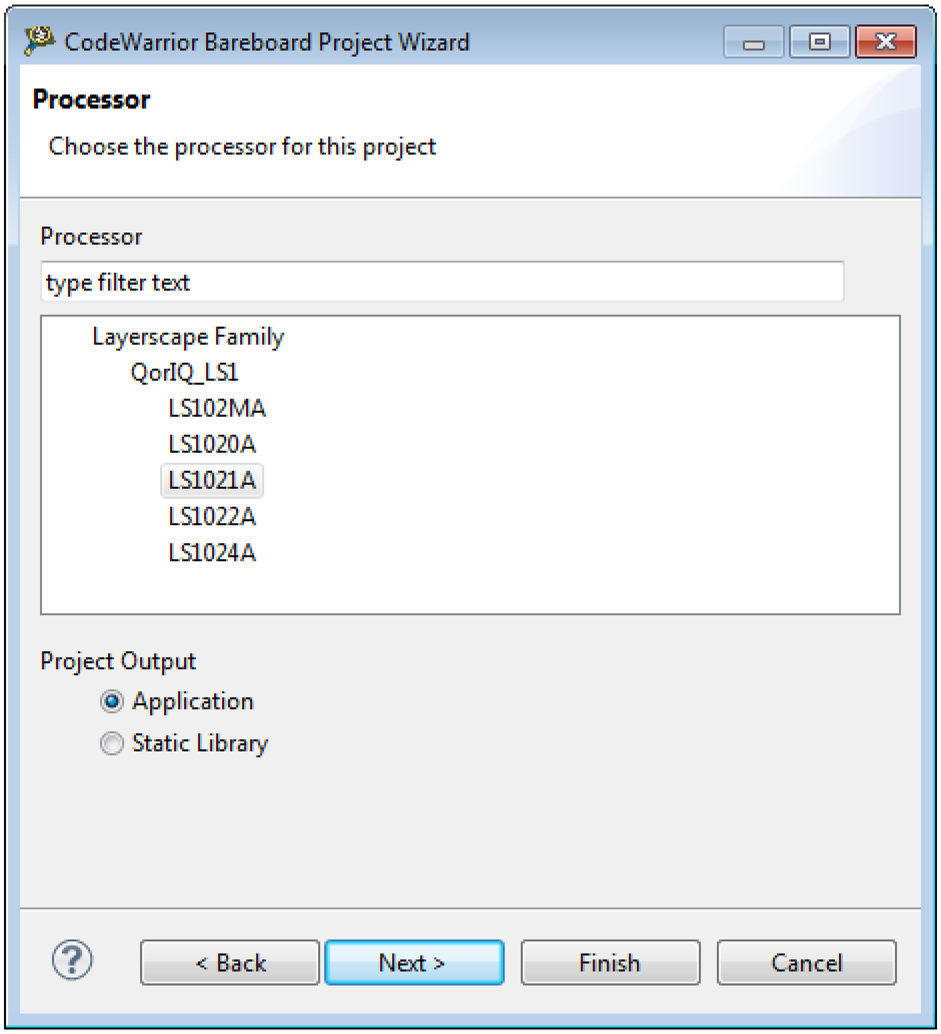
1. Start the CodeWarrior IDE.
2. From the IDE menu bar, choose **File > New > CodeWarrior Bareboard Project Wizard.**

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

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

   The **Processor** page appears.

Figure 2: Processor page



7. Select the required processor and click **Next**.
8. The **Debug Target Settings** page appears. Choose a board from the **Board** menu and click **Next**.
9. Do no change the default settings of the rest of the pages and click **Next** and **Finish**.
10. From the IDE menu bar, choose **Project** > **Build Project**.

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

NOTE

For more information on creating a CodeWarrior project, see *CodeWarrior for ARMv7 Targeting Manual* (CW_ARMv7_Targeting_Manual.pdf) in the `<CWInstallDir>\CW_ARMv7\ARMv7\Help\PDF` folder, where *<CWInstallDir>* is the installation directory for the CW4NET installer.

2.1.2 Configuring trace

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

To define a trace configuration:

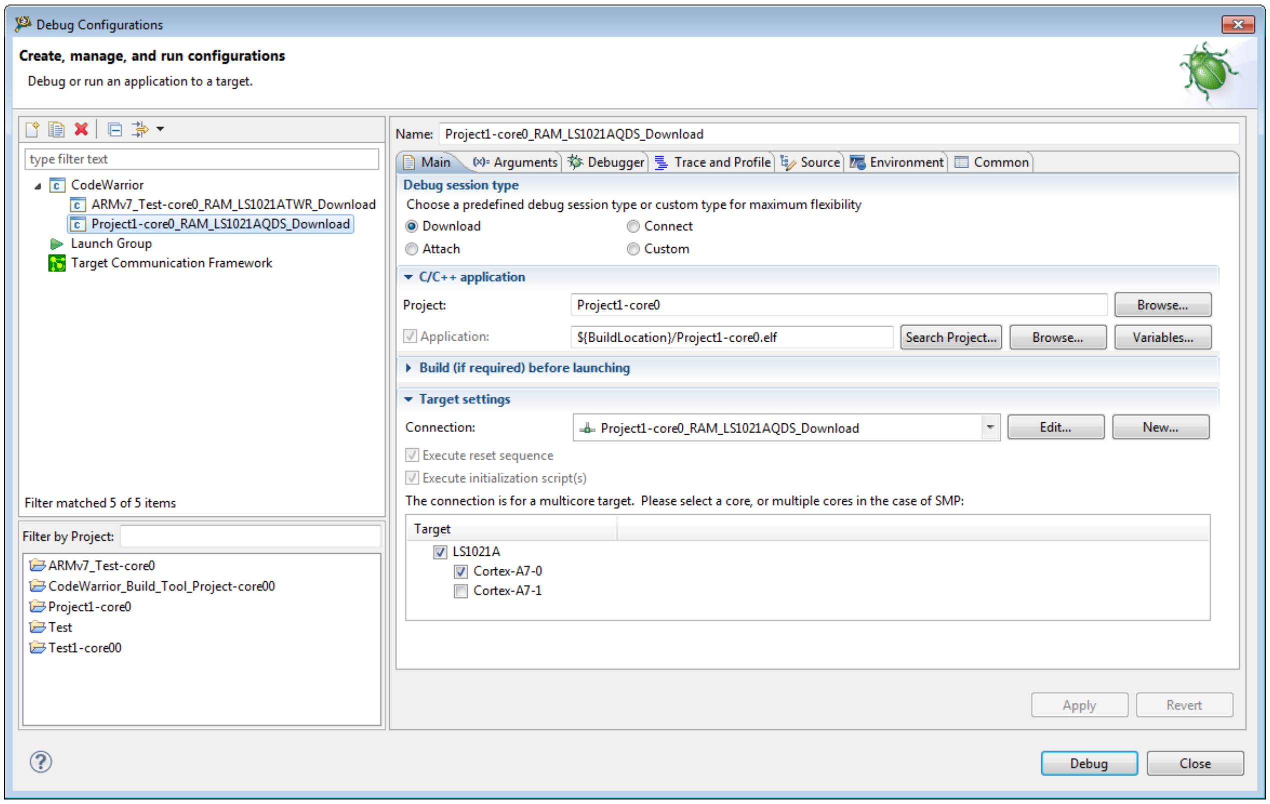
1. Choose **Run** > **Debug Configurations** from the CodeWarrior IDE menu bar.

The **Debug Configurations** dialog appears.

2. In the left pane of this dialog, expand the CodeWarrior tree control.
3. Select the launch configuration corresponding to your project. For example, **Project1-core0_RAM_LS1021AQDS_Download**.

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

Figure 3: Debug Configurations dialog



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

- CodeWarrior
  - ARMv7_Test-core0_RAM_LS1021ATWR_Download
  - Project1-core0_RAM_LS1021AQDS_Download
- Launch Group
- Target Communication Framework

Filter matched 5 of 5 items

Filter by Project:

- ARMv7_Test-core0
- CodeWarrior_Build_Tool_Project-core00
- Project1-core0
- Test
- Test1-core00

Name: Project1-core0_RAM_LS1021AQDS_Download

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

Debug session type

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

[● Download] ○ Connect
○ Attach ○ Custom

C/C++ application

Project: Project1-core0

Application: ${BuildLocation}/Project1-core0.elf

Build (if required) before launching

Target settings

Connection: Project1-core0_RAM_LS1021AQDS_Download

[✓] Execute reset sequence
[✓] Execute initialization script(s)

The connection is for a multicore target. Please select a core, or multiple cores in the case of SMP:

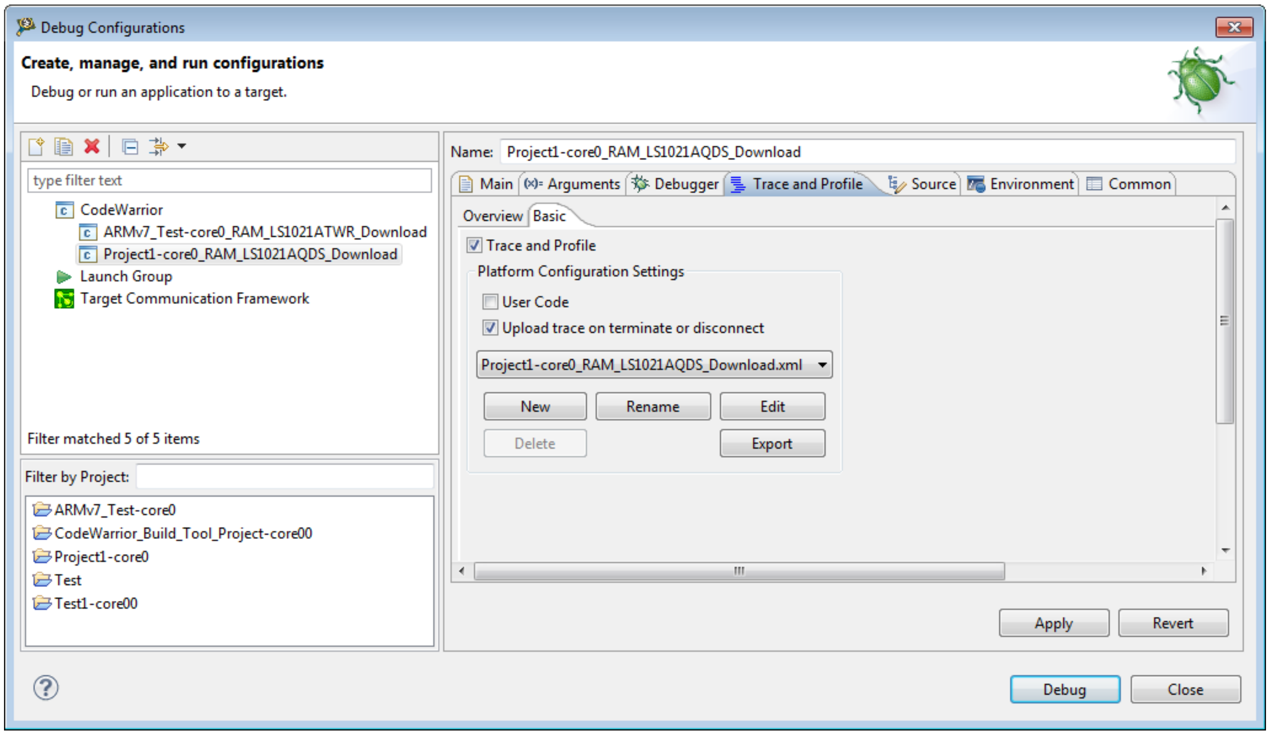
Target
- [✓] LS1021A
  - [✓] Cortex-A7-0
  - [ ] Cortex-A7-1

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

   The page appears with two tabs:

   - The **Overview** tab displays the information about the ARMv7 architecture.
   - The **Basic** tab displays the predefined settings. For **Platform Configuration Settings**, you can apply different values based on what you want to achieve. The values are called profiles, and the framework allows creation of different profiles for a configuration block. The profiles dialog is used to create new, rename, delete, or edit the settings of a profile for a configuration provider.

Figure 4: Trace and Profile tab


Debug Configurations
Create, manage, and run configurations
Debug or run an application to a target.
type filter text
CodeWarrior
ARMv7_Test-core0_RAM_LS1021ATWR_Download
Project1-core0_RAM_LS1021AQDS_Download
Launch Group
Target Communication Framework
Filter matched 5 of 5 items
Filter by Project:
ARMv7_Test-core0
CodeWarrior_Build_Tool_Project-core00
Project1-core0
Test
Test1-core00
Main
Arguments
Debugger
Trace and Profile
Source
Environment
Common
Overview
Basic
Platform Configuration Settings
New
Rename
Edit
Delete
Export
Apply
Revert
Debug
Close

Name: Project1-core0_RAM_LS1021AQDS_Download

[✓] Trace and Profile
[ ] User Code
[✓] Upload trace on terminate or disconnect
Project1-core0_RAM_LS1021AQDS_Download.xml

5. The **Trace and Profile** checkbox is selected for the trace session to start immediately on debug launch. If **User Code** checkbox is selected, it lets you upload trace into CodeWarrior when trace session is started from user code or tools other than CodeWarrior. That is, the application where you make your own configuration of trace registers in the trace hardware explicitly and CodeWarrior is not involved. If **Upload trace on terminate or disconnect** checkbox is selected, the trace data is automatically collected when debug session is terminated. In the combo box, you can see a default platform configuration file having same name as you have for launch configuration.

6. You can create, rename, edit, delete, or export the platform configuration file. The **Export** button allows you to export the platform configuration file to other platform for later use.

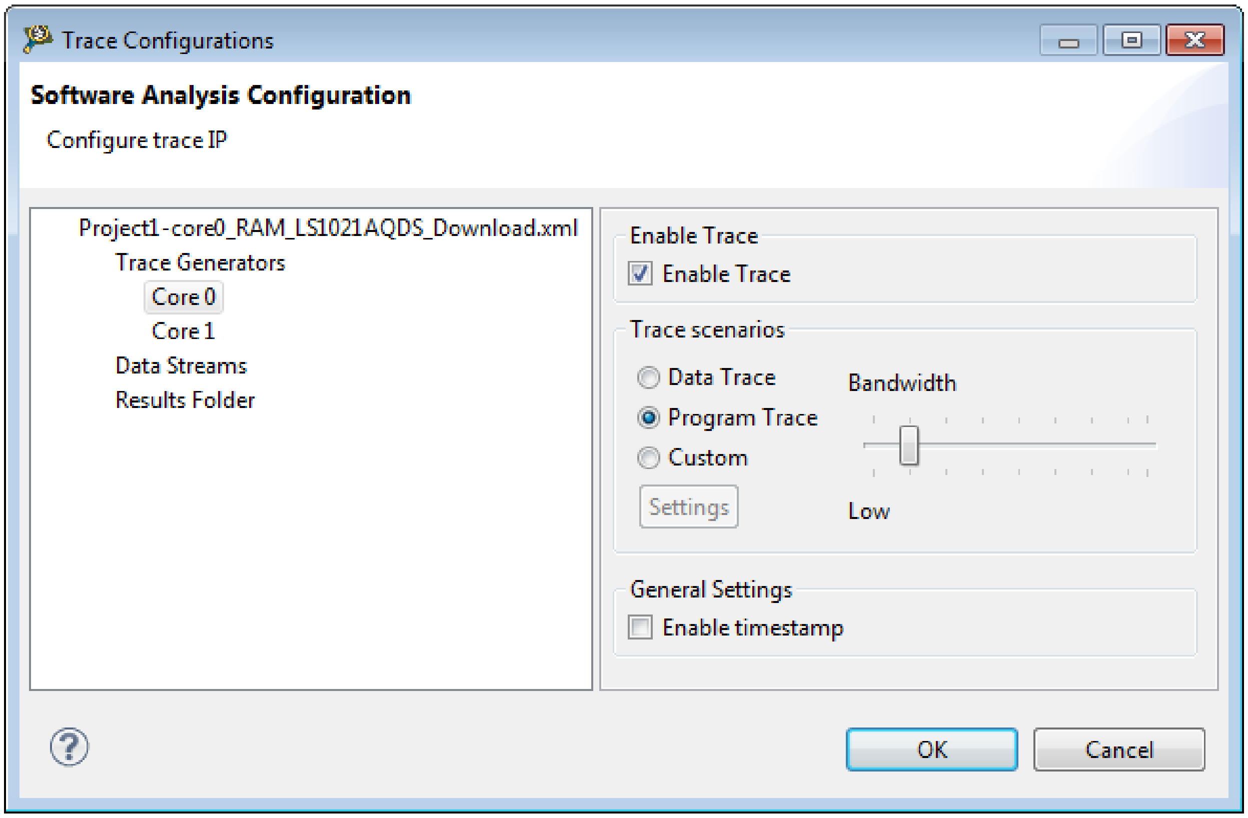
NOTE

You can create your own configuration file by using a template placed at: `<CWInstallDir>\CW_ARMv7\ARMv7\sa_ls\data\fsl.configs.sa.ls.configurators/`

7. If you want to create or manage new profiles, you can invoke the **Trace Configurations** dialog by clicking on the **Edit** button. You can see a tree viewer on the left side that lists all the profiles. The content in right pane is driven by the selection in the left pane.

   • If you select the required core under the **Trace Generators** on the left pane, the entries corresponding to that configuration are displayed on the right pane.

Figure 5: Trace Configurations



[✓] Enable Trace

Data Trace [● Program Trace] Custom

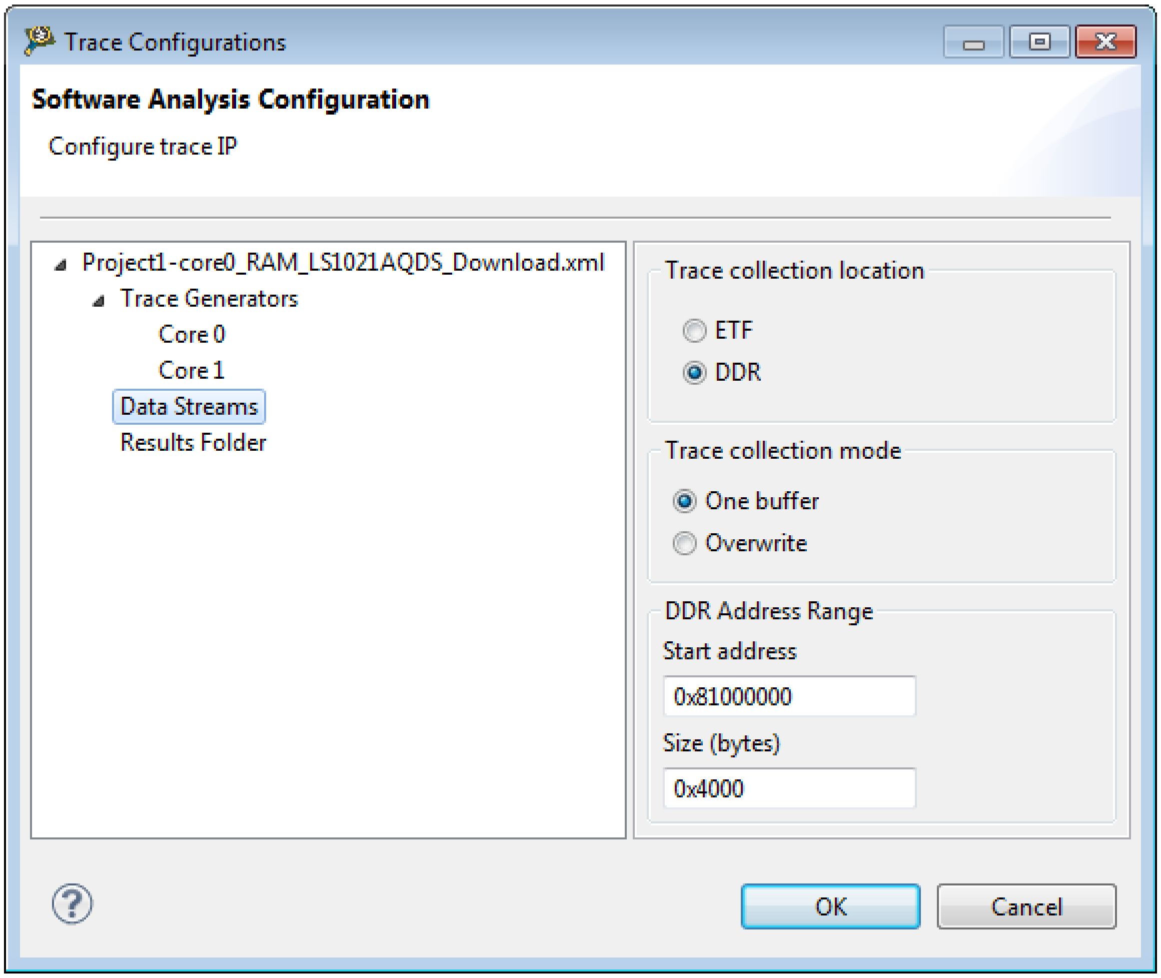
[ ] Enable timestamp

The following settings can be configured for each core or trace generator:

- **Trace Scenarios**: Choose a trace scenario type from the **Trace Scenarios** panel.
- **General Settings**: 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).

- If you select **Data Streams** to collect trace data, the entries corresponding to the selected buffer are displayed on the right pane.

Figure 6: Data Streams



Trace Configurations

Software Analysis Configuration

Configure trace IP

- Project1-core0_RAM_LS1021AQDS_Download.xml
  - Trace Generators
    - Core 0
    - Core 1
  - Data Streams
  - Results Folder

Trace collection location

( ) ETF
(●) DDR

Trace collection mode

(●) One buffer
( ) Overwrite

DDR Address Range

Start address: 0x81000000

Size (bytes): 0x4000

The following settings can be configured for each data stream:

- **Trace collection location**: Choose **ETF** or **DDR** as the trace collection location.
- **Trace collection mode**: Choose **One buffer** or **Overwrite** as the trace collection mode. If the **One buffer** option is choosen, then trace can only be collected until the buffer is filled. In this case, only the first buffer size part of the trace is kept. If the **Overwrite** option is choosen and the buffer is filled, then the pointer returns to the beginning of the buffer and overwrites the older trace. In this case, only the last buffer size part of the trace is kept.

In case the trace is collected in the DDR buffer, then you also need to specify the following options to set the address range for the DDR trace buffer:

- **Start address**: Indicates the start address of the trace buffer in the DDR memory.
- **Size**: Indicates the size of the DDR trace buffer in bytes.

NOTE

Each time you collect new trace data for a project, the existing trace data will be overwritten.

2.1.3 Collecting trace data

After creating a project and defining launch configuration and trace configuration, you can start a debug session and start collecting trace data.

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

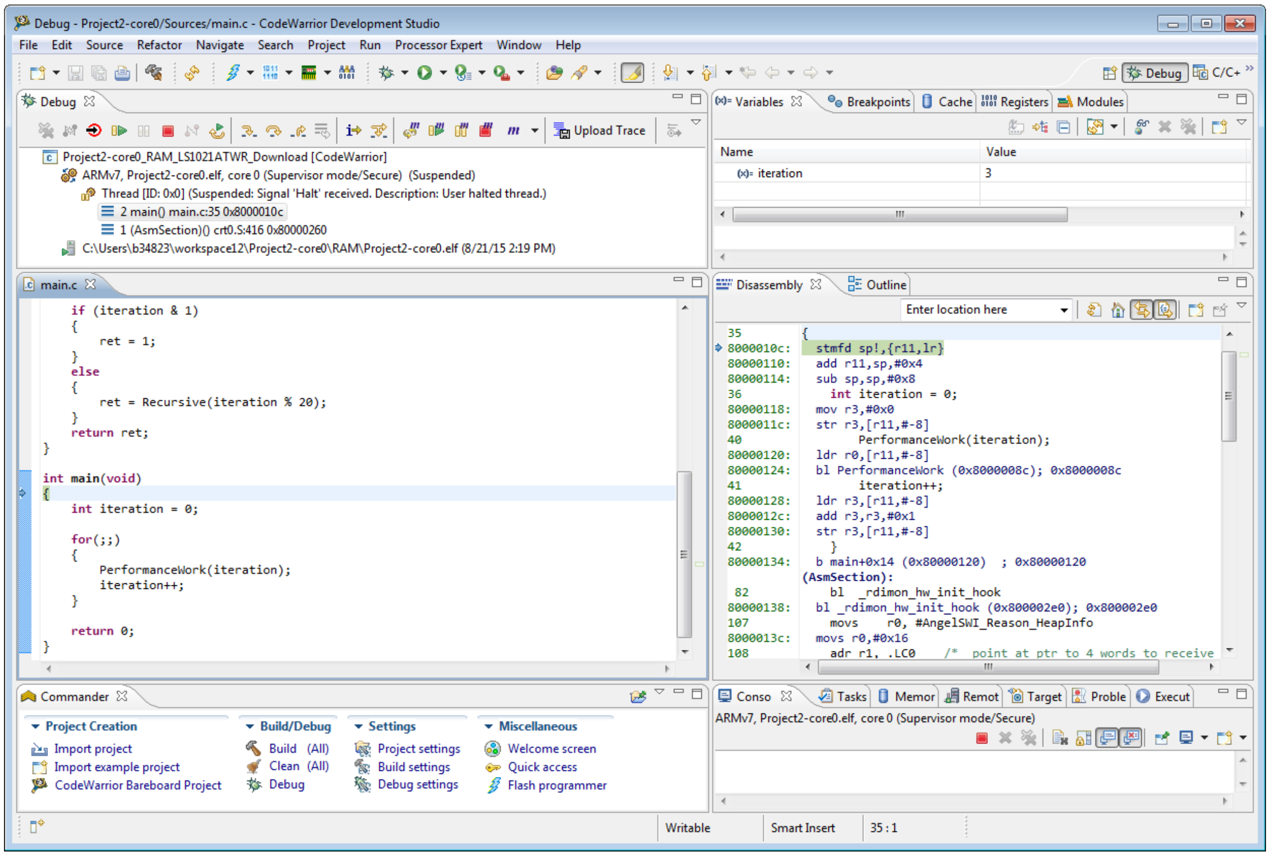
To collect trace data:

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

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

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

   Figure 7: Debug perspective



```
Project2-core0_RAM_LS1021ATWR_Download [CodeWarrior]
  ARMv7, Project2-core0.elf, core 0 (Supervisor mode/Secure)  (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() main.c:35 0x8000010c
      1 (AsmSection)() crt0.S:416 0x80000260
  C:\Users\b34823\workspace12\Project2-core0\RAM\Project2-core0.elf (8/21/15 2:19 PM)
```

Name	Value
iteration	3

```
    if (iteration & 1)
    {
        ret = 1;
    }
    else
    {
        ret = Recursive(iteration % 20);
    }
    return ret;
}

int main(void)
{
    int iteration = 0;

    for(;;)
    {
        PerformanceWork(iteration);
        iteration++;
    }

    return 0;
}
```

```
35         {
8000010c:    stmfd sp!,{r11,lr}
80000110:    add r11,sp,#0x4
80000114:    sub sp,sp,#0x8
36             int iteration = 0;
80000118:    mov r3,#0x0
8000011c:    str r3,[r11,#-8]
40                 PerformanceWork(iteration);
80000120:    ldr r0,[r11,#-8]
80000124:    bl PerformanceWork (0x8000008c); 0x8000008c
41                 iteration++;
80000128:    ldr r3,[r11,#-8]
8000012c:    add r3,r3,#0x1
80000130:    str r3,[r11,#-8]
42             }
80000134:    b main+0x14 (0x80000120)  ; 0x80000120
           (AsmSection):
 82            bl  _rdimon_hw_init_hook
80000138:    bl _rdimon_hw_init_hook (0x800002e0); 0x800002e0
107            movs    r0, #AngelSWI_Reason_HeapInfo
8000013c:    movs r0,#0x16
108            adr r1, .LC0    /*  point at ptr to 4 words to receive
```

ARMv7, Project2-core0.elf, core 0 (Supervisor mode/Secure)

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

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

3. Let the application execute for several seconds and click **Suspend** . if you want to stop the collection temporarily. Again, click **Resume** to start the trace collection and when it finishes, click **Terminate**.

2.1.4 Collecting trace data using an Attach configuration

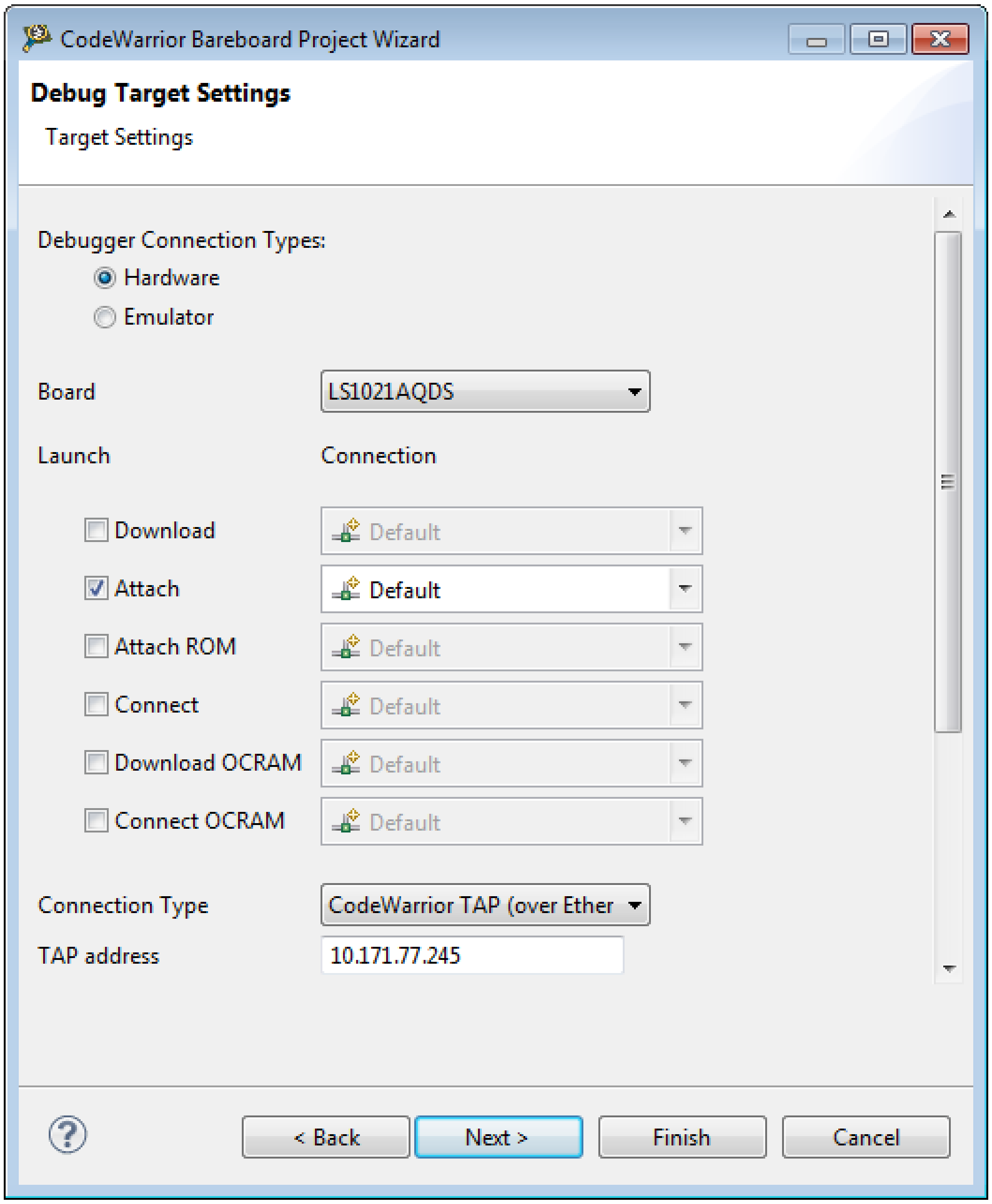
This section explains how to collect trace using an Attach launch configuration.

The Attach launch configuration is useful when you want to connect to a running target without resetting the target or downloading a different application on it. For more information on Attach launch configuration, see *CodeWarrior for ARMv7 Targeting Manual* (CW_ARMv7_Targeting_Manual.pdf) in the `<CWInstallDir>\CW_ARMv7\ARMv7\Help\PDF` folder.

To configure your application for trace collection using the Attach launch configuration:

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

Figure 8: Creating Attach launch configuration in Debug Target Settings page



CodeWarrior Bareboard Project Wizard

Debug Target Settings

Target Settings

Debugger Connection Types:
- [✓ Hardware]
- Emulator

Board: LS1021AQDS

Launch	Connection
☐ Download	Default
☑ Attach	Default
☐ Attach ROM	Default
☐ Connect	Default
☐ Download OCRAM	Default
☐ Connect OCRAM	Default

Connection Type: CodeWarrior TAP (over Ether

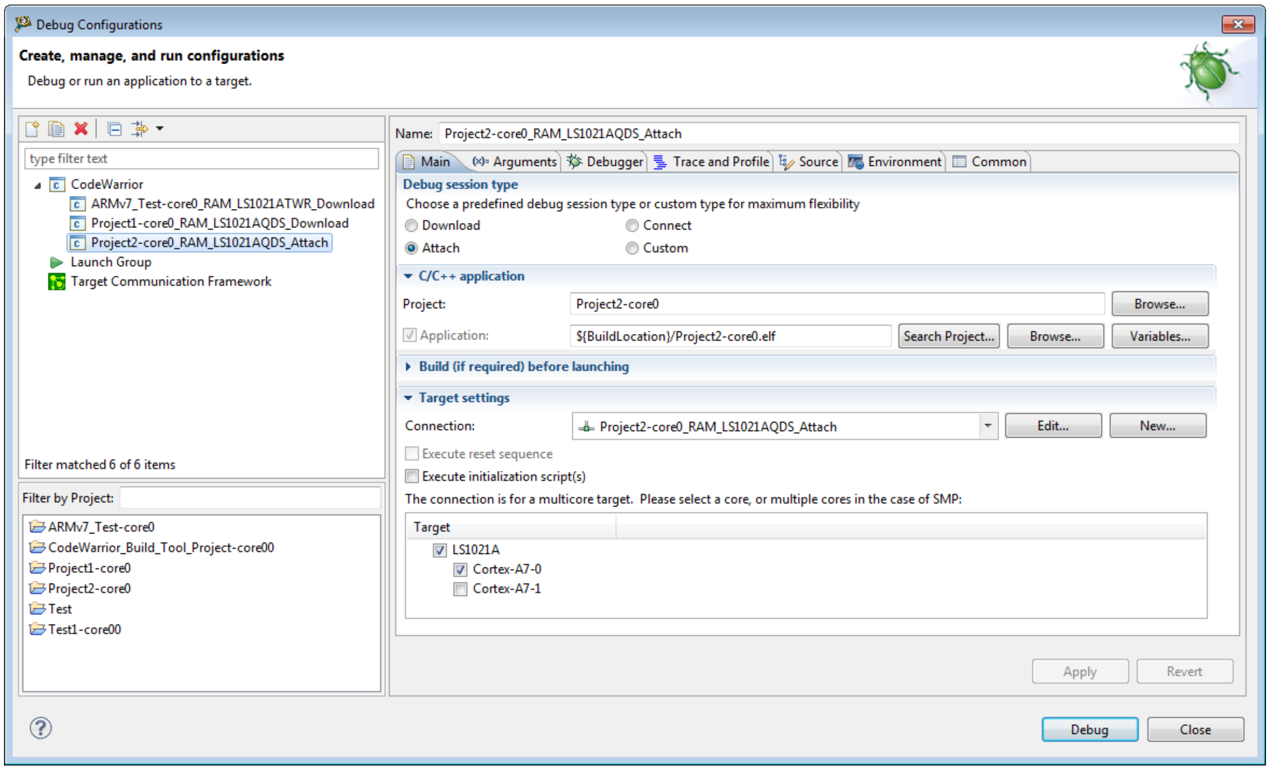
TAP address: 10.171.77.245

e. Select the required **Connection Type** and click **Next**.

f. Follow the remaining steps of the wizard and click **Finish**.

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

Figure 9: Selecting Attach launch configuration



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

- CodeWarrior
  - ARMv7_Test-core0_RAM_LS1021ATWR_Download
  - Project1-core0_RAM_LS1021AQDS_Download
  - Project2-core0_RAM_LS1021AQDS_Attach
- Launch Group
- Target Communication Framework

Filter matched 6 of 6 items

Filter by Project:

- ARMv7_Test-core0
- CodeWarrior_Build_Tool_Project-core00
- Project1-core0
- Project2-core0
- Test
- Test1-core00

Name: Project2-core0_RAM_LS1021AQDS_Attach

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

Debug session type

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

Download | Connect
[✓ Attach] | Custom

C/C++ application

Project: Project2-core0

Application: ${BuildLocation}/Project2-core0.elf

Build (if required) before launching

Target settings

Connection: Project2-core0_RAM_LS1021AQDS_Attach

Execute reset sequence

Execute initialization script(s)

The connection is for a multicore target. Please select a core, or multiple cores in the case of SMP:

Target
- [✓] LS1021A
  - [✓] Cortex-A7-0
  - [ ] Cortex-A7-1

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

5. In the **Main** tab of the dialog, make sure that **Attach** is selected in the **Debug Session Type** group.
6. Open the **Trace and Profile** tab.
7. Select the **Trace and Profile** checkbox on the **Basic** tab to enable trace and profile options. The **Upload trace on terminate or disconnect** checkbox is selected, by default.
8. Click **Debug**.
9. Click **Resume** to start collecting the trace data.
10. Click **Suspend** after some time to stop trace collection.
11. Click the **Upload Trace** button to get the collected trace.
12. Open the **Trace** viewer to view the collected data.

2.2 Viewing trace data using Analysis Results view

The **Analysis Results** view provides access to various trace data results collected on an application.

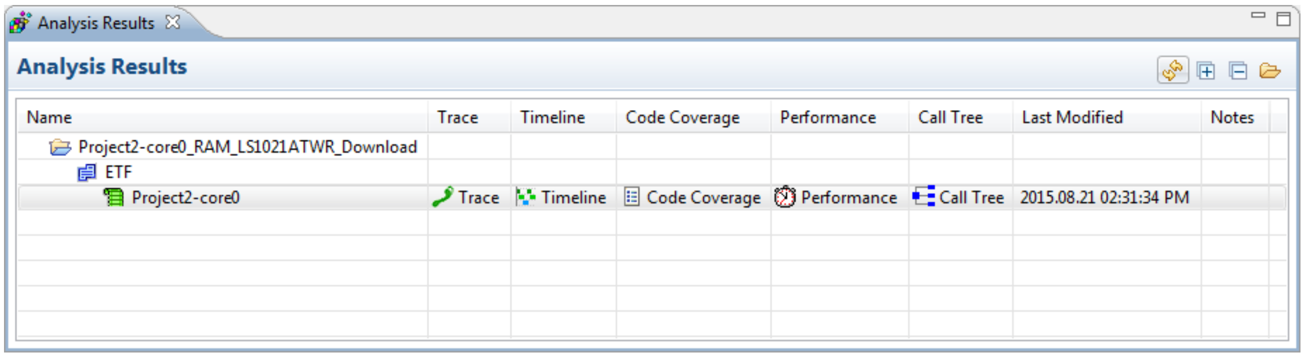
The **Analysis Results** view shows and manages collected trace and profiling data. It provides link to open any supported viewer: **Trace**, **Timeline**, **Code Coverage**, **Performance**, and **Call Tree**. This view provides functions like save results, rename, and delete to help you to organize collected results.

To open the **Analysis Results** view:

1. Choose **Window** > **Show View** > **Other** from the CodeWarrior IDE menu bar. The **Show View** dialog appears.
2. Select **Software Analysis** > **Analysis Results** and click **OK**. The **Analysis Results** view appears.

NOTE
The **Analysis Results** view pops up and gets focus after successful trace collection. It does not refresh automatically.

Figure 10: Analysis Results view



Name	Trace	Timeline	Code Coverage	Performance	Call Tree	Last Modified	Notes
Project2-core0_RAM_LS1021ATWR_Download							
ETF							
Project2-core0	Trace	Timeline	Code Coverage	Performance	Call Tree	2015.08.21 02:31:34 PM	

The **Analysis Results** view toolbar provides the following options:

Table 2: Analysis Results view - toolbar options

Option	Description
🔄	Refreshes the displayed data.
⊞	Expands all the nodes.
⊟	Collapses all the nodes.
📂	Select custom results folder.

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

Table 3: Analysis Results view - shortcut menu commands

Option	Description
Refresh	Refreshes the displayed data.
Expand All	Expands all the nodes.
Collapse All	Collapses all the nodes.
Results	Lets you select a custom trace results folder.
Copy Cell	Copies the currently selected cell. The name of the data source is copied to the clipboard.

Table continues on the next page...

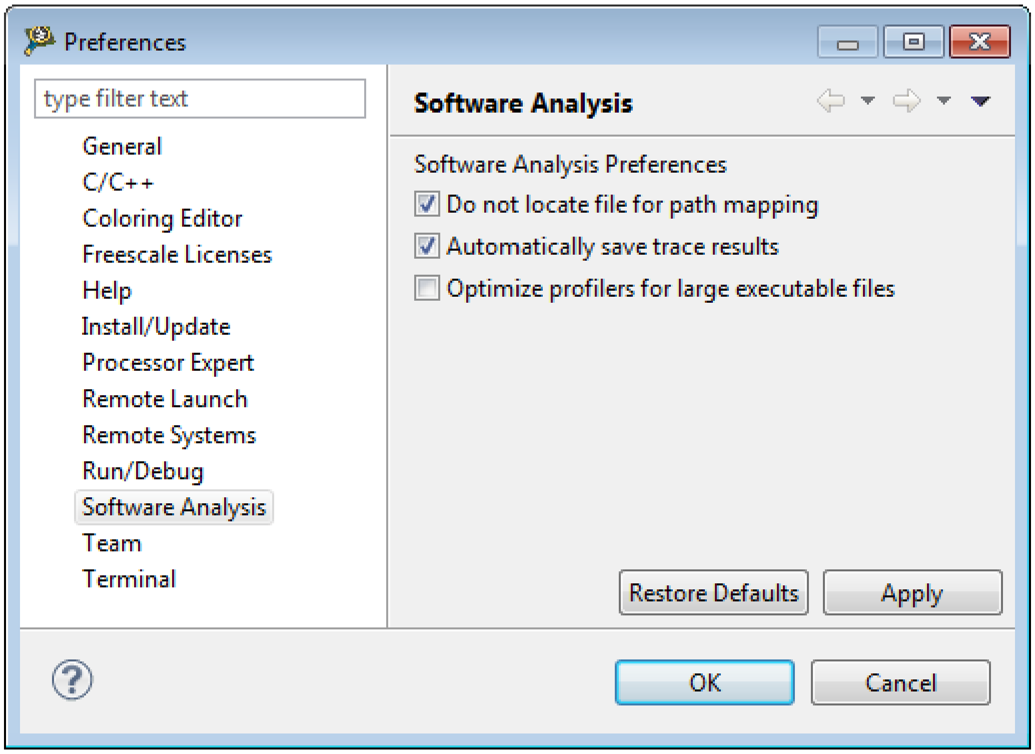
Table 3: Analysis Results view - shortcut menu commands (continued)

Option	Description
Copy Line	Copies the complete line of the data source.
Save Results	Saves the trace results.
Delete Results	Deletes all the saved results.

The trace file can automatically be saved after trace collection completes or is stopped. To enable automatic saving of trace file, follow these steps:

1. Choose **Window > Preferences**. The **Preferences** dialog appears.
2. Click **Software Analysis** in the left pane of the **Preferences** dialog, and select the **Automatically save trace results** checkbox in the right pane, as shown in the figure below.

Figure 11: Preferences dialog



Software Analysis

Software Analysis Preferences

- [✓] Do not locate file for path mapping
- [✓] Automatically save trace results
- [ ] Optimize profilers for large executable files

3. Click **OK** to apply the setting and close the **Preferences** dialog.

Now, when you will collect trace, the trace file will be saved in the **.AnalysisData** folder of the current workspace.

The **Analysis Results** view provides hyperlinks to open:

- Trace viewer on page 19
- Timeline viewer on page 20
- Code Coverage viewer on page 22
- Performance viewer on page 27
- Call Tree viewer on page 29

2.2.1 Trace viewer

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

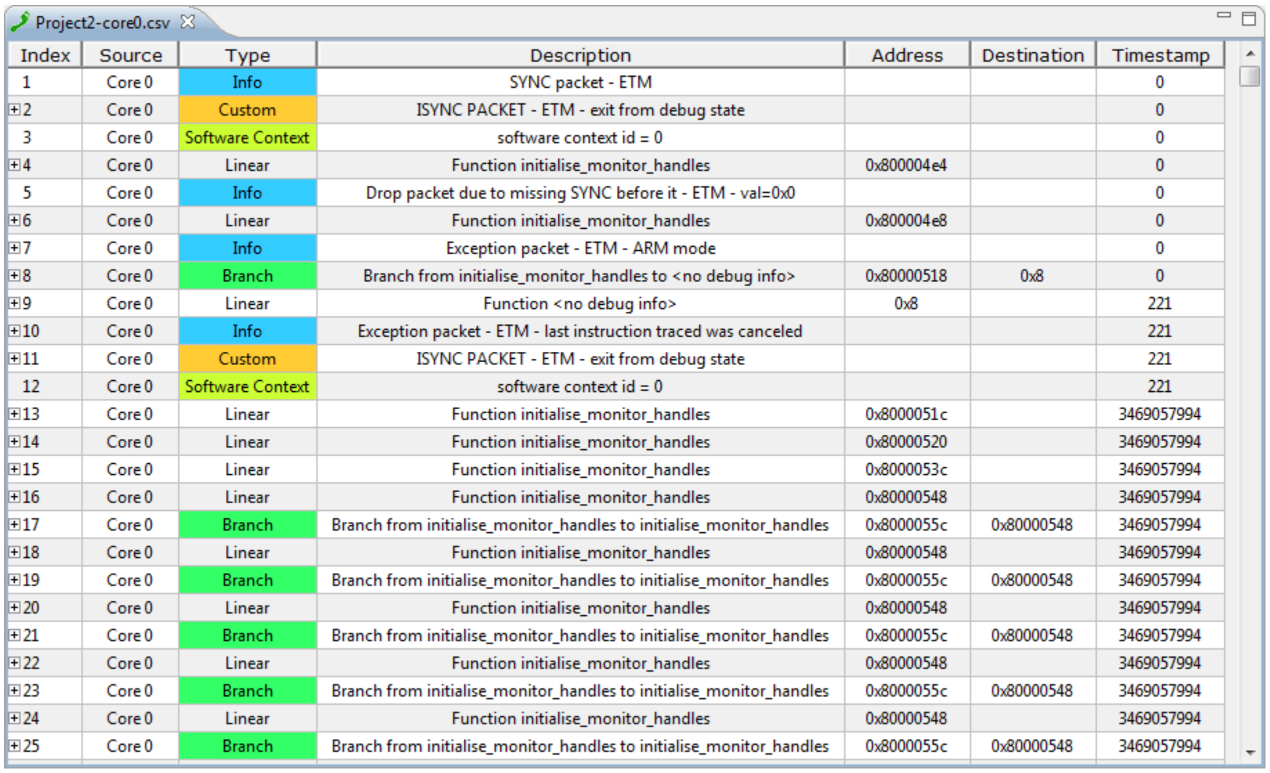
To view trace data:

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

   All trace collections performed for the project are displayed.

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

Figure 12: Trace viewer



Project2-core0.csv

Index	Source	Type	Description	Address	Destination	Timestamp
1	Core 0	Info	SYNC packet - ETM			0
+2	Core 0	Custom	ISYNC PACKET - ETM - exit from debug state			0
3	Core 0	Software Context	software context id = 0			0
+4	Core 0	Linear	Function initialise_monitor_handles	0x800004e4		0
5	Core 0	Info	Drop packet due to missing SYNC before it - ETM - val=0x0			0
+6	Core 0	Linear	Function initialise_monitor_handles	0x800004e8		0
+7	Core 0	Info	Exception packet - ETM - ARM mode			0
+8	Core 0	Branch	Branch from initialise_monitor_handles to <no debug info>	0x80000518	0x8	0
+9	Core 0	Linear	Function <no debug info>	0x8		221
+10	Core 0	Info	Exception packet - ETM - last instruction traced was canceled			221
+11	Core 0	Custom	ISYNC PACKET - ETM - exit from debug state			221
12	Core 0	Software Context	software context id = 0			221
+13	Core 0	Linear	Function initialise_monitor_handles	0x8000051c		3469057994
+14	Core 0	Linear	Function initialise_monitor_handles	0x80000520		3469057994
+15	Core 0	Linear	Function initialise_monitor_handles	0x8000053c		3469057994
+16	Core 0	Linear	Function initialise_monitor_handles	0x80000548		3469057994
+17	Core 0	Branch	Branch from initialise_monitor_handles to initialise_monitor_handles	0x8000055c	0x80000548	3469057994
+18	Core 0	Linear	Function initialise_monitor_handles	0x80000548		3469057994
+19	Core 0	Branch	Branch from initialise_monitor_handles to initialise_monitor_handles	0x8000055c	0x80000548	3469057994
+20	Core 0	Linear	Function initialise_monitor_handles	0x80000548		3469057994
+21	Core 0	Branch	Branch from initialise_monitor_handles to initialise_monitor_handles	0x8000055c	0x80000548	3469057994
+22	Core 0	Linear	Function initialise_monitor_handles	0x80000548		3469057994
+23	Core 0	Branch	Branch from initialise_monitor_handles to initialise_monitor_handles	0x8000055c	0x80000548	3469057994
+24	Core 0	Linear	Function initialise_monitor_handles	0x80000548		3469057994
+25	Core 0	Branch	Branch from initialise_monitor_handles to initialise_monitor_handles	0x8000055c	0x80000548	3469057994

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

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

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

Table 4: Trace viewer fields

Field	Description
Index	Each decoded event has a unique number starting with 1.
Source	Displays the source function of the trace line if it is a call or a branch.

Table continues on the next page...

Table 4: Trace viewer fields (continued)

Field	Description
Type	Specifies the type of event that has occurred, such as Info, Branch, Error, Function Return and Function Call.
Description	Displays detailed information about the trace line.
Address	Displays the starting address of the target function.
Destination	Displays the end address of the target function.
Timestamp	Specifies the timestamp value that is expressed as clock ticks. Depending on the configuration of the trace source, this can be relative to the event's trace source or relative to all events logged from all sources.

2.2.2 Timeline viewer

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

To view timeline data:

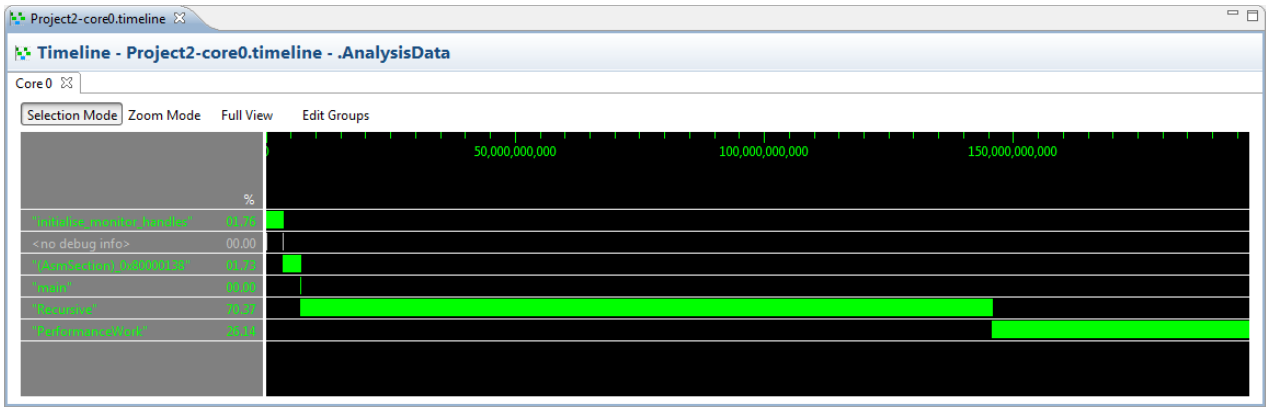
1. In the **Analysis Results** view, expand the project name.

   The data source is listed under the project name.

2. Click the **Timeline** hyperlink.

   The **Timeline** viewer appears.

Figure 13: Timeline viewer



	%
"initialise_monitor_handles"	01.76
<no debug info>	00.00
"(AsmSection)_0x80000138"	01.73
"main"	00.00
"Recursive"	70.37
"PerformanceWork"	26.14

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

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

- Selection Mode on page 21
- Zoom Mode on page 21
- Full View on page 21
- Edit Groups on page 21

2.2.2.1 Selection Mode

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

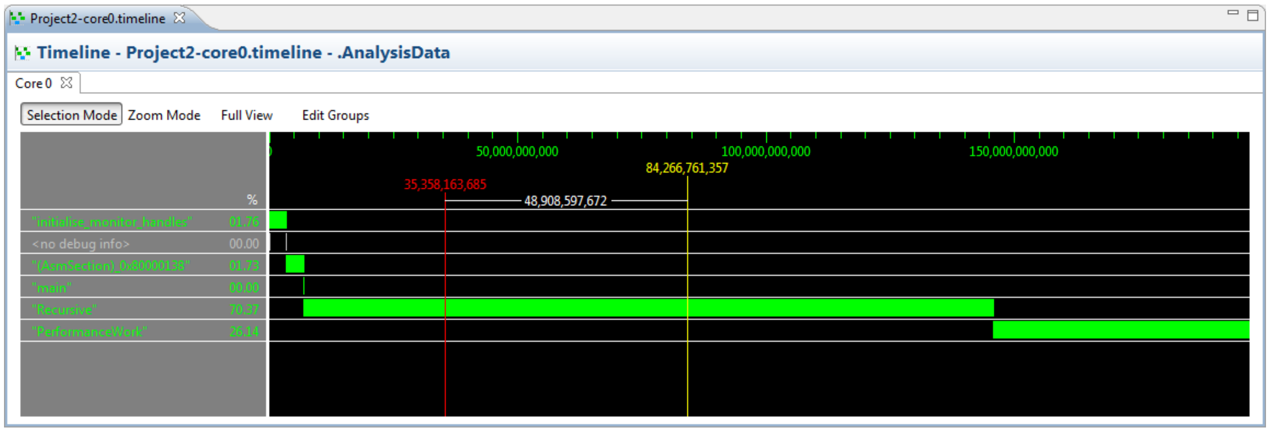
To mark a point in the bar:

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

   A yellow vertical line appears displaying the number of cycles at that point.
3. Right-click another point in the bar.

   A red vertical line appears displaying the number of cycles at that point along with the difference of cycles between two marked points.

Figure 14: Selection mode to measure difference of cycles between functions



	%
"initialise_monitor_handles"	01.76
<no debug info>	00.00
"(AsmSection)_0x80000138"	01.73
"main"	00.00
"Recursive"	70.37
"PerformanceWork"	26.14

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

2.2.2.2 Zoom Mode

The **Zoom Mode** allows you to zoom-in and zoom-out in the timeline graph.

Click **Zoom Mode** and then click the timeline graph to zoom-in. To zoom-out, right-click in the timeline graph. You can also move the mouse wheel up and down to zoom-in and zoom-out.

2.2.2.3 Full View

The **Full View** allows you to get back to the original view if you selected the Zoom mode.

NOTE

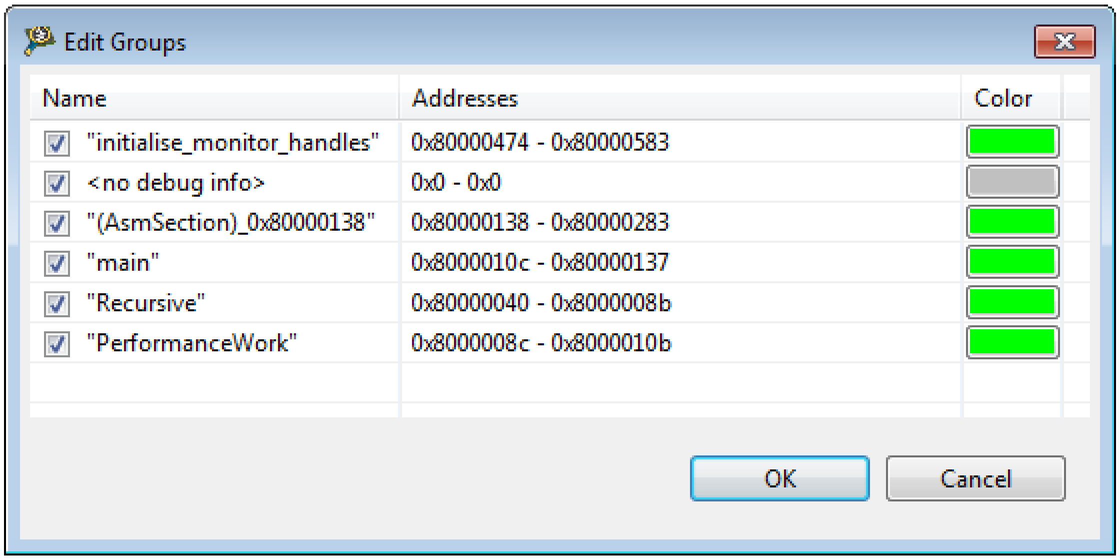
The **Selection Mode** is the default mode of the timeline view.

2.2.2.4 Edit Groups

The **Edit Groups** lets you customize the timeline according to your requirements.

For example, you can change the default color of the line bars representing the functions to differentiate between them. You can add/remove a function to/from the timeline. To perform these functions, click **Edit Groups**. The **Edit Groups** dialog appears.

Figure 15: Edit Groups dialog



Name	Addresses	Color
☑ "initialise_monitor_handles"	0x80000474 - 0x80000583	🟩
☑ <no debug info>	0x0 - 0x0	⬜
☑ "(AsmSection)_0x80000138"	0x80000138 - 0x80000283	🟩
☑ "main"	0x8000010c - 0x80000137	🟩
☑ "Recursive"	0x80000040 - 0x8000008b	🟩
☑ "PerformanceWork"	0x8000008c - 0x8000010b	🟩

2.2.3 Code Coverage viewer

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

To view code coverage data:

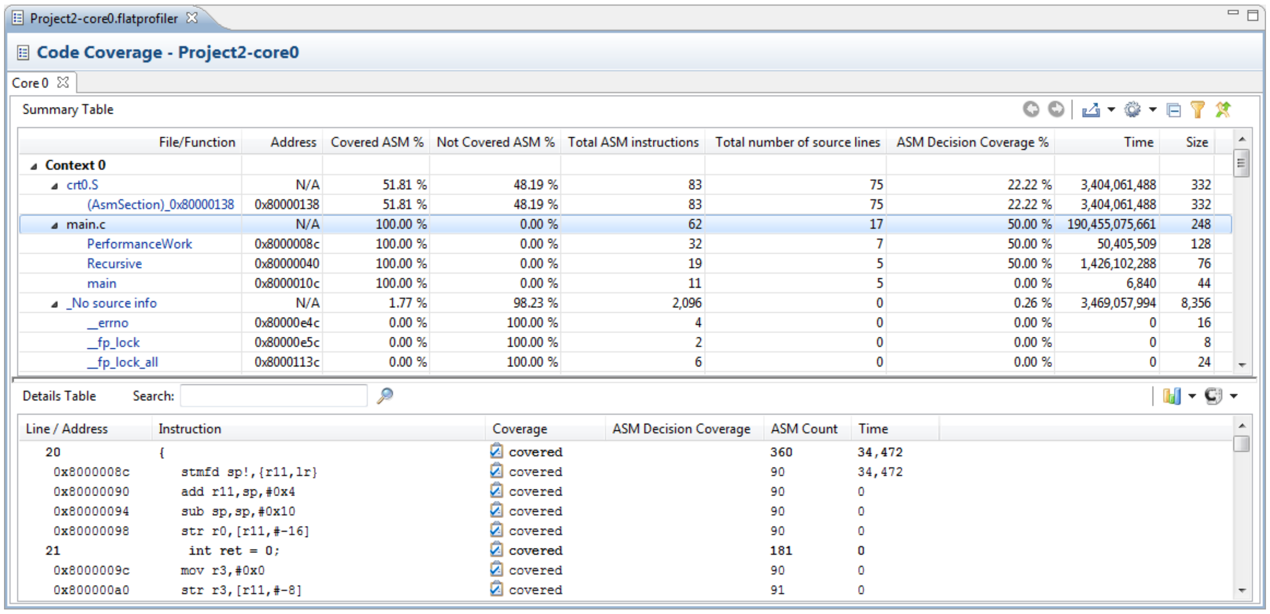
1. In the **Analysis Results** view, expand the project name.

   All trace collections performed for the project appear.

2. Click the **Code Coverage** hyperlink.

   The **Code Coverage** viewer appears.

Figure 16: Code Coverage viewer



File/Function	Address	Covered ASM %	Not Covered ASM %	Total ASM instructions	Total number of source lines	ASM Decision Coverage %	Time	Size
Context 0								
crt0.S	N/A	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
(AsmSection)_0x80000138	0x80000138	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
main.c	N/A	100.00 %	0.00 %	62	17	50.00 %	190,455,075,661	248
PerformanceWork	0x8000008c	100.00 %	0.00 %	32	7	50.00 %	50,405,509	128
Recursive	0x80000040	100.00 %	0.00 %	19	5	50.00 %	1,426,102,288	76
main	0x8000010c	100.00 %	0.00 %	11	5	0.00 %	6,840	44
_No source info	N/A	1.77 %	98.23 %	2,096	0	0.26 %	3,469,057,994	8,356
__errno	0x80000e4c	0.00 %	100.00 %	4	0	0.00 %	0	16
__fp_lock	0x80000e5c	0.00 %	100.00 %	2	0	0.00 %	0	8
__fp_lock_all	0x8000113c	0.00 %	100.00 %	6	0	0.00 %	0	24

Line / Address	Instruction	Coverage	ASM Decision Coverage	ASM Count	Time
20	{	☑ covered		360	34,472
0x8000008c	stmfd sp!,{r11,lr}	☑ covered		90	34,472
0x80000090	add r11,sp,#0x4	☑ covered		90	0
0x80000094	sub sp,sp,#0x10	☑ covered		90	0
0x80000098	str r0,[r11,#-16]	☑ covered		90	0
21	int ret = 0;	☑ covered		181	0
0x8000009c	mov r3,#0x0	☑ covered		90	0
0x800000a0	str r3,[r11,#-8]	☑ covered		91	0

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

This section contains the following subsections:

- Summary table on page 23
- Details table on page 25

2.2.3.1 Summary table

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

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

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

Table 5: Summary table - Description of ASM instructions statistics

Name	Description
File/Function	Displays the name of the function that has executed.
Address	Displays the start address of the function.
Covered ASM %	Displays the percentage of number of assembly instructions executed from the total number of assembly instructions per function or per source file.
Not Covered ASM %	Displays the percentage of number of assembly instructions not executed from the total number of assembly instructions per function or per source file.
Total ASM instructions	Displays the total number of assembly instructions per function and per source file.
ASM Decision Coverage %	Displays the decision coverage computed for direct and indirect conditional branches. It is the mean value of the individual decision coverages. Therefore, 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	Displays the total number of clock cycles that the function takes.
Size	Displays the number of bytes required by each 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.

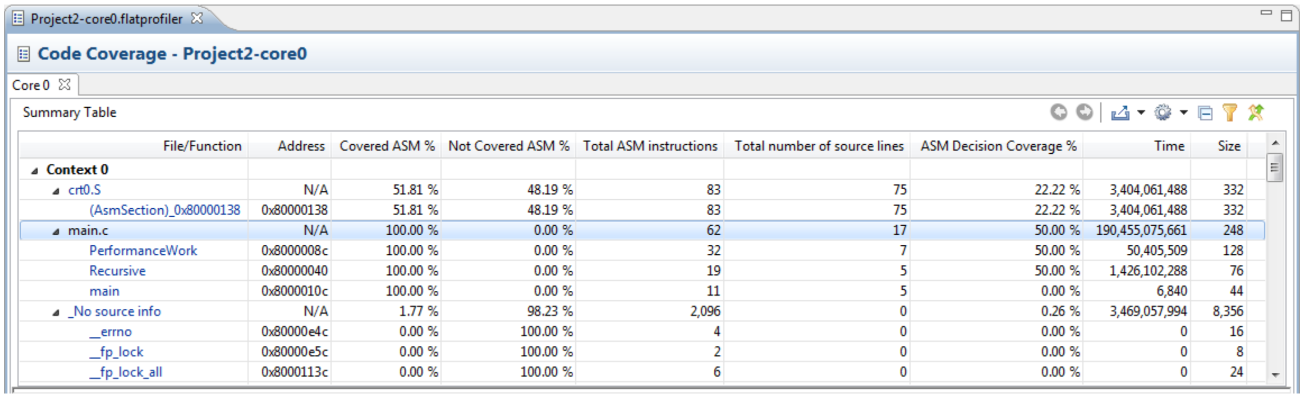


Figure 17: Summary table of Code Coverage viewer

Project2-core0.flatprofiler

Code Coverage - Project2-core0

Core 0

Summary Table

File/Function	Address	Covered ASM %	Not Covered ASM %	Total ASM instructions	Total number of source lines	ASM Decision Coverage %	Time	Size
Context 0								
crt0.S	N/A	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
(AsmSection)_0x80000138	0x80000138	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
main.c	N/A	100.00 %	0.00 %	62	17	50.00 %	190,455,075,661	248
PerformanceWork	0x8000008c	100.00 %	0.00 %	32	7	50.00 %	50,405,509	128
Recursive	0x80000040	100.00 %	0.00 %	19	5	50.00 %	1,426,102,288	76
main	0x8000010c	100.00 %	0.00 %	11	5	0.00 %	6,840	44
_No source info	N/A	1.77 %	98.23 %	2,096	0	0.26 %	3,469,057,994	8,356
__errno	0x80000e4c	0.00 %	100.00 %	4	0	0.00 %	0	16
__fp_lock	0x80000e5c	0.00 %	100.00 %	2	0	0.00 %	0	8
__fp_lock_all	0x8000113c	0.00 %	100.00 %	6	0	0.00 %	0	24

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

Table 6: Buttons available in Summary table of Code Coverage viewer

Name	Button	Description
Previous function	◀	Lets you view the details of the previous function that was selected in the **Summary** table before the currently selected function. Click it to view the details of the previous function.
Next function	▶	Lets you view the details of the next function that was selected in the **Summary** table. **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.

Table continues on the next page...

Table 6: Buttons available in Summary table of Code Coverage viewer (continued)

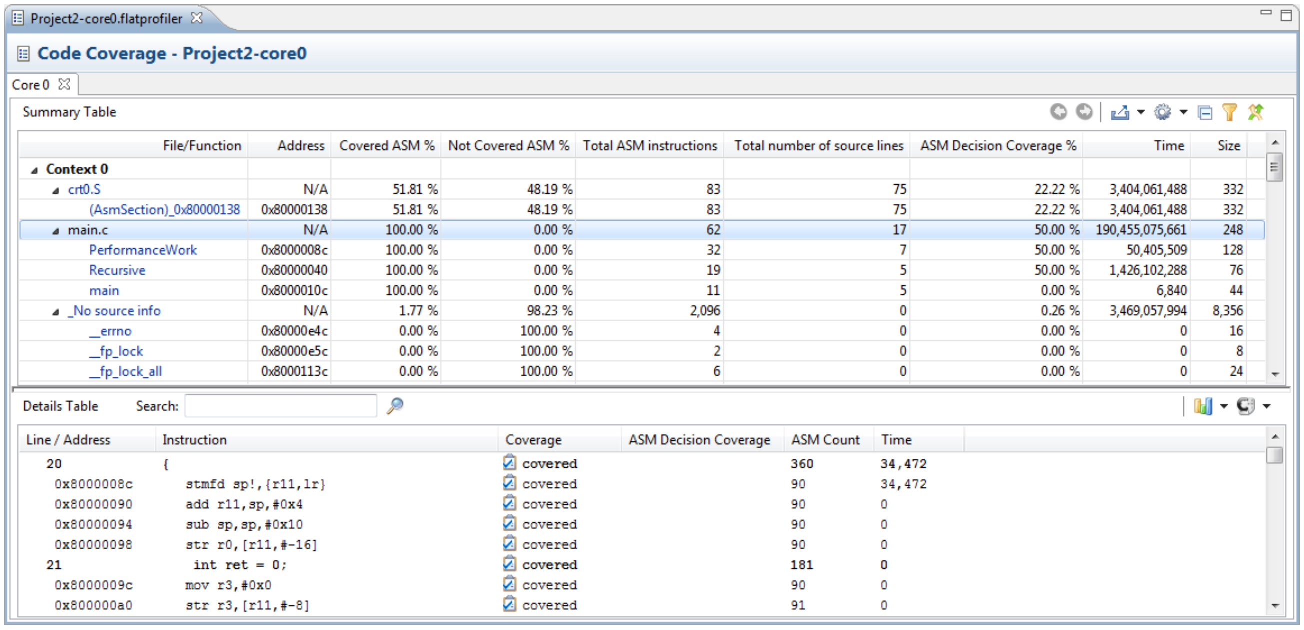
Name	Button	Description
Configure table	⚙ ▾	Lets you show and hide column(s) of the Code Coverage data. Click the button and select the appropriate option to show/hide columns of the **Summary/Details** table. The **Drag and drop to order columns** dialog appears in which you can select/deselect the checkboxes corresponding to the available columns to show/hide them in the **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.

2.2.3.2 Details table

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

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

Figure 18: Details table



Project2-core0.flatprofiler

Code Coverage - Project2-core0

Core 0

Summary Table

File/Function	Address	Covered ASM %	Not Covered ASM %	Total ASM instructions	Total number of source lines	ASM Decision Coverage %	Time	Size
Context 0								
crt0.S	N/A	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
(AsmSection)_0x80000138	0x80000138	51.81 %	48.19 %	83	75	22.22 %	3,404,061,488	332
main.c	N/A	100.00 %	0.00 %	62	17	50.00 %	190,455,075,661	248
PerformanceWork	0x8000008c	100.00 %	0.00 %	32	7	50.00 %	50,405,509	128
Recursive	0x80000040	100.00 %	0.00 %	19	5	50.00 %	1,426,102,288	76
main	0x8000010c	100.00 %	0.00 %	11	5	0.00 %	6,840	44
_No source info	N/A	1.77 %	98.23 %	2,096	0	0.26 %	3,469,057,994	8,356
__errno	0x80000e4c	0.00 %	100.00 %	4	0	0.00 %	0	16
__fp_lock	0x80000e5c	0.00 %	100.00 %	2	0	0.00 %	0	8
__fp_lock_all	0x8000113c	0.00 %	100.00 %	6	0	0.00 %	0	24

Details Table Search:

Line / Address	Instruction	Coverage	ASM Decision Coverage	ASM Count	Time
20	`{`	✓ covered		360	34,472
0x8000008c	`stmfd sp!,{r11,lr}`	✓ covered		90	34,472
0x80000090	`add r11,sp,#0x4`	✓ covered		90	0
0x80000094	`sub sp,sp,#0x10`	✓ covered		90	0
0x80000098	`str r0,[r11,#-16]`	✓ covered		90	0
21	`int ret = 0;`	✓ covered		181	0
0x8000009c	`mov r3,#0x0`	✓ covered		90	0
0x800000a0	`str r3,[r11,#-8]`	✓ covered		91	0

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

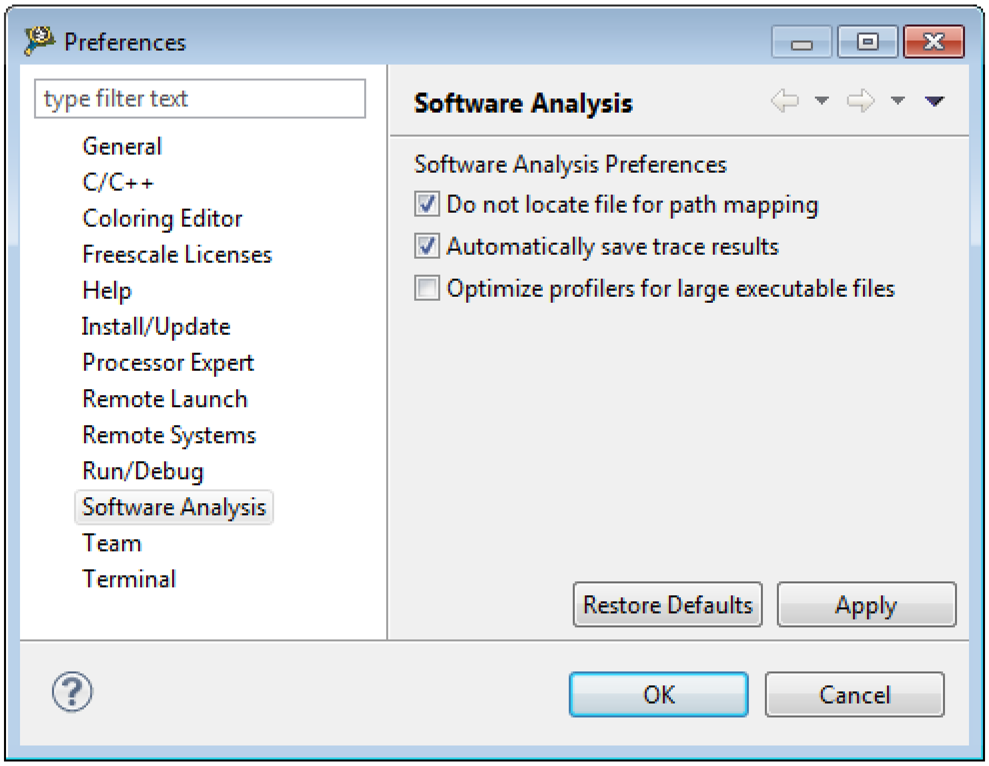
Table 7: Details table - description of fields

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

When you double-click the instruction in the **Details Table**, the corresponding source file is opened. To disable the path mapping option when you double-click the instruction, follow these steps:

1. Choose **Window > Preferences**. The **Preferences** dialog appears.
2. Click **Software Analysis** in the left pane of the **Preferences** dialog, and select the **Do not locate file for path mapping** checkbox in the right pane, as shown in the figure below.

Figure 19: Preferences dialog



Software Analysis

Software Analysis Preferences

- [✓] Do not locate file for path mapping
- [✓] Automatically save trace results
- [ ] Optimize profilers for large executable files

3. Click **OK** to apply the setting and close the **Preferences** dialog.

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

- Search - Lets you search for a particular text in the **Details** table. In the **Search** text box, type the data that you want to search and click the **Search** button. The first instance of the data is selected in the statistics view. Click the button again or press the *Enter* key to view the next instances of the data.
- Graphics - Lets you display the histograms in two colors for the **ASM Count** and **Time** columns in the bottom view of the 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.

2.2.4 Performance viewer

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

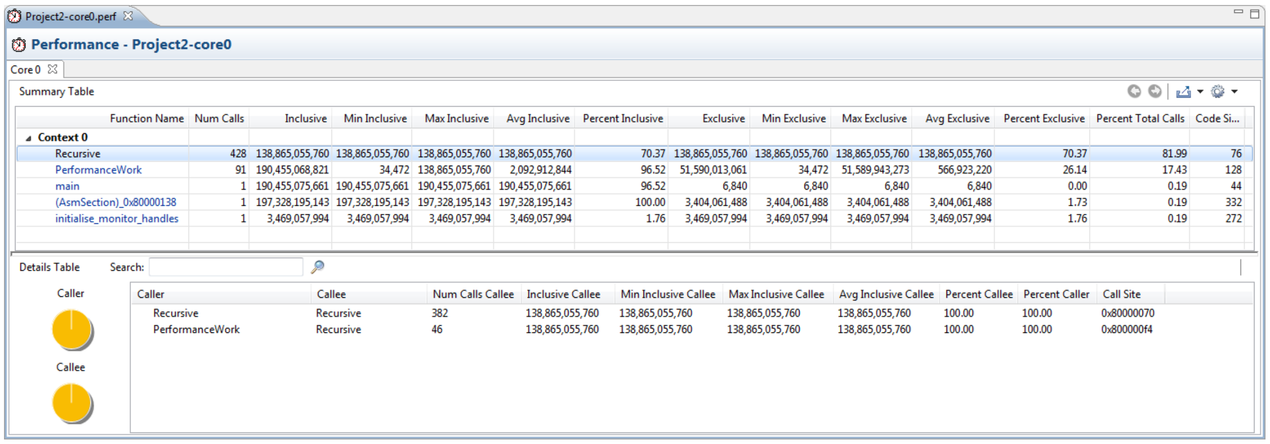
To view performance data:

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

   All trace collections performed for the project are displayed.
2. Click the **Performance** hyperlink.

   The **Performance** viewer appears.

Figure 20: Performance viewer



Function Name	Num Calls	Inclusive	Min Inclusive	Max Inclusive	Avg Inclusive	Percent Inclusive	Exclusive	Min Exclusive	Max Exclusive	Avg Exclusive	Percent Exclusive	Percent Total Calls	Code Si...
Context 0													
Recursive	428	138,865,055,760	138,865,055,760	138,865,055,760	138,865,055,760	70.37	138,865,055,760	138,865,055,760	138,865,055,760	138,865,055,760	70.37	81.99	76
PerformanceWork	91	190,455,068,821	34,472	138,865,055,760	2,092,912,844	96.52	51,590,013,061	34,472	51,589,943,273	566,923,220	26.14	17.43	128
main	1	190,455,075,661	190,455,075,661	190,455,075,661	190,455,075,661	96.52	6,840	6,840	6,840	6,840	0.00	0.19	44
(AsmSection)_0x80000138	1	197,328,195,143	197,328,195,143	197,328,195,143	197,328,195,143	100.00	3,404,061,488	3,404,061,488	3,404,061,488	3,404,061,488	1.73	0.19	332
initialise_monitor_handles	1	3,469,057,994	3,469,057,994	3,469,057,994	3,469,057,994	1.76	3,469,057,994	3,469,057,994	3,469,057,994	3,469,057,994	1.76	0.19	272

Caller	Callee	Num Calls Callee	Inclusive Callee	Min Inclusive Callee	Max Inclusive Callee	Avg Inclusive Callee	Percent Callee	Percent Caller	Call Site
Recursive	Recursive	382	138,865,055,760	138,865,055,760	138,865,055,760	138,865,055,760	100.00	100.00	0x80000070
PerformanceWork	Recursive	46	138,865,055,760	138,865,055,760	138,865,055,760	138,865,055,760	100.00	100.00	0x800000f4

The **Performance** viewer is divided into two views:

- The top view presents function performance data in the **Summary** table. It displays the count and invocation information for each function that executes during the measurement, enabling you to compare the relative data for various portions of your target program. The information in the **Summary** table can be sorted by column in ascending or descending order. Click the column header to sort the corresponding data. The table below describes the fields of the **Summary** table.
- The bottom view or the **Details** table presents call pair data for the function selected in the **Summary** table. The **Details** table displays call pair relationships for the selected function, that is which function called which function. Each function pair consists of a caller and a callee. The percent caller and percent callee data is also displayed graphically. The functions are represented in different colors in the pie chart, you can move the mouse cursor over the color to see the corresponding function. The next table below describes the fields of the **Details** table. You cannot sort the columns of this table.

Table 8: Field description of Summary table

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

Table 9: Field description of Details table

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

You can move the columns to the left or right of another column by dragging and dropping. You can perform the Export and Configure table actions on the performance data similar to critical code data. You can also view the previous and next functions of the performance data using the icons available in the lower section of the **Performance** viewer. For details on these icons, see Table 6. Buttons available in Summary table of Code Coverage viewer on page 24.

2.2.5 Call Tree viewer

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

To view call tree data:

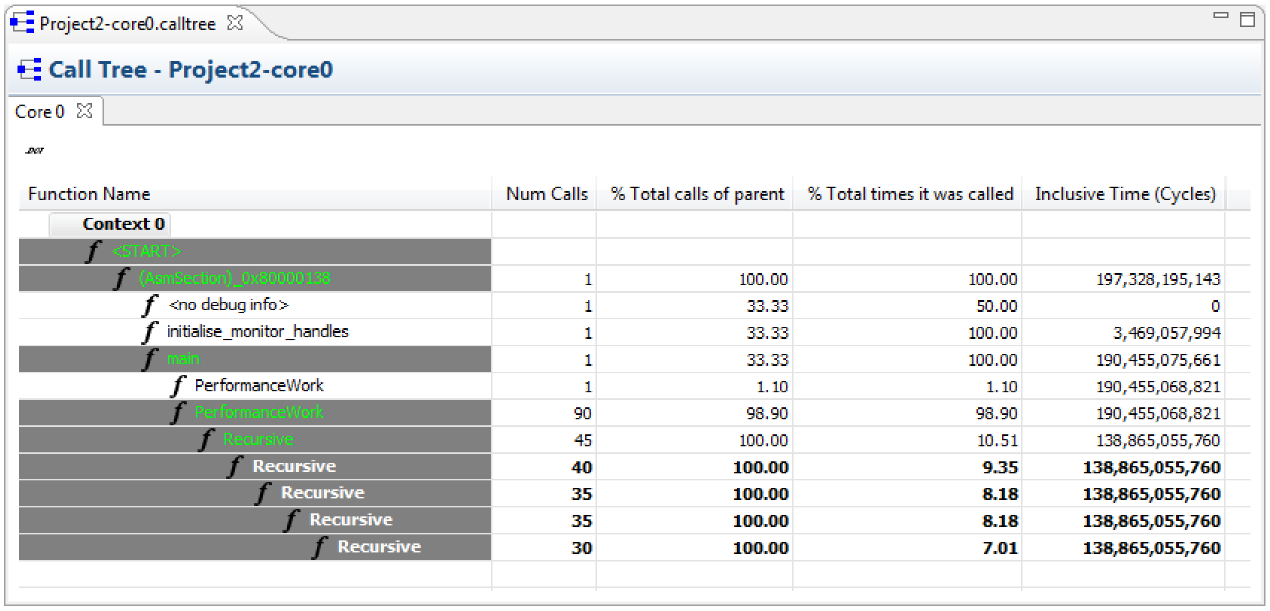
1. In the **Analysis Results** view, expand the project name.

   All trace collections performed for the project are displayed.

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

   The **Call Tree** viewer appears.

Figure 21: Call Tree viewer



Project2-core0.calltree

Call Tree - Project2-core0

Core 0

Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles)
Context 0				
<START>				
(AsmSection)_0x80000138	1	100.00	100.00	197,328,195,143
<no debug info>	1	33.33	50.00	0
initialise_monitor_handles	1	33.33	100.00	3,469,057,994
main	1	33.33	100.00	190,455,075,661
PerformanceWork	1	1.10	1.10	190,455,068,821
PerformanceWork	90	98.90	98.90	190,455,068,821
Recursive	45	100.00	10.51	138,865,055,760
Recursive	40	100.00	9.35	138,865,055,760
Recursive	35	100.00	8.18	138,865,055,760
Recursive	35	100.00	8.18	138,865,055,760
Recursive	30	100.00	7.01	138,865,055,760

In the **Call Tree** viewer, START is the root of the tree. You can click "+" to expand the tree and "-" to collapse the tree. It shows the biggest depth for stack utilization in **Call Tree** and the functions on this call path are displayed in green color.

The Call Tree nodes are synchronized with the source code. You can double-click the node to view the source code.

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

Table 10: Call Tree viewer fields

Name	Description
Function Name	Name of function that has executed.
Num Calls	Number of times function has executed.
% Total calls of parent	Percent of number of function calls from total number of calls in the application.
% Total times it was called	Percent of number of times a function was called.
Inclusive Time (Microseconds; 50.0 MHz)	Cumulative count during execution time spent from function entry to exit.

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

- **Export to dot** - Exports call tree data in the *.dot* format using the button .

Chapter 3
Collecting and Viewing Linux Trace

The Linux satrace tool allows you to collect the trace of a program without using any hardware probe.

The tool encapsulates the trace configurator and probe. It is delivered as a standalone component containing an executable and some shared libraries. The Linux satrace tool is independent of CodeWarrior; it does not provide any 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 contains the following sections:

- Collecting Linux trace using CodeWarrior on page 31
- Viewing Linux trace collected without using CodeWarrior on page 42

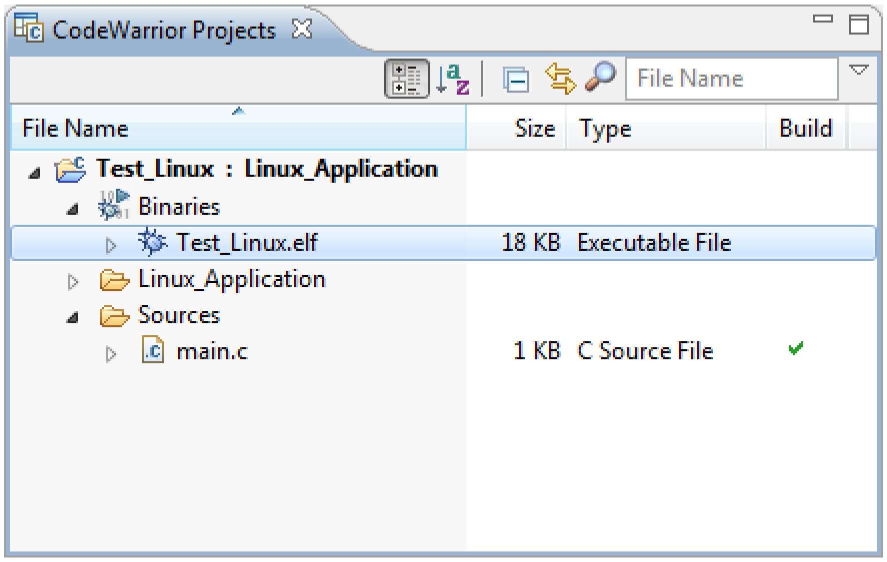
3.1 Collecting Linux trace using CodeWarrior

This section explains how to collect Linux trace by establishing a Remote System Explorer (RSE) connection with the board.

To collect trace using the Linux satrace tool, perform these steps:

1. Start CW for ARMv7 and create a CodeWarrior Linux Project.
2. Write the application using the Editor.
3. Build the project to generate an ELF file, as shown in the figure below.

Figure 22: ELF file

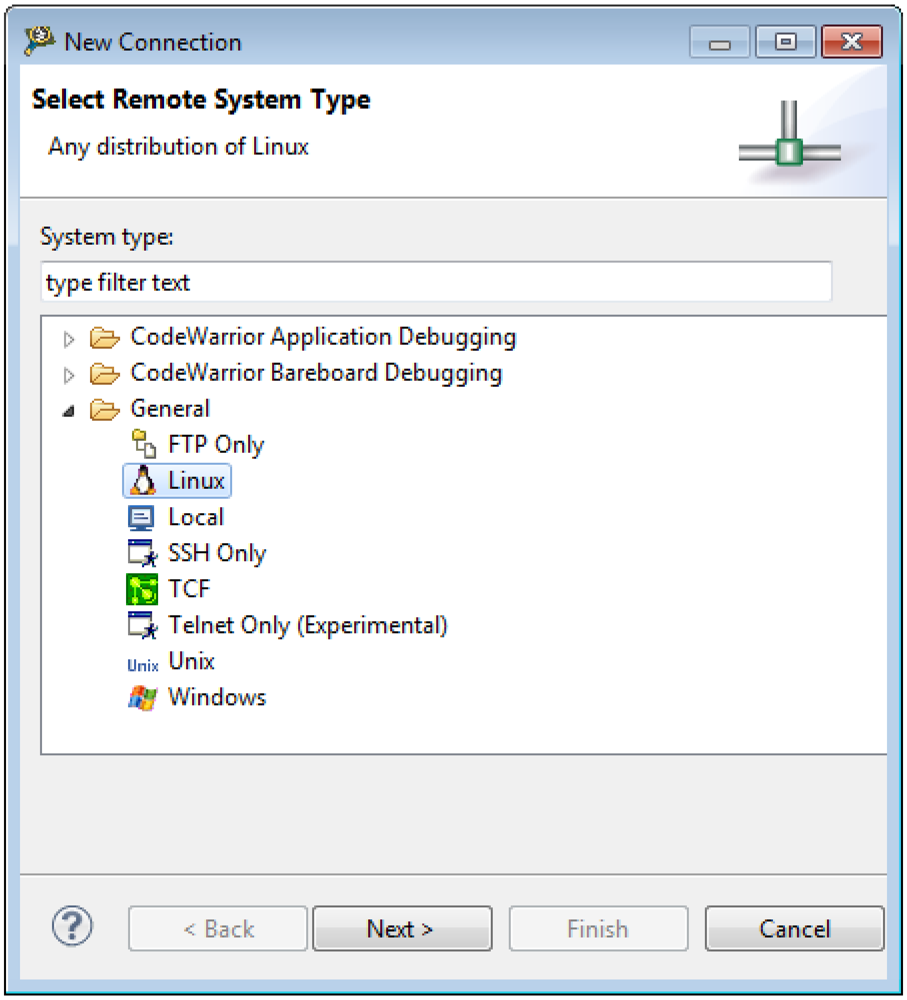


File Name	Size	Type	Build
Test_Linux : Linux_Application			
Binaries			
Test_Linux.elf	18 KB	Executable File	
Linux_Application			
Sources			
main.c	1 KB	C Source File	✓

4. Create a Linux connection using RSE:
   a. To open RSE, click **Window > Open Perspective > Other > Remote System Explorer.**
   b. On the toolbar of the **Remote Systems** view, click **Define a connection to remote system.** The RSE wizard starts.
   c. Expand **General** and select **Linux** option from the list.

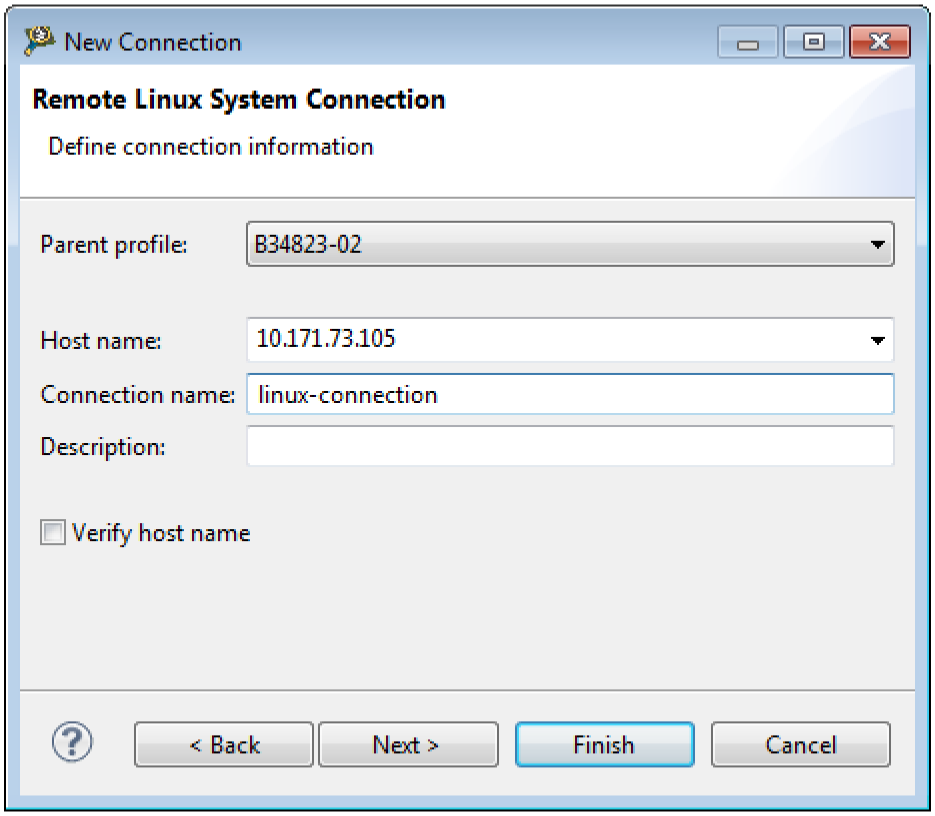
d. Click **Next**.

Figure 23: RSE wizard


New Connection
Select Remote System Type
Any distribution of Linux
System type:
type filter text
CodeWarrior Application Debugging
CodeWarrior Bareboard Debugging
General
FTP Only
Linux
Local
SSH Only
TCF
Telnet Only (Experimental)
Unix
Windows
< Back
Next >
Finish
Cancel

e. The **Remote Linux System Connection** page appears. Specify the host name and connection name and click **Next**.

Figure 24: Remote Linux System Connection page



New Connection

Remote Linux System Connection

Define connection information

Parent profile: B34823-02

Host name: 10.171.73.105

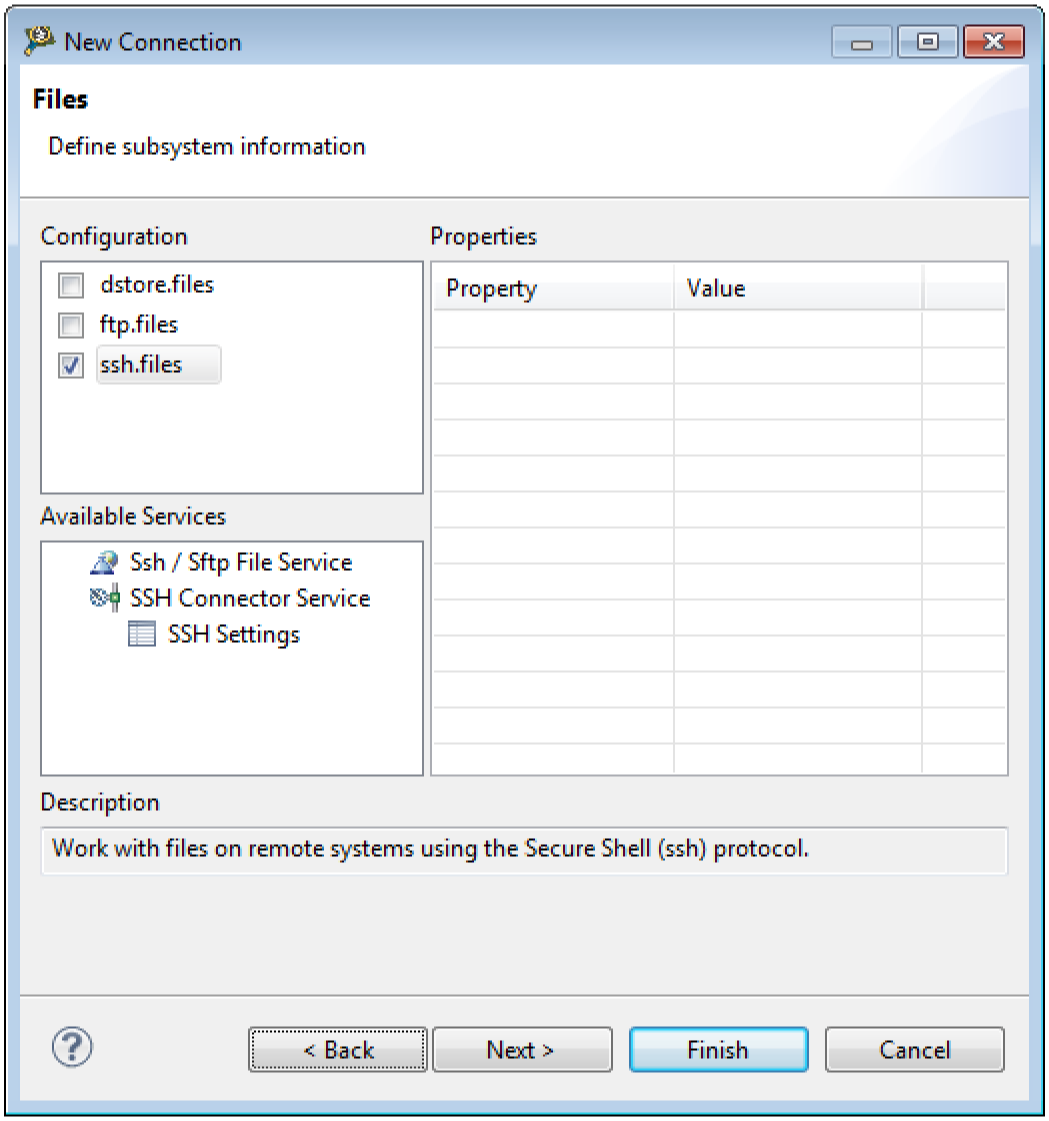
Connection name: linux-connection

Description:

[ ] Verify host name

f. The **Files** page appears. Select the **ssh.files** checkbox and click **Next**.

Figure 25: Files page



New Connection

Files

Define subsystem information

Configuration

- [ ] dstore.files
- [ ] ftp.files
- [x] ssh.files

Available Services

- Ssh / Sftp File Service
- SSH Connector Service
  - SSH Settings

Properties

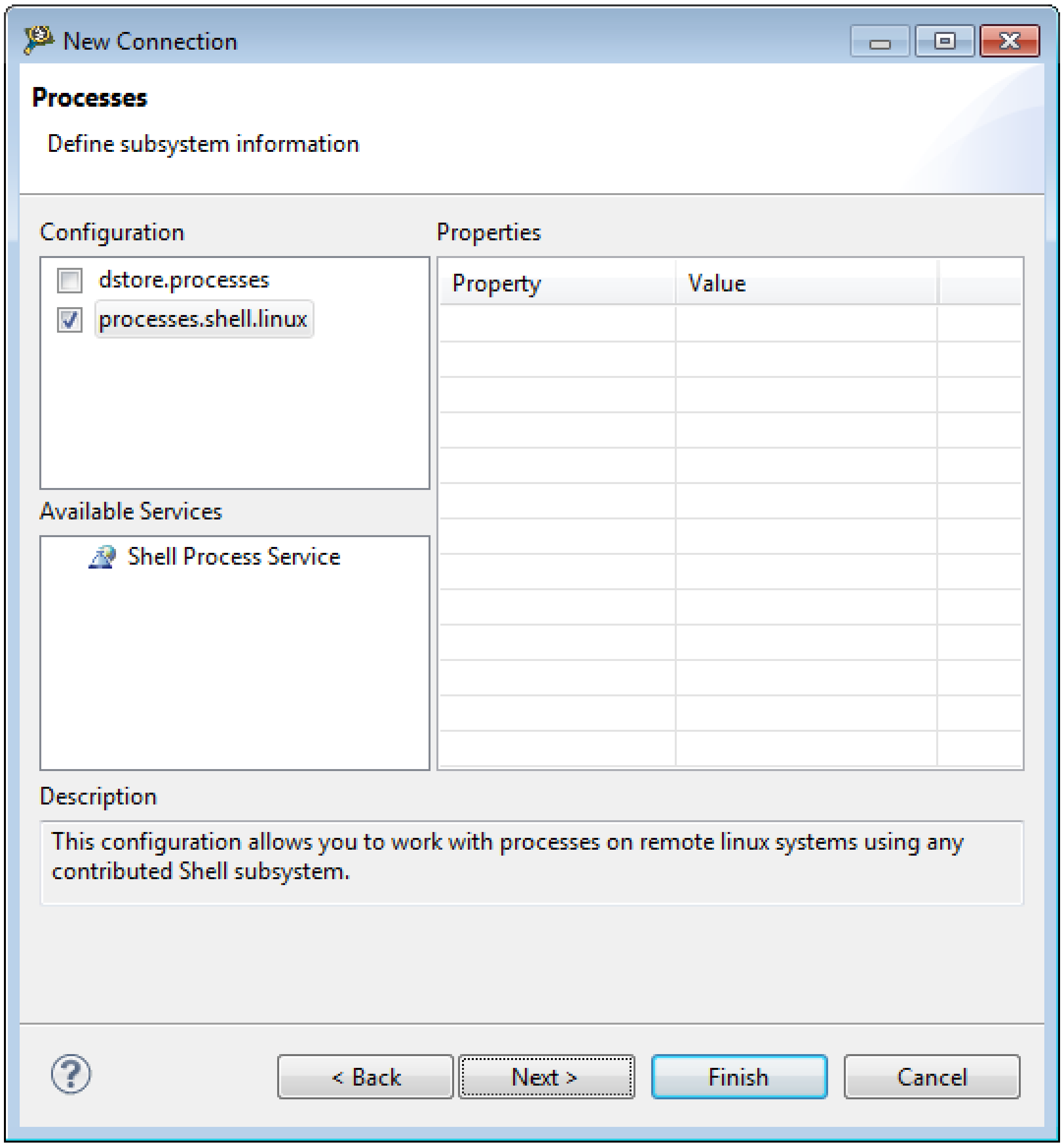
Property	Value

Description

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

g. The **Processes** page appears. Select the **processes.shell.linux** checkbox and click **Next**.

Figure 26: Processes page



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

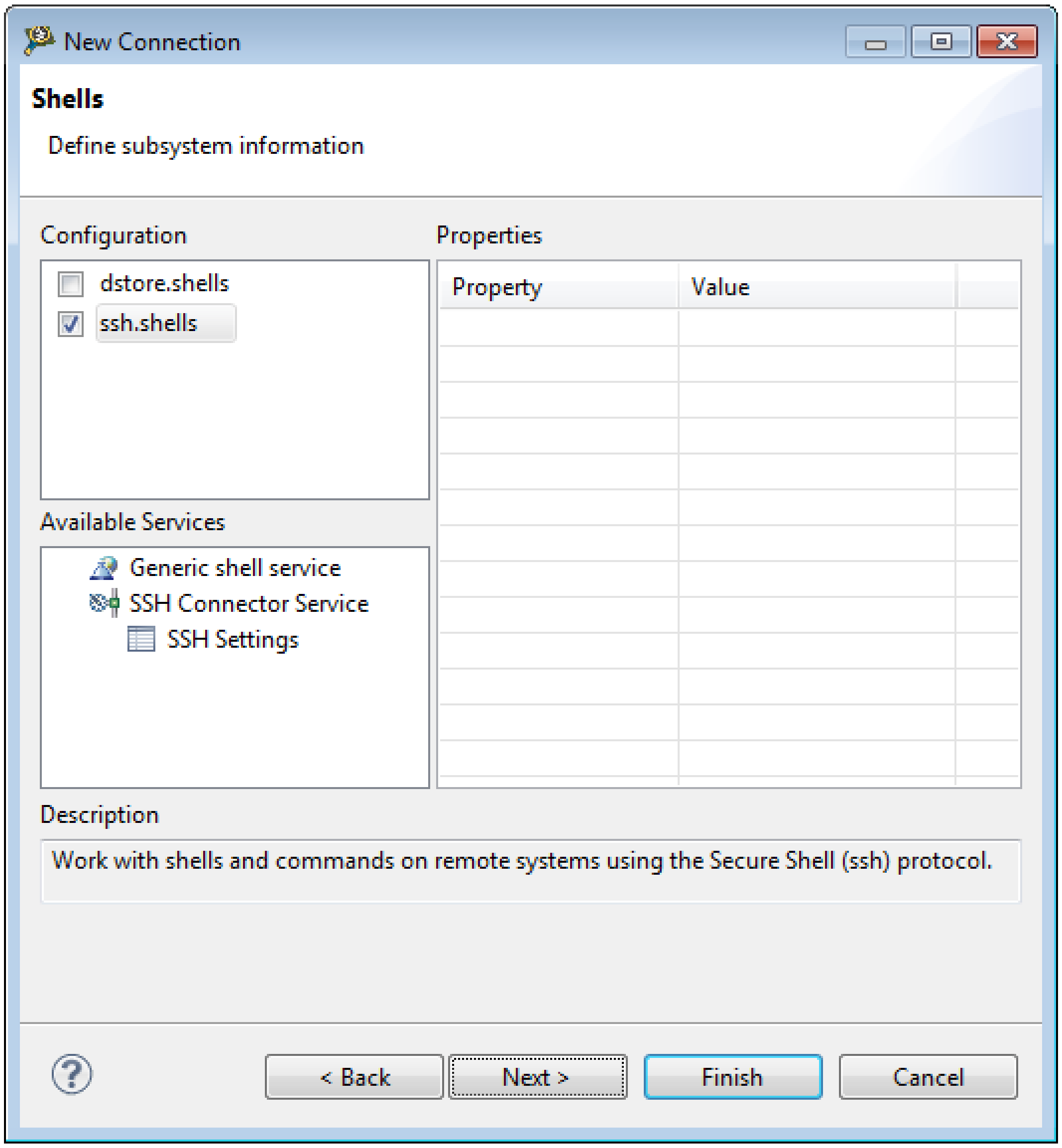
Shell Process Service

Property	Value

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

h. The **Shells** page appears. Select the **ssh.shells** checkbox and click **Finish**.

Figure 27: Shells page



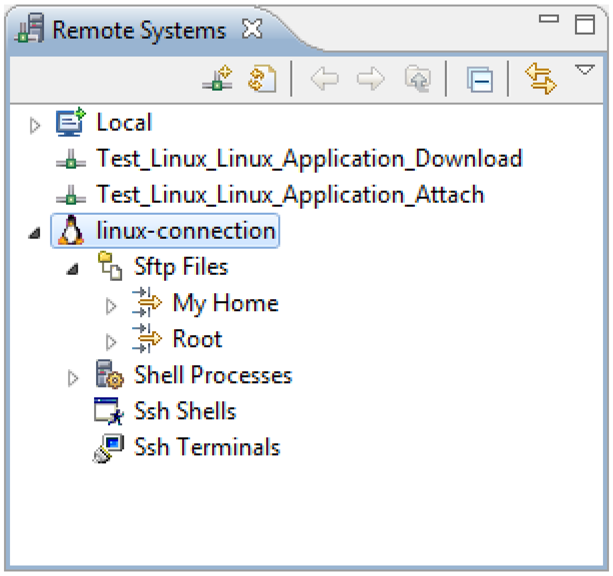
☐ dstore.shells
☑ ssh.shells

Generic shell service
SSH Connector Service
SSH Settings

Work with shells and commands on remote systems using the Secure Shell (ssh) protocol.

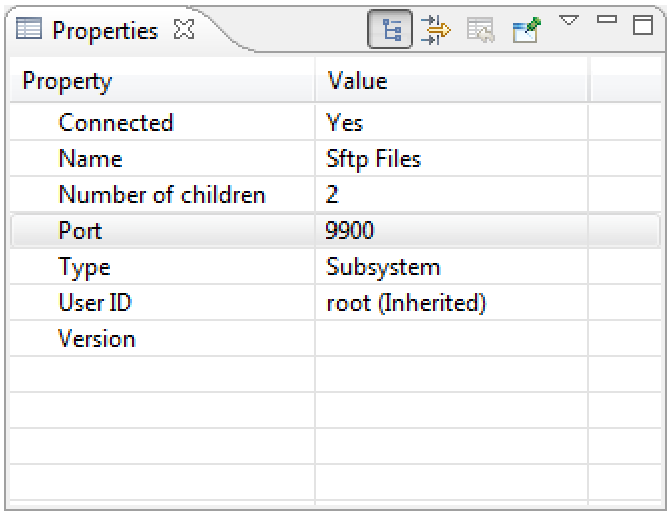
In the **Remote Systems** view, you can see that the connection with the board has been established. The connection name is **linux-connection**.

Figure 28: Remote Systems view



i. Select **Sftp Files** in the **Remote Systems** view and specify the port number in the **Properties** window, as shown in the figure below.

Figure 29: Setting port number



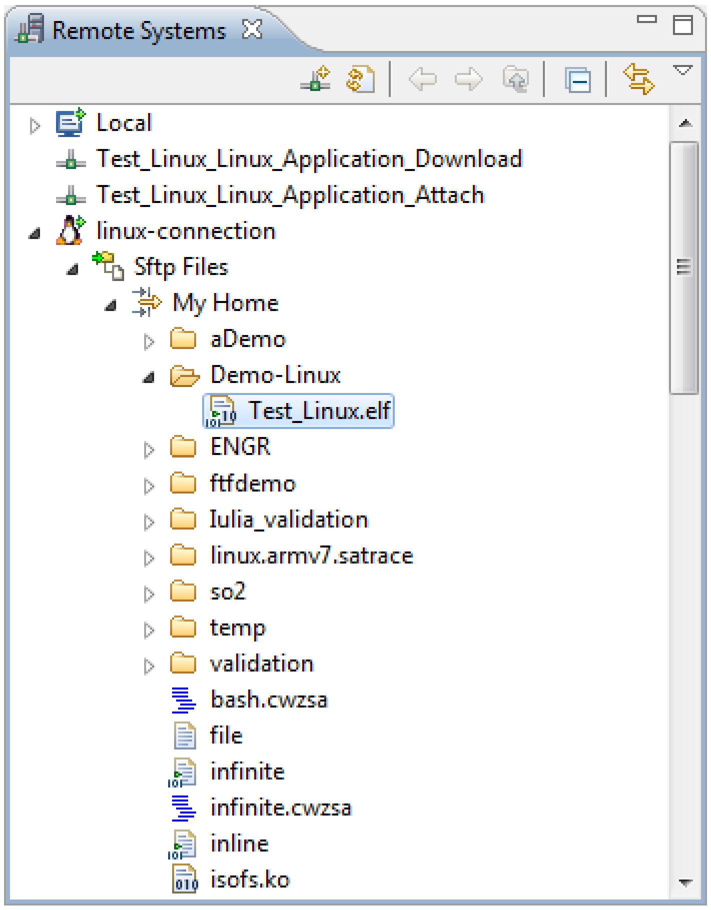
Property	Value
Connected	Yes
Name	Sftp Files
Number of children	2
Port	9900
Type	Subsystem
User ID	root (Inherited)
Version	

j. Connect to the Linux system by right-clicking **Sftp Files** and choosing **Connect**.

k. Create a folder under **Sftp Files > My Home** where you can store the files related to your current project.

l. Now, copy your ELF file from the **CodeWarrior Projects** view and paste it into the folder you just created, as shown in the figure below.

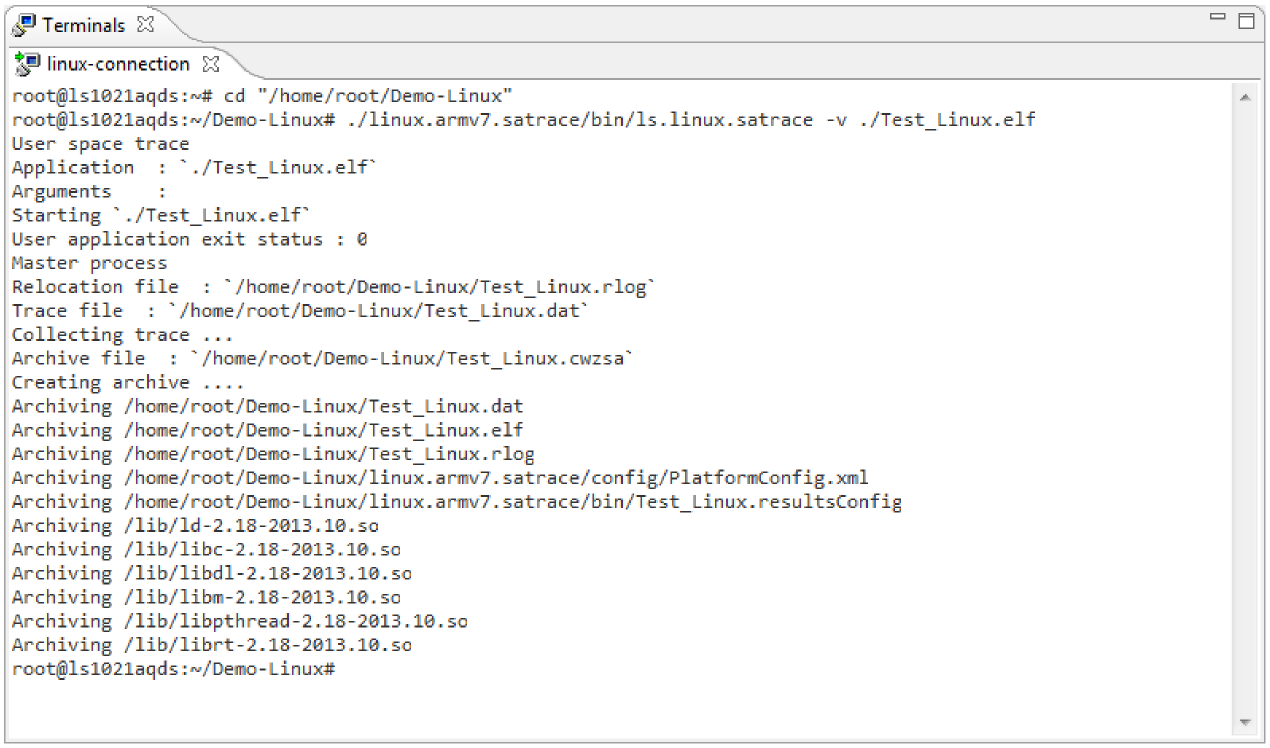
Figure 30: ELF file in RSE view



5. Right-click the parent folder where you have pasted the ELF file, choose **Add Trace Support** from the context menu to copy the satrace into a directory.
6. Right-click the parent folder and choose **Launch Terminal** from the context menu.
7. In the **Terminals** view, run the satrace using the following command:

```
./linux.armv7.satrace/bin/ls.linux.satrace -v ./Test_Linux.elf
```

Figure 31: Terminals view

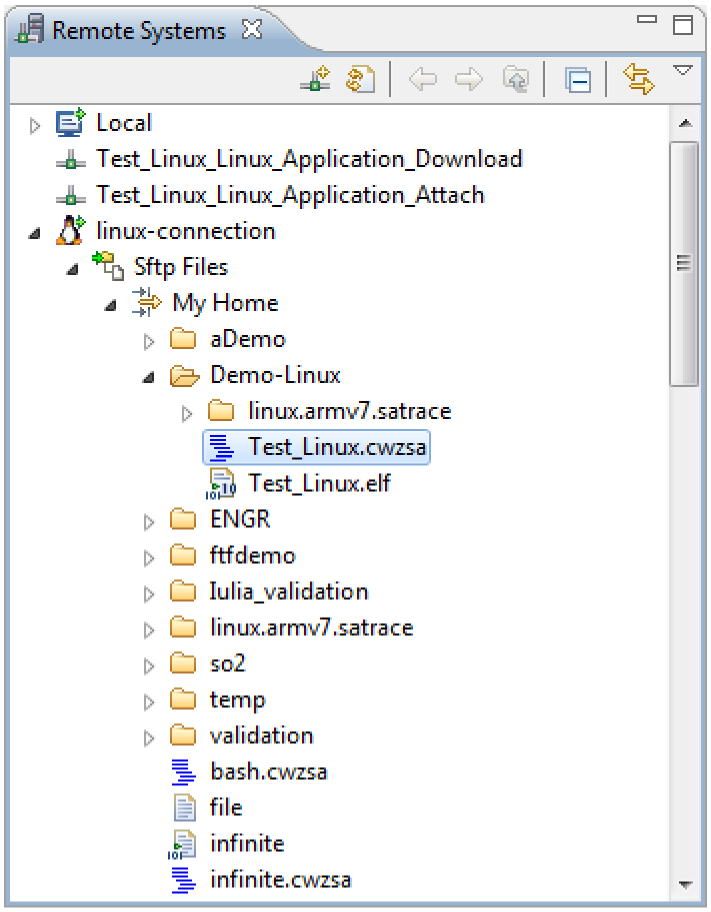


```
root@ls1021aqds:~# cd "/home/root/Demo-Linux"
root@ls1021aqds:~/Demo-Linux# ./linux.armv7.satrace/bin/ls.linux.satrace -v ./Test_Linux.elf
User space trace
Application  : `./Test_Linux.elf`
Arguments    :
Starting `./Test_Linux.elf`
User application exit status : 0
Master process
Relocation file  : `/home/root/Demo-Linux/Test_Linux.rlog`
Trace file  : `/home/root/Demo-Linux/Test_Linux.dat`
Collecting trace ...
Archive file  : `/home/root/Demo-Linux/Test_Linux.cwzsa`
Creating archive ....
Archiving /home/root/Demo-Linux/Test_Linux.dat
Archiving /home/root/Demo-Linux/Test_Linux.elf
Archiving /home/root/Demo-Linux/Test_Linux.rlog
Archiving /home/root/Demo-Linux/linux.armv7.satrace/config/PlatformConfig.xml
Archiving /home/root/Demo-Linux/linux.armv7.satrace/bin/Test_Linux.resultsConfig
Archiving /lib/ld-2.18-2013.10.so
Archiving /lib/libc-2.18-2013.10.so
Archiving /lib/libdl-2.18-2013.10.so
Archiving /lib/libm-2.18-2013.10.so
Archiving /lib/libpthread-2.18-2013.10.so
Archiving /lib/librt-2.18-2013.10.so
root@ls1021aqds:~/Demo-Linux#
```

The tracing starts and collects the data in the *.cwzsa file under the parent folder.

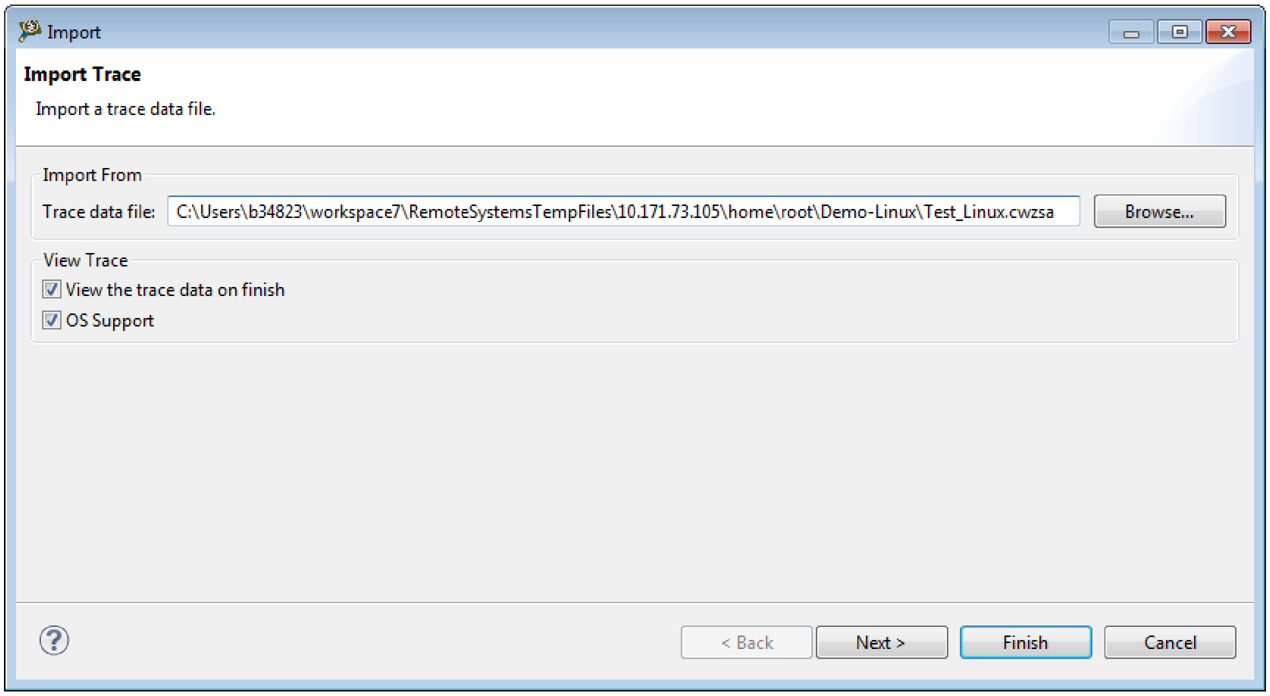
8. Right-click the parent folder and choose **Refresh**. You can see the **.cwzsa* file under the parent folder, as shown below.

Figure 32: Trace data file



9. Double-click the **.cwzsa* file. The **Import** wizard starts, as shown in the figure below.

Figure 33: Import wizard



Import

Import Trace

Import a trace data file.

Import From

Trace data file: C:\Users\b34823\workspace7\RemoteSystemsTempFiles\10.171.73.105\home\root\Demo-Linux\Test_Linux.cwzsa

View Trace

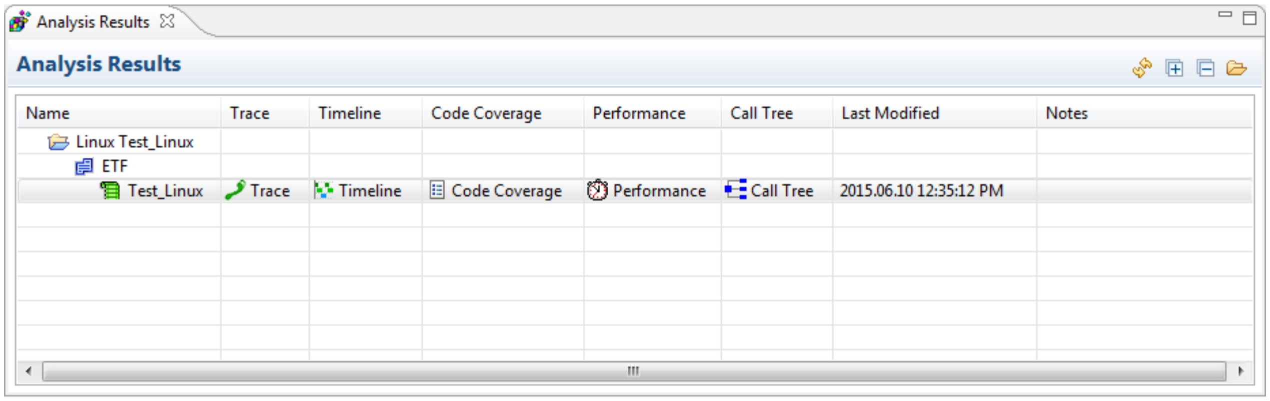
☑ View the trace data on finish

☑ OS Support

10.Click **Finish** to end the **Import** wizard. The file is imported and it is displayed in the **Analysis Results** view.

11.Click the **Trace** link under the **Trace** column in the **Analysis Results** view to view the trace data, as shown in the figure below.

Figure 34: Analysis Results view

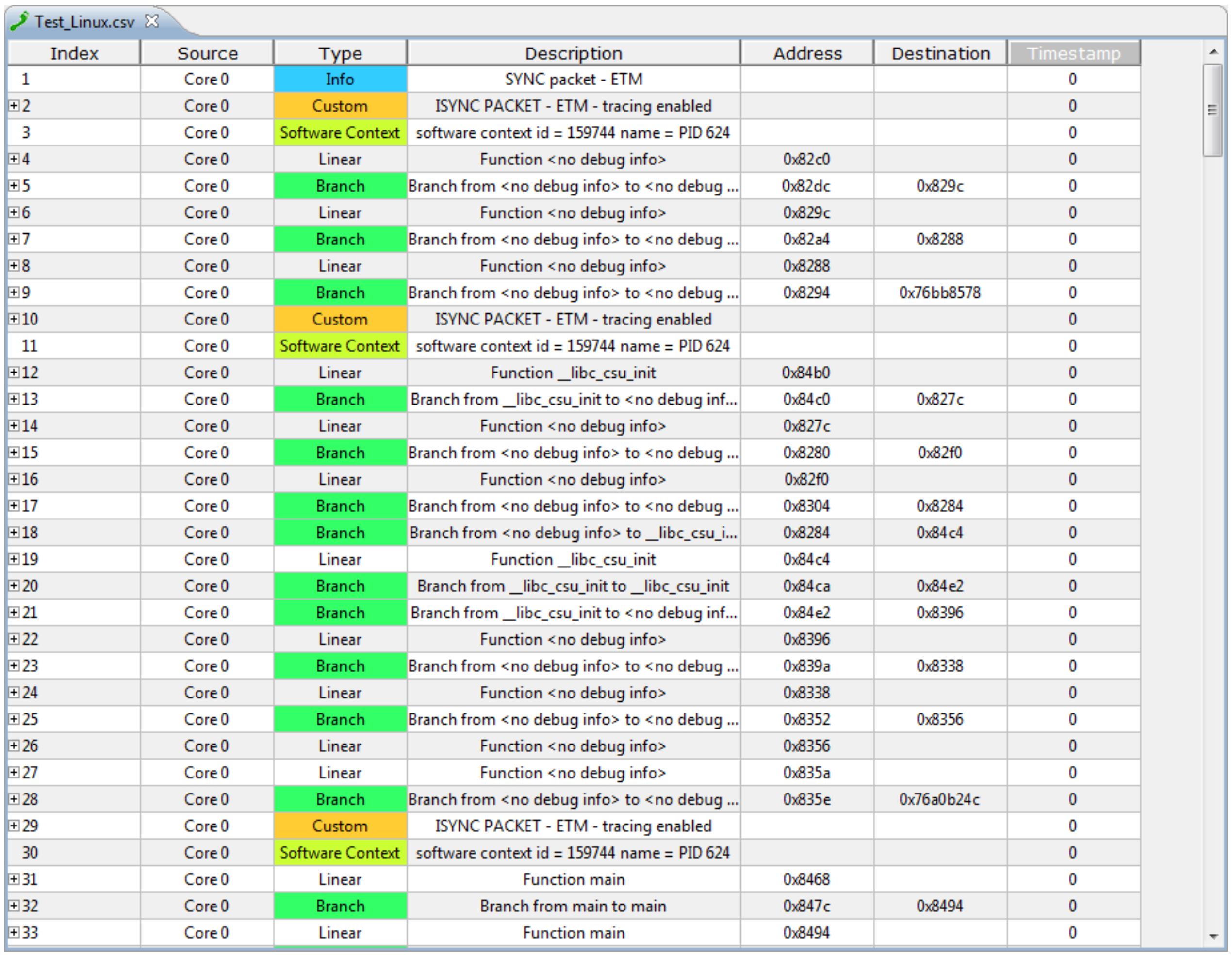


Analysis Results

Name	Trace	Timeline	Code Coverage	Performance	Call Tree	Last Modified	Notes
Linux Test_Linux							
ETF							
Test_Linux	Trace	Timeline	Code Coverage	Performance	Call Tree	2015.06.10 12:35:12 PM	

The trace data file opens in the **Trace** viewer showing the trace results, as shown in the figure below.

Figure 35: Trace data file



Test_Linux.csv

Index	Source	Type	Description	Address	Destination	Timestamp
1	Core 0	Info	SYNC packet - ETM			0
⊞2	Core 0	Custom	ISYNC PACKET - ETM - tracing enabled			0
3	Core 0	Software Context	software context id = 159744 name = PID 624			0
⊞4	Core 0	Linear	Function <no debug info>	0x82c0		0
⊞5	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x82dc	0x829c	0
⊞6	Core 0	Linear	Function <no debug info>	0x829c		0
⊞7	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x82a4	0x8288	0
⊞8	Core 0	Linear	Function <no debug info>	0x8288		0
⊞9	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x8294	0x76bb8578	0
⊞10	Core 0	Custom	ISYNC PACKET - ETM - tracing enabled			0
11	Core 0	Software Context	software context id = 159744 name = PID 624			0
⊞12	Core 0	Linear	Function __libc_csu_init	0x84b0		0
⊞13	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x84c0	0x827c	0
⊞14	Core 0	Linear	Function <no debug info>	0x827c		0
⊞15	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x8280	0x82f0	0
⊞16	Core 0	Linear	Function <no debug info>	0x82f0		0
⊞17	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x8304	0x8284	0
⊞18	Core 0	Branch	Branch from <no debug info> to __libc_csu_i...	0x8284	0x84c4	0
⊞19	Core 0	Linear	Function __libc_csu_init	0x84c4		0
⊞20	Core 0	Branch	Branch from __libc_csu_init to __libc_csu_init	0x84ca	0x84e2	0
⊞21	Core 0	Branch	Branch from __libc_csu_init to <no debug inf...	0x84e2	0x8396	0
⊞22	Core 0	Linear	Function <no debug info>	0x8396		0
⊞23	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x839a	0x8338	0
⊞24	Core 0	Linear	Function <no debug info>	0x8338		0
⊞25	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x8352	0x8356	0
⊞26	Core 0	Linear	Function <no debug info>	0x8356		0
⊞27	Core 0	Linear	Function <no debug info>	0x835a		0
⊞28	Core 0	Branch	Branch from <no debug info> to <no debug ...	0x835e	0x76a0b24c	0
⊞29	Core 0	Custom	ISYNC PACKET - ETM - tracing enabled			0
30	Core 0	Software Context	software context id = 159744 name = PID 624			0
⊞31	Core 0	Linear	Function main	0x8468		0
⊞32	Core 0	Branch	Branch from main to main	0x847c	0x8494	0
⊞33	Core 0	Linear	Function main	0x8494		0

3.2 Viewing Linux trace collected without using CodeWarrior

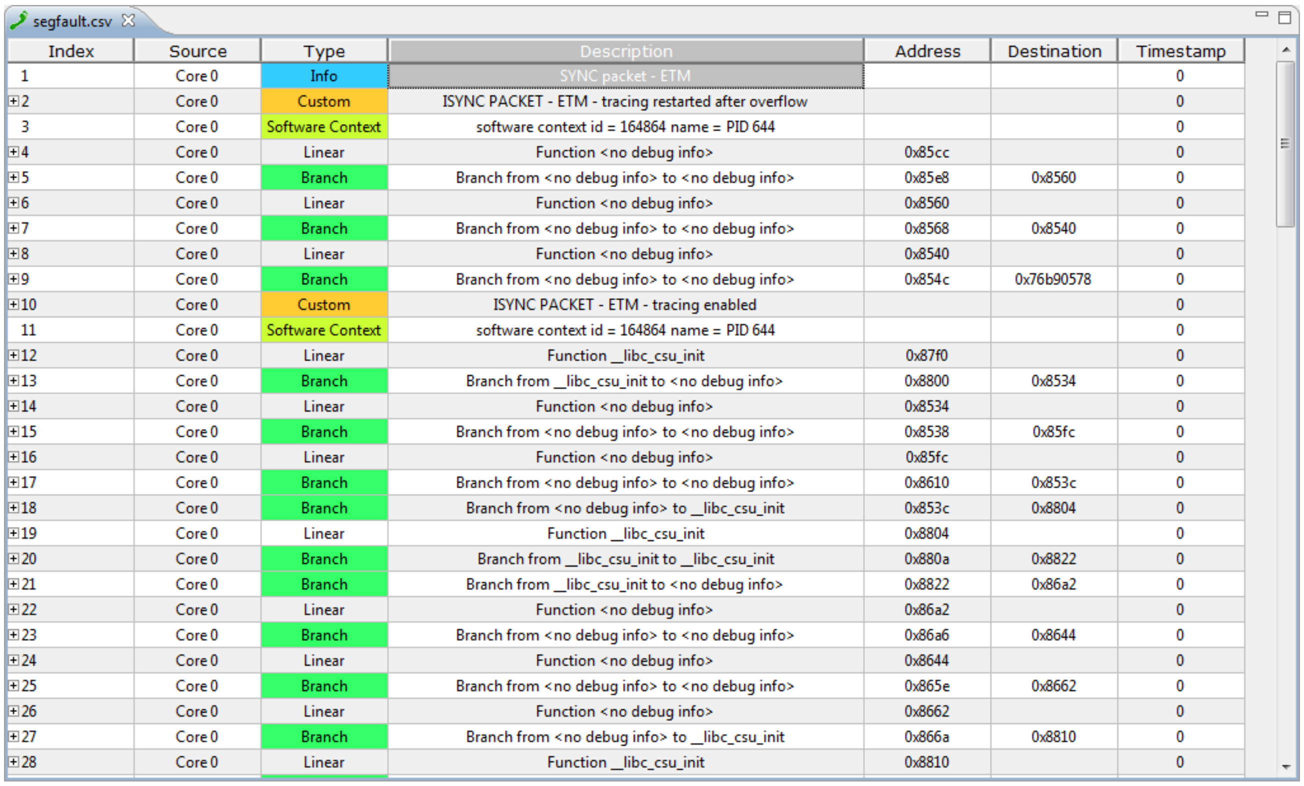
This section explains how to view Linux trace data collected from a board without using CodeWarrior.

To explain how to view Linux trace, this section uses trace data collected from the LS102xA TWR board without using CodeWarrior. See AN5001 for instructions on how to collect Linux trace without using CodeWarrior.

The standalone tracing tool generates a `*.cwzsa` file, an archive type that can be imported and fully decoded using ARMv7 decoder or ARMv7 CodeWarrior. To view the generated `segfault.cwzsa` file in CodeWarrior for ARMv7, perform these steps:

1. Drag the `segfault.cwzsa` file into the CodeWarrior IDE. The **Import** wizard starts.
2. Click **Finish** to end the **Import** wizard. The file is imported and it is displayed in the **Analysis Results** view.
3. Click the **Trace** link under the **Trace** column in the **Analysis Results** view to view the trace data. The decoding of trace and profiling data starts. The decoded trace data is displayed in **Trace** viewer, as shown in the figure below.

Figure 36: Trace data



segfault.csv

Index	Source	Type	Description	Address	Destination	Timestamp
1	Core 0	Info	SYNC packet - ETM			0
⊞2	Core 0	Custom	ISYNC PACKET - ETM - tracing restarted after overflow			0
3	Core 0	Software Context	software context id = 164864 name = PID 644			0
⊞4	Core 0	Linear	Function <no debug info>	0x85cc		0
⊞5	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x85e8	0x8560	0
⊞6	Core 0	Linear	Function <no debug info>	0x8560		0
⊞7	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x8568	0x8540	0
⊞8	Core 0	Linear	Function <no debug info>	0x8540		0
⊞9	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x854c	0x76b90578	0
⊞10	Core 0	Custom	ISYNC PACKET - ETM - tracing enabled			0
11	Core 0	Software Context	software context id = 164864 name = PID 644			0
⊞12	Core 0	Linear	Function __libc_csu_init	0x87f0		0
⊞13	Core 0	Branch	Branch from __libc_csu_init to <no debug info>	0x8800	0x8534	0
⊞14	Core 0	Linear	Function <no debug info>	0x8534		0
⊞15	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x8538	0x85fc	0
⊞16	Core 0	Linear	Function <no debug info>	0x85fc		0
⊞17	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x8610	0x853c	0
⊞18	Core 0	Branch	Branch from <no debug info> to __libc_csu_init	0x853c	0x8804	0
⊞19	Core 0	Linear	Function __libc_csu_init	0x8804		0
⊞20	Core 0	Branch	Branch from __libc_csu_init to __libc_csu_init	0x880a	0x8822	0
⊞21	Core 0	Branch	Branch from __libc_csu_init to <no debug info>	0x8822	0x86a2	0
⊞22	Core 0	Linear	Function <no debug info>	0x86a2		0
⊞23	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x86a6	0x8644	0
⊞24	Core 0	Linear	Function <no debug info>	0x8644		0
⊞25	Core 0	Branch	Branch from <no debug info> to <no debug info>	0x865e	0x8662	0
⊞26	Core 0	Linear	Function <no debug info>	0x8662		0
⊞27	Core 0	Branch	Branch from <no debug info> to __libc_csu_init	0x866a	0x8810	0
⊞28	Core 0	Linear	Function __libc_csu_init	0x8810		0

You can also find the profiling data files in the same folder where `segfault.cwzsa` file is present. You can drag-and-drop the profiling data files in CodeWarrior to view the profiling data in the Timeline, Code Coverage, Performance, and Call Tree viewers. The Timeline, Code Coverage, Performance, and Call Tree viewers are explained in Viewing trace data using Analysis Results view on page 16.

Chapter 4
Linux Kernel Debug Print Tool

This chapter describes how to work with the Linux Kernel Debug Print tool.

The Linux Kernel Debug Print tool consists of the following two components that work together to perform the debug print operation:

- Target server, which is responsible for collecting Kernel Ring Buffer log in unformatted way
- Host, which periodically requests 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 also clears the log and sends it to the clients using TCP/IP connection. It collects the redirected printf output from the user space applications.

  Debug Print dynamic library - Is responsible for redirection of the application's *prinf* 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 tool is independent of CodeWarrior and does not require a debug session.

The ARM binaries have been compiled with tool chain *gcc-linaro-arm-linux-gnuelfeabihf-4.8-2013.12_linux* and LS1 SDK version 1.1.

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.

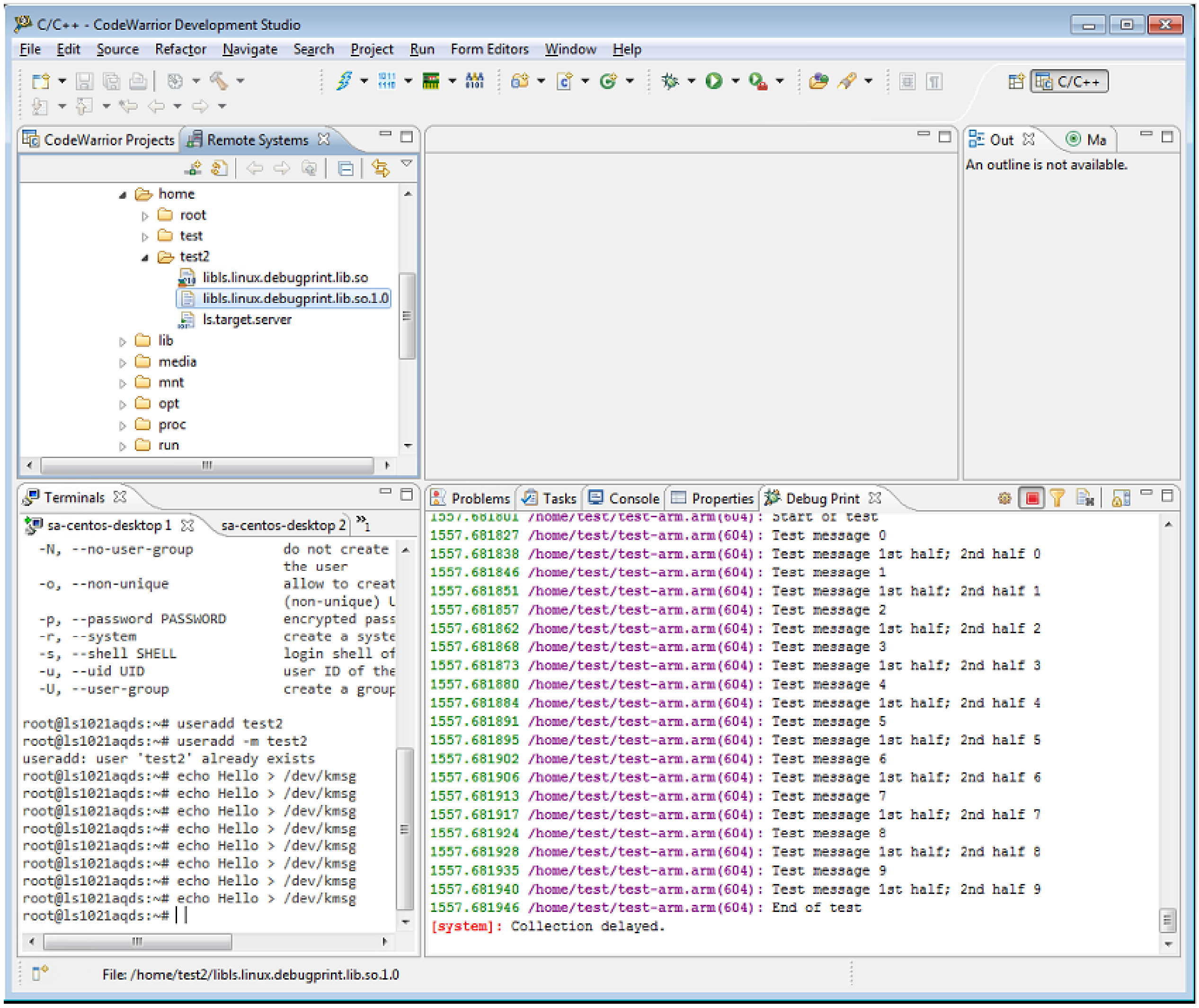
1. Deploy the Software Analysis target binaries on the target. For example, if you have the target root filesystem on NFS, you can copy *ls.target.server* and *libls.linux.debugprint.lib.so** to the host location *[NFS_PATH]/home/root*).
2. The debug print target server cross-compiled for ARM is located in CodeWarrior in directory: `[CWInstallDir]/CW_ARMv7/ARMv7/sa_ls/linux.armv7.debugprint/bin`, which needs to be copied on the target (for example, to the home directory). The server requires **sudo** access (default user on target is root) and requires a single argument; the port number on which clients will listen. If not specified, it will start on the default port 5000.

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

   ```
   # ssh root@target_ip_address
   ```

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

Figure 37: Configure Debug Print dialog


C/C++ - CodeWarrior Development Studio
File Edit Source Refactor Navigate Search Project Run Form Editors Window Help
C/C++
CodeWarrior Projects
Remote Systems
home
root
test
test2
libls.linux.debugprint.lib.so
libls.linux.debugprint.lib.so.1.0
ls.target.server
lib
media
mnt
opt
proc
run
Out
Ma
An outline is not available.
Terminals
sa-centos-desktop 1
sa-centos-desktop 2
Problems
Tasks
Console
Properties
Debug Print
File: /home/test2/libls.linux.debugprint.lib.so.1.0

```
  -N, --no-user-group           do not create
                                the user
  -o, --non-unique              allow to creat
                                (non-unique) U
  -p, --password PASSWORD       encrypted pass
  -r, --system                  create a syste
  -s, --shell SHELL             login shell of
  -u, --uid UID                 user ID of the
  -U, --user-group              create a group

root@ls1021aqds:~# useradd test2
root@ls1021aqds:~# useradd -m test2
useradd: user 'test2' already exists
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# echo Hello > /dev/kmsg
root@ls1021aqds:~# ||
```

```
1557.681801 /home/test/test-arm.arm(604): Start of test
1557.681827 /home/test/test-arm.arm(604): Test message 0
1557.681838 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 0
1557.681846 /home/test/test-arm.arm(604): Test message 1
1557.681851 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 1
1557.681857 /home/test/test-arm.arm(604): Test message 2
1557.681862 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 2
1557.681868 /home/test/test-arm.arm(604): Test message 3
1557.681873 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 3
1557.681880 /home/test/test-arm.arm(604): Test message 4
1557.681884 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 4
1557.681891 /home/test/test-arm.arm(604): Test message 5
1557.681895 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 5
1557.681902 /home/test/test-arm.arm(604): Test message 6
1557.681906 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 6
1557.681913 /home/test/test-arm.arm(604): Test message 7
1557.681917 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 7
1557.681924 /home/test/test-arm.arm(604): Test message 8
1557.681928 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 8
1557.681935 /home/test/test-arm.arm(604): Test message 9
1557.681940 /home/test/test-arm.arm(604): Test message 1st half; 2nd half 9
1557.681946 /home/test/test-arm.arm(604): End of test
[system]: Collection delayed.
```

3. The dynamic library cross-compiled for ARM is located in CodeWarrior directory at: *`[CWInstallDir]`*`/CW_ARMv7/ARMv7/sa_ls/linux.armv7.debugprint/lib`, which needs to be copied on the target. This library must be loaded by the Linux loader before the C runtime, when you are running the user space applications that need to be monitored by setting the environment variable, *LD_PRELOAD*.

   Start a new ssh console on the target and run a test application (in this case, test.arm) to see its original output:

   ```
   # ssh root@target_ip_address
   # ./test.arm
   ```

   Then preload the debug print library and run the test application again:

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

   You will notice next time that the test application will not output anything on the console. The 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 table below describes the icons displayed on the Debug Print viewer.

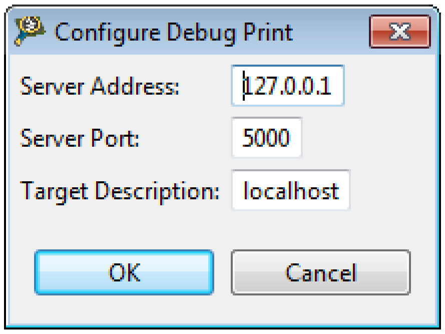
   Table 11: Debug Print viewer icons

Icons	Description
Clear All	Removes all text from the view.
Start/Stop	Two-state button used for starting and stopping the Debug Print monitor task.
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).

   To configure Debug Print server, click **Configure** icon on the toolbar. The **Configure Debug Print** dialog appears. You can specify the server address, port number at which the server will listen to client, and the target description.

   Figure 38: Configure Debug Print dialog



Configure Debug Print

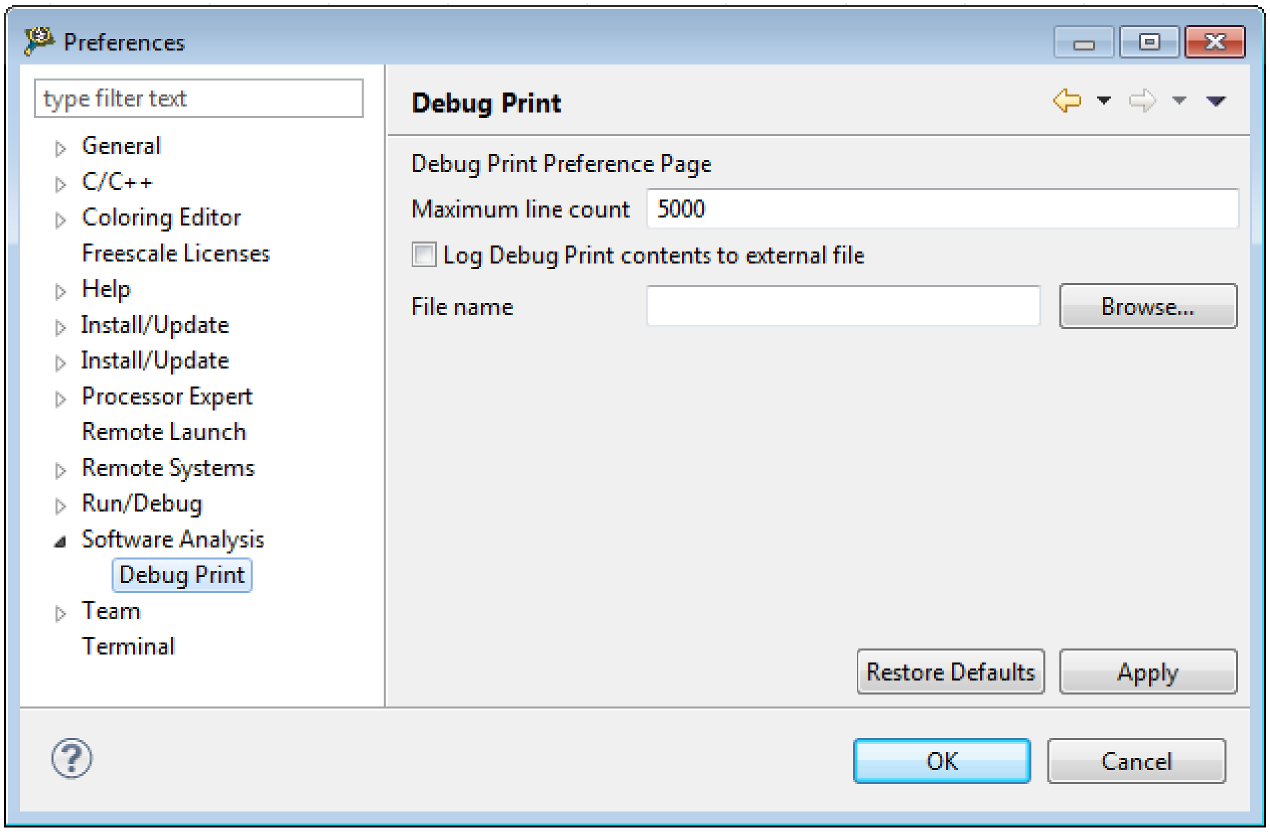
Server Address: 127.0.0.1

Server Port: 5000

Target Description: localhost

   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**.

Figure 39: Preferences dialog



Debug Print

Debug Print Preference Page

Maximum line count 5000

Log Debug Print contents to external file

File name

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

Table 12: Debug Print settings

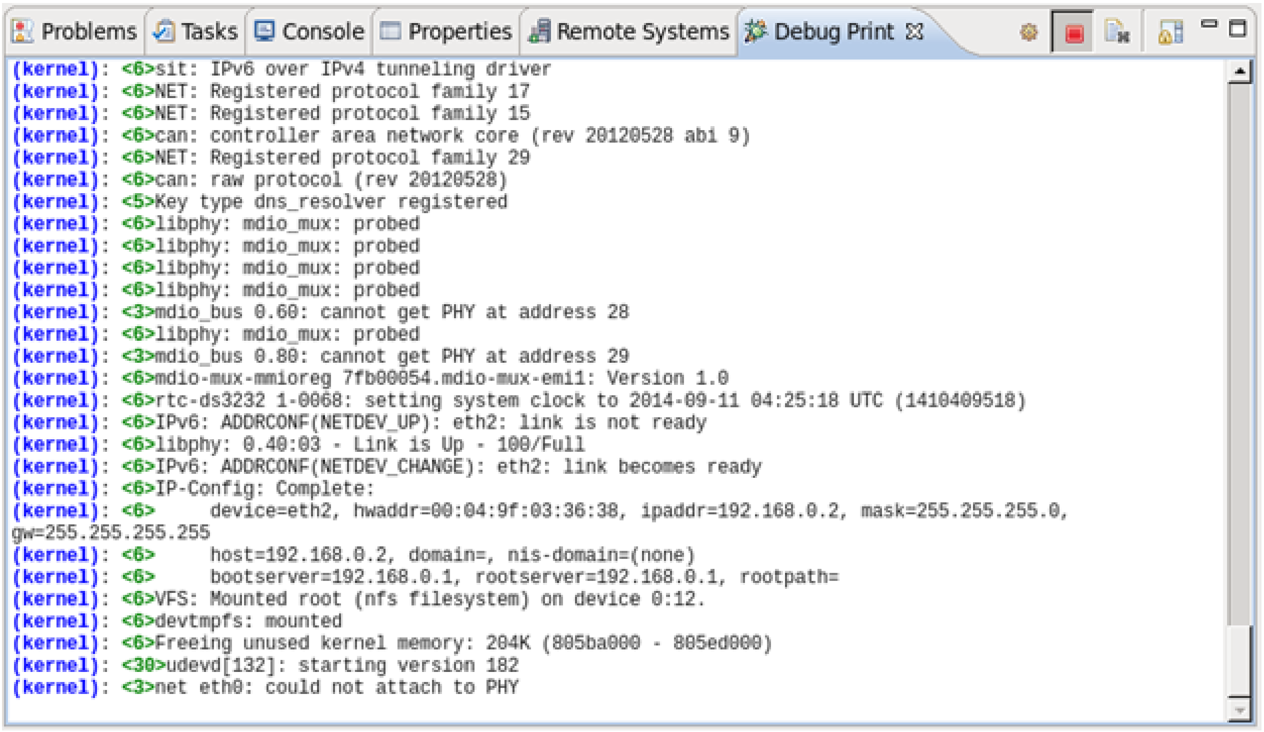
Options	Description
Maximum line count	Limits the number of lines the **Debug Print** view should display. If this limit is exceeded, the old messages are deleted.
Log Debug Print contents to external file	If selected, the messages will be appended to an external file besides displaying them into the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

NOTE

The ARM binaries are compiled with tool chain gcc-linaro-arm-linux-gnuelfeabihf-4.8-2013.12_linux and LS1 SDK version 1.1.

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

Figure 40: Debug Print view - messages from server



```
(kernel): <6>sit: IPv6 over IPv4 tunneling driver
(kernel): <6>NET: Registered protocol family 17
(kernel): <6>NET: Registered protocol family 15
(kernel): <6>can: controller area network core (rev 20120528 abi 9)
(kernel): <6>NET: Registered protocol family 29
(kernel): <6>can: raw protocol (rev 20120528)
(kernel): <5>Key type dns_resolver registered
(kernel): <6>libphy: mdio_mux: probed
(kernel): <6>libphy: mdio_mux: probed
(kernel): <6>libphy: mdio_mux: probed
(kernel): <6>libphy: mdio_mux: probed
(kernel): <3>mdio_bus 0.60: cannot get PHY at address 28
(kernel): <6>libphy: mdio_mux: probed
(kernel): <3>mdio_bus 0.80: cannot get PHY at address 29
(kernel): <6>mdio-mux-mmioreg 7fb00054.mdio-mux-emi1: Version 1.0
(kernel): <6>rtc-ds3232 1-0068: setting system clock to 2014-09-11 04:25:18 UTC (1410409518)
(kernel): <6>IPv6: ADDRCONF(NETDEV_UP): eth2: link is not ready
(kernel): <6>libphy: 0.40:03 - Link is Up - 100/Full
(kernel): <6>IPv6: ADDRCONF(NETDEV_CHANGE): eth2: link becomes ready
(kernel): <6>IP-Config: Complete:
(kernel): <6>     device=eth2, hwaddr=00:04:9f:03:36:38, ipaddr=192.168.0.2, mask=255.255.255.0,
gw=255.255.255.255
(kernel): <6>     host=192.168.0.2, domain=, nis-domain=(none)
(kernel): <6>     bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
(kernel): <6>VFS: Mounted root (nfs filesystem) on device 0:12.
(kernel): <6>devtmpfs: mounted
(kernel): <6>Freeing unused kernel memory: 204K (805ba000 - 805ed000)
(kernel): <30>udevd[132]: starting version 182
(kernel): <3>net eth0: could not attach to PHY
```

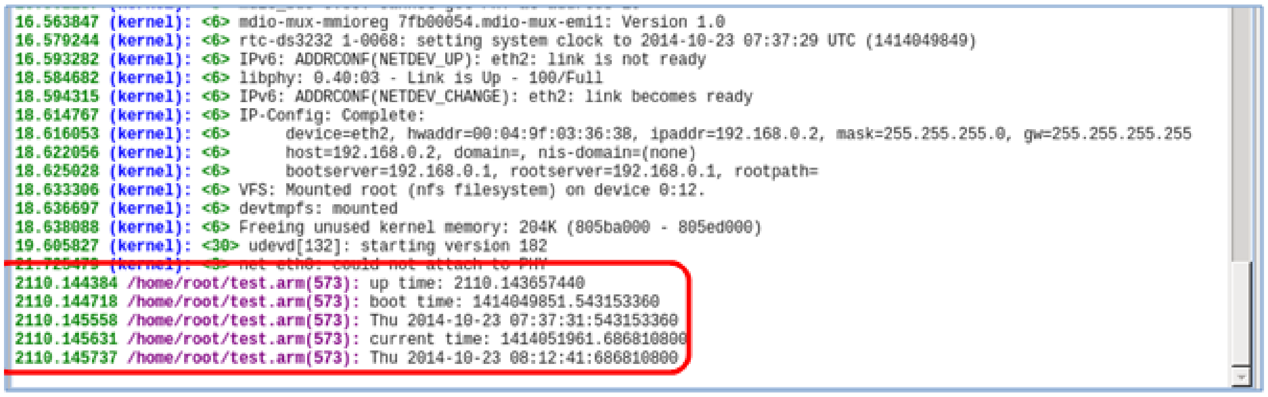
7. Open another console on the target in the same directory, preload the debug print library and run the test application:

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

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

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

Figure 41: Debug Print view - application messages



```
16.563847 (kernel): <6> mdio-mux-mmioreg 7fb00054.mdio-mux-emi1: Version 1.0
16.579244 (kernel): <6> rtc-ds3232 1-0068: setting system clock to 2014-10-23 07:37:29 UTC (1414049849)
16.593282 (kernel): <6> IPv6: ADDRCONF(NETDEV_UP): eth2: link is not ready
18.584682 (kernel): <6> libphy: 0.40:03 - Link is Up - 100/Full
18.594315 (kernel): <6> IPv6: ADDRCONF(NETDEV_CHANGE): eth2: link becomes ready
18.614767 (kernel): <6> IP-Config: Complete:
18.616053 (kernel): <6>      device=eth2, hwaddr=00:04:9f:03:36:38, ipaddr=192.168.0.2, mask=255.255.255.0, gw=255.255.255.255
18.622056 (kernel): <6>      host=192.168.0.2, domain=, nis-domain=(none)
18.625028 (kernel): <6>      bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
18.633306 (kernel): <6> VFS: Mounted root (nfs filesystem) on device 0:12.
18.636697 (kernel): <6> devtmpfs: mounted
18.638088 (kernel): <6> Freeing unused kernel memory: 204K (805ba000 - 805ed000)
19.605827 (kernel): <30> udevd[132]: starting version 182
21.725479 (kernel): <3> net eth0: could not attach to PHY
2110.144384 /home/root/test.arm(573): up time: 2110.143657440
2110.144718 /home/root/test.arm(573): boot time: 1414049851.543153360
2110.145558 /home/root/test.arm(573): Thu 2014-10-23 07:37:31:543153360
2110.145631 /home/root/test.arm(573): current time: 1414051961.686810800
2110.145737 /home/root/test.arm(573): Thu 2014-10-23 08:12:41:686810800
```

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:

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

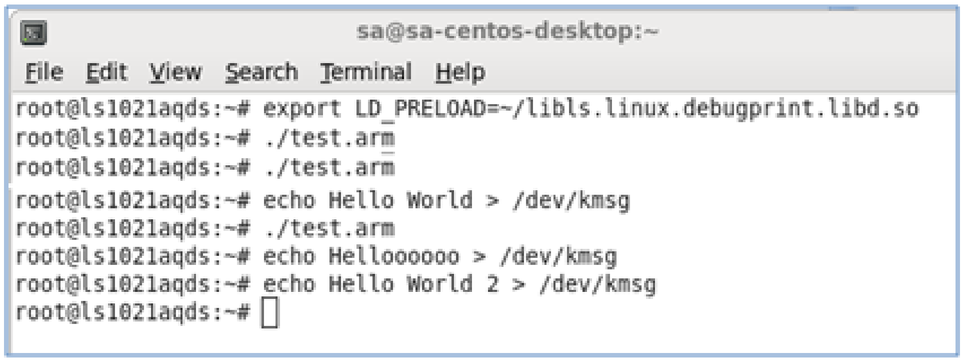
   ```
   # echo Hello World > /dev/kmsg
   ```

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

```
# echo Hellooooooo > /dev/kmsg

# echo Hello World 2 > /dev/kmsg
```

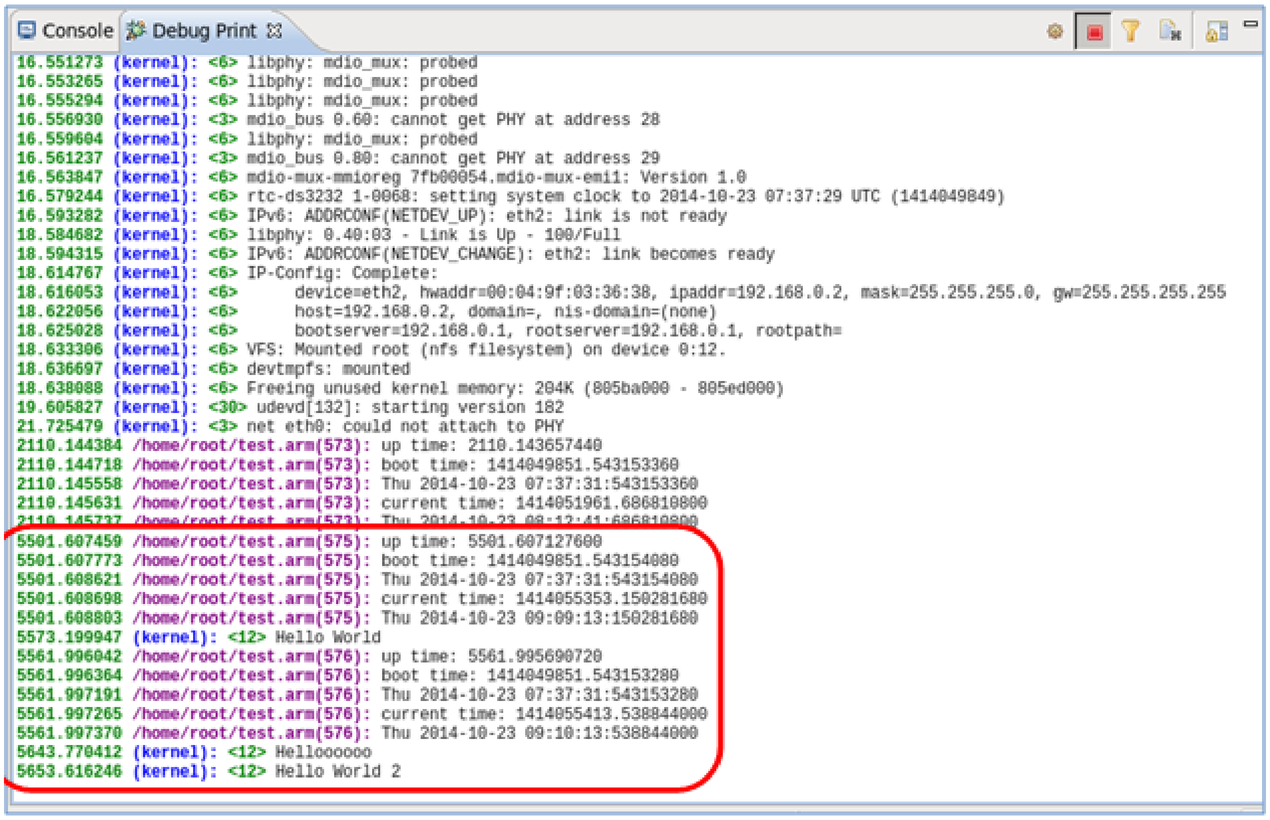
Figure 42: Target console - sending messages



```
root@ls1021aqds:~# export LD_PRELOAD=~/libls.linux.debugprint.libd.so
root@ls1021aqds:~# ./test.arm
root@ls1021aqds:~# ./test.arm
root@ls1021aqds:~# echo Hello World > /dev/kmsg
root@ls1021aqds:~# ./test.arm
root@ls1021aqds:~# echo Hellooooooo > /dev/kmsg
root@ls1021aqds:~# echo Hello World 2 > /dev/kmsg
root@ls1021aqds:~# 
```

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

Figure 43: Debug Print view - messages from server



```
16.551273 (kernel): <6> libphy: mdio_mux: probed
16.553265 (kernel): <6> libphy: mdio_mux: probed
16.555294 (kernel): <6> libphy: mdio_mux: probed
16.556930 (kernel): <3> mdio_bus 0.60: cannot get PHY at address 28
16.559604 (kernel): <6> libphy: mdio_mux: probed
16.561237 (kernel): <3> mdio_bus 0.80: cannot get PHY at address 29
16.563847 (kernel): <6> mdio-mux-mmioreg 7fb00054.mdio-mux-emi1: Version 1.0
16.579244 (kernel): <6> rtc-ds3232 1-0068: setting system clock to 2014-10-23 07:37:29 UTC (1414049849)
16.593282 (kernel): <6> IPv6: ADDRCONF(NETDEV_UP): eth2: link is not ready
18.584682 (kernel): <6> libphy: 0.40:03 - Link is Up - 100/Full
18.594315 (kernel): <6> IPv6: ADDRCONF(NETDEV_CHANGE): eth2: link becomes ready
18.614767 (kernel): <6> IP-Config: Complete:
18.616053 (kernel): <6>      device=eth2, hwaddr=00:04:9f:03:36:38, ipaddr=192.168.0.2, mask=255.255.255.0, gw=255.255.255.255
18.622056 (kernel): <6>      host=192.168.0.2, domain=, nis-domain=(none)
18.625028 (kernel): <6>      bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
18.633306 (kernel): <6> VFS: Mounted root (nfs filesystem) on device 0:12.
18.636697 (kernel): <6> devtmpfs: mounted
18.638088 (kernel): <6> Freeing unused kernel memory: 204K (805ba000 - 805ed000)
19.605827 (kernel): <30> udevd[132]: starting version 182
21.725479 (kernel): <3> net eth0: could not attach to PHY
2110.144384 /home/root/test.arm(573): up time: 2110.143657440
2110.144718 /home/root/test.arm(573): boot time: 1414049851.543153360
2110.145558 /home/root/test.arm(573): Thu 2014-10-23 07:37:31:543153360
2110.145631 /home/root/test.arm(573): current time: 1414051961.686810000
2110.145737 /home/root/test.arm(573): Thu 2014-10-23 08:12:41:686810000
5501.607459 /home/root/test.arm(575): up time: 5501.607127600
5501.607773 /home/root/test.arm(575): boot time: 1414049851.543154080
5501.608621 /home/root/test.arm(575): Thu 2014-10-23 07:37:31:543154080
5501.608698 /home/root/test.arm(575): current time: 1414055353.150281680
5501.608803 /home/root/test.arm(575): Thu 2014-10-23 09:09:13:150281680
5573.199947 (kernel): <12> Hello World
5561.996042 /home/root/test.arm(576): up time: 5561.995690720
5561.996364 /home/root/test.arm(576): boot time: 1414049851.543153280
5561.997191 /home/root/test.arm(576): Thu 2014-10-23 07:37:31:543153280
5561.997265 /home/root/test.arm(576): current time: 1414055413.538844000
5561.997370 /home/root/test.arm(576): Thu 2014-10-23 09:10:13:538844000
5643.770412 (kernel): <12> Hellooooooo
5653.616246 (kernel): <12> Hello World 2
```

This chapter contains the following section:

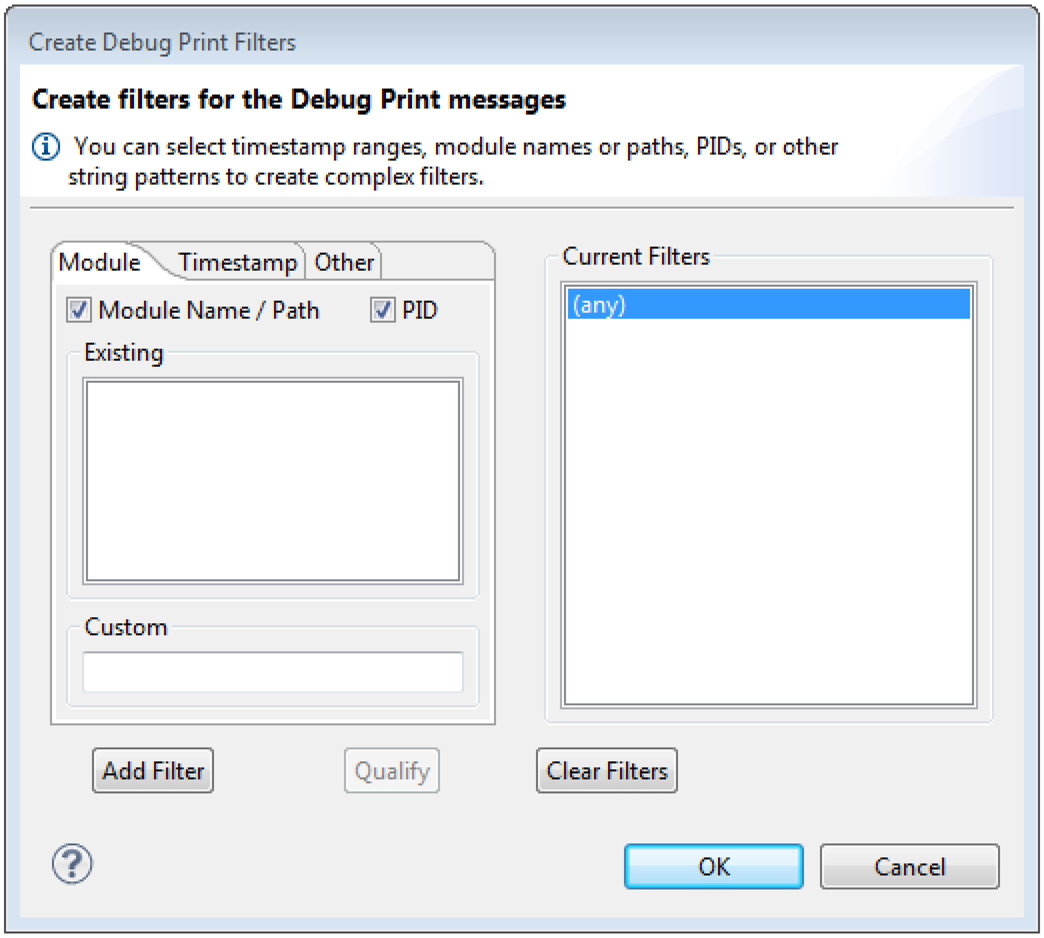
- Filtering debug print messages on page 51

4.1 Filtering debug print messages

The Linux Kernel Debug Print tool has a powerful filtering engine that allows you to see the desired information.

The filtering engine allows you to display 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.

Figure 44: Create Debug Print Filters dialog


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.

This dialog has three tabs:

- **Modulename/Path** 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** tol add the filter in the **Current Filters** list. These filters can be qualified with a timestamp range or a string pattern.

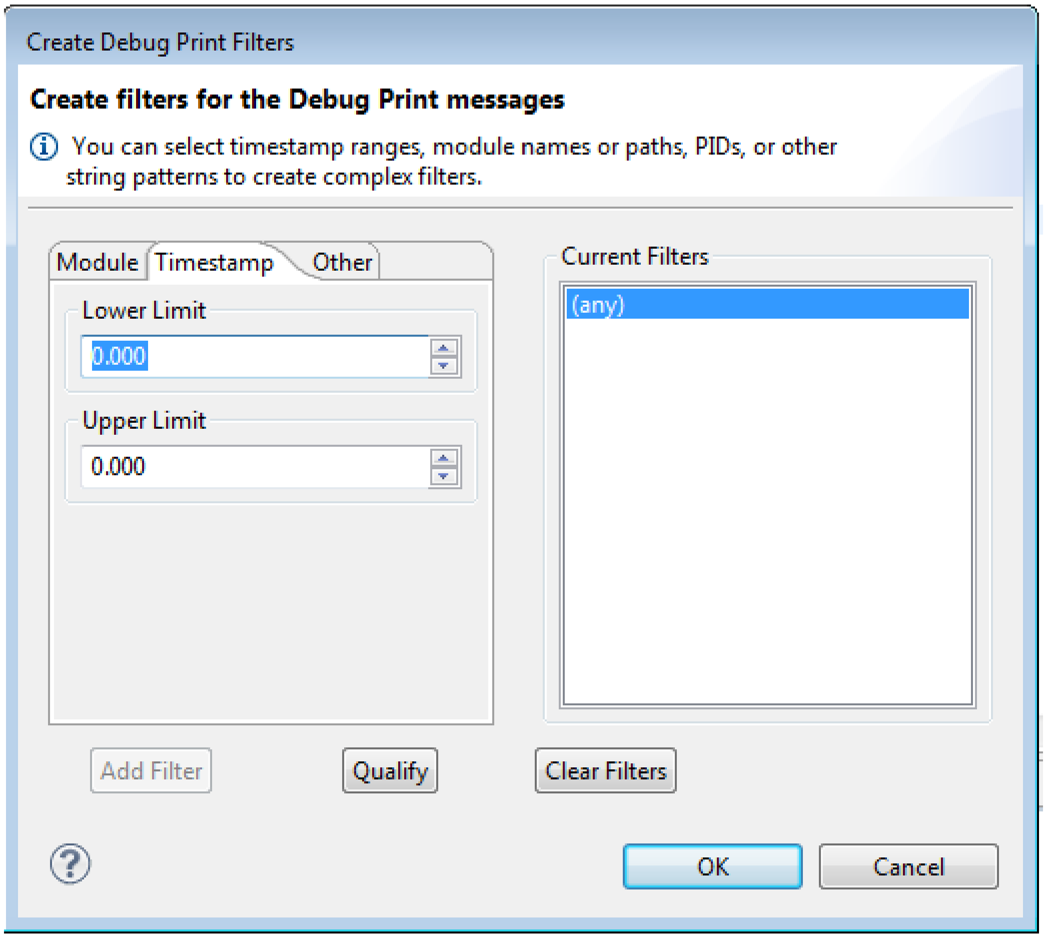
  The **Existing** list contains all the module names/application paths/PIDs from the messages already displayed in the **Debug Print** view. When you want to filter messages from a certain module or application that is not started or did not print any messages yet, you can manually enter the module name/path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

- **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.

Figure 45: Create Debug Print Filters dialog - Timestamp tab

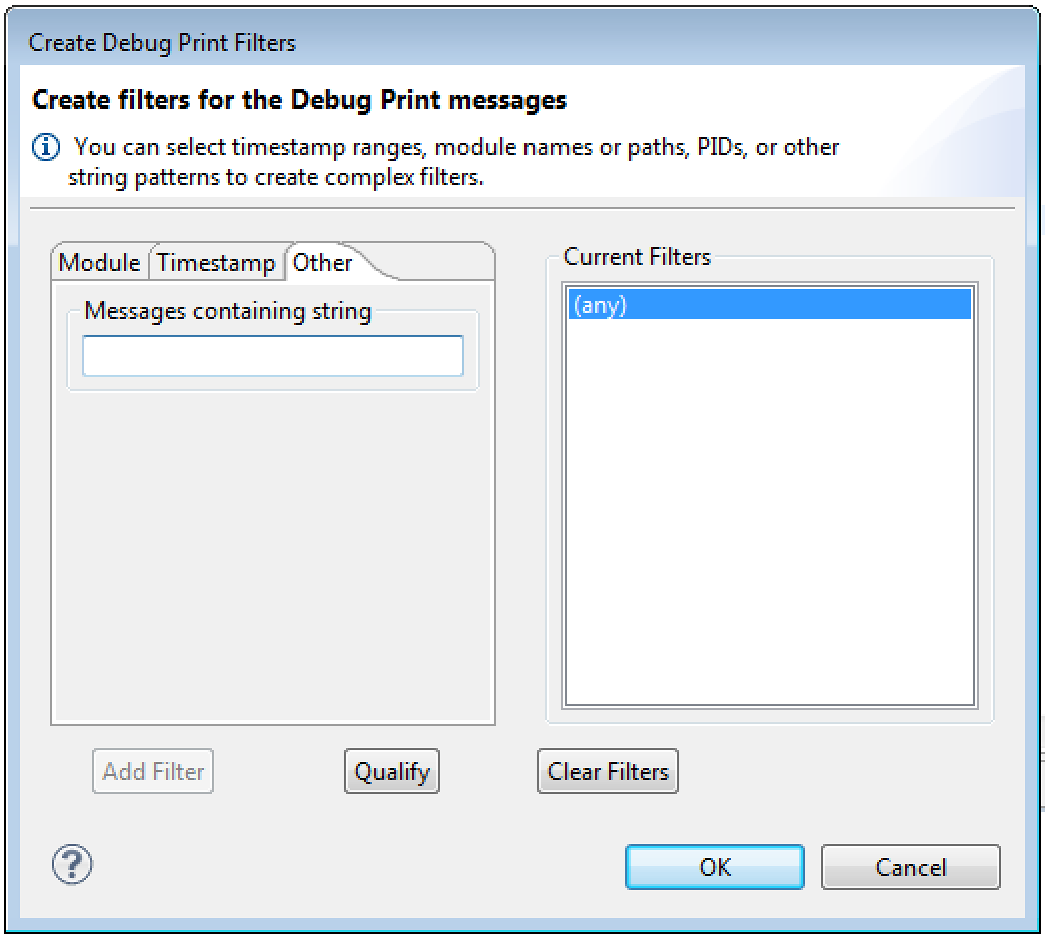

Create Debug Print Filters

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.

- **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.

Figure 46: Create Debug Print Filters dialog - Other tab


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
Current Filters
(any)
Add Filter
Qualify
Clear Filters
OK
Cancel

How To Reach Us

Home Page:

freescale.com

Web Support:

freescale.com/support

CW_ARMv7_Tracing_User_Guide
Rev. 10.0.8
01/2016