Lattice Propel 2024.2 SDK

User Guide

FPGA-UG-02218-1.0

November 2024

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
ASCII	American Standard Code for Information Interchange.
BSP	Board Support Package. The layer of software containing hardware-specific drivers and libraries to function in a particular hardware environment.
CXU	Composable Extension Unit.
CDT	C/C++ Development Tools.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
DUT	Design Under Test.
GUI	Graphical User Interface.
FPGA	Field Programmable Gate Array.
FreeRTOS	A market-leading RTOS for microcontrollers and small microprocessors.
HDL	Hardware description language.
IBIS	Input Output Buffer Information System.
IDE	Integrated Development Environment.
IP	Intellectual Property.
JEDEC	Joint Electron Device Engineering Council.
OCD	On-Chip-Debugging.
OEM	Original Equipment Manufacturer.
OpenOCD	Open On-Chip Debugger.
QEMU	A generic and open source machine emulator and virtualizer.
RISC-V	Reduced Instruction Set Computer-V. A free and open instruction set architecture (ISA) enabling a new era of processor innovation through open standard collaboration.
RISC-V MC	Lattice RISC-V for Micro-Controller Soft IP.
RISC-V SM	Lattice RISC-V for State-Machine Soft IP.
RISC-V RX	Lattice RISC-V for RTOS Soft IP.
RTOS	Real Time Operating System
RX	RISC-V for RTOS applications
SDK	Software Development Kit. A set of software development tools that allows the creation of applications for software package on the Lattice embedded platform.
SHA	Secure Hash Algorithm.
SoC	System-on-Chip. An integrated circuit that integrates all components of a computer or other electronic systems.
SRAM	Static Random Access Memory.
UART	Universal Asynchronous Receiver/Transmitter.
UFM	User Flash Memory.
UI	User Interface.
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language.

1. Introduction

Lattice Propel™ design environment is a complete set of graphical and command-line tools to create, analyze, compile, and debug both FPGA-based hardware and software processor systems.

1.1. Purpose

Embedded system solutions play an important role in FPGA system design, allowing you to develop software for a processor in an FPGA device. It provides the flexibility for you to control various peripherals from a system bus.

To develop an embedded system on an FPGA, you need to design the System-on-Chip (SoC) with an embedded processor and develop system software on the processor. Lattice Propel helps you develop your system with a RISC-V processor, peripheral IP, and a set of tools.

The purpose of this document is to introduce Lattice Propel SDK tool and flow to help you quickly get started to build a small demo system. You can find recommended flows of using Lattice Propel SDK in this document as well.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice FPGA devices. The complete list of supported devices can be found in Lattice Propel Release Notes. The technical guidelines assume readers have expertise in the embedded system area and FPGA technologies.

2. Lattice Propel Development Suite

Lattice Propel development suite includes:

- an integrated development environment (IDE), which is the framework of the Lattice Propel design suite;
- Lattice Propel Builder, which is for SoC design;
- Lattice Propel SDK, which is for system software development.

2.1. Eclipse IDE

Eclipse IDE provides the Lattice Propel development suite a platform to manage the SoC project and the Embedded C/C++ Project in the same workspace.

The SoC project, which extends from the Lattice Propel Builder project, provides easy interaction with other Lattice design tools, such as the Lattice Diamond™ software within Lattice Propel design environment.

The Embedded C/C++ Project provides a platform for developing or debugging application code within Eclipse IDE. The project can be created directly from the SoC project with a pre-set Board Support Package (BSP) and applications by using the Lattice Propel development suite.

2.2. Lattice Propel Builder

Lattice Propel Builder allows you to assemble the larger functional blocks of the design hierarchy. Lattice Propel Builder enables you to instantiate modules and IP from the IP Catalog in a schematic view, and can easily connect the modules. Lattice Propel Builder also helps you customize address spaces within modules, such as a processor. In the Lattice Propel development suite, Lattice Propel Builder is used to create a microprocessor integrated platform for both hardware and software development.

Refer to Lattice Propel 2024.2 Builder User Guide (FPGA-UG-02219) for more detailed information.

2.3. Lattice Propel SDK

Lattice Propel SDK is based on Eclipse Embedded C/C++ Development Tools (CDT). It allows you to create, build, and debug software application projects that drive the platform within the Eclipse framework.

The main features are:

- Create, build, debug, or manage embedded applications for Lattice RISC-V CPU/SoC solution.
- Provide extra build steps to generate the binary and memory files required for deployment.
- Build using the latest industry standard open source components and tools for RISC-V firmware development and debugging.
- Support Picolibc for RISC-V, and provide lightweight standard library implementation.
- Provide fully-configurable toolchain definitions.

3. Lattice Propel Tool Flows

The Lattice Propel tool flows including SoC project design flow, C/C++ project design flow, system simulation flow, and programming and On-Chip-Debugging (OCD) flow, are discussed in detail in the following sections.

3.1. Lattice Propel Environment

3.1.1. Running Lattice Propel

After installing the Lattice Propel software, you can launch Lattice Propel SDK from the desktop shortcut icon or from the Windows Start menu. When Lattice Propel SDK is invoked, a dialog (Figure 3.1) pops up. You can browse to select where to locate the workspace. For normal needs, simply click **Launch** to pick the default location and continue running Lattice Propel SDK.

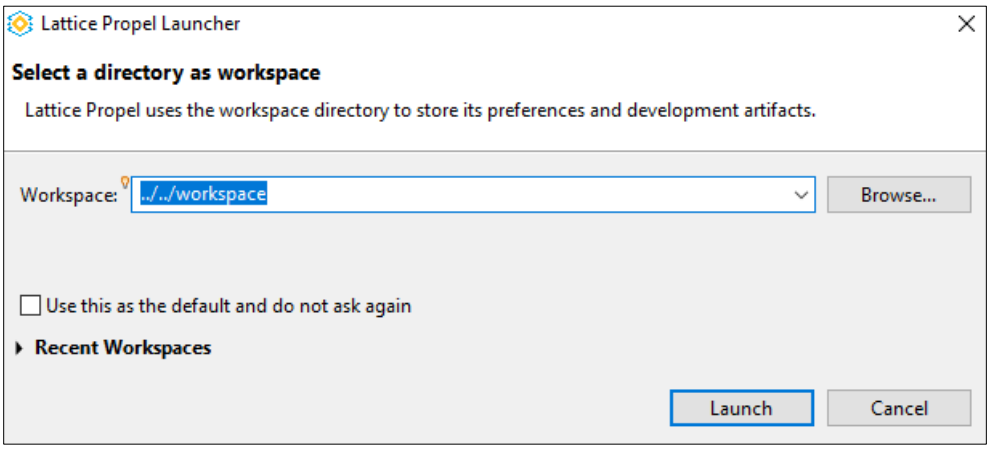


Figure 3.1. Select Workspace Dialog

After the workspace location is chosen, a single workbench window is displayed using default Lattice Propel SDK perspective. The default Lattice Propel SDK perspective contains the following five functional areas (Figure 3.2).

Note: A perspective is a group of views and editors in the Workbench window. A workspace is the directory where stores your work and it is used as the default content area for your projects as well as for holding any required metadata. A workbench is the desktop development environment in Eclipse IDE platform.

1. Menu bar and Toolbar, including: File menu, Edit menu, Source menu, Refactor menu, Navigate menu, Search menu, Project menu, Run menu, LatticeTools menu, Window menu, and Help menu.
2. Project Explorer view: displays projects in the workspace.
3. Editor view: provides capability of editing source files.
4. Outline view: displays an outline of a file that is currently open in the editor area.
5. Log area includes these views: Problem view, Tasks view, Console view, Properties view, and Terminal view.

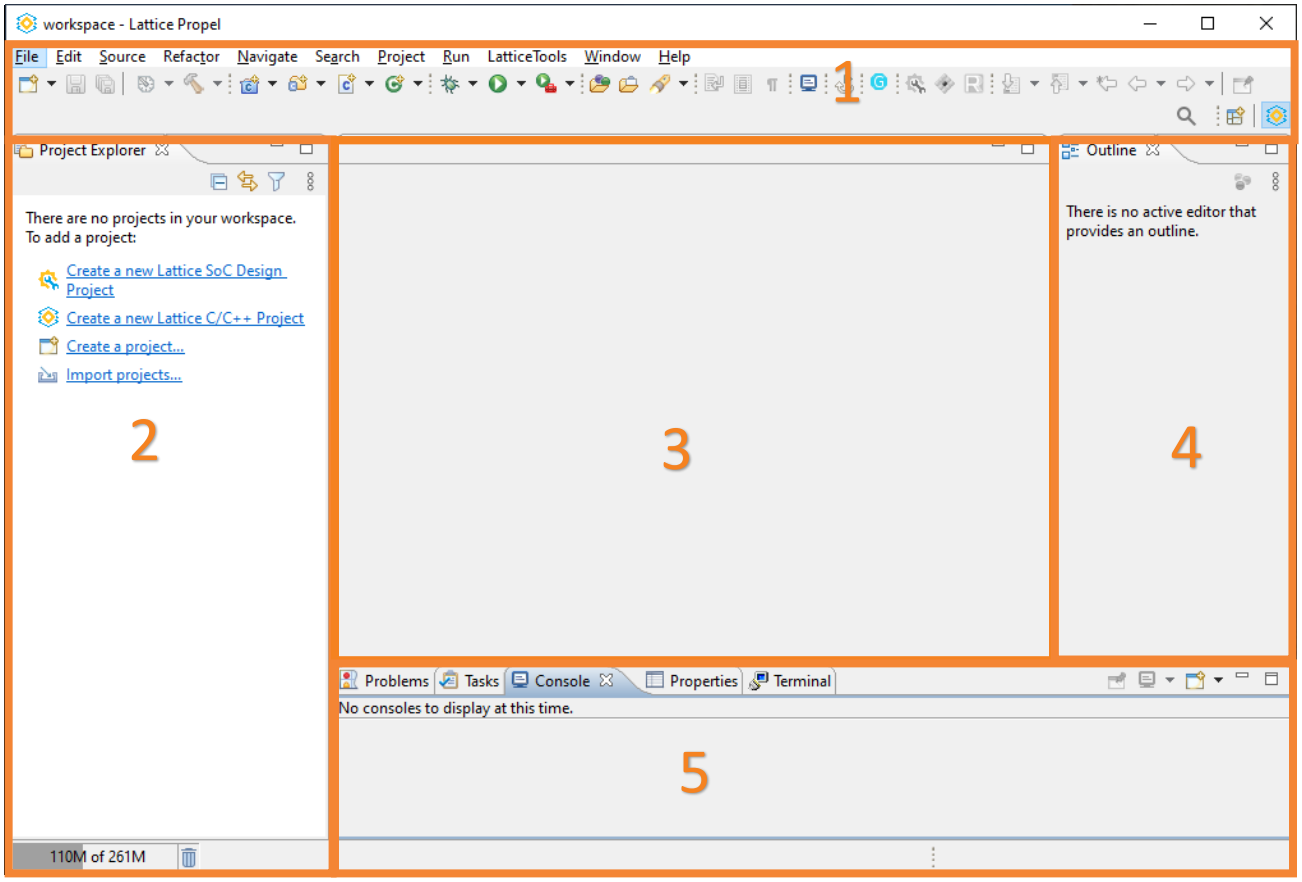


Figure 3.2. Lattice Propel Workbench Window

3.1.2. Importing Lattice SoC Design Projects

In Lattice Propel SDK, you can use the Import Wizard to import Lattice SoC design projects into workspace. Existing SoC design projects created by either Lattice Propel SDK or Lattice Propel Builder can also be imported into Workspace by choosing **Lattice Propel > Lattice SoC Design Projects**.

1. From Lattice Propel SDK, choose **File** > **Import…**.

   The **Select** wizard opens (Figure 3.3).

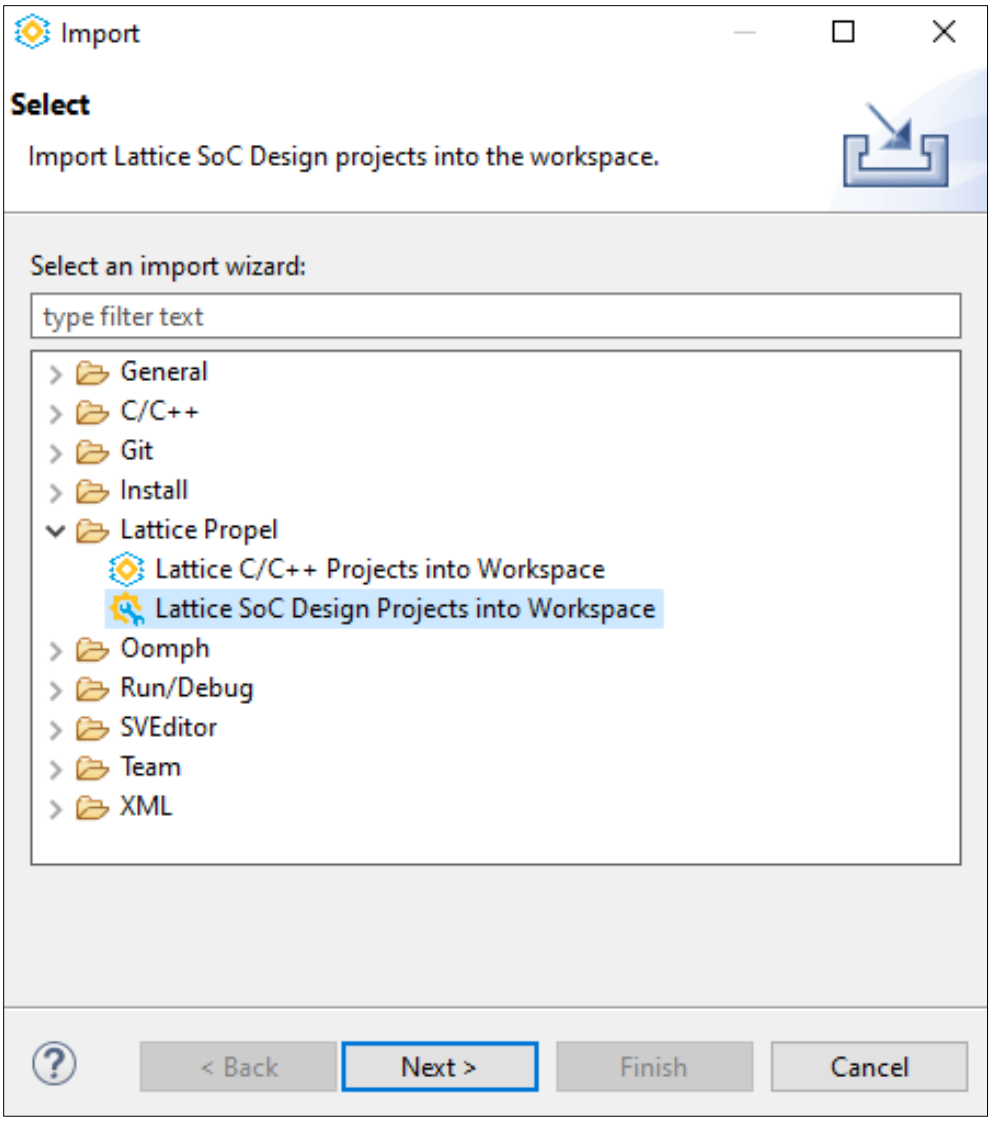


Figure 3.3. Select Wizard – Import Lattice SoC Design Projects

2. Select **Lattice Propel > Lattice SoC Design Projects into Workspace**. Click **Next**.
   The **Select** wizard switches to **Import Lattice SoC Design Projects** wizard page (Figure 3.4).

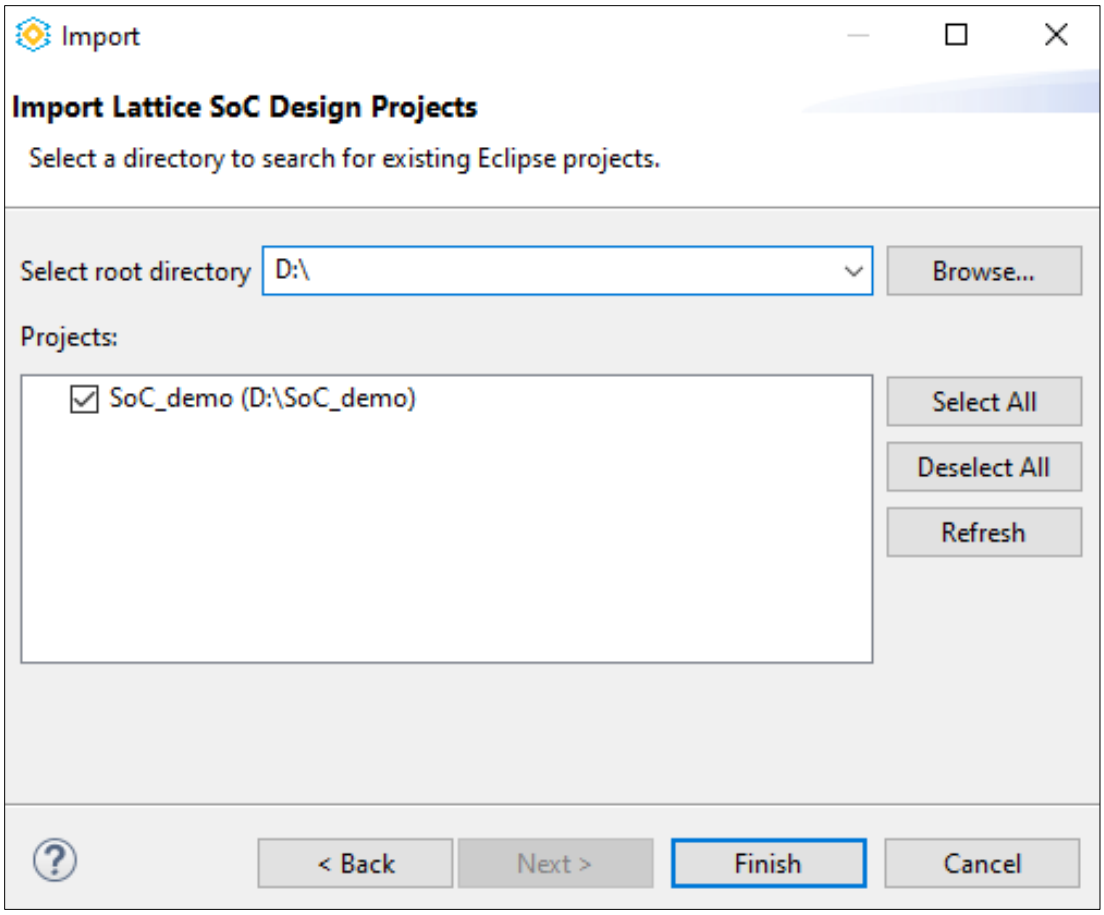


Figure 3.4. Import Lattice SoC Design Projects Wizard

3. Locate the directory containing the projects by clicking the **Browse** button.
4. In **Projects** area, select the SoC design project or projects you want to import.
5. Click **Finish** to start the importing process.

3.1.3. Importing Lattice C/C++ Projects

In Lattice Propel SDK, you can use the Import Wizard to import existing Lattice C/C++ projects created by Lattice Propel SDK 2023.2 or later into workspace by choosing **Lattice Propel > Lattice C/C++ Projects**.

1. From Lattice Propel SDK, choose **File** > **Import…**.

   The **Select** wizard opens (Figure 3.5).

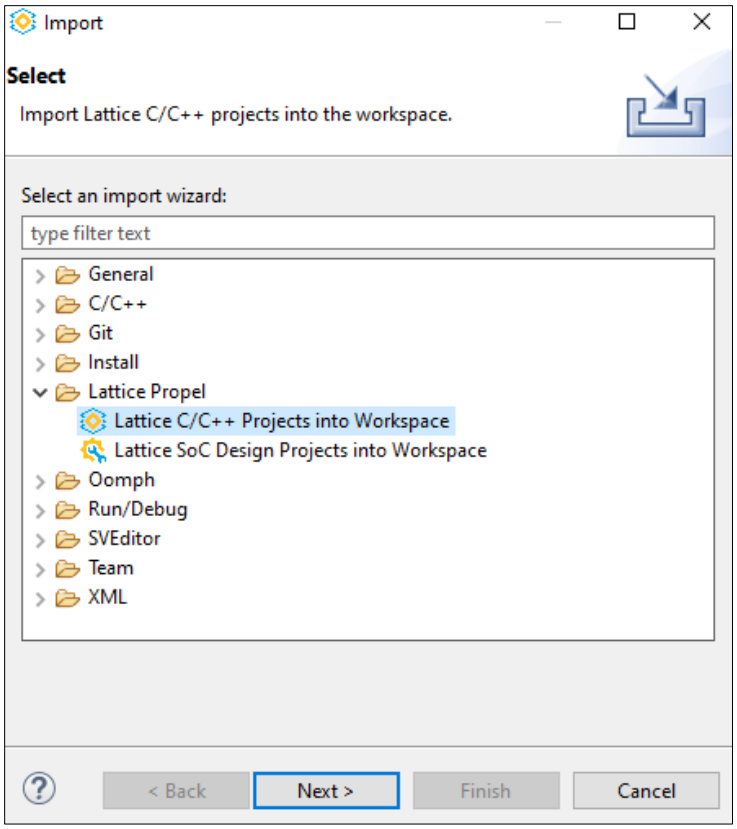


Figure 3.5. Select Wizard – Import Lattice C/C++ Projects

2. Select **Lattice Propel > Lattice C/C++ Projects into Workspace**. Click **Next**.
   The **Select** wizard switches to the **Import Lattice C/C++ Projects** wizard page (Figure 3.6).

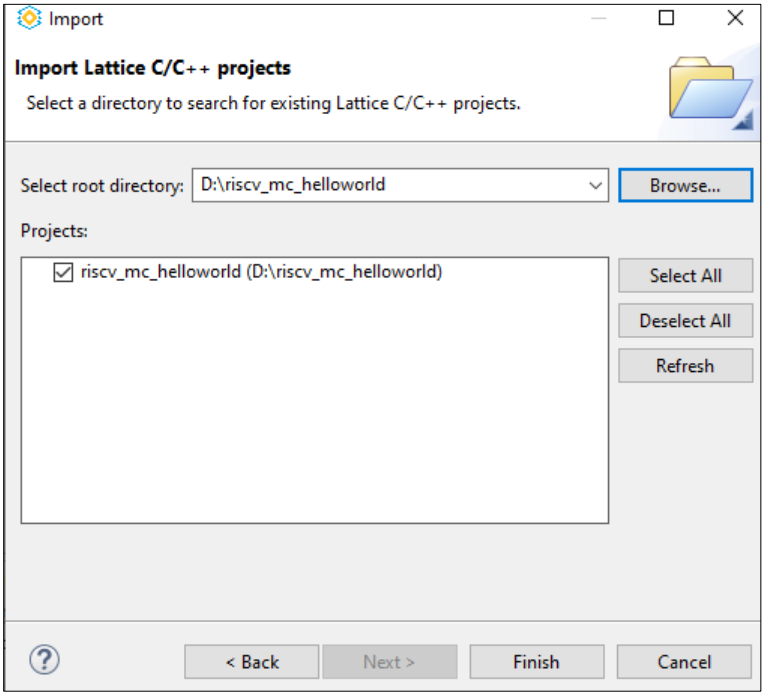


Figure 3.6. Import Lattice C/C++ Projects Wizard

3.1.4. Create Customized C/C++ Template

In Lattice Propel SDK, you can select a Lattice C/C++ project in the current workspace to create a user application template for creating new Lattice C/C++ project.

1. From Lattice Propel SDK, choose **Project** > **Create Lattice Application Template** or right click on a project and select **Create Lattice Application Template**. The **Create Application Template** wizard opens (Figure 3.7).

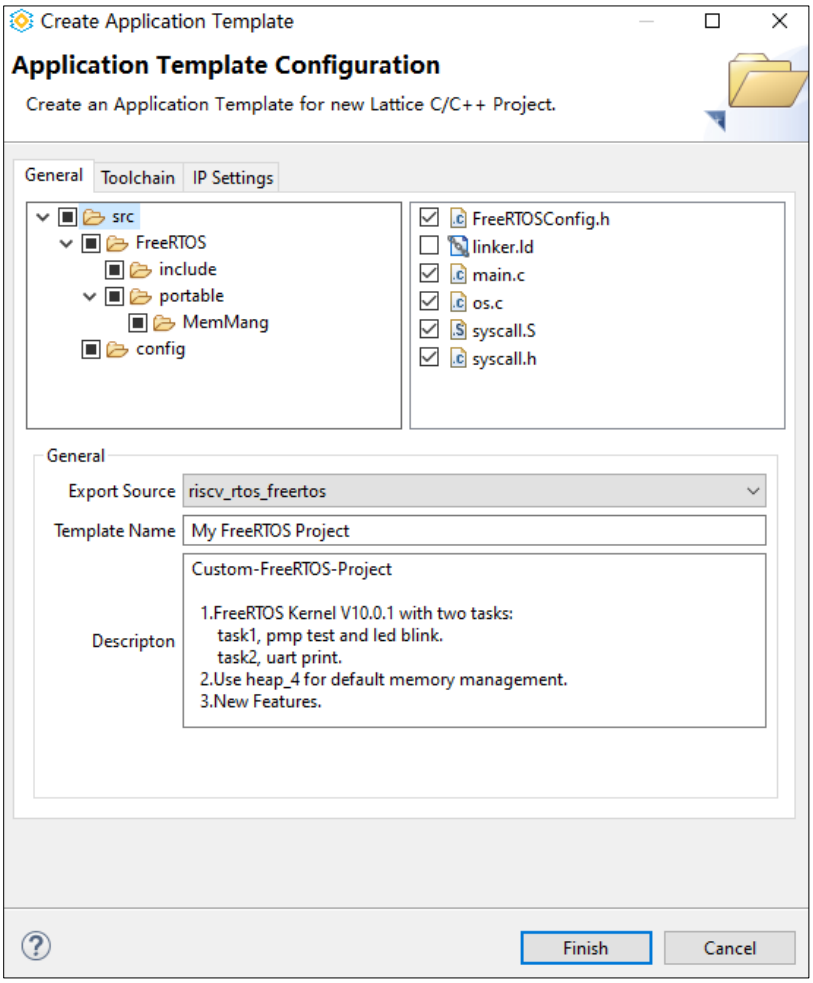


Figure 3.7. Create Application Template

2. You can input Template name, Description, Toolchain Configurations, IP Settings etc., and select files for creating template.
3. Click **Finish**.

3.1.5. Export and Deploy Customized C/C++ Template

In Lattice Propel SDK, you can export customized C/C++ templates into a single ZIP archive that is suitable for deploy to other Lattice Propel SDK.

1. From Lattice Propel SDK, choose **File** > **Export…**.

   The **Select** wizard opens (Figure 3.8).

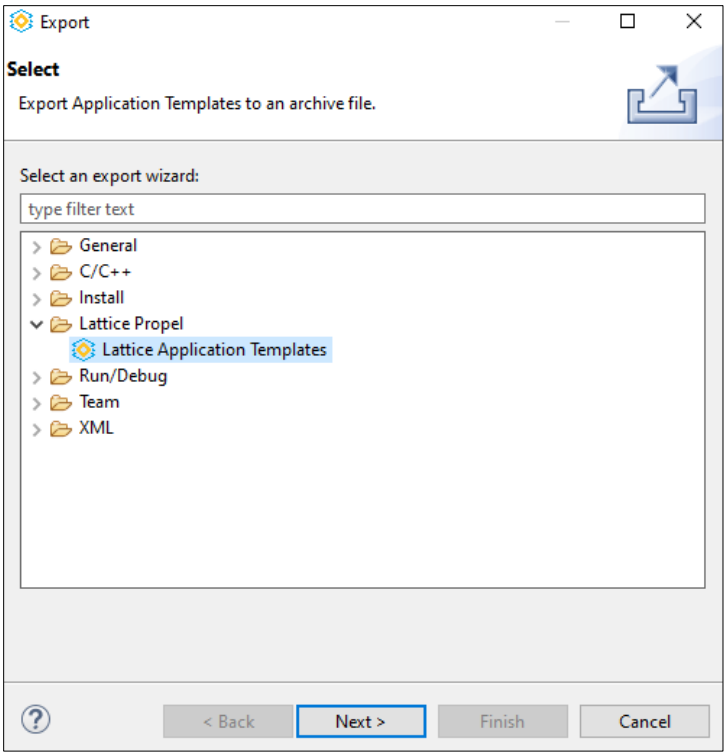


Figure 3.8. Select Wizard for Lattice Application Templates

2. Select **Lattice Propel > Lattice Application Templates**. Click **Next**.
   The **Select** wizard switches to the **Export Lattice Application Templates** wizard page (Figure 3.9).

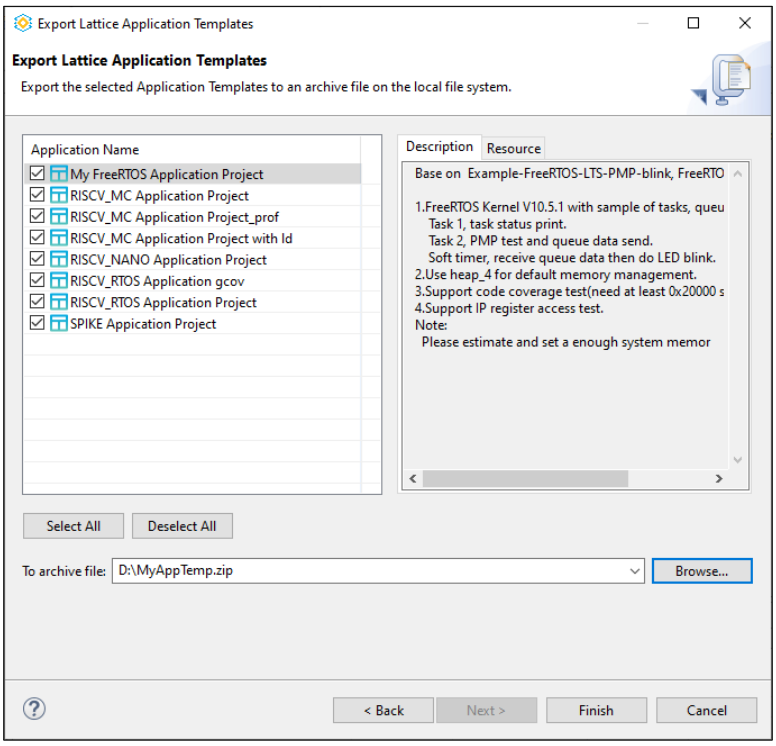


Figure 3.9. Export Lattice Application Templates Wizard

3. You can locate the destination directory by clicking the **Browse** button.
4. In the **Application Name** area, select the application templates you want to export.
5. Click **Finish** to start the exporting process.
6. Copy the exported zip archive to other SDK environments.
7. Extract the zip archive to the **Application Templates Install Path** (Figure 3.10).

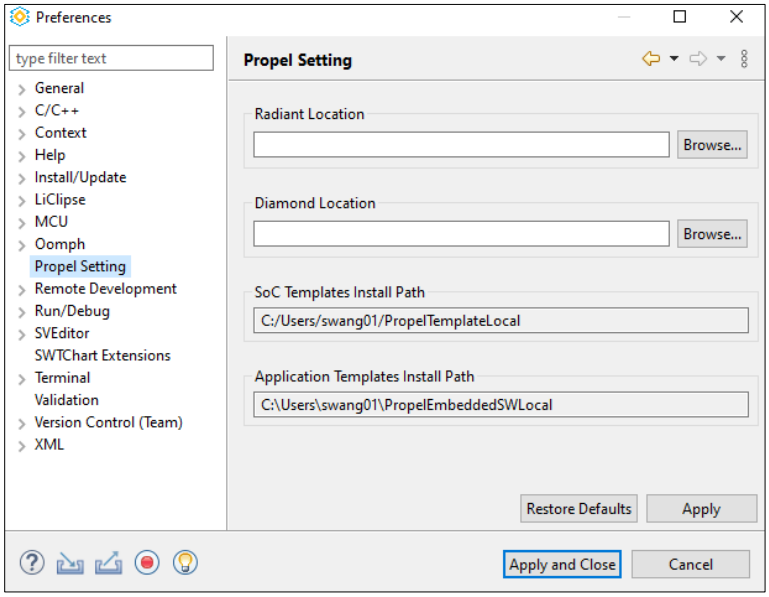


Figure 3.10. Lattice Propel Setting Page

3.2. SoC Project Design Flow

A new SoC design project including a Lattice Propel Builder design can be started from the Lattice Propel sets. Follow steps below to create a new SoC design project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

3.2.1. Creating an SoC Design Project

To start a Lattice SoC design project from Lattice Propel SDK:

1. In Lattice Propel SDK, choose **File** > **New** > **Lattice SoC Design Project**.

   The **Create SoC Project** wizard opens (Figure 3.11). In the **Create SoC Project** wizard, you can specify a device or a board for a Template SoC project.

   - To specify a device for your new Template SoC project, use the drop-down menu to select the desired device information, including **Processor**, **Family**, **Device**, **Package**, **Speed**, and **Condition**. Also, select RISC-V SoC Project or Empty Project in the **Template Design** field (Figure 3.11).

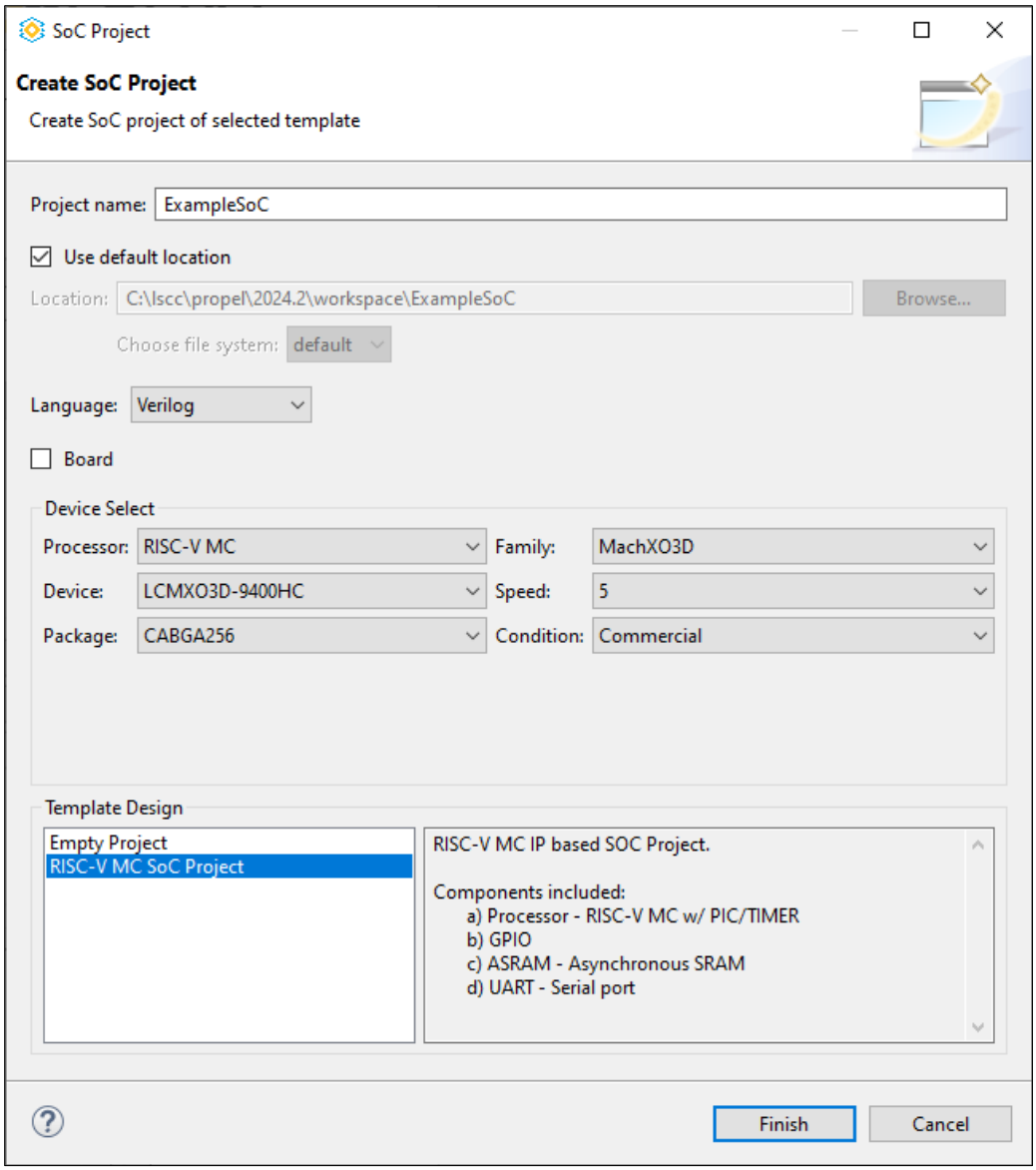


Figure 3.11. Specify a Device for Template SoC Project

- Or, to specify a board for a new Template SoC project, check the **Board** checkbox (Figure 3.12).

  Note: You can choose VHDL/Verilog in the **Language** field.

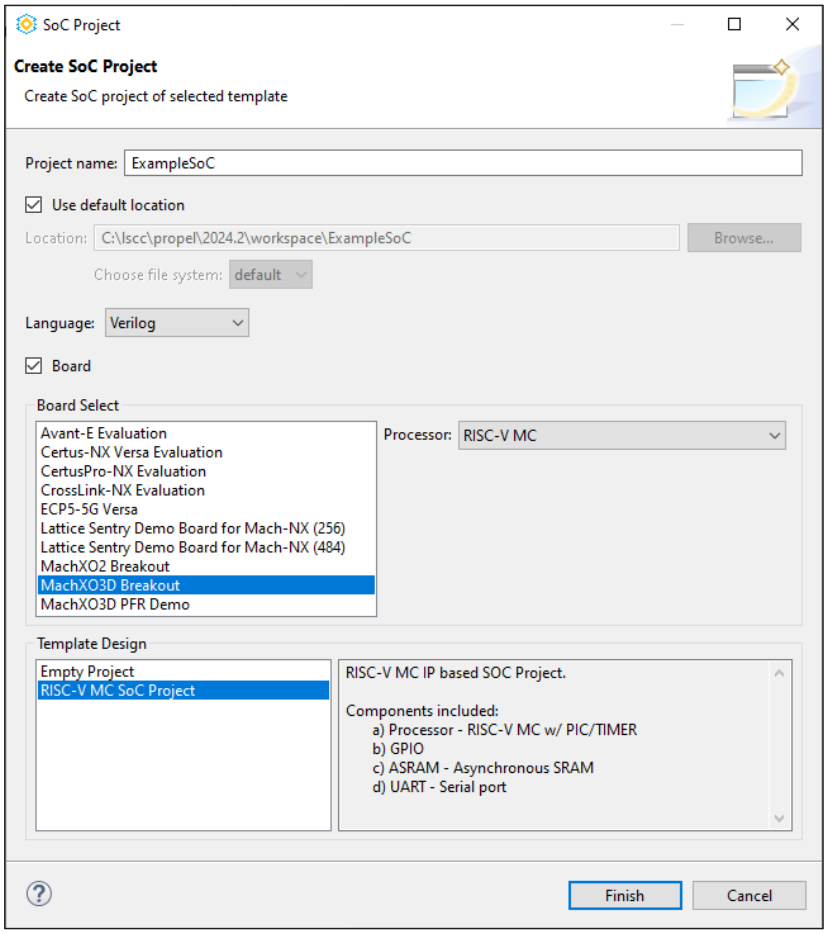


Figure 3.12. Specify a Board for Template SoC Project

2. From the **Board Select** area, select the desired board, such as the MachXO3D™ Breakout Board.
3. Enter a project name.

   Note: Do not include periods, colons, or spaces in the project name.
4. (Optional) To change the default location, clear the **Use default location** option, then browse for another location. Choose a file system.
5. Select a desired platform template design. In particular, select empty project for building system from scratch.
6. Click **Finish**.

   The SoC design project is created in workbench, and its design is opened and displayed in Lattice Propel Builder (Figure 3.15).

3.2.2. Open an SoC Design in Lattice Propel Builder

Within an SoC project, there is a Lattice Propel Builder design.

To open Lattice Propel Builder for an SoC project:

1. In the **Project Explorer** view, select an SoC project.
2. Open the SoC project in one of the following ways from Lattice Propel SDK:
   - Choose **LatticeTools >** **Open Design in Propel Builder** (Figure 3.13).

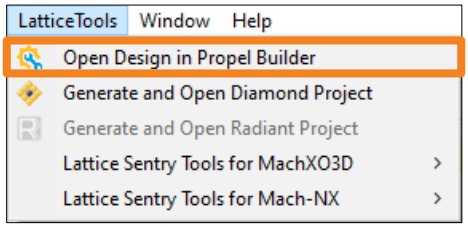


Figure 3.13. LatticeTools Menu

   - Click the Lattice Propel Builder icon on the toolbar.
   - Right-click the SoC project from **Project Explorer**. Choose **Open Design In** > **Propel Builder** from the popup menu (Figure 3.14).

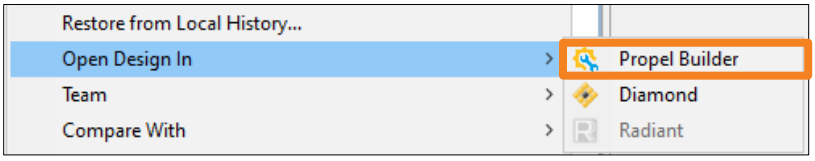


Figure 3.14. Project Explorer Popup Menu

3. The SoC Design is opened and displayed in Lattice Propel Builder (Figure 3.15).

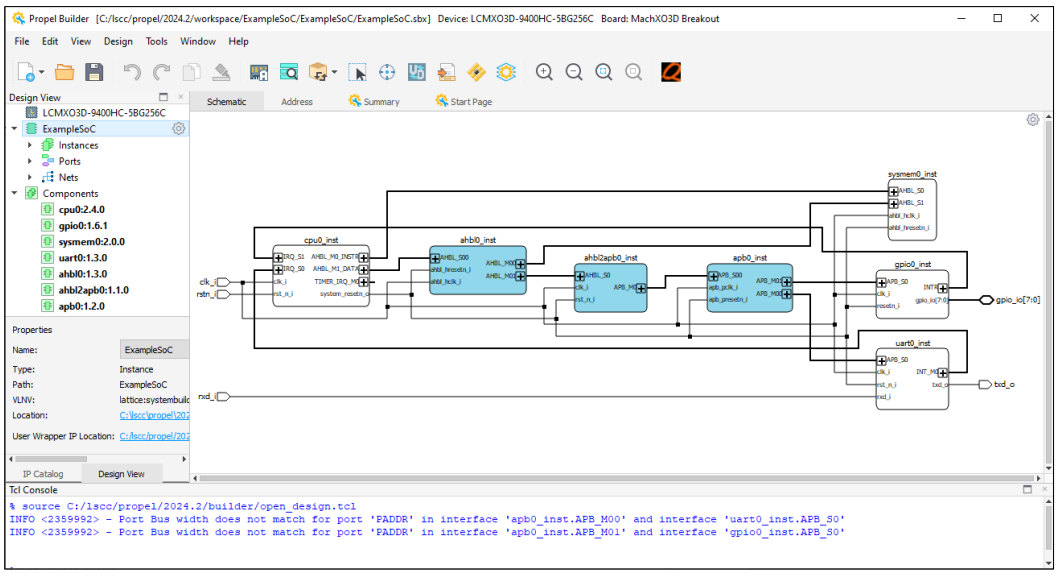


Figure 3.15. Lattice Propel Builder Window 1

4. (Optional) Modify the design in Lattice Propel Builder as desired. Most of the templates include a functional-ready SoC design.

   Note: You can only create an SoC design using **Empty Project** template inside Lattice Propel Builder. Refer to Lattice Propel Builder 2024.2 User Guide (FPGA-UG-02219) for more details on how to create an SoC design using **Empty Project** template.

3.2.3. Open Design in Lattice FPGA Design Software

Within an SoC project, you can create a Lattice FPGA design project including a Lattice Propel Builder design, and then open the FPGA design project in appropriate software. There are two FPGA Design Software available, the Lattice Diamond software and Lattice Radiant™ software. Depending on the device family used in the SoC project, only one of the FPGA Design software can be selected from the User Interface (UI), the other is grayed out. If the MachXO3D or Mach™-NX device family is used, the Lattice Diamond software related menu items are active from the Lattice Propel UI. If the CrossLink™-NX or Certus™-NX device family is used, the Lattice Radiant software related menu items are active from the Lattice Propel UI.

To open FPGA Design Software for an SoC project from Lattice Propel SDK:

1. (Optional) Set Lattice FPGA design software installation location from Lattice Propel SDK. By default, Lattice Propel SDK can find the proper Lattice FPGA design software installation location, usually the latest version installed on the PC. You can overwrite it following steps below.

   Choose **Window** > **Preferences**. The **Preferences** dialog opens (Figure 3.16).

   Select **Propel Setting** from the left pane. Click the **Browse** button to pick up the installation location of the Lattice Diamond software or Lattice Radiant software. Or, leave the **Radiant Location** and **Diamond Location** fields blank, as default. Lattice Propel SDK can find the location automatically.

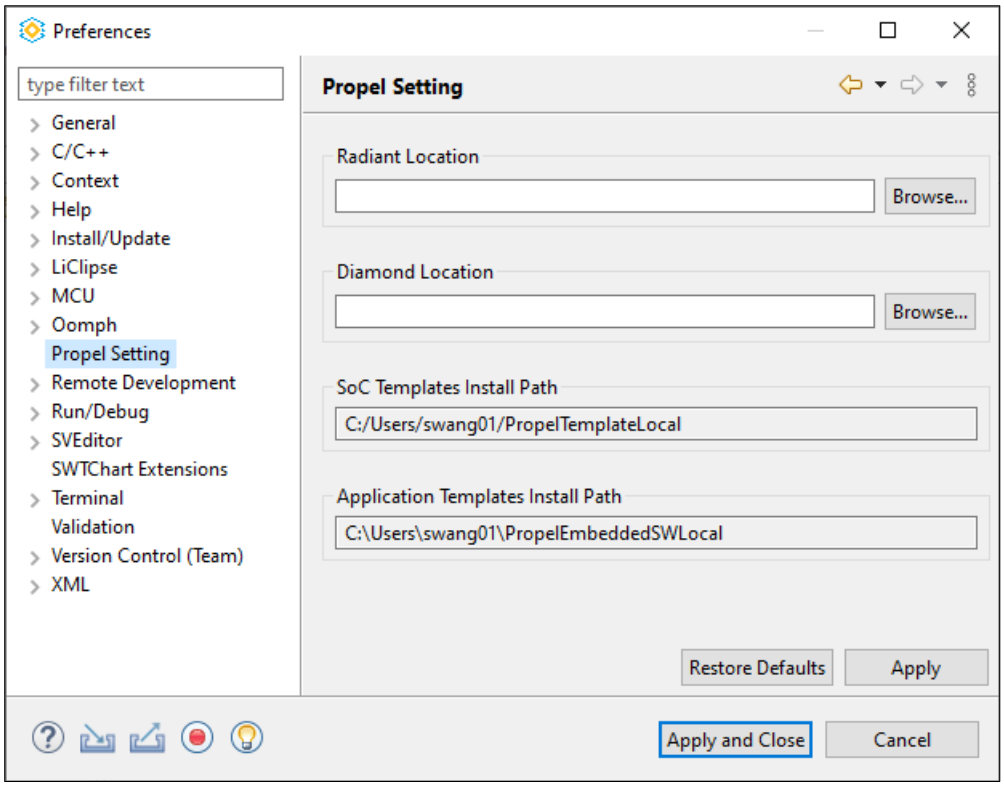


Figure 3.16. Lattice Propel Preferences Dialog

2. In the **Project Explorer** view from the Lattice Propel main Graphical User Interface (GUI), select an SoC project.
3. Open the SoC project in one of the following ways:

- Choose **LatticeTools > Generate and Open Diamond Project**; Or, choose **LatticeTools > Generate and Open Radiant Project**.
- Click the Lattice Diamond software icon or the Lattice Radiant software icon from the toolbar.
- Right-click an SoC project from the **Project Explorer**. Choose **Open Design In** > **Diamond**. Or, choose **Open Design In** > **Radiant** from the right-click menu.

4. The Lattice Diamond/Radiant project for SoC is generated at background and is launched (Figure 3.17/Figure 3.18).

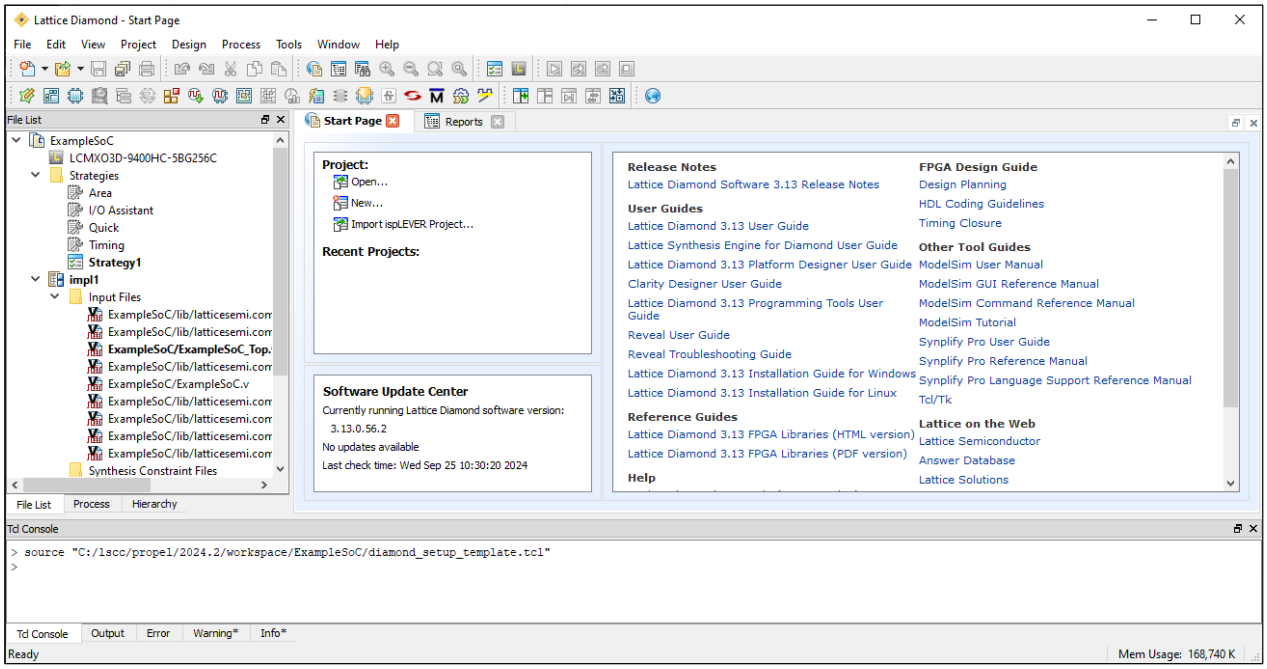


Figure 3.17. Lattice Diamond Software Project

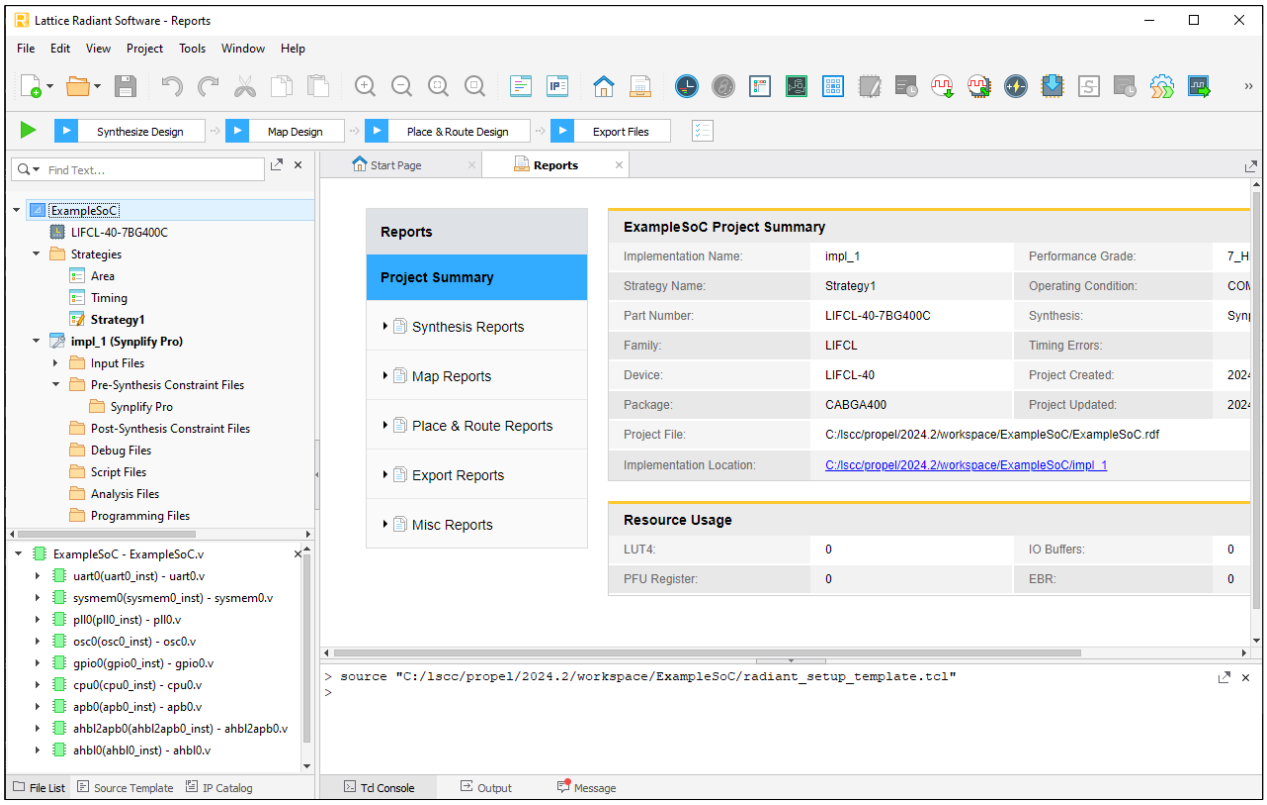


Figure 3.18. Lattice Radiant Software Project

5. (Optional) From the **File List** view of the Lattice Diamond/Radiant software:
   - modify the top-level RTL file (<*proj_name*>_Top.v) to match the SoC design, presupposition of which is that there is a top-level RTL file in your SoC design; or
   - create a top-level RTL file (<*proj_name*>_Top.v) to match the SoC design, if the SoC design is created from an Empty Project template and there is no top-level RTL file in your SoC design.
6. (Optional) Modify constraint file (<*proj_name*>.lpf/<*proj_name*>.pdc) to match the SoC design, if you have modified the SoC design.

   Note: This step is a must for the SoC design created from the Empty Project template.
7. Process the design in the Lattice Diamond/Radiant software.

   In the Lattice Diamond software, switch to **Process** view of the project (Figure 3.19). Make sure at least one file, IBIS Model, Verilog Simulation File, VHDL Simulation File, Bitstream File, or JEDEC file, is checked in the **Export Files** section for programming. Choose **Process > Run**.

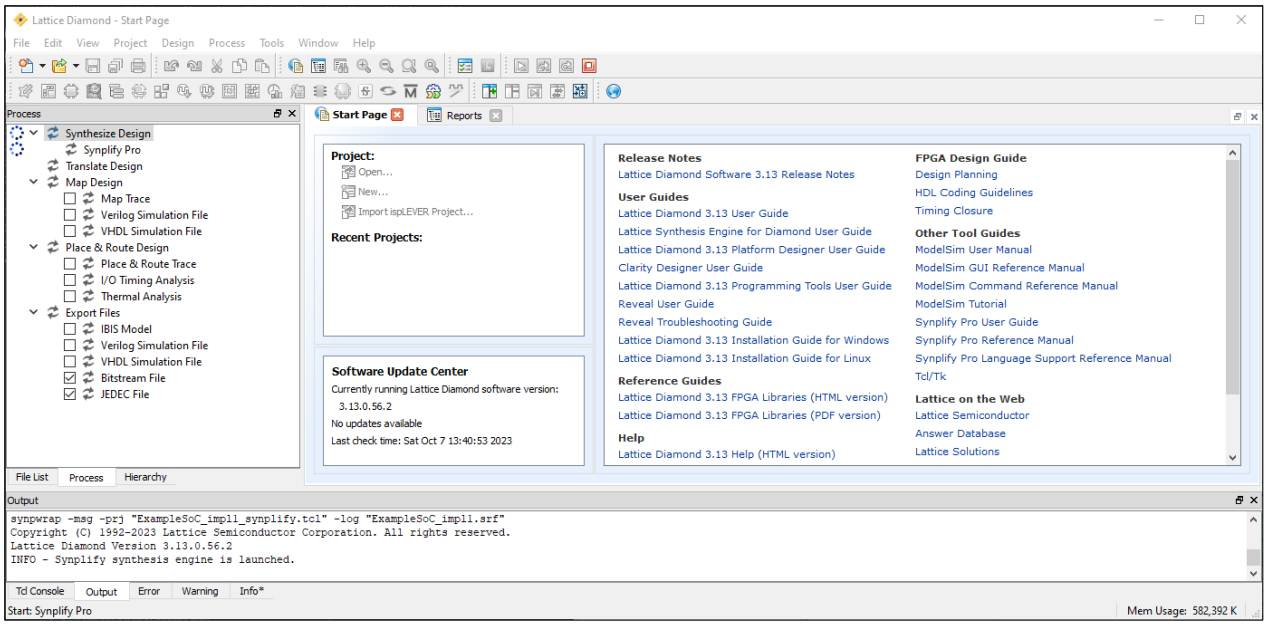


Figure 3.19. Generate Programming File in Lattice Diamond Software

In Lattice Radiant software, from the **Process** Toolbar, click **Export Files** (Figure 3.20).

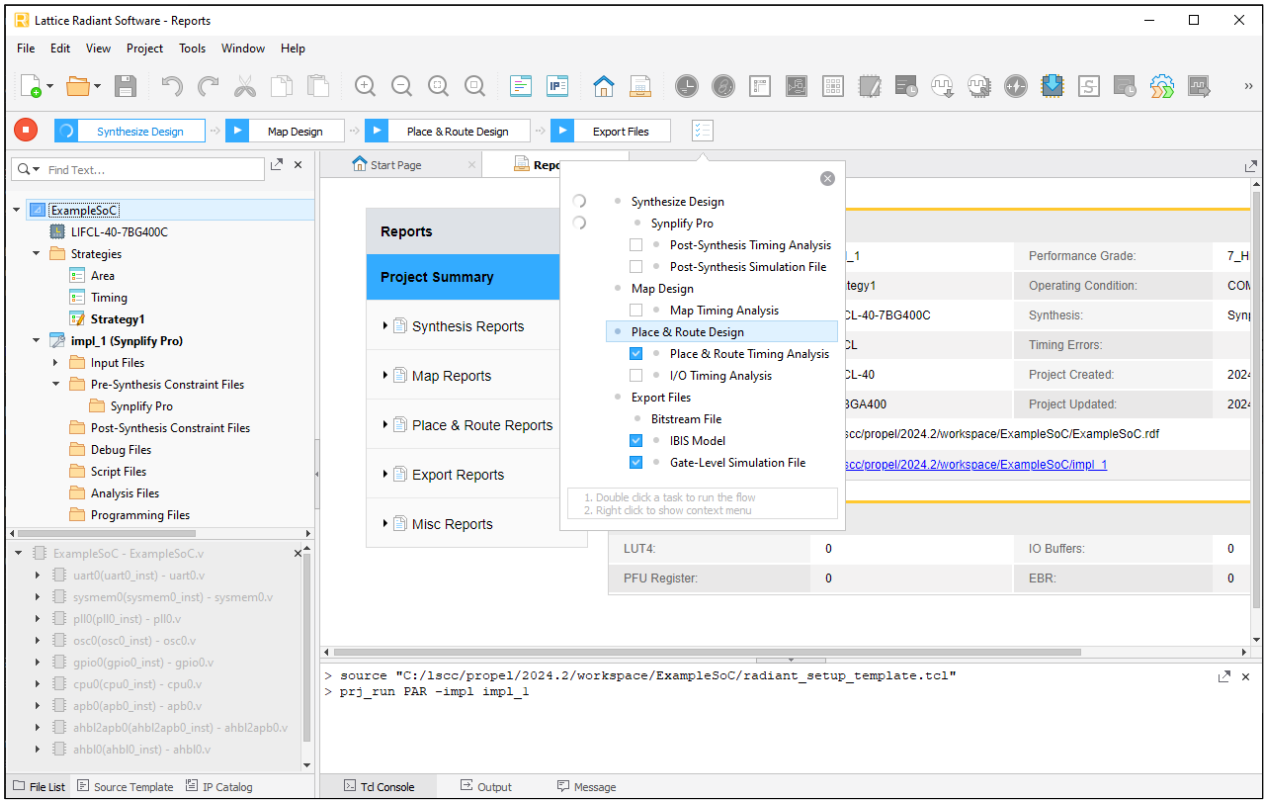


Figure 3.20. Generate Programming File in Lattice Radiant Software

The Programming file is generated. The generated programming file can be used in Programmer.

Note: Programmer is a tool that can program Lattice FPGA SRAM and external SPI Flash through various interfaces, such as JTAG, SPI, and I^2C.

3.2.4. Generating System Environment by Building Project

System environment package including the system environment file and the BSP package is required for the embedded C/C++ project.

To generate system environment package from Lattice Propel SDK:

1. In the **Project Explorer** view, select an SoC project.
2. Choose **Project** > **Build Project**.
3. Check the building result in the **Console** view (Figure 3.21).

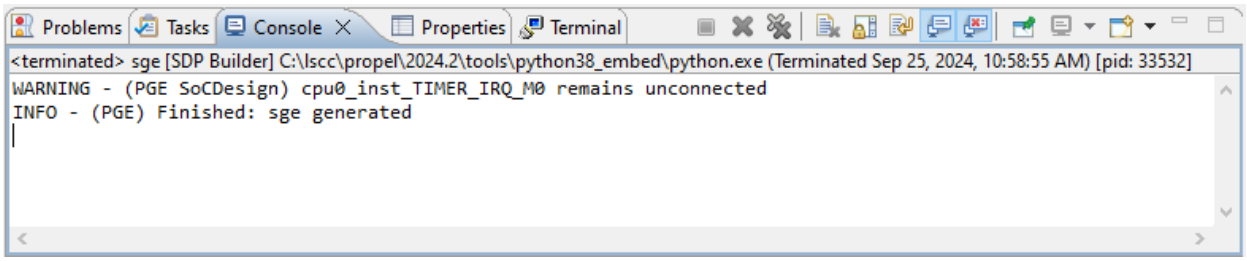


Figure 3.21. Build Result of SoC Project

3.2.5. About SoC Design Project

The SoC project creating starts with a functional-ready SoC design and a default simulation environment. In the **Project Explorer** view, open an SoC project folder and all its sub-folders. The project contains but is not limited to the following files (Figure 3.22), some of which may vary upon opening the SoC design project in the Lattice Diamond or Radiant software:

- <*proj_name*>: folder containing a Lattice Propel Builder design including the .sbx file.
- <*proj_name*>/application: folder containing functional-ready embedded application source codes.
- impl1: folder containing the implementation of the Lattice Diamond/Radiant project.
- sge: folder containing generated package necessary for creating C/C++ project.
- verification: folder containing the SoC verification project.
- verification/sim: folder containing the simulation environment.
- <*proj_name*>.ldf: Lattice Diamond project file.
- <*proj_name*>.lpf: Lattice Diamond project logical preference file.
- <*proj_name*>.rdf: Lattice Radiant project file.
- <*proj_name*>.pdc: Lattice Radiant project post-synthesis constraints.
- <*proj_name*>.txt: description file from the template.

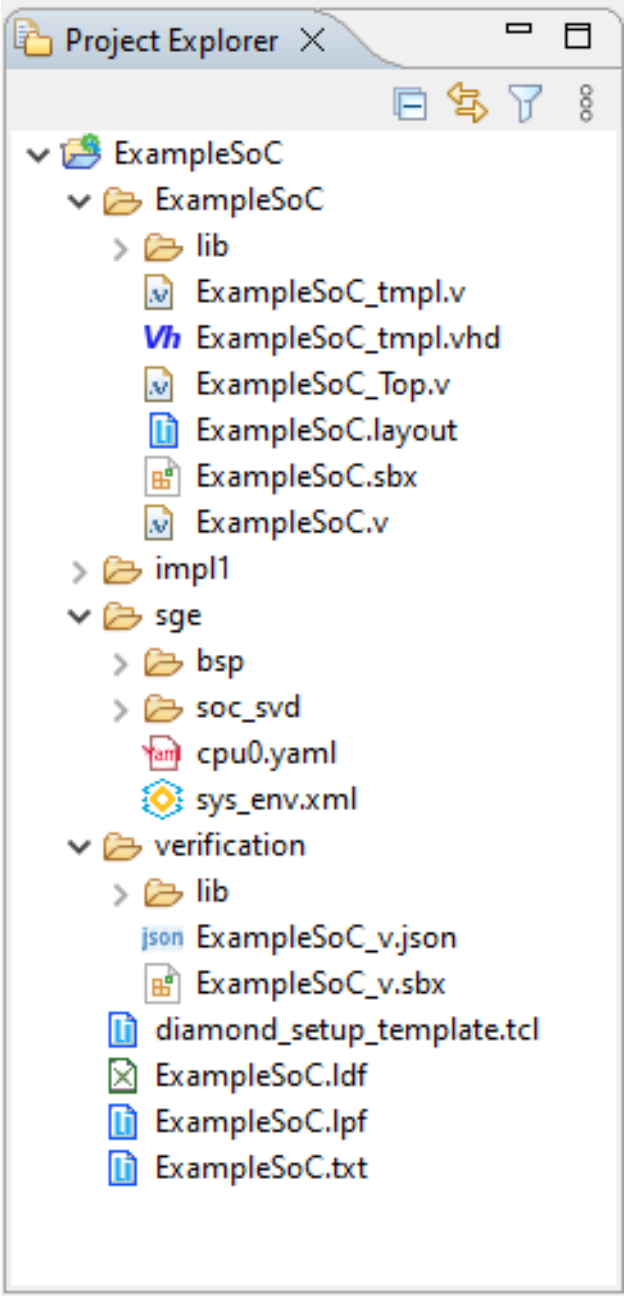


Figure 3.22. Contents of SoC Project

3.3. C/C++ Project Design Flow

3.3.1. Creating a Lattice C/C++ Project

To start a Lattice C/C++ Project from Lattice Propel SDK:

1. Choose **File** > **New** > **Lattice C/C++ Project**.
   The C/C++ Project wizard opens with the **Load System and BSP** page (Figure 3.23).

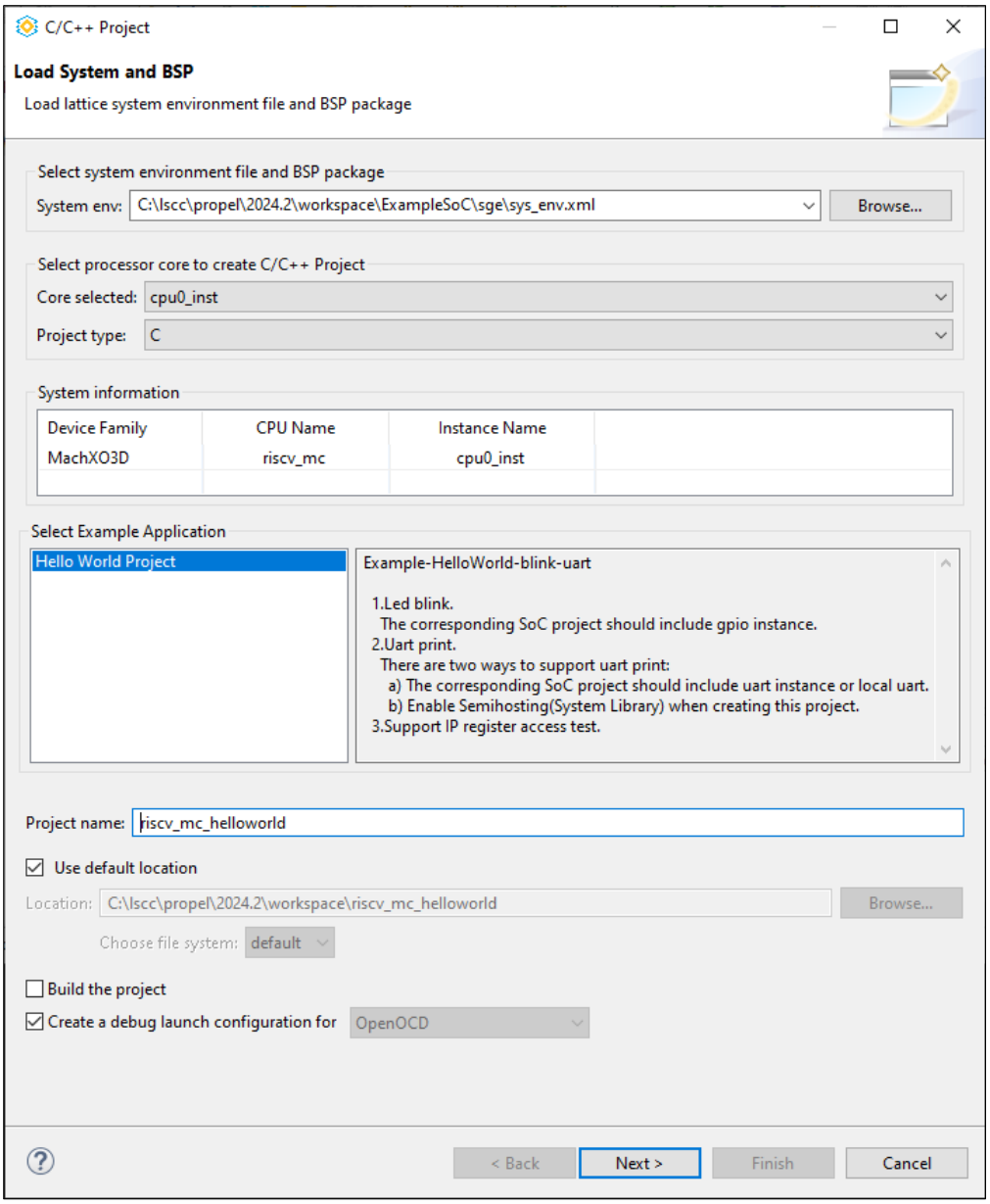


Figure 3.23. Load System and BSP Page 1

2. Browse to the SoC project folder and select the system environment file sys_env.xml.

   All system environment files available in the current workspace can be selected from the System env drop-down menu. If you select or enter **QEMU RISC-V Virtual SoC System**, you can select QEMU application template to create project (Figure 3.24).

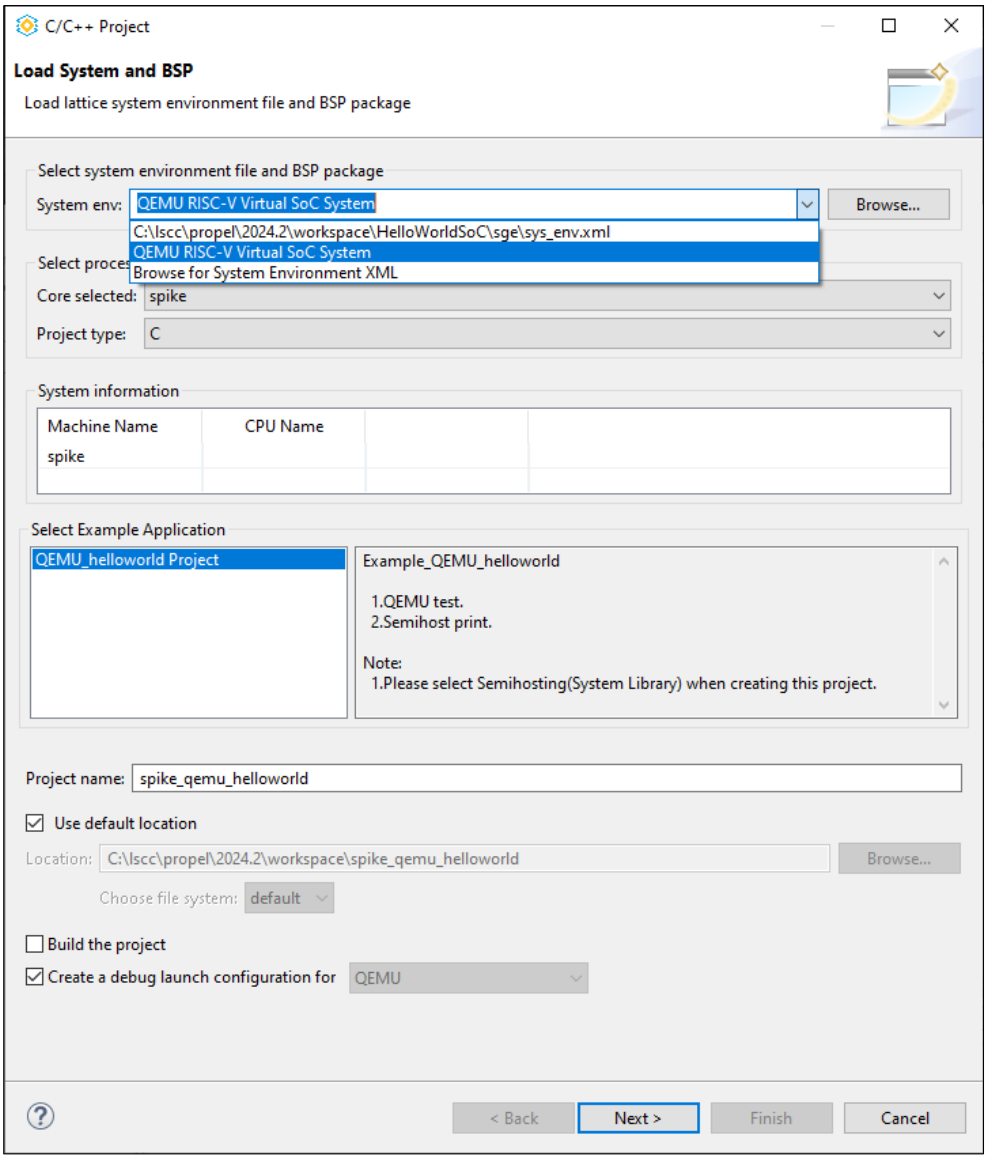


Figure 3.24. Load System and BSP Page 2

3. If the platform has more than one processor, choose one core.
4. Select the project type, C or C++.
5. Select the application in the Example application list.
6. There is a default project name. You need to check it. Suggest not using periods, colons, or spaces in your project name. Though spaces are allowed, they may cause certain issue with some tools.
7. By default, the **Use default location** option is checked. The default file system is selected automatically. Suggest using the default location unless you have special need to a special location.
8. The **Build the project** option is unchecked by default. If you want to build the project automatically, you can check the option.
9. By default, the **Create a debug launch configuration for** option is checked and a default launch configuration is created accordingly.
10. Click **Next**. The **Lattice Toolchain Setting** dialog opens (Figure 3.25).

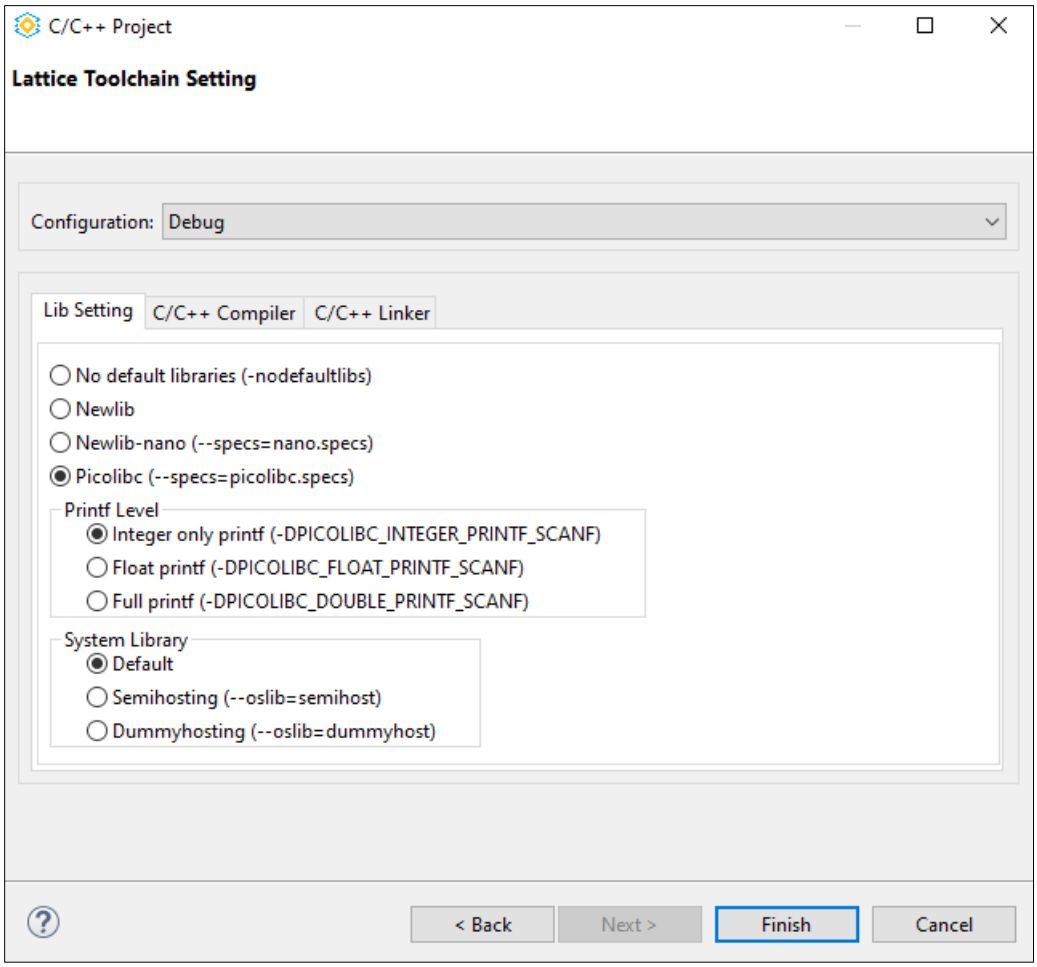


Figure 3.25. Lattice Toolchain Setting Dialog 1

11. By default, two toolchain configuration modes, **Debug** and **Release**, can be chosen from the Configuration drop-down menu.
    - **Debug** configuration creates executables containing additional debug information that lets the debugger make direct associations between the source code and the binary files generated from the original source.
    - **Release** configuration provides the tools with options setting to create an application with the best performance.

    You can modify frequently-used library, compiler, and linker options for each configuration. For a complete toolchain setting, go to project properties after creating the project. Refer to the Advanced Tool Chain Setting section.
    - In **Lib Setting** tab, standard C library can be reconfigured. Picolibc (C Libraries for Smaller Embedded Systems) is selected by default and it supports different printf levels.
    - In **C/C++ Compiler** tab, optimization level and debug level can be reconfigured for each toolchain configuration.
    - In **C/C++ Linker** tab, Remove unused code (--gc-sections) is checked by default for garbage collection of unused code.
12. Click **Finish**.

    The Lattice C/C++ project is created and is displayed using the Lattice Propel SDK perspective. A perspective is a collection of tool views for a particular purpose. The Lattice Propel SDK perspective is for creating Lattice C/C++ programs.

3.3.2. Updating a Lattice C/C++ Project

When you make changes to an SoC project, sometimes you want to synchronize the changes to an existing Lattice C/C++ project instead of creating a new Lattice C/C++ project. In this case, you can use the update C/C++ project feature.

Note: This feature overwrites the corresponding files or settings of your existing C/C++ project. Be sure to back up your C/C++ project before using this feature.

To update a Lattice C/C++ Project from Lattice Propel SDK:

1. Generate the latest system environment package according to the Generating System Environment by Building Project section.
2. In the **Project Explorer** view, select a C/C++ project.
3. Choose **Project** > **Update Lattice C/C++ Project…**.

   The C/C++ Project wizard opens for updating system and BSP (Figure 3.26).

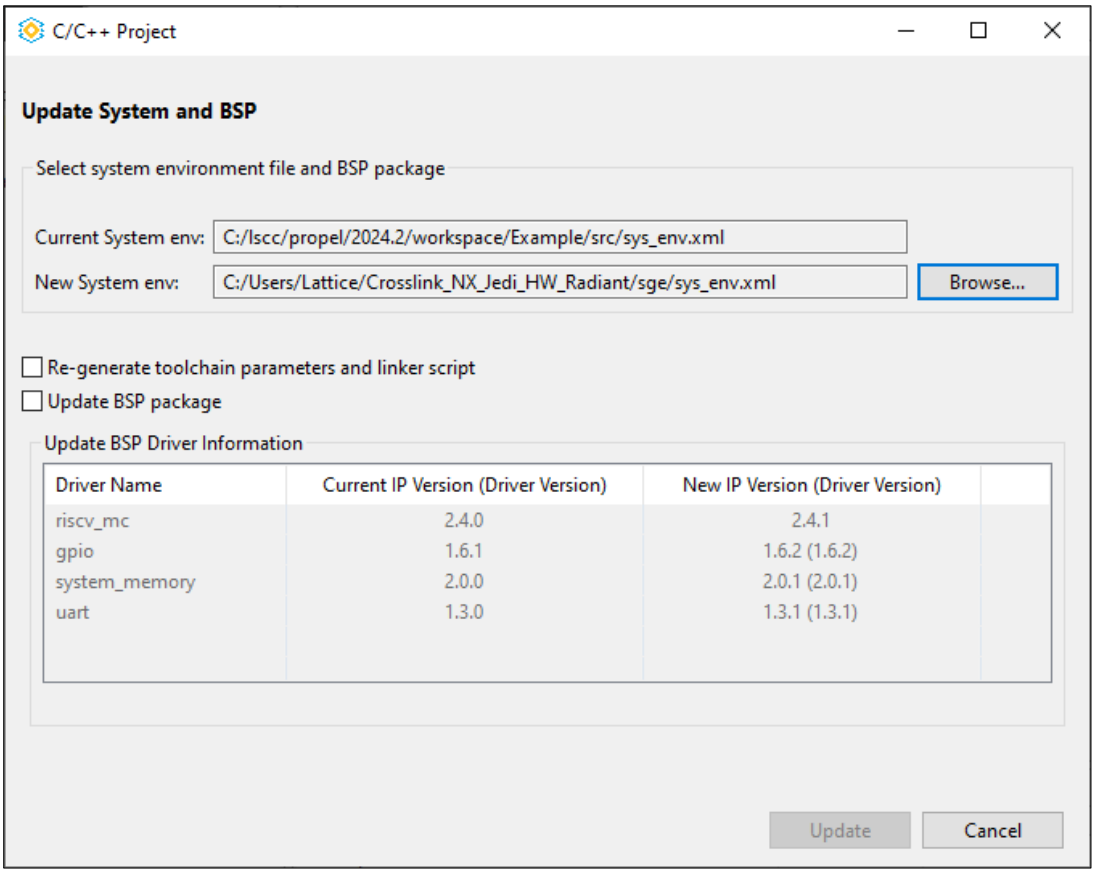


Figure 3.26. Update System and BSP Dialog

4. Browse to the SoC project folder and select the system environment file sys_env.xml.
5. Select the checkbox for what you can update:
   - Re-generate toolchain parameters and linker script: check this option if you want to modify CPU or memory in the system.
   - Update BSP package: check this option if you want to add additional IP components into the system.
6. Click **Update** to make changes for the selected C/C++ project.
7. Click Yes. (Figure 3.27).

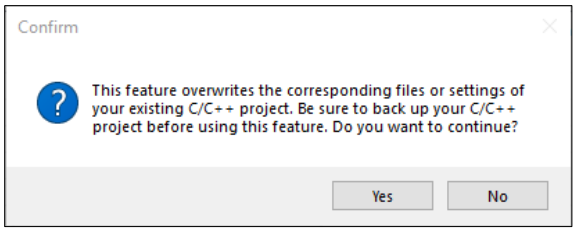


Figure 3.27. Update System and BSP Confirm Dialog

3.3.3. Building a Lattice C/C++ Project

To build a Lattice C/C++ project in Lattice Propel SDK:

1. In the **Project Explorer** view, select a C/C++ project.
2. Follow steps below if you want to change the active build configuration:
   a. Choose **Project > Build Configurations > Manage...**. Or, click the **Configuration** icon on the toolbar.
   b. The **Manage Configurations** dialog opens (Figure 3.28) for choosing active configuration. By default, a **Debug** configuration creates executables containing additional debug information that lets the debugger make direct associations between the source code and the binary files generated from the original source. A **Release** configuration provides the tools with options setting to create an application with the best performance.

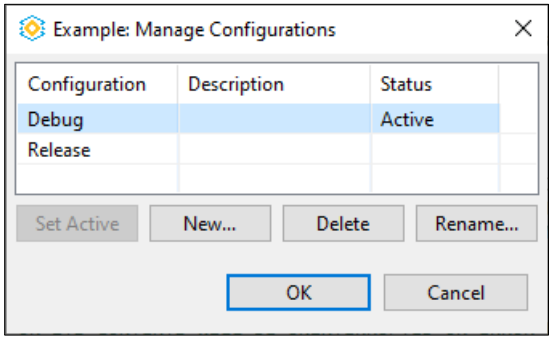


Figure 3.28. Manage Configurations Dialog

3. Choose **Project** > **Build Project**. Or, click the **Build** icon on the toolbar.
4. The results of the build command are displayed in the **Console** view (Figure 3.29).

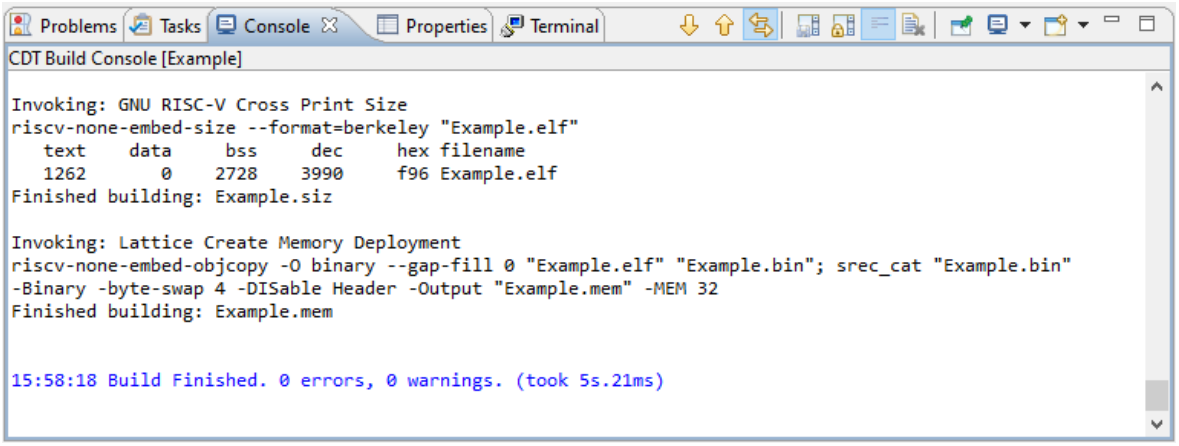


Figure 3.29. Build Result of C/C++ Project

3.3.4. About Lattice C/C++ Project

The Lattice C/C++ project starts with source code. In the **Project Explorer** view, open a C/C++ project folder and all its sub-folders. The project contains:

- src/bsp/driver: folder containing driver codes from the IP in the platform.
- src/bsp/sys_platform.h: header file that defines DEVICE_FAMILY (the Lattice FPGA), address mapping, and any IP parameters that can be used by the drivers.
- src/main.c: source file containing the main routine, which is the entry-point of a C/C++ program.
- src/cpu.svd: system view description file used for peripherals registers view at debug perspective.
- src/cpu.yaml: processor description file used at debugging time.
- src/linker.ld: linker script file.
- src/sys_env.xml: system environment file describing aspects of the platform, such as memory spaces.

After building the project, the build output can be found in each build configuration folder, the **Debug** folder or the **Release** folder (Figure 3.30). The Debug or Release folder contains:

- <*proj_name*>.elf: executable file used in on-chip debugging.
- <*proj_name*>.bin: binary file used in deploying the application to flash memory.
- <*proj_name*>.lst: extended listing file generated by tool objdump.
- <*proj_name*>.map: linker map file.
- <*proj_name*>.mem: Lattice system memory initialization file used in System_Memory IP.
- <*proj_name*>.launch: Debug launch configuration.

Note: Some of the files listed in Figure 3.30 are intermediate files that you do not need to take care of.

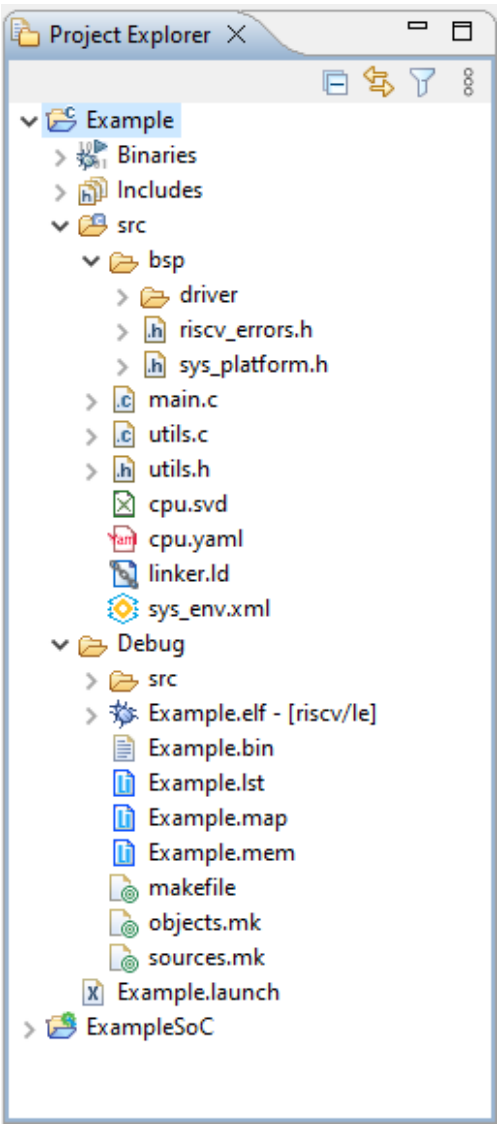


Figure 3.30. Contents of C/C++ Project

3.3.5. Assisting in developing Code

Lattice Propel SDK is based on Eclipse IDE. You can write application code following the process and usage of the same tools as any in Eclipse IDE. You can get more detailed information regarding Eclipse IDE from the Lattice Propel online help.

For writing code, Lattice Propel SDK provides two extra aids:

- Lattice System Platform: An overview of the processor platform can be displayed (Figure 3.31).
- Linker Editor: An overview of the memory regions of linker script can be displayed. You can modify key linker parameters via the graphical interface (Figure 3.32).

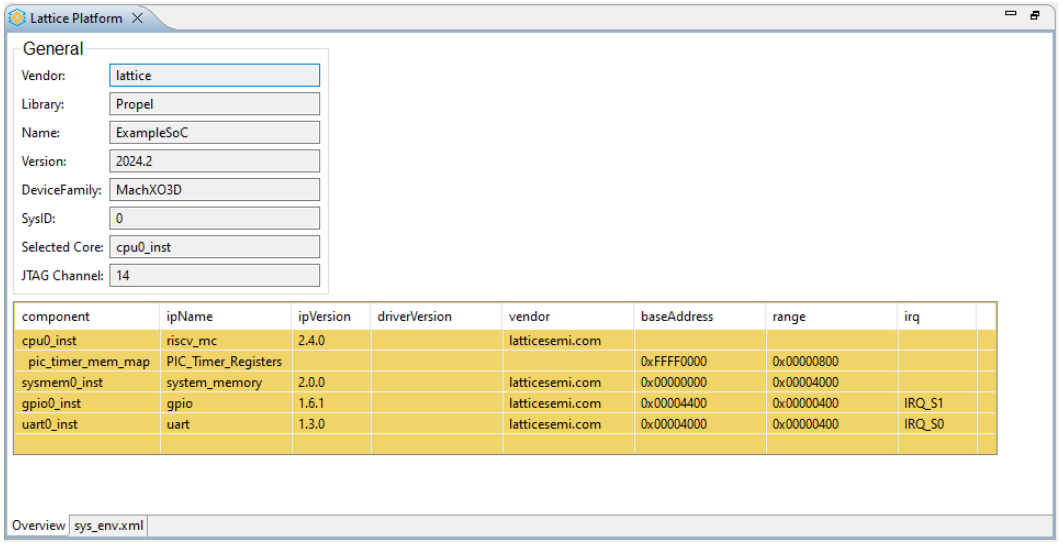


Figure 3.31. Lattice System Platform

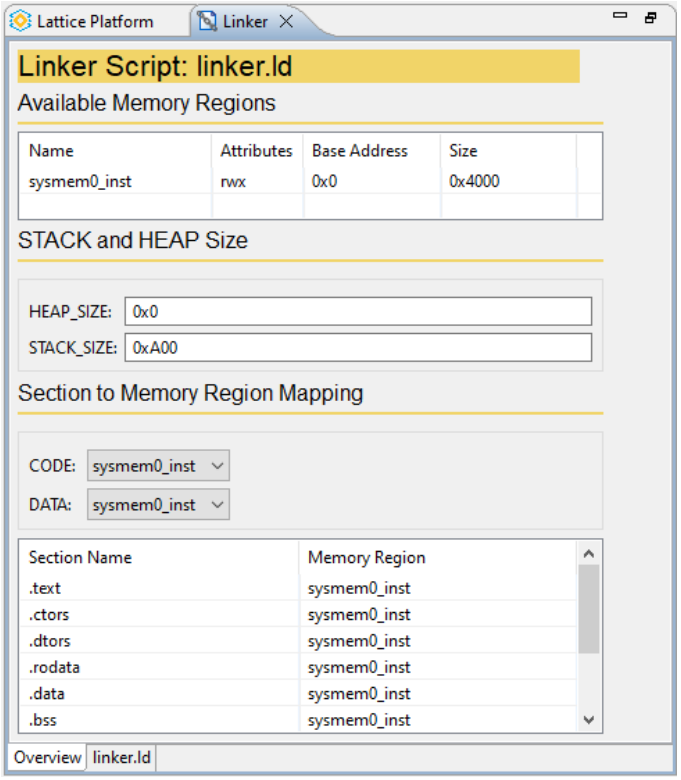


Figure 3.32. Linker Editor

3.3.6. Advanced Tool Chain Setting

Follow the process below to modify the tool chain settings of a C/C++ project.

To change tool chain setting in Project Properties in Lattice Propel SDK:

1. In the **Project Explorer** view of Lattice Propel SDK, select a C/C++ project.
2. Choose **Project** > **Properties.** The Properties for the current project opens (Figure 3.33).
3. Select **Settings** of **C/C++ Build** category from the left pane. Select the **Tool Settings** tab.

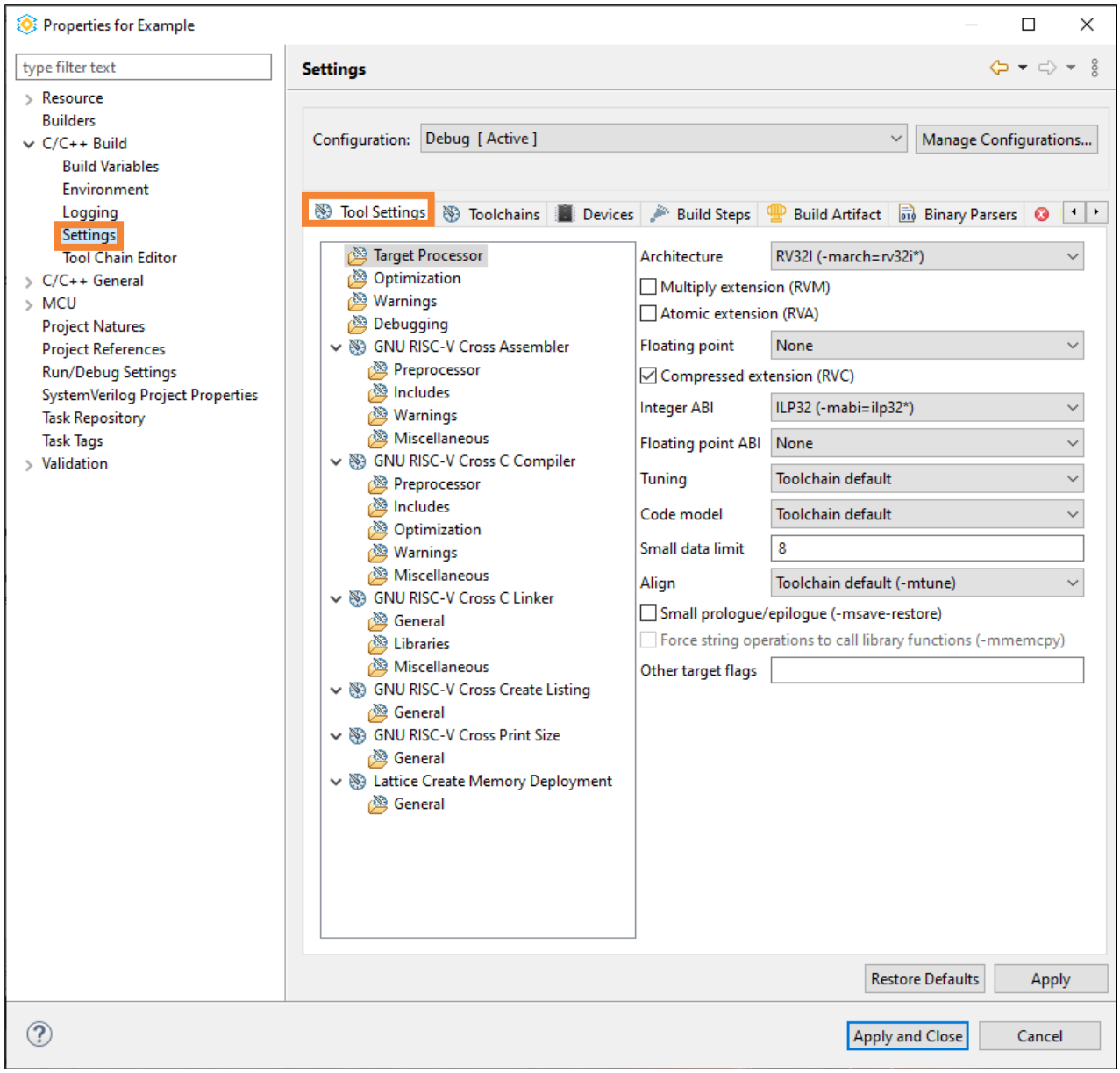


Figure 3.33. Properties of C/C++ Project

4. Customize the tools and tool options. All your customization can be resulted in the build configuration in the **Tool Settings** properties tab. The build configuration is used during your C/C++ project building.

   Note: The setting for each configuration, **Debug** or **Release**, is independent.

5. Click **Apply and Close** to save the change.

   Note: You may need to clean the project to make the new setting take effect for the whole project.

3.4. System Simulation Flow

The SoC Project created from template has a default simulation environment for you to setup and start functional simulation. It is generated automatically along with the SoC project creation. You can use it as a start point and customize accordingly.

The default simulation environment is with the following features:

- Provides similar user experience as real board-level debugging, such as for Hello World SoC, key components including RISC-V MC, System memory, and UART.
- Simulates user-modified template SoC with extended HDL designs.
- Simulates the whole system using real C/C++ projects as stimulus with the necessary modification and with all the details for debugging.
- Supports user extension with a friendly and flexible approach.

3.4.1. Launch Simulation

To launch simulation:

1. In Lattice Propel Builder, update the SoC design to enable simulation features.

   Enable the checkbox for **Initialize Memory** for system_memory IP module from the **Initialization** area of the **General** tab. Then, set the **Initialization File** generated from the corresponding C/C++ project (Figure 3.34).

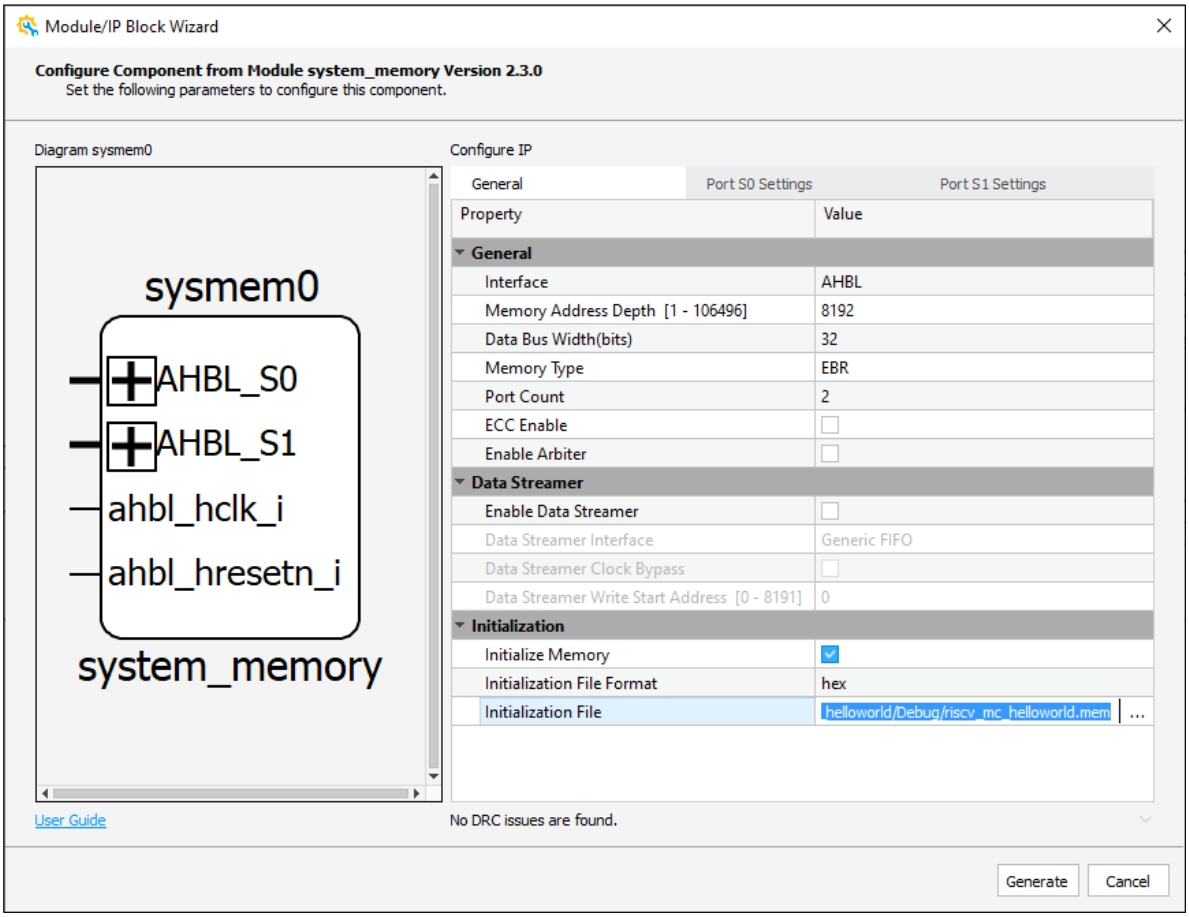


Figure 3.34. Configure Module System Memory 1

2. Click the **Switch** icon on the toolbar to switch between SoC design and SoC verification project (Figure 3.35).
3. After the SoC design is switched to an SoC verification project, click the **Generate** icon to generate the simulation environment. Click the **Launch Simulation** icon (Figure 3.35).

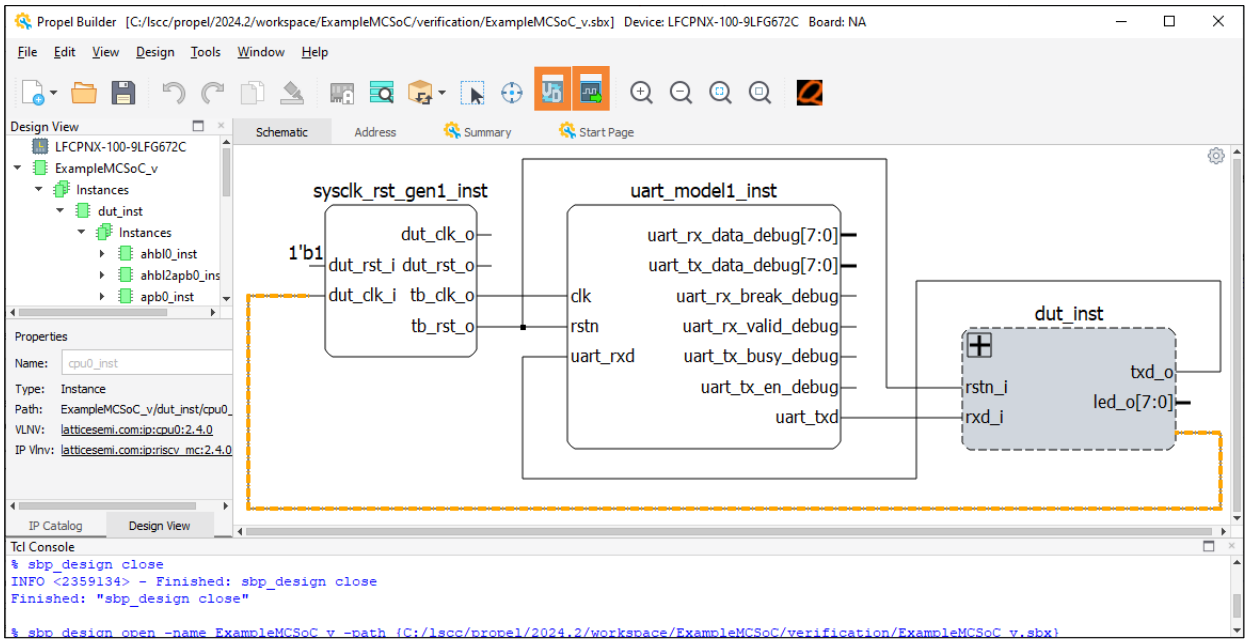


Figure 3.35. SoC Verification Project

4. QuestaSim is launched running simulation for the SoC verification project. The corresponding waveform of the SoC verification project for the Hello World project is shown (Figure 3.36). Check the waveform.

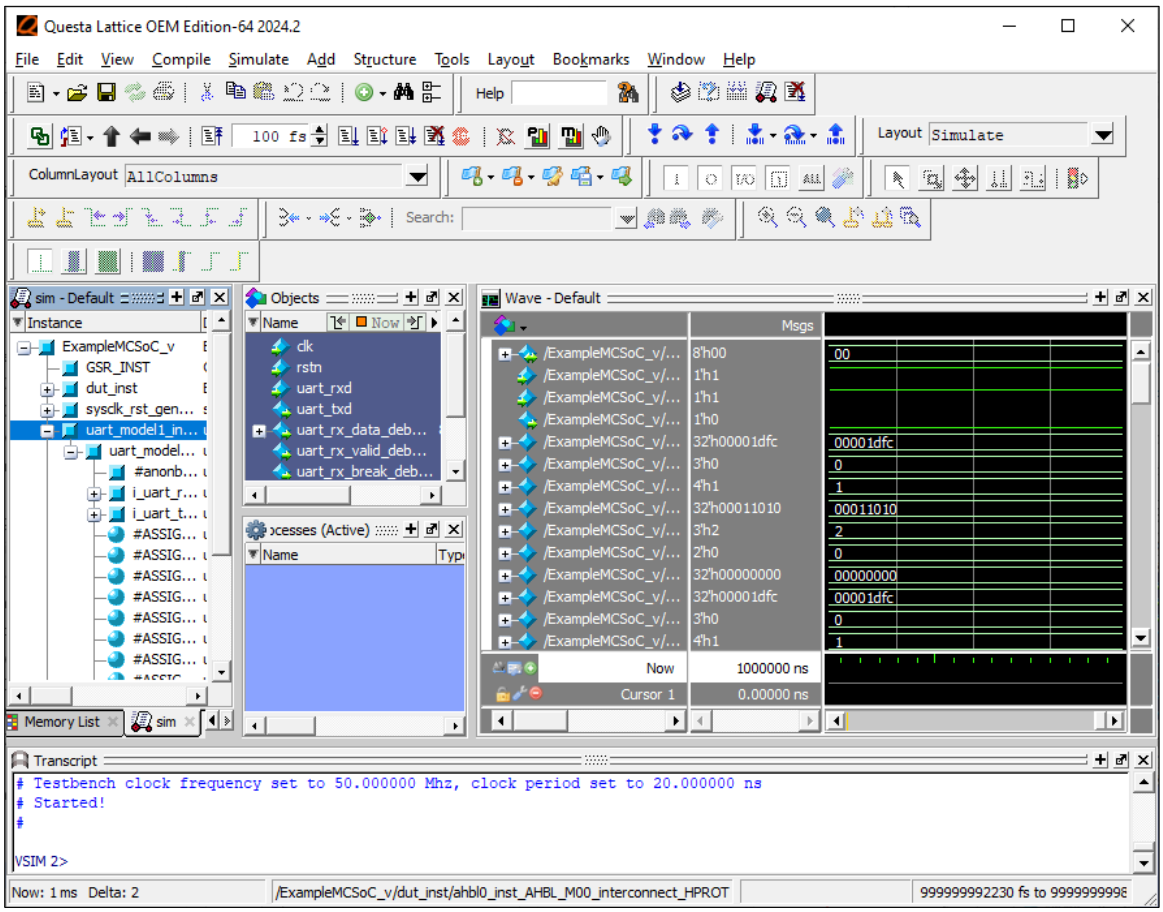


Figure 3.36. Questa Simulation GUI

3.4.2. Simulation Details

The default simulation environment is located at the generated sim folder inside the SoC verification project in Lattice Propel SDK. It contains:

```
+--- [sim]                    -- Generated simulation environment folder
|   +--- [hdl_header]
|   |   +--- soc_regs.v       -- Register definitions of all the components in DUT/SOC
|   |   +--- sys_platform.v   -- Base address, user settings of all the components in DUT/SOC
|   +--- [misc]
|   |   +--- *.*              -- All the mem, hex, txt files are copied here
|   +--- flist.f              -- File list for HDLs
|   +--- flist_sim.f          -- File list for all files used in simulation
|   +--- qsim.do               -- Do script for simulator,
                                 qsim.do: QuestaSim.
                                 This file compiles project and invokes simulator with
|                                some default settings using the generated testbench.
|   +--- wave.do              -- Do script for adding signals in waveform window
|   +--- <project_name>_v.sv  -- Top testbench, it's SystemVerilog based.
```

You can extend more verification features in the top testbench.

3.5. Programming and On-Chip-Debugging Flow

This section describes the process of testing and debugging application code on the actual hardware including the Lattice FPGA with the hardware design installed. Debugging with Lattice Propel SDK follows the same process and uses of the same tools in Eclipse IDE.

Before debugging, download the hardware design created from the Lattice Diamond/Radiant Programmer. Refer to the User Guide of the specific evaluation board for more details on the evaluation board.

3.5.1. Creating a Debug Launch Configuration

To debug a program, a debug launch configuration must be created. Most of the settings for a debug launch configuration can be automatically entered. Only a few settings need to be manually configured.

To create a debug launch configuration:

1. In the **Project Explorer** view of Lattice Propel SDK, select a C/C++ project.
2. Build the project and ensure the executable file is available. Refer to the Building a Lattice C/C++ Project section for details on the process.
3. Choose **Run** > **Debug Configurations…**.

   The **Debug Configurations** dialog opens (Figure 3.37).

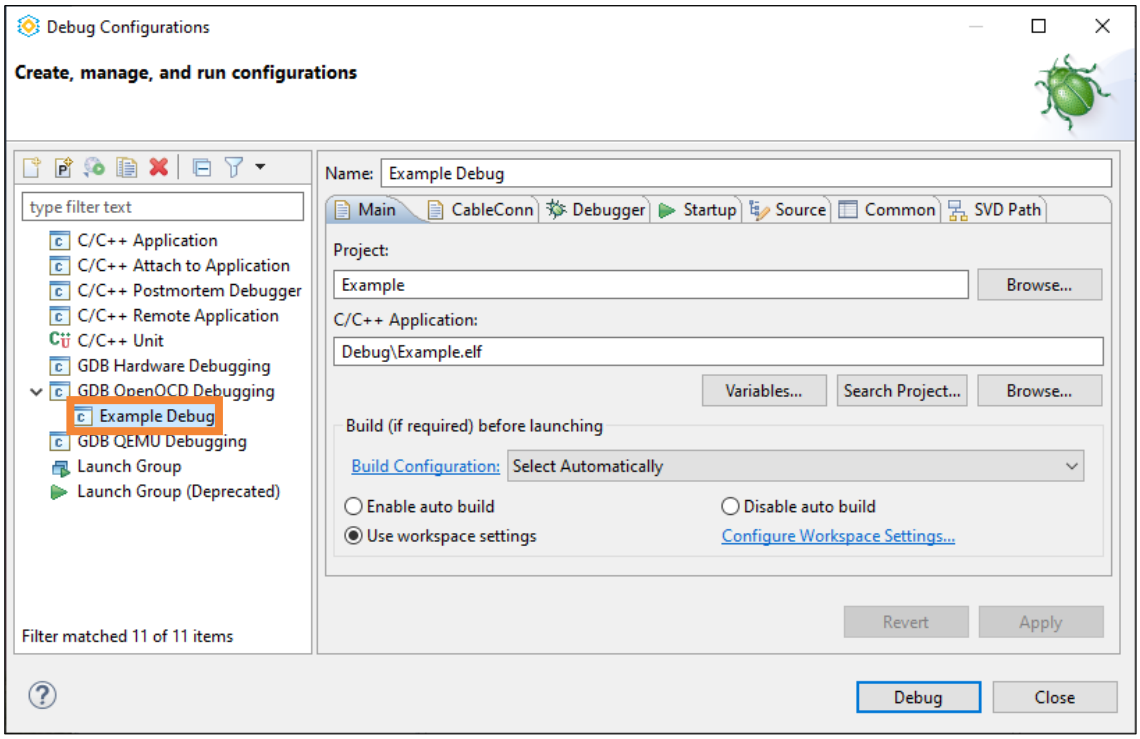


Figure 3.37. Debug Configurations Dialog 1

4. Double-click **GDB OpenOCD Debugging** to create a new launch configuration.

   A multi-tab page is displayed. The Main tab should already be filled in with the project name, application file name, and location.

5. Select the **CableConn** tab (Figure 3.38). This tab enables you to select a specific device on a specific cable port.

   Click the **Detect Cable** button. Select the specific cable port from the **Port** drop-down list. By default, the first available cable port: FTUSB-0 is selected.

   Click the **Scan Device** button. Select the specific device from **Device** drop-down list. By default, the first available device on selected cable port is selected.

   Select the JTAG channel number from **Channel** drop-down list. By default, channel 14 is selected with the same value as the processor preset.

   Keep the cable speed so that you can use the default clock divider.

   Note: You need to repeat the **Detect Cable** and **Scan Device** steps if you have plugged or unplugged the cable.

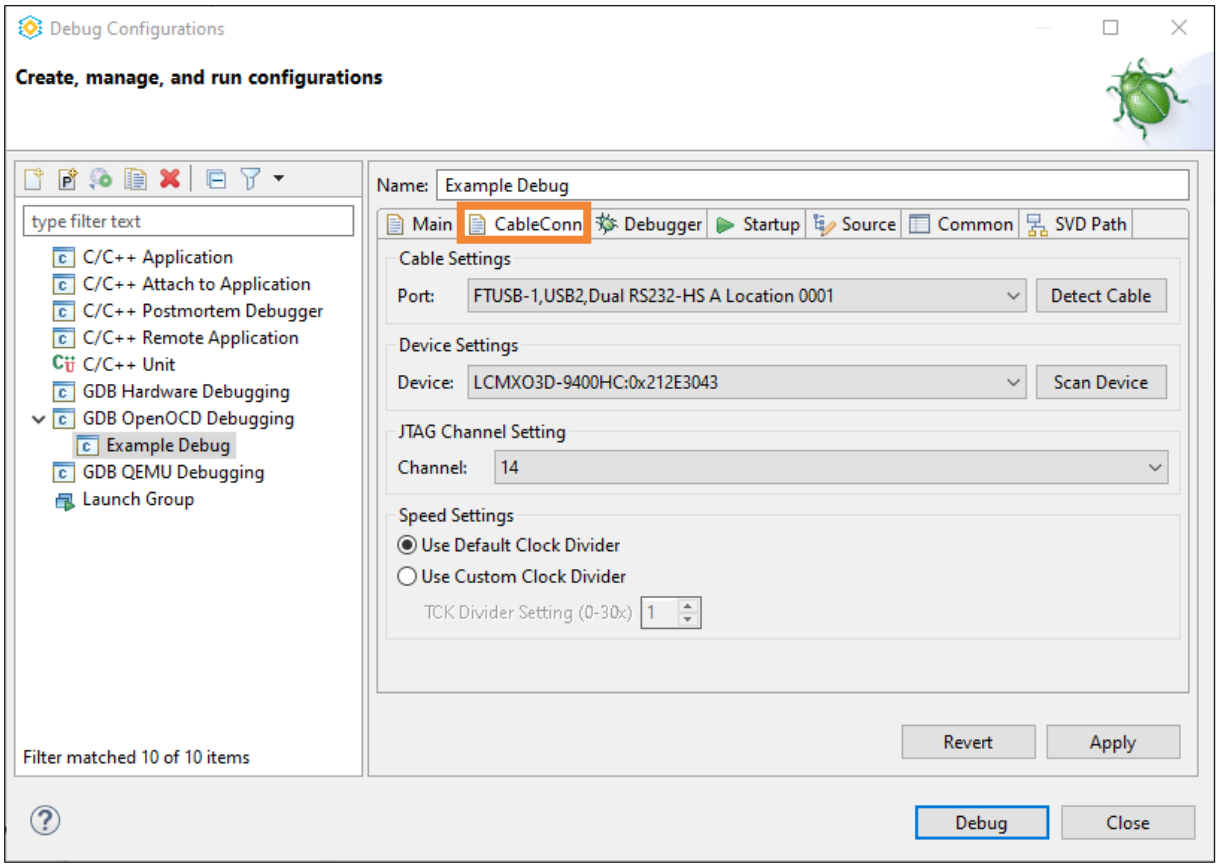


Figure 3.38. CableConn Tab of Debug Configurations

6. Select the **Debugger** tab (Figure 3.39). It is critical that the **Config options** field contains the correct command line options to be passed to OpenOCD.

   -c "set port ${PORT}"* is required for the selection connected to the Lattice cable. The value of variable "*${PORT}" comes from the cable settings of the **CableConn** tab.

   -c "set target ${DEVICE}"* is required for the selection of a specific device on the Lattice cable. The value of variable "*${DEVICE}" comes from the device settings of **CableConn** tab.

   -c "set channel ${CHANNEL}"* is required for setting the JTAG channel. The value of variable *"${CHANNEL}" comes from the jtag channel setting of the **CableConn** tab.

   -c "set tck ${TCKDIV}"* is required for setting the clock divider of the Lattice cable. The value of variable *"${TCKDIV}" comes from the speed setting of the **CableConn** tab.

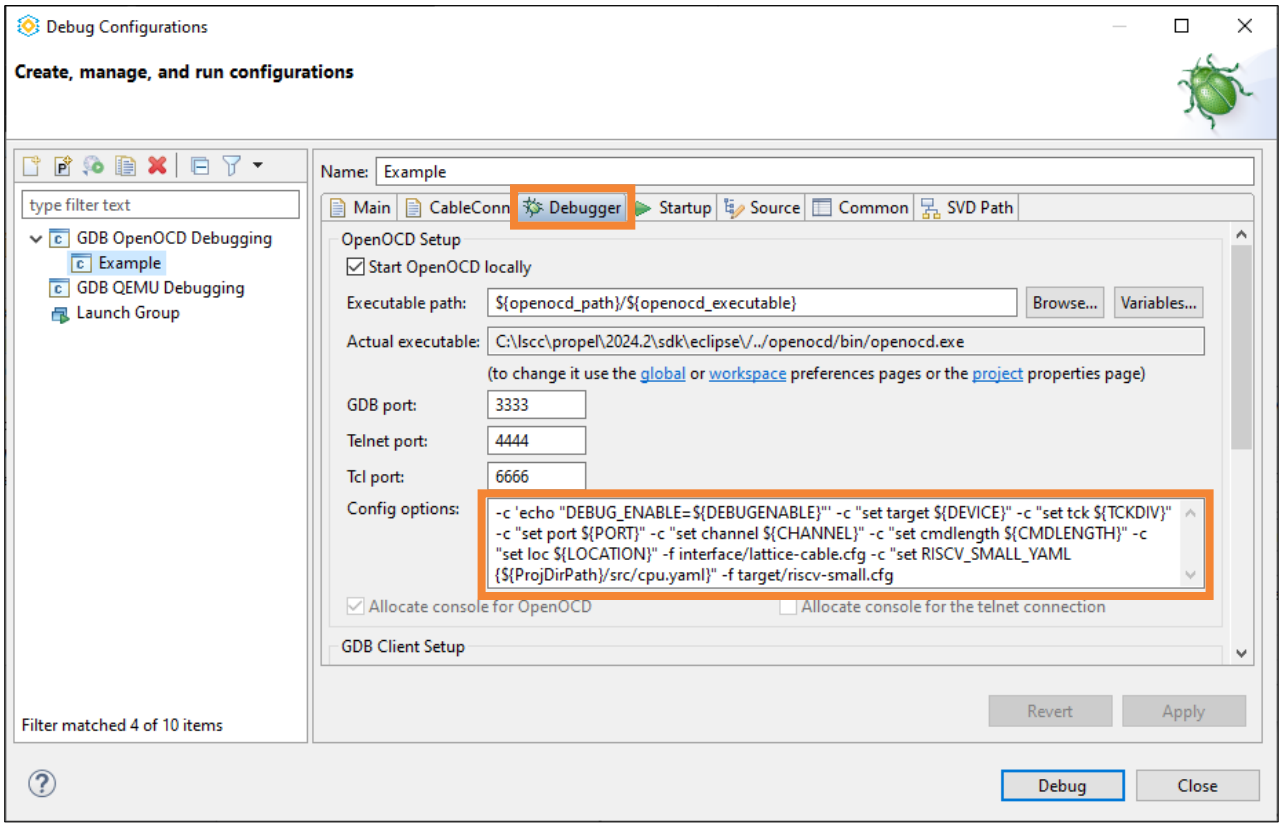


Figure 3.39. Debugger Tab of Debug Configurations

7. (Optional) Select the **Common** tab (Figure 3.40). The **Save as** > **Local file** option is selected by default. This causes the debug launch configuration to be saved into the workspace.

   You can change the setting of the **Save as** field to **Shared file**. In this way, the debug launch configuration is saved into the project and this aids the project portability.

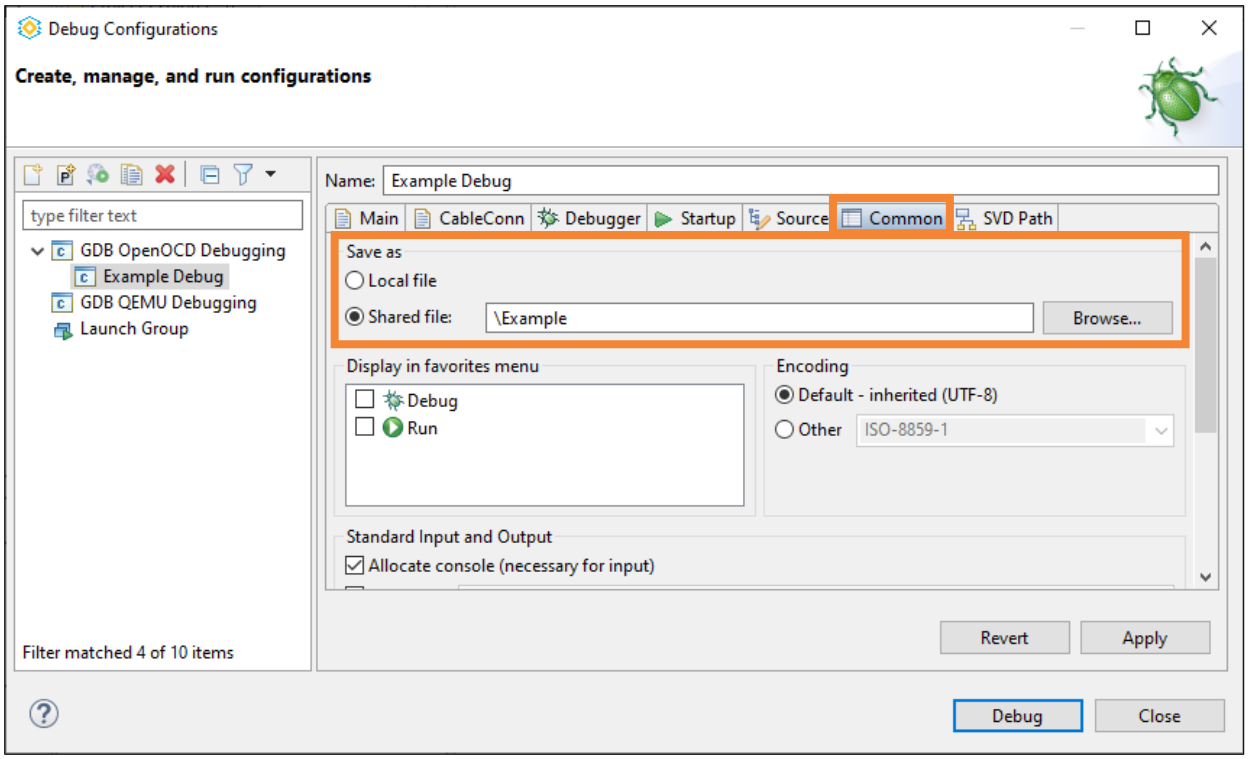


Figure 3.40. Common Tab of Debug Configurations

8. Remain settings as default. Do not change the settings unless necessary, or unless you understand what effect these changes may bring.
9. Click **Apply** to keep the current settings.
10. Click **Close**.

Note: By default, **C/C++ Application, C/C++ Attach to Application, C/C++ Postmortem Debugger, C/C++ Remote Application, C/C++ Unit, GDB Hardware Debugging** are hidden on the Debug Configurations page. If you want to use them, you can disable the **Filter checked launch configuration types** (Figure 3.41).

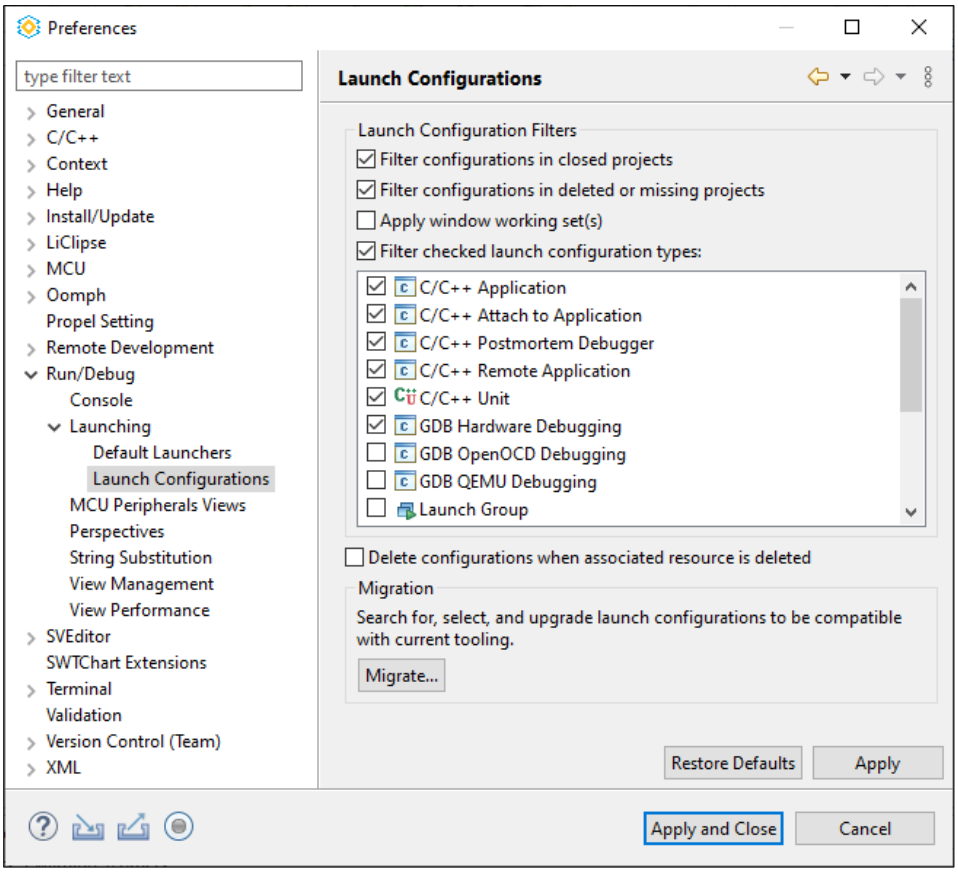


Figure 3.41. Launch Configurations

3.5.2. Starting a Debug Session

Before starting a debug session, be sure that:

- The Lattice cable is connected to the computer.
- The target device is power ON.
- The hardware design has a debug enabled processor module and already programming into the target device.

With the above steps completed properly, follow the steps below to start the debug session from Lattice Propel SDK.

1. Choose **Run** > **Debug Configurations…**.
2. If necessary, expand the **GDB OpenOCD Debugging** group.
3. Select the newly-defined configuration.
4. Click the **Debug** button (Figure 3.40).

   Alternatively, for later sessions, use the **Debug** icon on the toolbar. Do not click the Debug icon directly. Instead, click the down arrow beside the **Debug** icon. Select the desired debug configuration from the drop-down menu (Figure 3.42).

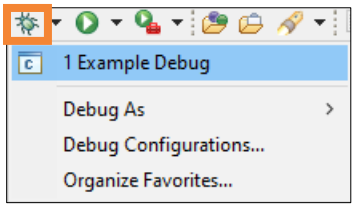


Figure 3.42. Debug Icon on Toolbar

5. Wait for a few seconds for switching to debug perspective, starting the server, connecting to the target device, starting the gdb client, downloading the application, and starting the debugging session.
6. The Lattice Propel Window displays, as shown in Figure 3.43. The execution stops right at the beginning of the main() function.

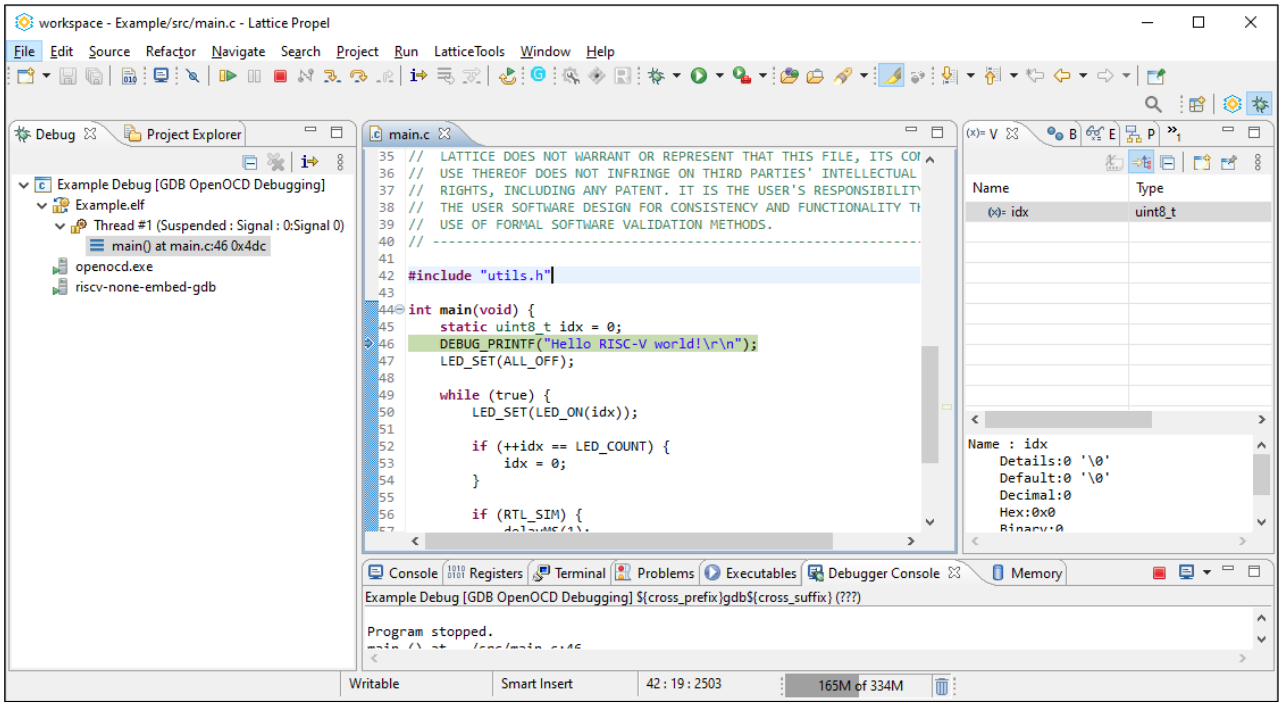


Figure 3.43. Debug Perspective 1

Note: If you need to do Reveal and Lattice Propel On-Chip-Debug concurrently, it is recommended to launch Lattice Propel SDK On-Chip-Debug first before launching Reveal.

3.5.3. Peripherals Registers View

Peripherals registers view provides an easy-to-use interface for examining or modifying the values of peripheral registers during a debug session.

To use peripherals registers view in Lattice Propel SDK (Figure 3.44):

1. Make sure an active debug session is run and shown in the debug perspective.
2. Find the **Peripherals** view which is in the same window with the **Variables** and **Breakpoints** views. For any reason, if this view is not found, re-open it from **Window** > **Show View** > **Peripherals**.

   The **Peripherals** view lists all peripherals available in the system view description svd file within the C/C++ Project.
3. Selecting a peripheral in the **Peripherals** view can open a **Memory Monitor** that is mapped to the corresponding peripheral memory area.
4. You can examine and modify the value of the peripherals register in the **Memory** view.

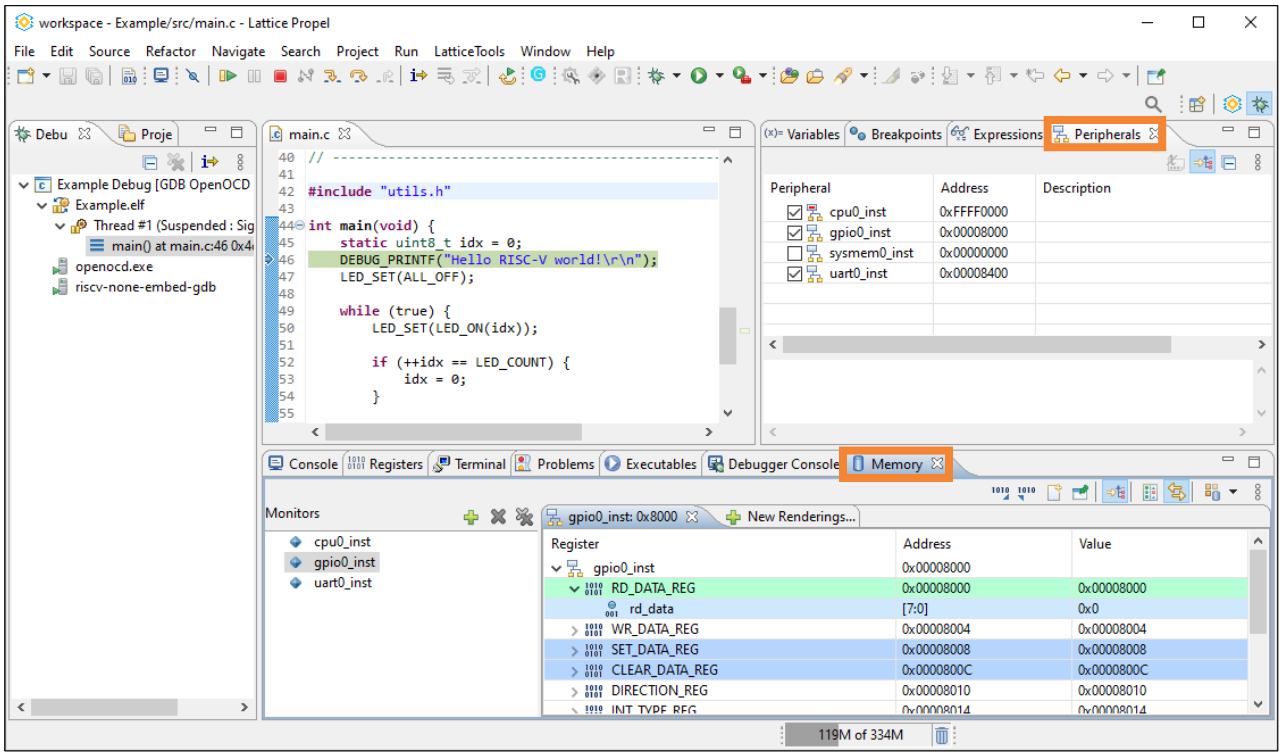


Figure 3.44. Peripherals View in Debug Perspective

3.5.4. Serial Terminal Tool

Serial port communication is frequently used during the microcontroller debugging. Lattice Propel SDK provides a built-in terminal tool including serial support for debugging.

To launch a serial terminal:

1. Find the **Terminal** view nested to the **Console** view. If this view is not found, re-open it from **Window** > **Show View** > **Terminal**.
2. In the **Terminal** view, click the **Open a Terminal** icon . The **Launch Terminal** dialog opens (Figure 3.45).
3. Choose the **Serial Terminal** and configure the **Serial port** with **Baud rate**.

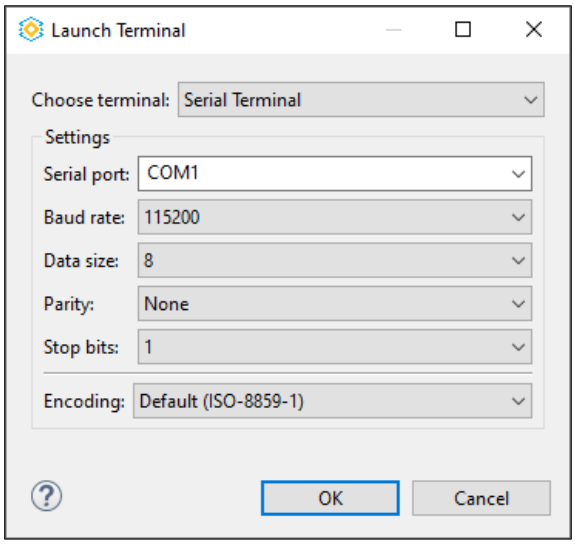


Figure 3.45. Launch Terminal Dialog 1

4. Click **OK**. A connection opens.
5. (Optional) Click the **Toggle Command Input** icon that adds an edit box to enter text (Figure 3.46).

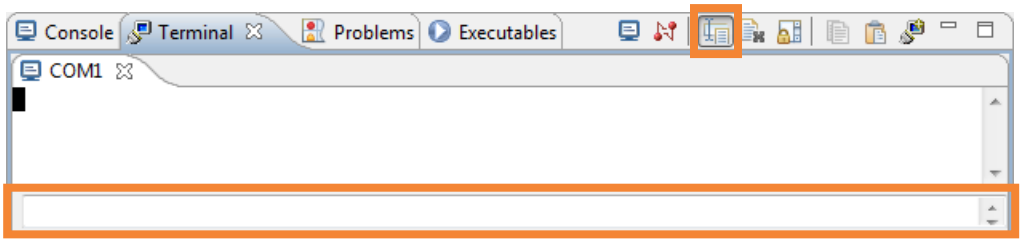


Figure 3.46. Terminal View

4. Lattice Propel Tutorial – Hello World

This Hello World project can be found from the template. The Hello World project provides a template of using hardware and software design with the minimal resource required.

Following this tutorial, you can easily create a hardware and software project. After that, you can run the project on your evaluation board.

The tutorial uses a MachXO3D breakout board for demonstration. It uses RS232 UART function. To enable RS232 UART function on the MachXO3D breakout board, the following reworks on the board are required.

1. Hardware reworks on MachXO3D breakout board.

   Short R14 and R15 using 0 Ω resistors, as shown in Figure 4.1.

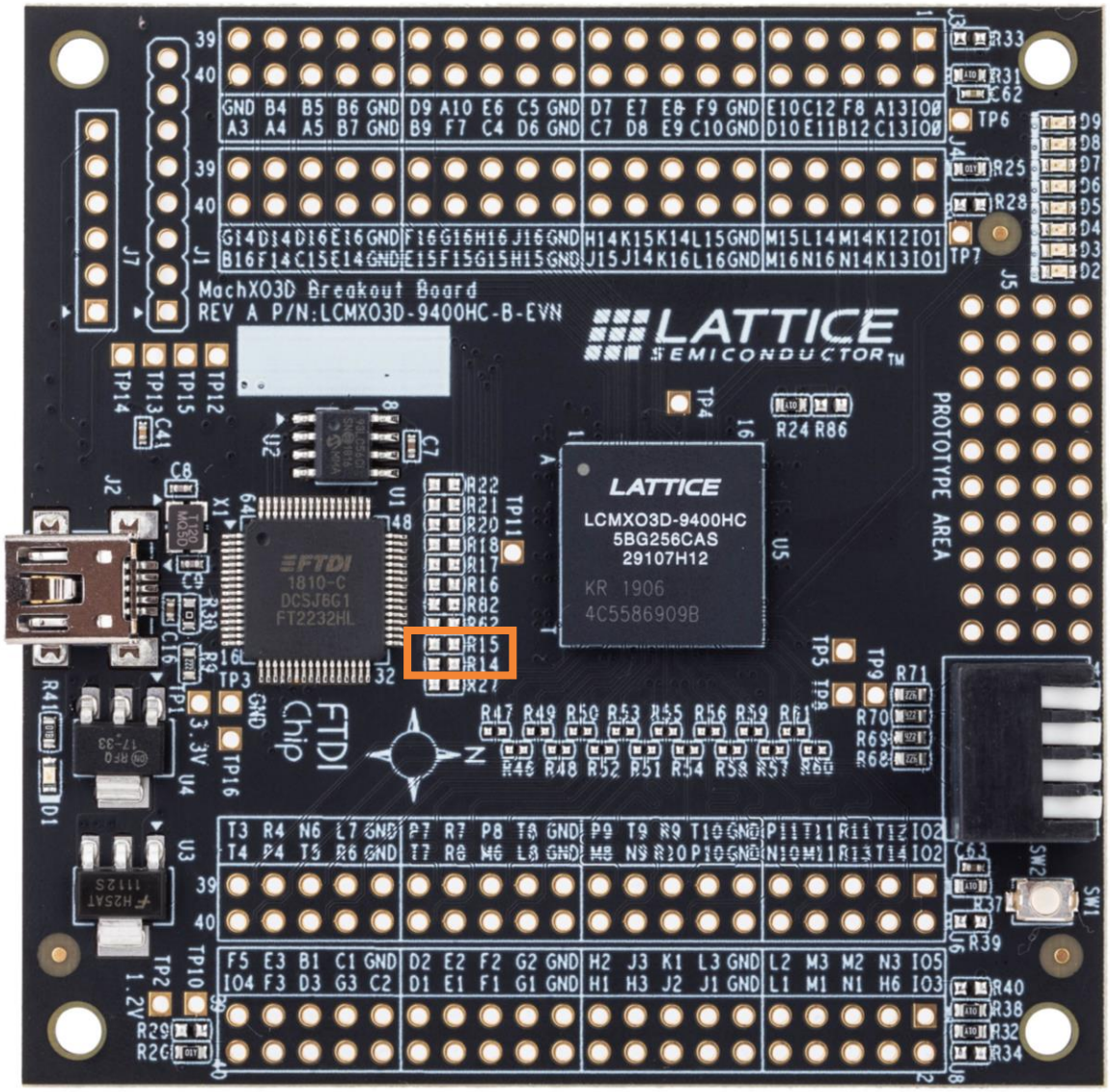


Figure 4.1. MachXO3D Breakout Board

2. Configure the FTDI device with FT_Prog software to enable the UART function.

   Connect the MachXO3D breakout board to host PC and power ON.

   Open the FT_Prog software. Select **DEVICES > Program**.

   Make sure Port B is configured as RS232 UART in Hardware and Virtual COM Port in Driver. See Figure 4.2.

   Select **DEVICES > Program** from the FT_Prog software again. From the opened Program Devices dialog, click **Program**.

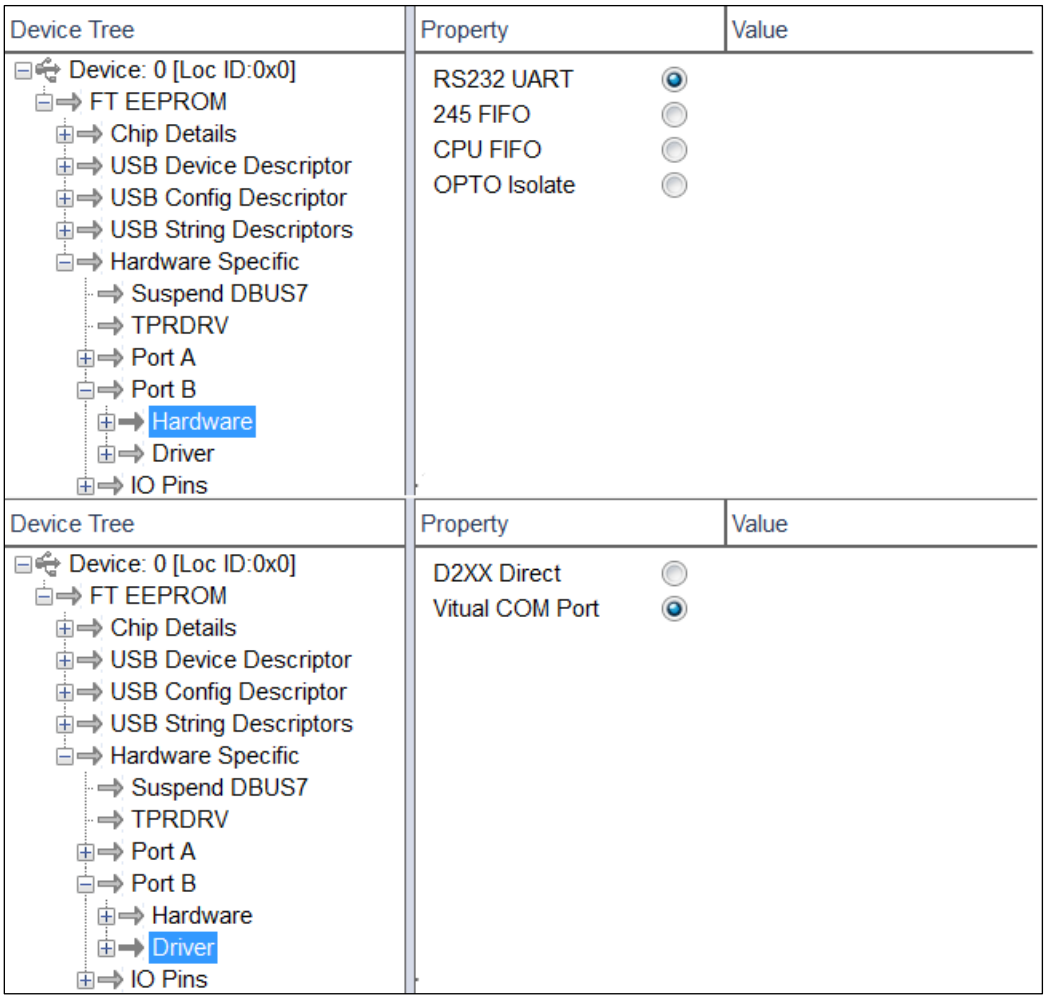


Figure 4.2 Configure the FTDI Device

3. All the reworks for the MachXO3D breakout board are completed.

4.1. Creating SoC Design Project and Preparing Hardware Design – Hello World

This section introduces how to create an SoC design project and prepare hardware design for the Hello World project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

To start an SoC Design Project from Lattice Propel SDK:

1. Choose **File** > **New** > **Lattice SoC Design Project**.
2. The **Create SoC Project** wizard opens (Figure 4.3). Enter a project name, such as **HelloWorldSoC**. Select the **RISC-V MC SoC Project** template.
3. Click **Finish.** An SoC project is created.

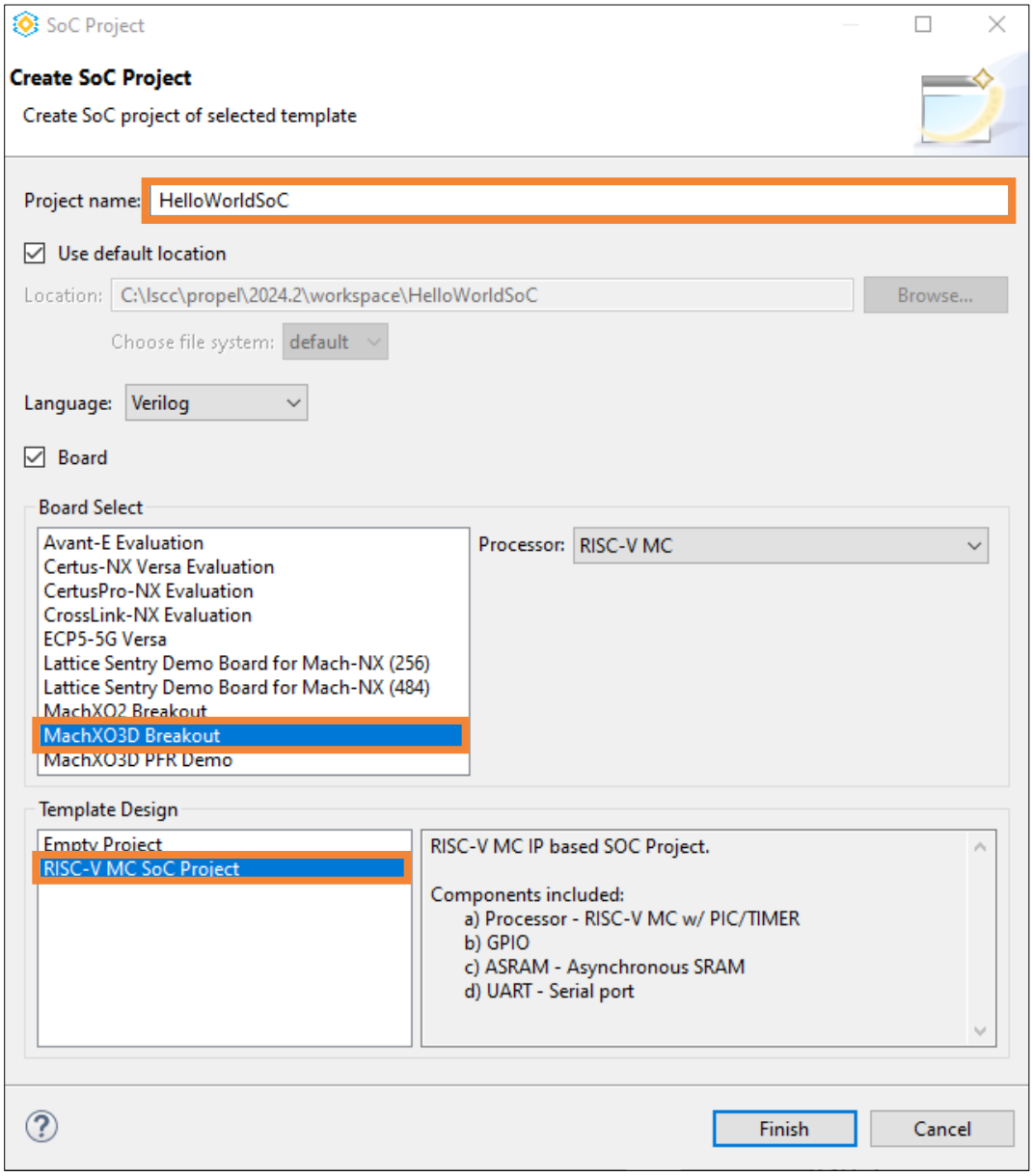


Figure 4.3. Create SoC Project Wizard 1

4. The created SoC project can be found in the workbench. Its design is opened and displayed in Lattice Propel Builder for review (Figure 4.4).

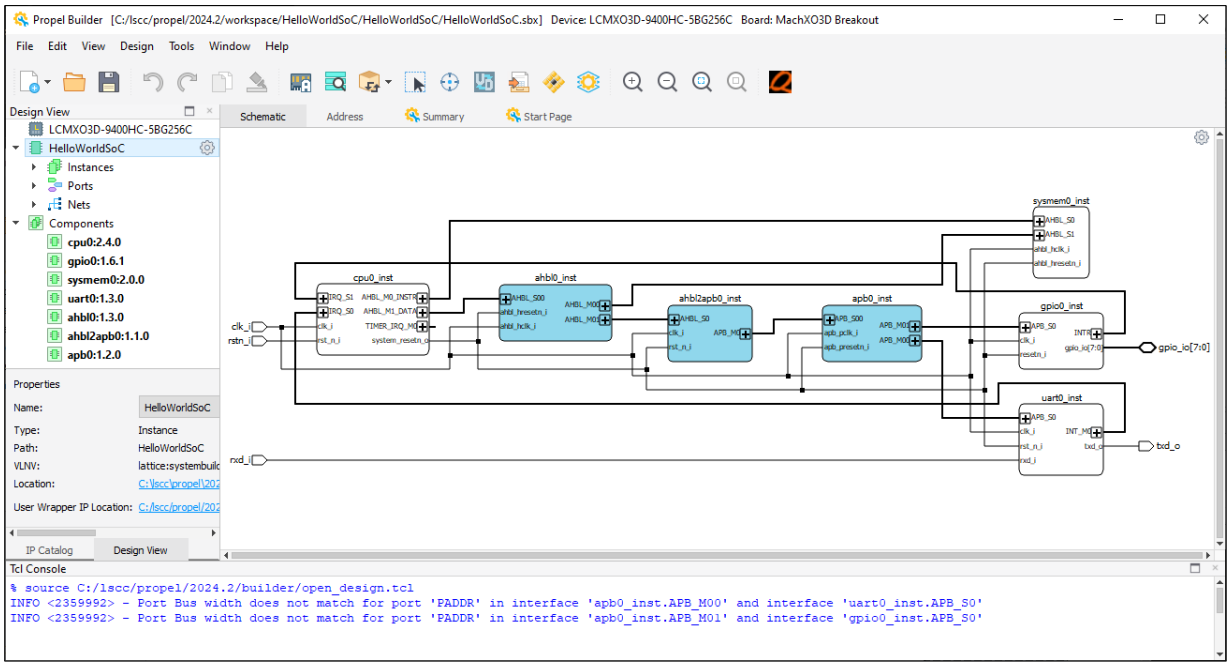


Figure 4.4. Lattice Propel Builder Window 2

5. (Optional) In Lattice Propel Builder, update the SoC design to set the preloaded software. Enable the checkbox for **Initialize Memory** for system_memory IP module from the **Initialization** area of the **General** tab, and set **Initialization File** (Figure 4.5) generated from the corresponding Hello World C project in the Creating Hello World C Project section.

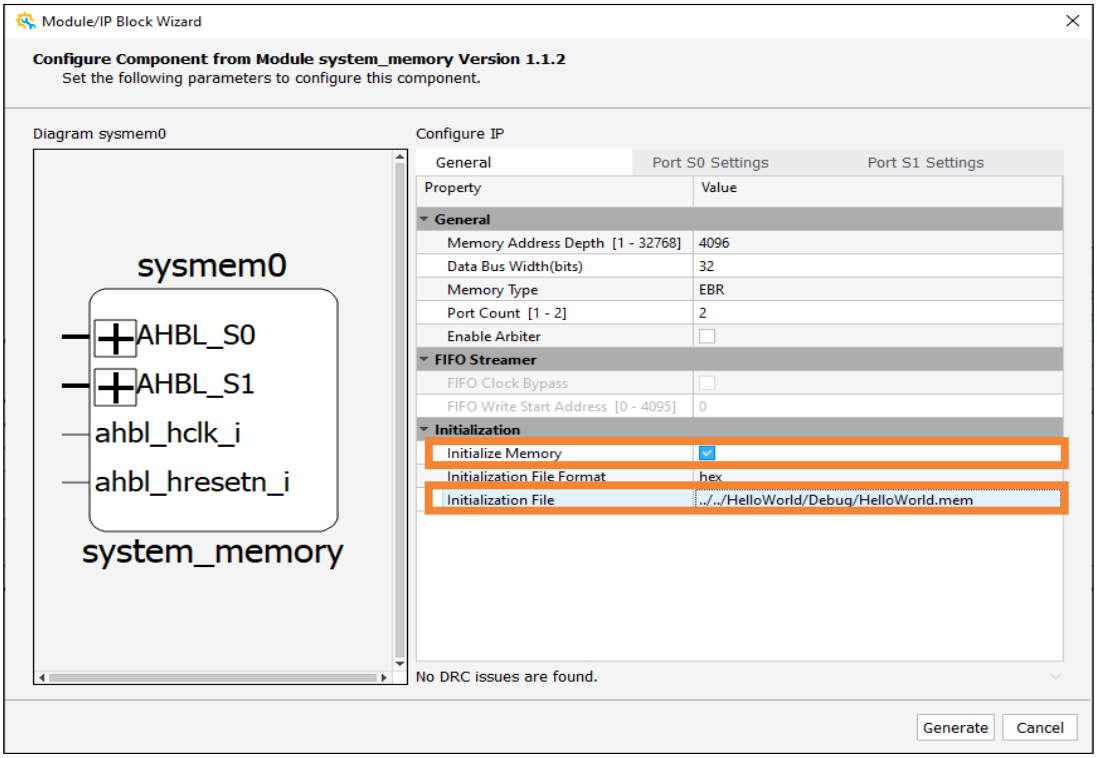


Figure 4.5. Configure Module System Memory 2

4.2. Launching Lattice Diamond Software

Launch the Lattice Diamond software from the created SoC project. To do that:

1. In Lattice Propel **Project Explorer** view, select the SoC project HelloWorldSoC.
2. Click the Lattice Diamond software icon on the toolbar. A Lattice Diamond project is created and thus opened automatically in the Lattice Diamond software.
3. Switch to **Process** view of the Lattice Diamond project and make sure **Bitstream File** or **JEDEC File** is checked in the **Export Files** section (Figure 4.6).
4. Choose **Process** > **Run**. Wait for generating programming file successfully. You can see a green checkmark before each successfully completed process.

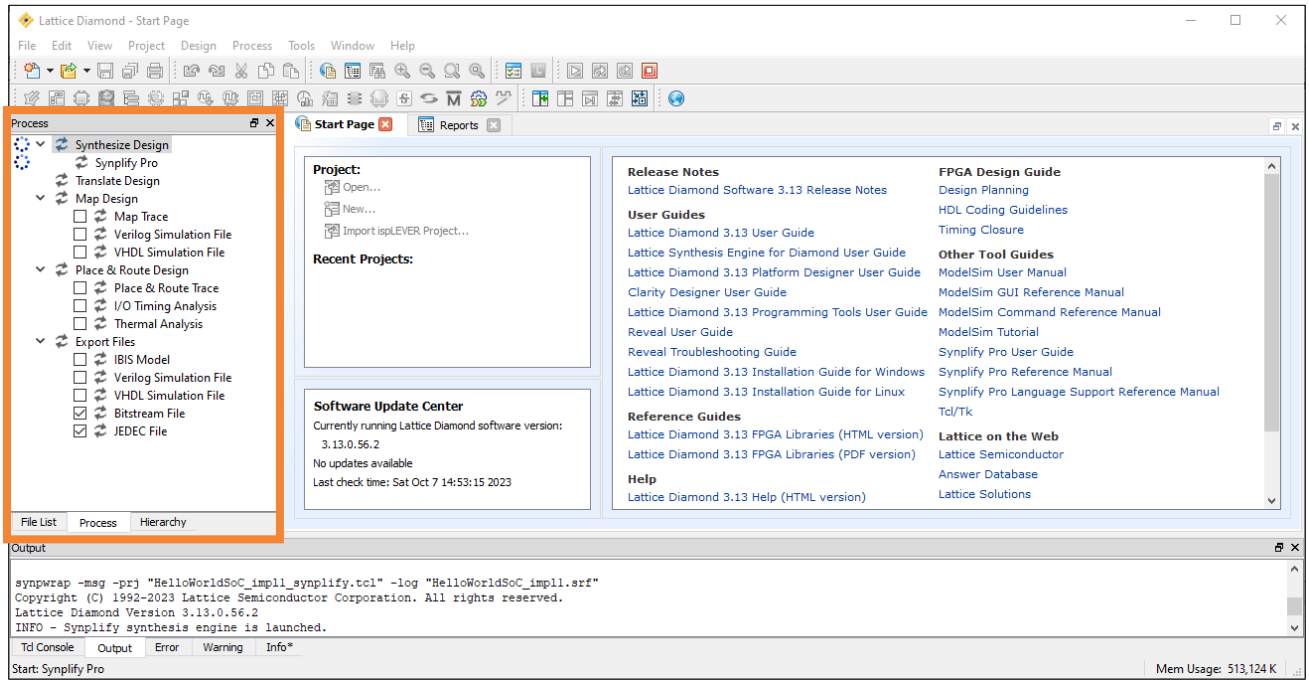


Figure 4.6. Generate Programming File

4.3. Programming the Target Device – Hello World

Once the programming file is exported successfully in the last section, it is ready to program the target device. Make sure the evaluation board is powered ON and connected correctly to the host PC before performing the following procedure.

1. Click the **Programmer** icon on the toolbar of the Lattice Diamond Project Explorer.
2. The **Programmer: Getting Started** dialog pops up (Figure 4.7). Click **OK**.

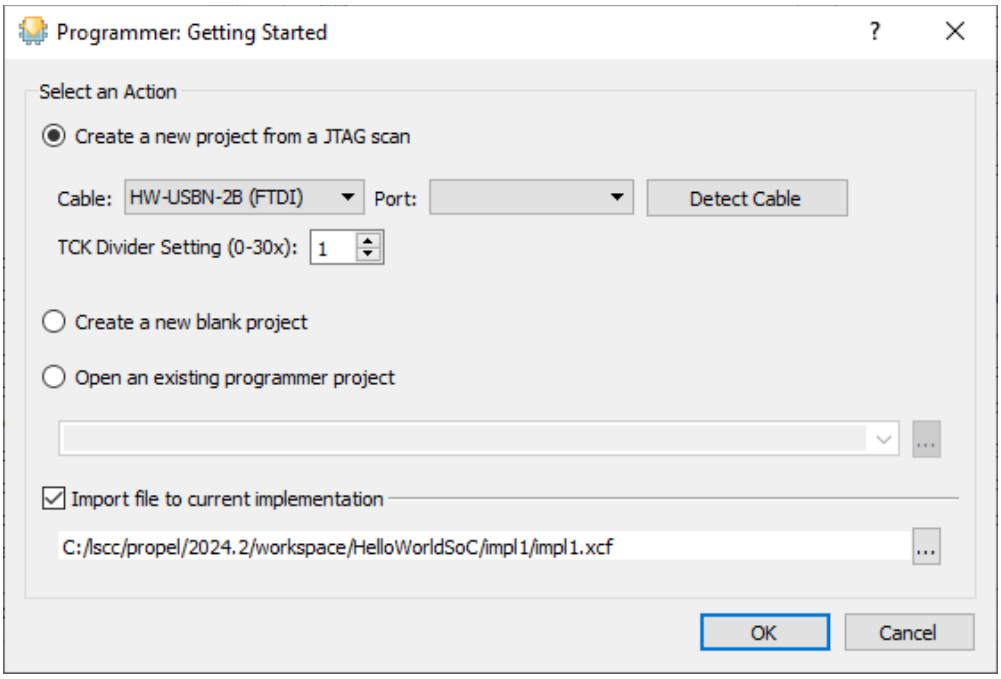


Figure 4.7. Programmer Getting Started Dialog

3. Review the **Device Family**, **Device**, **Operation**, and **File Name** in the Programmer window (Figure 4.8).

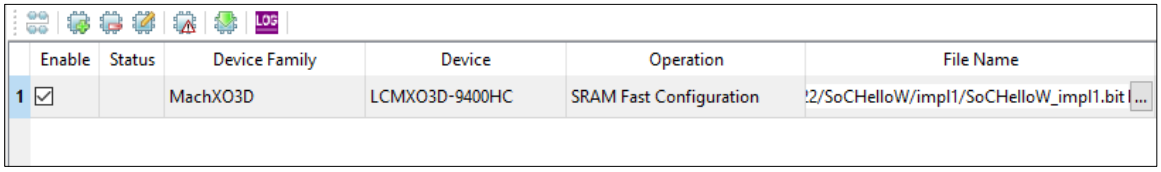


Figure 4.8. Programmer Window

4. Click the **Program** icon to download the programming data file to the device.

4.4. Creating Hello World C Project

Creating C project requires a system environment from SoC project as input.

1. In Lattice Propel **Project Explorer** view, select the SoC project HelloWorldSoC.
2. Choose **Project** > **Build Project**.

   System environment of the select SoC project is generated under the SoC project folder. Check the result from the **Console** view (Figure 4.9).

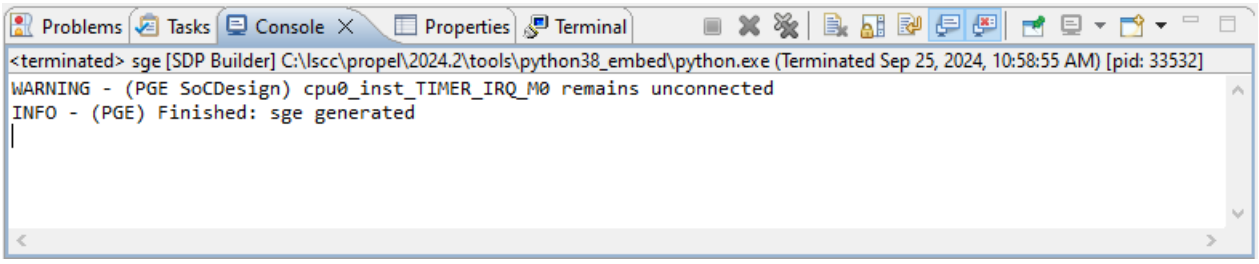


Figure 4.9. Build Result of HelloWorld SoC Project

3. Choose **File** > **New** > **Lattice C/C++ Project**.
   The C/C++ Project wizard opens with the **Load System and BSP** page (Figure 4.10).

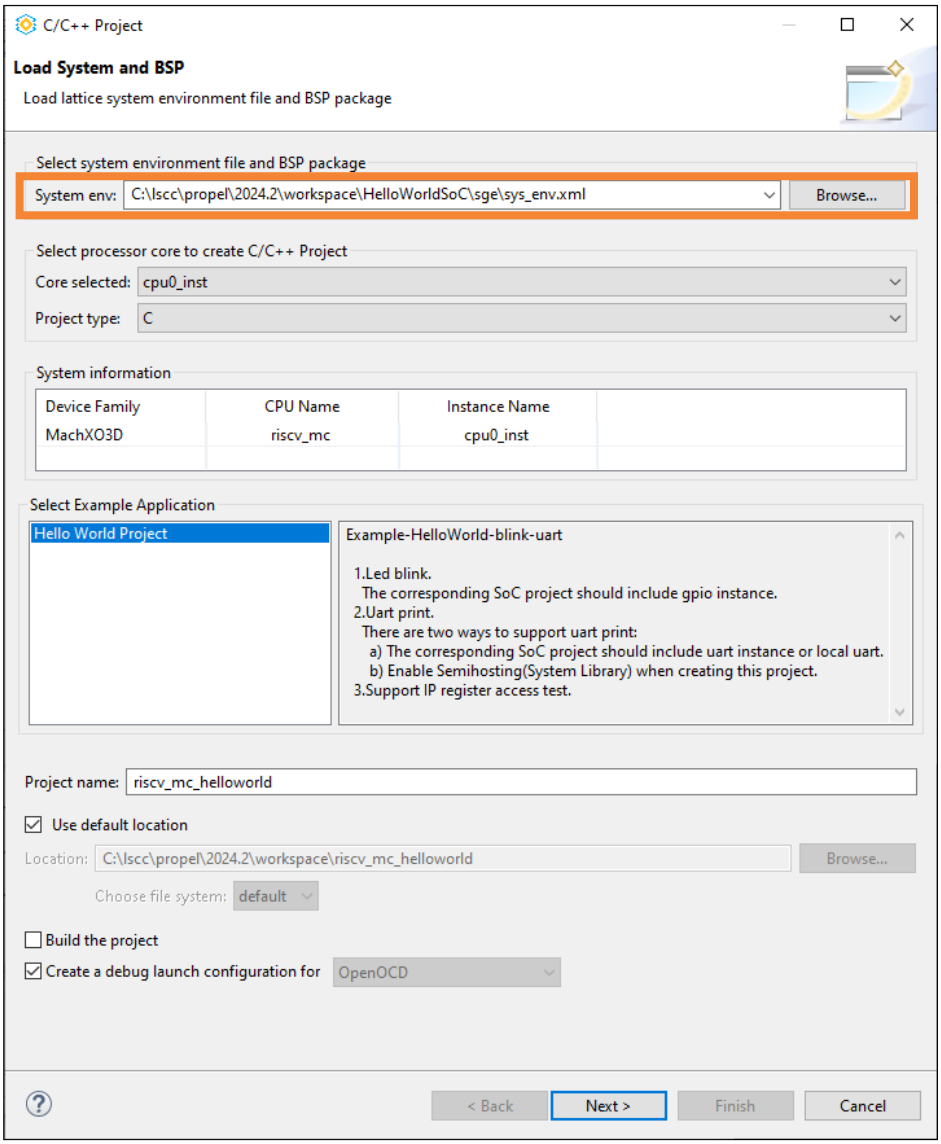


Figure 4.10. Load System and BSP Page 3

4. Select the system environment file just generated (Figure 4.10).
5. Change project name to HelloWorld (Figure 4.10). Click **Next**. Then click **Finish**.
   The C project is created and displayed in the workbench.
6. Choose **Project** > **Build Project**.
7. Check the build result from the **Console** view (Figure 4.11).

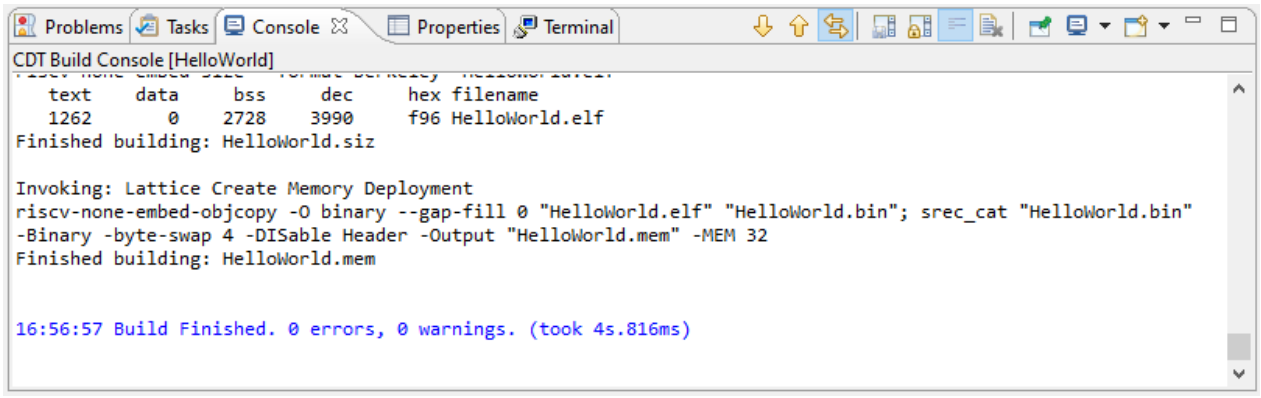


Figure 4.11. Build Result of HelloWorld C Project

4.5. Running Demo on MachXO3D Breakout Board – Hello World

1. Find the **Terminal** view nested to the **Console** view. If this view is not found, re-open it from **Window** > **Show View** > **Terminal**.
2. In the **Terminal** view, click the **Open a Terminal** icon .
3. Choose the **Serial Terminal** and configure the **Serial port** with **Baud rate** 115200 (Figure 4.12).

   Note: The serial port number depends on specific PC.

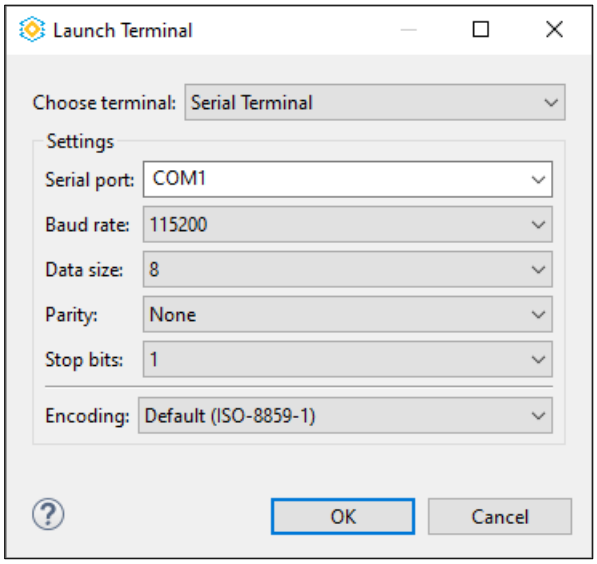


Figure 4.12. Launch Terminal Dialog 2

4. Click **OK**. A serial connection communicating with UART is ready.
5. In the **Project Explorer** view, select the C project, HelloWorld.
6. Choose **Run** > **Debug Configurations…**.
7. Double-click **GDB OpenOCD Debugging** to create a new launch configuration (Figure 4.13).
8. Click the **Debug** button.

   Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then starting the debugging session.

   Note: This demo uses the default debug configuration options.

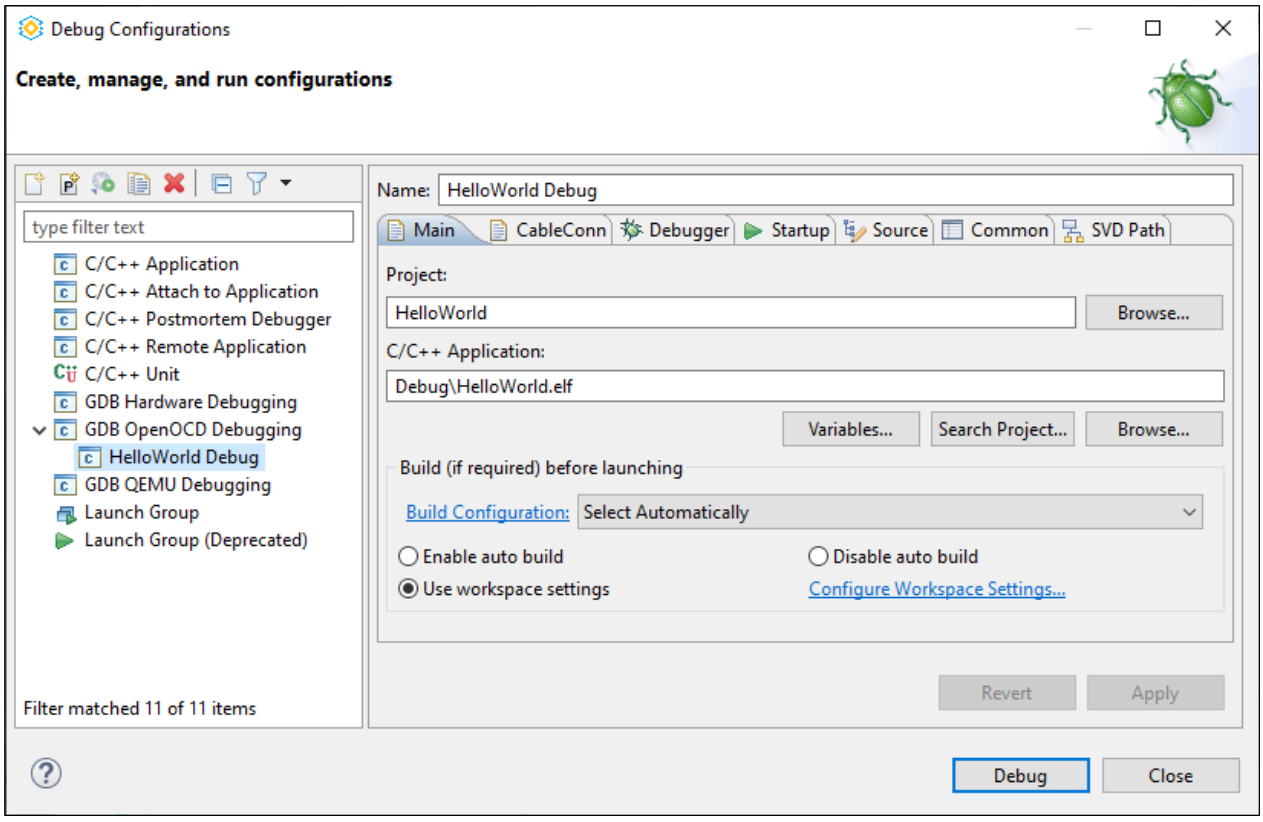


Figure 4.13. Debug Configurations Dialog 2

9. Click the **Resume** icon on the toolbar. The serial terminal outputs *Hello RISC-V world!* (Figure 4.14).

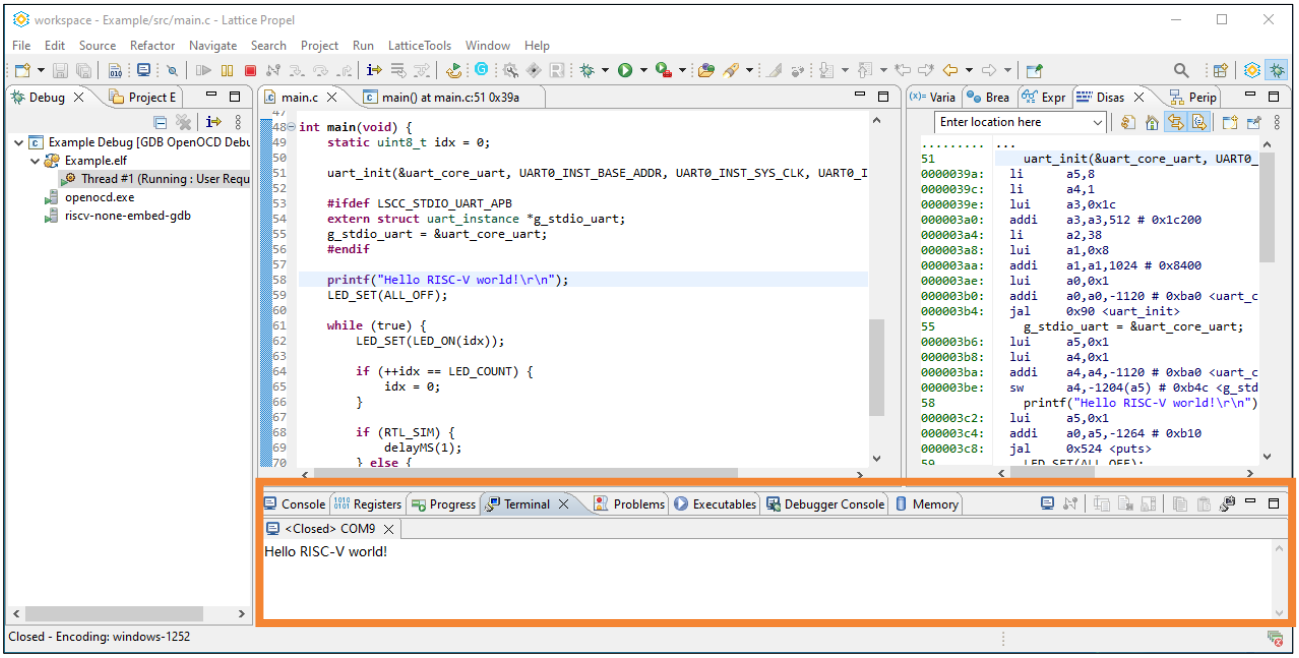


Figure 4.14. Run Result of Hello World Project

5. Lattice Propel Tutorial – FreeRTOS

The FreeRTOS templates are based on FreeRTOS 202210.01 LTS. Refer to https://www.freertos.org/ for more details.

There are two FreeRTOS templates, FreeRTOS-LTS-minimal and FreeRTOS-LTS-PMP-Blinky.

FreeRTOS-LTS-minimal is the same as the opensource FreeRTOS, except choosing RISC-V portable code. You can start self-development from this template.

FreeRTOS-LTS-PMP-Blinky is a sample project, which demonstrates how to create tasks, create queues, perform queue data send and receive, and how to use soft timer.

This tutorial uses a CertusPro™-NX Evaluation Board with RISC-V RX SoC Project for demonstration.

Example C project is FreeRTOS-LTS-PMP-Blinky.

5.1. Preparing the Hardware and Programming the Target Device – FreeRTOS

This section introduces how to prepare the hardware and program the target device for the FreeRTOS project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

1. Create RISC-V RX SoC Project with CertusPro-NX Evaluation Board (Figure 5.1). Click **Finish.**
2. Select **Project** > **Generate**.
3. Generate the programing file by running Lattice Radiant software in this SoC Project. Refer to the SoC Project Design Flow section for detailed steps.
4. Program the programing file to the target device, CertusPro-NX Evaluation Board.

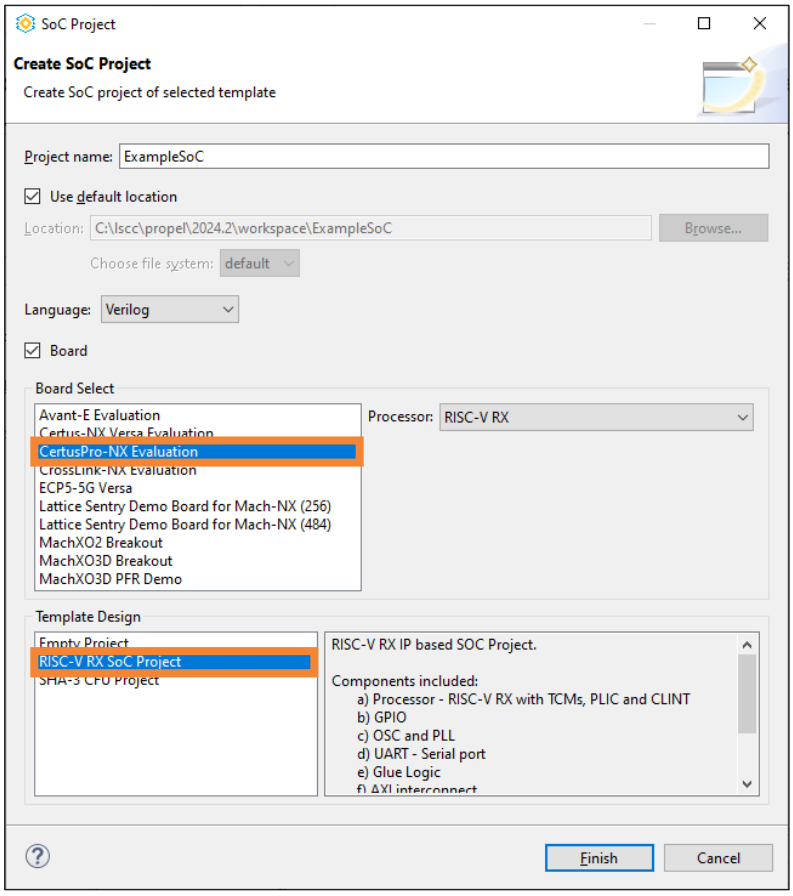


Figure 5.1. Create SoC Project Wizard 2

5.2. Creating FreeRTOS-LTS-PMP-Blinky C Project

1. Select **File** > **New** > **Lattice C/C++ Project**.
   The C/C++ Project wizard opens with the **Load System and BSP** page (Figure 5.2).

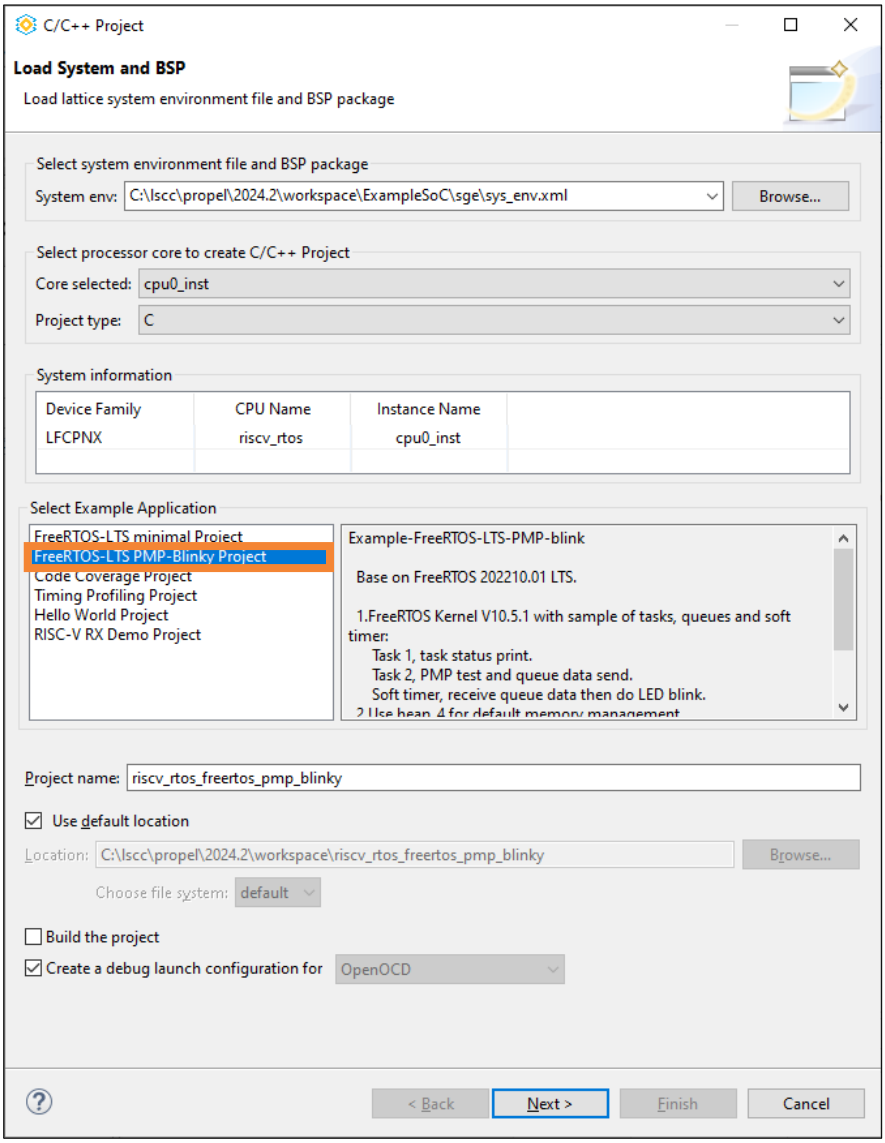


Figure 5.2. Load System and BSP Page 4

2. In **System env** field, select the system environment file from the RX core SoC project just generated.
3. Select **Project** > **Build Project**.
4. Check the build result from the **Console** view (Figure 5.3).

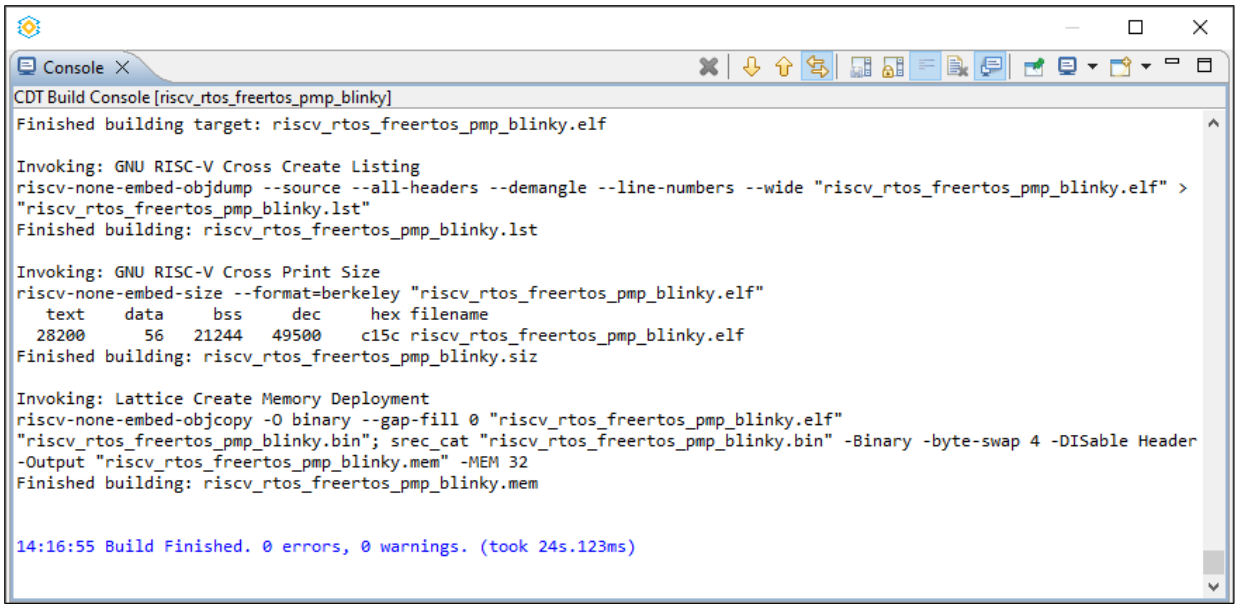


Figure 5.3. Build Console 1

5.3. Running FreeRTOS C Project

1. In the **Project Explorer** view, select the C project just created, riscv_rtos_freertos_pmp_blinky.
2. Choose **Run** > **Debug Configurations…**.
3. Choose riscv_rtos_freertos_pmp_blinky in **GDB OpenOCD Debugging** (Figure 5.4).
4. Click the **Debug** button.

   Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then starting the debugging session.

   Note: This demo uses the default debug configuration options.
5. Find the **Terminal** view nested to the **Console** view. If this view is not found, re-open it from **Window** > **Show View** > **Terminal**.
6. In the **Terminal** view, click the **Open a Terminal** icon .
7. Choose the **Serial Terminal** and configure the **Serial port** with **Baud rate** 115200 (Figure 5.5).

   Note: The serial port number depends on the specific PC.
8. Click **OK**. A serial connection communicating with UART is ready.
9. Click the **Resume** icon on the toolbar. The serial terminal outputs running logs (Figure 5.6).

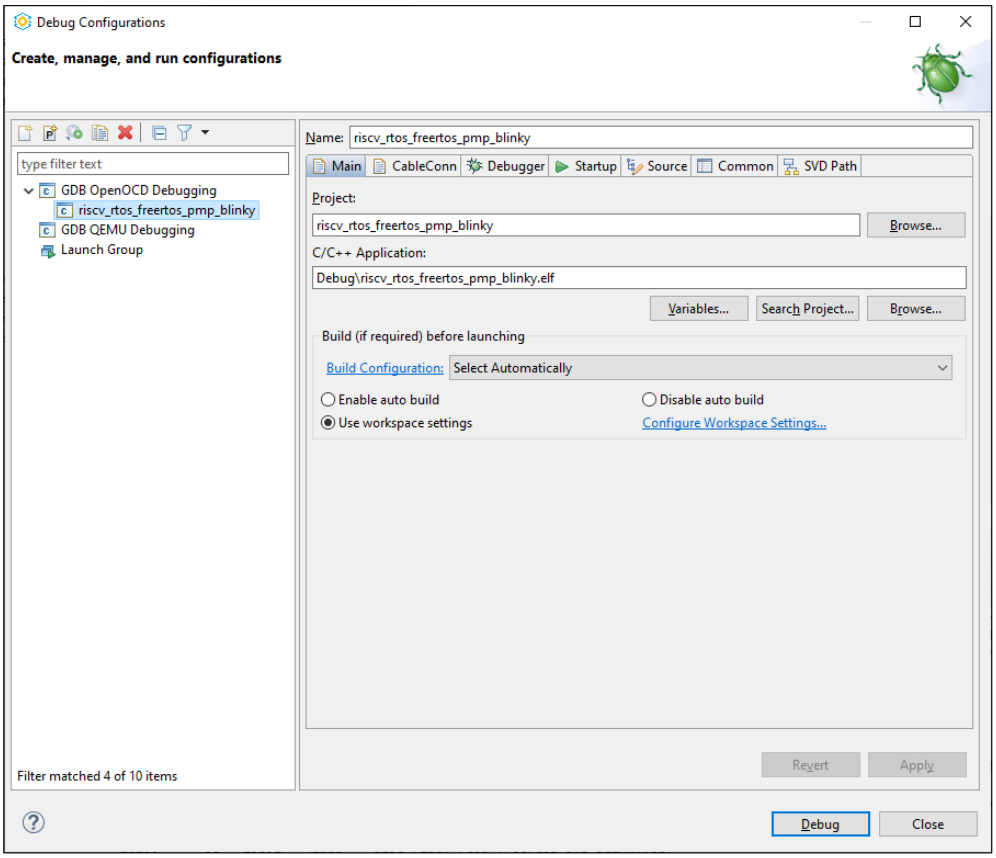


Figure 5.4. Debug Configurations Dialog 3

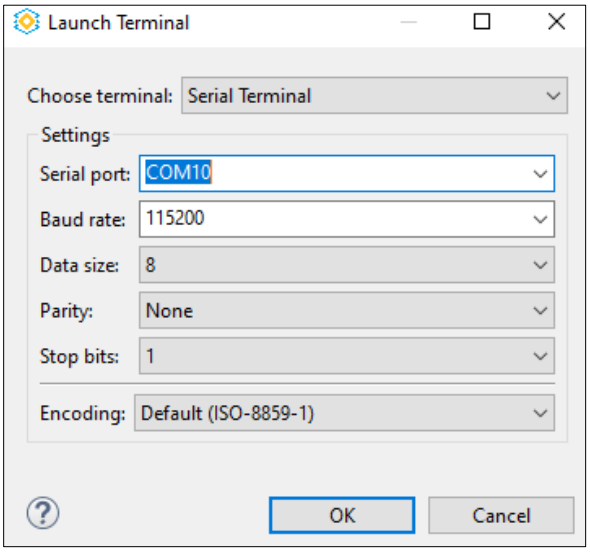


Figure 5.5. Launch Terminal Dialog 3

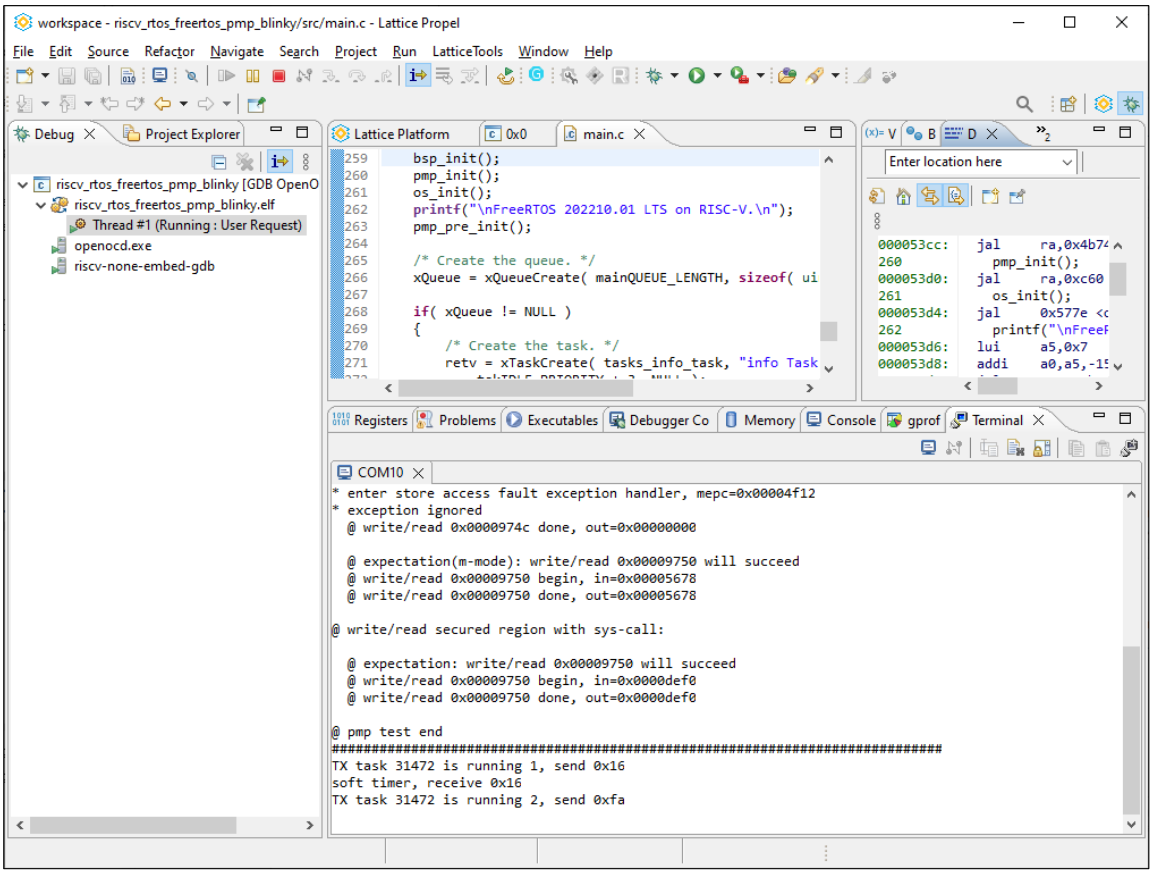


Figure 5.6. Running/Debugging Windows 1

6. Lattice Propel Tutorial – Code Coverage

Code coverage is about validating the number of lines of codes executed under a test process. This, in turn, helps in analyzing how well and comprehensively a software application is being tested. In other words, it is the quantitative measurement of the percentage or degree of executed source code of software application during testing. Code coverage enables you to gauge the completeness of your test cases more accurately.

The code coverage function is supported in the Code Coverage Project.

The RISC-V RX Demo C project and FreeRTOS-LTS-PMP-blinky C project also support this function, but you need to enable this function when creating the project.

This tutorial uses a CertusPro-NX Evaluation Board with the RISC-V RX SoC project for demonstration.

The example C project is the Code Coverage project.

Notes:

- This example C project needs more system memory. The suggested value is 0x20000.
- Semihosting is required for writing files back.

6.1. Preparing the Hardware and Programming the Target Device – Code Coverage

This section introduces how to prepare the hardware and program the target device for the Code Coverage project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

1. Create a RISC-V RX SoC project with the CertusPro-NX Evaluation Board (Figure 5.1). Click **Finish.**
2. Lattice Propel Builder automatically opens. In Lattice Propel Builder, double-click tcm0_inst in the **Schematic** view, set a larger value for Address Depth both in Port S0 Settings (Figure 6.1) and Port S1 Settings (Figure 6.2). Click **Generate**.
3. Select **Project** > **Generate**.
4. Generate the programing file by running Lattice Radiant software in this SoC project. Refer to the SoC Project Design Flow section for detailed steps.
5. Program the programing file to the target device, CertusPro-NX Evaluation Board.

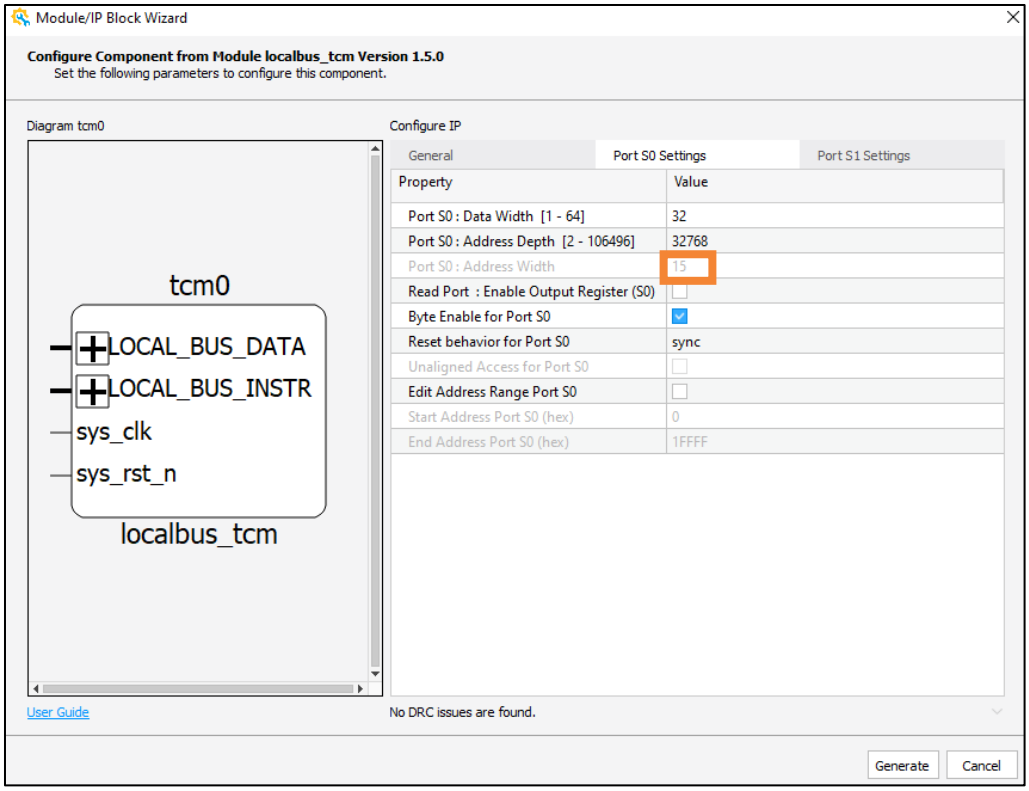


Figure 6.1. tcm0_inst Port S0 Address Depth Settings 1

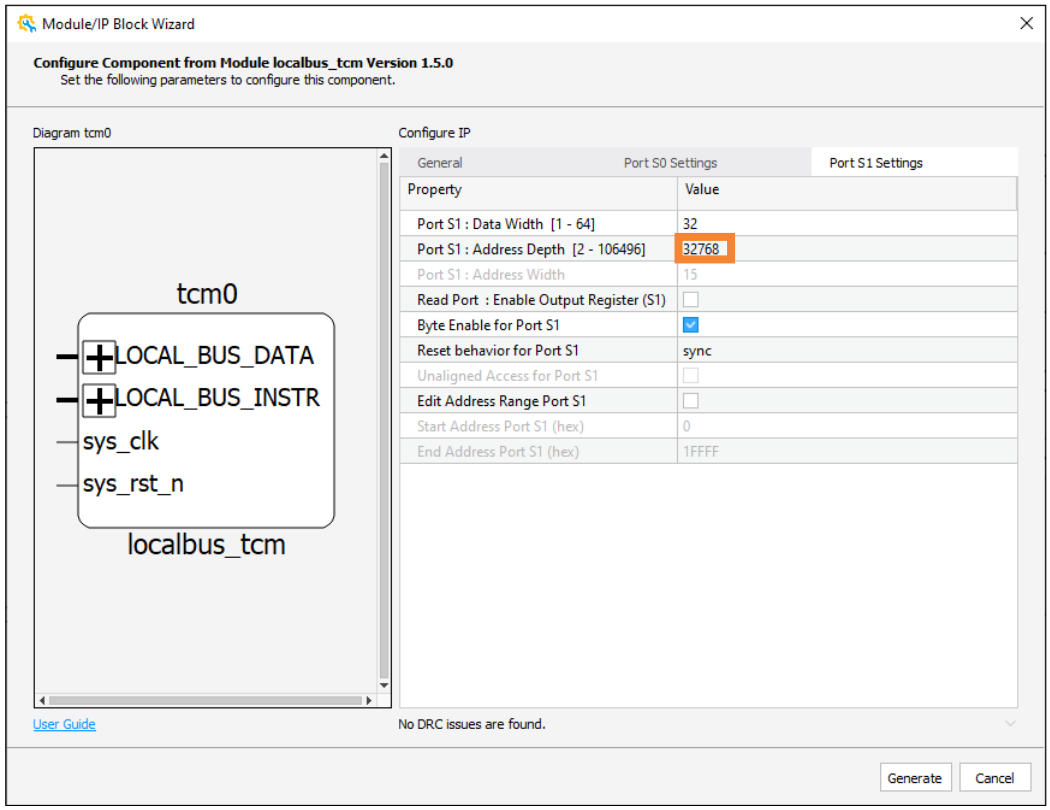


Figure 6.2. tcm0_inst Port S1 Address Depth Settings 1

6.2. Creating RX Demo C Project

1. The RX Demo C project can be created through the **Load System and BSP** page (Figure 6.3).

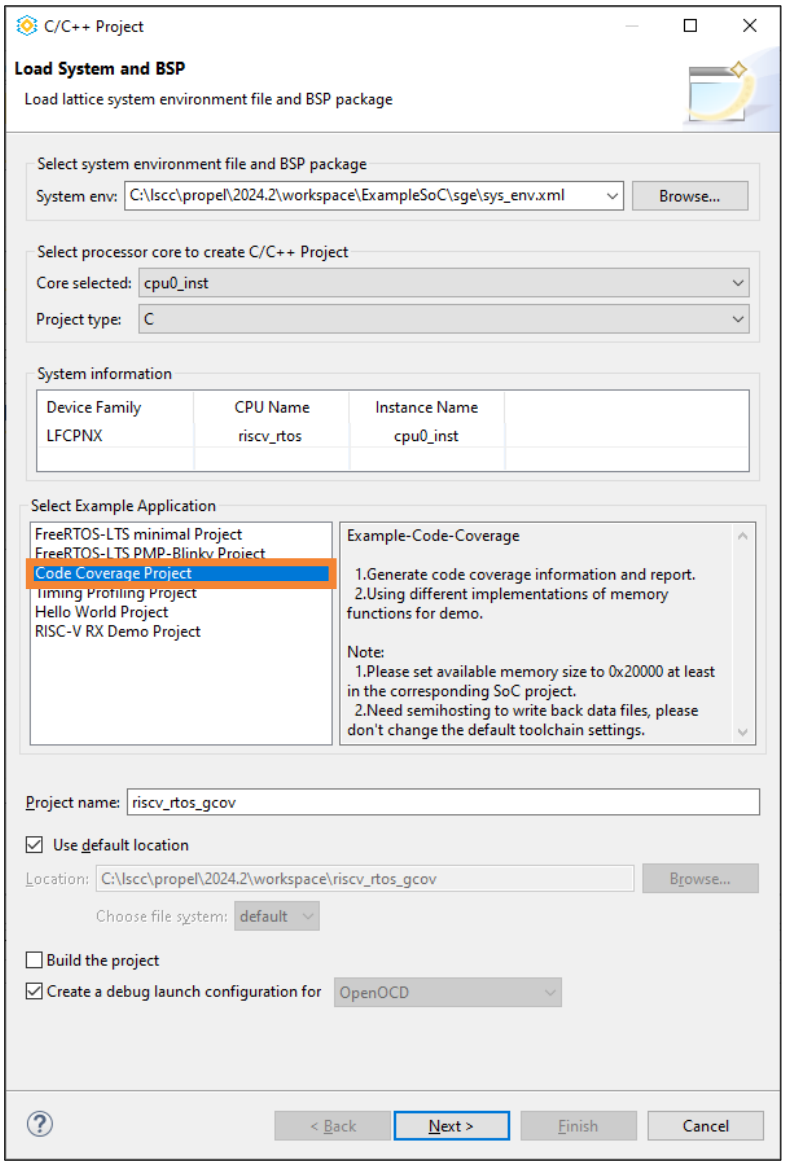


Figure 6.3. Load System and BSP Page 5

2. In **System env** field, select the system environment file from the RX core SoC project just generated.
3. Select **RISC-V RX Demo Project** and check the project name.
4. Click **Next**. The **Lattice Toolchain Setting** dialog opens. Keep the default settings in Lib Settings tab.
5. Click the **C/C++ Compiler** tab (Figure 6.4). The checkbox **Generate gcov information** is selected in this project by default. If you need to use the code coverage function in RISC-V RX Demo C Project and FreeRTOS-LTS-PMP-blinky C Project, you should select this checkbox manually.
6. Click **Finish**.

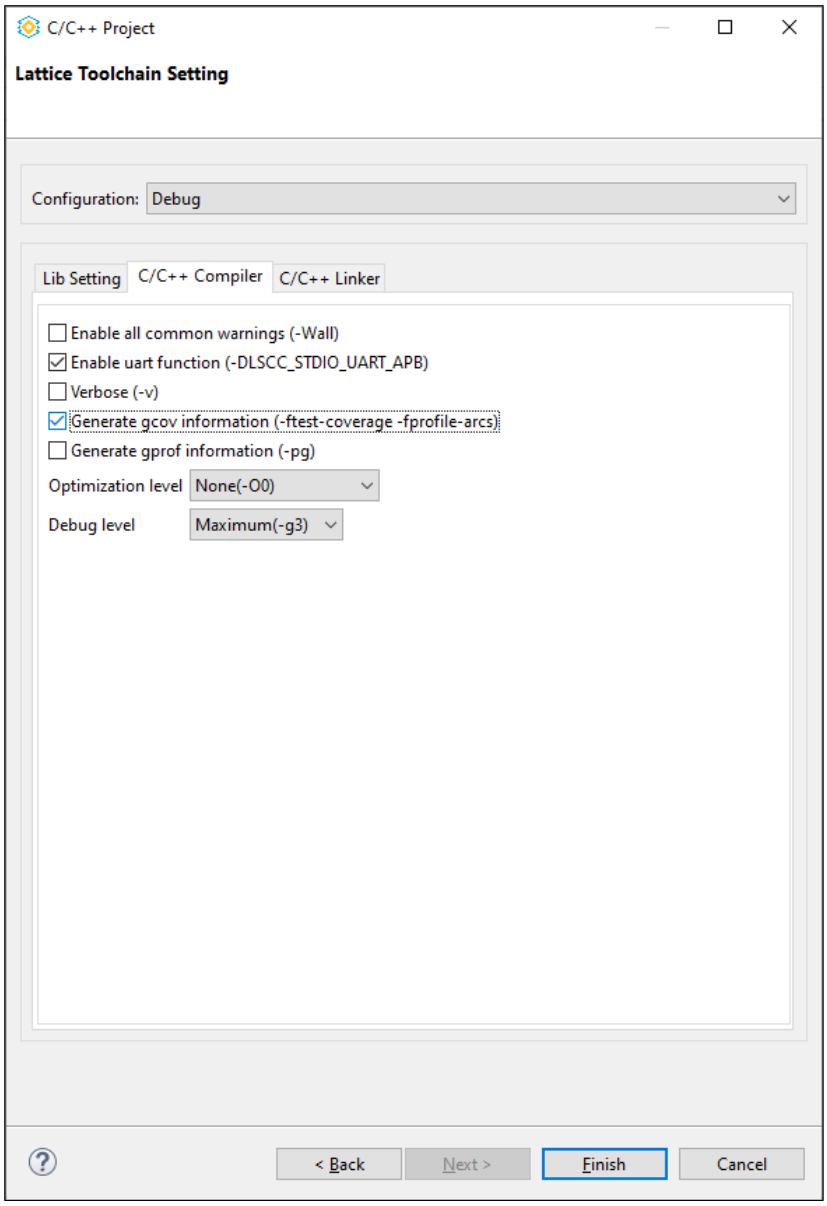


Figure 6.4. C/C++ Compiler

6.3. Compiling and Running Demo – Code Coverage

6.3.1. Compiling C Project – Code Coverage

1. In the **Project Explorer** view, select the C project, riscv_rtos_gcov.
2. Select **Project** > **Build Project**.

6.3.2. Running Demo – Code Coverage

1. In the **Project Explorer** view, select the C project, riscv_rtos_gcov.
2. Select **Run** > **Debug Configurations…**.
3. Select riscv_rtos_gcov in **GDB OpenOCD Debugging** (Figure 6.5).
4. Click the **Debug** button.

Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then starting the debugging session.

Note: This demo uses the default debug configuration options.

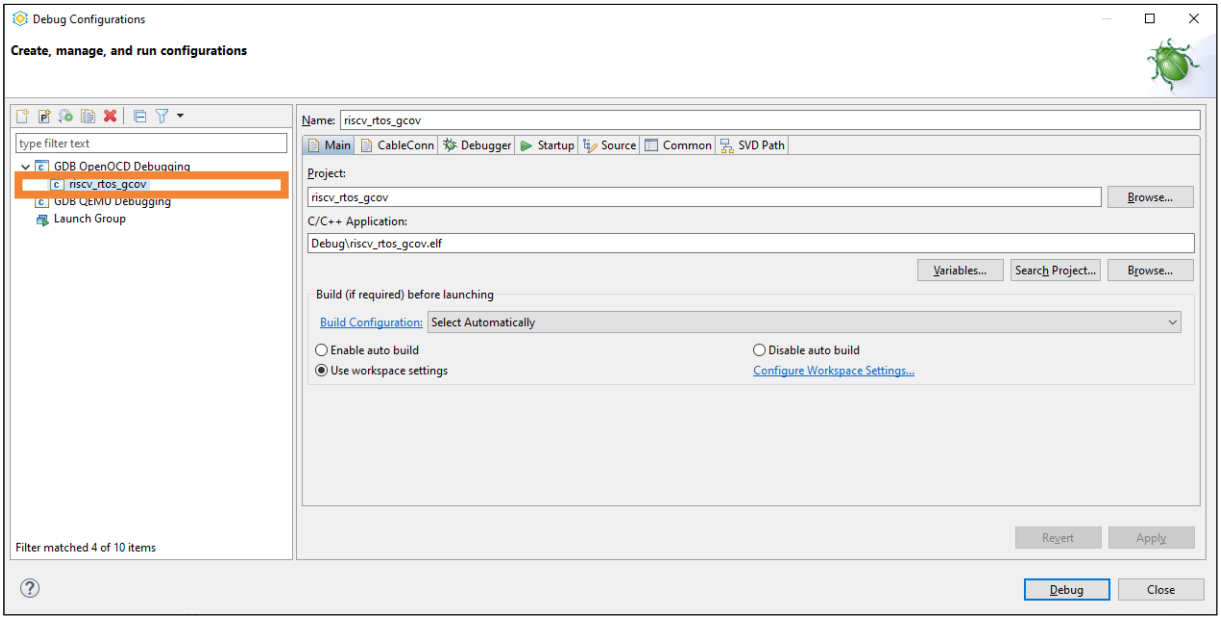


Figure 6.5. Debug Configurations Dialog 4

5. Select **Run** > **Resume**. Then the console output logs.

   Wait for a few minutes, until the console logs: *do coverage data dump .. Please double click one of the .gcda/.gcno files (found from Debug/src) to see the report* (Figure 6.6).

6. Select **Run** > **Terminate** to stop running.

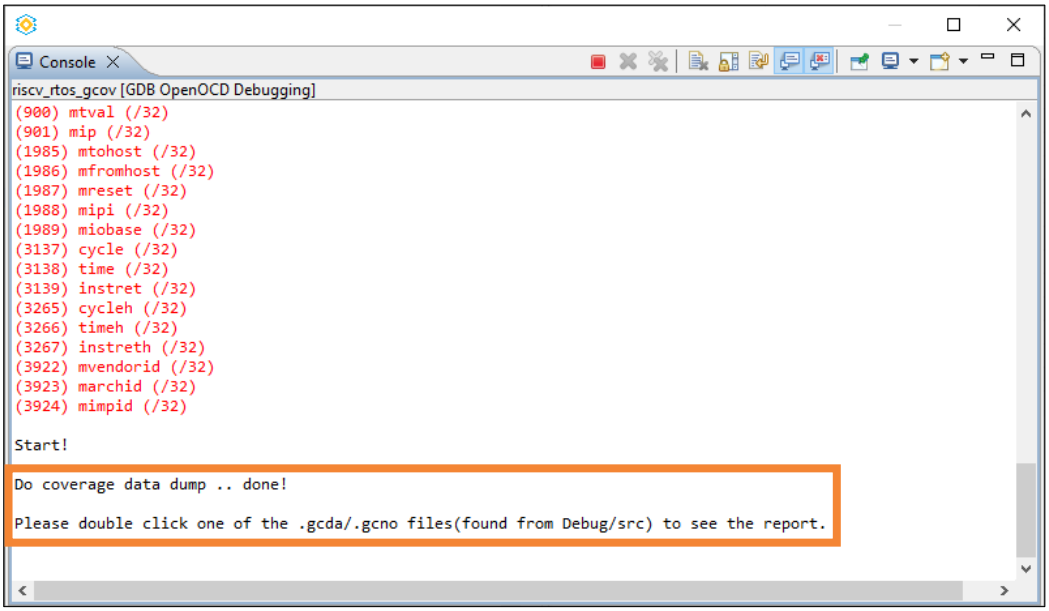


Figure 6.6. Console Logs 1

6.3.3. Display Coverage Information

1. Check the coverage files (Figure 6.7).
2. Double click one of the coverage files and click **OK** (Figure 6.8).
3. Wait for a few seconds, the coverage information is displayed (Figure 6.9).
   For detailed usage of the code coverage, refer to Help - Eclipse Platform and enter Gcov View in the Search box.

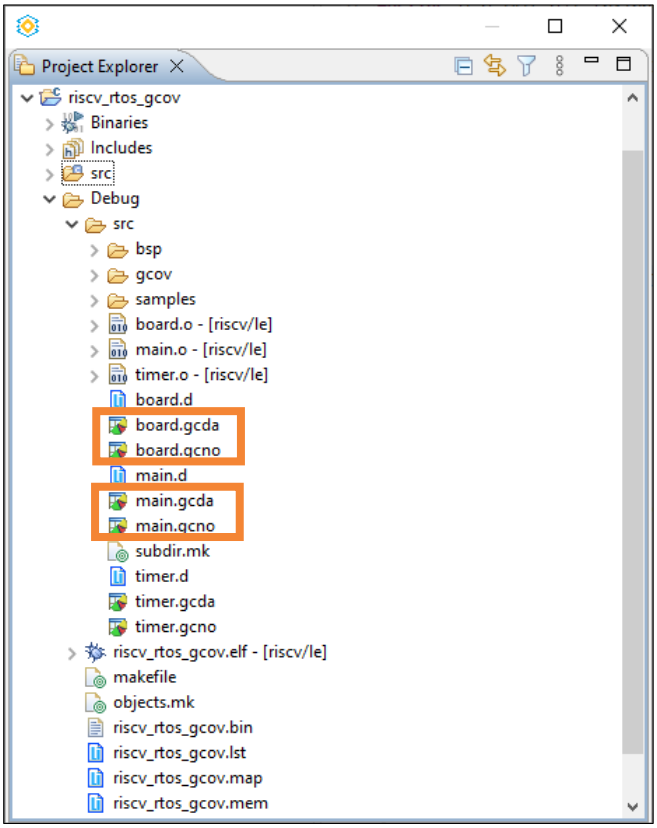


Figure 6.7. Coverage Files

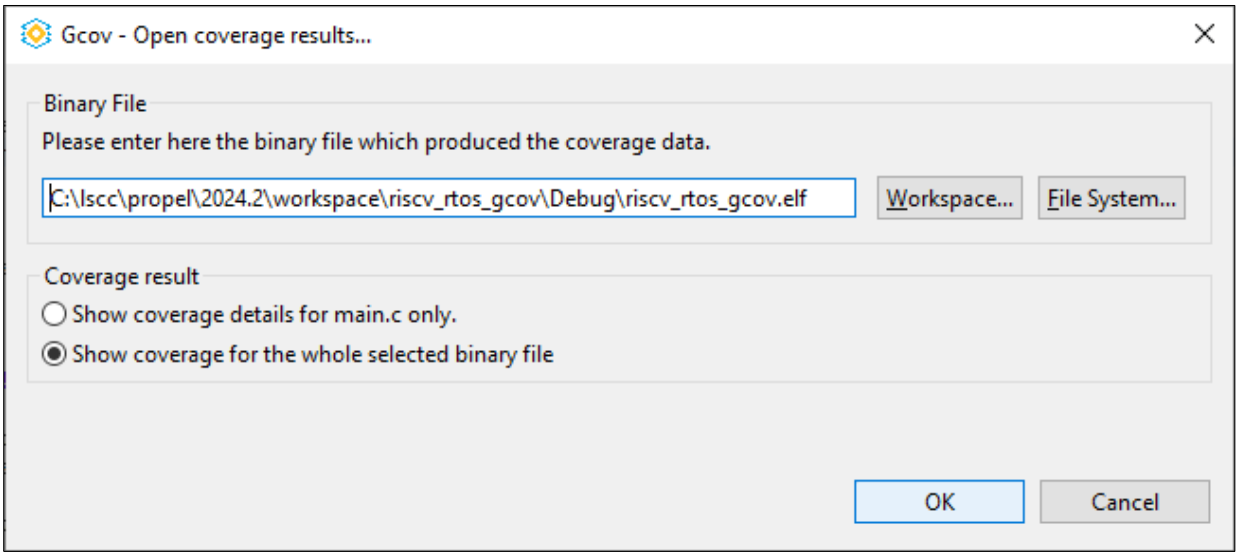


Figure 6.8. Open Coverage Results

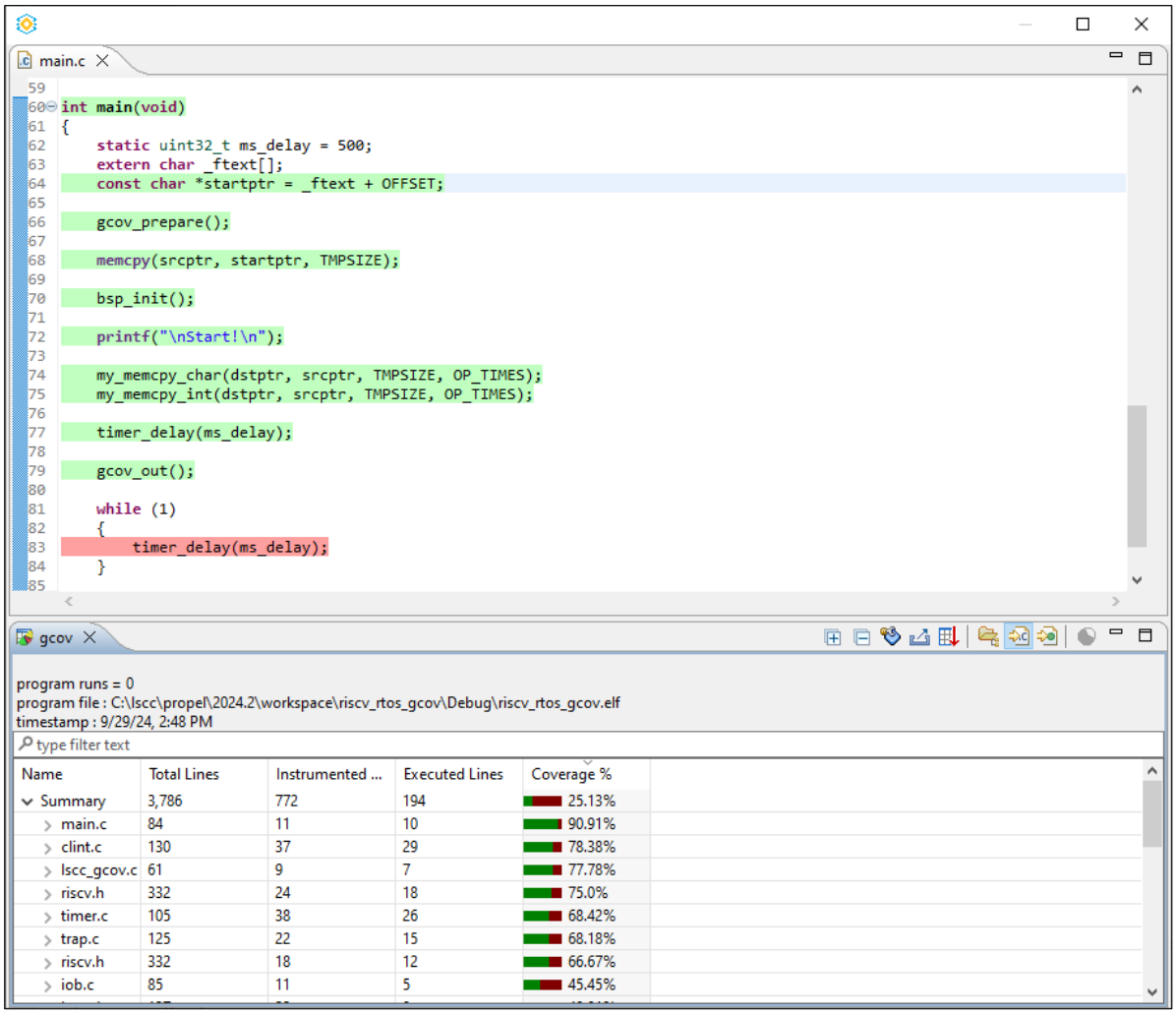


Figure 6.9. Code Coverage Information

6.4. Enabling Code Coverage for Existing C Project

1. In the **Project Explorer** view, select the existing C project.
2. Choose **Project** > **Properties** > **C/C++ Build** > **Settings.**
3. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_COVERAGE (Figure 6.10).
4. Select **GNU RISC-V Cross C Compiler** > **Miscellaneous**. Add the compiler flag, -fprofile-arcs -ftest-coverage (Figure 6.11).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgcov (Figure 6.12).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Add the linker flag, --defsym=_HEAP_SIZE=0x1000 (Figure 6.13).

   Note: The 0x1000 is a suggested value and it can be changed to a proper value if necessary.
7. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags in the label, Other linker flags. Make sure the linker flag, --oslib=semihost, is supported (Figure 6.14).
8. Click **Apply and Close**.
9. Add the line `scoverage_init();` to the head of code section. Add the line `scoverage_dump();` to the end of code section. You can refer to the sample template.
10. Rebuild this C project.

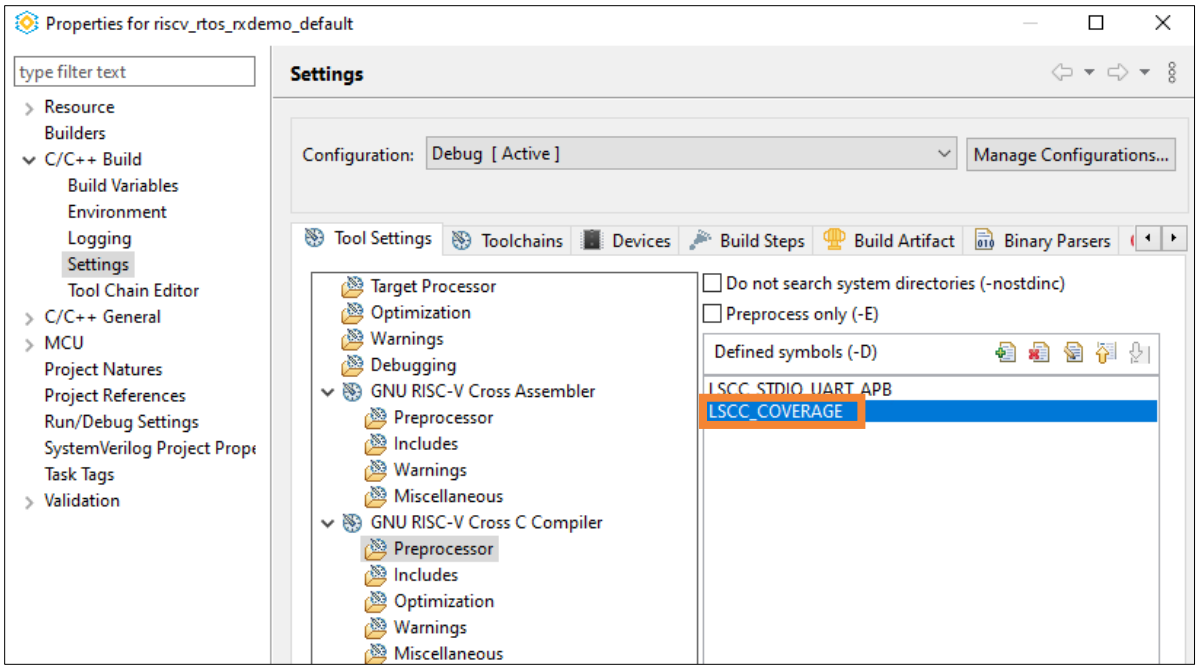


Figure 6.10. LSCC_COVERAGE Symbol

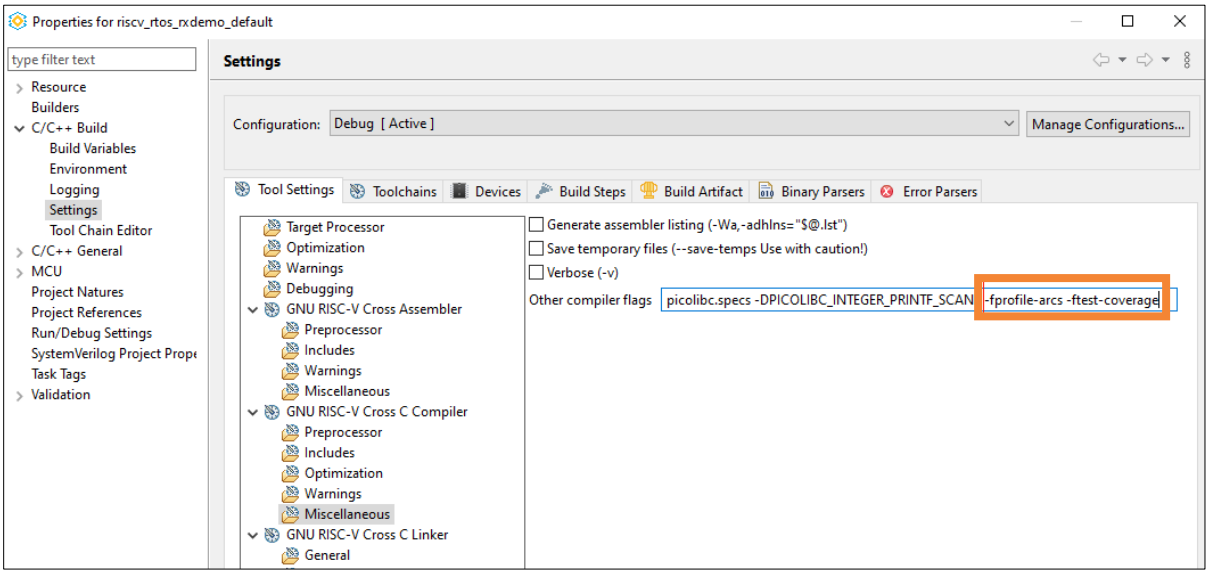


Figure 6.11. -fprofile-arcs -ftest-coverage Compiler Flag

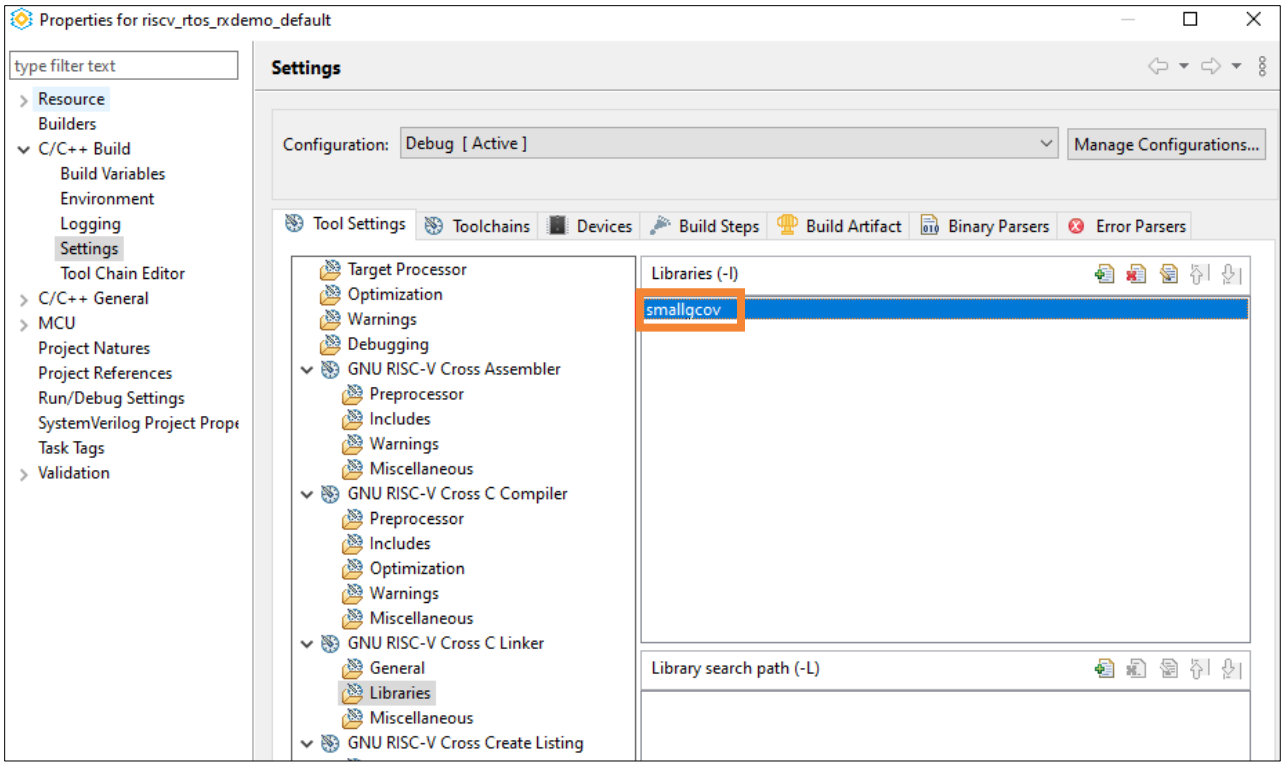


Figure 6.12. smallgcov Library

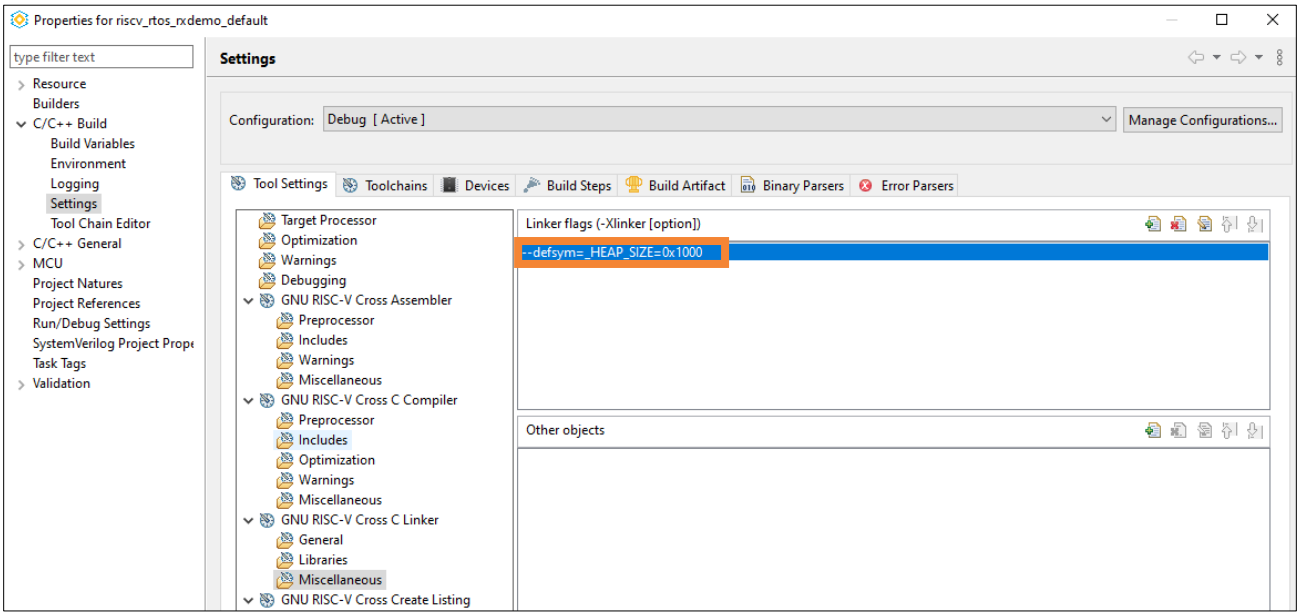


Figure 6.13. --defsym=_HEAP_SIZE=0x1000 Linker Flag

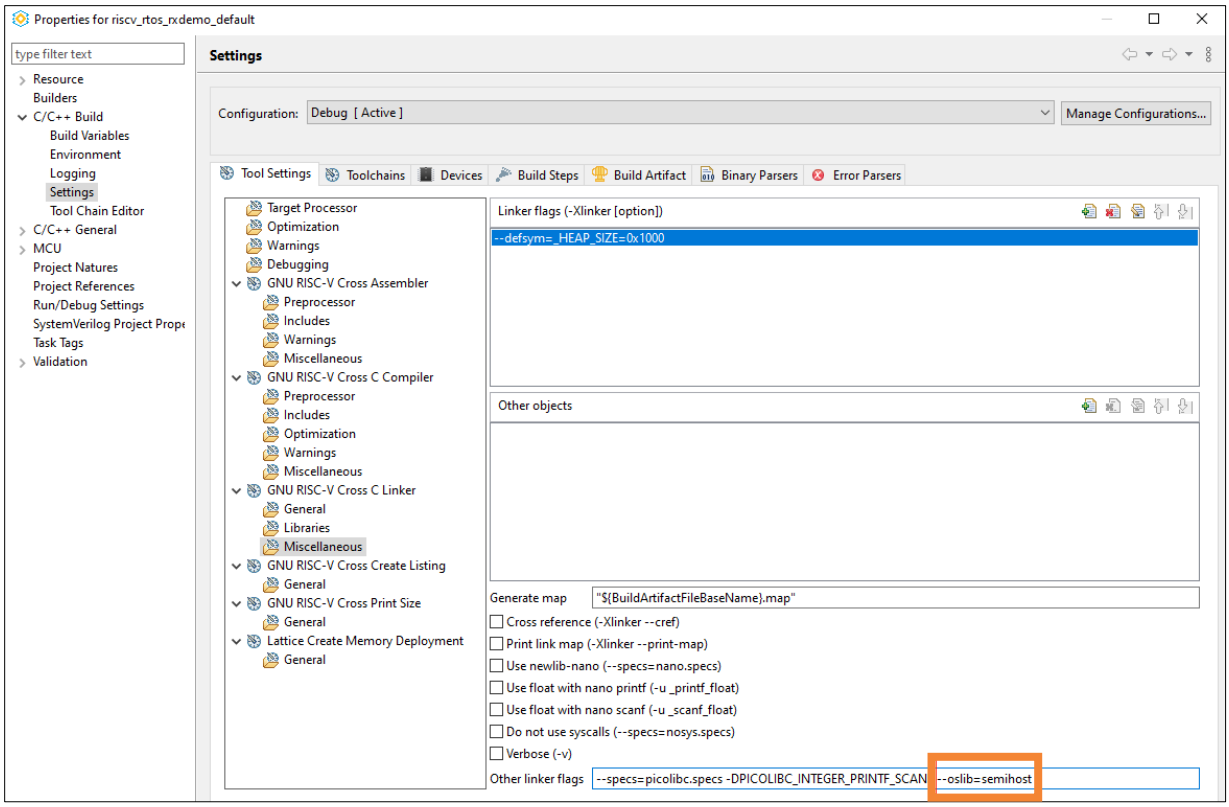


Figure 6.14. --oslib=semihost Linker Flag 1

7. Lattice Propel Tutorial – Timing Profiling

Timing profiling is an important aspect of software programming. Through profiling one can determine the parts in program code that are time consuming and need to be re-written. This helps make user program execution faster, which is always desired.

The timing profiling function is supported in Timing Profiling Project.

This tutorial uses a CertusPro-NX Evaluation Board with RISC-V RX SoC Project for demonstration.

Notes:

- Timing Profiling Project needs more system memory, the suggestion value is 0x20000.
- Semihosting is required for writing files back.

7.1. Prepare the Hardware and Programming the Target Device – Timing Profiling

This section introduces how to prepare the hardware and program the target device for the Timing Profiling project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

1. Create RISC-V RX SoC Project with CertusPro-NX Evaluation Board (Figure 5.1). Click **Finish**.
2. Lattice Propel Builder automatically opens. In Lattice Propel Builder, double-click tcm0_inst in the **Schematic** view and set a larger value for Address Depth both in Port S0 Settings (Figure 6.1) and Port S1 Settings (Figure 6.2). Click **Generate**.
3. Select **Project** > **Generate**.
4. Generate the programing file by running Lattice Radiant software in this SoC Project. See the SoC Project Design Flow section for detailed steps.
5. Program the programing file to the target device, CertusPro-NX Evaluation Board.

7.2. Creating Timing Profiling C Project

1. The Timing Profiling C project can be created through the **Load System and BSP** page (Figure 7.1).
2. In **System env** field, select the system environment file from the RX core SoC project just generated.
3. Select **Timing Profiling Project** and check the project name.
4. Click **Next**. Keep the default settings in the **Lattice Toolchain Setting** page.
5. Click **Finish**.

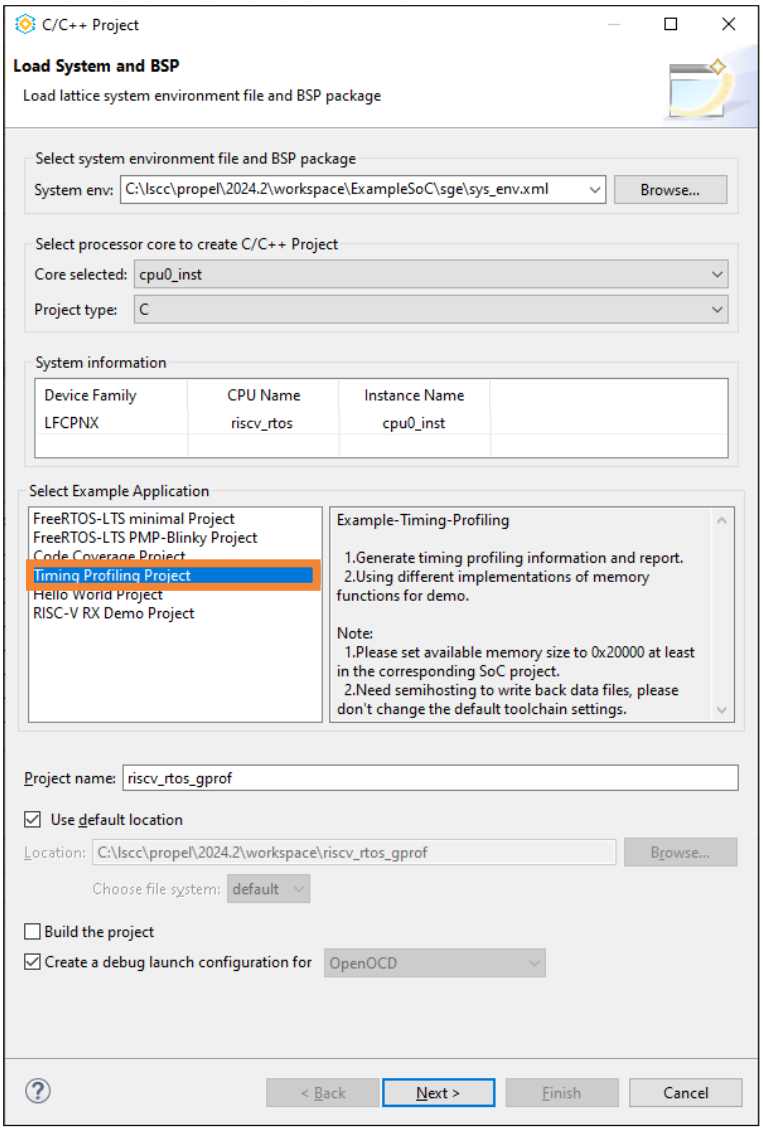


Figure 7.1. Load System and BSP Page 6

7.3. Compiling and Running Demo – Timing Profiling

7.3.1. Compiling C Project – Timing Profiling

1. In the **Project Explorer** view, select the C project, riscv_rtos_gprof.
2. Select **Project** > **Build Project**.

7.3.2. Running Demo – Timing Profiling

1. In the **Project Explorer** view, select the C project, riscv_rtos_gprof.
2. Select **Run** > **Debug Configurations…**.
3. Select riscv_rtos_gprof in **GDB OpenOCD Debugging** (Figure 7.2).
4. Click the **Debug** button.

   Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then starting the debugging session.

   Note: This demo uses the default debug configuration options.

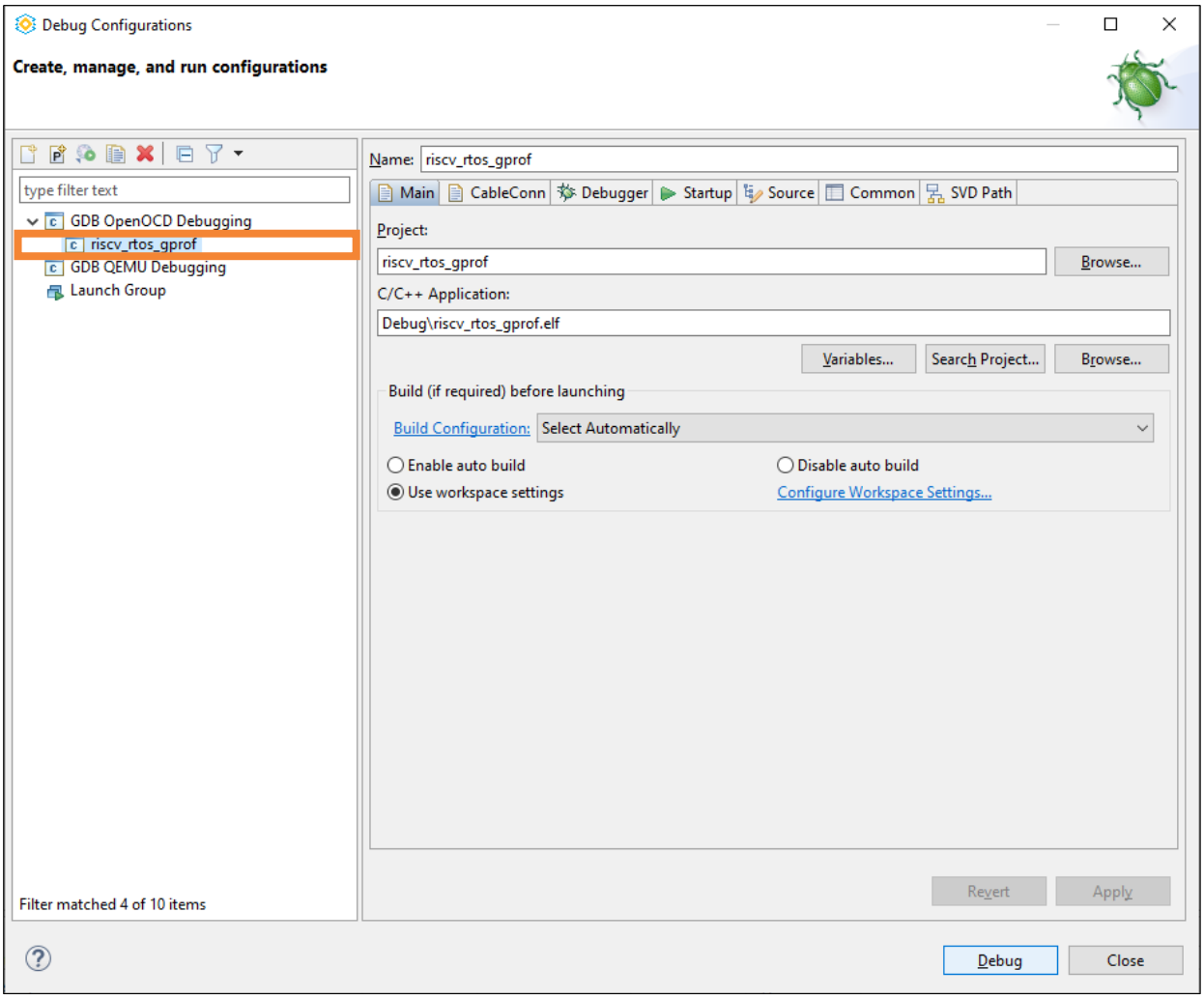


Figure 7.2. Debug Configurations Dialog 5

5. Select **Run** > **Resume**. Then the console output logs.

   Wait for a few minutes, until the console logs: *Test success! file "gmon.out" has been generated.*
6. Select **Run** > **Terminate** to stop running.

7.3.3. Display gprof Viewer

1. Find the gmon.out file in Project Explore of the riscv_rtos_gprof project. Double-click this file (Figure 7.3). Click **OK**.
2. Wait for a few minutes. The gprof Viewer opens (Figure 7.4).

   You can click the icons in the gprof Viewer to change the display style, such as **sort samples per function** (Figure 7.4).

   You can also click the title of every column to sort.

   For detailed usage of gprof Viewer, refer to Help - Eclipse Platform and type in GProf View in the Search box.

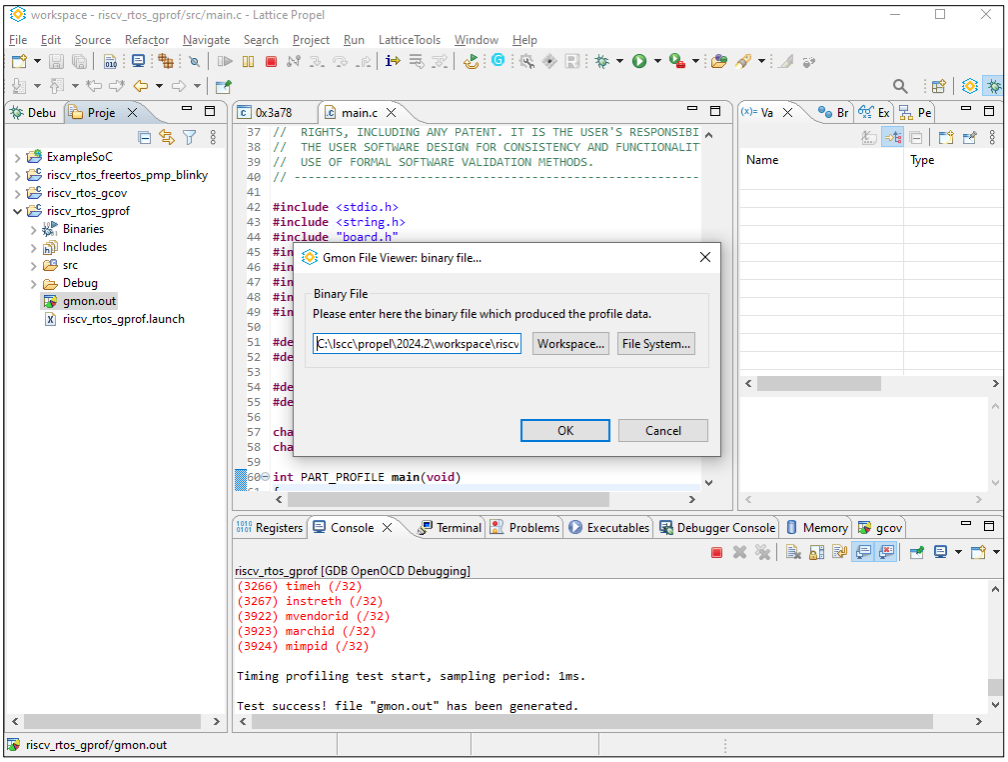


Figure 7.3. gmon File Viewer

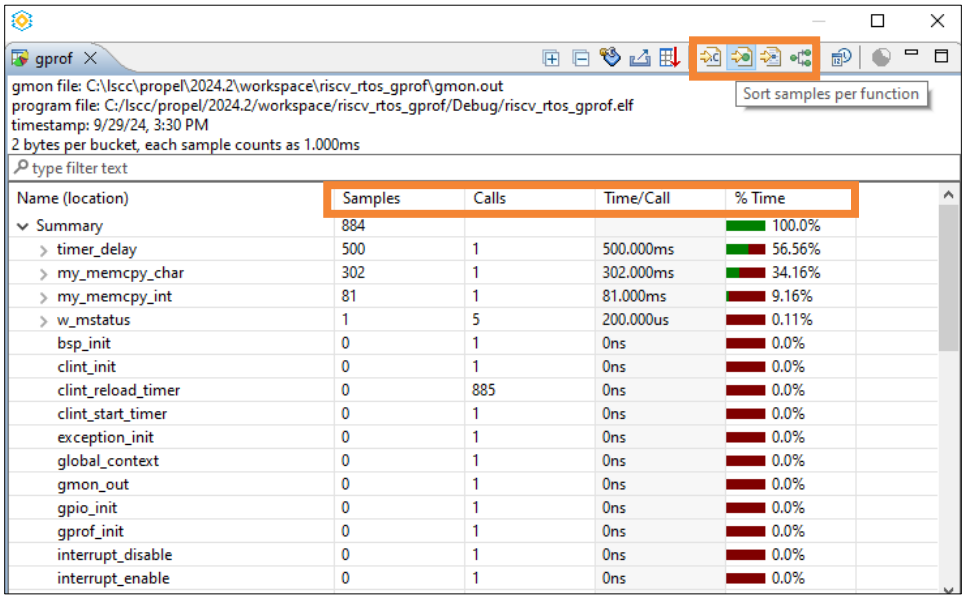


Figure 7.4. gprof Viewer 1

7.4. Enabling Timing Profiling for Existing C Project

1. In the **Project Explorer** view, select the existing C project.
2. Select **Project** > **Properties** > **C/C++ Build** > **Settings.**
3. Select **Debugging.** Enable the **Generate gprof information** checkbox (Figure 7.5).
4. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_GPROF (Figure 7.6).

5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgprof (Figure 7.7).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags in the label, Other linker flags. Make sure the linker flag, --oslib=semihost, is supported (Figure 7.8).
7. Click **Apply and Close**.
8. Open file **linker.ld** in the src folder. Set an enough value for _HEAP_SIZE (Figure 7.9).

   Note: _HEAP_SIZE is the size of heap memory, used for malloc. Timing profiling function requires additional heap memory. The size is approximately 125%/175% of .text size.
9. Copy gprof.c and gprof.h to src folder, the files can be found from demo project created in the Creating Timing Profiling C Project section.
10. Enable a hard timer. Add function `profile_handler()` into timer handle.
11. Add line `gprof_init();` to the line where profiling starts. Add line `gmon_out();` to the line where profiling stops.
12. Refer to the demo project and make code correct.
13. Rebuild this C project.

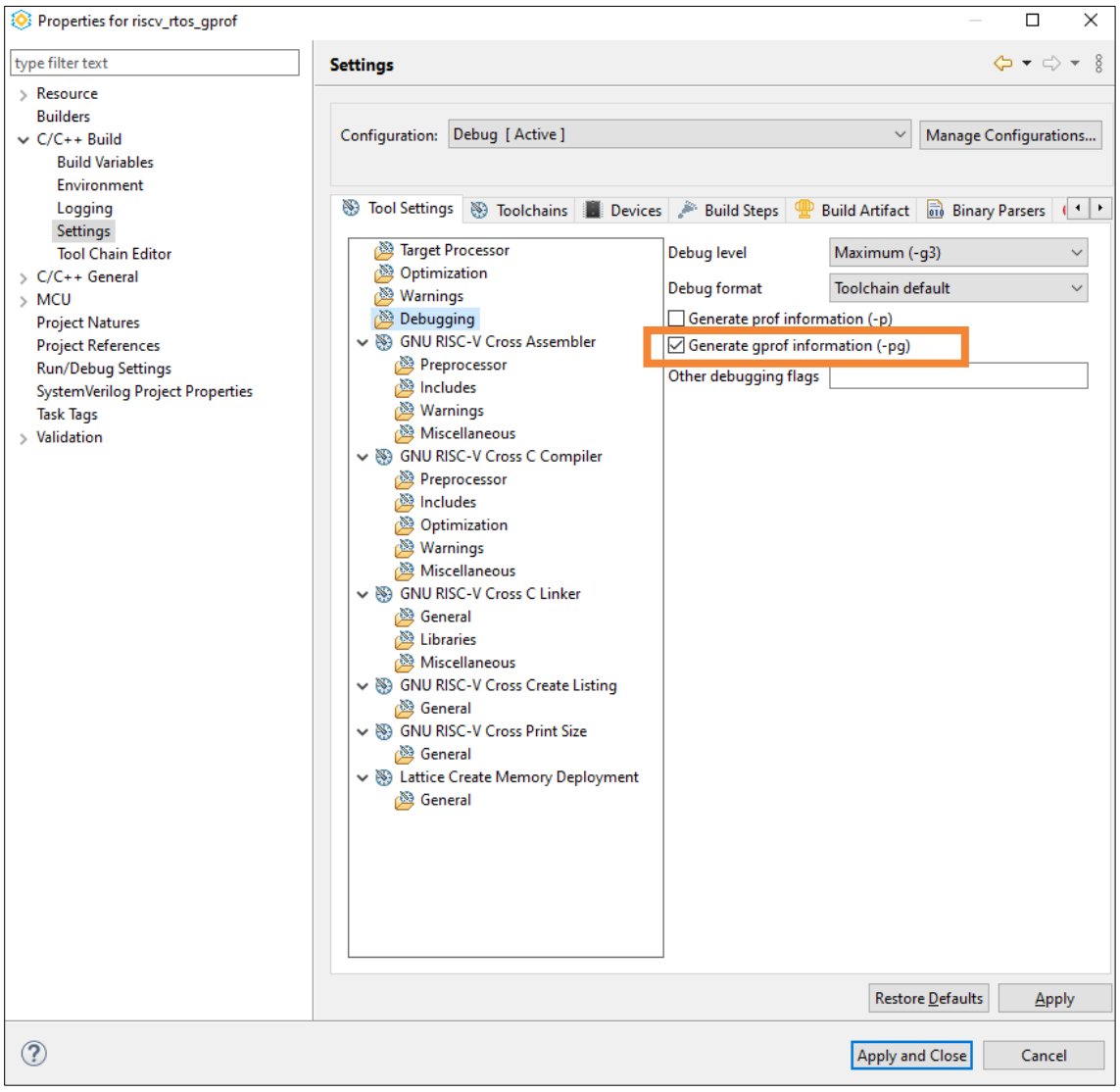


Figure 7.5. Generate gprof Information

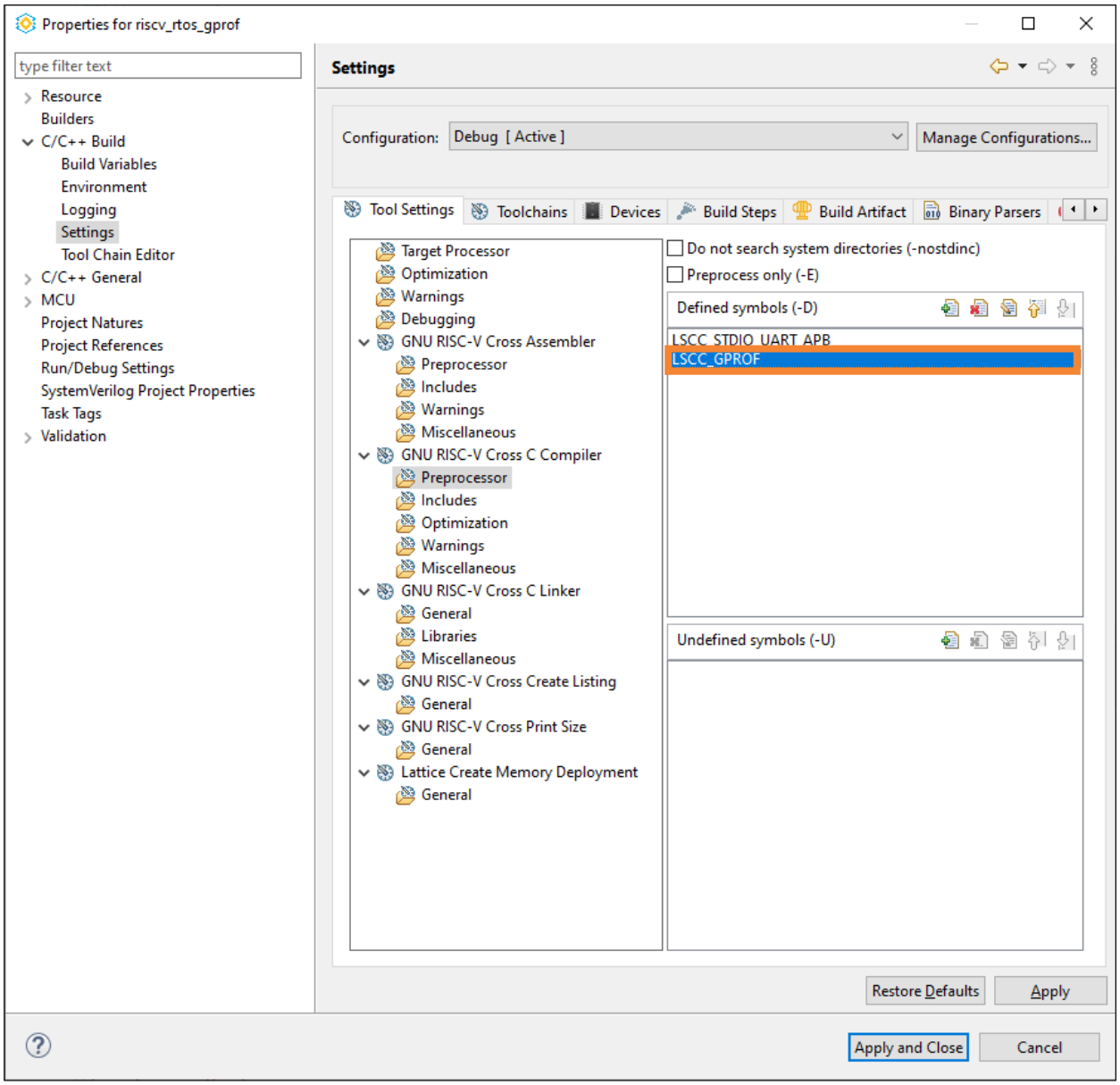


Figure 7.6. LSCC_GPROF Symbol

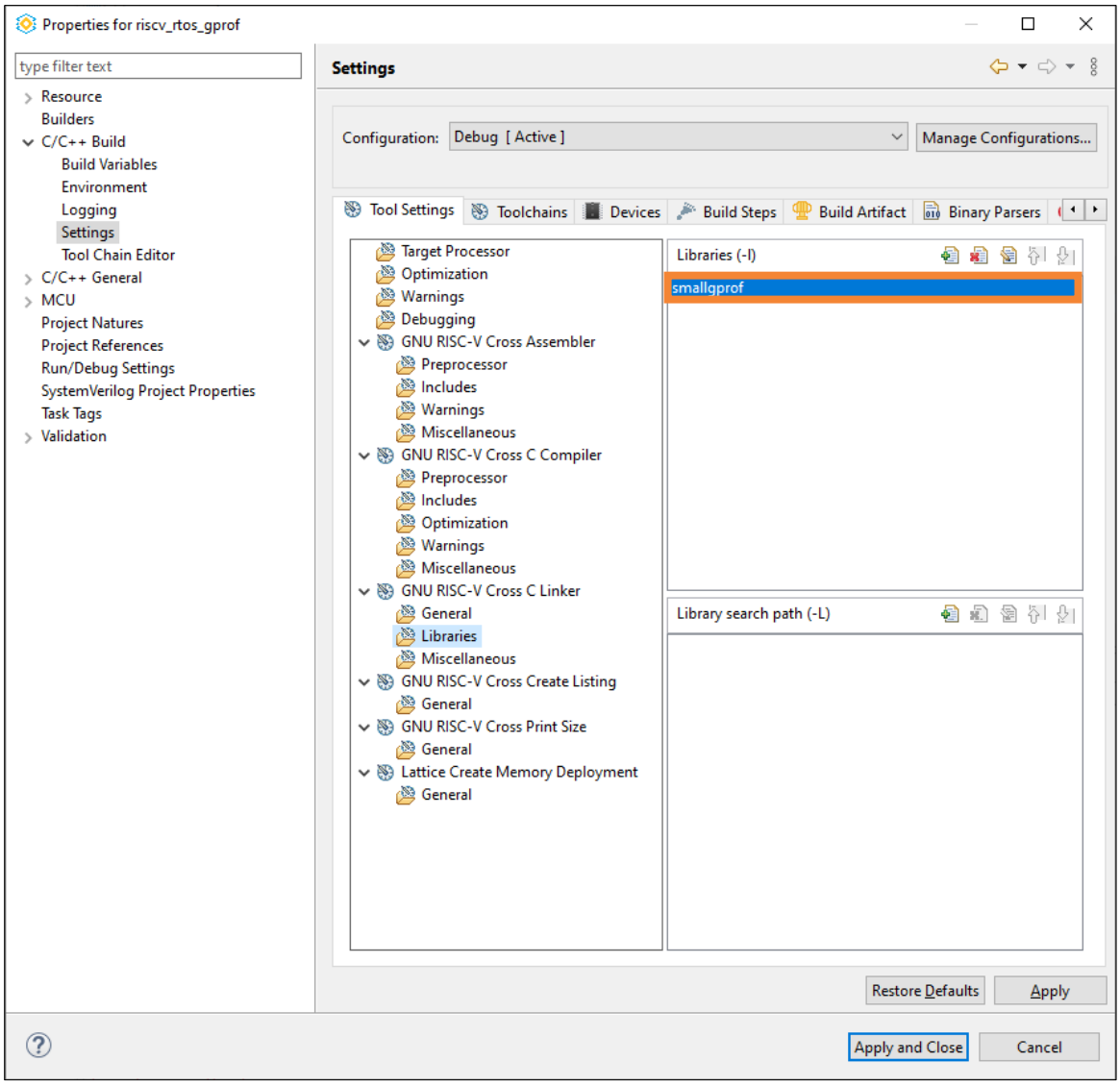


Figure 7.7. smallgprof Link Library

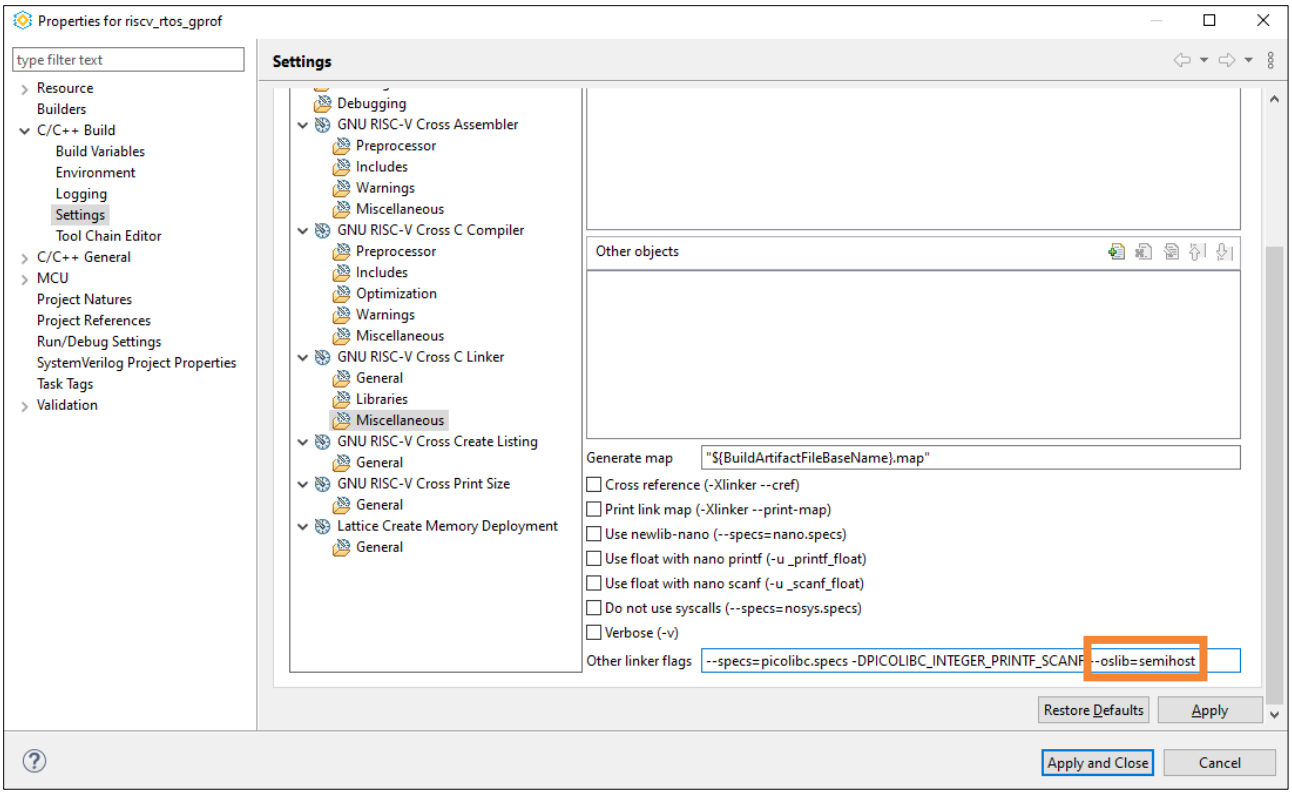


Figure 7.8. --oslib=semihost Linker Flag 2

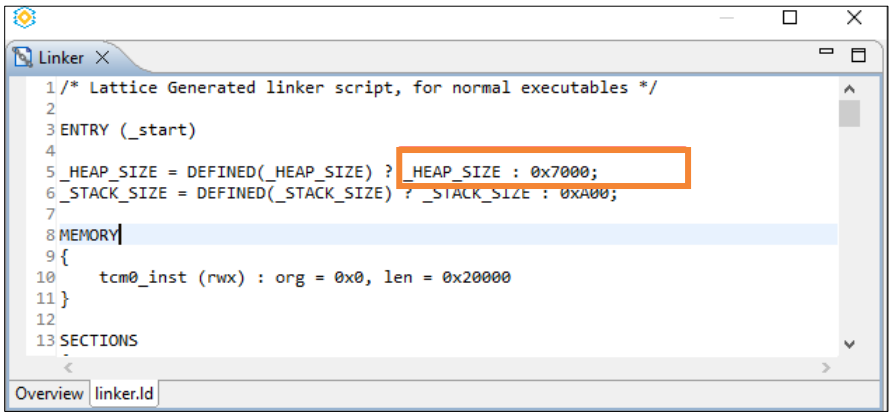


Figure 7.9. _HEAP_SIZE in Linker Script File

8. Lattice Propel Tutorial – CXU Demo

Composable Extension Unit (CXU) is supported in version 2.5.0 of the RX CPU IP. A CXU is a light-weight, customized arithmetic accelerator. With the support of the CXU hardware, you can add custom instructions to deploy CXU according to the actual software demand.

This tutorial uses a CertusPro-NX Evaluation Board with the Secure Hash Algorithm 3 (SHA-3) CXU SoC project for demonstration.

SHA-3 CXU SoC Project template includes a corresponding C project. This C project does software/CXU SHA3-256 operation and software/CXU Convolutional Neural Network (CNN) convolution operation. Then, it displays the performance results.

This C project also supports timing profiling and code coverage functions. You can enable these functions manually.

Note: It is recommended that these two functions should not be enabled at the same time.

8.1. Preparing the Hardware and Programming the Target Device – CXU Demo

This section introduces how to prepare the hardware and program the target device for the CXU demo project.

Note: The SoC project templates will be gradually migrated to the new scalable SoC project templates that are only available from Lattice Propel Builder. If the following flow for creating an SoC design project is unreachable, create it from Lattice Propel Builder. See Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219) for more details.

1. Create a SHA-3 CXU SoC project with the CertusPro-NX Evaluation Board (Figure 8.1). Click Finish.
2. Select **Project** > **Generate**.
3. Generate the programing file by running Lattice Radiant software in this SoC Project. Refer to the SoC Project Design Flow section for detailed steps.
4. Program the programing file to the target device, CertusPro-NX Evaluation Board.

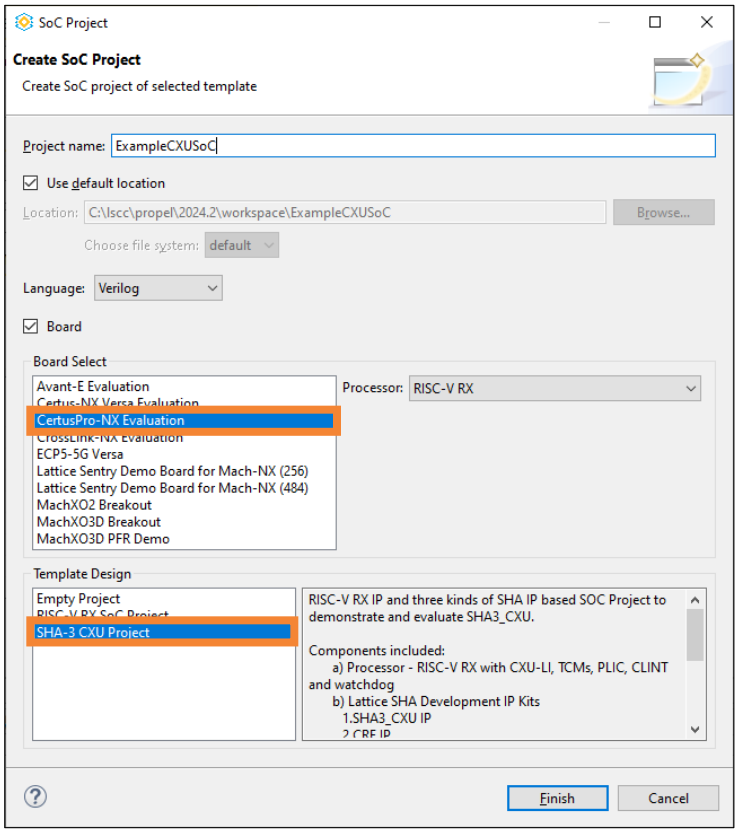


Figure 8.1. Create SoC Project Wizard 3

8.2. Creating CXU C Project

1. The Timing Profiling C project can be created through the **Load System and BSP** page (Figure 8.2).
2. In the **System env** field, select the system environment file from the CXU SoC project just generated.
3. Check the C project name.
4. Click **Next**. Click **Finish**.

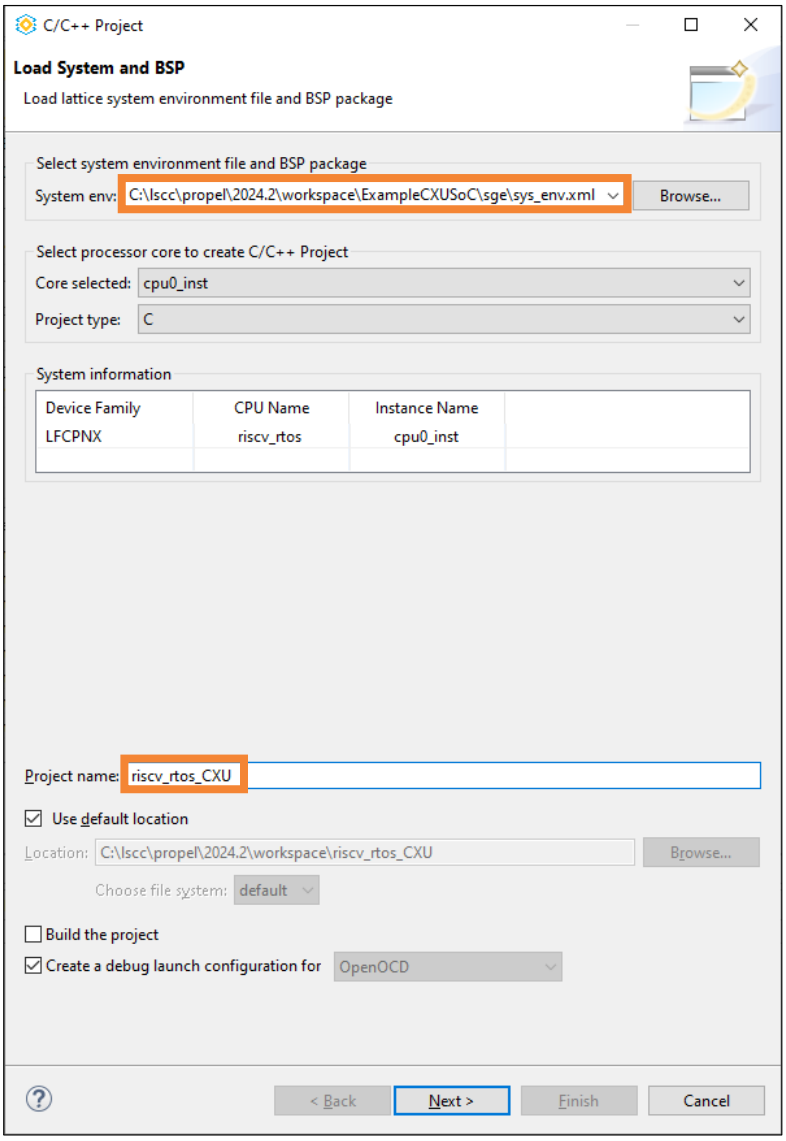


Figure 8.2. Load System and BSP Page 7

8.3. Compiling and Running Demo – CXU Demo

8.3.1. Compiling C Project – CXU Demo

1. In the **Project Explorer** view, select the C project, riscv_rtos_CXU.
2. Select **Project** > **Build Project**.

8.3.2. Running Demo – CXU Demo

1. In the **Project Explorer** view, select the C project, riscv_rtos_CXU.
2. Select **Run** > **Debug Configurations…**.
3. Select riscv_rtos_CXU in **GDB OpenOCD Debugging** (Figure 8.3).
4. Click the **Debug** button.

   Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then, starting the debugging session.

   Note: This demo uses the default debug configuration options.

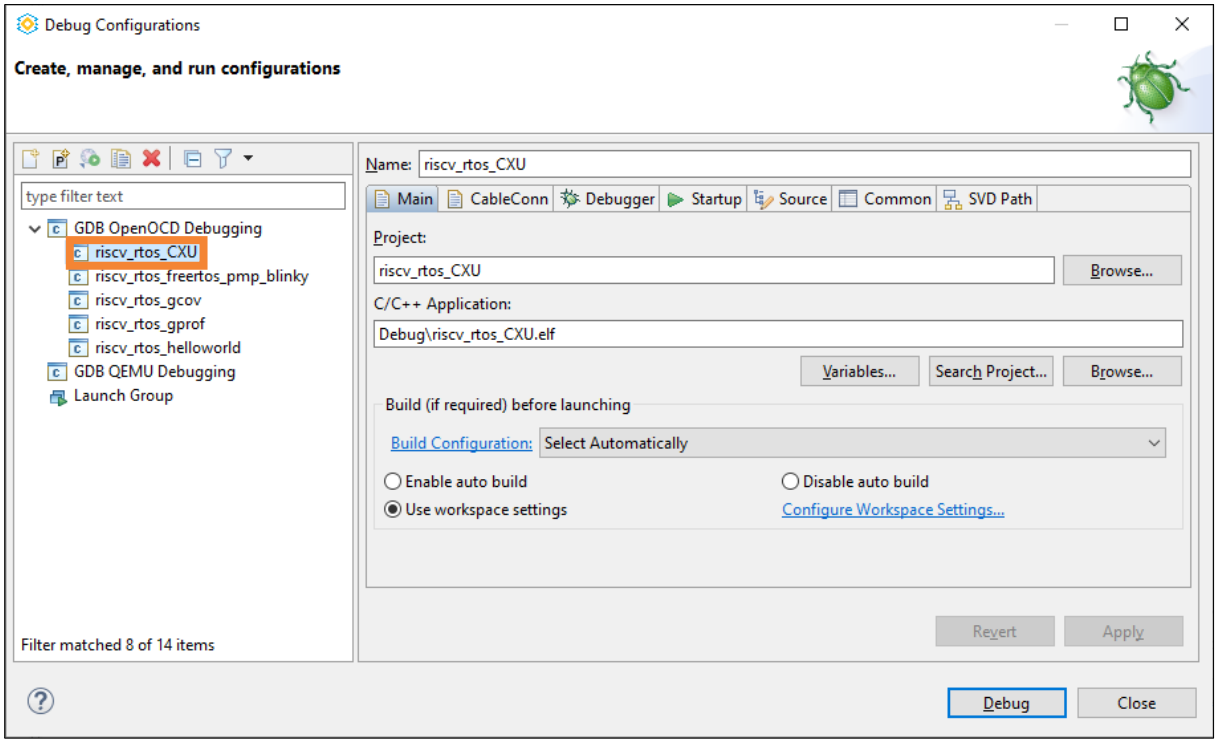


Figure 8.3. Debug Configurations Dialog 6

5. Find the **Terminal** view nested to the **Console** view. If this view is not found, re-open it from **Window** > **Show View** > **Terminal**.
6. In the **Terminal** view, click the **Open a Terminal** icon .
7. Choose the **Serial Terminal** and configure the **Serial port** with **Baud rate** 115200 (Figure 5.5).

   Note: The serial port number depends on the specific PC.
8. Click **OK**. A serial connection communicating with UART is ready.
9. Click the **Resume** icon on the toolbar. The serial terminal outputs running logs (Figure 8.4).

   Wait for a few minutes. The performance results logs are output (Figure 8.4).
10. Select **Run** > **Terminate** to stop running.

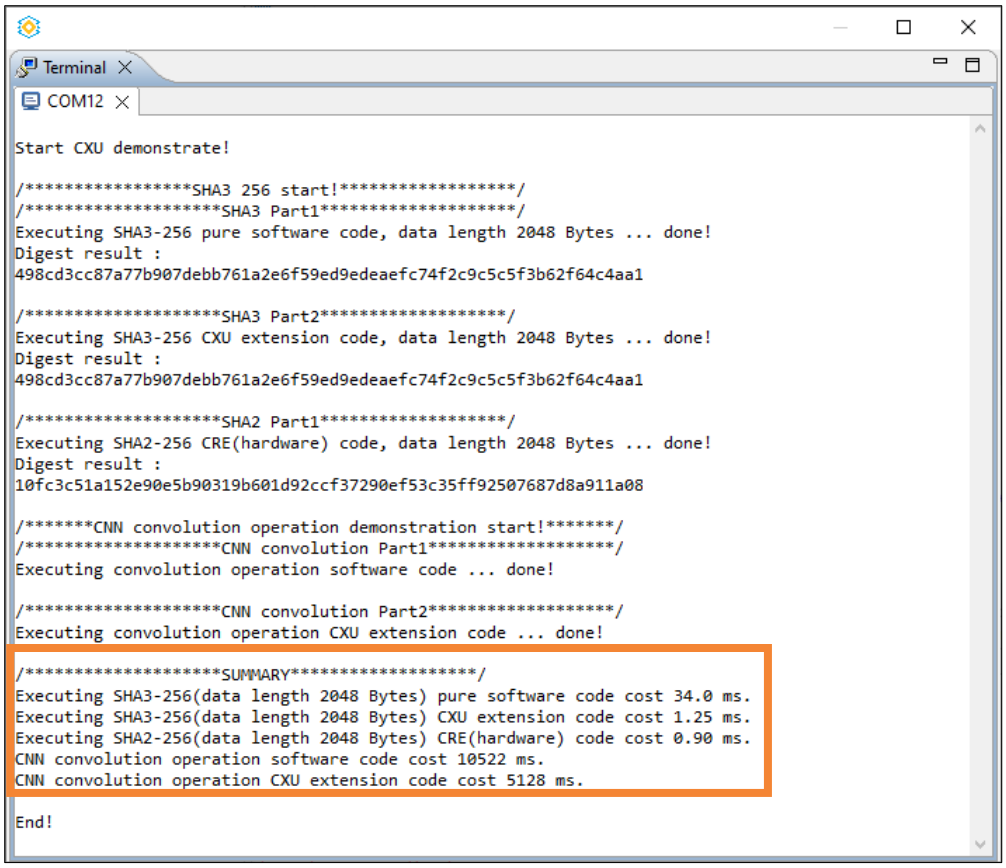


Figure 8.4. Terminal Logs

8.4. Using the Timing Profiling Function

For the details of timing profiling function, you can refer to the Lattice Propel Tutorial – Timing Profiling section.

1. In the **Project Explorer** view, select the existing C project, riscv_rtos_CXU.
2. Select **Project** > **Properties** > **C/C++ Build** > **Settings.**
3. Select **Debugging.** Check the **Generate gprof information** checkbox (Figure 7.5).
4. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_GPROF (Figure 7.6).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgprof (Figure 7.7).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags in the label, Other linker flags. Make sure the linker flag, --oslib=semihost, is supported (Figure 7.8).
7. Click **Apply and Close**.
8. Compile and run the demo, as shown in the Compiling and Running Demo – CXU Demo section. Then, the console output logs (Figure 8.5).
9. Double-click the file gmon.out to display the gprof Viewer. Refer to the Display gprof Viewer section.
10. Parse the gprof report (Figure 8.6).
11. Restore the settings in the above step 3 to step 6.

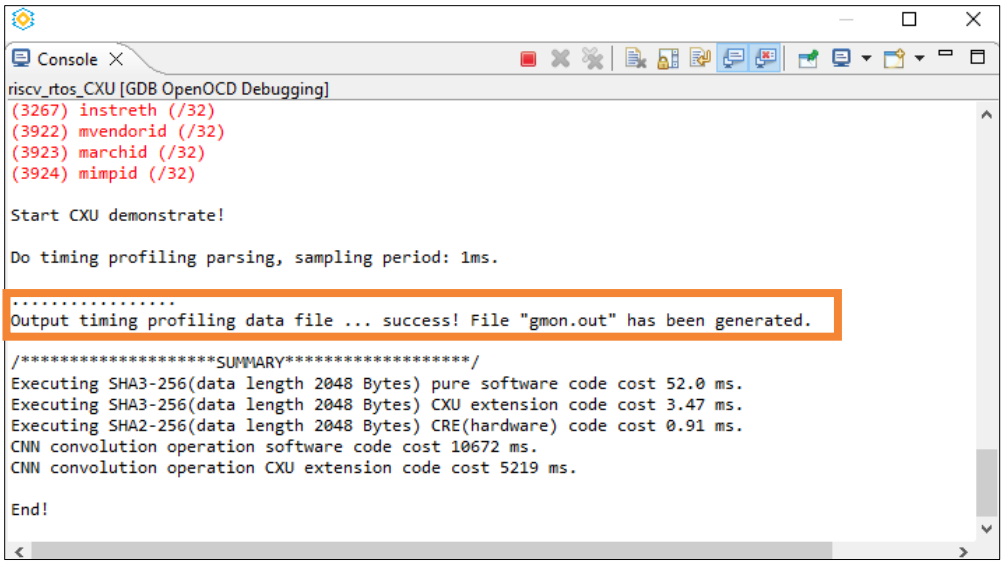
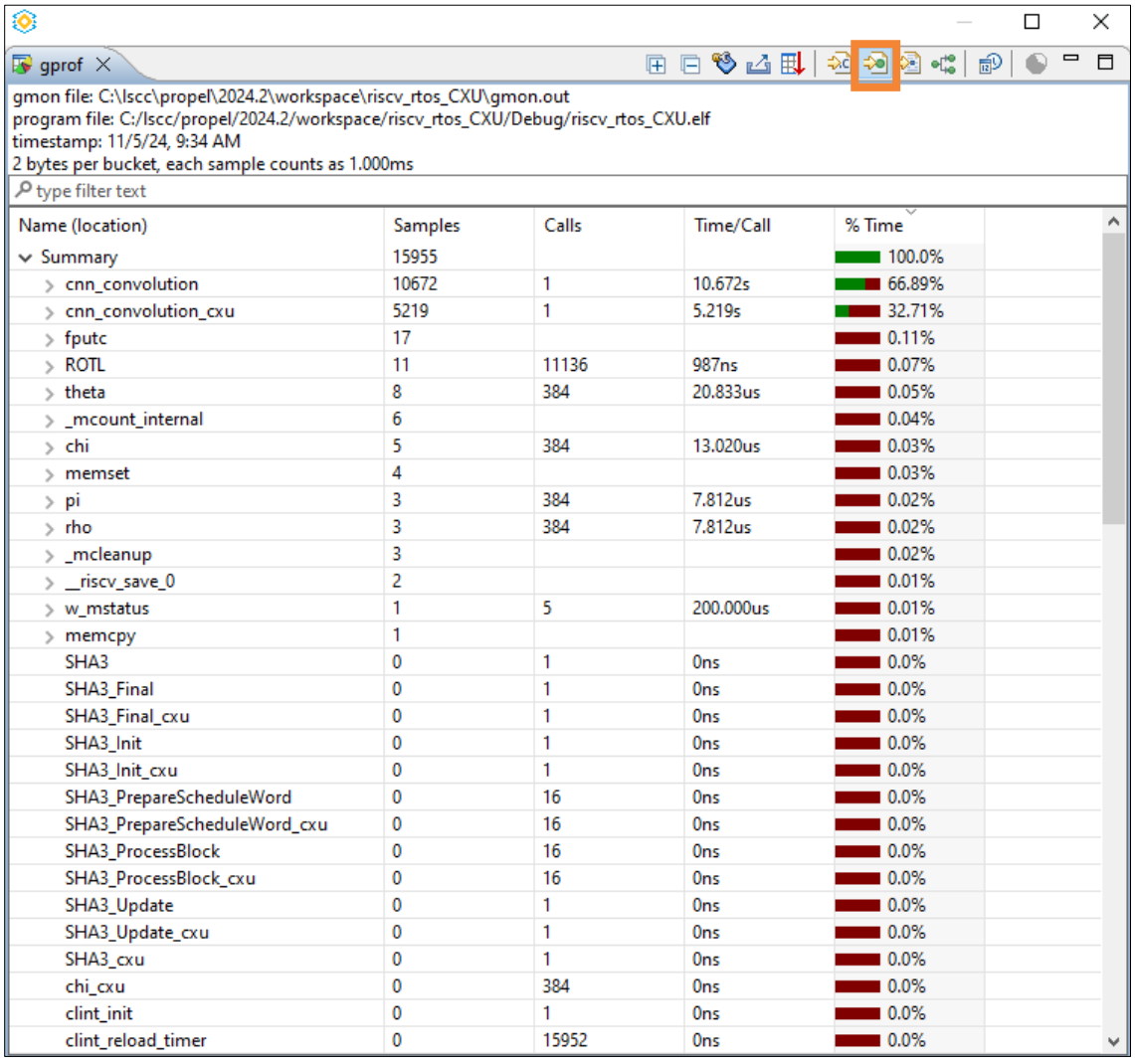


Figure 8.5. Console Logs 2



Name (location)	Samples	Calls	Time/Call	% Time
Summary	15955			100.0%
cnn_convolution	10672	1	10.672s	66.89%
cnn_convolution_cxu	5219	1	5.219s	32.71%
fputc	17			0.11%
ROTL	11	11136	987ns	0.07%
theta	8	384	20.833us	0.05%
_mcount_internal	6			0.04%
chi	5	384	13.020us	0.03%
memset	4			0.03%
pi	3	384	7.812us	0.02%
rho	3	384	7.812us	0.02%
_mcleanup	3			0.02%
__riscv_save_0	2			0.01%
w_mstatus	1	5	200.000us	0.01%
memcpy	1			0.01%
SHA3	0	1	0ns	0.0%
SHA3_Final	0	1	0ns	0.0%
SHA3_Final_cxu	0	1	0ns	0.0%
SHA3_Init	0	1	0ns	0.0%
SHA3_Init_cxu	0	1	0ns	0.0%
SHA3_PrepareScheduleWord	0	16	0ns	0.0%
SHA3_PrepareScheduleWord_cxu	0	16	0ns	0.0%
SHA3_ProcessBlock	0	16	0ns	0.0%
SHA3_ProcessBlock_cxu	0	16	0ns	0.0%
SHA3_Update	0	1	0ns	0.0%
SHA3_Update_cxu	0	1	0ns	0.0%
SHA3_cxu	0	1	0ns	0.0%
chi_cxu	0	384	0ns	0.0%
clint_init	0	1	0ns	0.0%
clint_reload_timer	0	15952	0ns	0.0%

Figure 8.6. gprof Viewer 2

8.5. Using the Code Coverage Function

For the details of the timing profiling function, refer to the Lattice Propel Tutorial – Timing Profiling section.

1. In the **Project Explorer** view, select the existing C project, riscv_rtos_CXU.
2. Choose **Project** > **Properties** > **C/C++ Build** > **Settings.**
3. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_COVERAGE (Figure 6.10).
4. Select **GNU RISC-V Cross C Compiler** > **Miscellaneous**. Add the compiler flag, -fprofile-arcs -ftest-coverage (Figure 6.11).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgcov (Figure 6.12).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags in the label, Other linker flags. Make sure the linker flag, --oslib=semihost, is supported (Figure 6.14).
7. Click **Apply and Close**.
8. Compile and run the demo, as shown in the Compiling and Running Demo – CXU Demo section. Then, the console output logs (Figure 8.7).
9. Double-click one of the coverage files to display the gcov Viewer, see the Display gprof Viewer section.
10. Parse the gcov report (Figure 8.8).
11. Restore settings in the above step 3 to step 6.

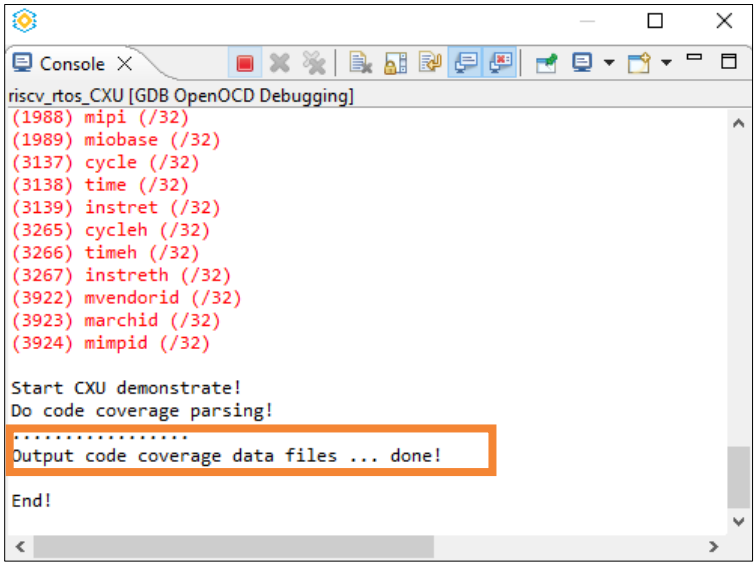


Figure 8.7. Console Logs 3

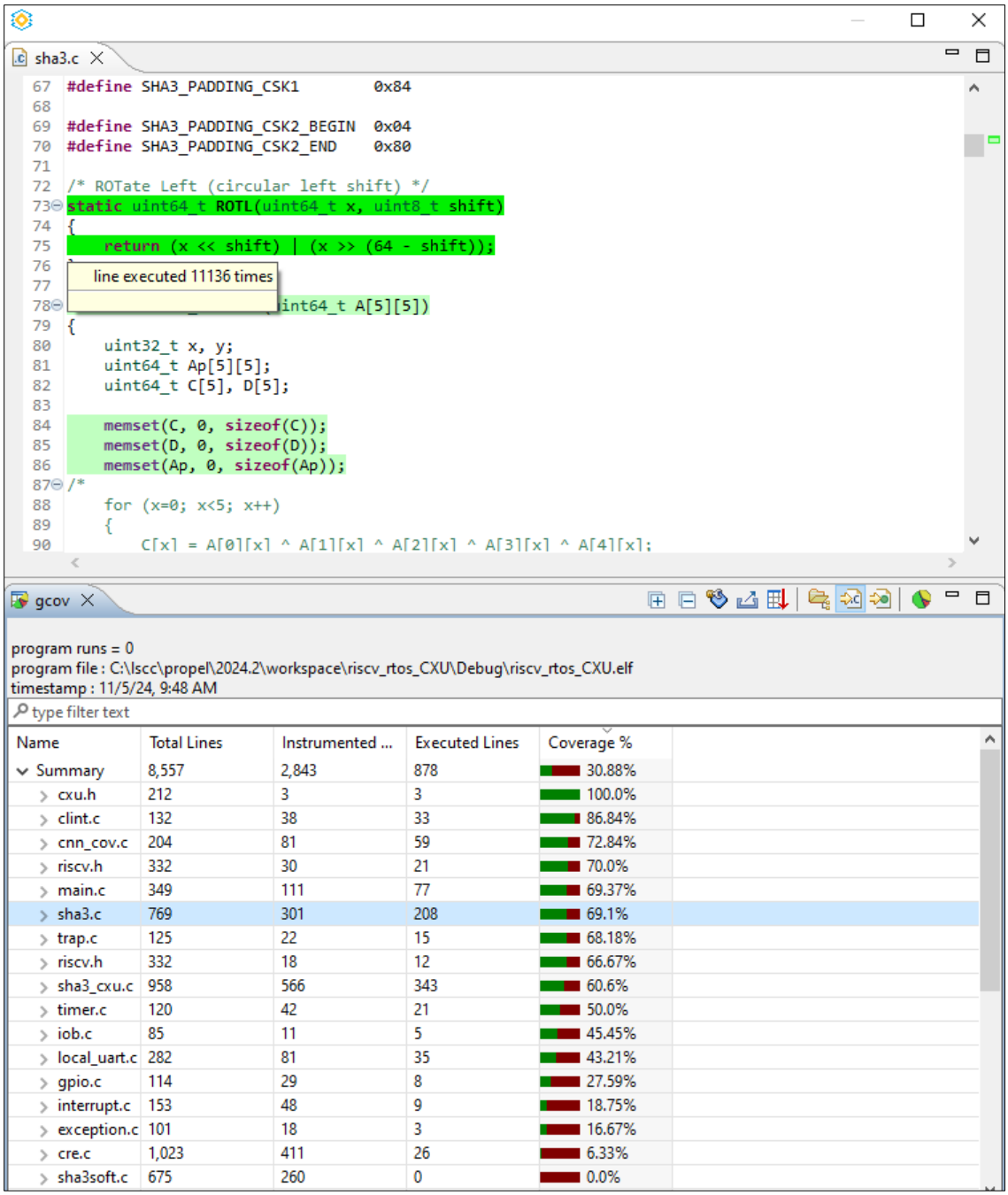


Figure 8.8. gcov Viewer

9. Lattice Propel Tutorial – QEMU

QEMU is a generic and open-source machine emulator and virtualizer.

For the detailed manual of QEMU, refer to the System Emulation — QEMU documentation.

Lattice Propel SDK includes a RISC-V QEMU simulator with no hardware required and a QEMU_helloworld template for demonstration.

9.1. Creating QEMU Hello World C Project

1. Select **File** > **New** > **Lattice C/C++ Project**.

   The C/C++ Project wizard opens with the **Load System and BSP** page (Figure 9.1).

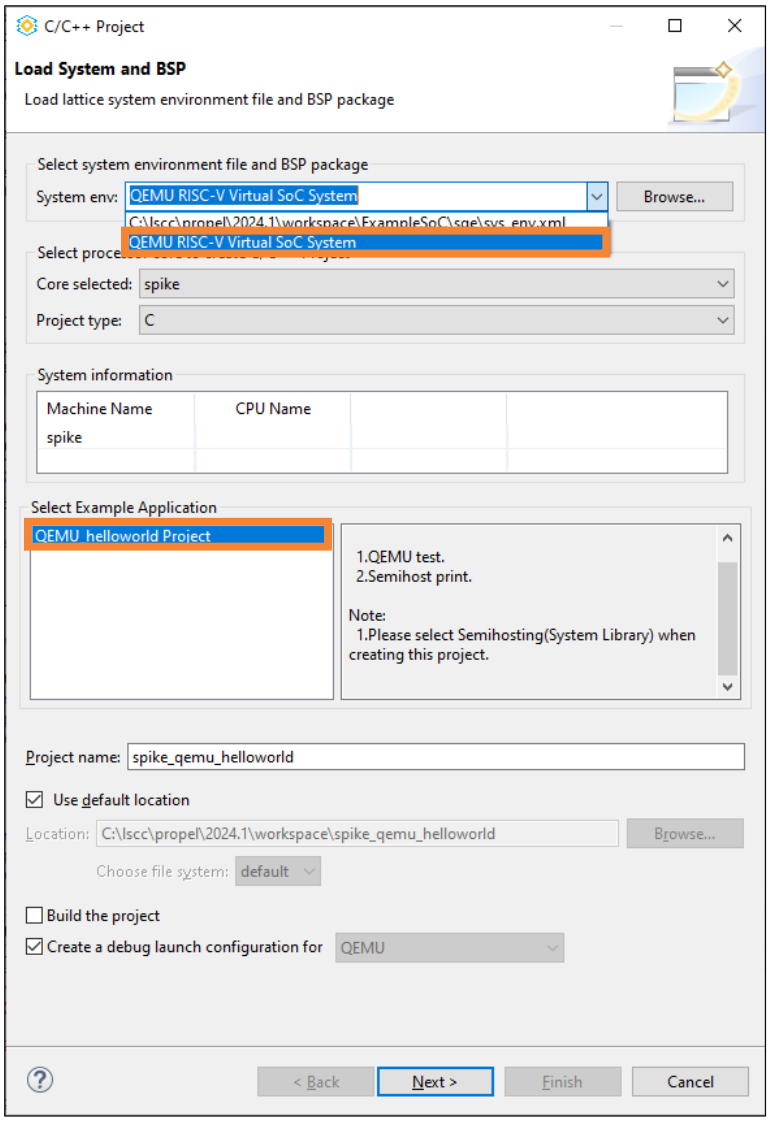


Figure 9.1. Load System and BSP Page 8

2. Select **QEMU RISC-V Virtual SoC System** using the drop-down menu of the **System env** field (Figure 9.1).
3. Select QEMU_helloworld Project and check the project name.
4. Click **Next**. Keep the default settings in the **Lattice Toolchain Setting page**, as **Semihosting** is required for the printf output.

5. The C project is created and displayed in the workbench.
6. In the **Project Explorer** view, select the C project just created, spike_qemu_helloworld.
7. Choose **Project** > **Build Project**.
8. Check the build result from the **Console** view (Figure 9.2).

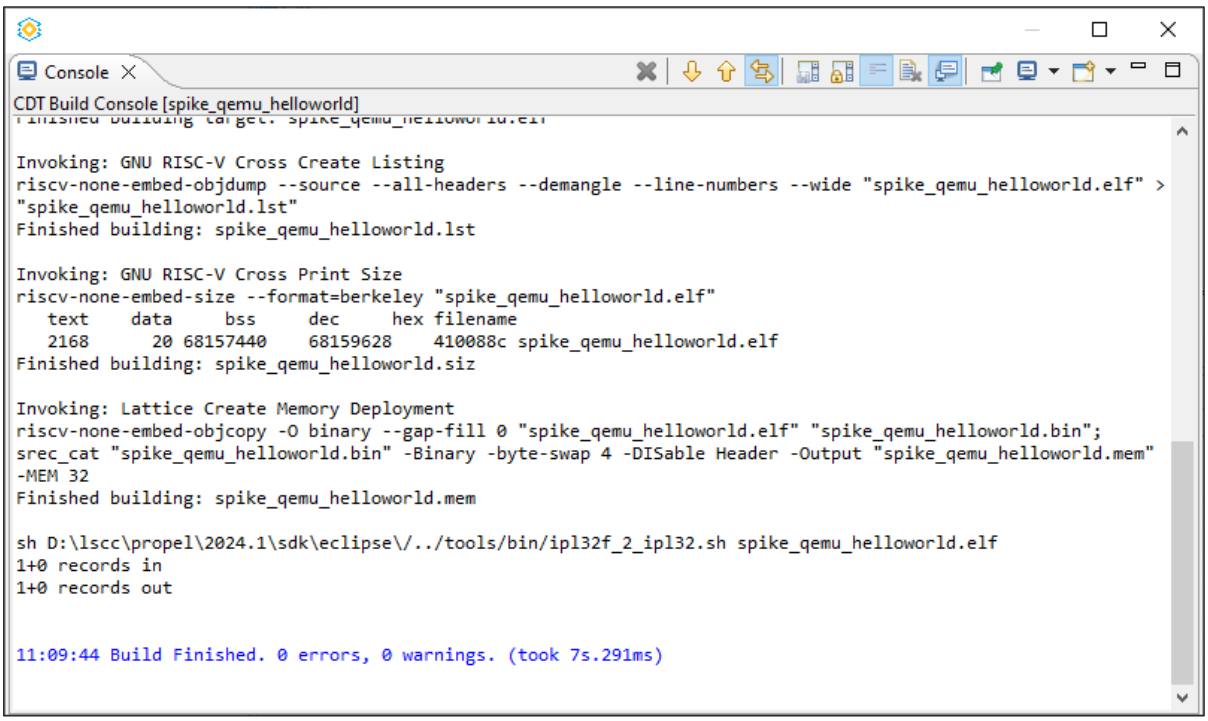


Figure 9.2. Build Console 2

9.2. Running QEMU C Project

1. In the **Project Explorer** view, select the C project just created, spike_qemu_helloworld.
2. Choose **Run** > **Debug Configurations…**.
3. Choose spike_qemu_helloworld in **GDB QEMU Debugging** (Figure 9.3).
4. Click the **Debug** button.

   Wait for a few seconds for switching to the debug perspective, starting the server, allowing it to connect to the target device, starting the gdb client, downloading the application, and then starting the debugging session.

   Note: This demo uses the default debug configuration options.
5. Click the **Resume** icon on the toolbar. The Console outputs running logs (Figure 9.4).

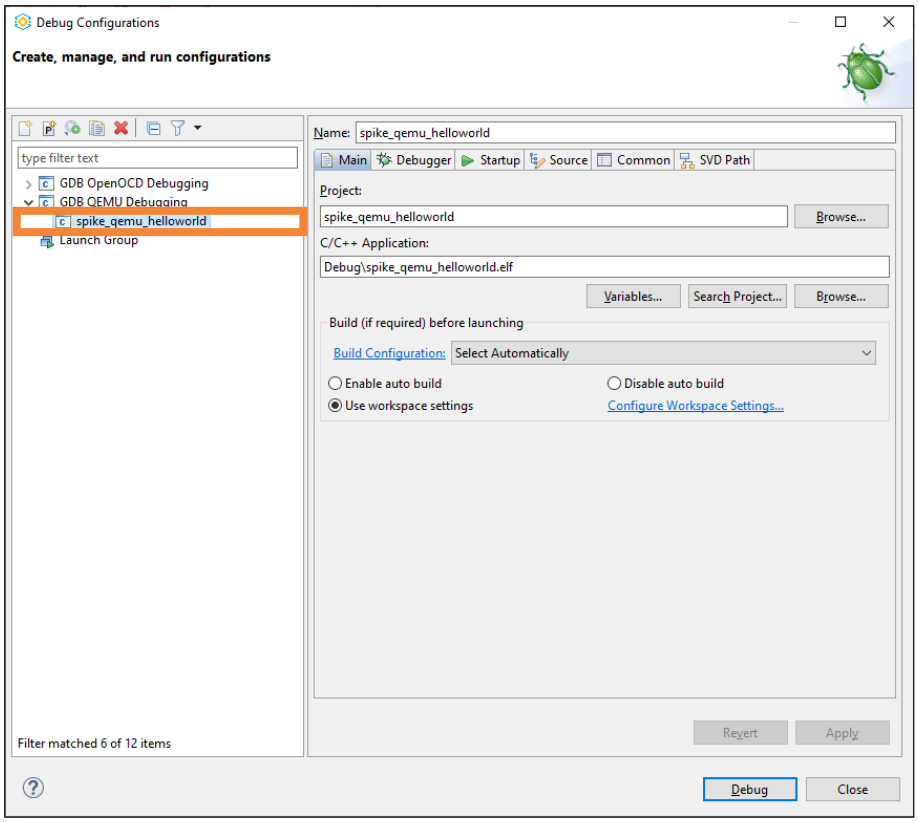


Figure 9.3. Debug Configurations Dialog 7

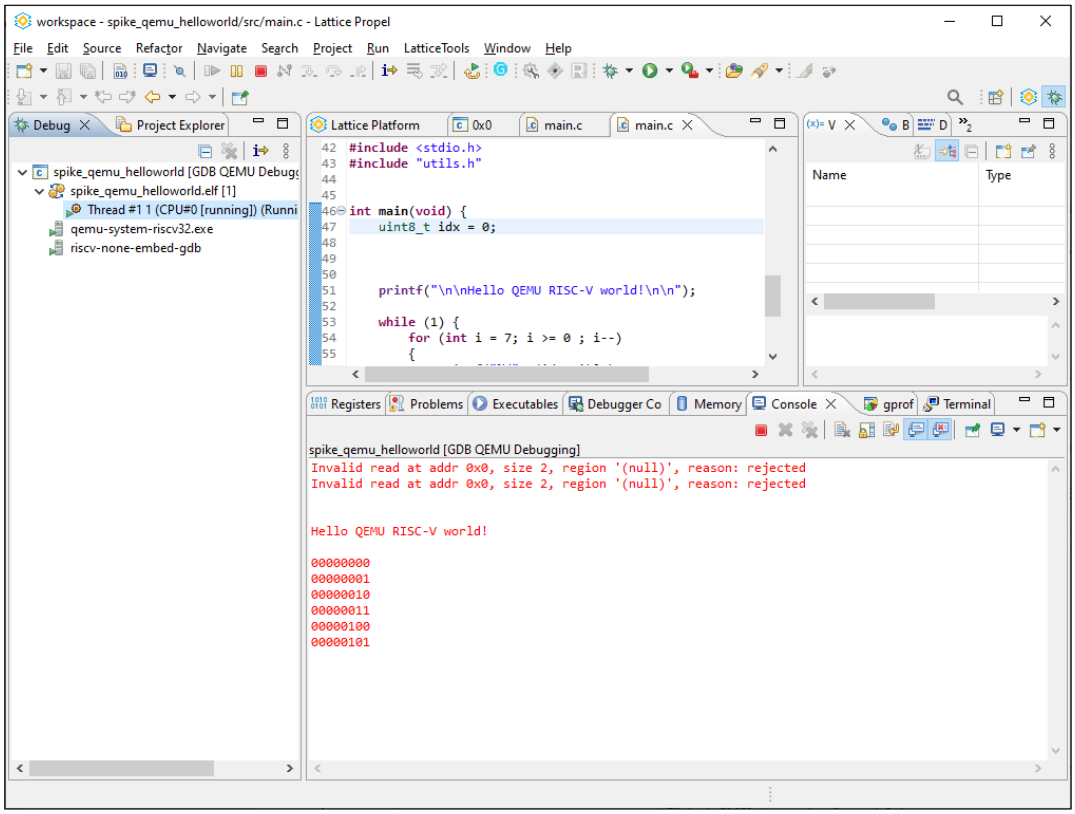


Figure 9.4. Running/Debugging Windows 2

Appendix A. Linker Script and System Memory Deployment

Introduction

During the Lattice C/C++ Project creation, Lattice Propel SDK generates a linker script file, linker.ld, within the project. This linker script file contains a memory region list parsing from the corresponding SoC design.

Note: Illegal memory regions are not imported to linker script. A memory region is considered illegal if it has any of the following conditions:

- No connection to CPU
- Address space conflict

Each memory region has a list of attributes to specify whether or not using a particular memory region for an input section (Figure A.1).

- r: Read-only section.
- w: Read/Write section indicating the memory region is connected to the data port of CPU.
- x: Executable section indicating the memory region is connected to the instruction port of CPU.

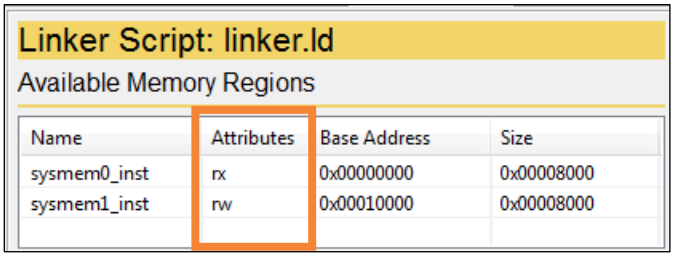


Figure A.1. Memory Regions in Linker Script

The generated linker script contains a mapping table of section pointing to the memory region (Figure A.2). Depending on the attributes of each memory region, code section and data section can point to the same or different memory regions.

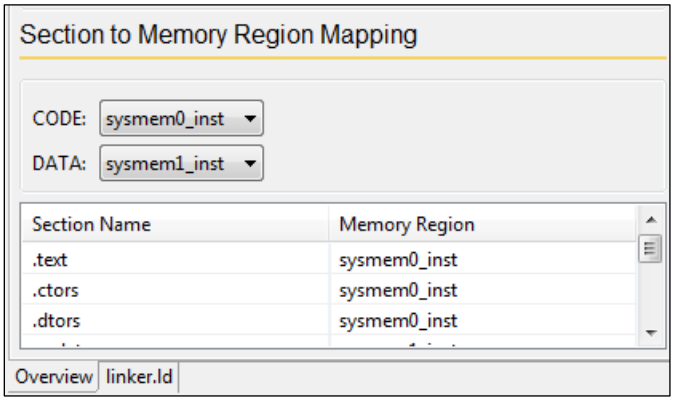


Figure A.2. Section to Memory Region Mapping

During the Lattice C/C++ Project building, Lattice Propel SDK generates Lattice system memory initialization files. Depending on the number of the memory regions used, it generates single memory initialization file or multiple memory initialization files. The following picture (Figure A.3) shows an example of multiple memory files being generated, separated for code and data segments.

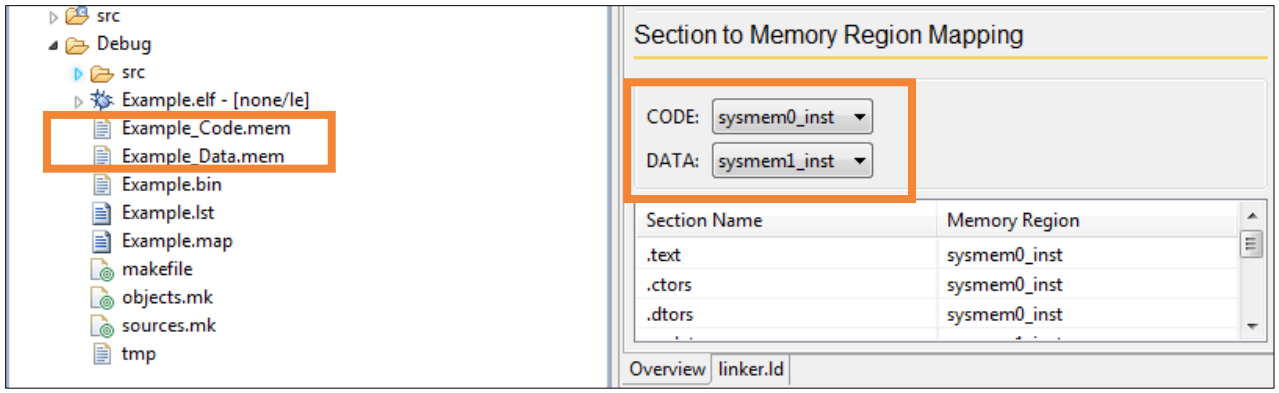


Figure A.3. Linker Script and Generated Memory Files

If you modify the linker script manually after the Lattice C/C++ project creation, especially changing the number of the used memory regions, the number of memory files generated during project building cannot be changed automatically. You can re-configure the generation of memory files in Lattice Propel SDK through the following ways:

1. In the **Project Explorer** view of Lattice Propel SDK, select a C/C++ project.
2. Choose **Project** > **Properties.** The **Properties** dialog opens showing the properties of the current project.
3. Select **Settings** of **C/C++ Build** category from the left pane. Select the **Toolchains** tab (Figure A.4).

   Check **Create memory file**, if you point the code and data segments to the same memory region. Or, check **Create multiple memory files**, if you point the code and data segments to separate memory regions.
4. Click **Apply**.

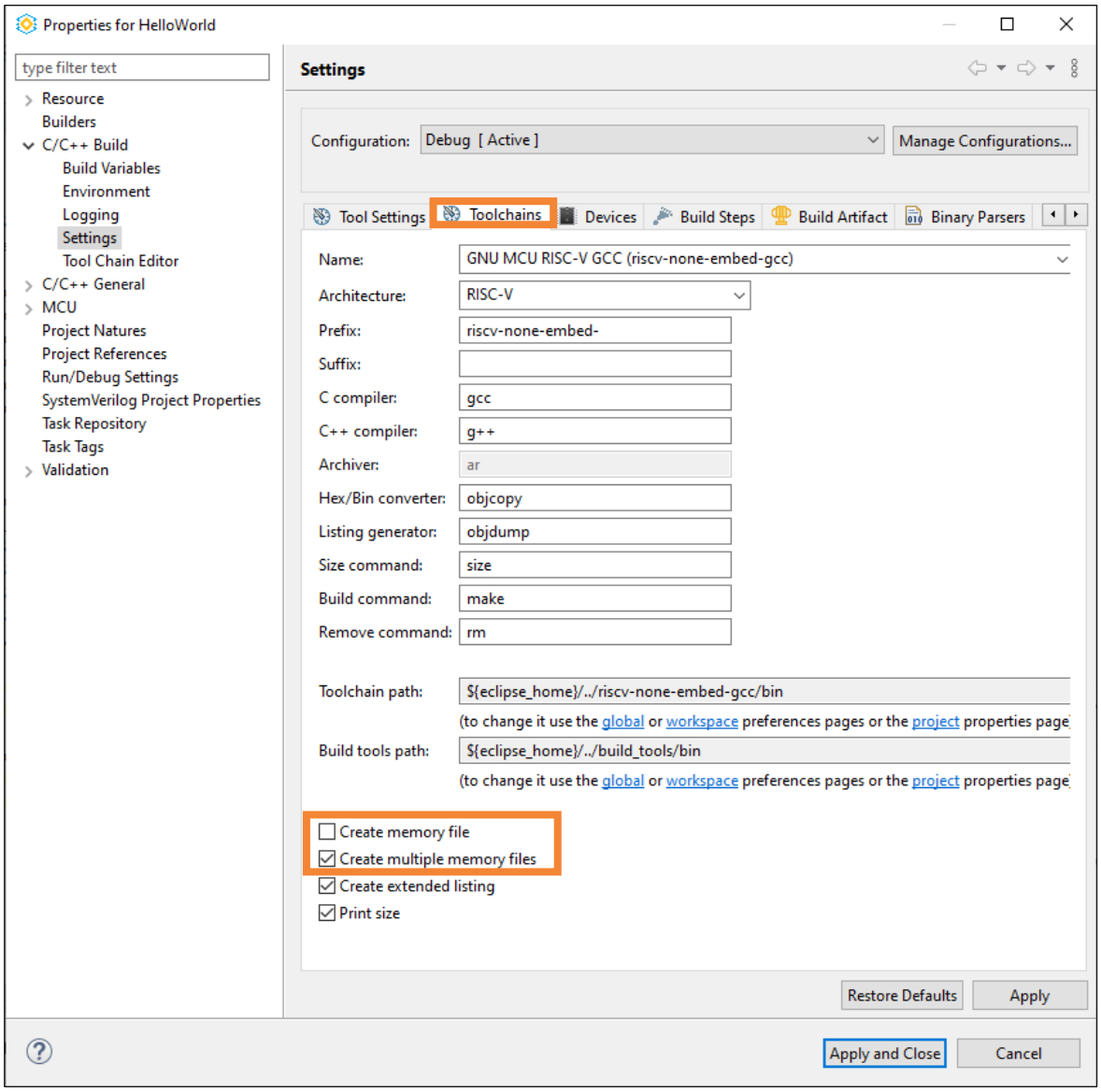


Figure A.4. Toolchains Tab of C/C++ Build Settings

5. Go back to the **Tool Settings** tab. The relevant Lattice memory deployment tools can be found (Figure A.5). Customize the tool options as needed.
6. Click **Apply and Close** to save the change.

   Note: The setting for each configuration, **Debug** or **Release**, is independent.

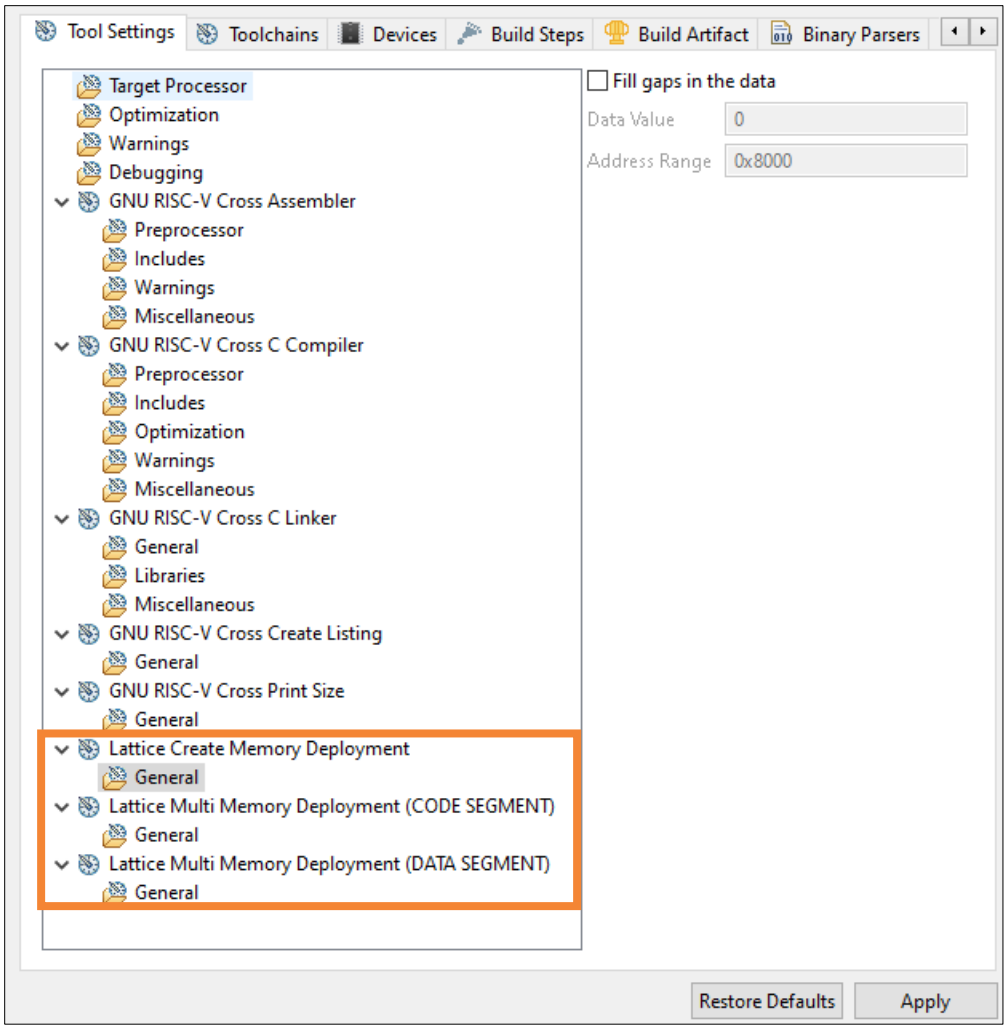


Figure A.5. Tool Settings Tab of C/C++ Build Settings

How to Fix the Region Overflowed Error

If you add too many code to your C project, it might cause the build error shown in the Figure A.6 error log.

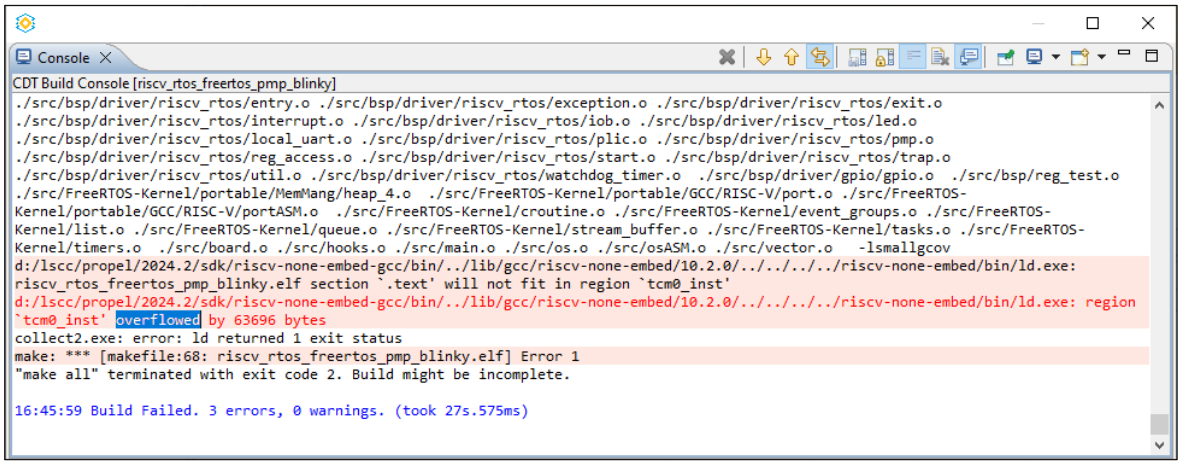


Figure A.6. Build Project Console 1

You can find the max size you can use from the linker script file linker.ld, as shown in Figure A.7.

In this example project, the max size is 0x10000, which is your C project target file's max size.

The error shown in Figure A.6 means the additional space size needed for this project is 63696 bytes, which is 0xF8D0 in hexadecimal. Therefore, the total space size needed is 0x1F8D0 bytes.

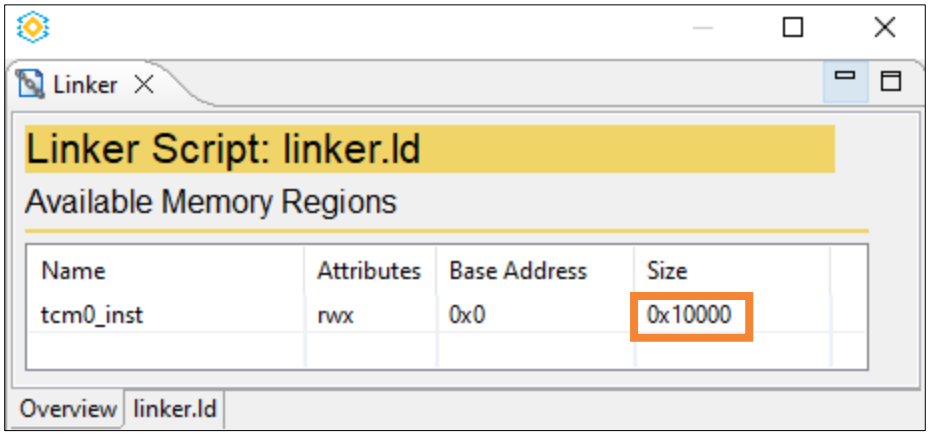


Figure A.7. Linker Script

You cannot modify the linker script file directly. This kind of modification can fix the C project build error. However, the corresponding SoC does not have more code space. The program cannot run on your board.

We can fix this error correctly by the following steps:

Note: This tutorial just uses single memory IP. If the project has more than one memory IP, you should check every section's size in corresponding storage spaces.

1. Switch to the corresponding SoC project, as shown in Figure A.8.
2. Double-click the tcm0_inst IP. The Port S0 address depth settings and the Port S1 address depth settings are shown in the Module/Block IP Wizard GUI (Figure A.9 and Figure A.10).
3. Set the Port S0 and Port S1 address depth to an adequate value. In the error shown in Figure A.6, we need 0x1F8D0 bytes, which needs to be 1k aligned. Consequently, the minimum size needed is 0x1FC00 bytes, equivalent to 0x7F00 DWORDs or 32512 in decimal. Refer to Figure A.11 and Figure A.12 for the settings described in this step. click **Generate**.
4. Re-generate this SoC project.
5. Run the Lattice Radiant software or Lattice Diamond software to generate the bit file, program the bit file to the device board. This process is important.
6. Switch back to the C project by selecting **Project** > **Update Lattice C/C++ Project…**. Check the checkbox for **Re-generate toolchain parameters and linker script**, click **Update** (Figure A.13). Choose **Yes** in the Confirm box.
7. Check the linker script file linker.ld (Figure A.14). The region size is changed to 0x1FC00 automatically.
8. Re-build the C project. You can see the correct build log (Figure A.15).

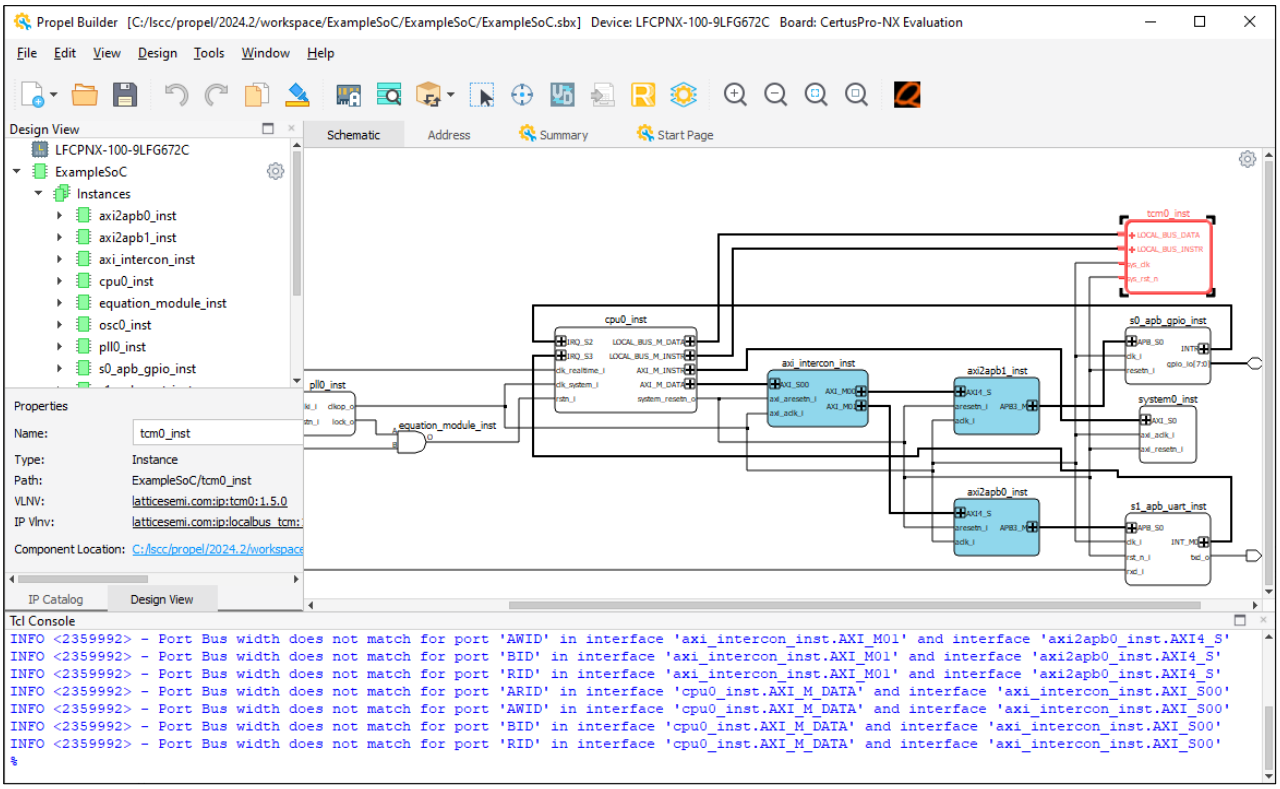


Figure A.8. Corresponding SoC Project

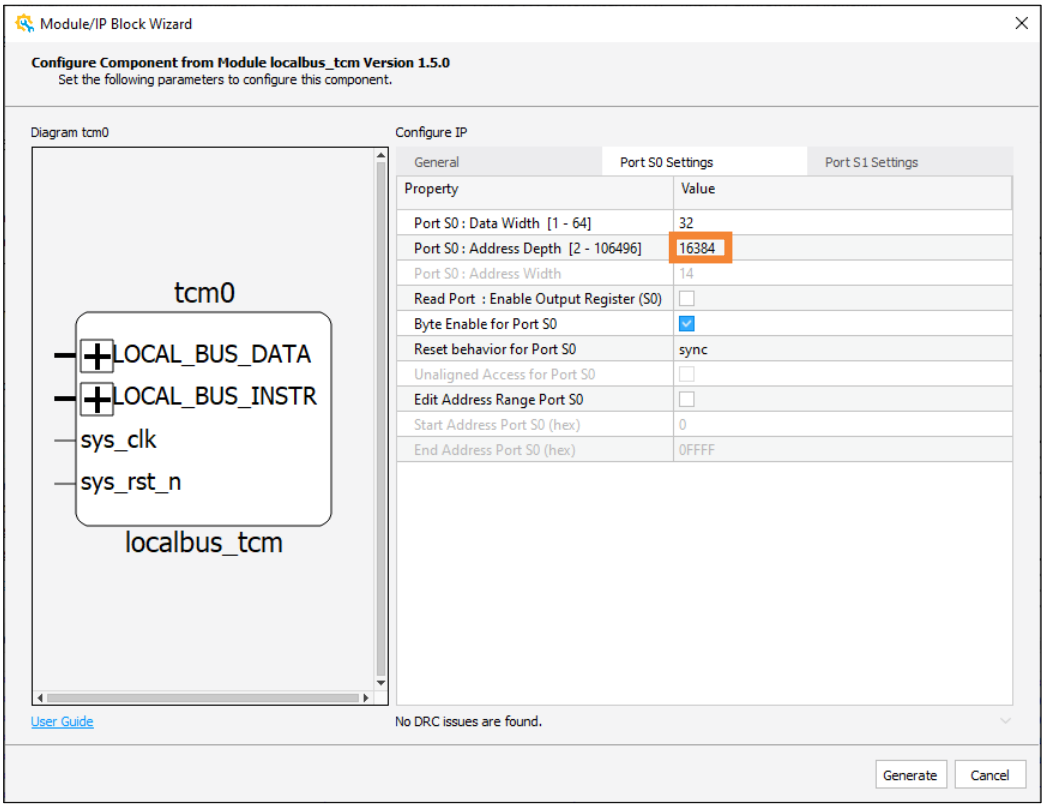


Figure A.9. tcm0_inst Port S0 Address Depth Settings 2

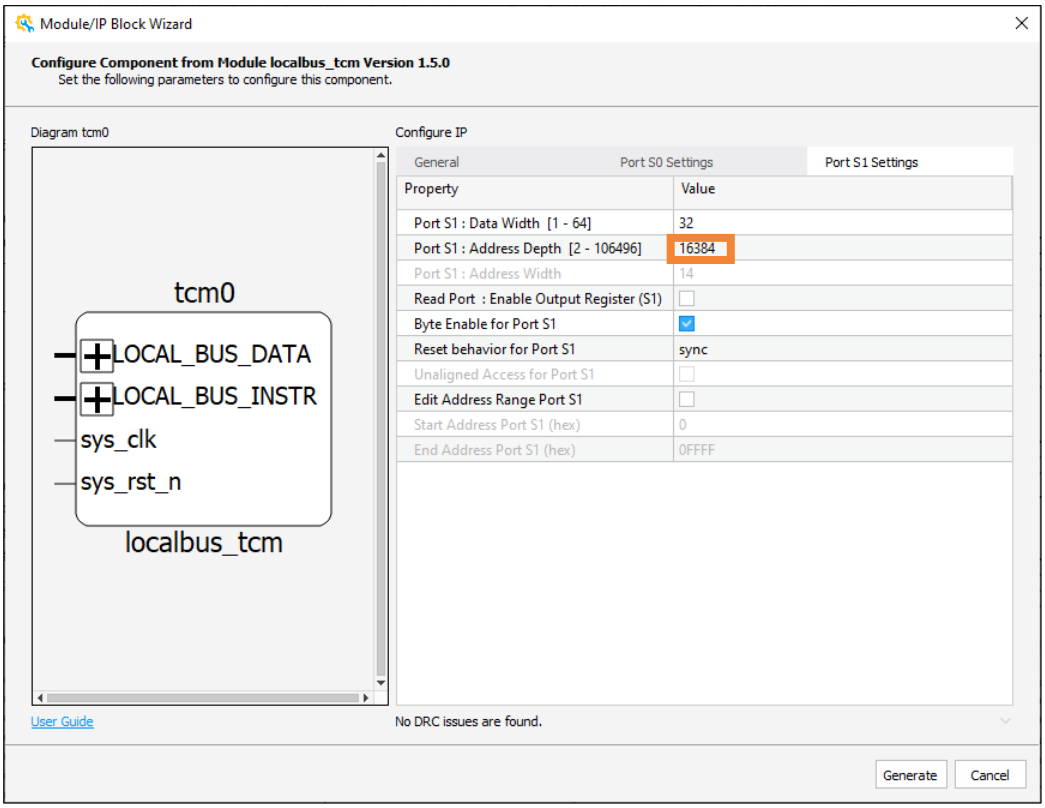


Figure A.10. tcm0_inst Port S1 Address Depth Settings 2

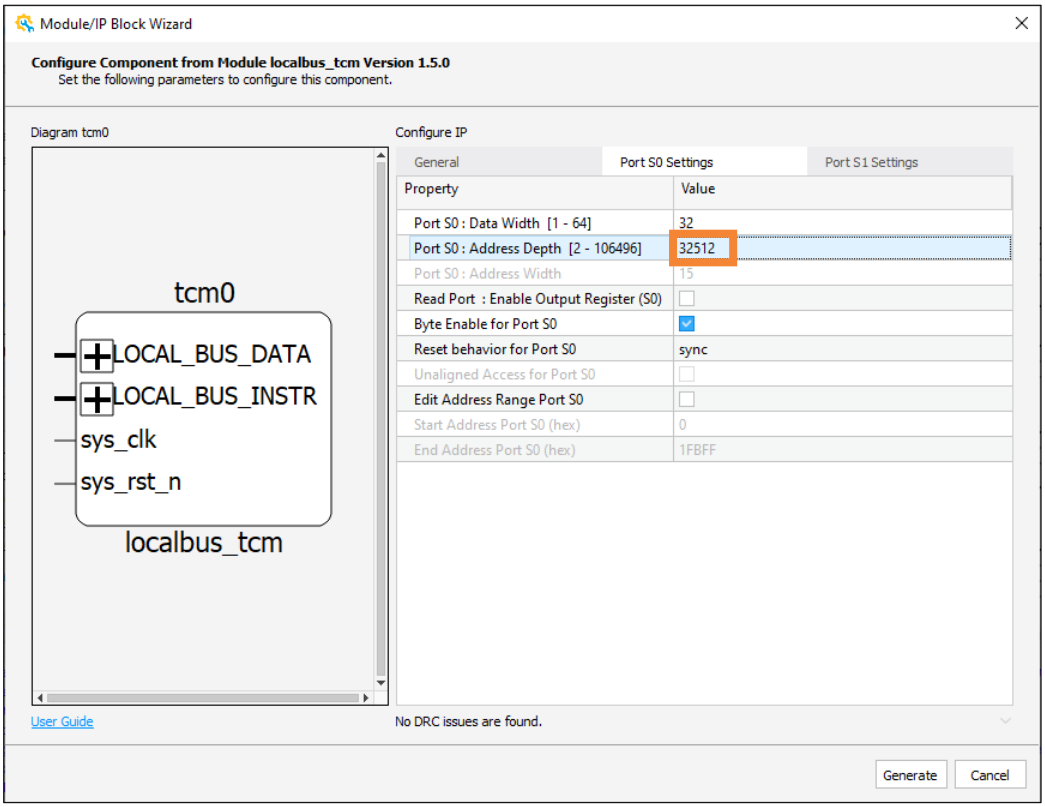


Figure A.11. modify tcm0_inst Port S0 Address Depth Settings 3

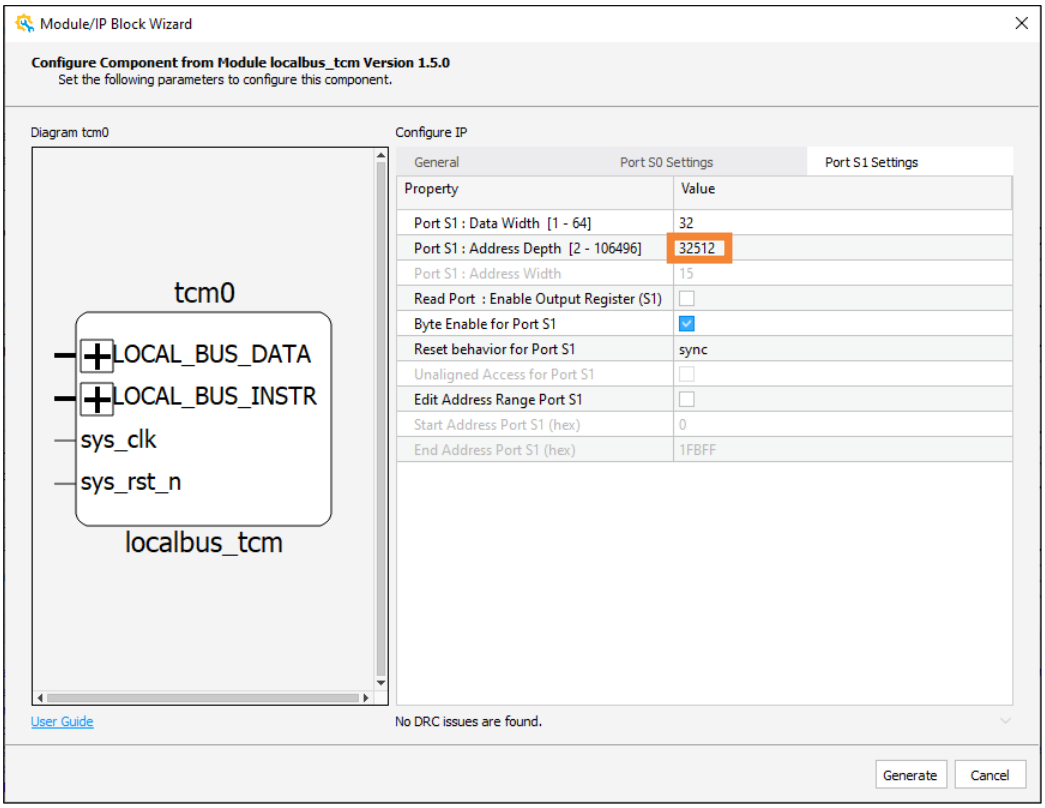


Figure A.12. modify tcm0_inst Port S1 Address Depth Settings 3

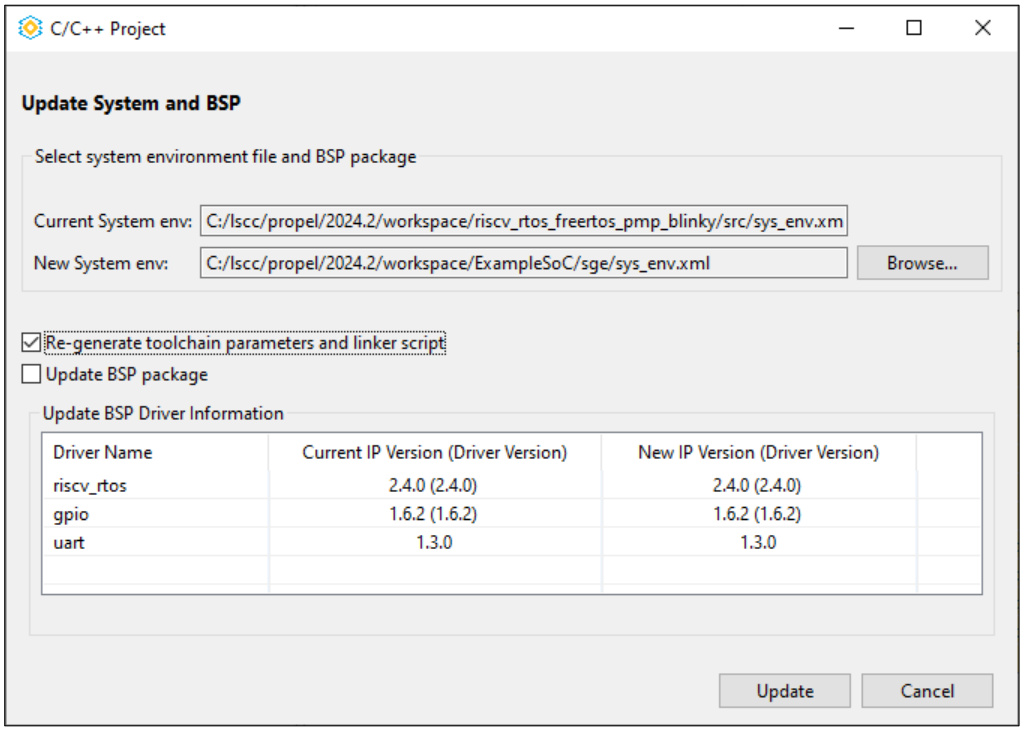


Figure A.13. Update System and BSP

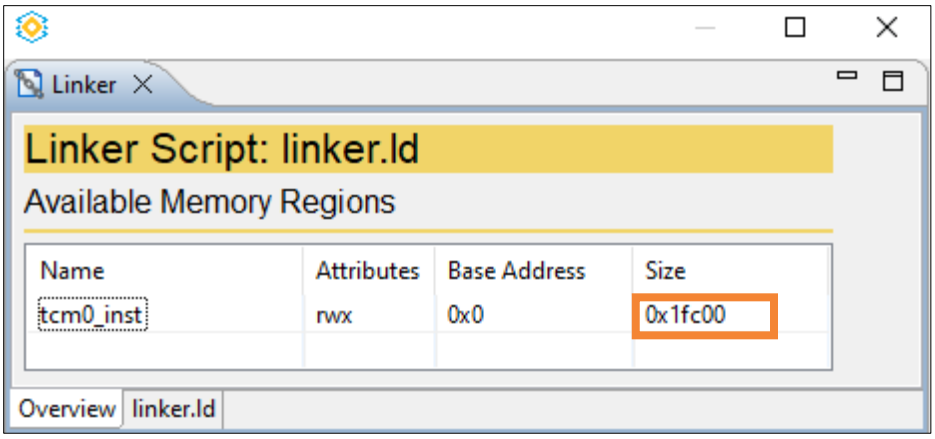


Figure A.14. Updated Linker Script

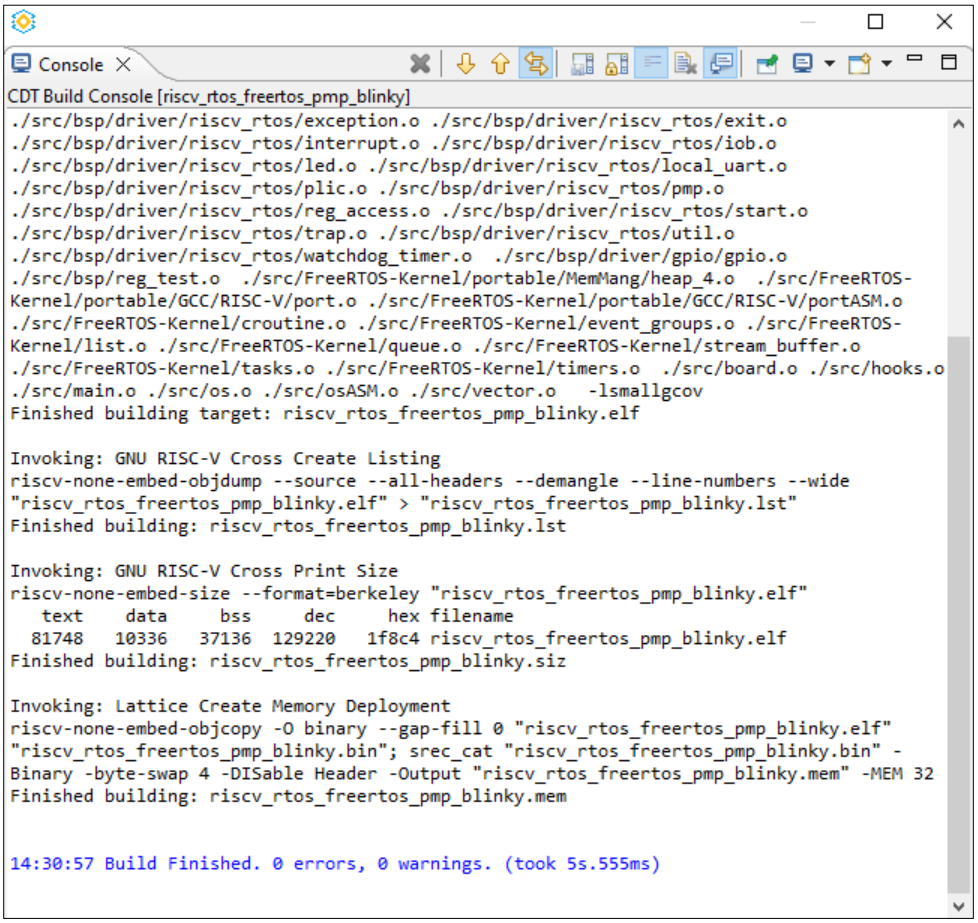


Figure A.15. Build Project Console 2

Appendix B. Standard C Library Support

Lattice Propel SDK bundles Picolibc as the standard library support (https://keithp.com/picolibc/). Picolibc is a set of standard C libraries, both libc and libm, designed for smaller embedded systems with limited ROM and RAM. Picolibc includes code from Newlib (https://sourceware.org/newlib/) and AVR Libc (https://www.nongnu.org/avr-libc/).

Printf and Scanf Levels in Lattice Propel SDK

Lattice Propel SDK provides three levels of printf support with the support of Picolibc, which can be selected when creating a Lattice C/C++ project (Figure B.1), or be modified in the toolchain settings in the project properties after the project is created, noting that the options in both compiler and linker need to be set (Figure B.2 and Figure B.3).

- Integer only printf (-DPICOLIBC_INTEGER_PRINTF_SCANF). This is the default selection in Lattice Propel SDK, which removes the support for all float and double conversions to save code size.
- Float only printf (-DPICOLIBC_FLOAT_PRINTF_SCANF). It requires a special macro for float values: `printf_float`. To make it easier to switch between that and other two levels, that macro should also be correctly defined for the other two levels.

  Here is a sample program to demonstrate the usage:

```
#include <stdio.h>

void main(void) {
      printf(" 2^61 = %lld, Pi = %.17g\n", 1ll << 61,
printf_float(3.141592653589793));
}
```

- Full printf (-DPICOLIBC_DOUBLE_PRINTF_SCANF). This offers full printf functionality, including both float and double conversions.

System Library Interfaces Used in Lattice Propel SDK

Lattice Propel SDK provides three system library for stdio support with the help of Picolibc, which can be selected when creating a Lattice C/C++ project (Figure B.1), or be modified in the toolchain settings in the project properties after the project is created, noting that only the option in linker needs to be set (Figure B.3).

- Default. Use the default system library interface (UART) in BSP, this requires a UART instance inside the SoC design. The default library is implemented in the processor driver code and no additional linker options are required.
- Semihosting (--oslib=semihost). Semihosting is a mechanism that enables code running on the target to communicate with and use the I/O of the host computer. Lattice Propel SDK provides semihosting support in On-Chip-Debugging flow. It allows printing messages to the debugger console without relying on the UART instance, and it also supports file I/O.
- Dummyhosting (--oslib=dummyhost). Dummy stdio hooks. This allows programs to link without requiring any system-dependent functions. This is only used if the program does not provide its own version of stdin, stdout, and stderr.

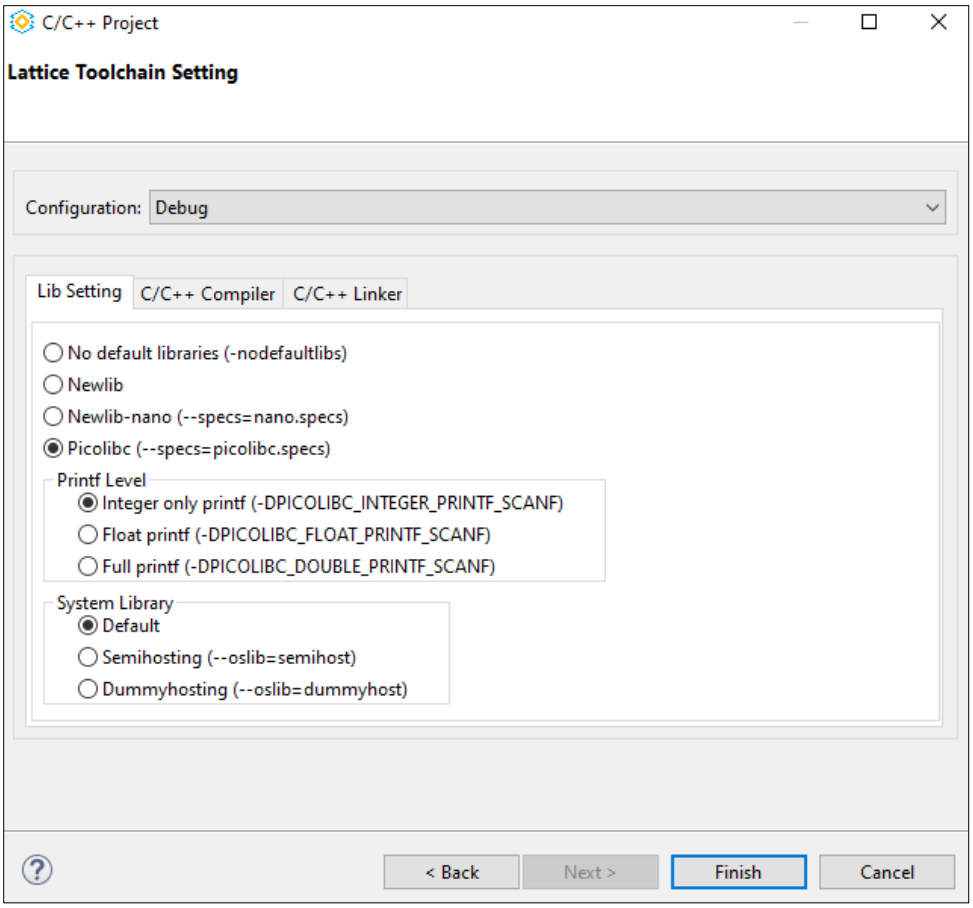


Figure B.1. Lattice Toolchain Setting Dialog 2

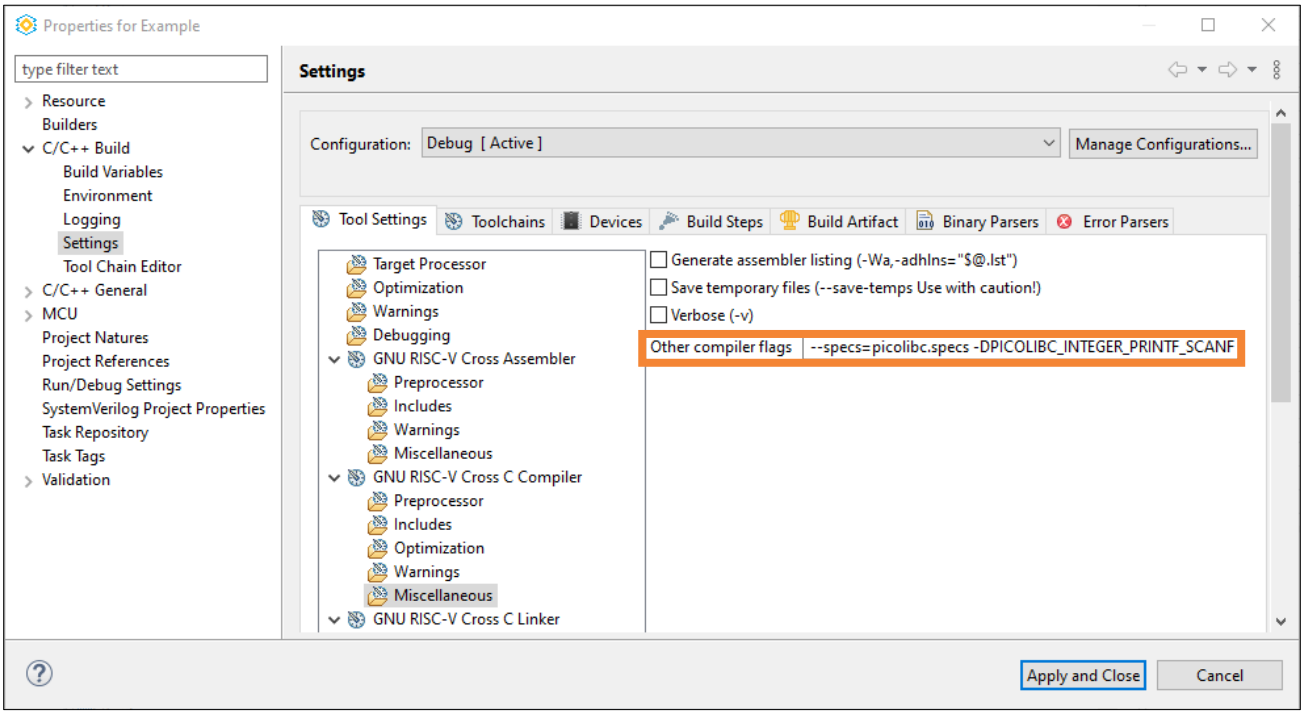


Figure B.2. Properties of C/C++ Project – Compiler Options

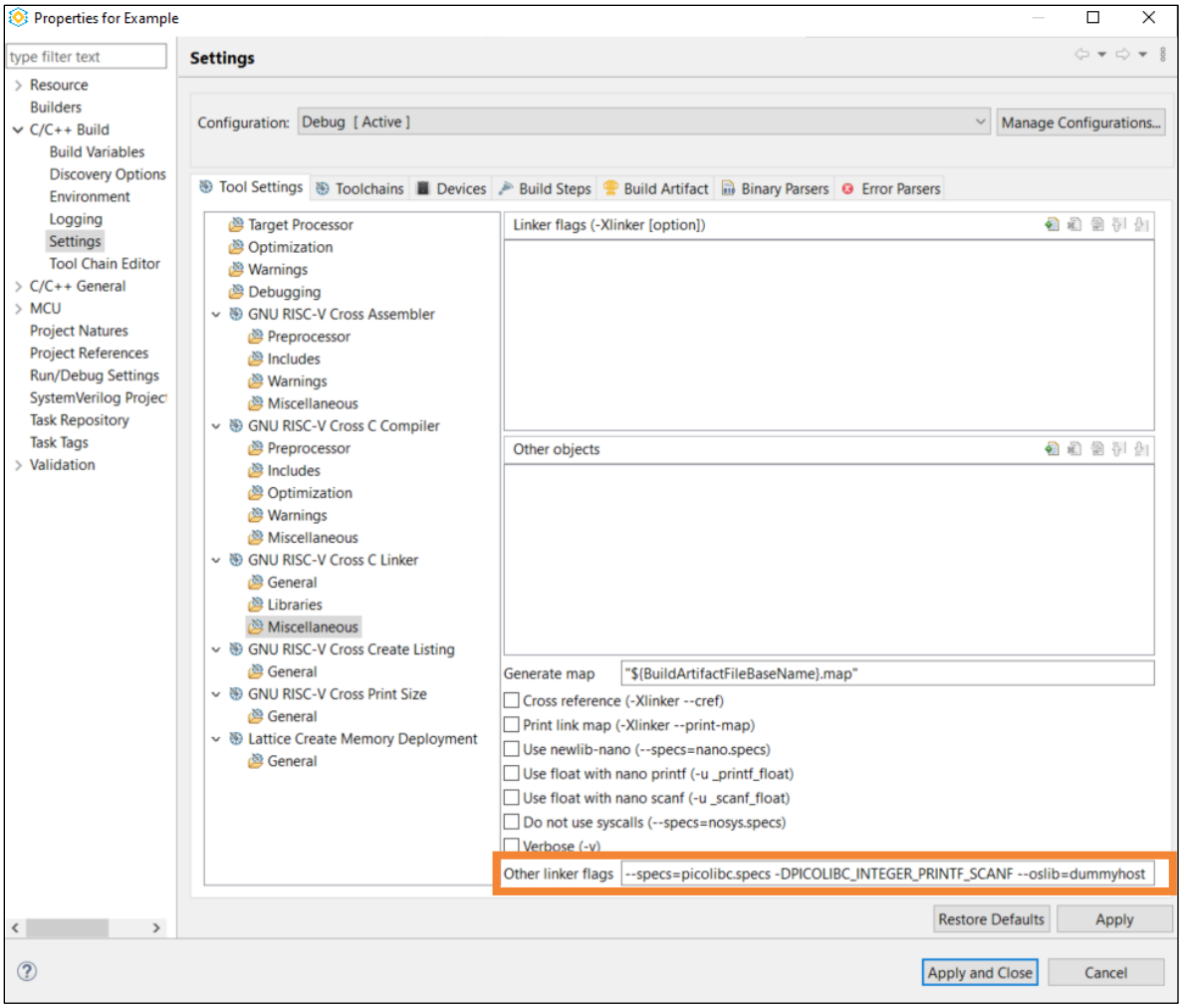


Figure B.3. Properties of C/C++ Project – Linker Options

Appendix C. Third-party Command-line Tools in Lattice Propel SDK

Lattice Propel SDK integrates with a number of third-party command-line tools, which can be used with Lattice Propel User Interface or as stand-alone command-line tools. You can find these command-line tools and their documentation according to the following list.

1. Windows Build tools

   Version: 4.2.1-1

   Binaries Location: <*Propel Installation Location*>\sdk\build_tools\bin

   Documents Location: <*Propel Installation Location*>\sdk\build_tools\share

2. OpenOCD

   Version: 0.10.0

   Binaries Location: <*Propel Installation Location*>\sdk\openocd\bin

   Documents Location: <*Propel Installation Location*>\sdk\openocd\doc

3. GNU RISC-V Embedded GCC

   Version: 10.2.0-1.2

   Binaries Location: <*Propel Installation Location*>\sdk\riscv-none-embed-gcc\bin

   Documents Location: <*Propel Installation Location*>\sdk\riscv-none-embed-gcc\share\doc

4. Picolibc

   Version: 1.7.4

   Binaries Location: <*Propel Installation Location*>\sdk\riscv-none-embed-gcc\riscv-none-embed\picolibc\riscv-none-embed\lib

   Documents Location: <*Propel Installation Location*>\sdk\riscv-none-embed-gcc\doc

5. SRecord

   Version: 1.6.4

   Binaries Location: <*Propel Installation Location*>\sdk\tools\bin

   Documents Location: <*Propel Installation Location*>\sdk\tools\bin

6. QEMU RISC-V

   Version: 7.2.5

   Binaries Location: <*Propel Installation Location*>\sdk\qemu-riscv\bin

   Documents Location: <*Propel Installation Location*>\sdk\qemu-riscv\bin

Appendix D. Command-line Environment Setting Script in Lattice Propel SDK

Lattice Propel SDK provides a script to set necessary environment variables for running all Lattice Propel SDK command line tools. It enables you to fast get up to using command line tools, especially for Lattice specific tools.

To use the script:

1. Run the script propel_commandline.cmd (Windows) or propel_commandline.sh (Linux) under <*Propel Installation Location*>.

   The message Lattice Propel Commandline is printed, and it lets you go to a new command line interface.

2. All command line tools in the Propel SDK can be run from this command line interface without any special configuration.

   Here are examples of using the command line tools within this script.

   Example A: Build a C project under command line.

   ```
   $ cd <C project Location>/Debug
   $ make
   ```

   Example B: Launching openocd under command line.

   ```
   $ cd <C project Location>
   $ openocd -c "gdb_port 3333" -c "set target 0" -c "set tck 1"  -c "set port
   FTUSB-0" -c "set RISCV_SMALL_YAML src/cpu.yaml" -f interface/lattice-cable.cfg
   -f target/riscv-small.cfg
   ```

Appendix E. Debugging with Attach to Running Target

The Attach to running target function means to do on-chip debugging on a running board, without resetting CPU and reloading the .elf file to the board.

This function is particularly useful when debugging a system whose CPU runs software from preloaded memory.

The debugger is used to check CPU registers for narrowing down software problems.

Setting Memory Initialization to SoC Project

1. Open the corresponding SoC Project of the debugging C project.
2. Set Memory Initialization file (Figure E.1). Use the C project memory file. Refer to Appendix A on how to create memory files.
3. Re-generate the programing file by running Lattice Radiant software in this SoC Project.
4. Program the programing file into the target device board.
5. The board can now run itself after power on. Keep it running.

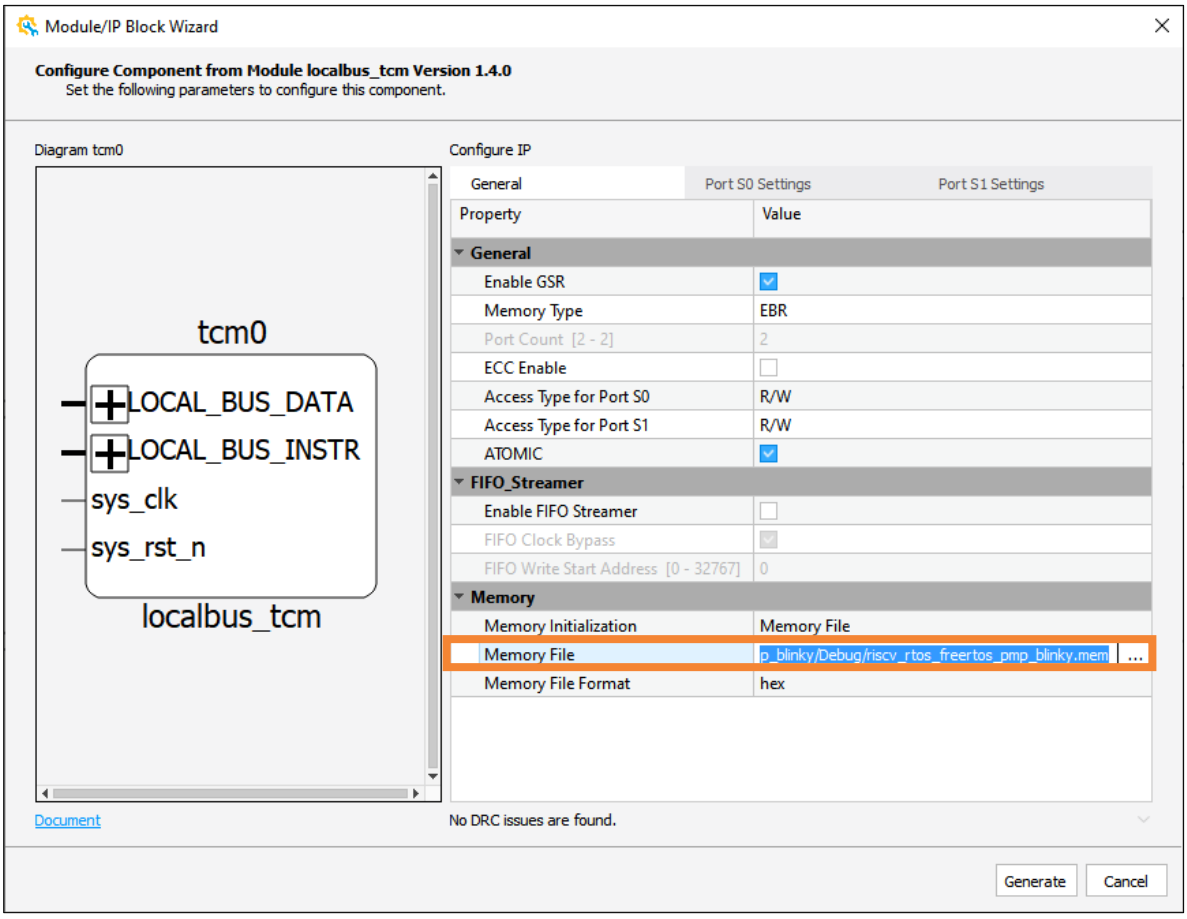


Figure E.1. Set Memory Initialization

Attach to Running Target

1. In the **Project Explorer** view, select the corresponding C project.
2. Select **Run** > **Debug Configurations…**. Select the corresponding item (Figure E.2).
3. Select **Startup** tab. Check **Attach to running target** checkbox (Figure E.2). Click **Apply**. Click **Debug**.

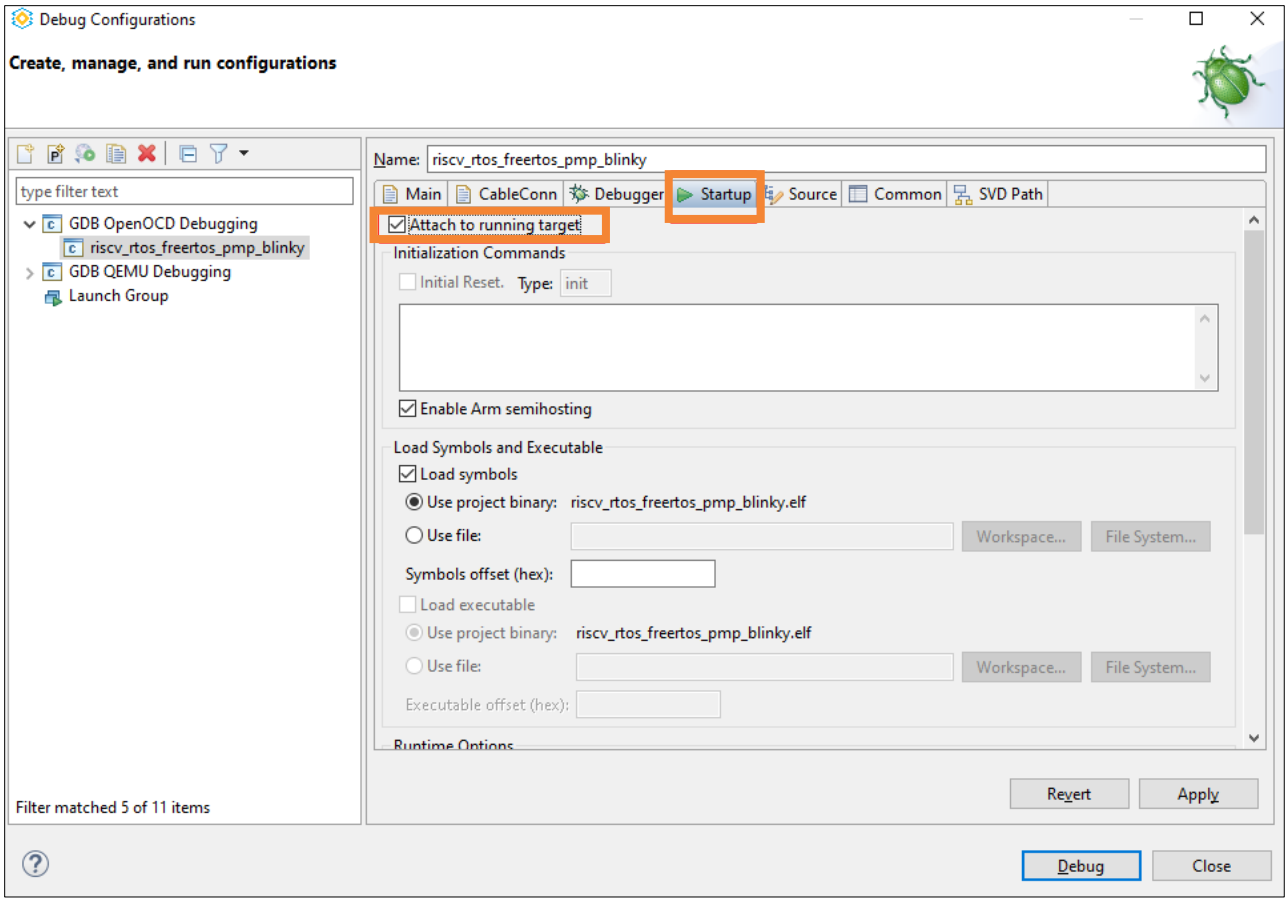


Figure E.2. Debug Configurations Dialog 8

4. Wait for a few seconds for switching to the debug perspective, as shown in Figure E.3. Click **View Disassembly** to display assembly code or C code.

Note: The C project should correspond to the memory file in the running board. If you modify the code, the C project should be re-built and you need to go through steps in Set Memory Initialization to SoC Project again. If the memory file in the running board does not correspond to the C project, it causes misalignment in running line and symbol table and you get error information.

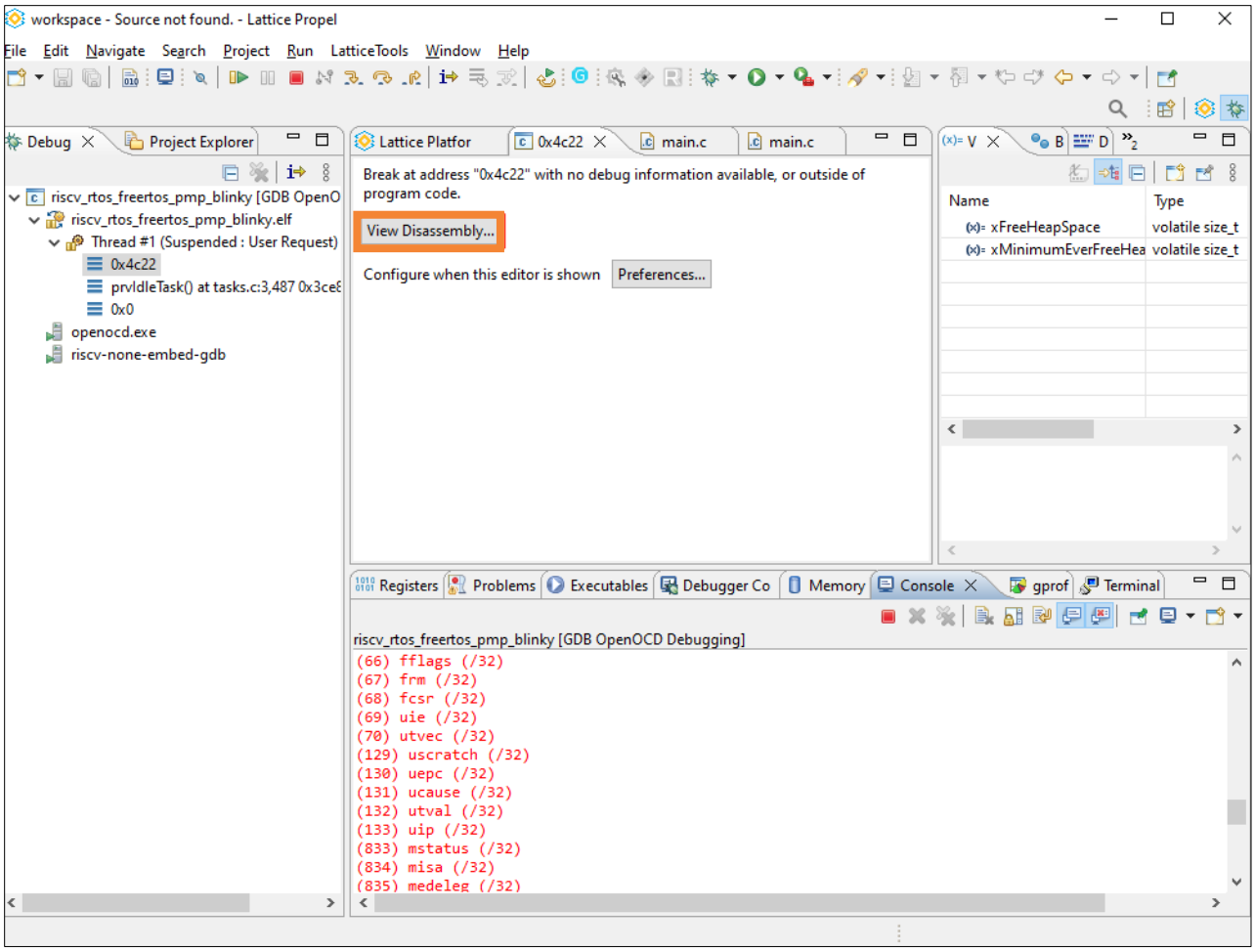


Figure E.3. Debug Perspective 2

Switching Back to Default Mode

If checking and unchecking the Attach to running target function several times, the configuration might become incorrect. The suggested operation is to click **Restore default** (Figure E.4).

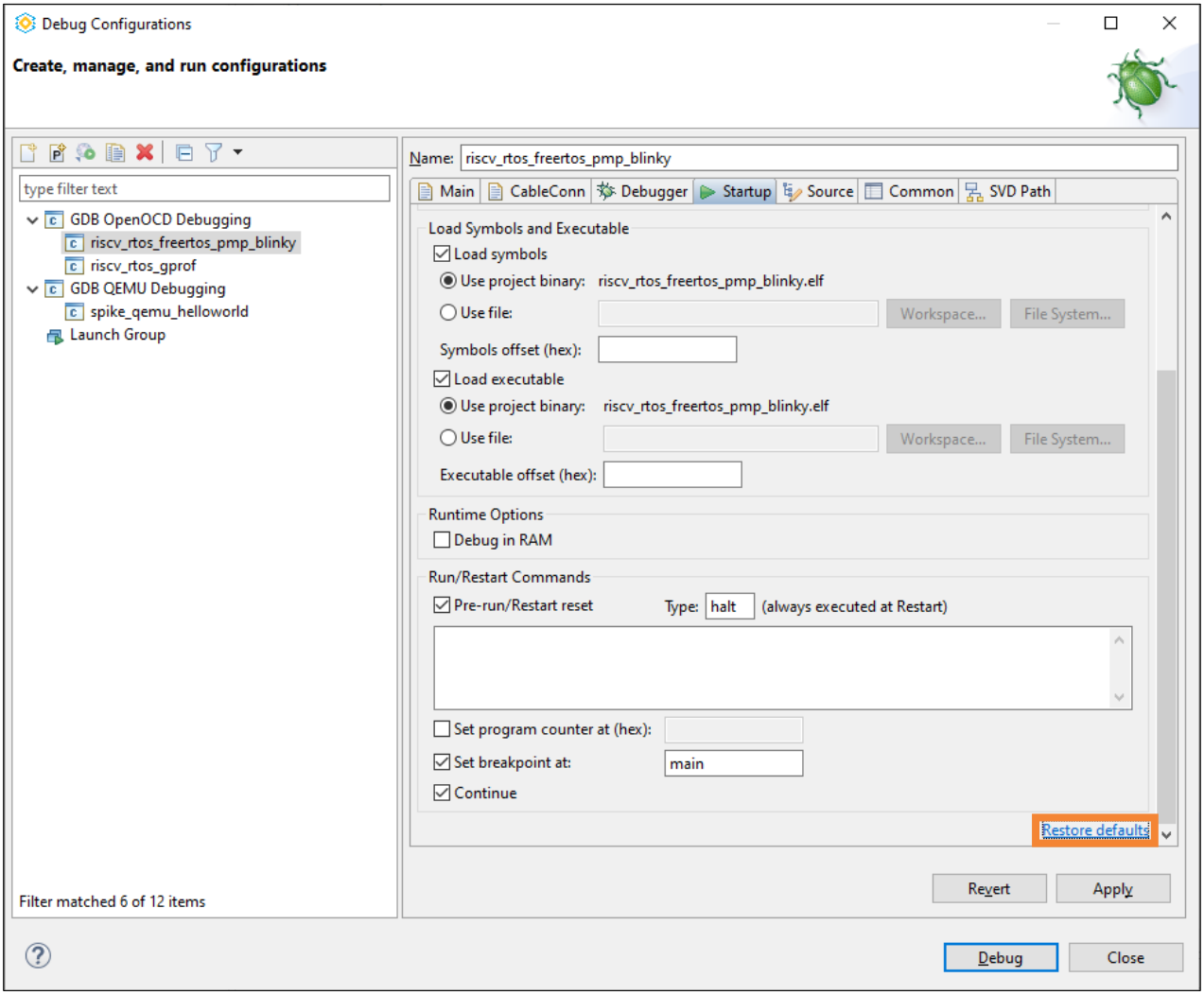


Figure E.4. Restore Defaults

Appendix F. Register Access Test

The register access test is to test the accessibility of the design by accessing the assessable registers of some testable IP instances. It might affect the environment during testing.

Note: Approach this test with caution, as it is designed to do connectivity check for the SoC project. Make sure to disable the test while developing the application firmware.

Generate Test Code

1. Create an SoC project. Refer to the Creating an SoC Design Project section for more details.
2. Build this SoC project by Lattice Propel SDK or Builder. Then, you can find the test files shown in Figure F.1.
3. Create a corresponding C project, choose an example application that supports IP register access test (Figure F.2).

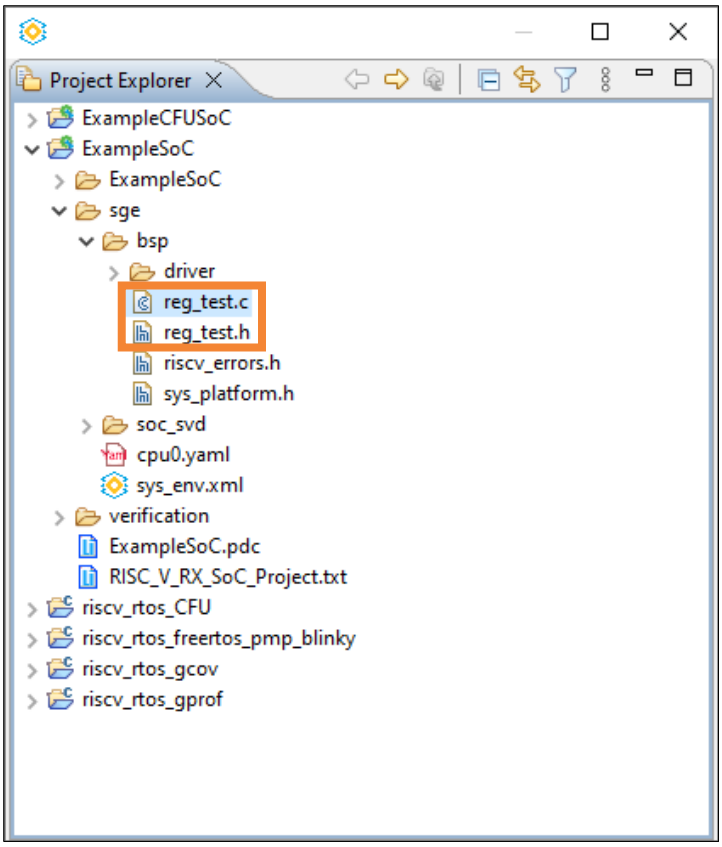


Figure F.1. Project Explorer

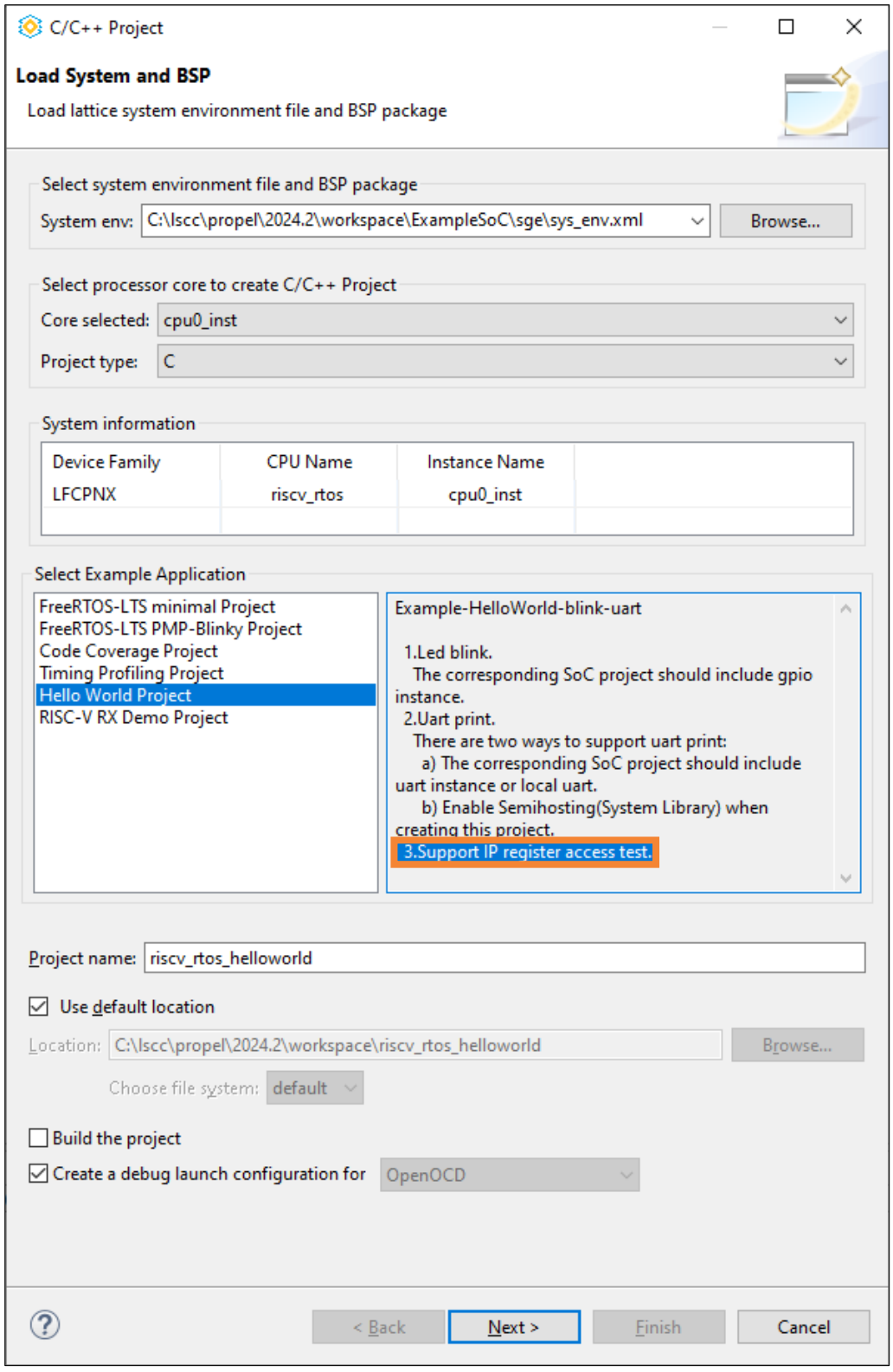


Figure F.2. Load System and BSP Page 9

Enable Test Code

This function is disabled by default. You need to enable this function manually.

1. Find the test entrance from the C project just created (Figure F.3).
2. Set `define REG_TEST_ENABLE` to 1 in the file **sys_platform.h** (Figure F.4).

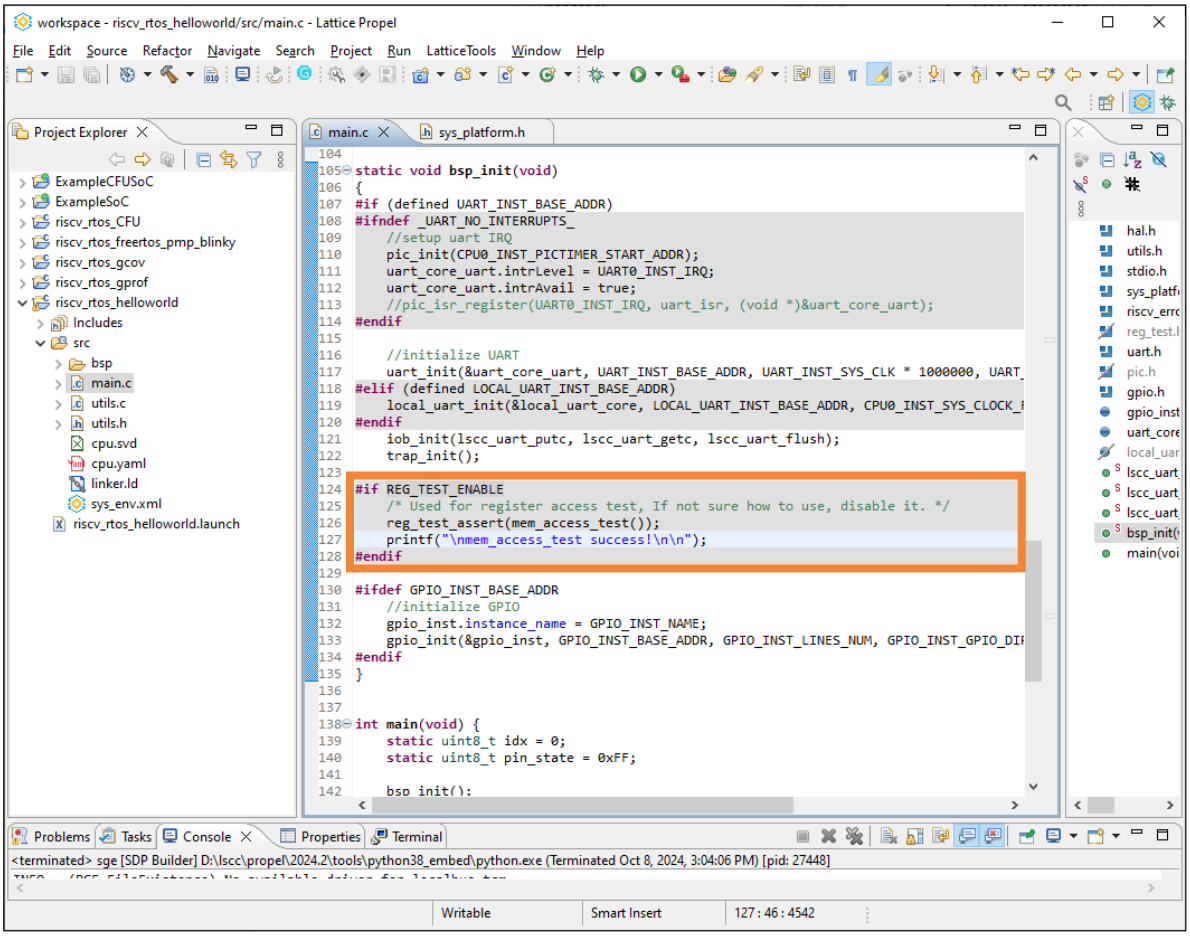


Figure F.3. Test Entrance

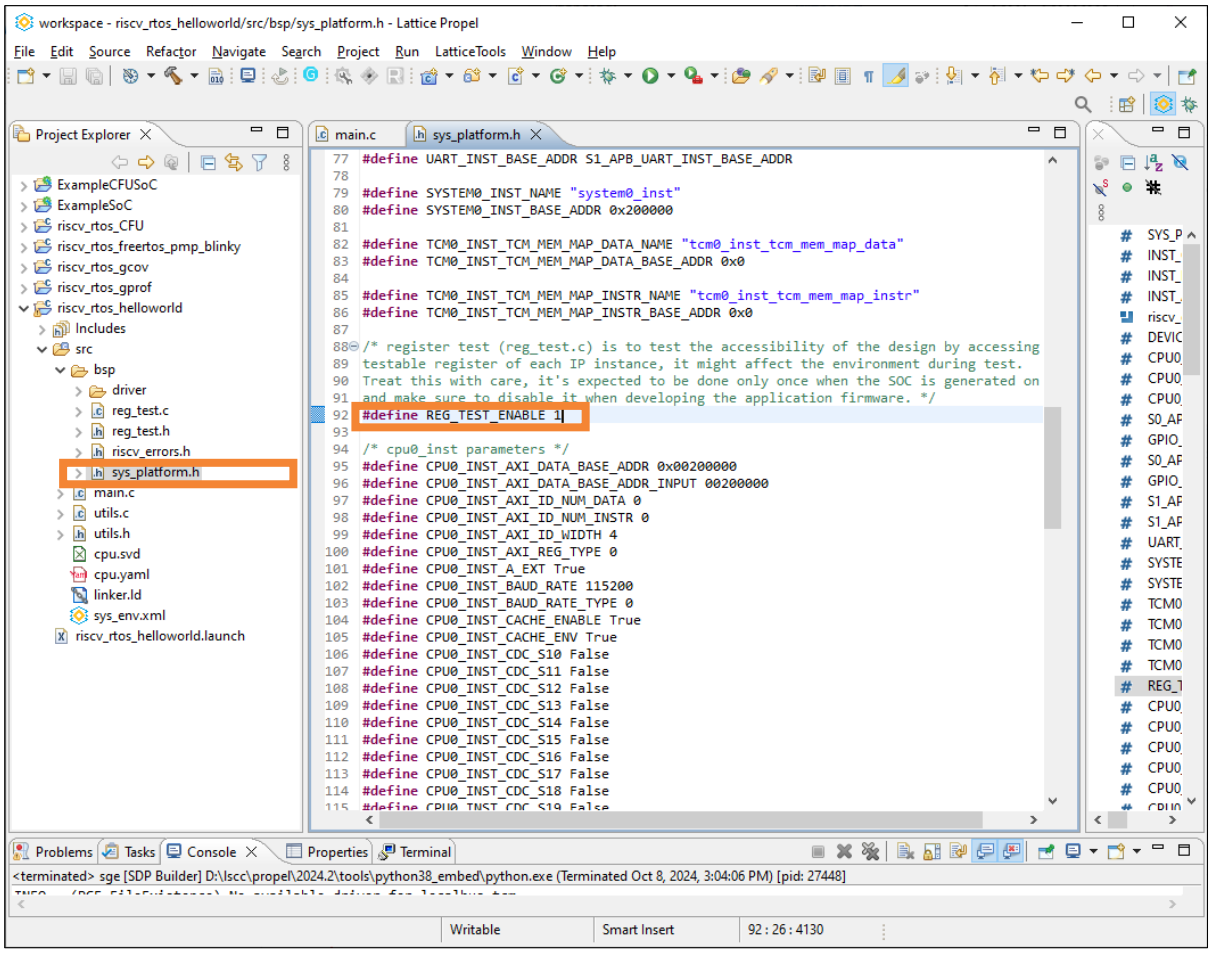


Figure F.4. Enable Test Code

Running Test

1. Generate the bit file from the SoC project just created. Then, program the bit file to the corresponding board.
2. Build the C project just created.
3. Run on-chip debug. Check the terminal printf log.
4. Success log (Figure F.5) or failure log (Figure F.6) is shown.

Note: This function is an advanced option. The code changes the register value, and it may cause some IP error after running the test code. Consequently, it is recommended to disable this function after the test.

Figure F.5. Success Log

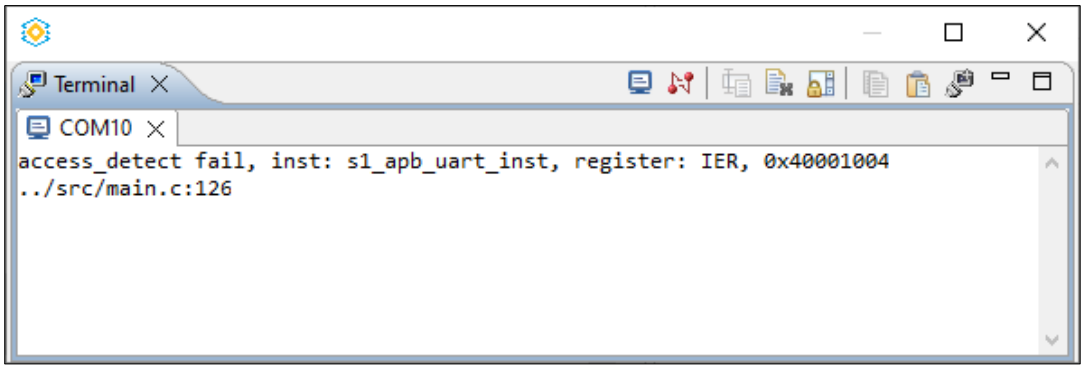


Figure F.6. Failure Log

References

- Lattice Propel 2024.2 Release Notes (FPGA-AN-02095)
- Lattice Propel 2024.2 Installation for Windows User Guide (FPGA-AN-02093)
- Lattice Propel 2024.2 Installation for Linux User Guide (FPGA-AN-02094)
- Lattice Propel Builder 2024.2 Usage Guide (FPGA-UG-02219)
- IP Packager 2024.2 User Guide (FPGA-UG-02220)
- Lattice Propel Revision Control User Guide (FPGA-UG-02221)
- MachXO3D Family Data Sheet (FPGA-DS-02026)
- MachXO3D Programming and Configuration Usage Guide (FPGA-TN-02069)
- MachXO3D Breakout Board User Guide (FPGA-UG-02084)
- CertusPro-NX Evaluation Board User Guide (FPGA-EB-02046)

For more information, refer to:

- Lattice Propel Design Environment Web Page
- Lattice Insights for Training Series and Learning Plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, November 2024

Section	Change Summary
All	Production release.

www.latticesemi.com