Lattice Propel 2025.1 SDK

User Guide

FPGA-UG-02234-1.0

June 2025

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Contents

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figures

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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
CLINT	Core Local Interruptor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DUT	Design Under Test
GUI	Graphical User Interface
FD-SOI	Fully-Depleted Silicon On Insulator
FMC	FPGA Mezzanine Card
FPGA	Field Programmable Gate Array
FreeRTOS	A market-leading RTOS for microcontrollers and small microprocessors
FTDI	Future Technology Devices Intl.Ltd
HDL	Hardware Description Language
HPC	High Pin Connector
IBIS	Input Output Buffer Information System
IDE	Integrated Development Environment
I2C	Inter-Integrated Circuit
IP	Intellectual Property
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
MSIP	Machine-Mode Software Interrupt
OCD	On-Chip-Debugging
OEM	Original Equipment Manufacturer
OpenOCD	Open On-Chip Debugger
PC	Personal Computer
PIC	Programmable Interrupt Controller
PLIC	Platform-Level Interrupt Controller
PMOD	Peripheral Module
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 RX	Lattice RISC-V for RTOS Soft IP
RISC-V SM	Lattice RISC-V for State-Machine 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.
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Abbreviation	Definition
UART	Universal Asynchronous Receiver/Transmitter
UFM	User Flash Memory
UI	User Interface
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language
WDT	Watch Dog Timer

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 Builder 2025.1 User Guide (FPGA-UG-02235) 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 or 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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

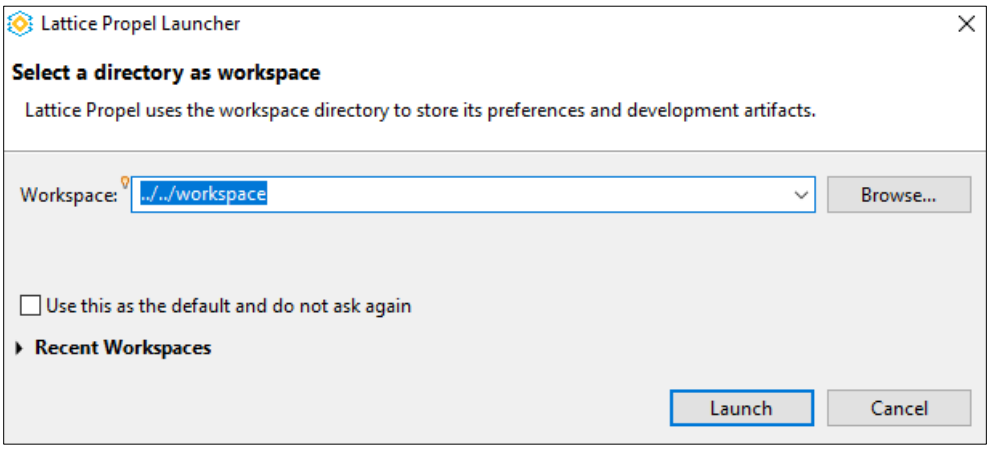


Figure 3.1. Select Workspace Dialog

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

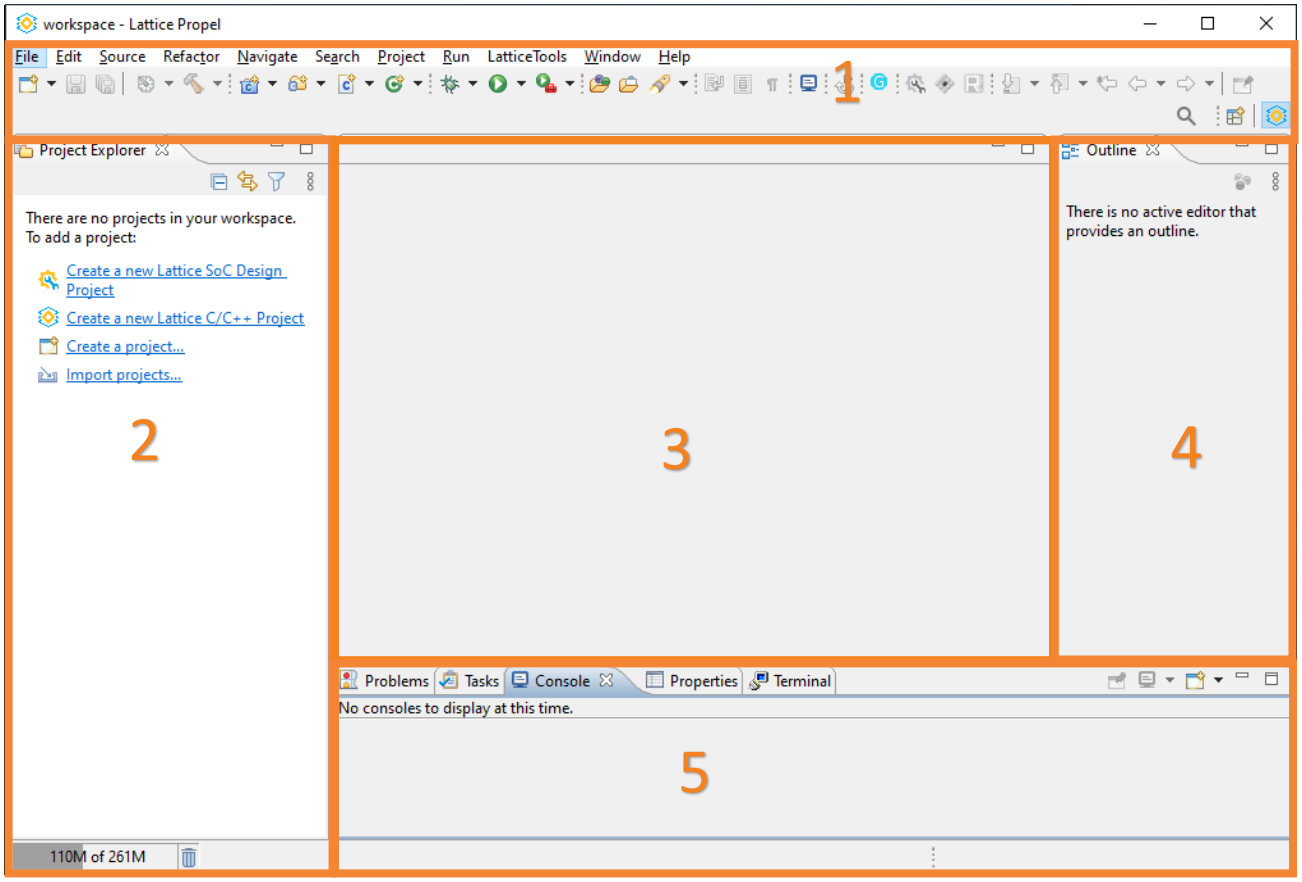


Figure 3.2. Lattice Propel Workbench Window

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).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

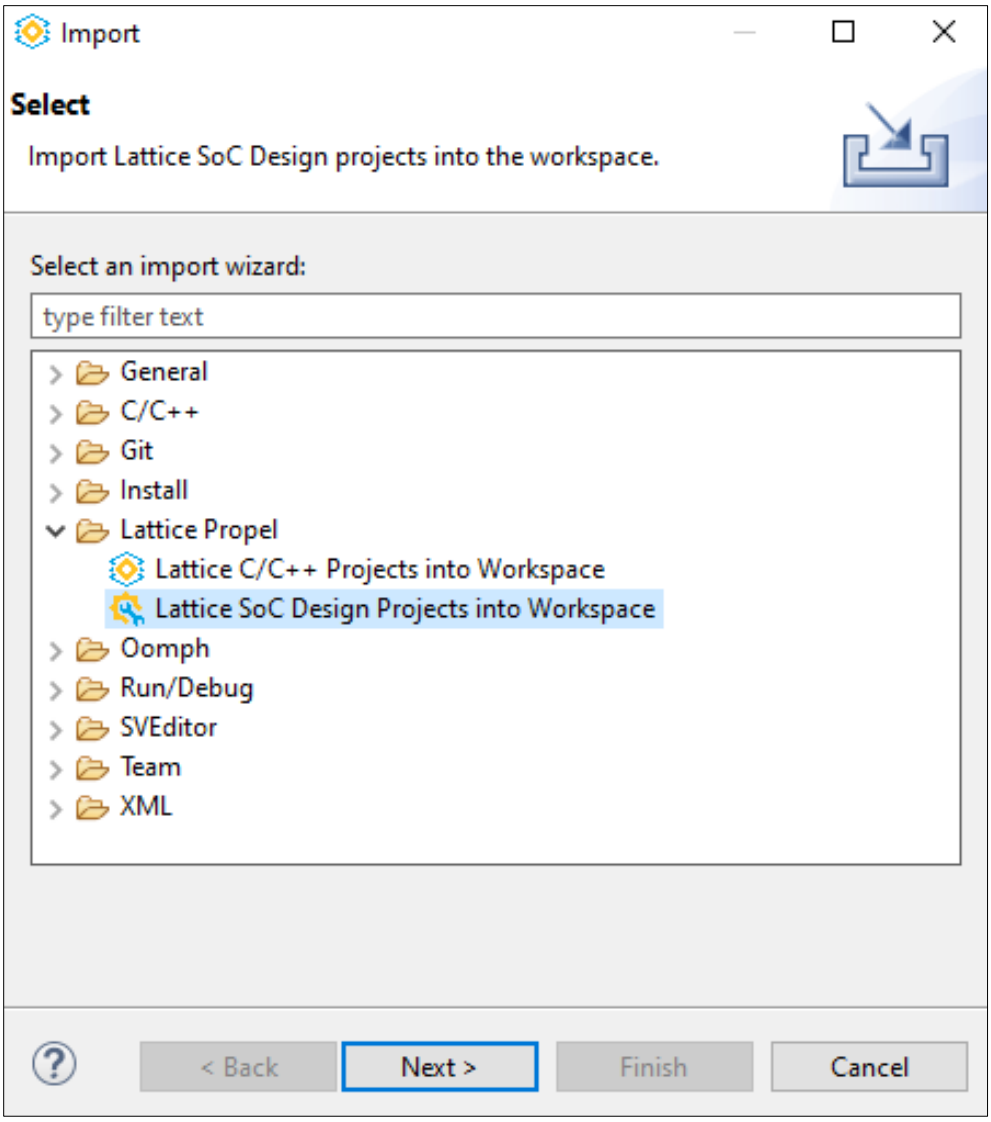


Figure 3.3. Select Wizard – Import Lattice SoC Design Projects

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).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

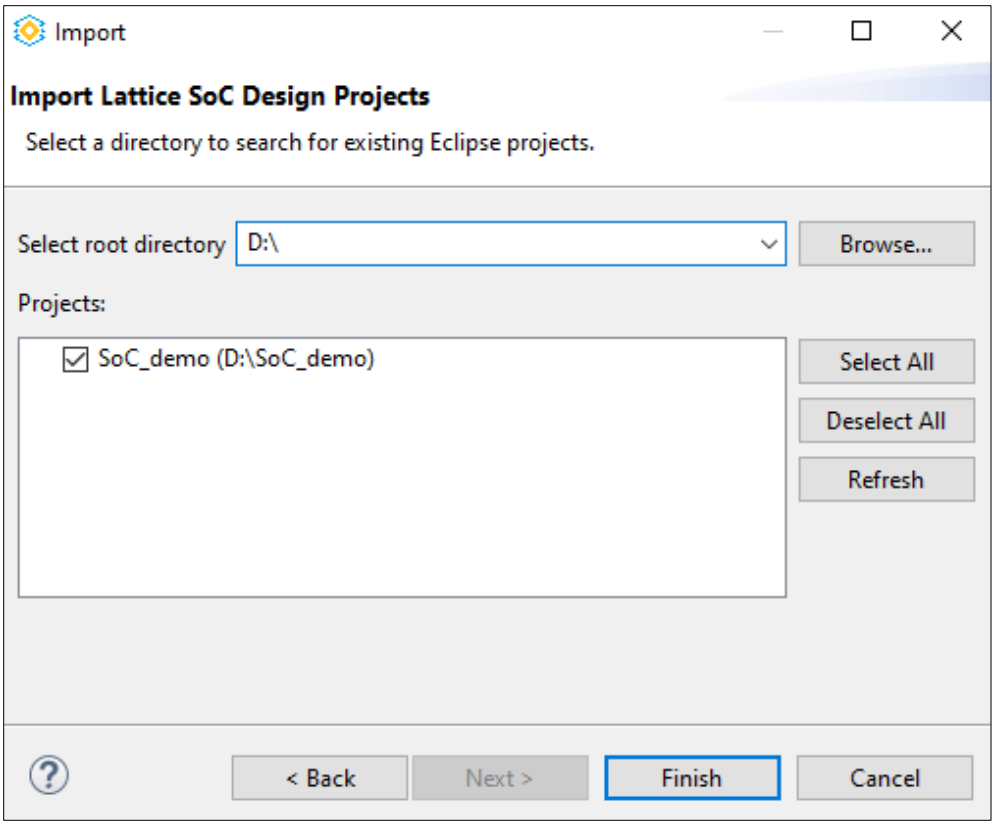


Figure 3.4. Import Lattice SoC Design Projects Wizard

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).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

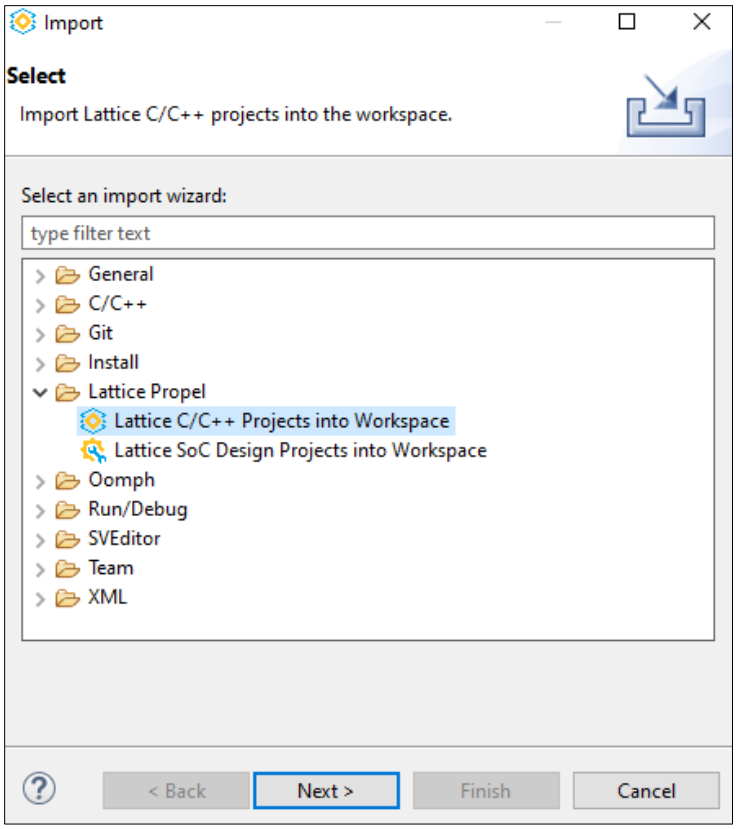


Figure 3.5. Select Wizard – Import Lattice C/C++ Projects

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).

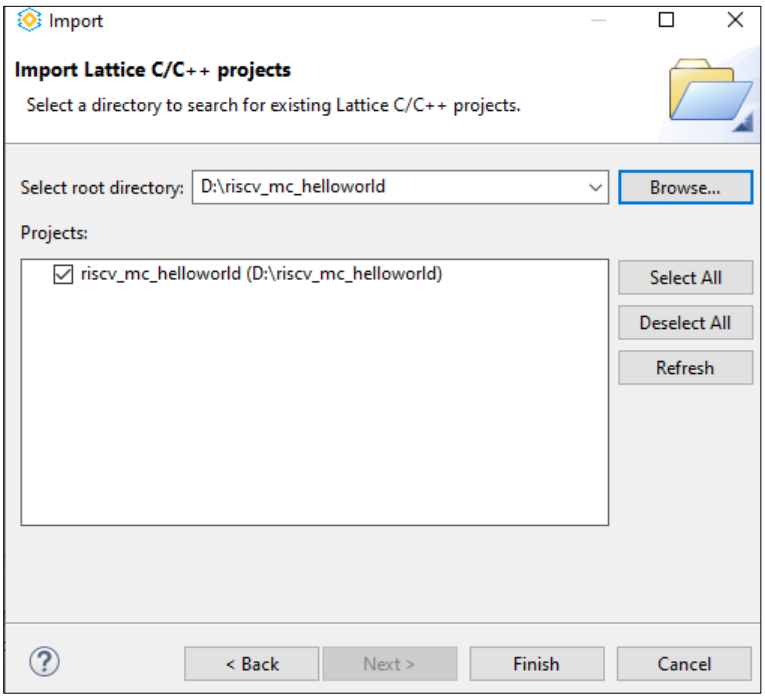


Figure 3.6. Import Lattice C/C++ Projects Wizard

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3.1.4. Creating Customized C/C++ Templates

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.

Note: You must use this template management function on Lattice Propel SDK 2024.2 or later.

1. From Lattice Propel SDK, select a C/C++ project and choose **Project** > **Create Lattice Application Template**. The **Create Application Template** wizard opens Figure 3.7.

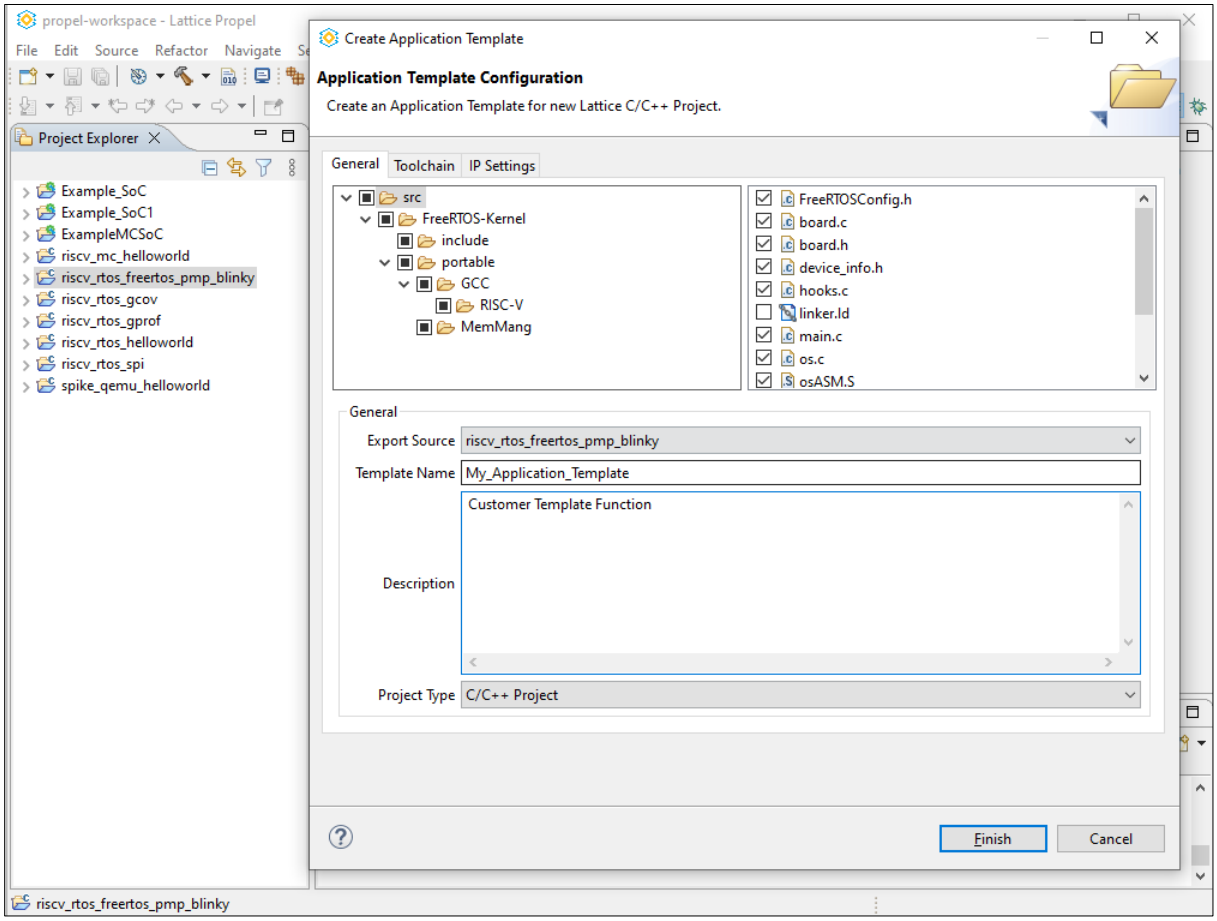


Figure 3.7. Create Application Template – General

2. You can input Template name, Description, Toolchain Configurations, IP Settings, and so on, and select files for creating template.

   Notes:

   - Under the General tab (Figure 3.7), select project code files, all files except linker.ld are checked. It is recommended to keep these selections to avoid potential errors in building.
   - Under the IP Settings tab (Figure 3.8), Propel SDK generates a filter according to your settings. This means if you create a C/C++ project using this C/C++ template, the corresponding SoC project should include the versions of IPs shown in the IP Settings tab. Be careful when modifying the IP-related settings under this tab. If these settings are not set properly, you might encounter errors when using these templates on other versions of Lattice Propel SDK.

3. Click **Finish**.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

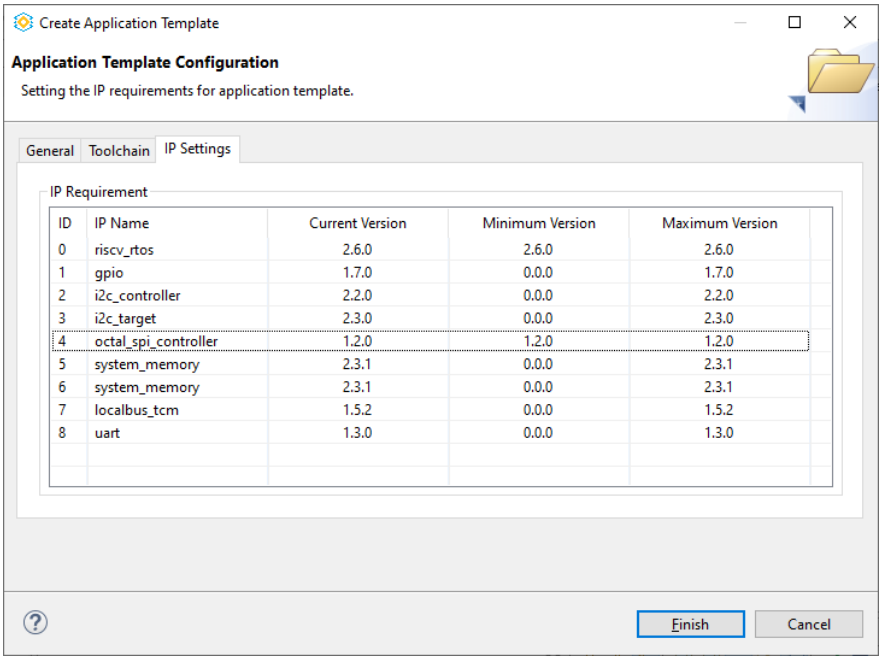


Figure 3.8. Create Application Template – IP Settings

3.1.5. Exporting and Deploying Customized C/C++ Templates

In Lattice Propel SDK, you can export customized C/C++ templates into a single ZIP archive that is ready for deployment in other users' Lattice Propel SDK environments.

Note: You must use this template management function on Propel SDK 2024.2 or later.

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

   The **Select** wizard opens (Figure 3.9).

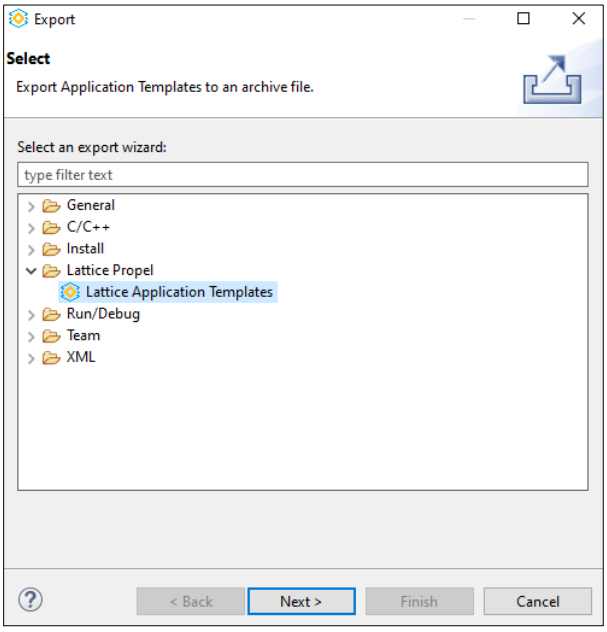


Figure 3.9. Select Wizard for Lattice Application Templates

2. Select **Lattice Propel > Lattice Application Templates**. Click **Next**.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

The **Select** wizard switches to the **Export Lattice Application Templates** wizard page (Figure 3.10).

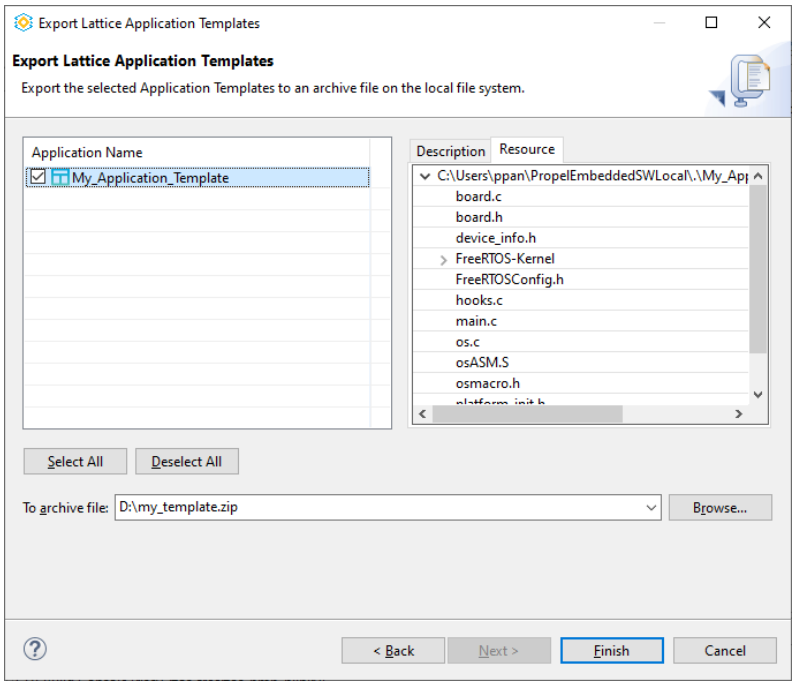


Figure 3.10. Export Lattice Application Templates Wizard

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.

   The exported zip archive file is ready to be delivered to other Lattice Propel SDK users.
6. With this zip archive file, extract it to the **Application Templates Install Path** in the user's environment. Select **Window** > **Preferences** > **Propel Setting** to find this setting (Figure 3.11).

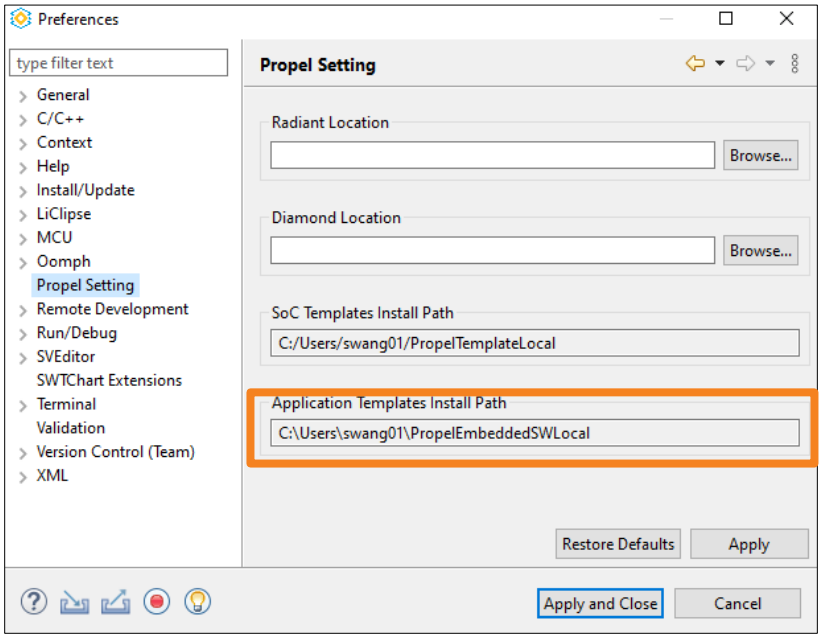


Figure 3.11. Lattice Propel Setting Page

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 are gradually being 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 2025.1 User Guide (FPGA-UG-02235) for more details.

3.2.1. Creating an SoC Design Project (Deprecated)

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 3.12). 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 3.12).

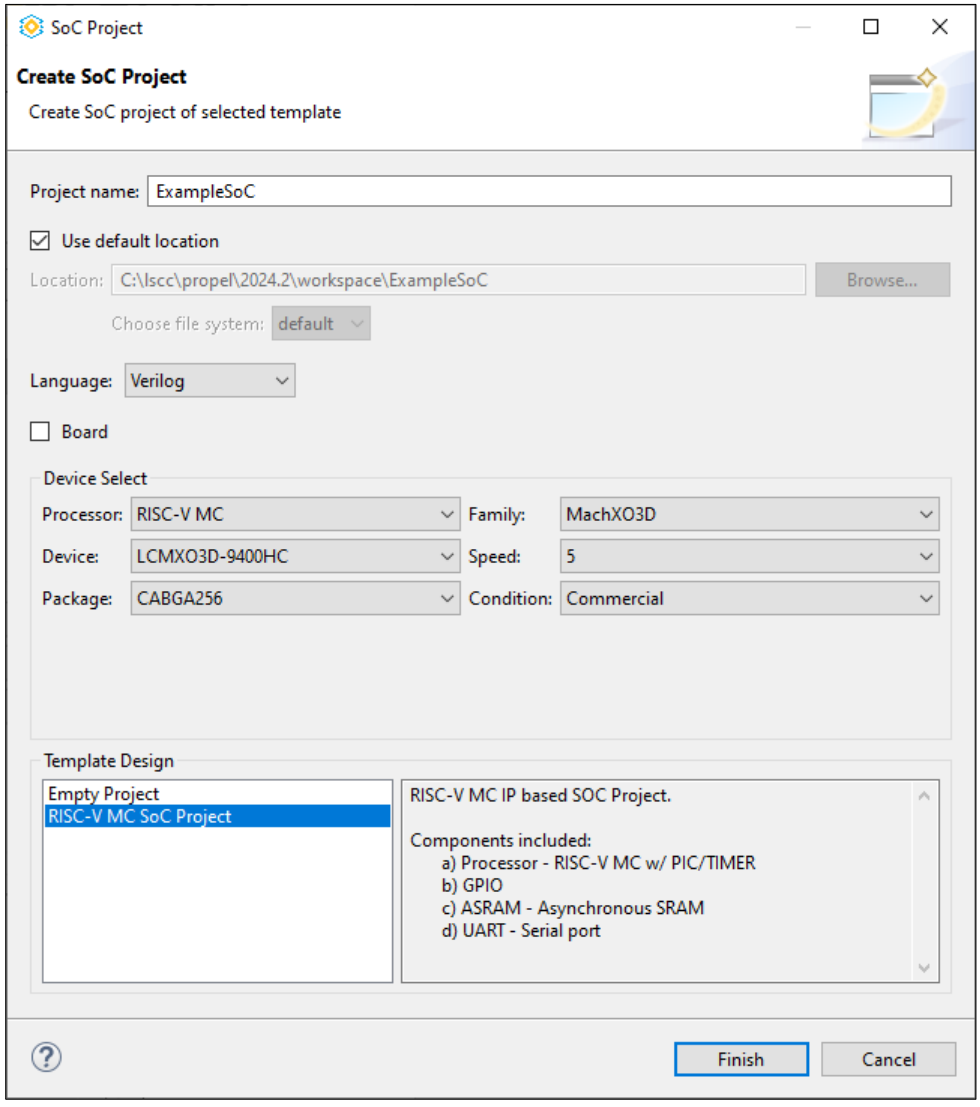


Figure 3.12. Specify a Device for Template SoC Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- Or, to specify a board for a new Template SoC project, check the **Board** checkbox (Figure 3.13).

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

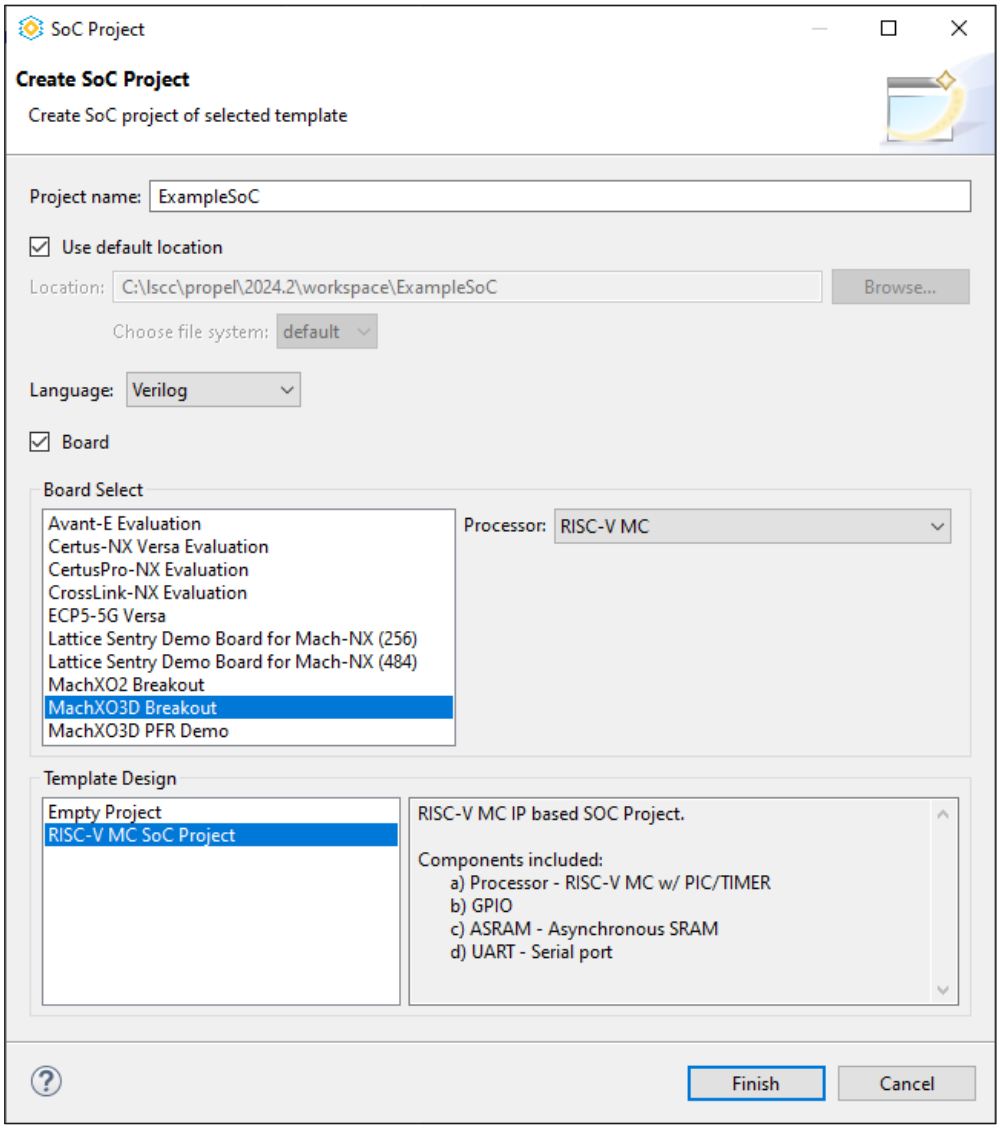


Figure 3.13. Specify a Board for Template SoC Project

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 the workbench, and its design is opened and displayed in Lattice Propel Builder (Figure 3.16).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3.2.2. Opening 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 3.14).

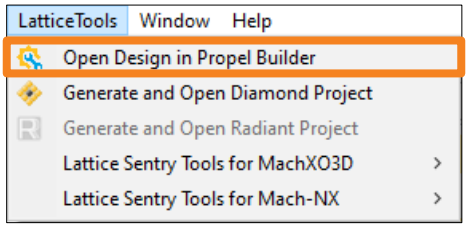


Figure 3.14. LatticeTools Menu

   - 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 pop-up menu (Figure 3.15).

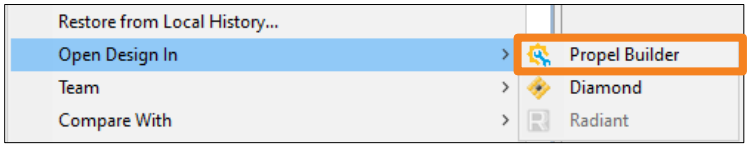


Figure 3.15. Project Explorer Pop-up Menu

3. The SoC Design is opened and displayed in Lattice Propel Builder (Figure 3.16).

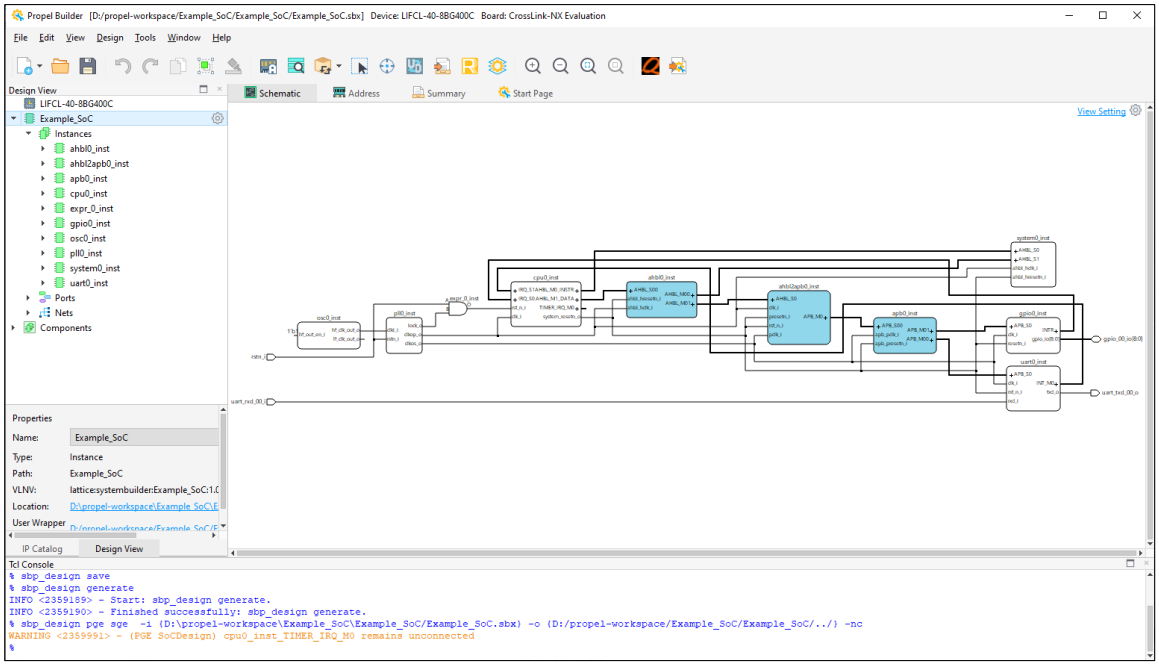


Figure 3.16. Lattice Propel Builder Window

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 the **Empty Project** template inside Lattice Propel Builder. Refer to Lattice Propel 2025.1 Builder User Guide (FPGA-UG-02235) for more details on how to create an SoC design using the **Empty Project** template.

3.2.3. Opening a 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 an 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), and 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 3.17).

   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.

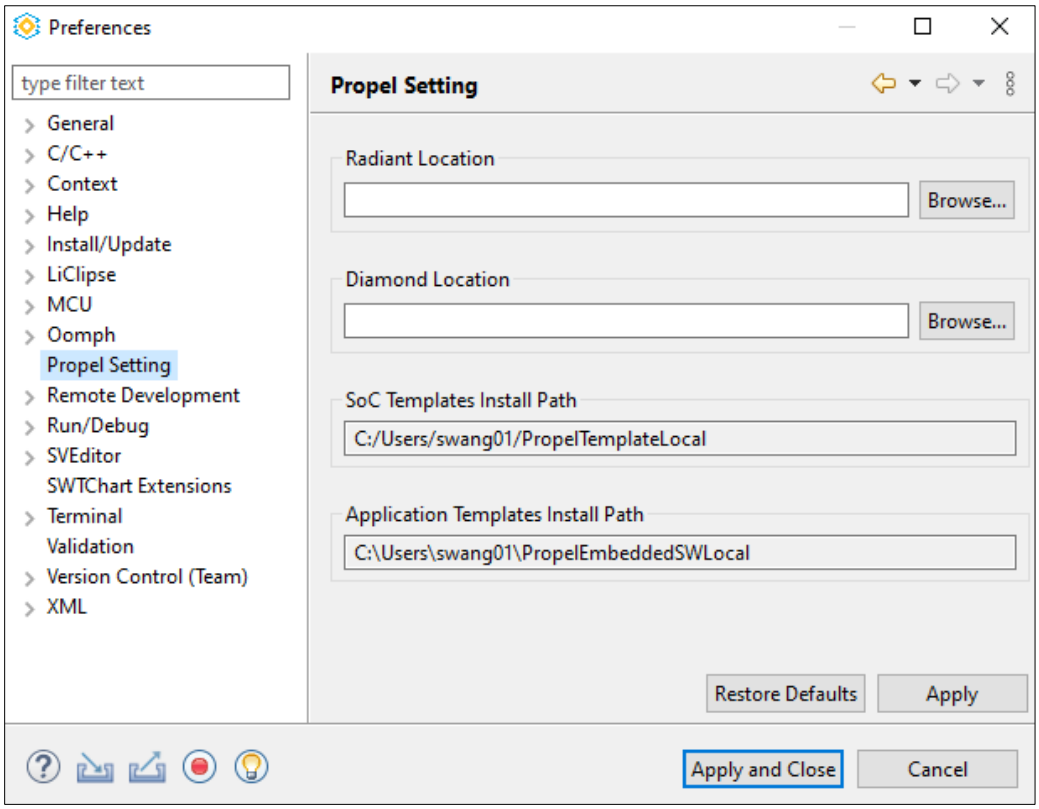


Figure 3.17. Lattice Propel Preferences Dialog

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:

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- 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 or Radiant project for SoC is generated at background and is launched (Figure 3.18/ Figure 3.19).

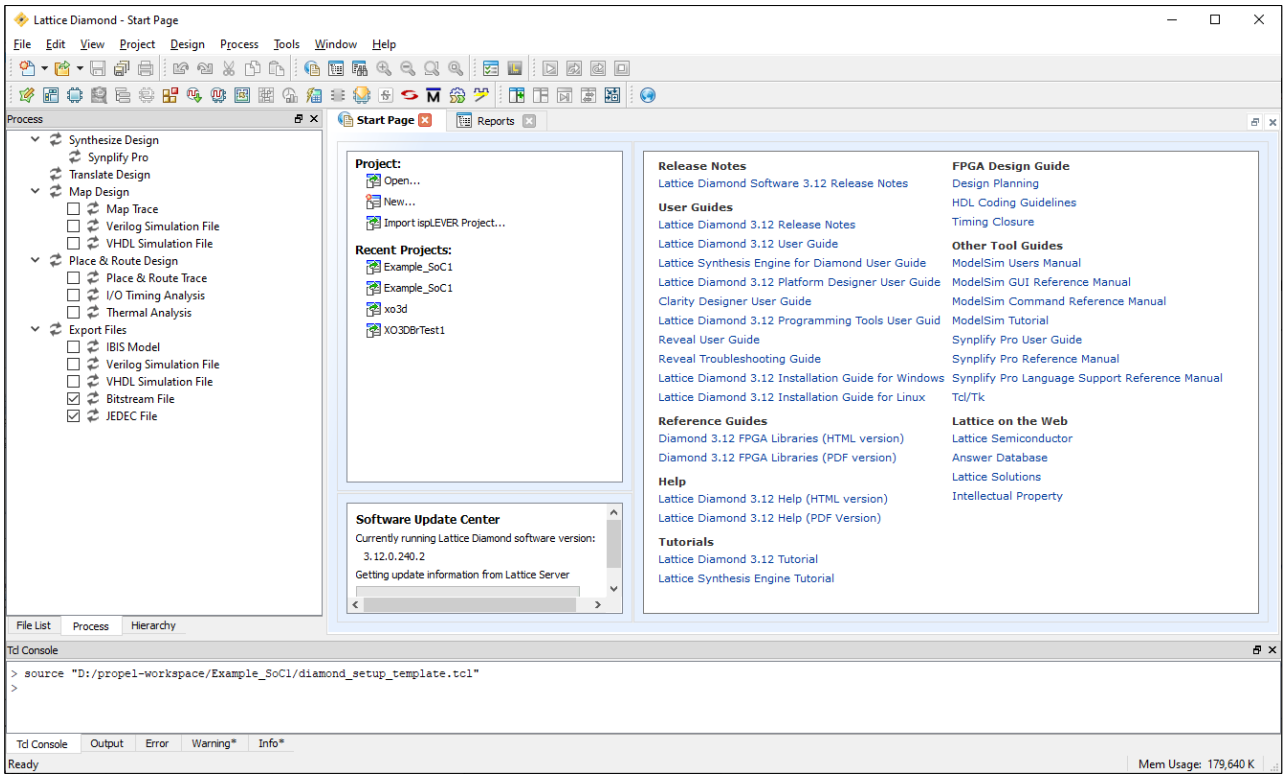


Figure 3.18. Lattice Diamond Software Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

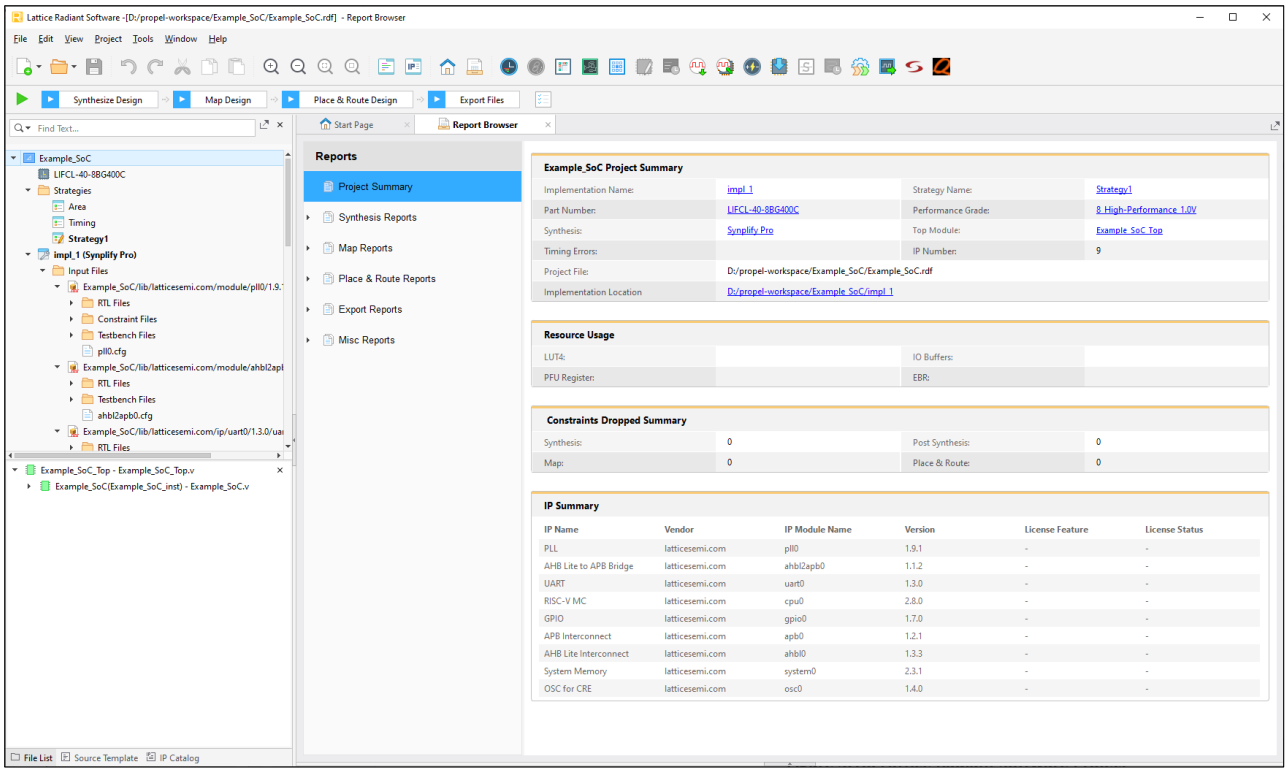


Figure 3.19. Lattice Radiant Software Project

5. (Optional) From the **File List** view of the Lattice Diamond or 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 or Radiant software.

   In the Lattice Diamond software, switch to the **Process** view of the project (Figure 3.20). 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**.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

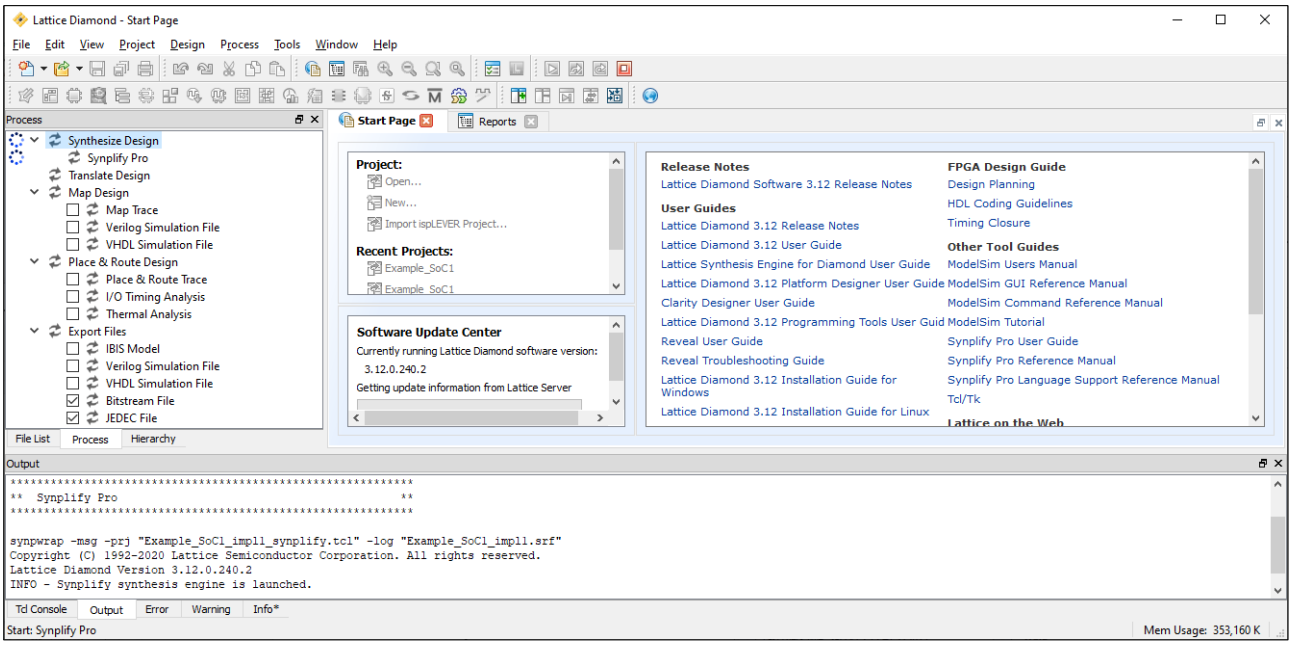


Figure 3.20. Generate Programming File in Lattice Diamond Software

In Lattice Radiant software, from the **Process** Toolbar, click **Export Files** (Figure 3.21).

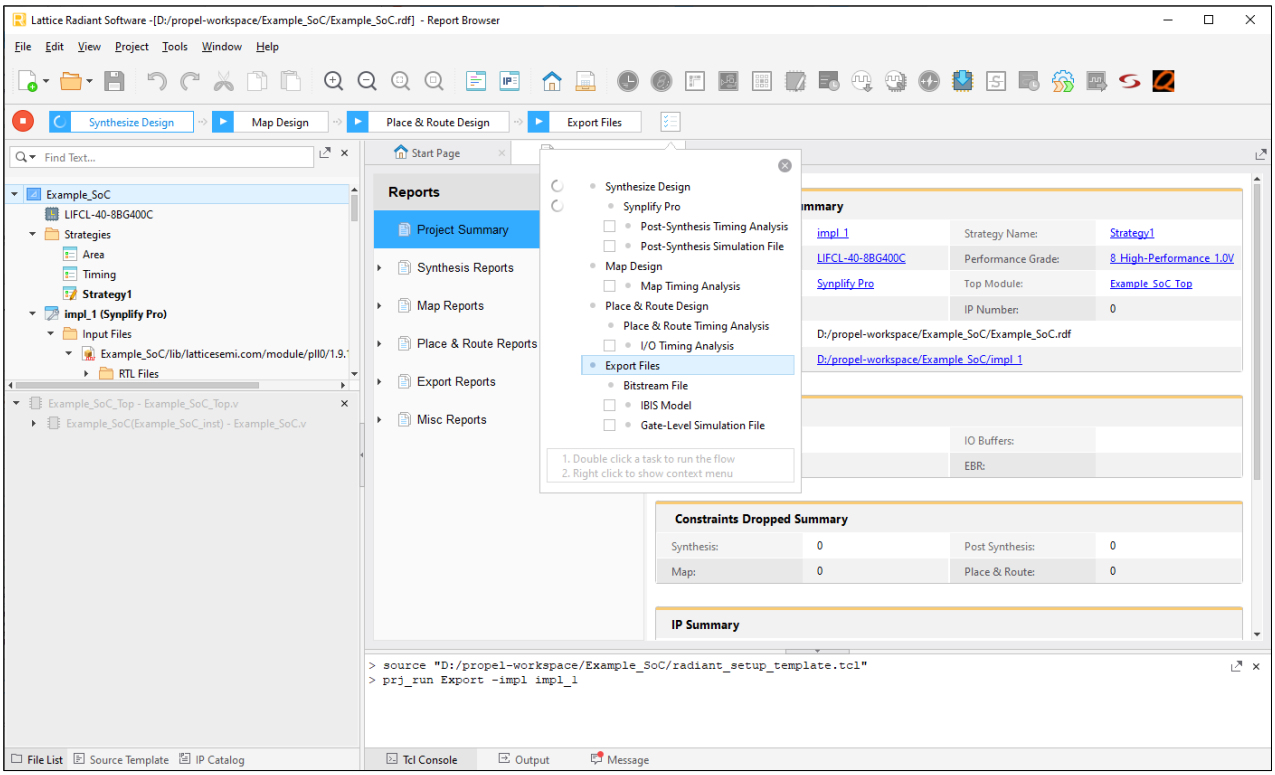


Figure 3.21. Generate Programming File in Lattice Radiant Software

The Programming file is generated. The 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 I2C.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 3.22).

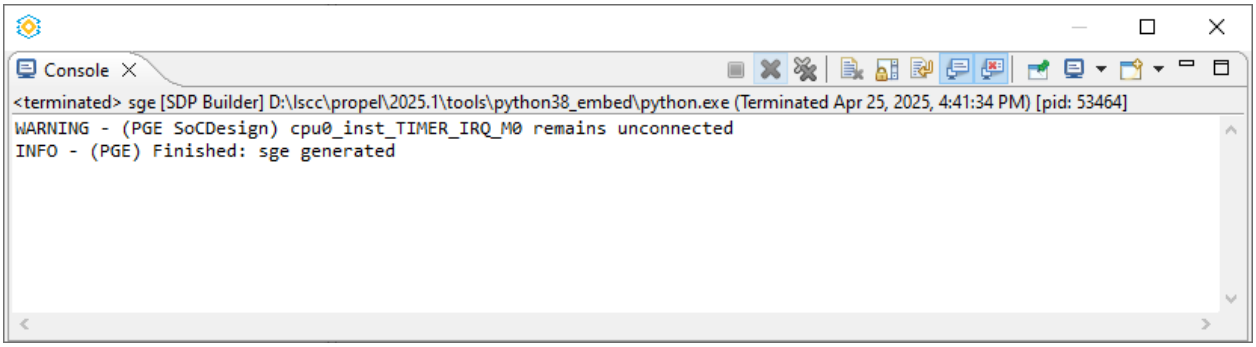


Figure 3.22. Build Result of SoC Project

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 3.23), 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 or Radiant project.
- sge: folder containing generated package necessary for creating a 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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

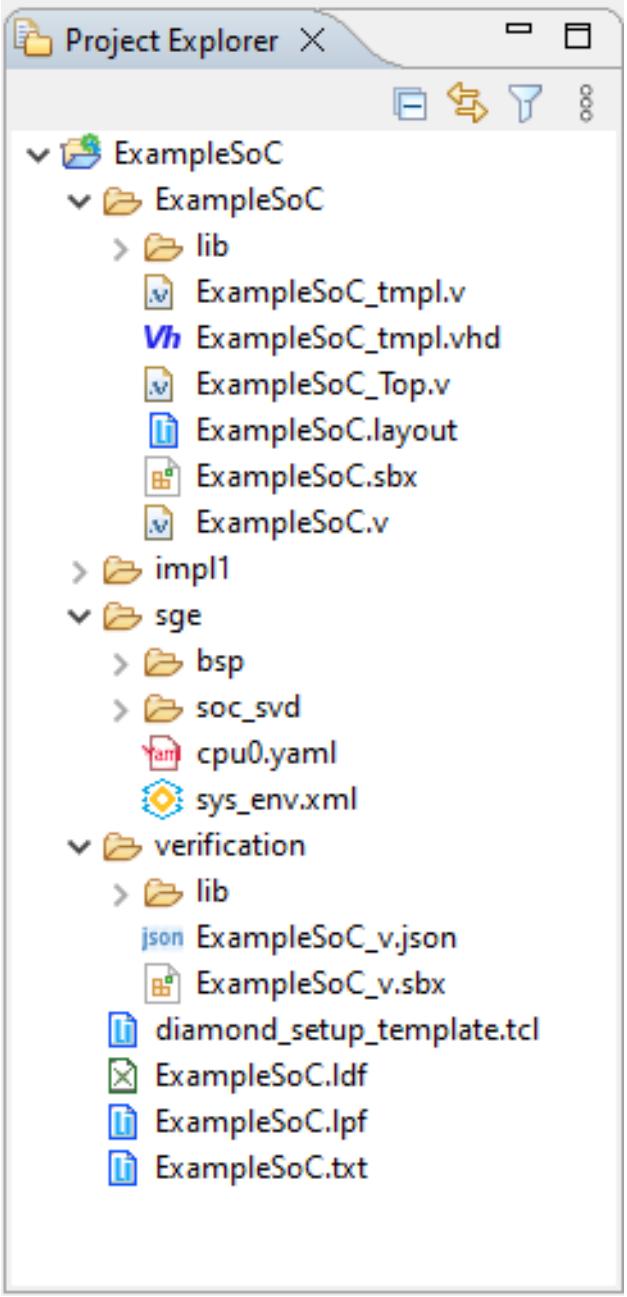


Figure 3.23. Contents of SoC Project

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 3.24).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

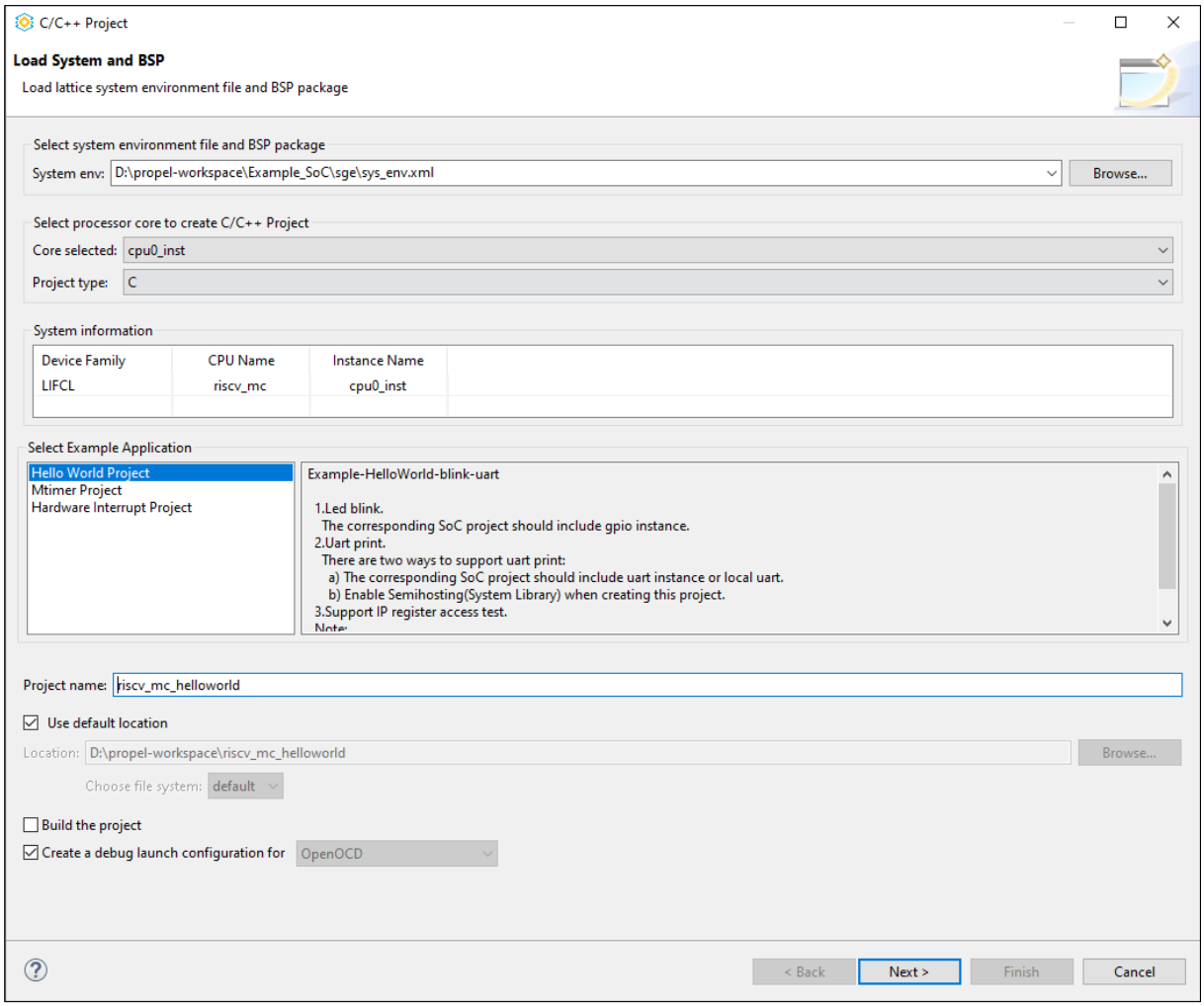


Figure 3.24. Load System and BSP Page 1

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 the QEMU application template to create a project (Figure 3.25).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

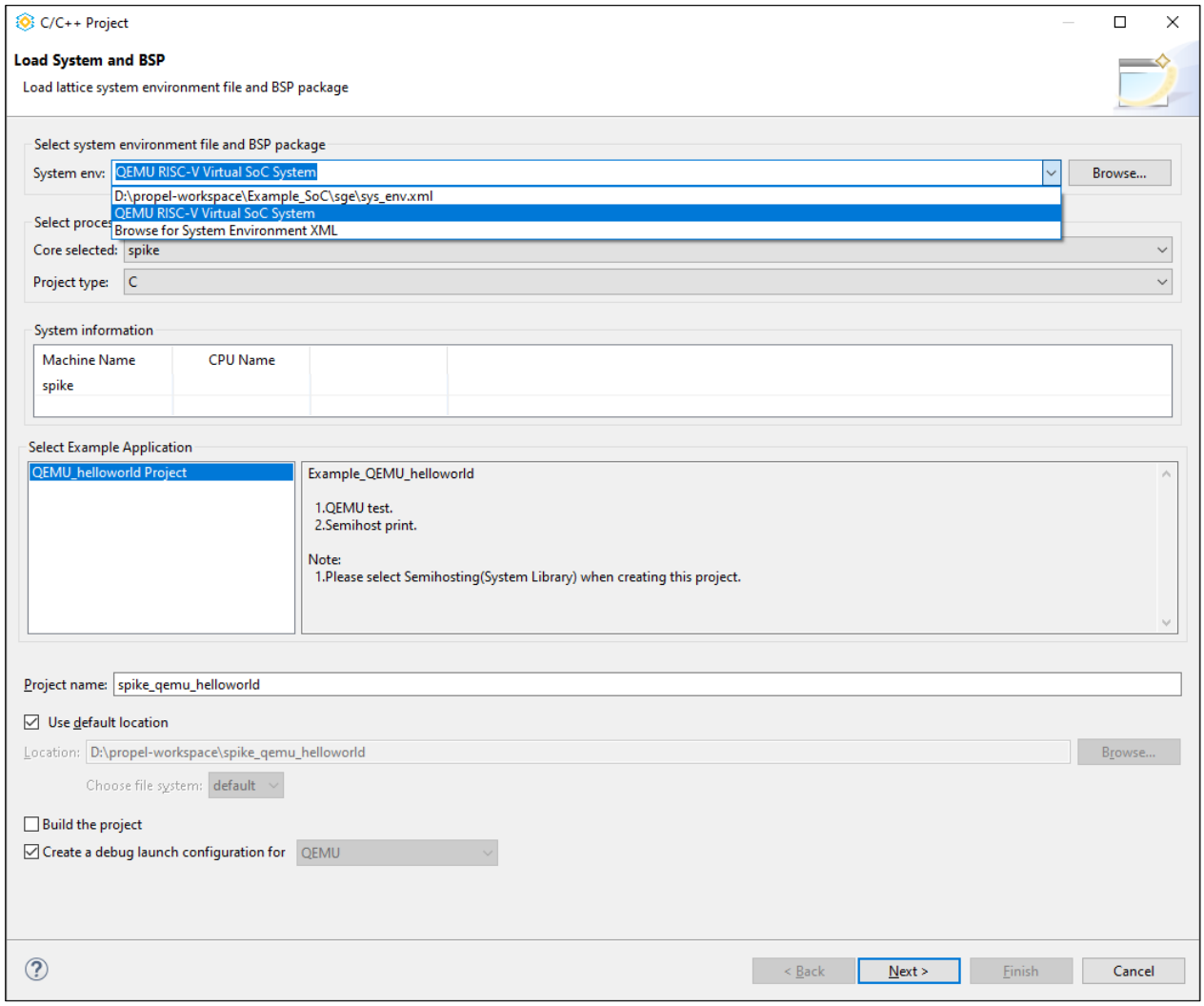


Figure 3.25. Load System and BSP Page 2

3. If the platform has more than one processor, choose one core.
4. Select the project type, C or C++.
5. Select the application from 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 3.26).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

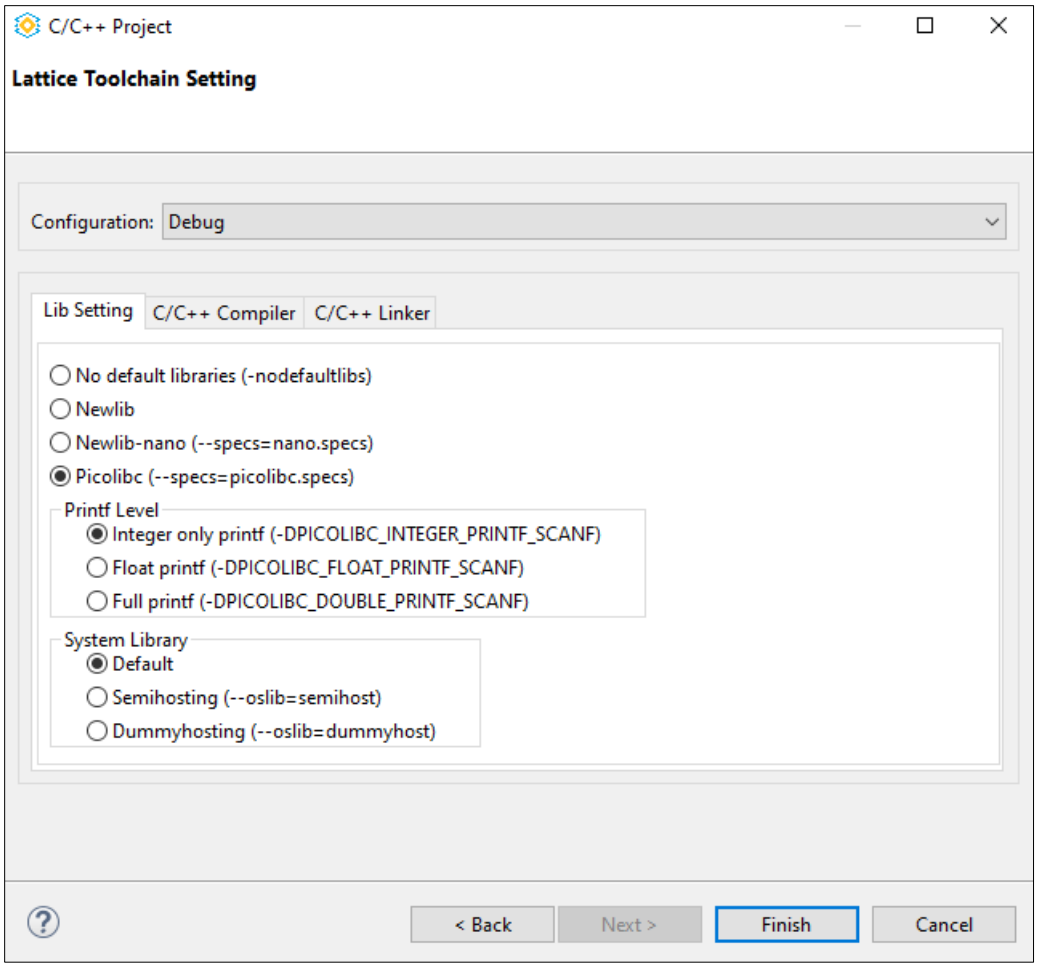


Figure 3.26. Lattice Toolchain Setting Dialog 1

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 the **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 the **C/C++ Compiler** tab, optimization level and debug level can be reconfigured for each toolchain configuration.
    - In the **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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 3.27).

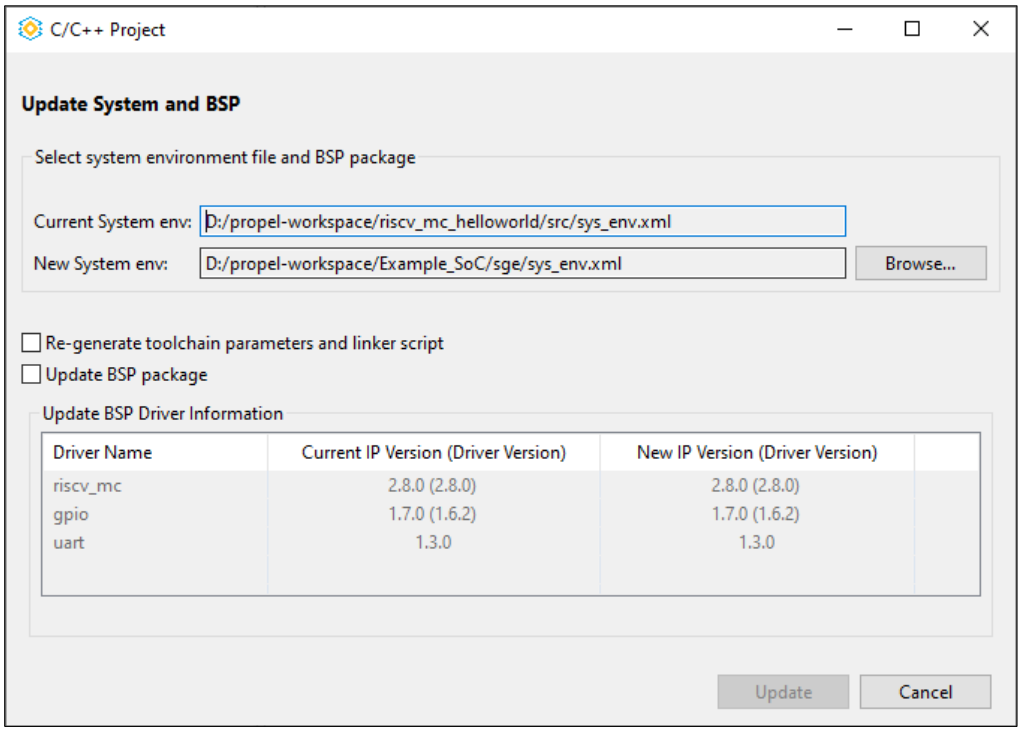


Figure 3.27. Update System and BSP Dialog

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 3.28).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

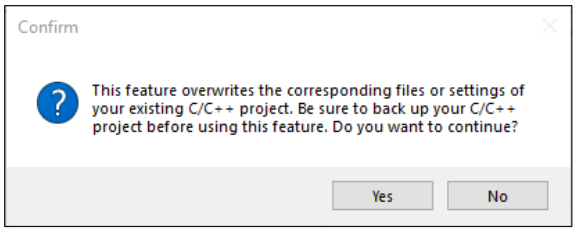


Figure 3.28. Update System and BSP Confirm Dialog

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 3.29) 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.

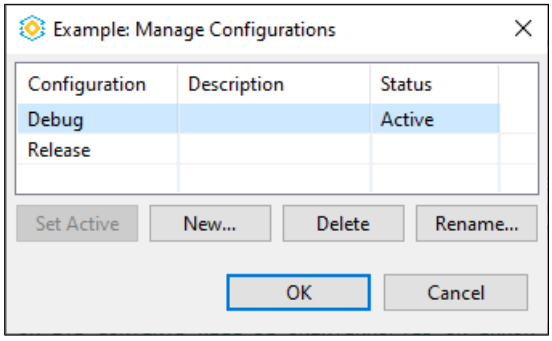


Figure 3.29. Manage Configurations Dialog

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 3.30).

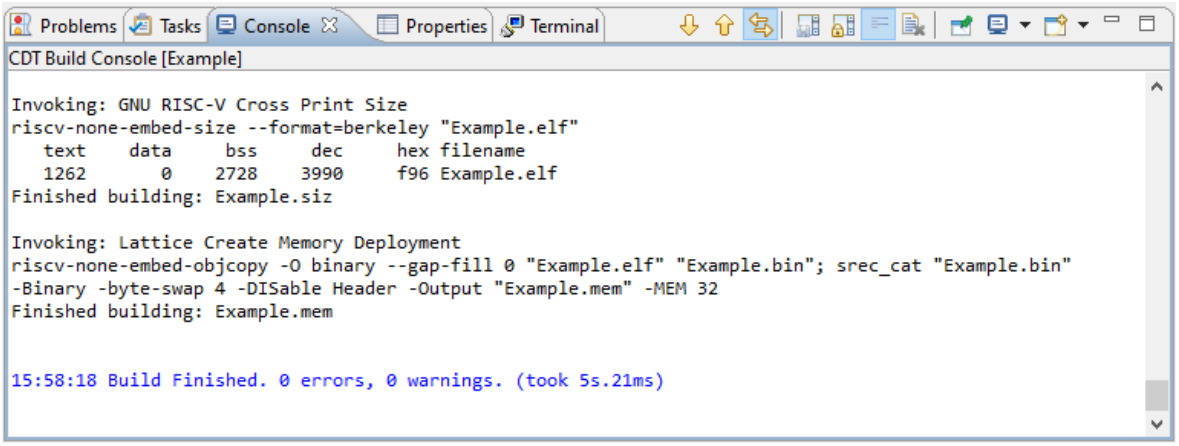


Figure 3.30. Build Result of C/C++ Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 when debugging.
- 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 3.31). 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 the System Memory IP.
- <*proj_name*>.launch: Debug launch configuration.

Note: Some of the files listed in Figure 3.31 are intermediate files that you do not need to take care of.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

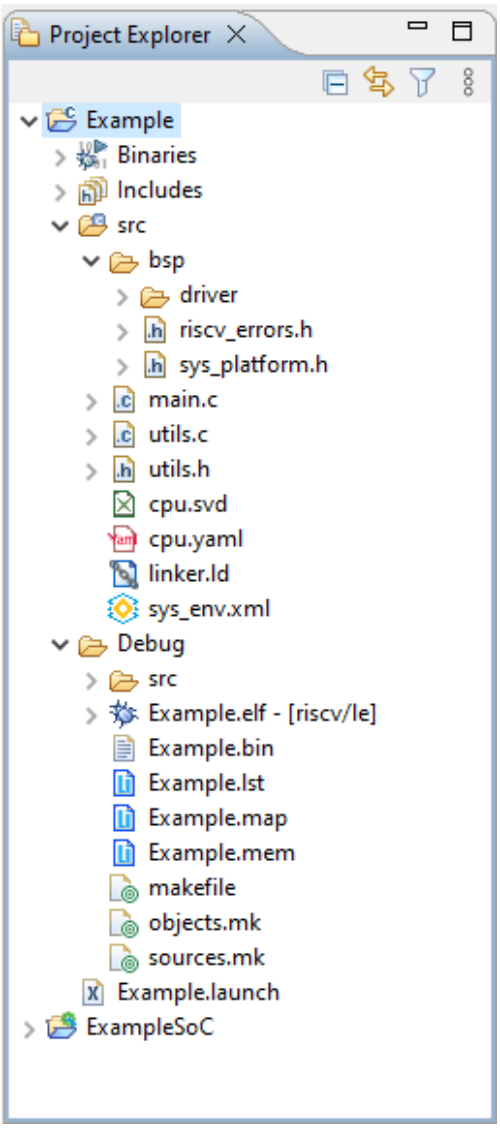


Figure 3.31. Contents of C/C++ Project

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 3.32).
- Linker Editor: An overview of the memory regions of linker script can be displayed. You can modify key linker parameters through the graphical interface (Figure 3.33).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

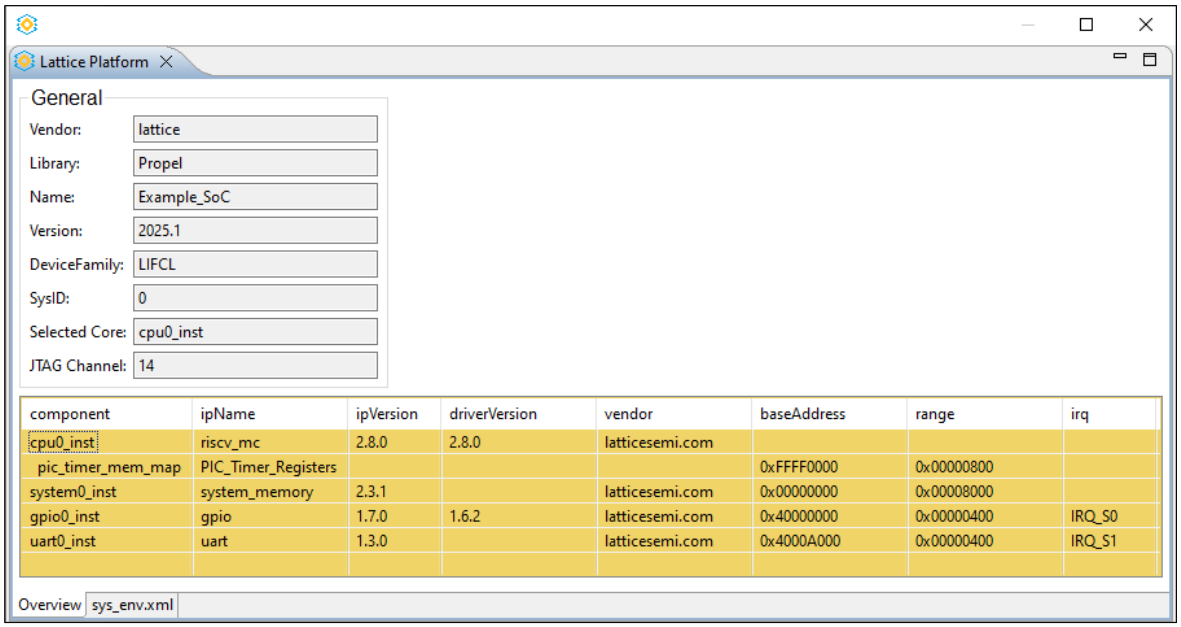


Figure 3.32. Lattice System Platform

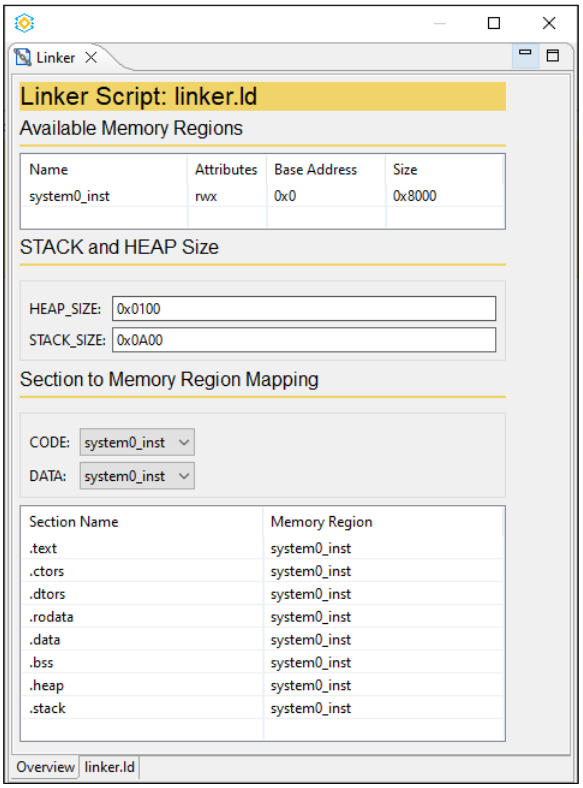


Figure 3.33. Linker Editor

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 3.34).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Select **Settings** of the **C/C++ Build** category from the left pane. Select the **Tool Settings** tab.

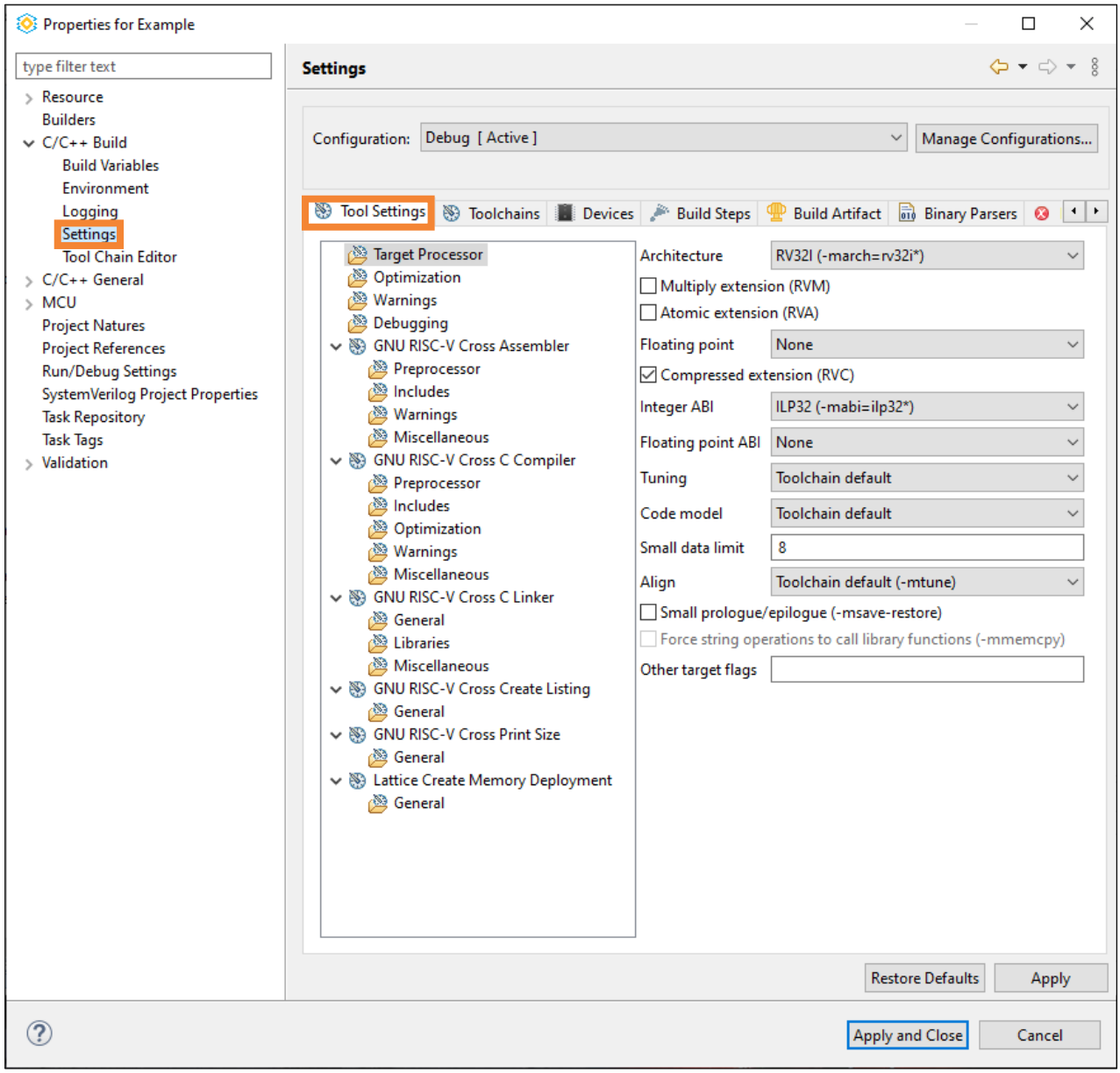


Figure 3.34. Properties of C/C++ Project

4. Customize the tools and tool options. All your customization can be made 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 set up 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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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. Launching Simulation

To launch simulation:

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

   Enable the checkbox for **Initialize Memory** for the System Memory module from the **Initialization** area of the **General** tab. Then, set the **Initialization File** generated from the corresponding C/C++ project (Figure 3.35).

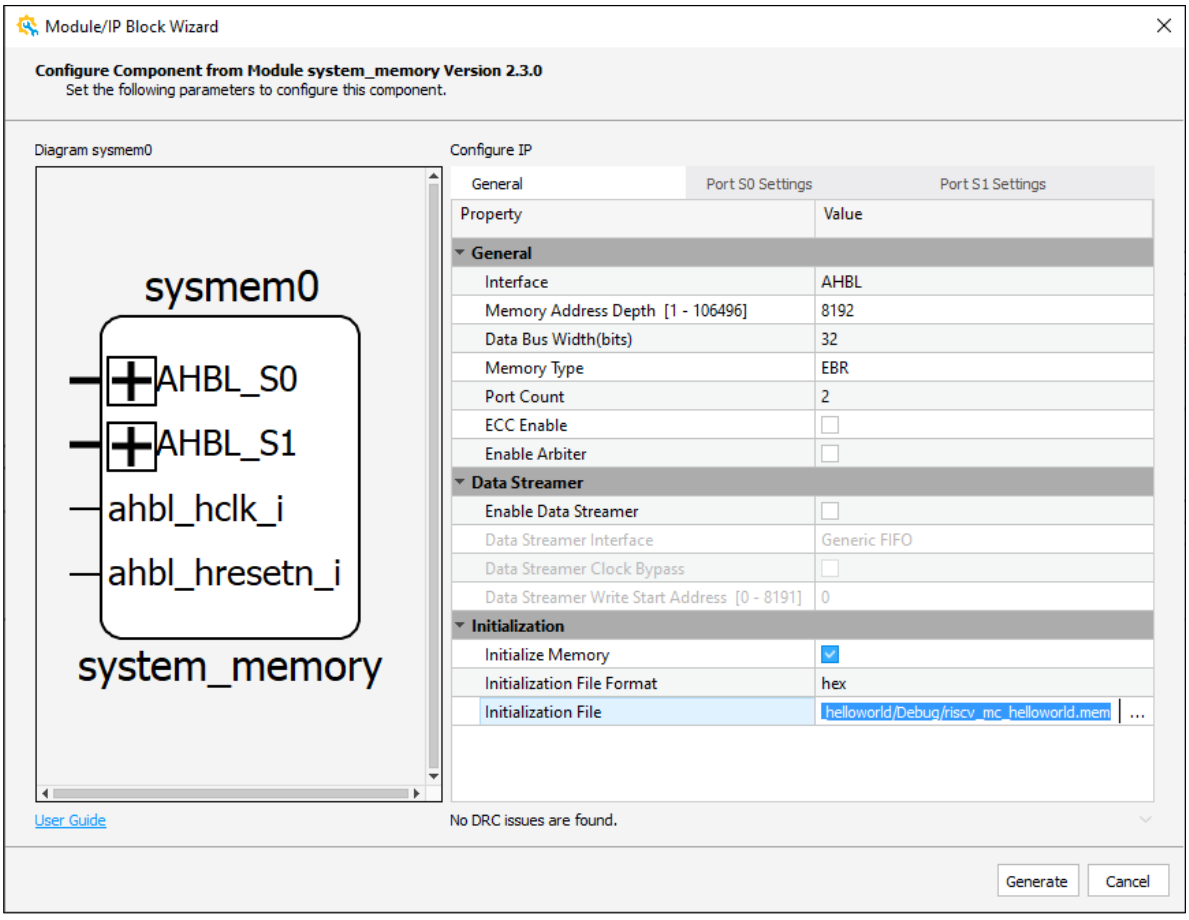


Figure 3.35. Configuring System Memory Module 1

2. Click the **Switch** icon on the toolbar to switch between SoC design and SoC verification project (Figure 3.36).
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 3.36).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

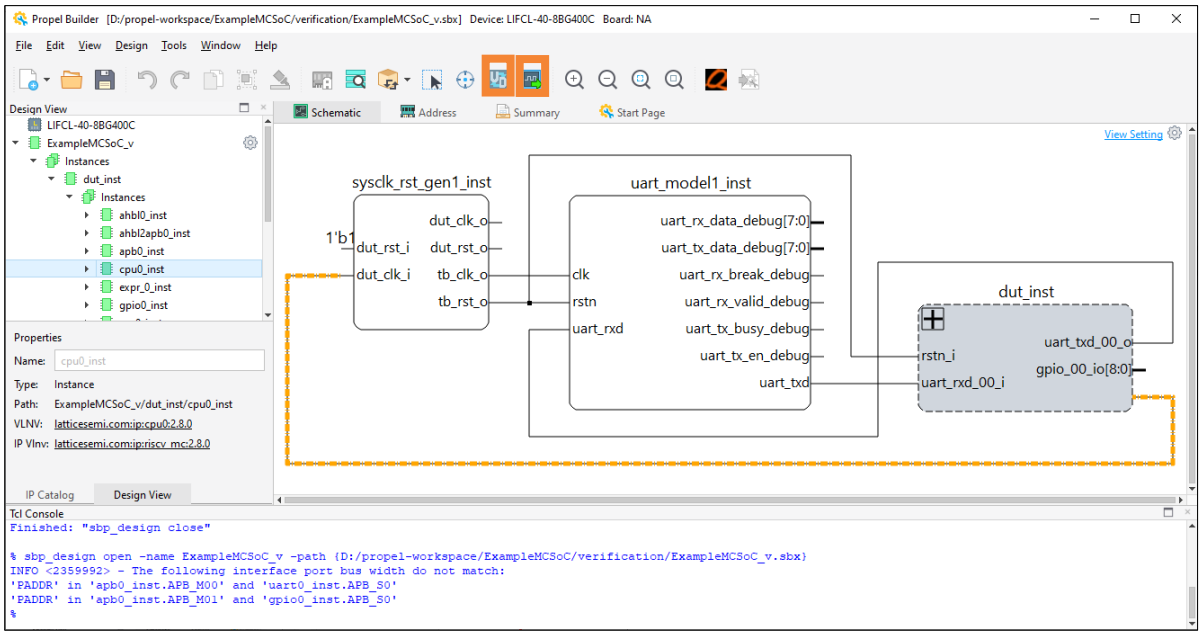


Figure 3.36. SoC Verification Project

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 3.37). Check the waveform.

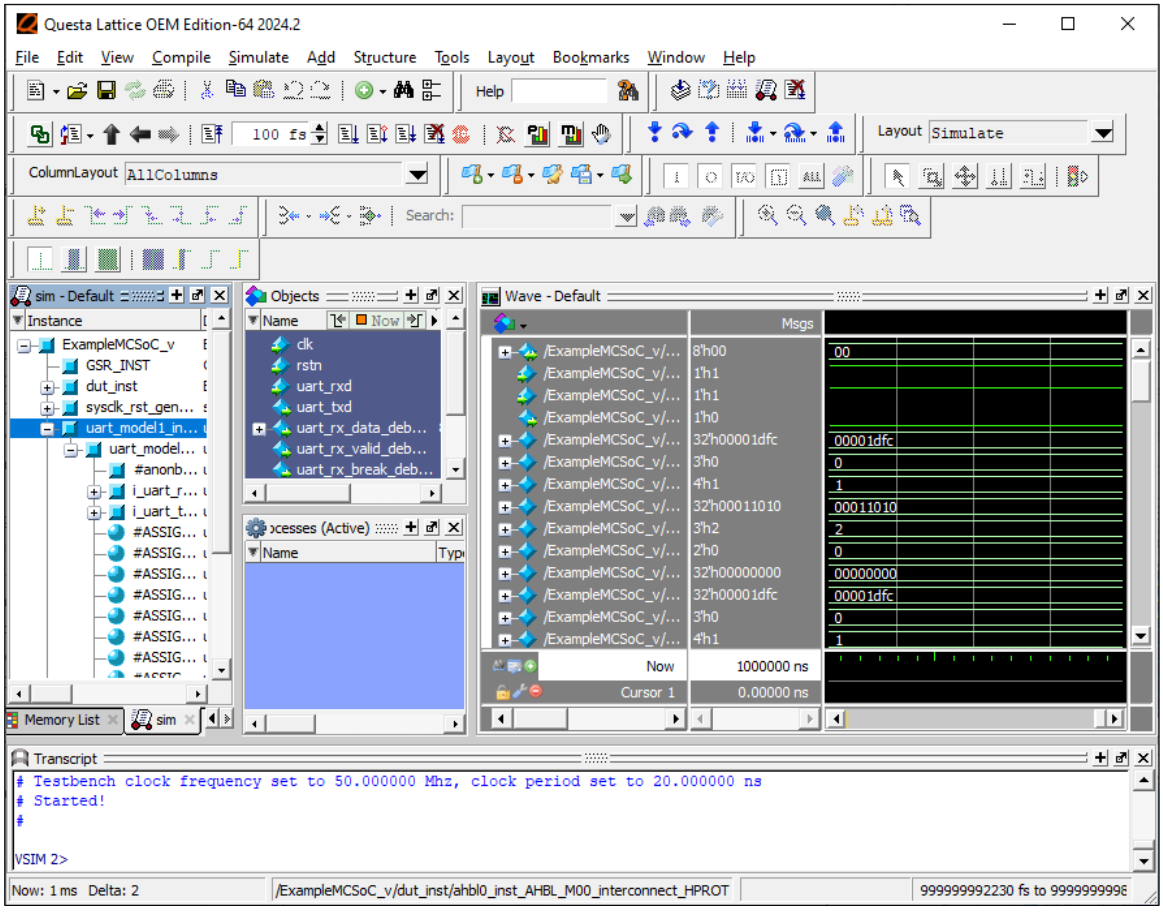


Figure 3.37. Questa Simulation GUI

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 is 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 or 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 3.39).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

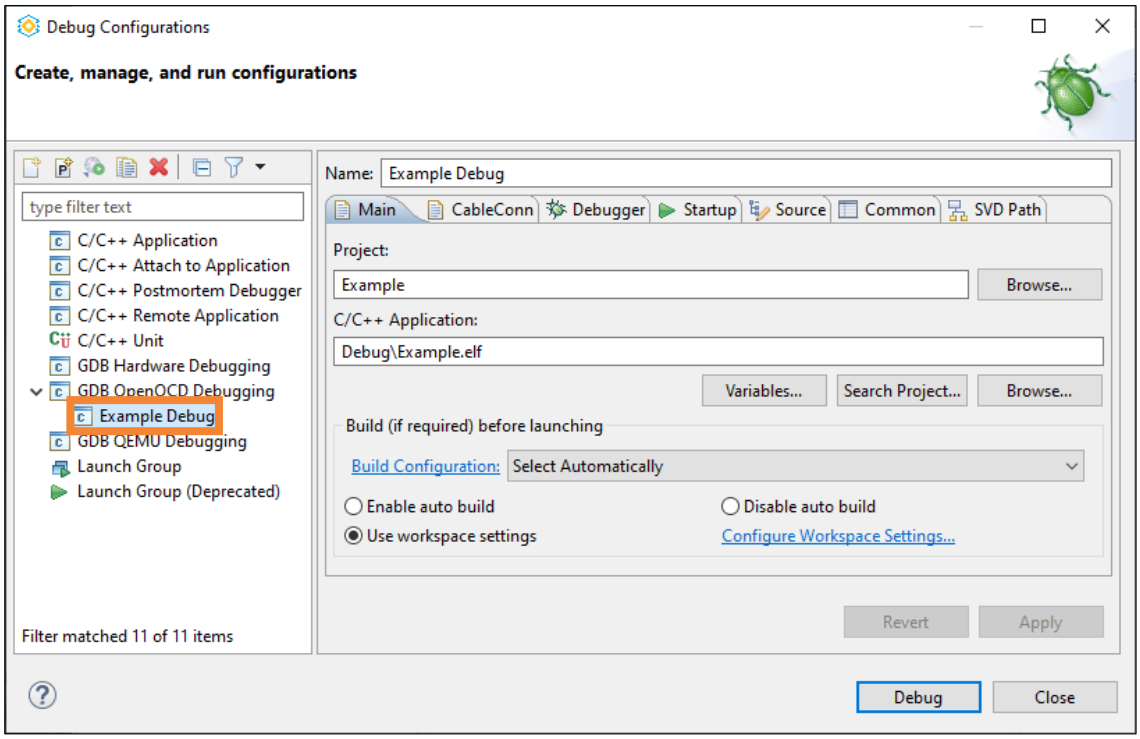


Figure 3.38. Debug Configurations Dialog 1

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 3.39). 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 the **Device** drop-down list. By default, the first available device on the selected cable port is selected.

   Select the JTAG channel number from the **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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

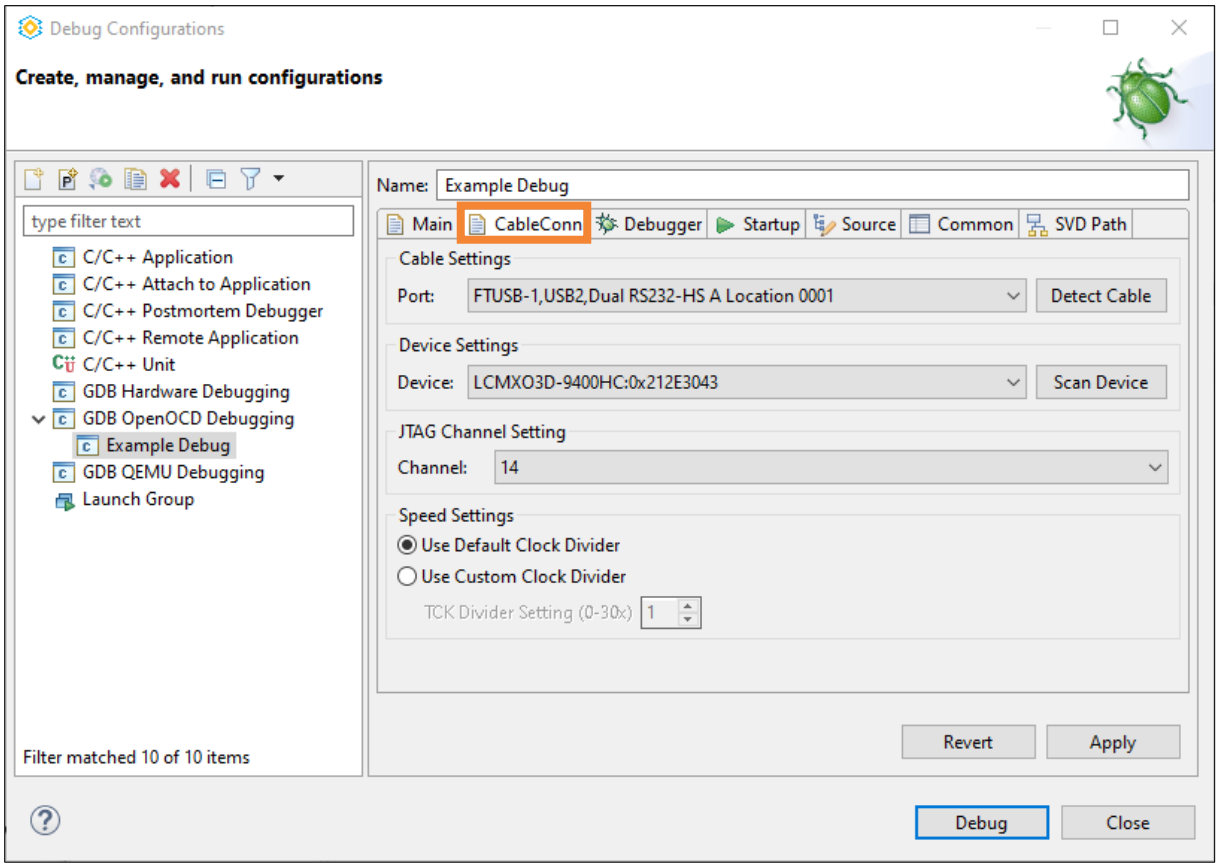


Figure 3.39. CableConn Tab of Debug Configurations

6. Select the **Debugger** tab (Figure 3.40). It is critical that the **Config options** field contains the correct command line options to be passed to OpenOCD.

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

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

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

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

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

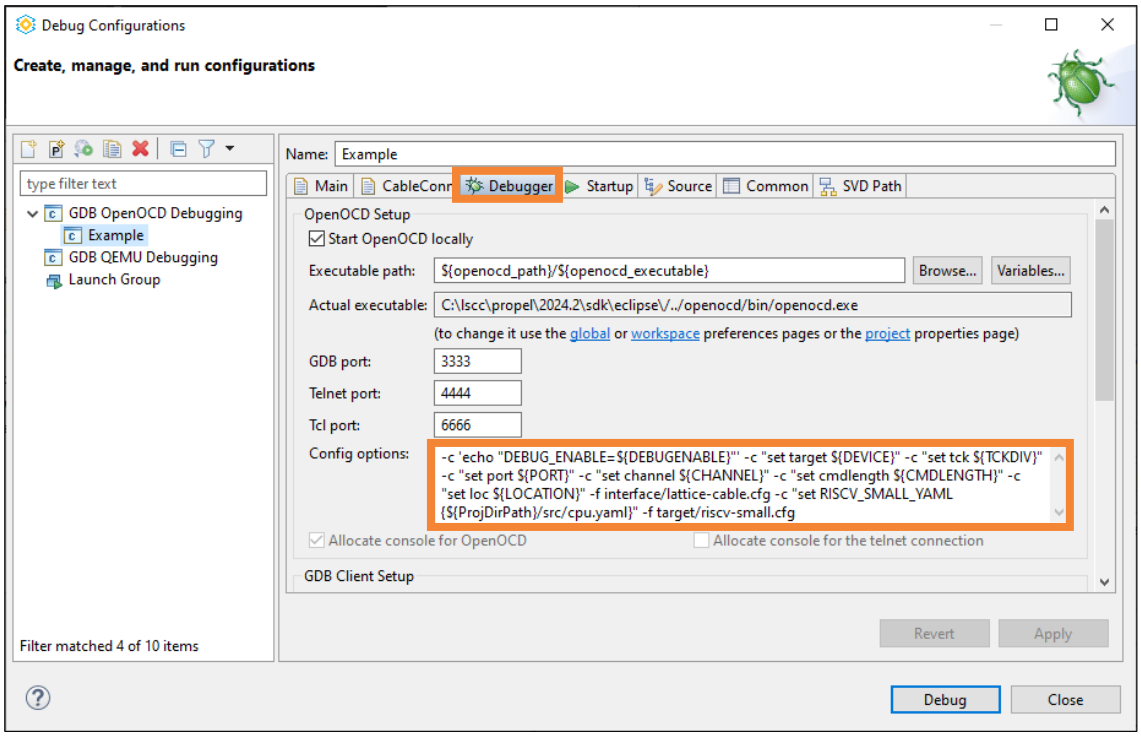


Figure 3.40. Debugger Tab of Debug Configurations

7. (Optional) Select the **Common** tab (Figure 3.41). 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.

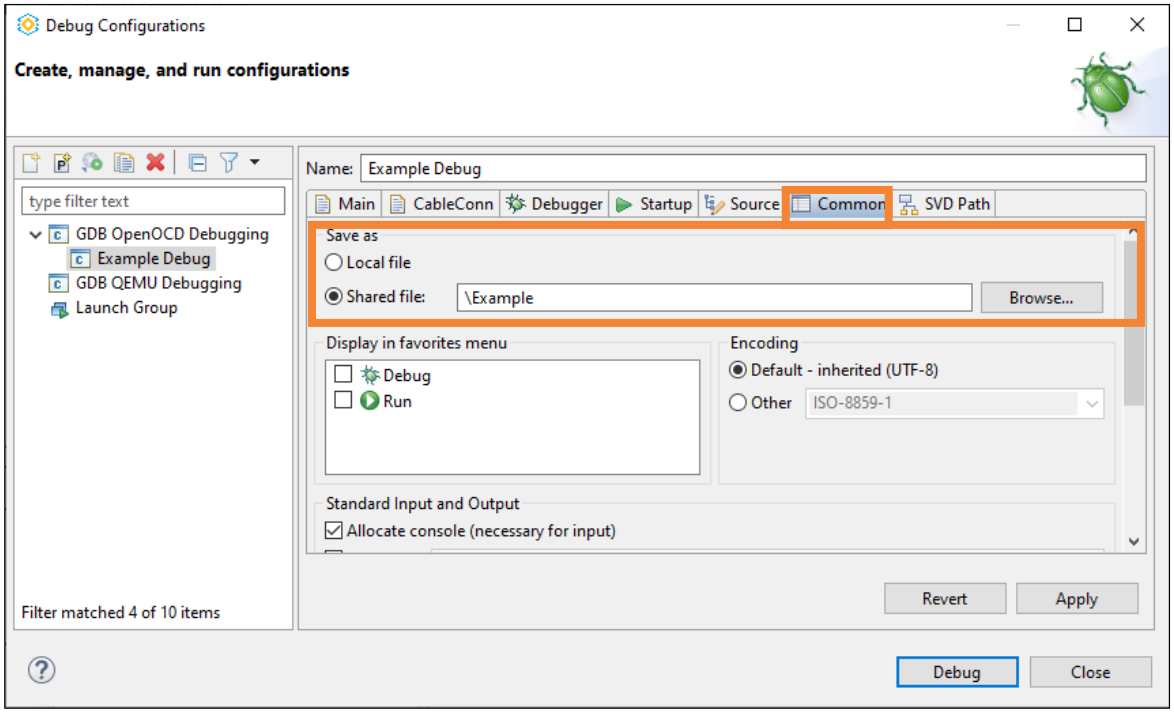


Figure 3.41. Common Tab of Debug Configurations

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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**, and **GDB Hardware Debugging** are hidden on the Debug Configurations page. If you want to use them, you can disable **Filter checked launch configuration types** (Figure 3.42).

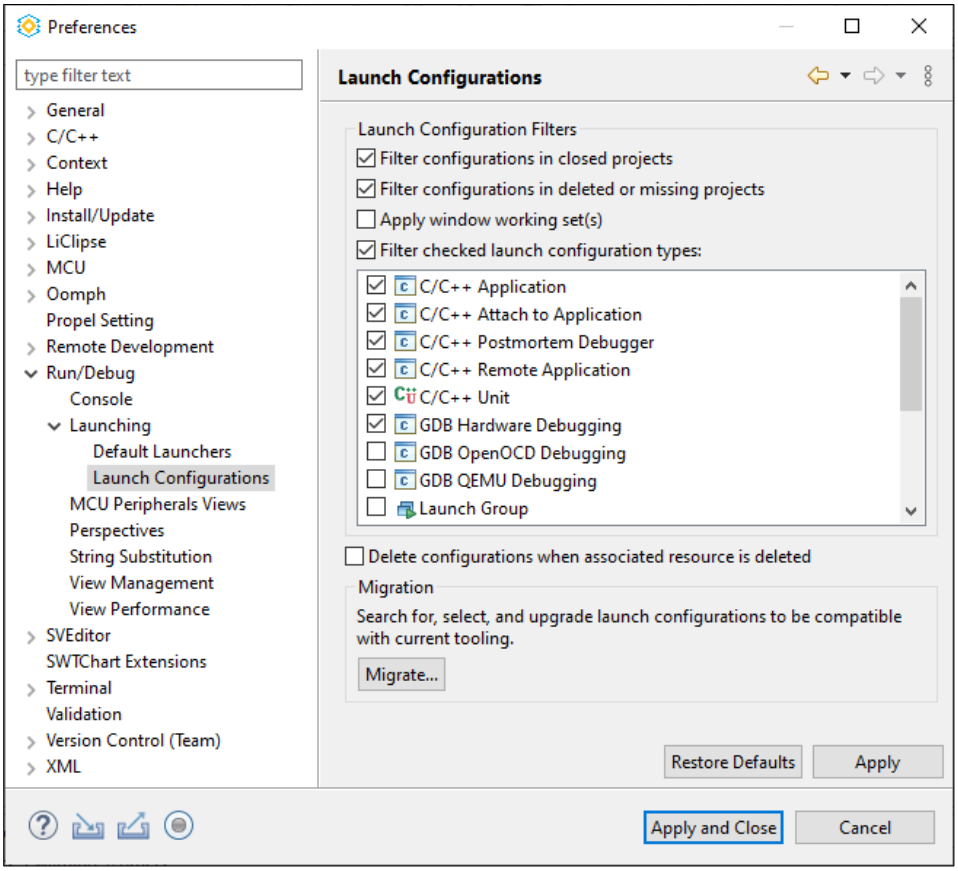


Figure 3.42. Launch Configurations

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 powered ON.
- The hardware design has a debug enabled processor module and already programmed into the target device.

With the above steps completed properly, follow 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 3.41).

   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 3.43).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

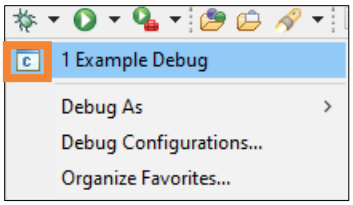


Figure 3.43. Debug Icon on Toolbar

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 3.44. The execution stops right at the beginning of the main() function.

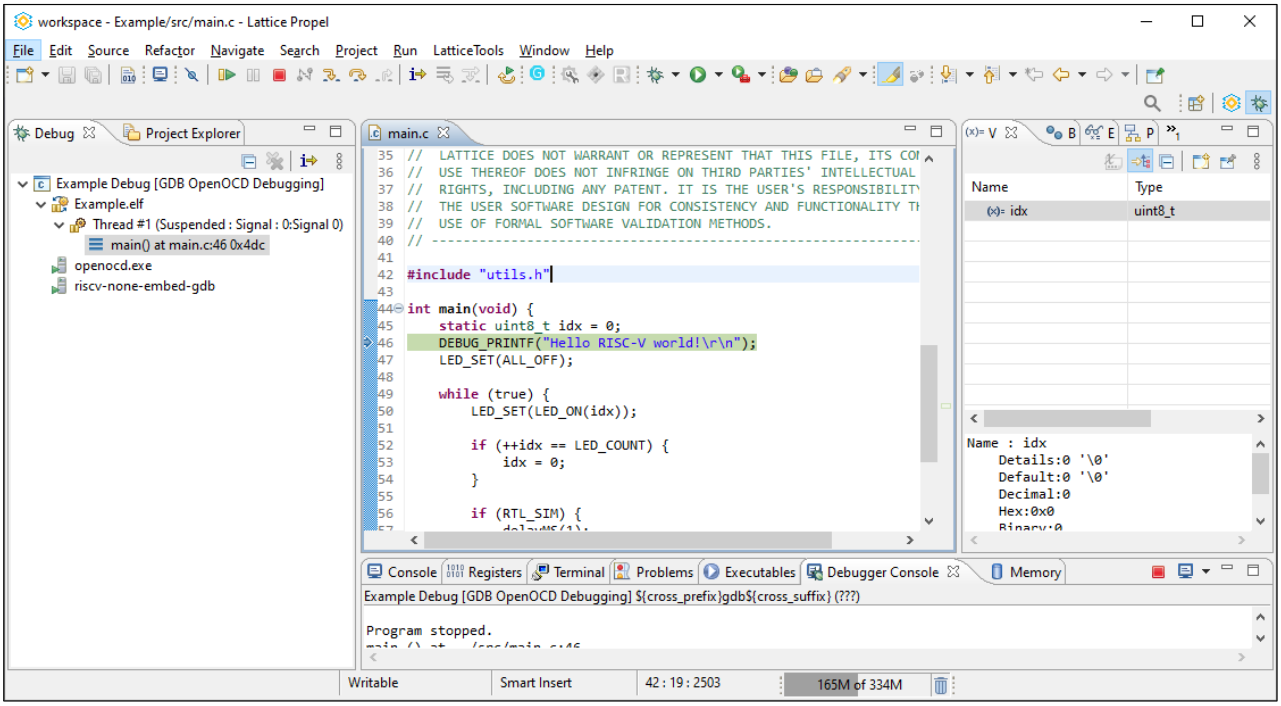


Figure 3.44. Debug Perspective 1

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

3.5.3. Peripherals Registers View

The 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 3.45):

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

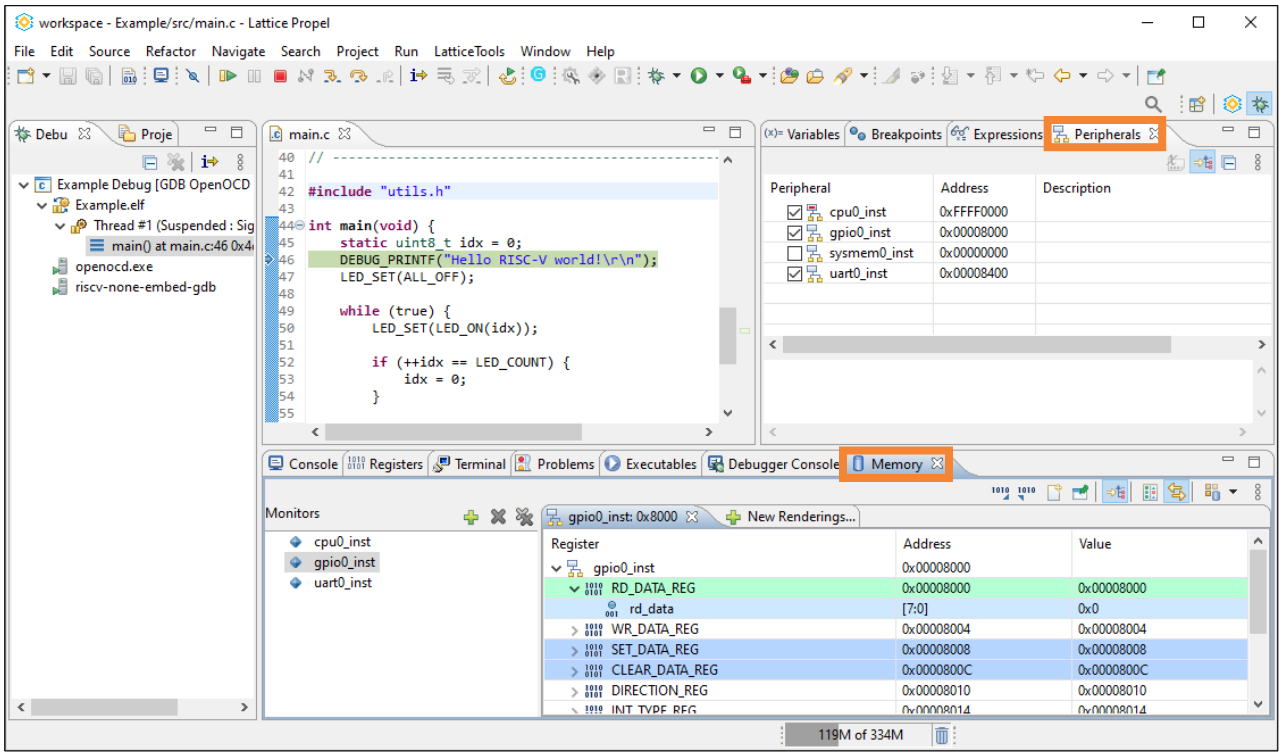


Figure 3.45. Peripherals View in Debug Perspective

3.5.4. Serial Terminal Tool – Windows

Serial port communication is frequently used during 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 3.46).
3. Choose **Serial Terminal** and configure the **Serial port** with **Baud rate**.

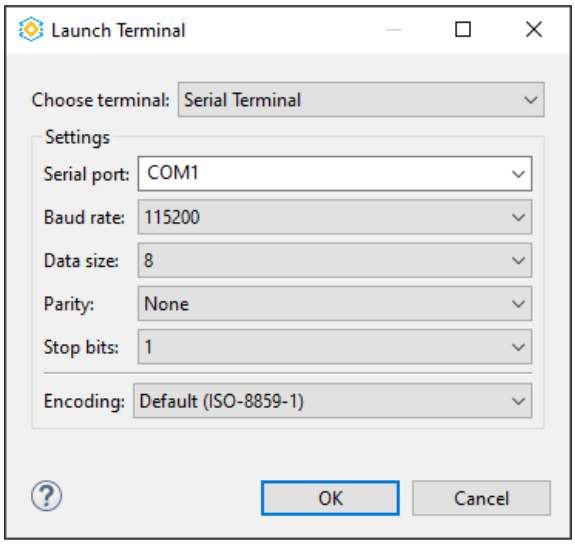


Figure 3.46. Launch Terminal Dialog 1

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Click **OK**. A connection opens.
5. (Optional) Click the **Toggle Command Input** icon that adds an edit box to enter text (Figure 3.47).

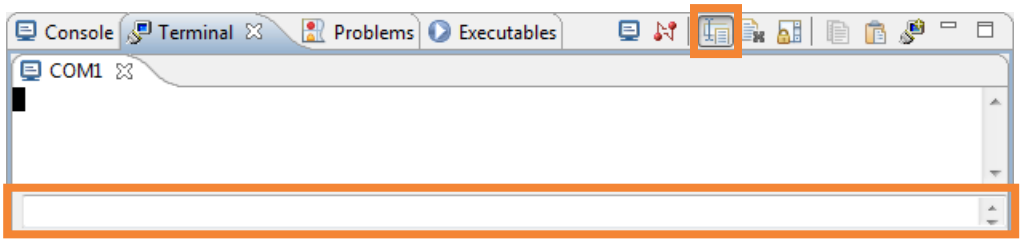


Figure 3.47. Terminal View

3.5.5. Serial Terminal Tool – Linux

In Linux, the VCP driver and D2XX driver are incompatible with each other. For more details, refer to FTDI Drivers Installation Guide for Linux.
Lattice Propel SDK 2025.1 provides a Linux terminal tool to fix this compatibility limitation, based on PyFtdi.

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 3.48).
3. Choose **Local Terminal** and the default encoding **UTF-8**.

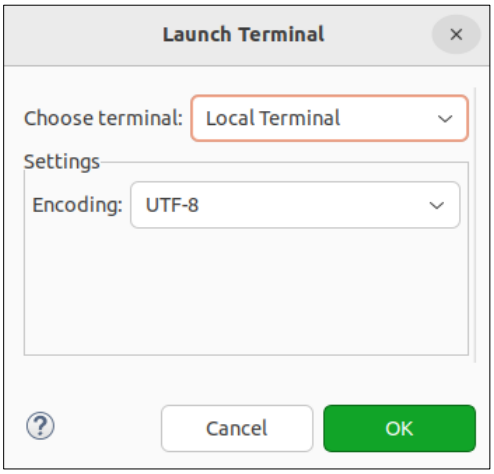


Figure 3.48. Launch Terminal Dialog 2

4. Click **OK**. A Linux shell command window opens.

   Input the following command.

```
$cd <Propel SDK install Location>
$./terminal_cli
```

   Then input the device index, as suggested in the terminal (Figure 3.49).

   Execute on-chip debug with the UART output (Figure 3.50).

   Note: Make sure to select the correct FTDI device shown in Figure 3.49. Otherwise, on-chip debug failure can occur and you need to re-launch Lattice Propel SDK.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

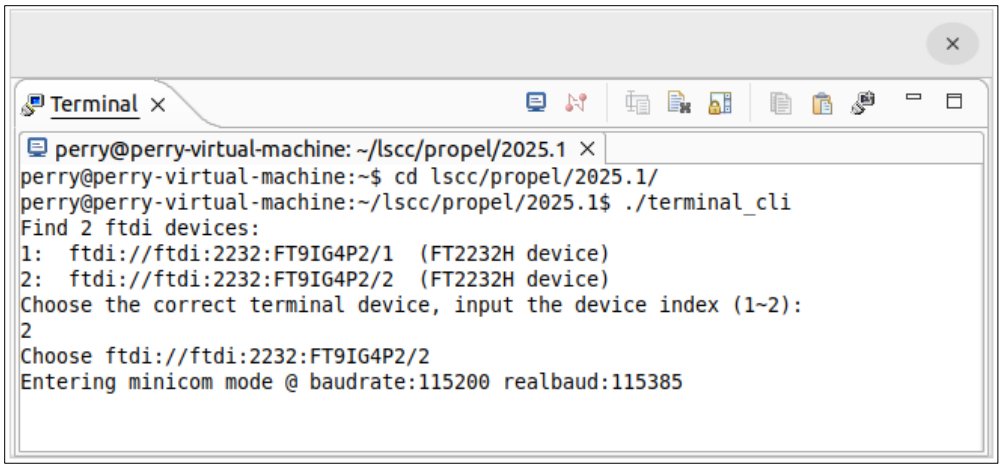


Figure 3.49. Terminal cli

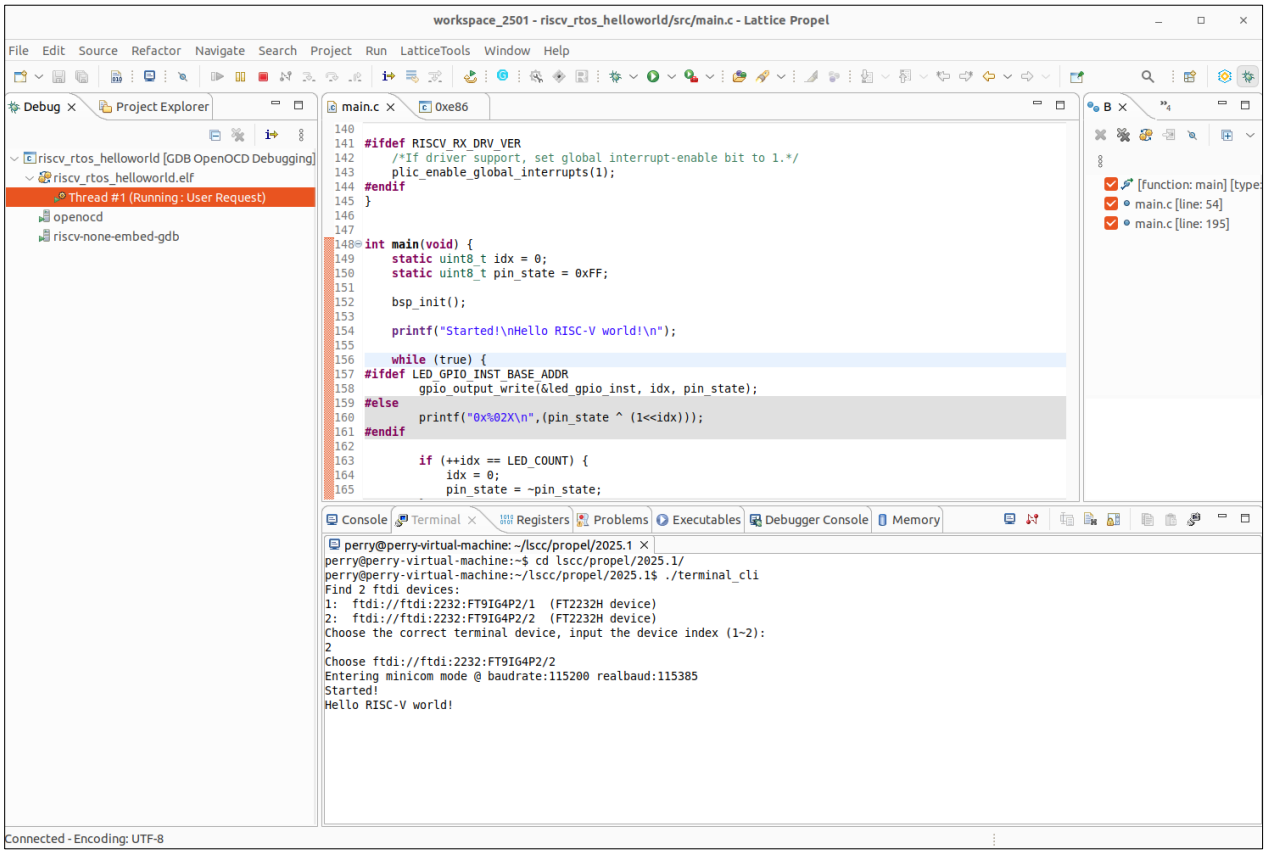


Figure 3.50. On-Chip Debug with UART Output

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. How to Start with Lattice FPGA Board

This chapter is a tutorial on how to run a program on a Lattice FPGA board.

This tutorial uses CertusPro-NX Evaluation Board for example.

All Lattice Propel related products can be found from Lattice Propel Design Environment.

4.1. Board Introduction

The CertusPro-NX Evaluation Board features the CertusPro-NX FPGA in the LFG672 package, which is built on Lattice Nexus™ FPGA platform using low power 28 nm FD-SOI technology. The board expands the usability of the CertusPro-NX FPGA with FMC HPC connector, PMOD, Raspberry PI, along with access to 8× SerDes channels.

Easy-to-use board resources of the jumper, LED indicator, push button and switch are available for user-defined applications. Refer to CertusPro-NX Evaluation Board User Guide (FPGA-EB-02046) for more details of this board. Figure 4.1 shows the top view of the CertusPro-NX Evaluation Board.

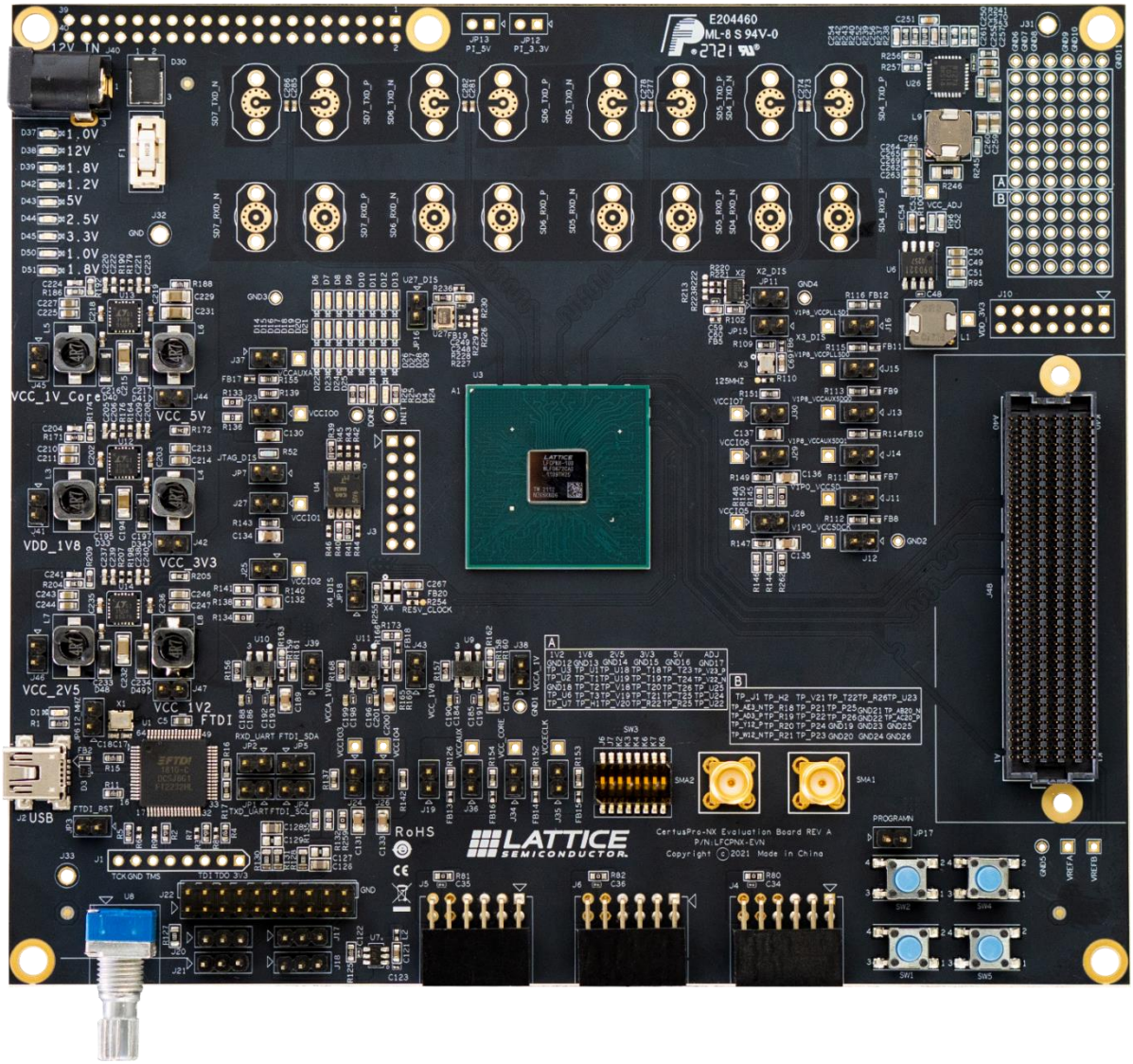


Figure 4.1. CertusPro-NX Evaluation Board

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.2. Creating an SoC Project

The first step is to Create an SoC project. The SoC project is the hardware environment for a program. For detailed flow for creating an SoC project, refer to Lattice Propel Builder 2025.1 User Guide (FPGA-UG-02235).

From Lattice Propel Builder, you can create scalable SoC projects and other special SoC projects (Figure 4.2).

Note: In this document, most of the guided flow is based on scalable SoC projects.

In the **Select Device** GUI, select the corresponding board (Figure 4.3). In this example, CertusPro-NX Evaluation Board is selected.

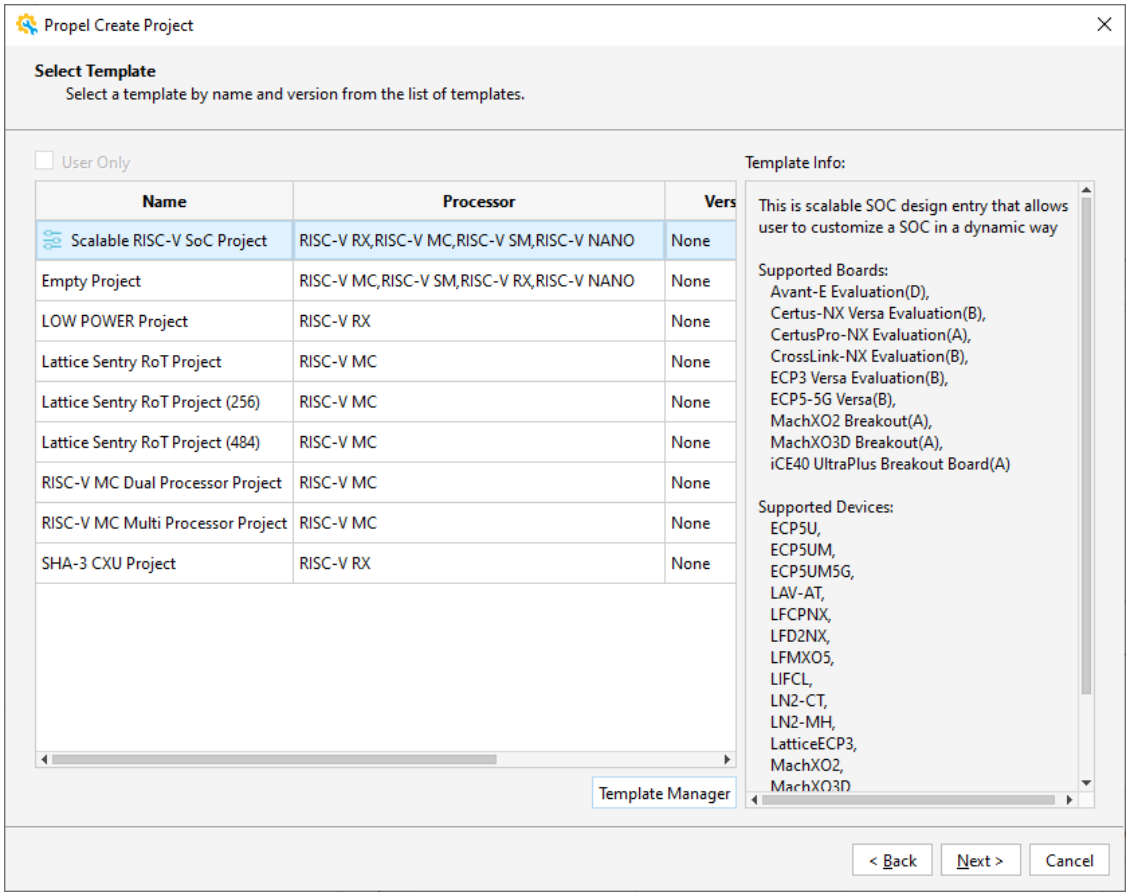


Figure 4.2. Select Template GUI

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

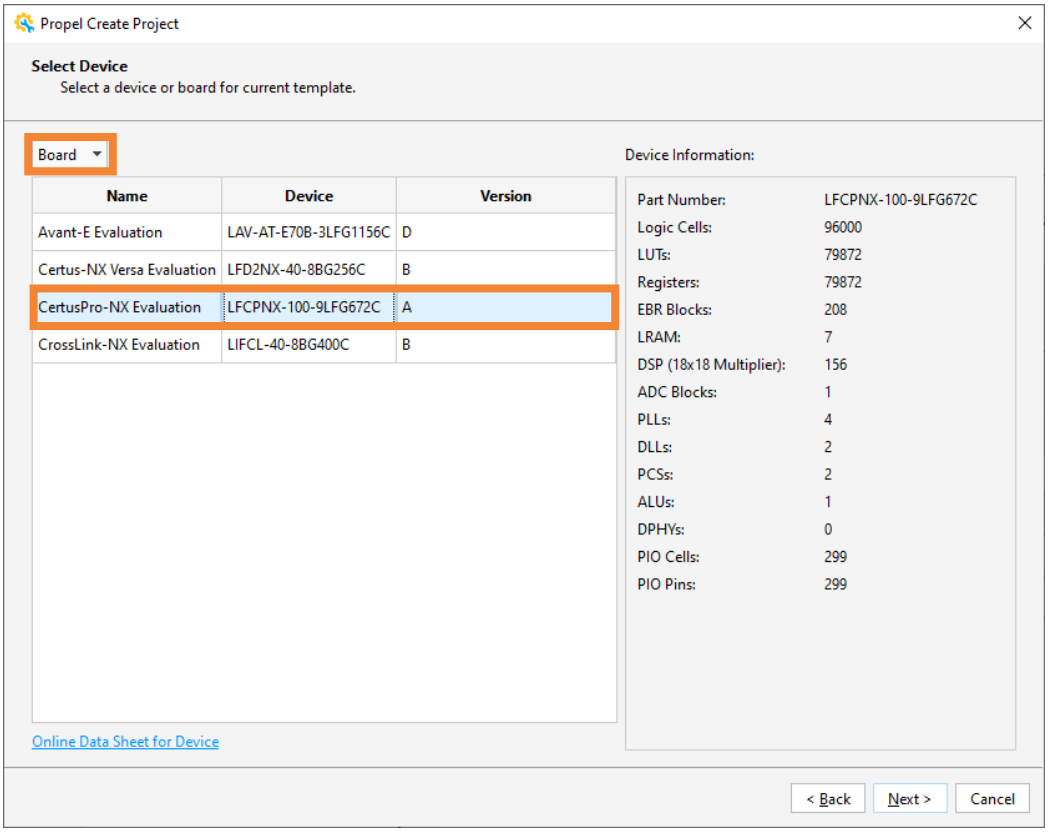


Figure 4.3. Select Device GUI

4.3. Creating a C/C++ Project

The second step is to create a C/C++ project.

Refer to the C/C++ Project Design Flow section for details of creating a C/C++ project.

Different C project templates need different types of SoC project.

4.4. Memory Initialization (Optional)

The third step is to initialize the memory file for the SoC project.

This step is optional. If you need the board to run after power-on, you need to execute this step.

Refer to the Memory Initialization to SoC Project section for memory initialization details.

4.5. Generating and Programing a Bit File

The fourth step is to generate a bit file and program it to board.

Save the SoC project created in steps above. Launch the Lattice Radiant software or Lattice Diamond software to generate the bit file, as shown in the Opening a Design in Lattice FPGA Design Software section.

When the bit file is generated correctly, program it to the board. Refer to the corresponding board user guide for details of programming the bit file. In this example, refer to CertusPro-NX Evaluation Board User Guide (FPGA-EB-02046).

4.6. On-Chip Debugging

The last step is on-chip debugging. You can refer to the Programming and On-Chip Debugging Flow section.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5. General Application Templates

5.1. Template List and Requirements

General application templates provide reference code for RISC-V processors.

These general templates depend on scalable RISC-V SoC projects.

For the project flow, refer to the How to Start with Lattice FPGA Board section.

Table 5.1 lists all the general templates and displays the requirements for every template.

Table 5.1. Template List

Template Name	Processor Requirement (Minimal Supported Version)	Required IP	Other Requirement	Default Soc Project
Hello World Project	RISC-V RX (V2.2.0), RISC-V MC (V2.4.0), RISC-V SM (V1.4.0), RISC-V Nano (V1.0.0)	UART, GPIO	—	Scalable RISC-V SoC Project (RISC-V RX, RISC-V MC, RISC-V SM, RISC-V Nano)
FreeRTOS-LTS Minimal Project	RISC-V RX (V2.2.0)	UART	—	Scalable RISC-V SoC Project (RISC-V RX)
FreeRTOS-LTS PMP-Blinky Project	RISC-V RX (V2.2.0)	UART, GPIO	—	Scalable RISC-V SoC Project (RISC-V RX)
RISC-V RX Demo Project	RISC-V RX (V2.2.0)	UART	—	Scalable RISC-V SoC Project (RISC-V RX)
I2C Communication Project	RISC-V RX (V2.6.0)	I2C Target, I2C Controller	For the connection of pins, refer to the I2C Communication Project section.	Scalable RISC-V SoC Project (RISC-V RX)
SPI Controller Project	RISC-V RX (V2.6.0)	Octal SPI Controller	—	Scalable RISC-V SoC Project (RISC-V RX)
Hardware Interrupt Project (PIC)	RISC-V MC (V2.8.0)	UART	Enable PIC	Scalable RISC-V SoC Project (RISC-V MC)
Hardware Interrupt Project (PLIC)	RISC-V RX (V2.6.0)	UART	Enable PLIC	Scalable RISC-V SoC Project (RISC-V RX)
Mtimer Project	RISC-V MC (V2.8.0), RISC-V SM (V1.8.0)	UART	Enable Mtimer	Scalable RISC-V SoC Project (RISC-V MC, RISC-V SM)
Real Timer Project	RISC-V RX (V2.6.0)	UART	Enable CLINT	Scalable RISC-V SoC Project (RISC-V RX)
Software Interrupt Project	RISC-V RX (V2.6.0)	UART	Enable CLINT	Scalable RISC-V SoC Project (RISC-V RX)
Watchdog Timer Project	RISC-V RX (V2.6.0)	UART	Enable CLINT	Scalable RISC-V SoC Project (RISC-V RX)
Code Coverage Project	RISC-V RX (V2.4.0)	—	Enable Semihost 128 kB memory range	Scalable RISC-V SoC Project (RISC-V RX)
Timing Profiling Project	RISC-V RX (V2.4.0)	—	Enable Semihost 128 kB memory range	Scalable RISC-V SoC Project (RISC-V RX)

5.2. Hello World

This Hello World C project supports all the scalable RISC-V SoC projects.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

It requires UART IP for terminal print-out (Figure 5.1) and GPIO IP for LED display.

Figure 5.1. Hello World Project Terminal

5.3. RTOS

FreeRTOS-LTS Minimal Project, FreeRTOS-LTS PMP-Blinky Project, and RISC-V RX Demo Project are RTOS application templates.

These templates need the Scalable RISC-V SoC project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2.

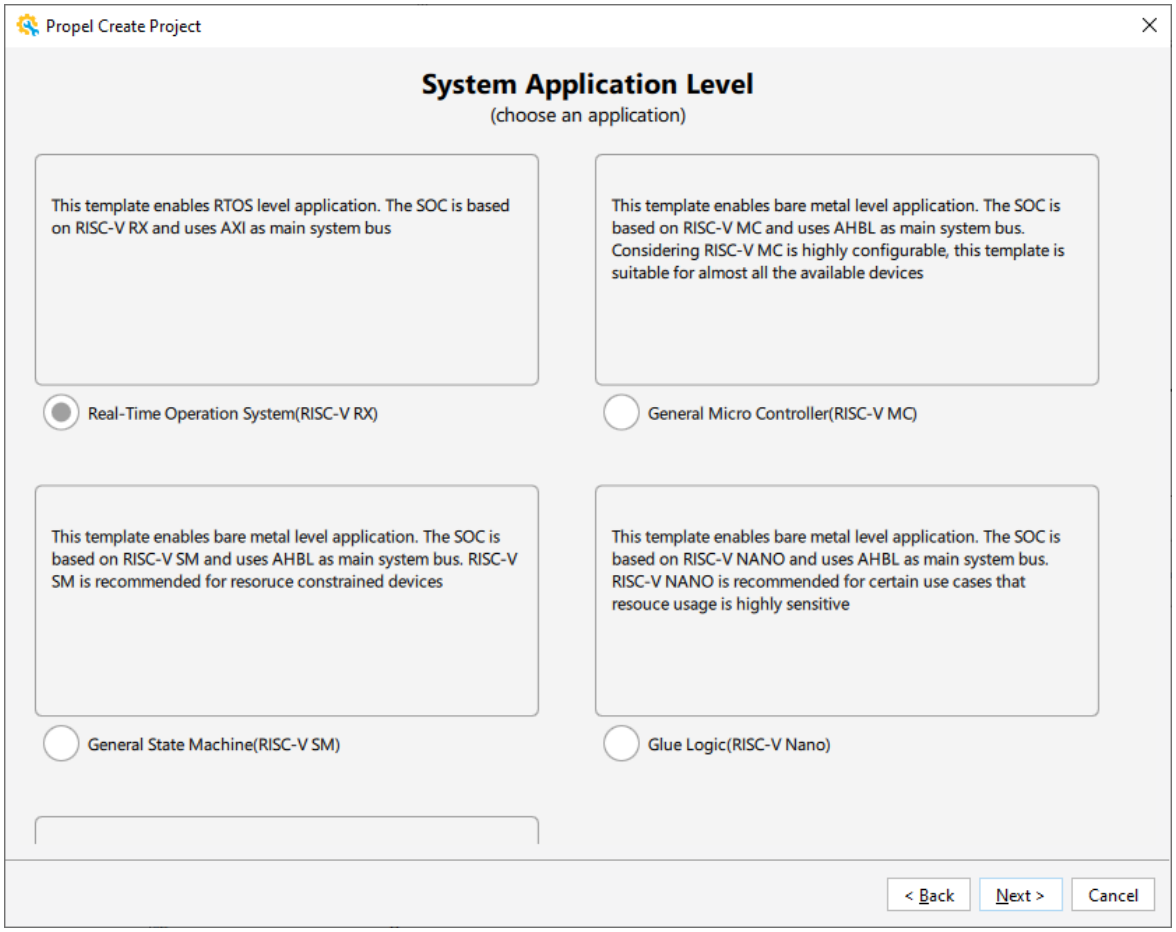


Figure 5.2. Scalable SoC Project – Real-Time Operation System (RISC-V RX)

The FreeRTOS-LTS PMP-Blinky Project template shows how to create tasks, timer, and running tasks. When the project runs correctly, the terminal print-out is shown in Figure 5.3.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

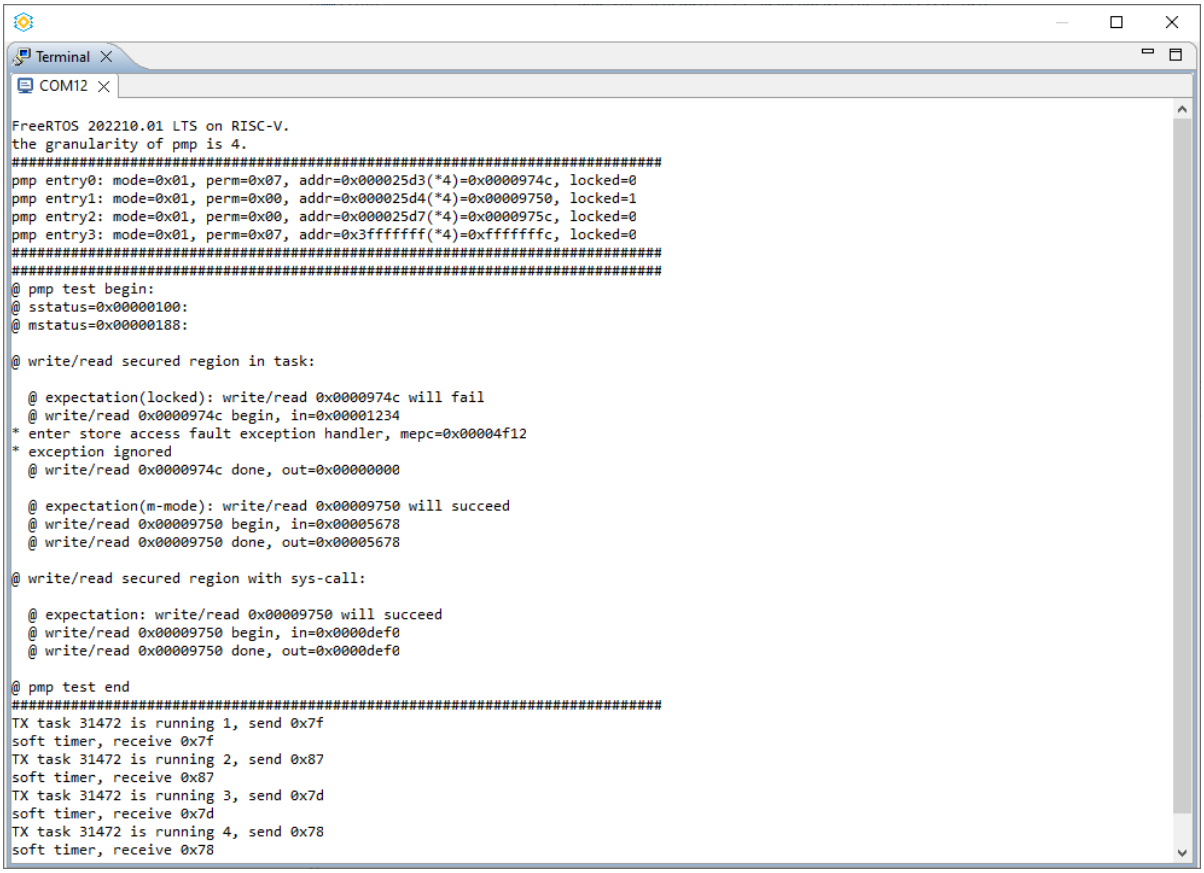


Figure 5.3. FreeRTOS-LTS PMP-Blinky Project Terminal Print-out

5.4. Single Function

Single function here refers to the corresponding module of each RISC-V processor. The single-function template refers to each processor module's reference code.

For details of each RISC-V processor and the corresponding module, refer to its IP user guide (Table 5.2).

Table 5.2. RISC-V Processor User Guide

RISC-V RX	RISC-V RX CPU IP Core
RISC-V MC	RISC-V MC CPU IP Core
RISC-V SM	RISC-V SM CPU IP Core
RISC-V Nano	RISC-V Nano CPU IP Core

5.4.1. Hardware Interrupt Project (PIC)

This template shows how to use the RISC-V MC processor's Programmable Interrupt Controller (PIC) module.

This template requires the Scalable RISC-V SoC project, General Micro Controller (RISC-V MC), as shown in Figure 5.4.

When this project is running, you can push the button mentioned in the corresponding code. Then, you can see the terminal print-out (Figure 5.5) and LED display.

Note: This template supports only three boards by default: CertusPro-NX Evaluation Board, Certus-NX Versa Evaluation Board, and CrossLink-NX Evaluation Board.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

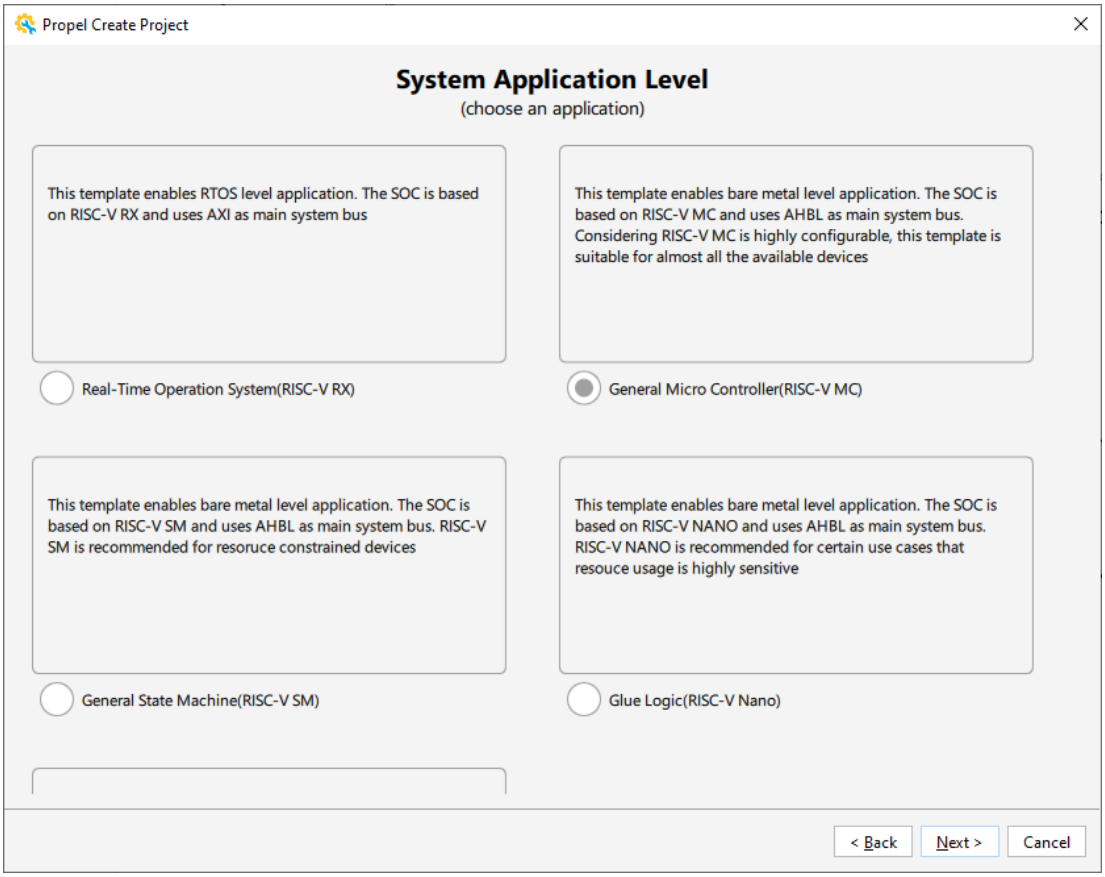


Figure 5.4. Scalable SoC Project – General Micro Controller (RISC-V MC)

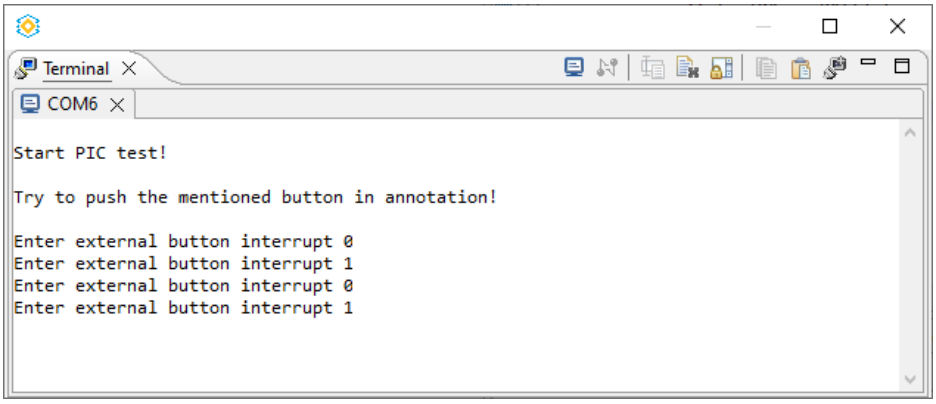


Figure 5.5. Hardware Interrupt Project (PIC) Project Terminal Print-out

5.4.2. Mtimer Project

This template shows how to use RISC-V MC processor and RISC-V SM processor's Mtimer module.

This template requires the Scalable RISC-V SoC Project, General Micro Controller (RISC-V MC) shown in Figure 5.4, or the Scalable RISC-V SoC Project, General Micro Controller (RISC-V SM) shown in Figure 5.6.

This project uses mtimer for delay function and provides the mdelay() function using the timer cycle counter and the irq_mdelay() function using the timer interrupt. The terminal print-out is shown in Figure 5.7.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

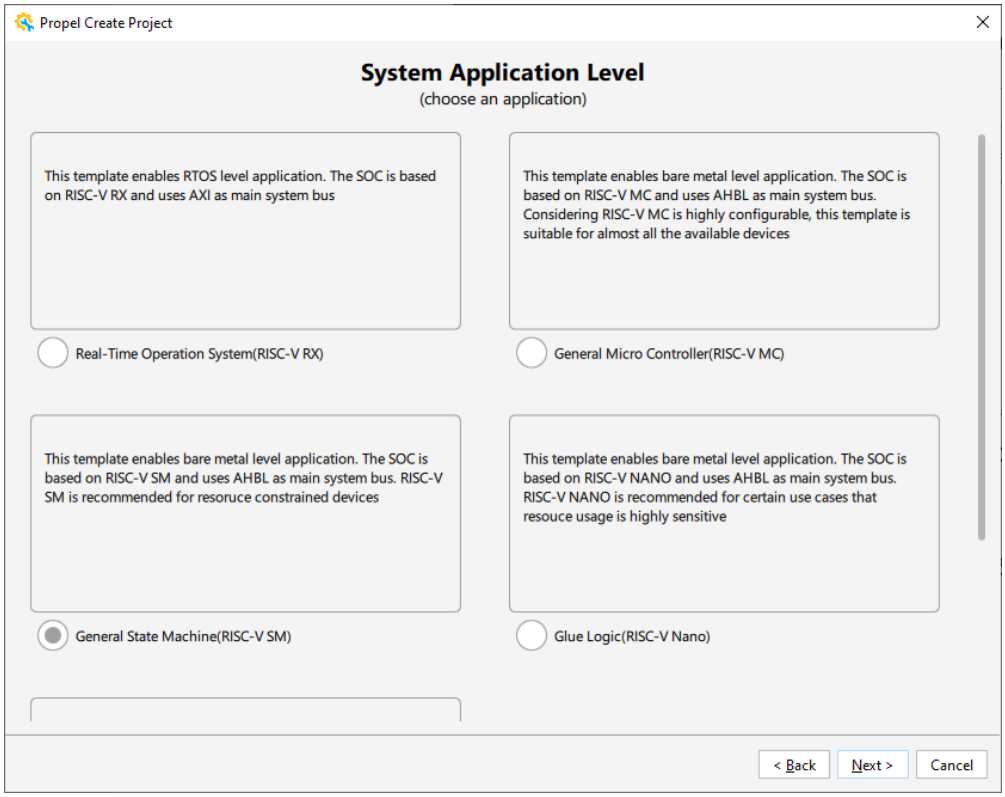


Figure 5.6. Scalable SoC Project – General State Machine (RISC-V SM)

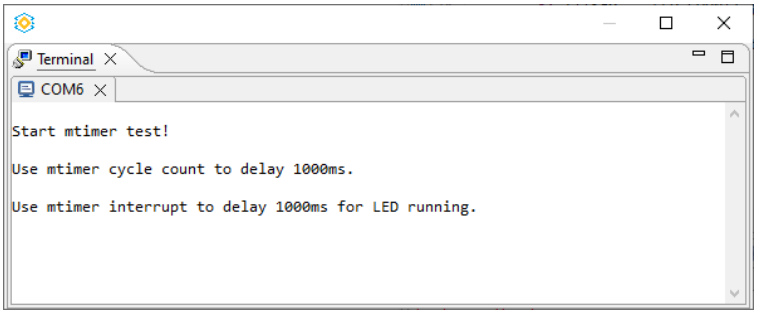


Figure 5.7. Mtimer Project

5.4.3. Hardware Interrupt Project (PLIC)

This template shows how to use the RISC-V RX processor's Platform-Level Interrupt Controller (PLIC) module.

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2.

When this project is running, you can push the button mentioned in the corresponding code. Then, you can see the terminal print-out (Figure 5.8) and LED display.

Note: This template supports only four boards by default: CertusPro-NX Evaluation Board, Certus-NX Versa Evaluation Board, CrossLink-NX Evaluation Board, and Avant-E Evaluation Board.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

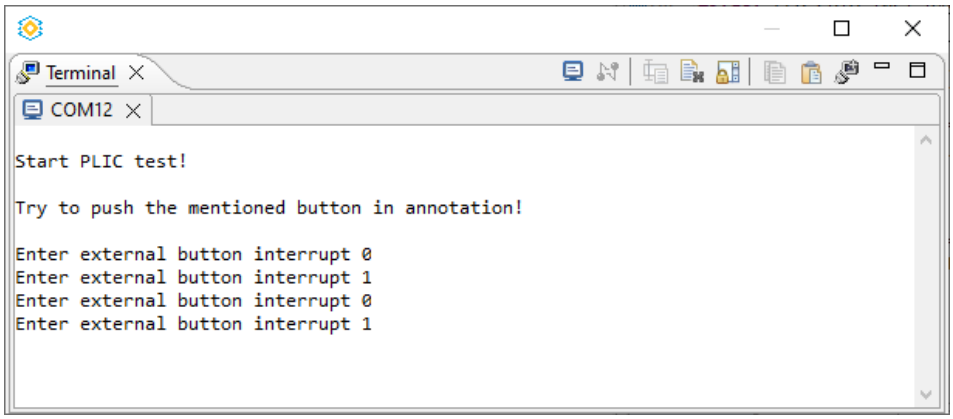


Figure 5.8. Hardware Interrupt Project (PLIC) Project Terminal Print-out

5.4.4. Real Timer Project

This template shows how to use the RISC-V RX processor's CLINT-mtimer module. This timer is a real-time clock of 32 kHz.

RTOS projects always use this real-time clock to generate ticks.

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. This project uses the real timer to generate a simple stopwatch. The terminal print-out is shown in Figure 5.9.

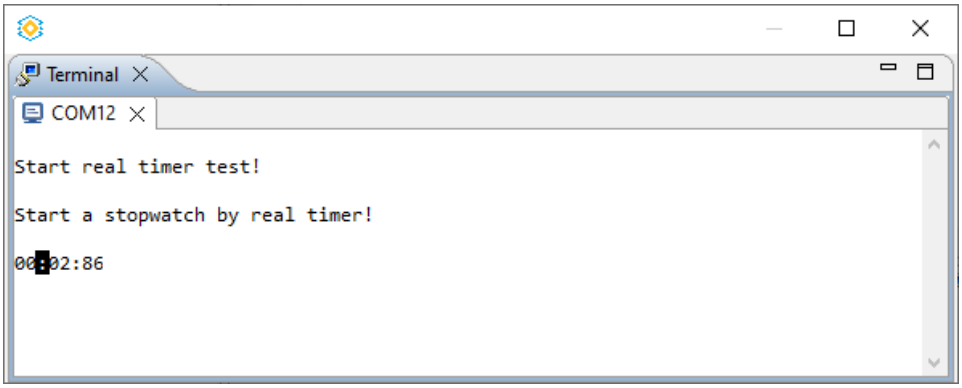


Figure 5.9. Real Timer Project

5.4.5. Software Interrupt Project

This template shows how to use the RISC-V RX processor's CLINT-MSIP module.

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. This project uses CLINT-MSIP to generate a simple task scheduler, which act as a multi-thread program. The terminal print-out is shown in Figure 5.10.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

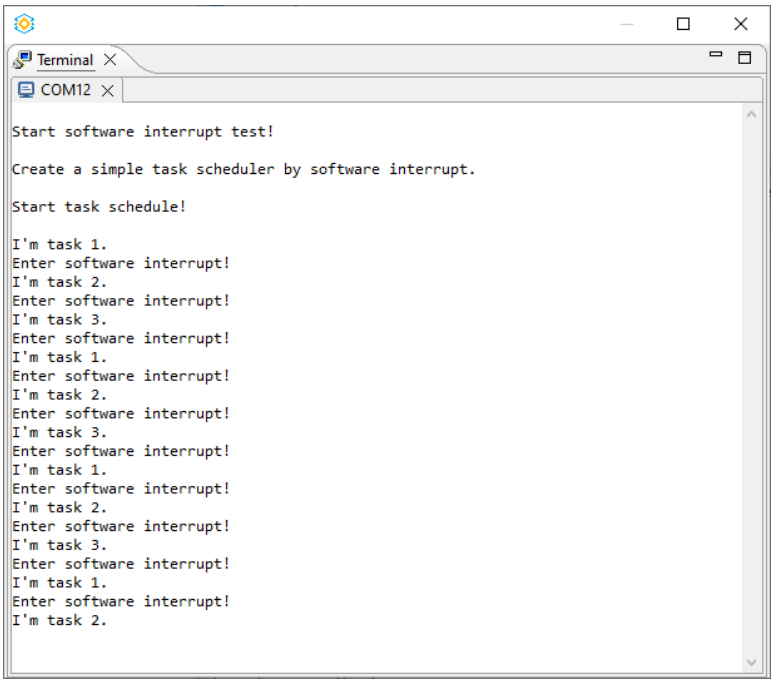


Figure 5.10. Software Interrupt Project

5.4.6. Watchdog Timer Project

This template shows how to use the RISC-V RX processor's watchdog timer (WDT).

This template needs the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2.

This WDT is a software WDT. It generates a system reset signal when the program goes into endless loop and cannot feed the WDT in time.

Watchdog Timer Project sets a main loop and feeds WDT in every big cycle. There is a simulation busy status, which is set by the set_busy_for_demo() function and get by the get_busy_status_for_demo() function. This busy status is like a peripheral's status and the program should check its status before using it.

The terminal print-out is shown in Figure 5.11.

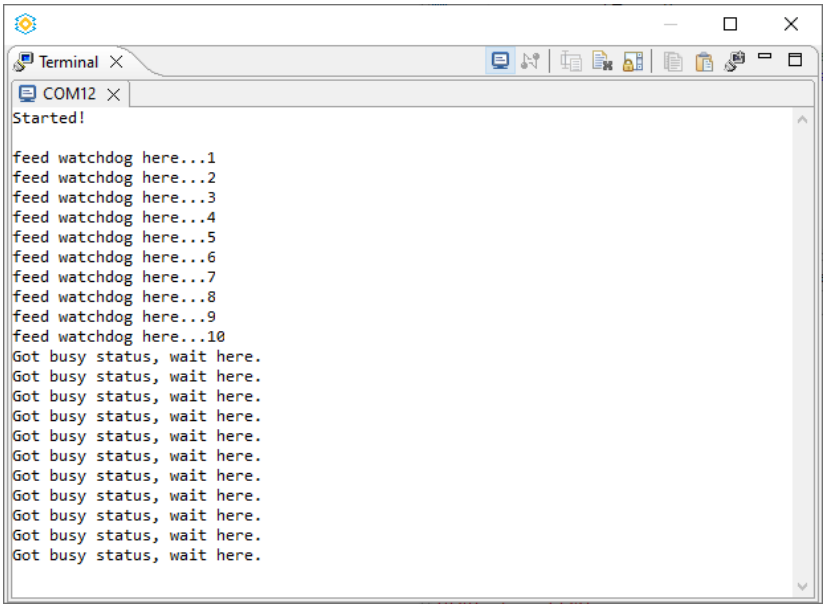


Figure 5.11. Watchdog Timer Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5.5. IP Usage Reference

5.5.1. I2C Communication Project

This template needs the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. This SoC project includes an I2C controller instance and I2C target instance.

This project requires connecting pins according to the annotation in the main() function.

When communicating successfully, the terminal print-out is shown in Figure 5.12. I2C controller can send to or get message from the existing target at the address 0x31. Communication fails with an invalid target at address 0x32. This is expected upon the current design.

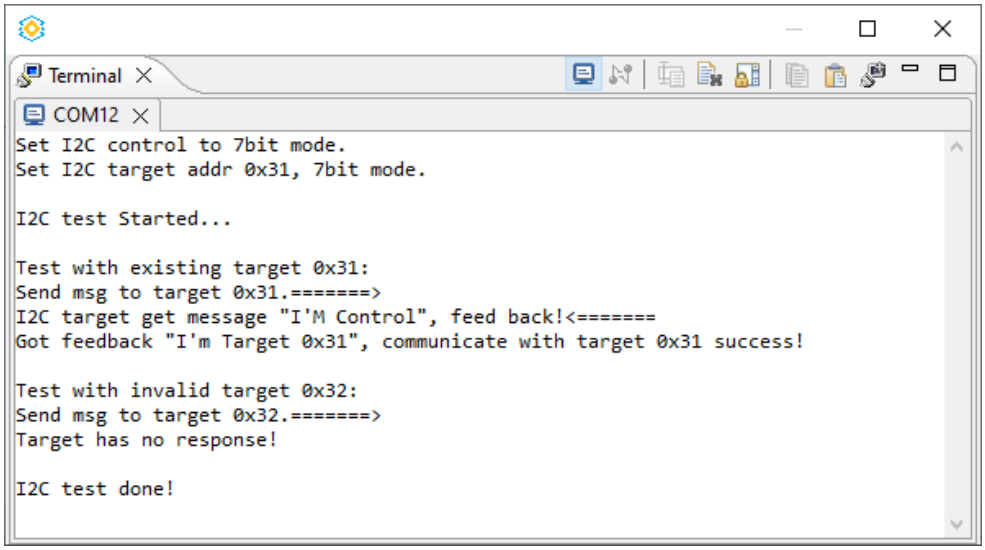


Figure 5.12. I2C Communication Project

5.5.2. SPI Controller Project

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. This SoC project includes the Octal SPI Controller instance. The Octal SPI Controller is designed to be connect with an SPI NOR flash.

This project shows how to use an SPI controller to read or write an SPI NOR flash.

This project can be created in two forms: read or write between a RAM and a flash; read or write between a file and a flash. The different forms depend on the System Library selection in the C project creating flow.

- Read or write between a RAM and a flash

  In the C project creating flow, under the **Lattice Toolchain Setting** page, select Default for **System Library** (Figure 5.13). The terminal print-out is shown in Figure 5.14.

  The program logic is designed as follows:

  a. Match the flash ID and get the flash size.
  b. Write data to the end of the flash.
  c. Read back the data from the end of the flash.
  d. Compare the data and the report.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

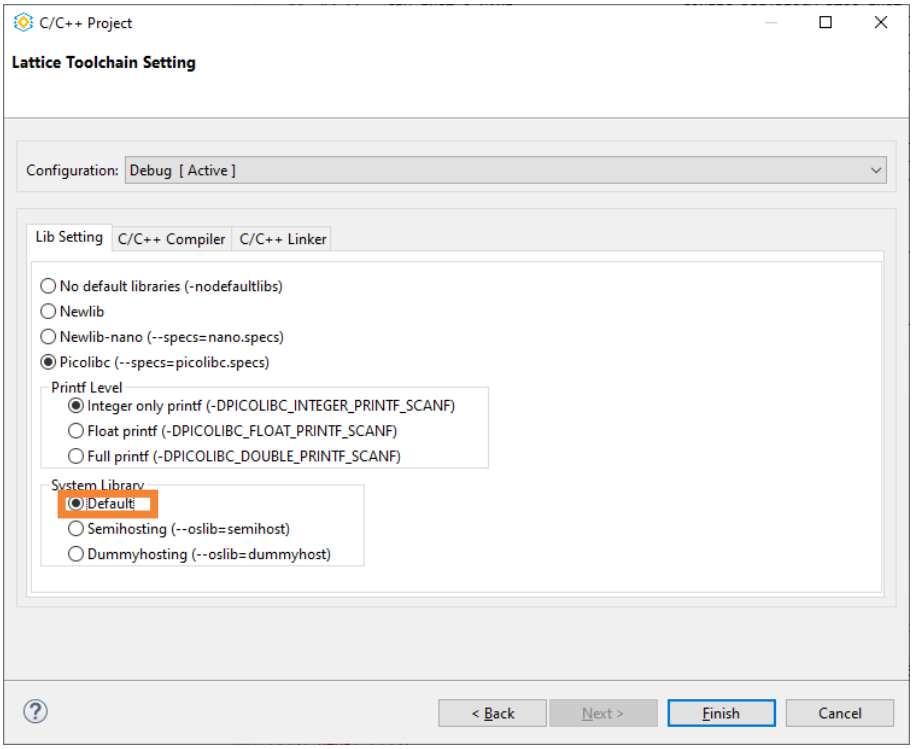


Figure 5.13. Selecting Default for System Library

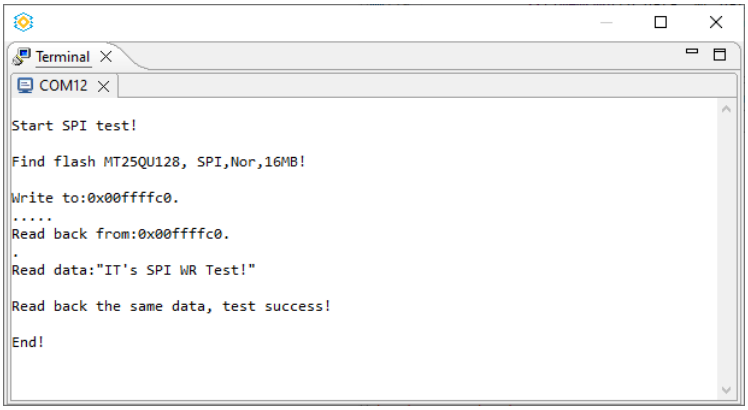


Figure 5.14. Read or Write between the RAM and the Flash

- Read or write between a file and a Flash

  In the C project creating flow, under the **Lattice Toolchain Setting** page, select Semihosting for **System Library**, (Figure 5.15).

  The print-out message shows in the console window (Figure 5.16).

  The program logic is designed as follows:

  a. Match the flash ID and get the flash size.

  b. Read data from the start of the flash and save it into the file flash_use2write.bin.

  c. Write the file flash_use2write.bin to the end of the flash.

  d. Read back the data from the end of flash and save it into the file flash_readback.bin.

When the program finishes running, you can find the two files, flash_use2write.bin and flash_readback.bin in the project folder (Figure 5.17). Compare these two files to verify if the read or write operation is correct.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

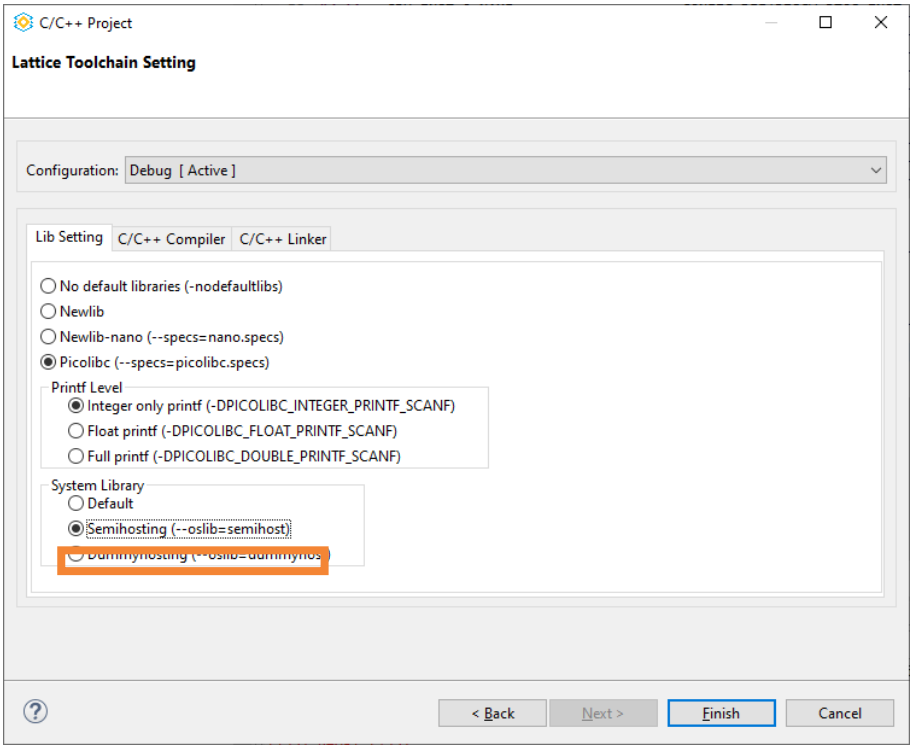


Figure 5.15. Selecting Semihosting System Library

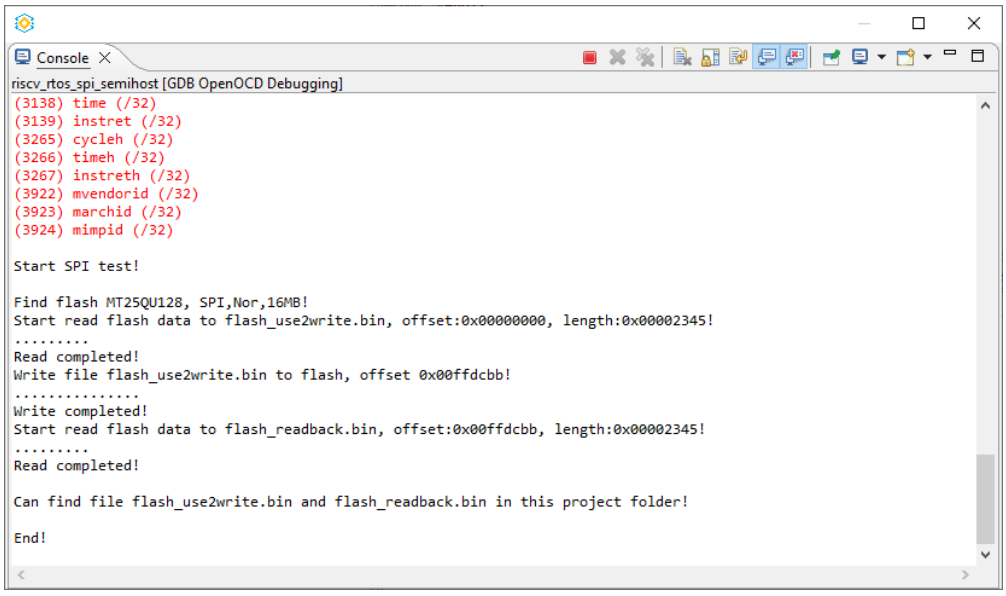


Figure 5.16. Read or Write between File and Flash

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

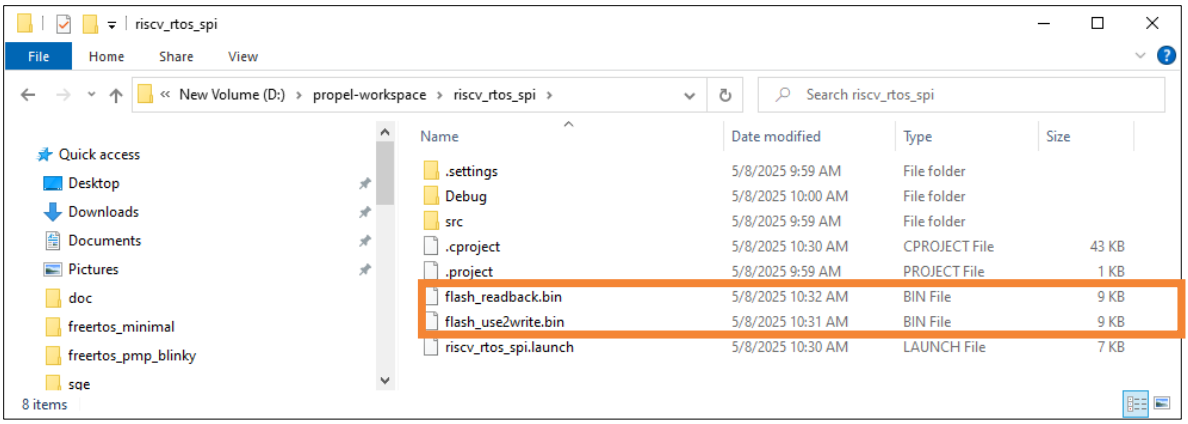


Figure 5.17. Project Folder

5.6. Profiling Tool

5.6.1. Code Coverage Project

Code coverage is about validating the number of lines of code 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.

Refer to gcov—a Test Coverage Program for more details.

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. As this template has more code and larger memory size, the corresponding RISC-V RX project created needs to have 128 kB Memory Range (Figure 5.22).

In the C project creating flow, some default selections are different from those of other templates. Keep these default selections.

The print-out log is shown in Figure 5.18. Then, you can find the coverage files in **Project Explorer** shown in Figure 5.19. Double click one of the coverage files and click **OK** (Figure 5.20).

Wait for a few seconds, the coverage information is displayed (Figure 5.21).

For detailed usage of code coverage, refer to Help - Eclipse Platform and enter Gcov View in the Search box.

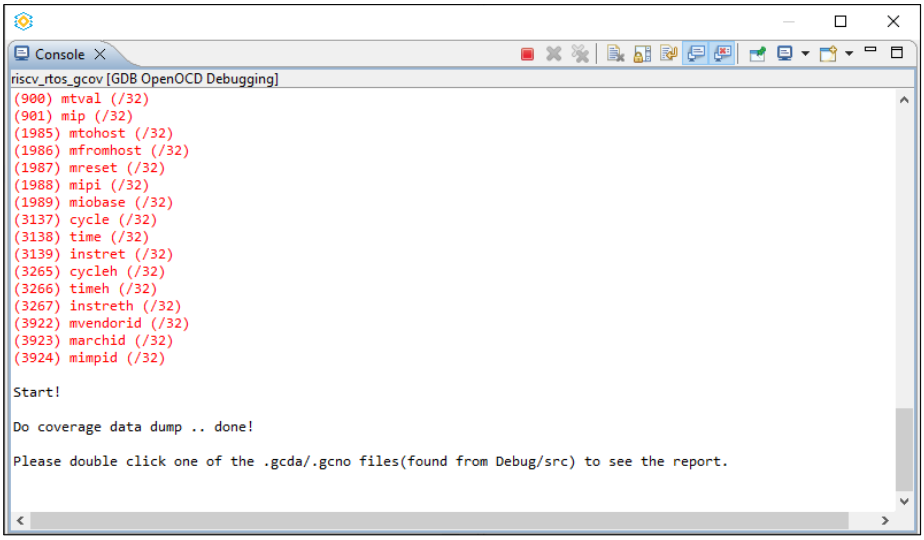


Figure 5.18. Code Coverage Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

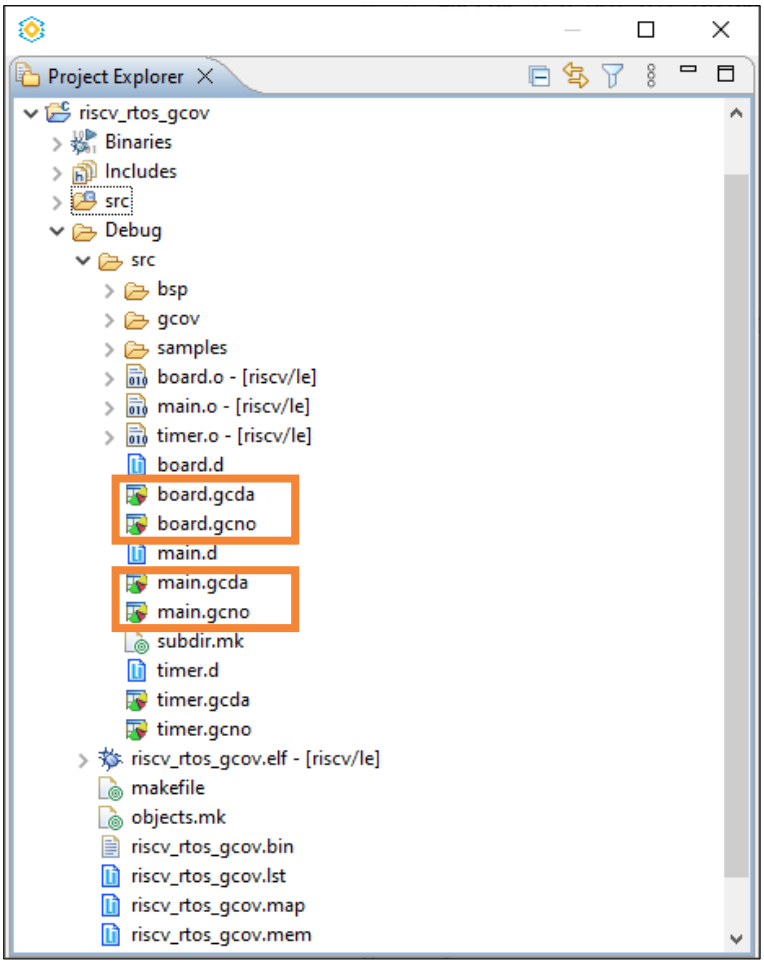


Figure 5.19. Code Coverage Files

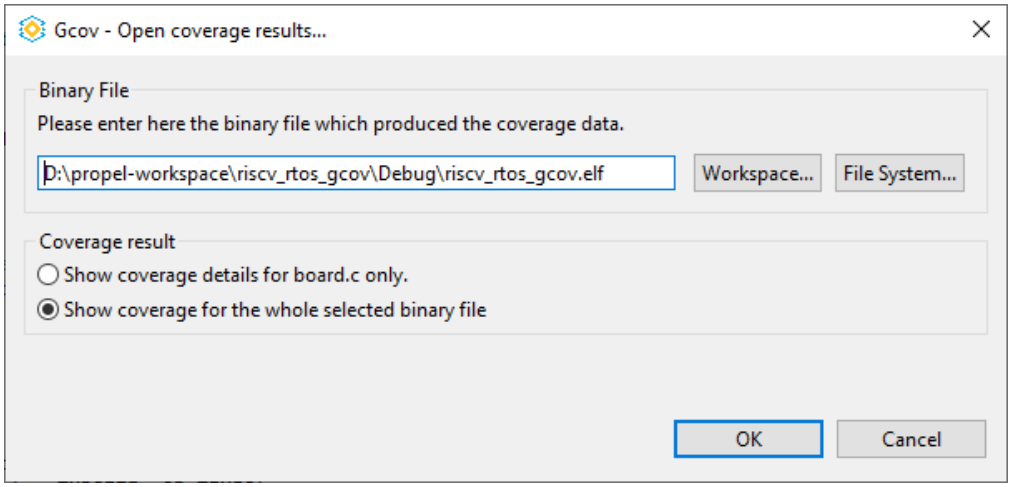


Figure 5.20. Open Coverage Results

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

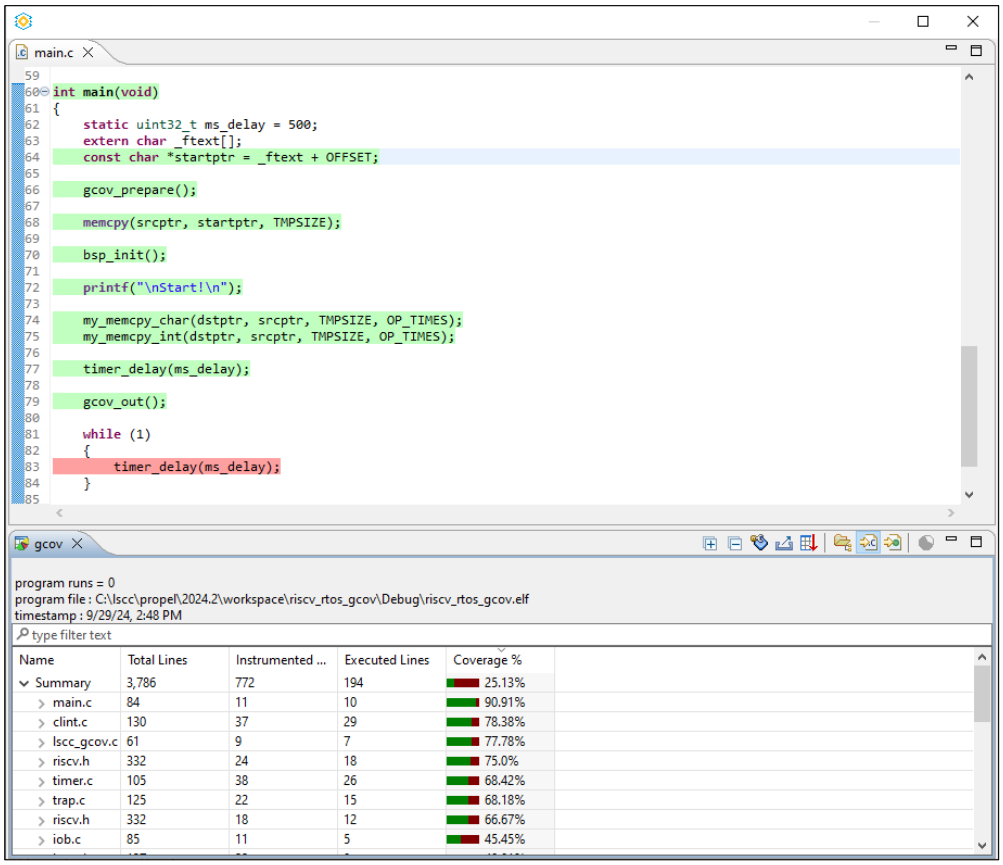


Figure 5.21. Code Coverage Information

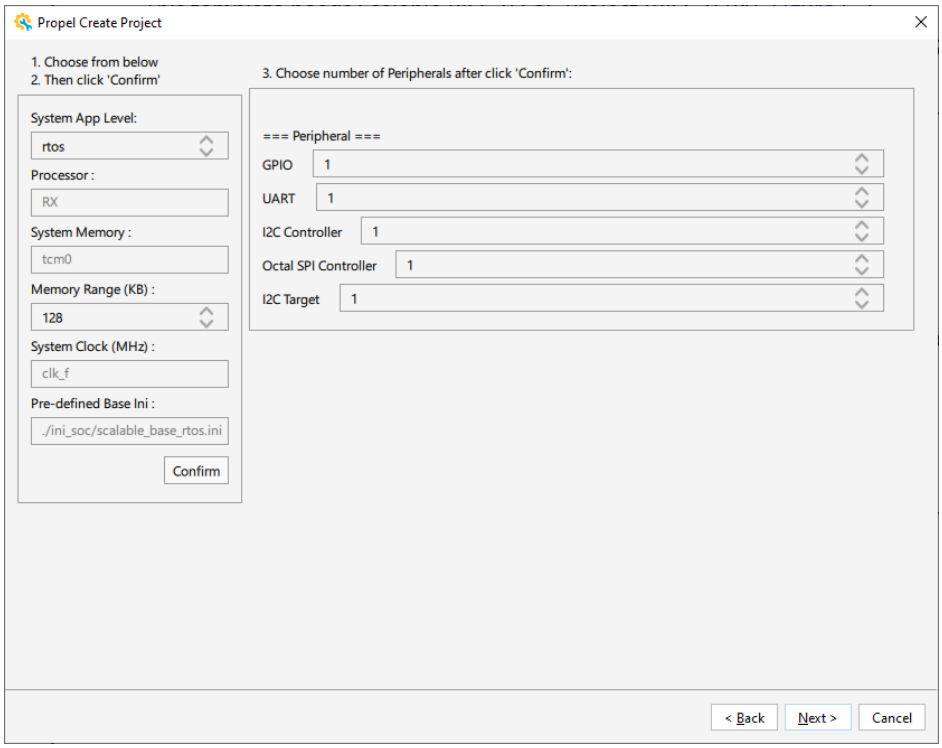


Figure 5.22. Scalable RISC-V SoC Project – Real-Time Operation System (RISC-V RX) Confirm Page

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5.6.2. How to Add Code Coverage Function to an 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 5.23).
4. Select **GNU RISC-V Cross C Compiler** > **Miscellaneous**. Add the compiler flag, -fprofile-arcs -ftest-coverage (Figure 5.24).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgcov (Figure 5.25).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Add the linker flag, --defsym=_HEAP_SIZE=0x1000 (Figure 5.26). **Note:** 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 under the label, Other linker flags. Make sure the linker flag, --oslib=semihost, is supported (Figure 5.27).
8. Click **Apply and Close**.
9. Add the line `scoverage_init();` to the head of the code section. Add the line `scoverage_dump();` to the end of the code section. You can refer to the sample template.
10. Rebuild this C project.

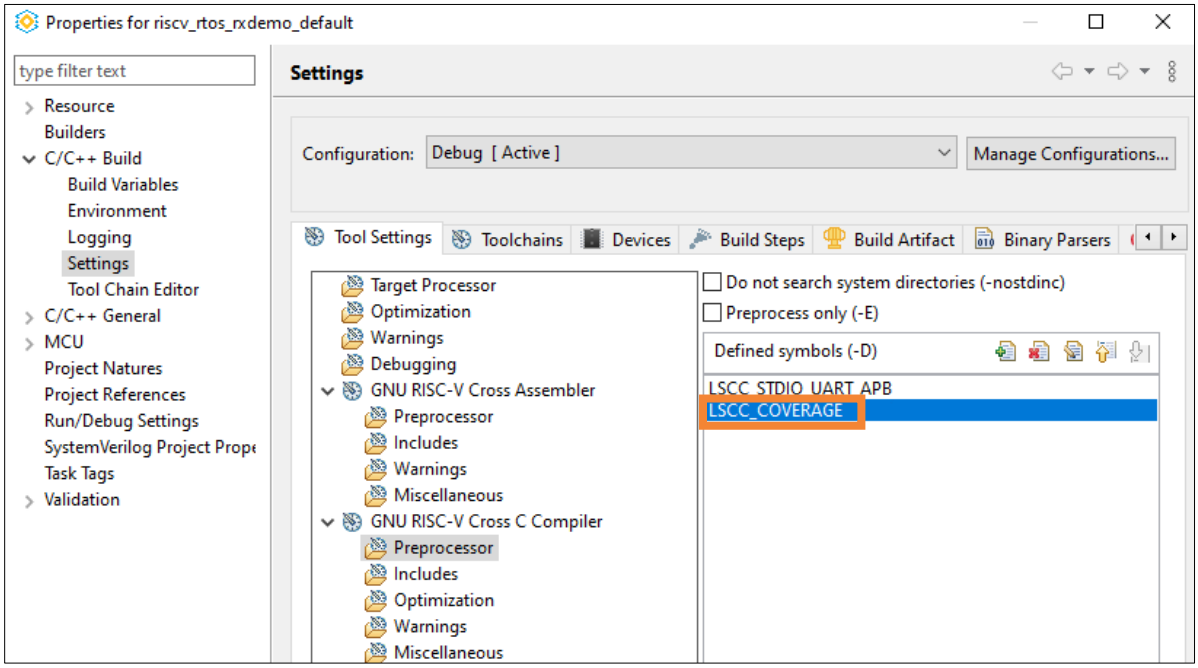


Figure 5.23. LSCC_COVERAGE Symbol

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

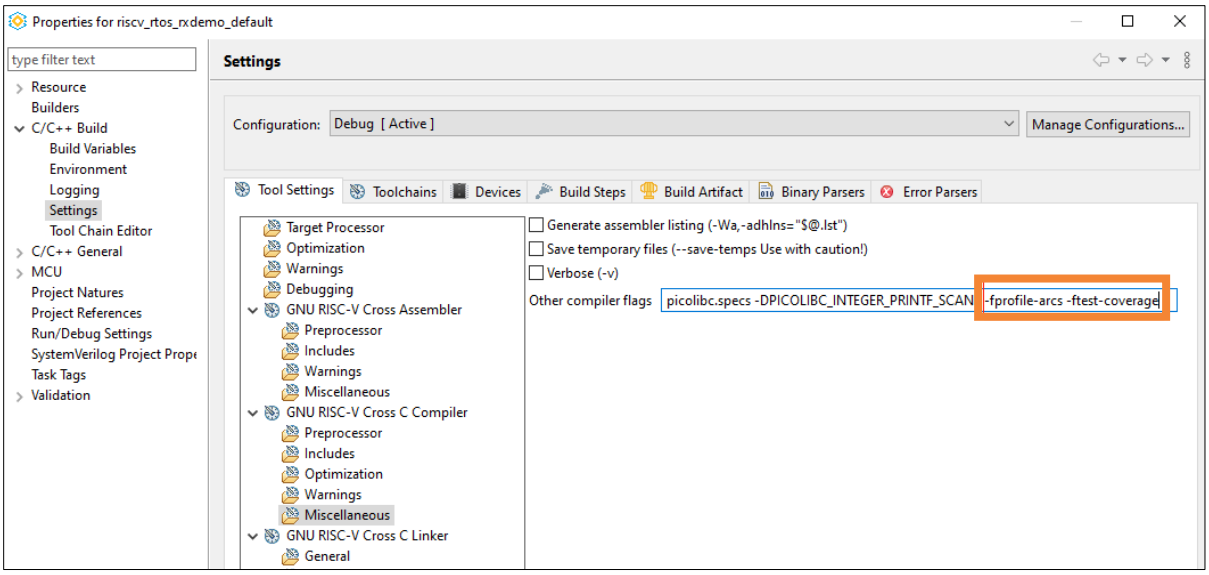


Figure 5.24. -fprofile-arcs -ftest-coverage Compiler Flag

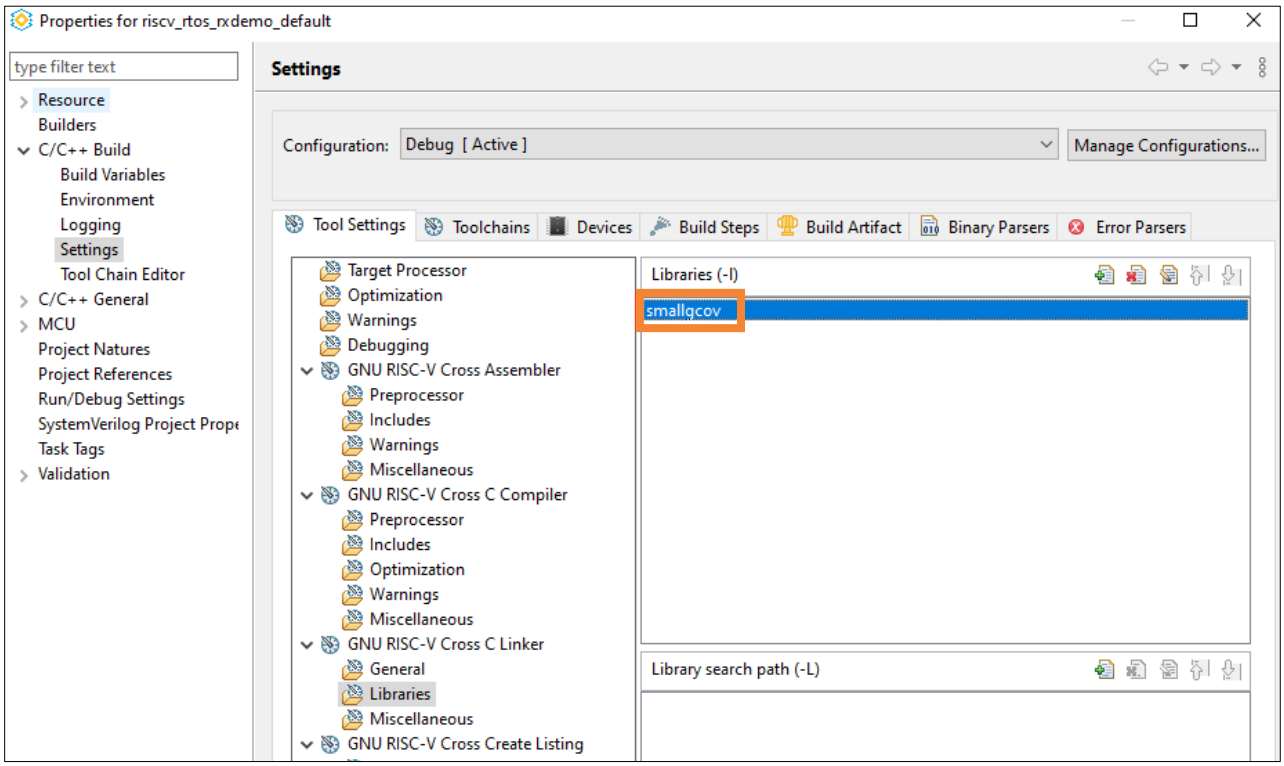


Figure 5.25. smallgcov Library

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

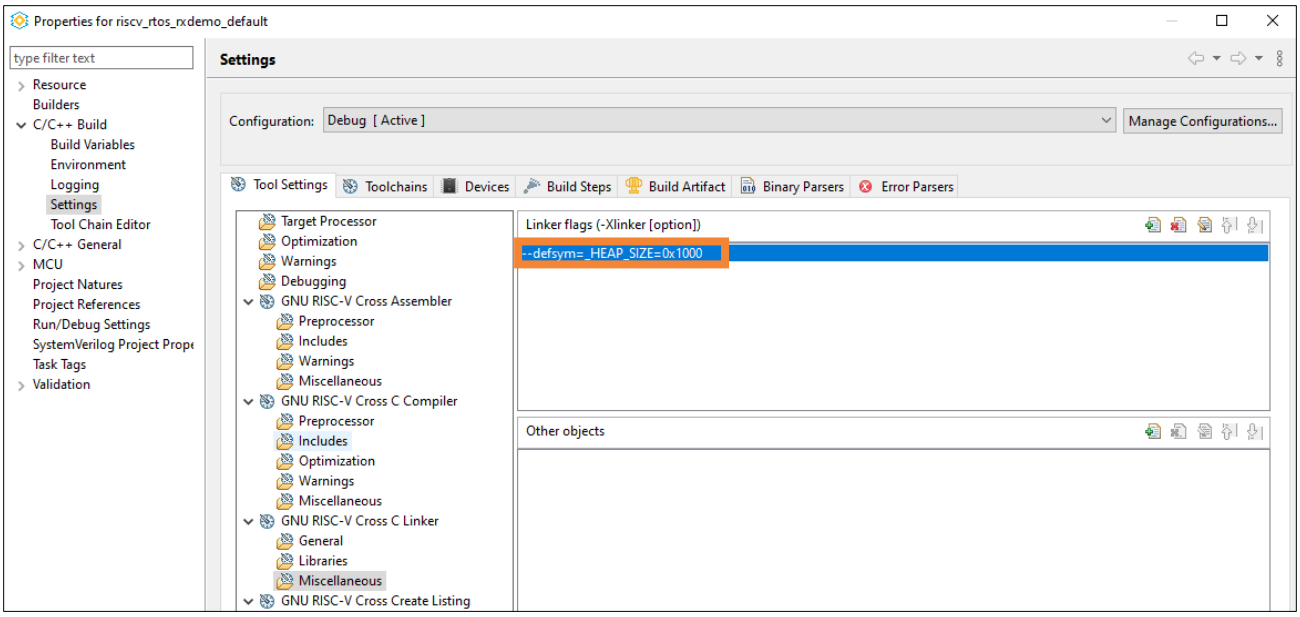


Figure 5.26. --defsym=_HEAP_SIZE=0x1000 Linker Flag

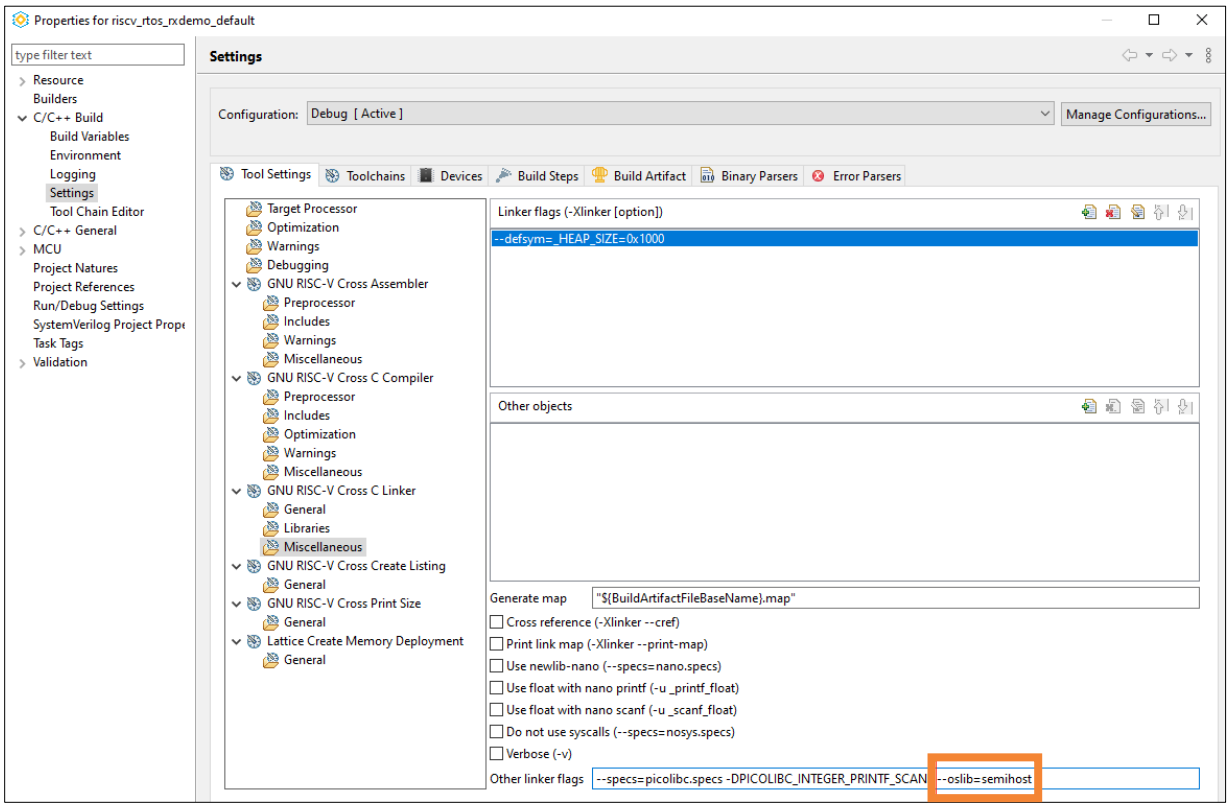


Figure 5.27. --oslib=semihost Linker Flag

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5.6.3. Timing Profiling Project

Timing profiling is an important aspect of software programming. Through profiling, you 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.

Refer to the GNU gprof document for more details.

This template requires the Scalable RISC-V SoC Project, Real-Time Operation System (RISC-V RX), as shown in Figure 5.2. As this template has more code and larger memory size, the corresponding RISC-V RX project created needs to have 128 kB Memory Range (Figure 5.22).

In this C project creating flow, some default selections are different from other templates. Keep these default selections.

The print-out log is shown in Figure 5.28. Then, you can find the gmon.out file in Project Explorer shown in Figure 5.29. Double-click this file (Figure 5.29). Click **OK**.

Wait for a few minutes. The gprof Viewer opens (Figure 5.30).

You can click the icons in the gprof Viewer to change the display style, such as **sort samples per function** (Figure 5.30).

You can also click the title of every column to sort (Figure 5.30).

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

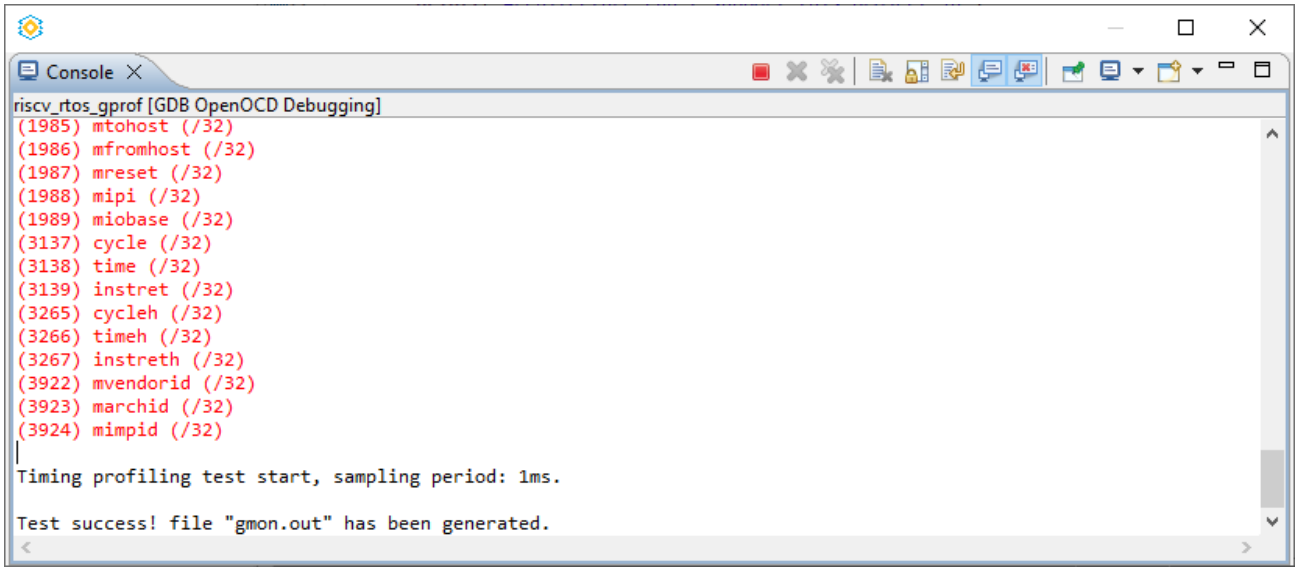


Figure 5.28. Timing Profiling Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

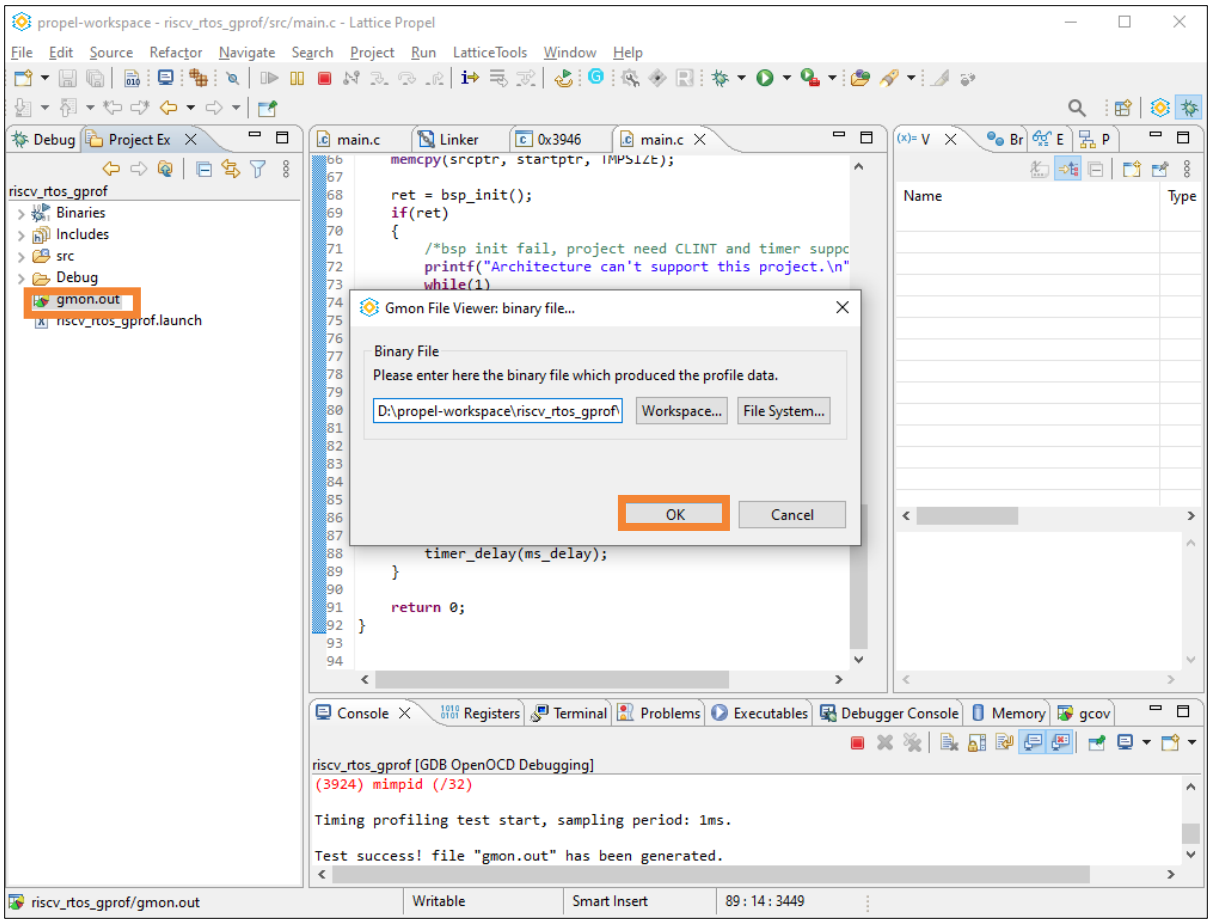


Figure 5.29. gmon File Viewer

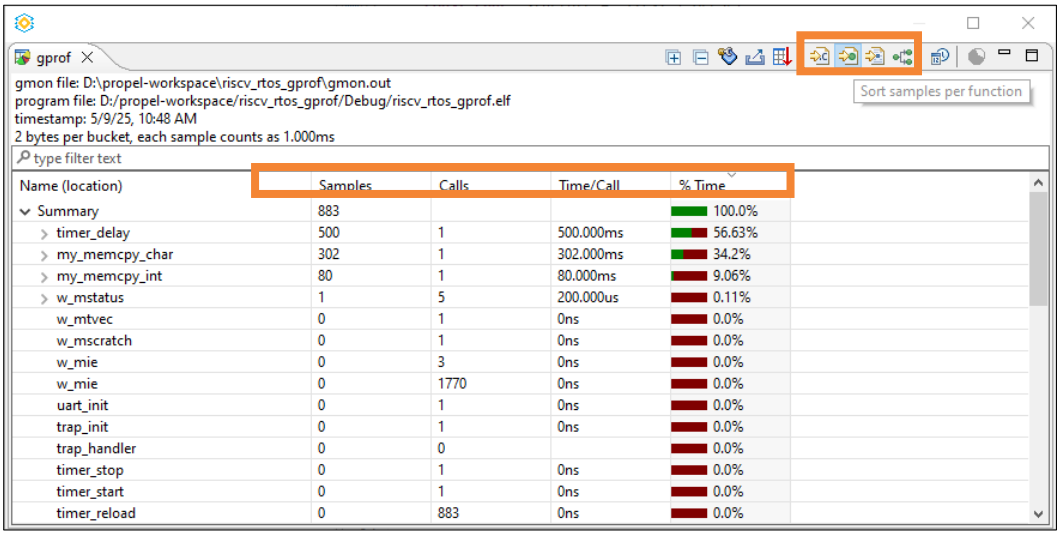


Figure 5.30. gprof Viewer

5.6.4. How to Add Timing Profiling Function to an Existing C Project

1. In the **Project Explorer** view, select the existing C project.
2. Select **Project** > **Properties** > **C/C++ Build** > **Settings.**

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Select **Debugging.** Enable the **Generate gprof information** checkbox (Figure 5.31).
4. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_GPROF (Figure 5.32).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgprof (Figure 5.33).
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 5.34).
7. Click **Apply and Close**.
8. Open file **linker.ld** in the src folder. Set an enough value for _HEAP_SIZE (Figure 3.35).

   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 the src folder. These files can be found in the Timing Profiling Project.
10. Enable a hard timer. Add the `profile_handler()` function into timer handle. Refer to Timing Profiling Project.
11. Add the line `gprof_init();` to the line where the profiling starts. Add the line `gmon_out();` to the line where the profiling stops.
12. Refer to Timing Profiling Project and make sure the code is correct.
13. Rebuild this C project.

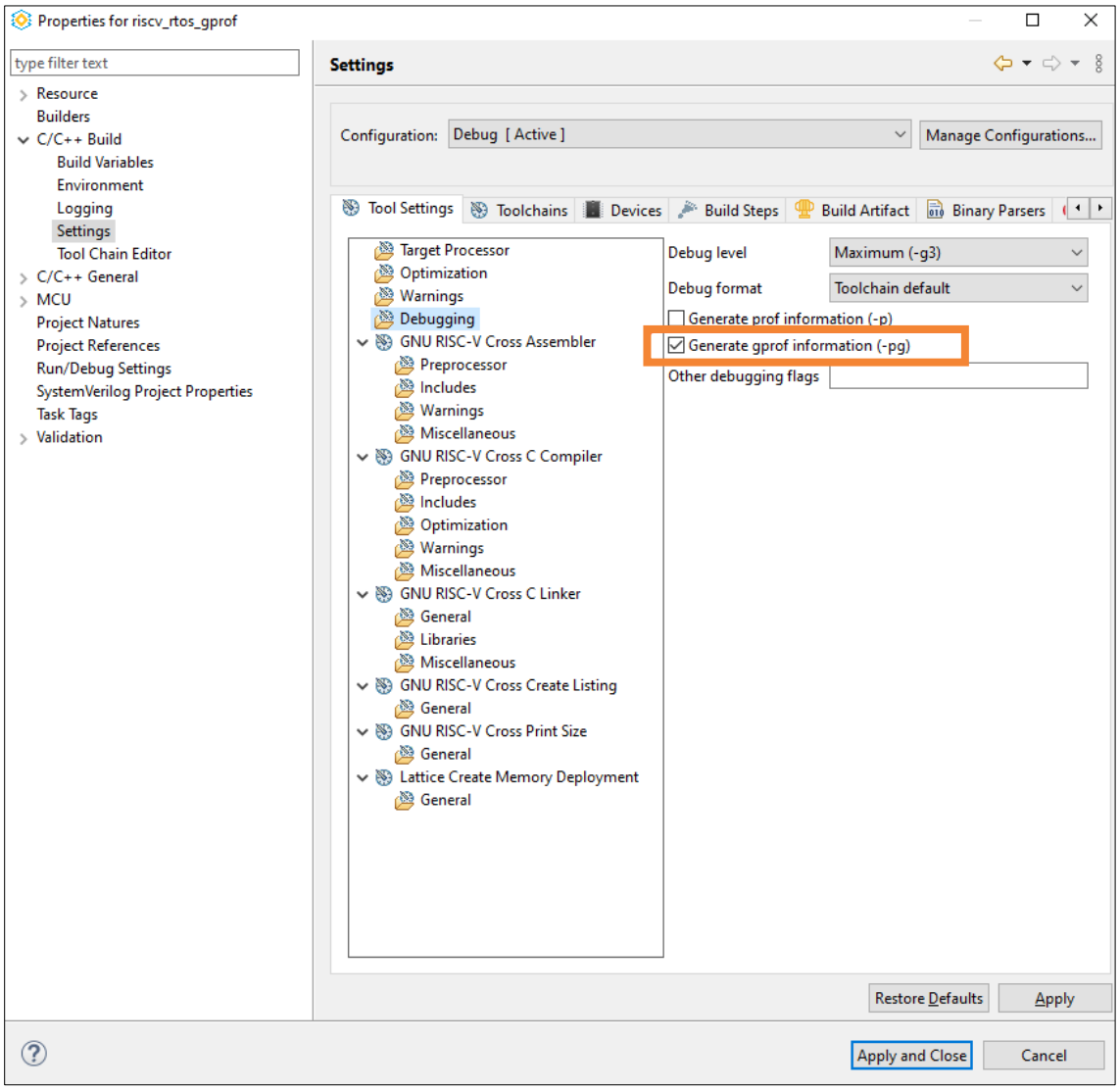


Figure 5.31. Generate gprof Information Checkbox

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

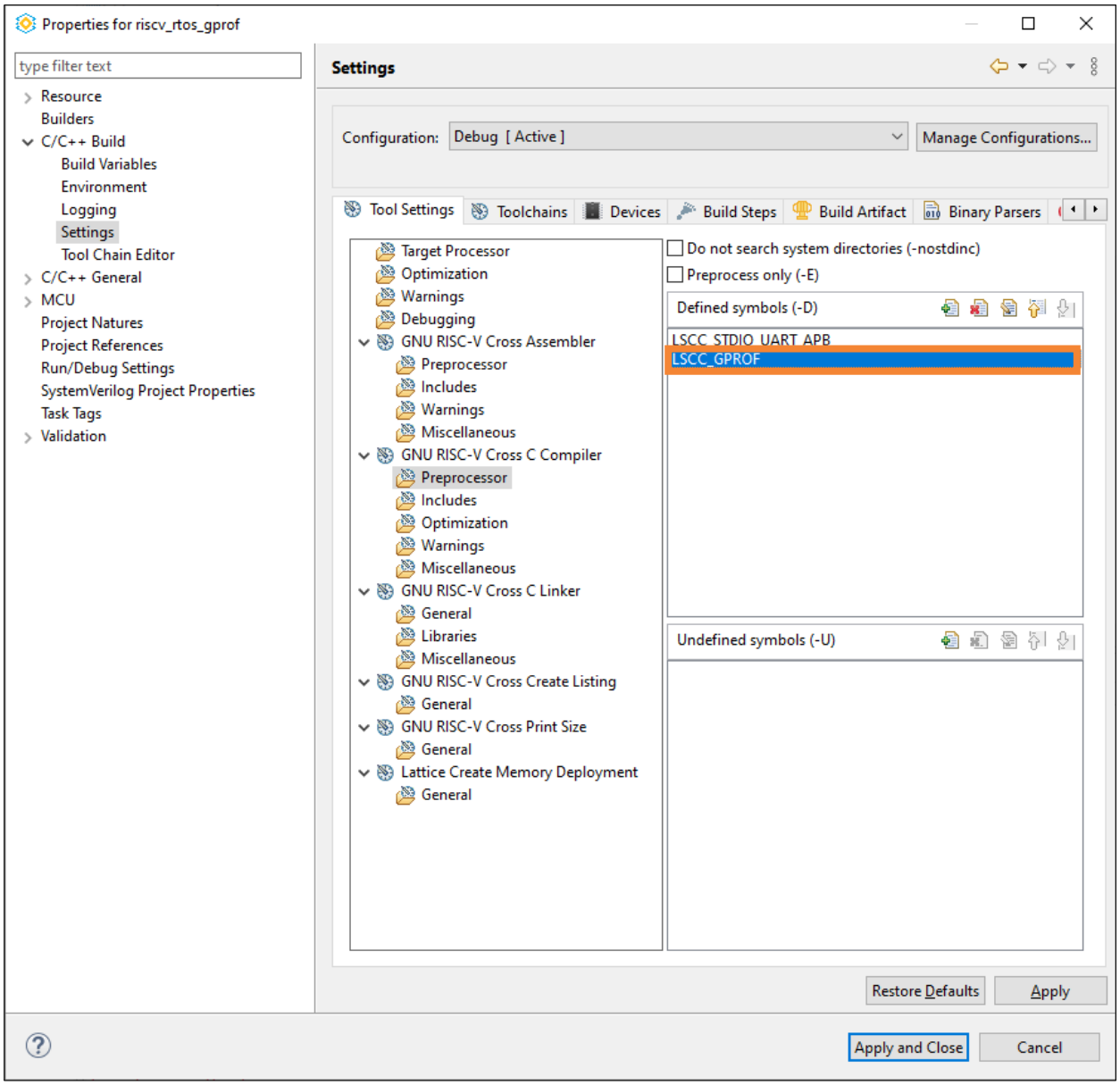


Figure 5.32. LSCC_GPROF Symbol

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

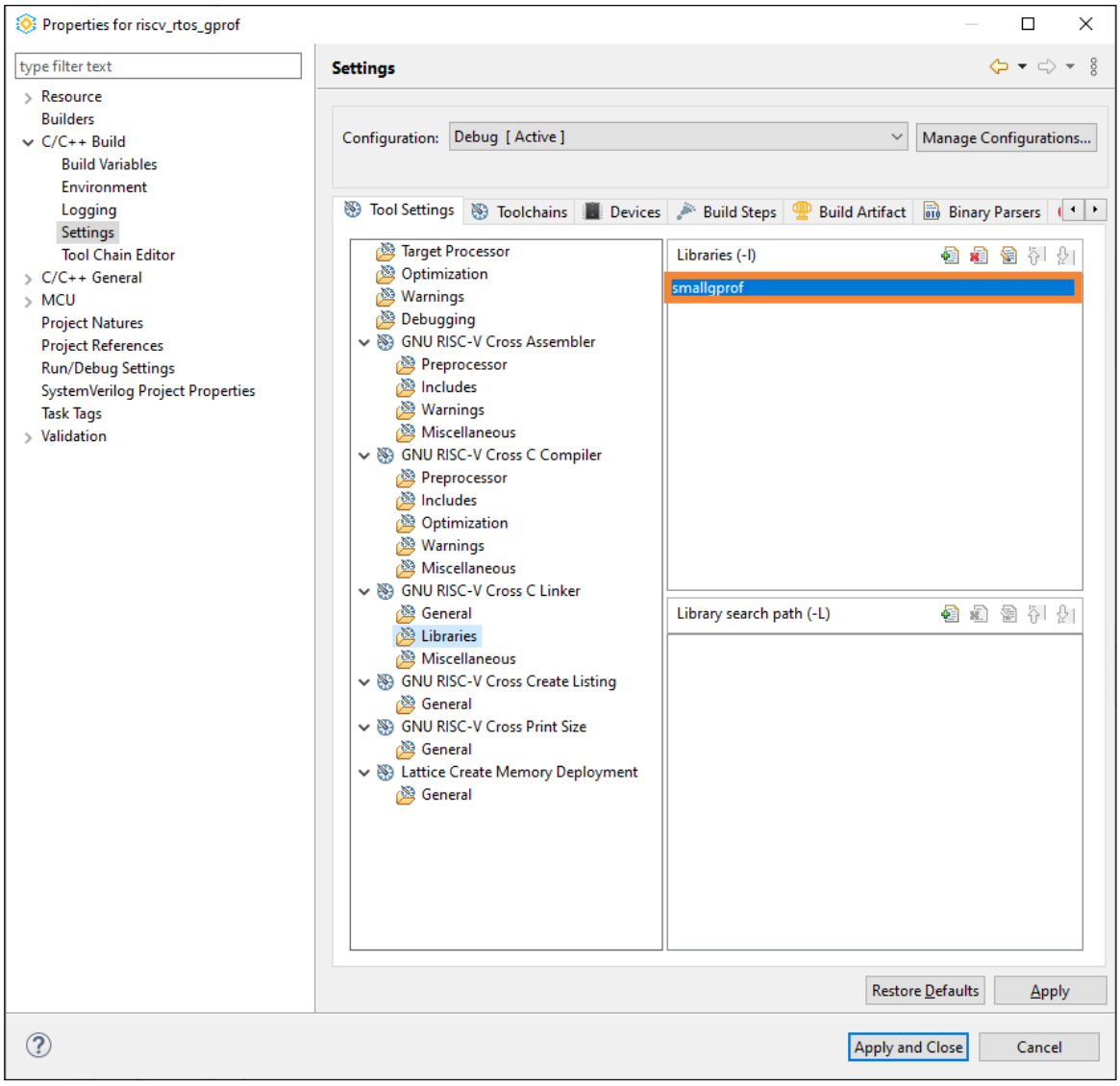


Figure 5.33. smallgprof Link Library

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

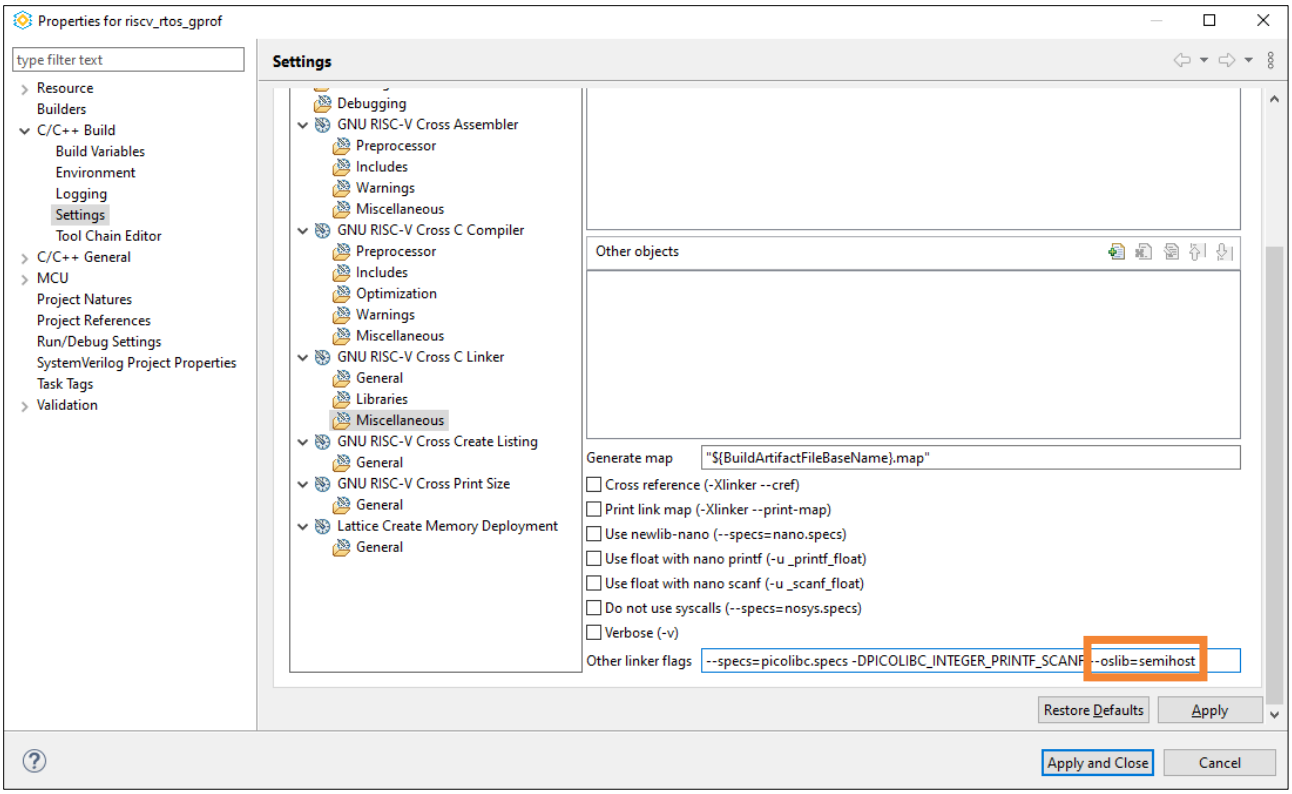


Figure 5.34. --oslib=semihost Linker Flag

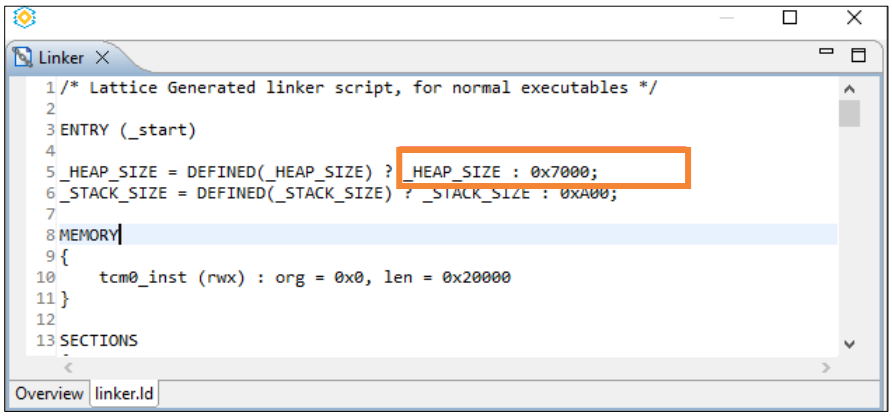


Figure 5.35. _HEAP_SIZE in Linker Script File

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

6. Lattice Propel Tutorial – Hello World

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 the RS232 UART function. To enable RS232 UART function on the MachXO3D Breakout Board, the following reworks on the board are required.

For more details of this board, refer to MachXO3D Breakout Board User Guide (FPGA-UG-02084).

1. Perform hardware reworks on the MachXO3D Breakout Board.

   Short R14 and R15 using 0 Ω resistors, as shown in Figure 6.1.

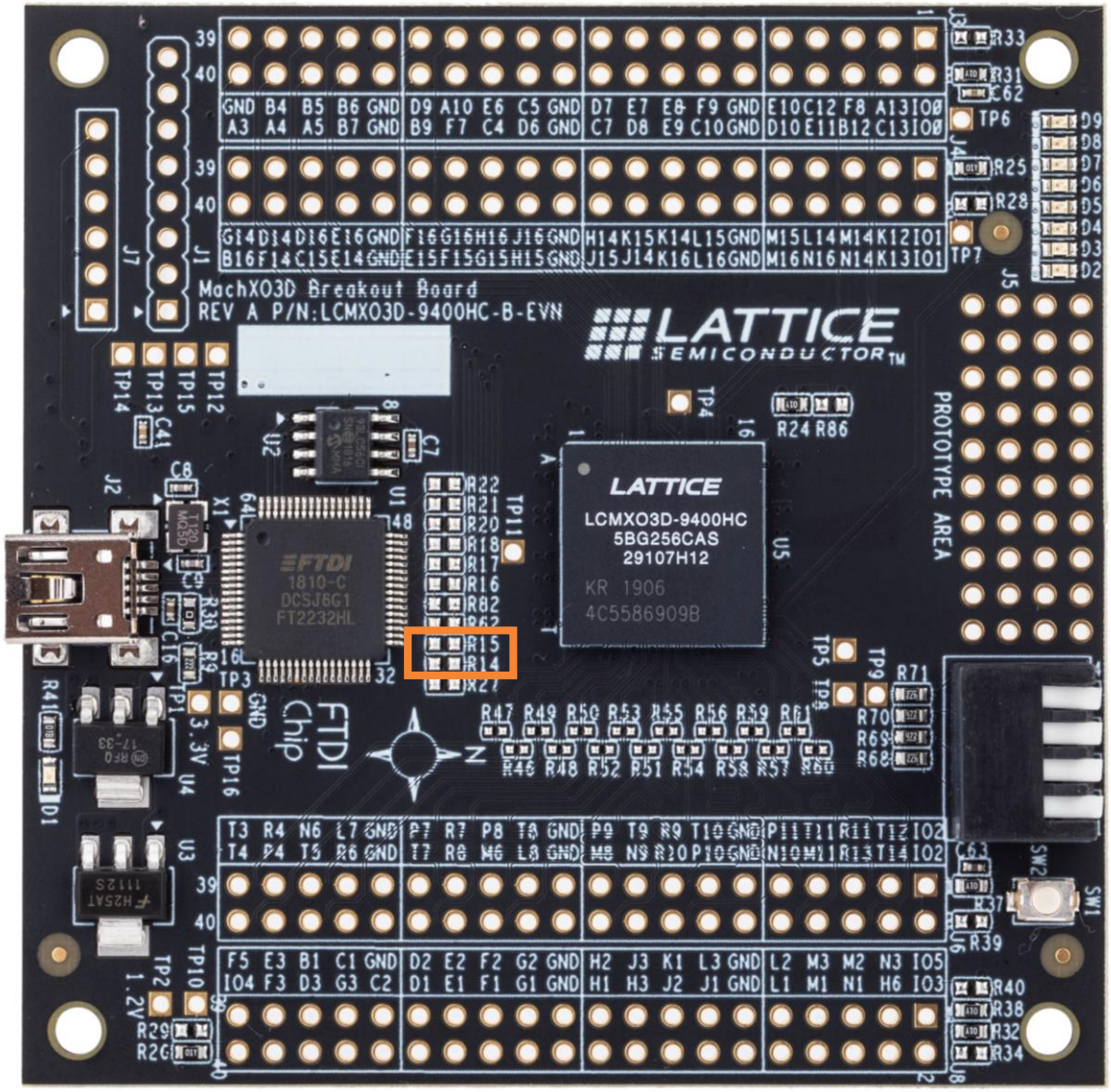


Figure 6.1. MachXO3D Breakout Board

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

   Connect the MachXO3D Breakout Board to the host PC and power on the board.

   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 6.2.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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

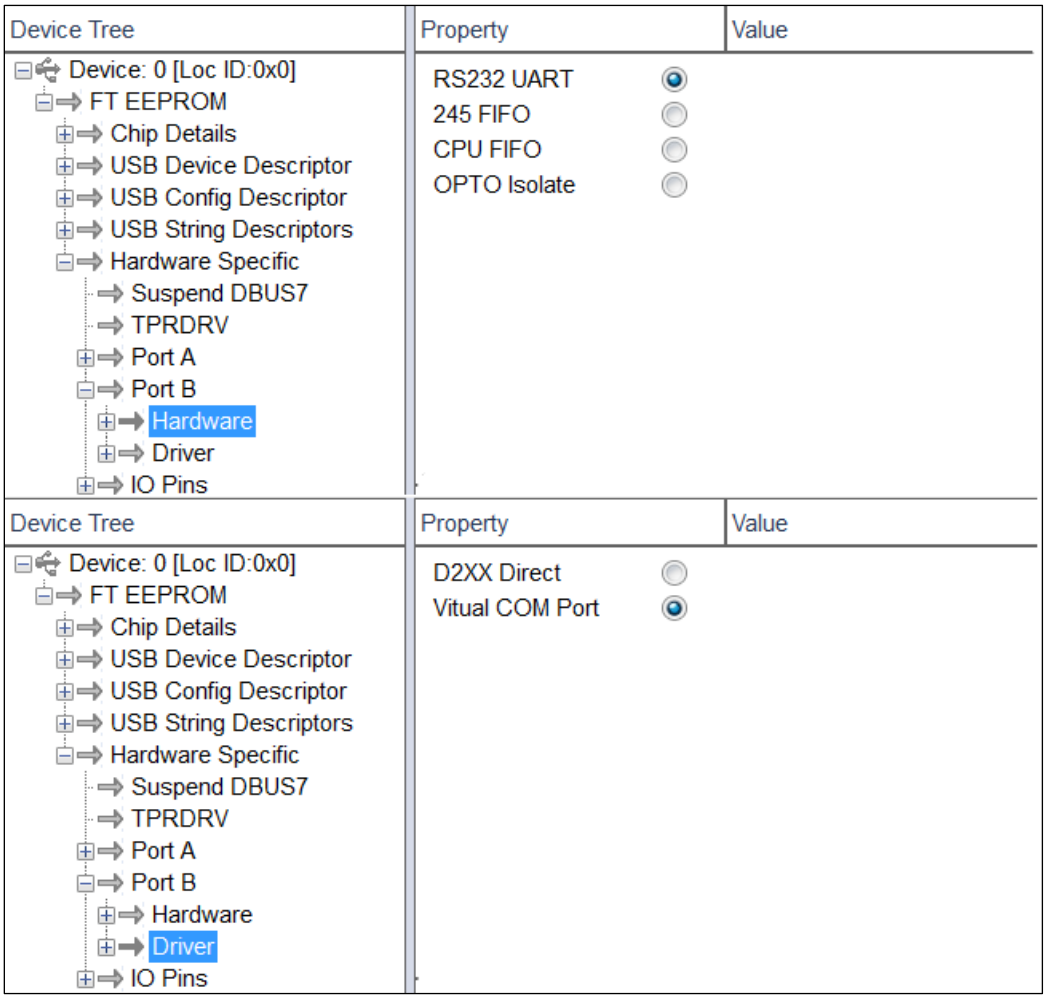


Figure 6.2. Configuring the FTDI Device

3. All the reworks for the MachXO3D Breakout Board are completed.

6.1. Creating an SoC Design Project and Preparing Hardware Design

The MachXO3D Breakout Board includes a minimal volume FPGA, which cannot support an RX SoC project. Hence, an MC SoC project is created in this tutorial.

We need to launch Lattice Propel Builder to create the MC SoC project.

Follow steps below for a simple SoC creating flow. If you encounter errors, refer to Lattice Propel Builder 2025.1 User Guide (FPGA-UG-02235) if meet errors.

Launch Lattice Propel Builder:

1. Choose **File** > **New SoC Design**. set **Design Information** or leave it as default (Figure 6.3).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

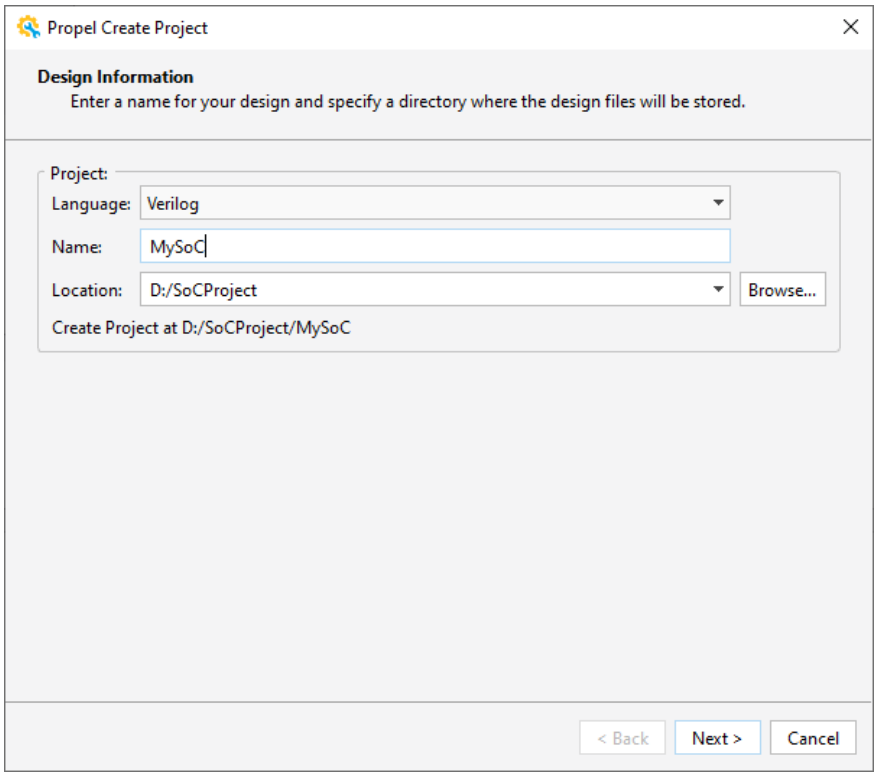


Figure 6.3. Design Information Settings

2. Click **Next**. Choose **Scalable RISC-V SoC Project** (Figure 6.4).

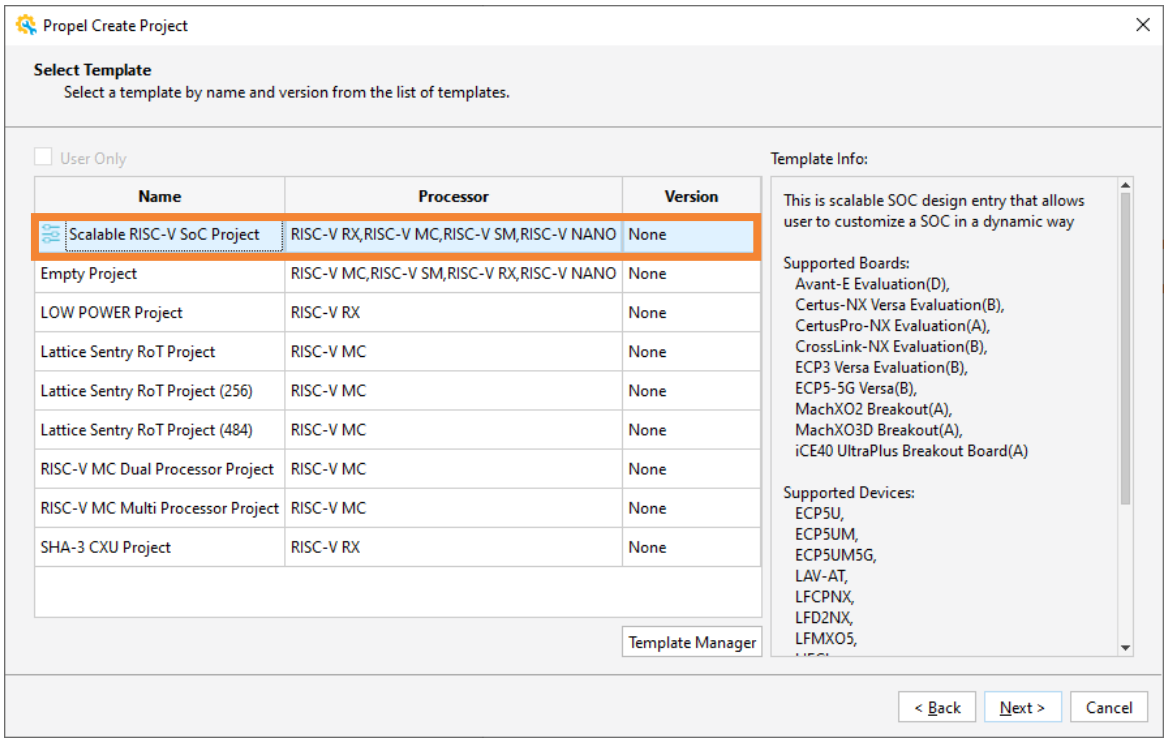


Figure 6.4. Select Template Page

3. Click **Next**. Choose **General Micro Controller RISC-V MC** (Figure 6.5).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

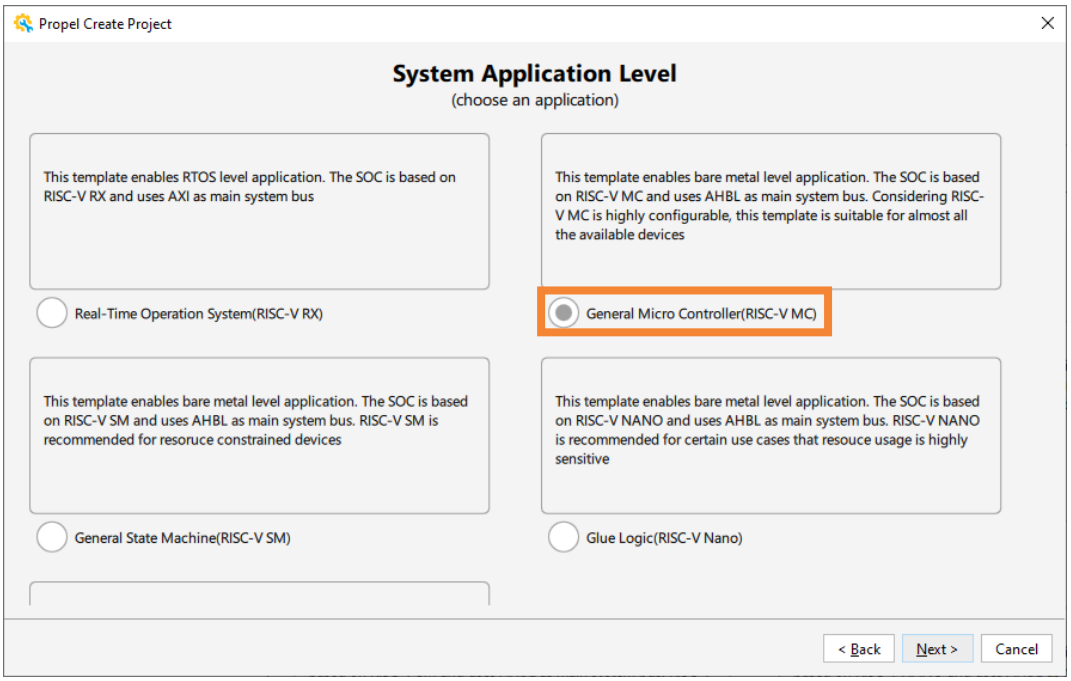


Figure 6.5. Selecting General Micro Controller RISC-V MC for System Application Level

4. Click **Next**. Use the **Board** filter and select MachXO3D Breakout (Figure 6.6).

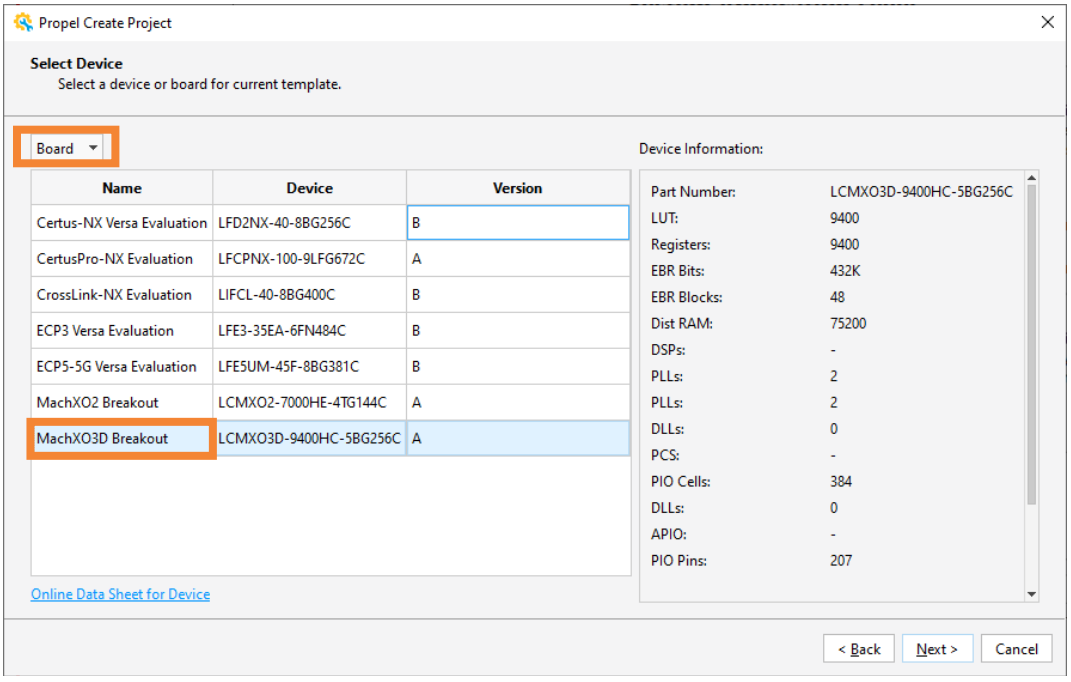


Figure 6.6. Selecting MachXO3D Breakout Board

5. Click **Next** and click **Confirm** (Figure 6.7).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

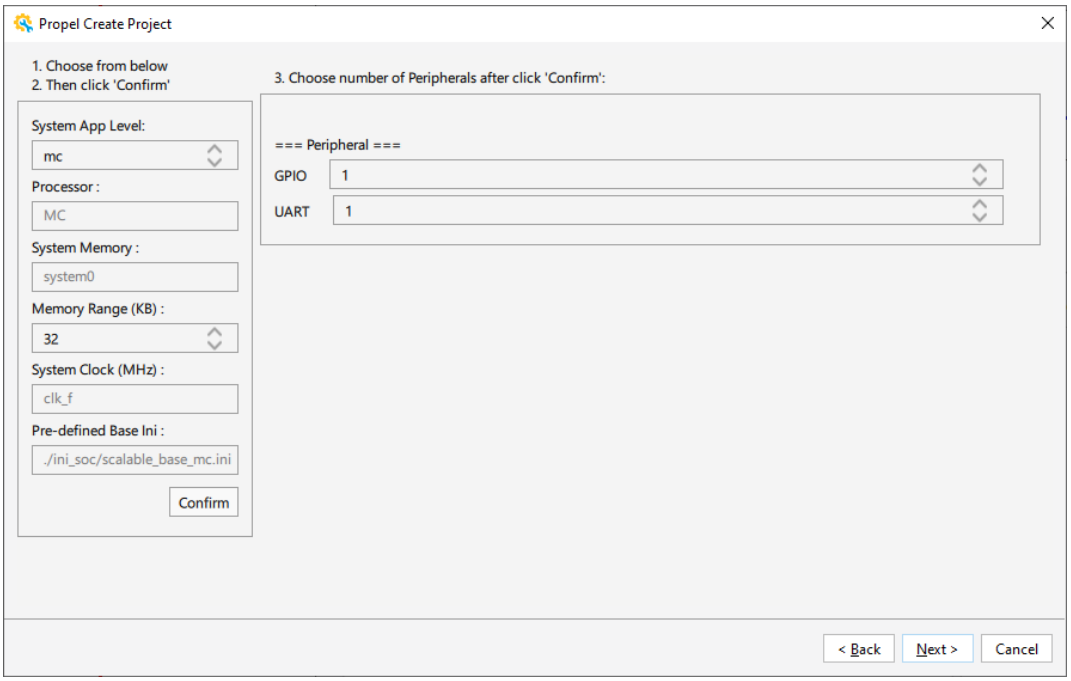


Figure 6.7. Confirm Page

6. Click **Next** on the architecture page (Figure 6.8).

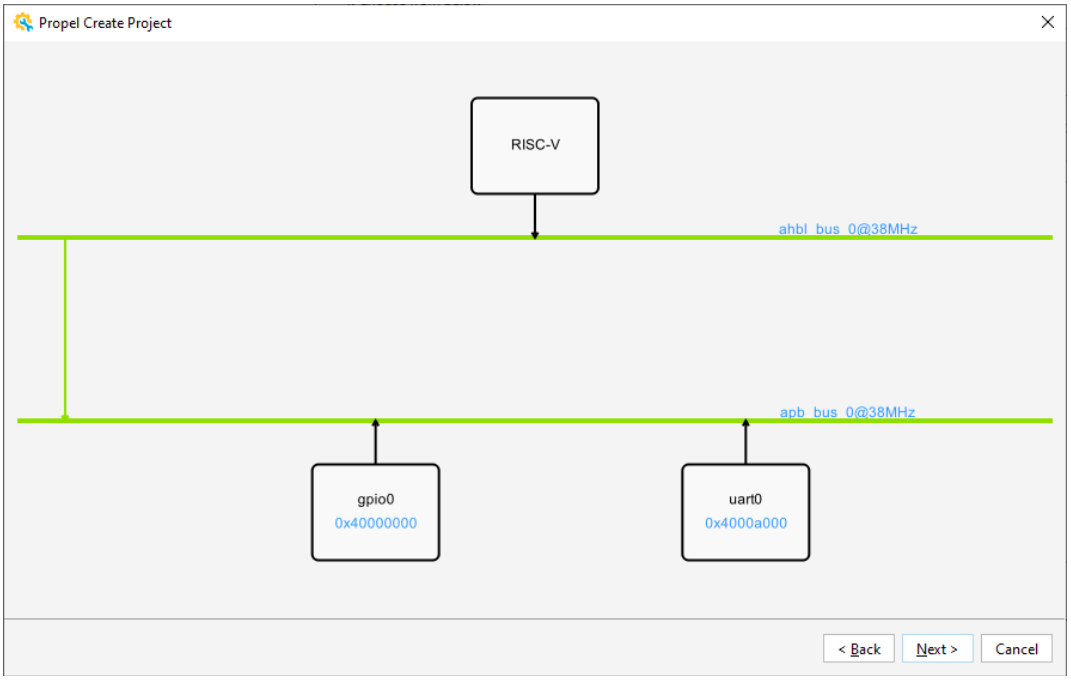


Figure 6.8. Architecture Page

7. Click **Next** to view the project information (Figure 6.9).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

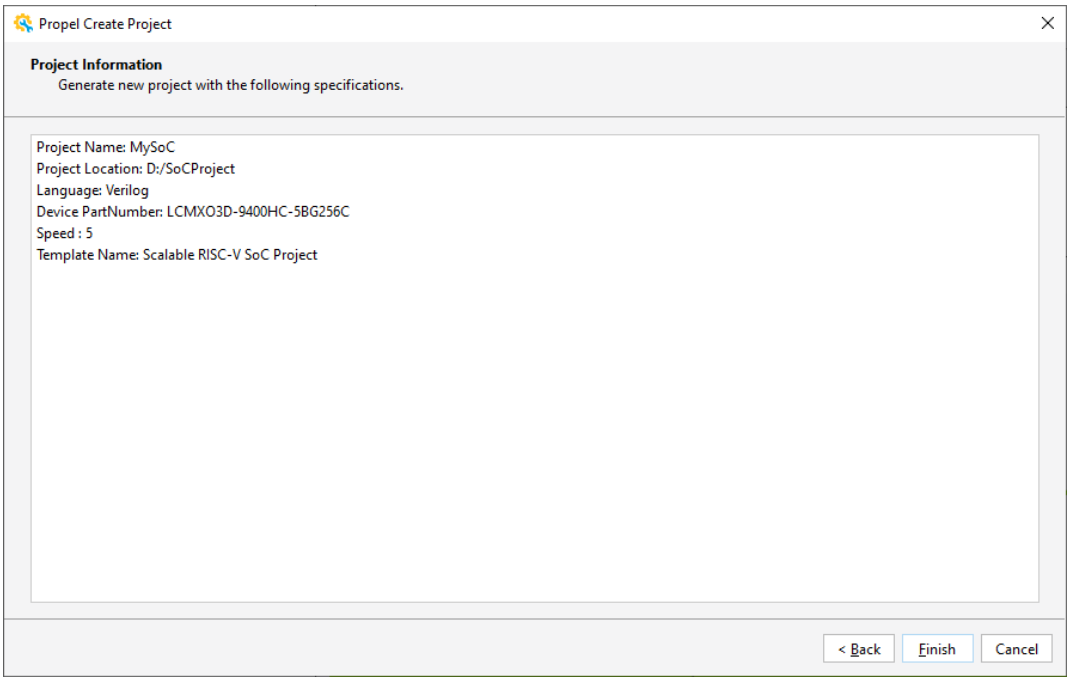


Figure 6.9. Project Information

8. Click **Finish.** An SoC project is created, as shown in Figure 6.10.

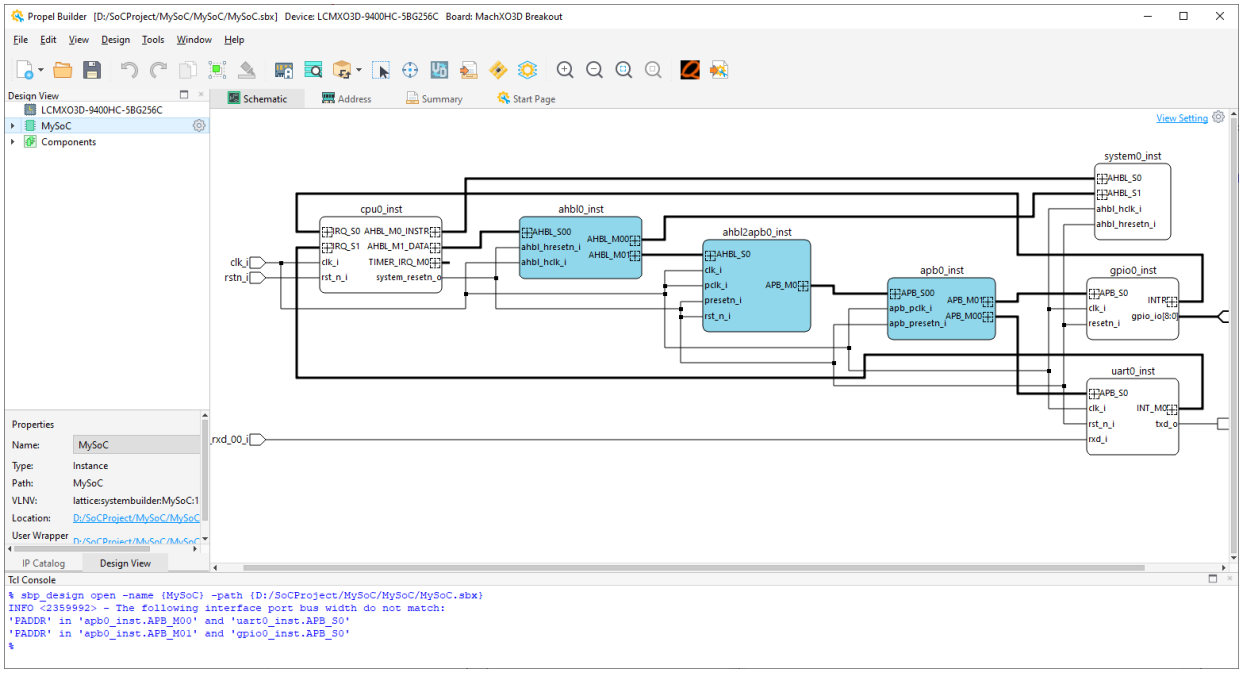


Figure 6.10. Project Workbench

6.2. Creating a Hello World C Project

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

1. In SoC project workbench (Figure 6.10), Choose **Design** > **Generate** > **Generate** to create C project requirement files. Choose **Design** > **Run Propel** to switch to Lattice Propel SDK.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. In Propel SDK, set the workspace. In the C/C++ project creating page (Figure 6.11), select **Hello World Project**.
3. Click **Next**, Click **Finish**. The C project is created and displayed on the workbench.
4. Choose **Project** > **Build Project**.
5. Check the build result from the **Console** view (Figure 6.12).

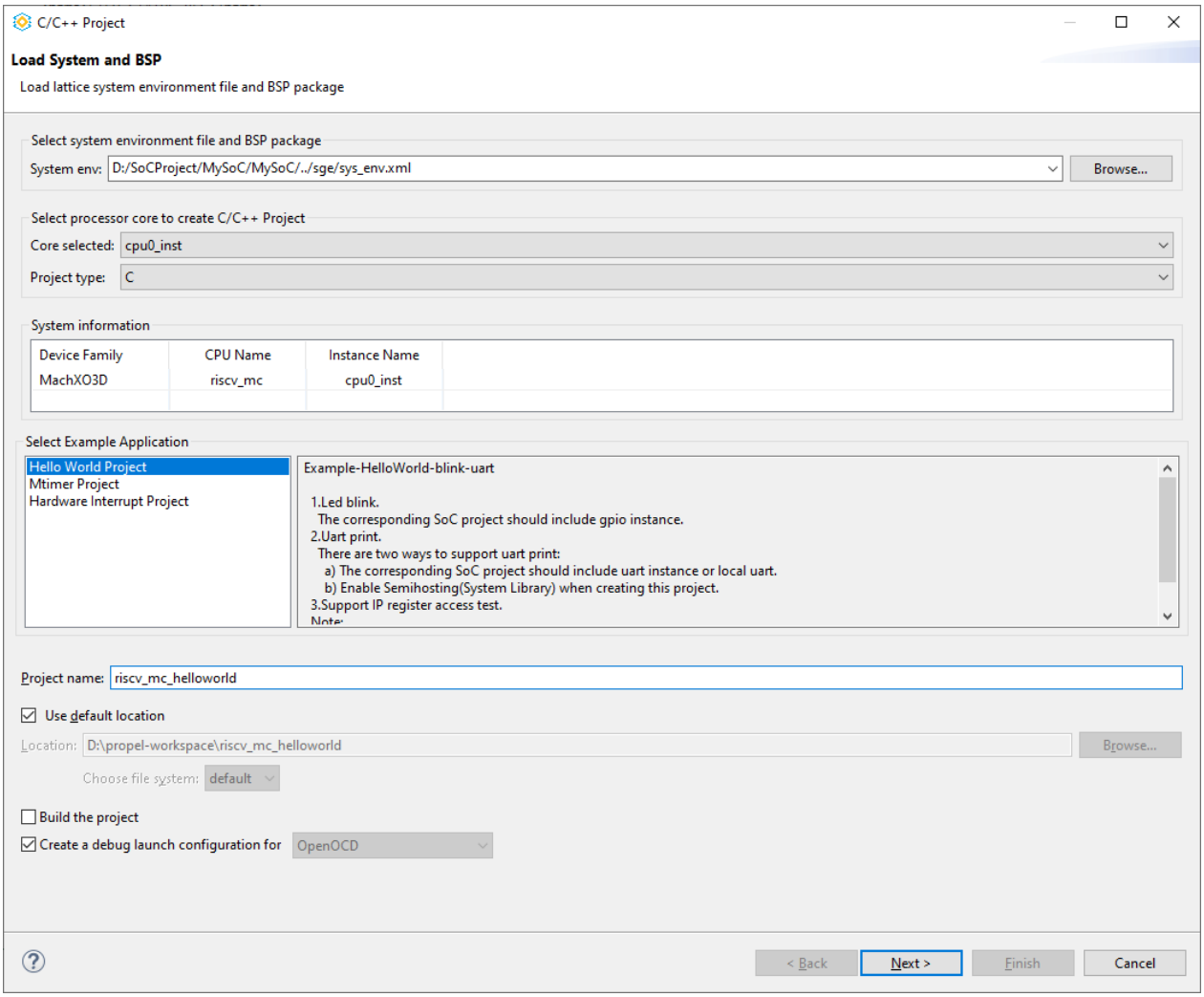


Figure 6.11. Creating a C/C++ Project

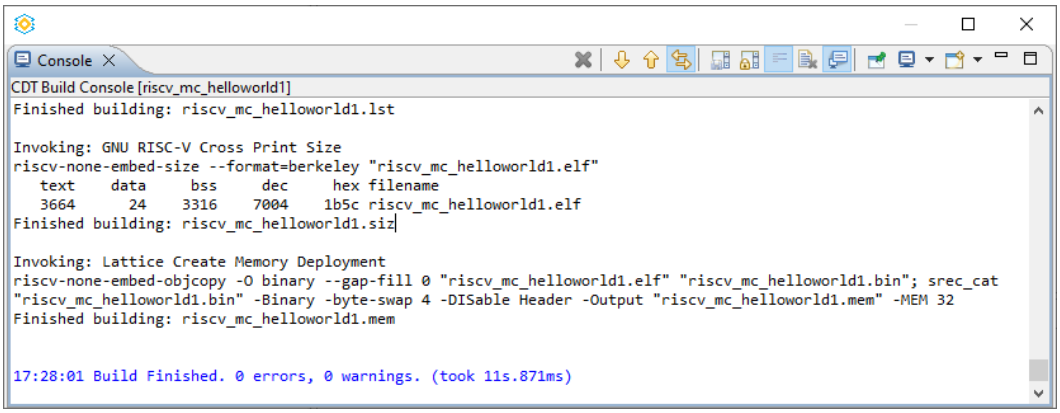


Figure 6.12. Build Result of HelloWorld C Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

6.3. Memory Initialization (Optional)

In the SoC project workbench (Figure 6.10), update the SoC design to set the preloaded software. Enable the **Initialize Memory** checkbox for the System Memory IP instance from the **Initialization** area of the **General** tab. Set **Initialization File** (Figure 6.13) generated from the corresponding Hello World C project in the Creating a Hello World C Project section. Click **Generate** (Figure 6.13).

In the SoC project workbench (Figure 6.10), choose **Design** > **Generate** > **Generate**.

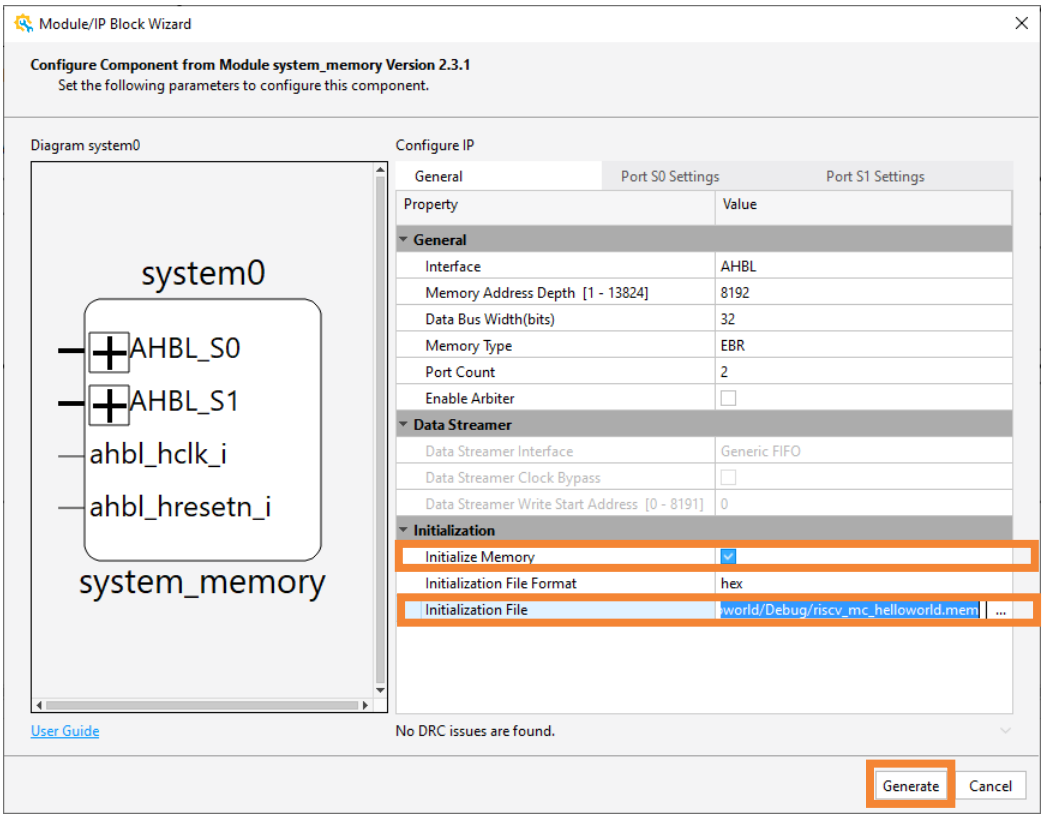


Figure 6.13. Configuring System Memory Module 2

6.4. Launching Lattice Diamond Software

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

1. In the SoC project workbench (Figure 6.10), choose **Design** > **Run Diamond**. A Lattice Diamond project is created and opened automatically in the Lattice Diamond software.
2. Switch to the **Process** view of the Lattice Diamond project and make sure Bitstream File or JEDEC File is checked in the **Export Files** section (Figure 6.14) .
3. Choose **Process** > **Run**. Wait for generating the programming file successfully. You can see a green checkmark before each successfully completed process.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

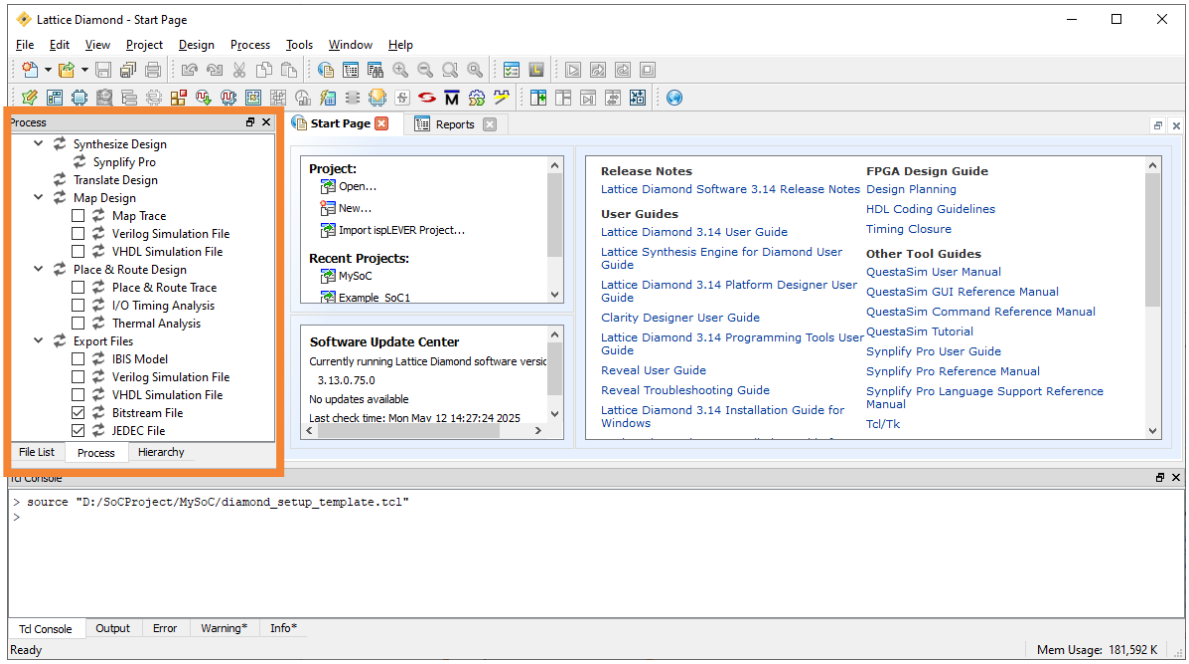


Figure 6.14. Generating the Programming File

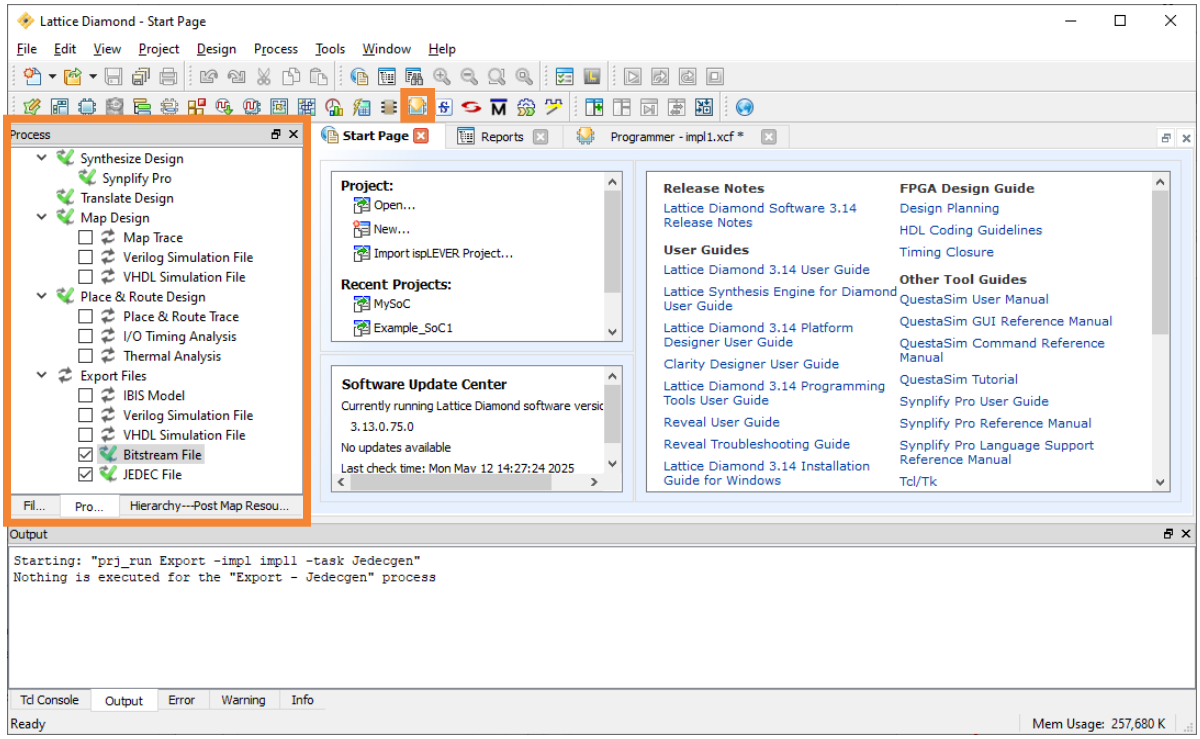


Figure 6.15. Generating the Programming File Successfully

6.5. Programming the Target Device

Once the programming file is exported successfully in the last section (Figure 6.15), you can 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** (Figure 6.15).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. The **Programmer: Getting Started** dialog pops up (Figure 6.16). Click **OK**.

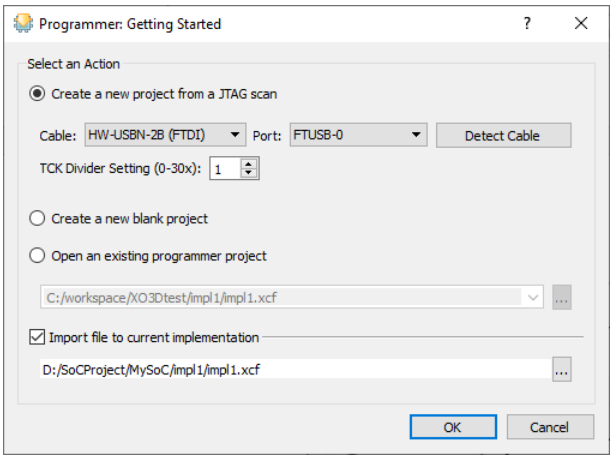


Figure 6.16. Programmer Getting Started Dialog

3. Review the **Device Family**, **Device**, **Operation**, and **File Name** in the Programmer window (Figure 6.17).

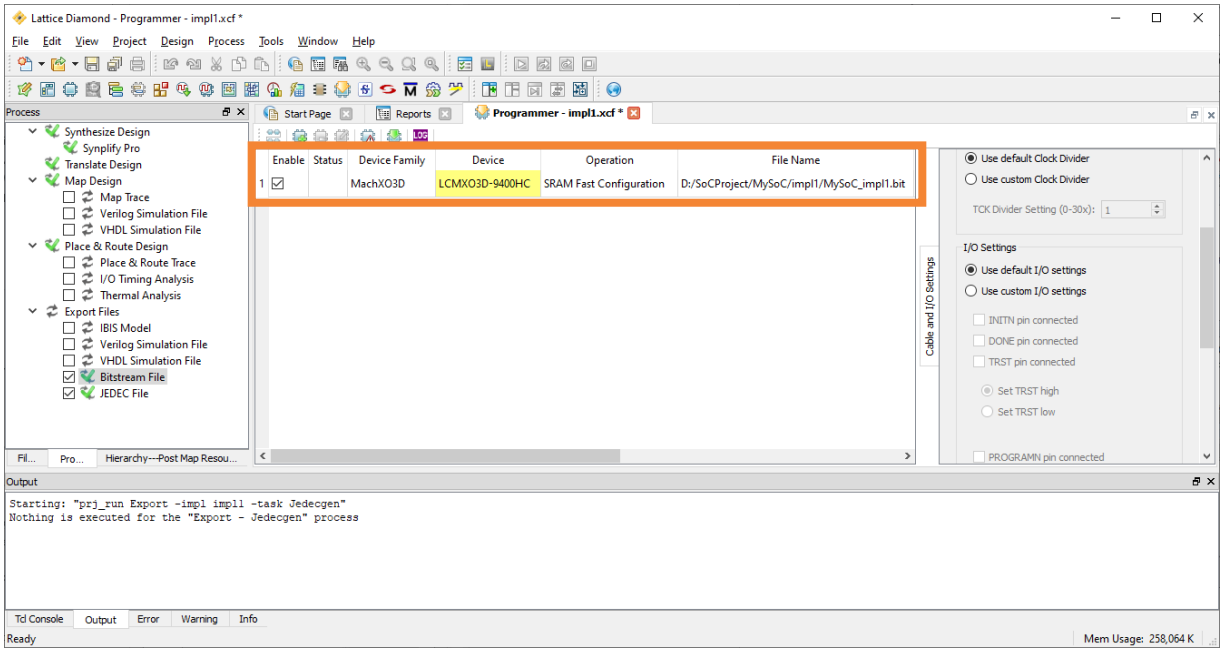


Figure 6.17. Programmer Window

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

6.6. Running Demo on the MachXO3D Breakout Board

1. Switch back to Lattice Propel SDK, the C project workbench.
2. In the **Project Explorer** view, select the C project, riscv_mc_helloworld.
3. Choose **Run** > **Debug Configurations…**.
4. Choose riscv_mc_helloworld item (Figure 6.18).
5. Click the **Debug** button.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

6. If hardware reworks on MachXO3D breakout board described in Lattice Propel Tutorial – Hello World are done, you can open the terminal to see the print-out message.

   Refer to Serial Terminal Tool – Windows or Serial Terminal Tool – Linux.

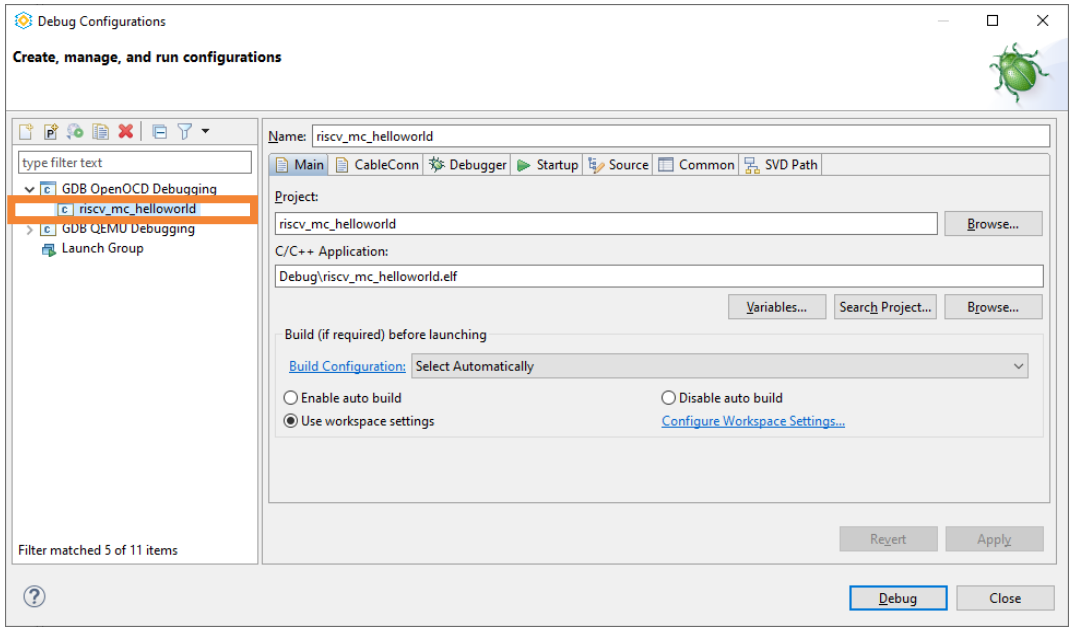


Figure 6.18. Debug Configurations Dialog 2

7. Click the **Resume** icon on the toolbar. The serial terminal output is shown in Figure 6.19.

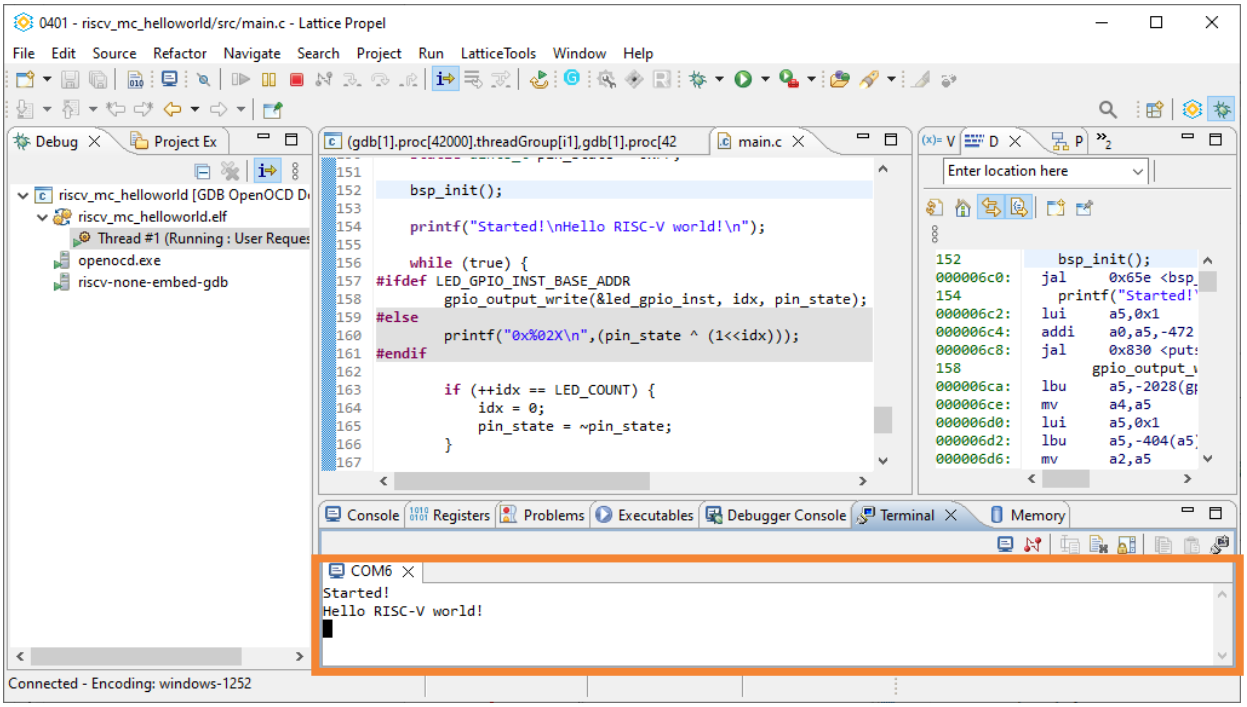


Figure 6.19. Run Result of Hello World Project

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7. 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 or 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 not being enabled at the same time.

7.1. 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 are gradually being 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 the project from Lattice Propel Builder. See Lattice Propel Builder 2025.1 User Guide (FPGA-UG-02235) for more details.

1. Create a SHA-3 CXU SoC project with the CertusPro-NX Evaluation Board (Figure 7.1). 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.

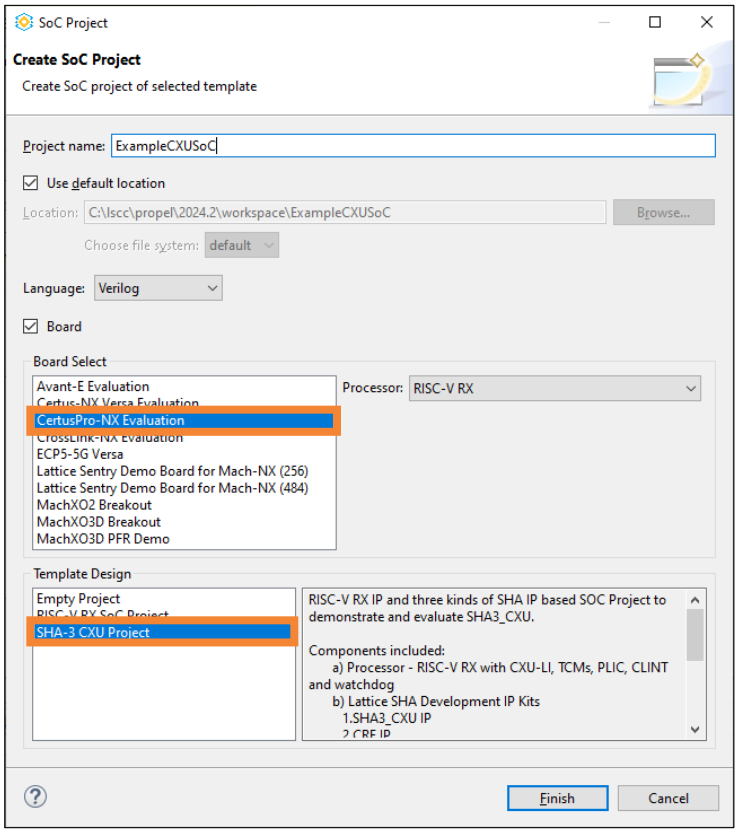


Figure 7.1. Create SoC Project Wizard

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7.2. Creating a CXU C Project

1. The Timing Profiling C project can be created through the **Load System and BSP** page (Figure 7.2).
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**.

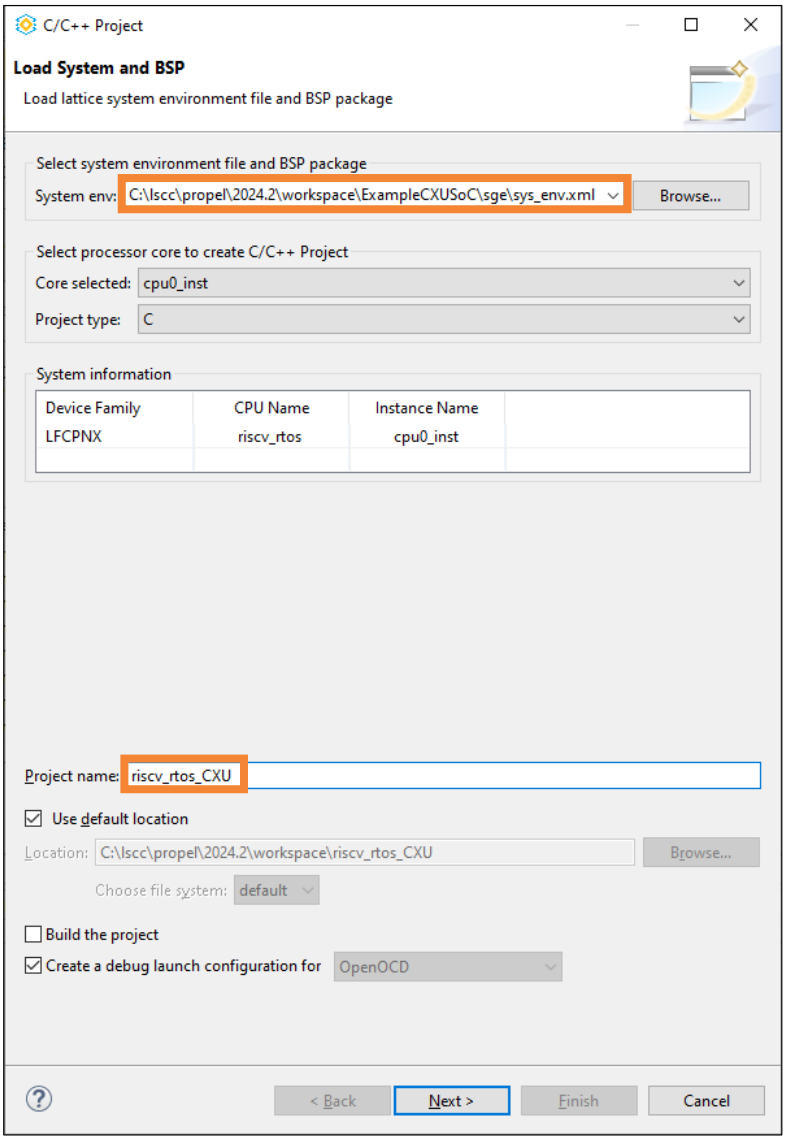


Figure 7.2. Load System and BSP Page 3

7.3. Compiling and Running Demo – CXU Demo

7.3.1. Compiling C Project – CXU Demo

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

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7.3.2. 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 7.3).
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.

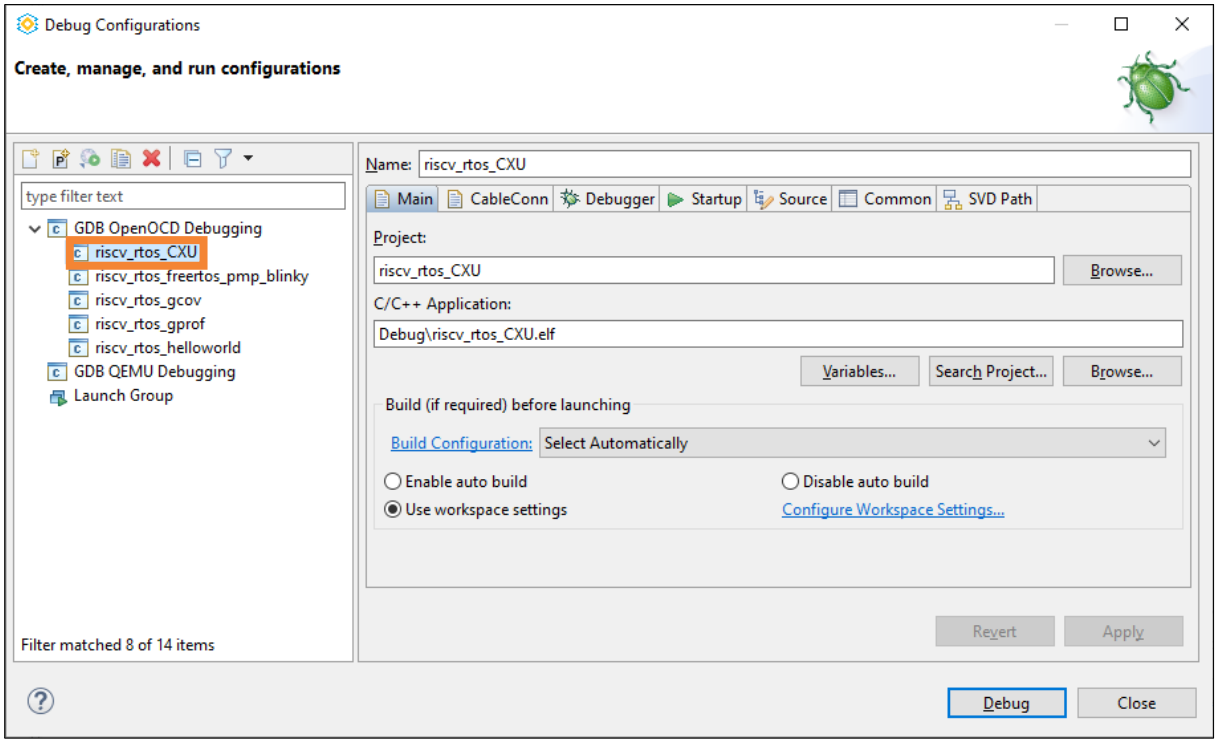


Figure 7.3. Debug Configurations Dialog 3

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**.

   Refer to Serial Terminal Tool – Windows or Serial Terminal Tool – Linux.

   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 7.4).

   Wait for a few minutes. You can see the performance result logs in the terminal (Figure 7.4).
10. Select **Run** > **Terminate** to stop running.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

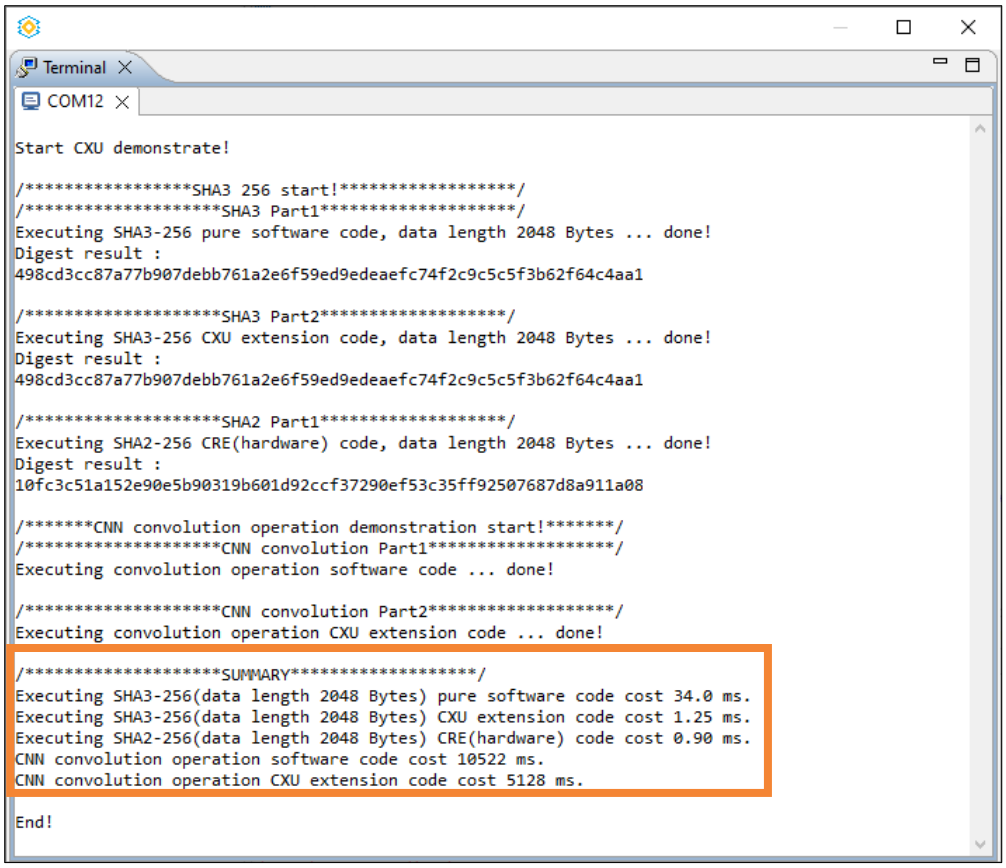


Figure 7.4. Terminal Logs

7.4. Using the Timing Profiling Function

For the details of timing profiling function, you can refer to the How to Add Timing Profiling Function to an Existing C Project 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 5.31).
4. Select **GNU RISC-V Cross C Compiler** > **Preprocessor**. Add the defined symbol, LSCC_GPROF (Figure 5.32).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgprof (Figure 5.33).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags under the label, **Other linker flags**. Make sure the linker flag, --oslib=semihost, is supported (Figure 5.34).
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 7.5).
9. Double-click the file gmon.out to display the gprof Viewer. Refer to the Timing Profiling Project section.
10. Parse the gprof report (Figure 7.6).
11. Restore the settings in step 3 to step 6.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

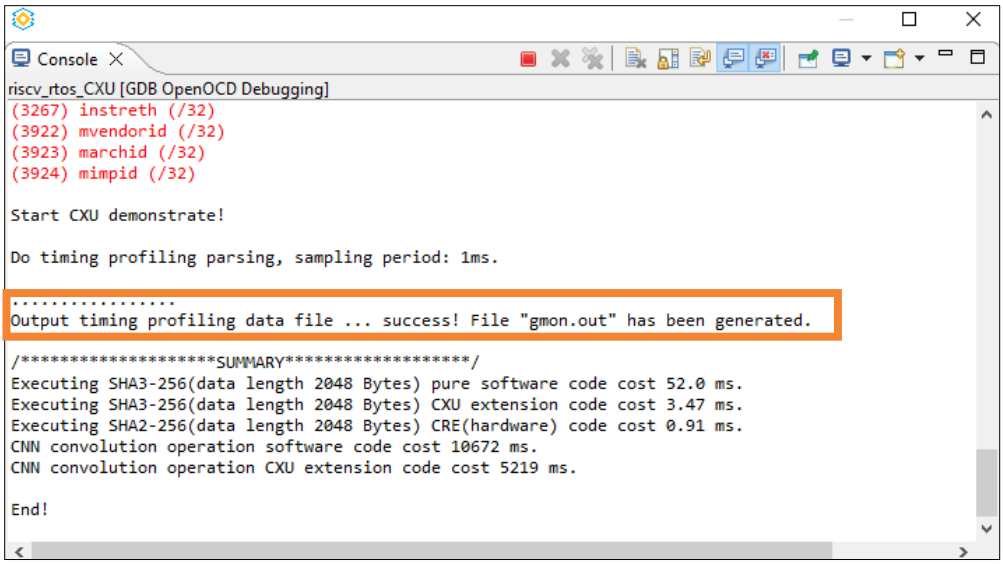
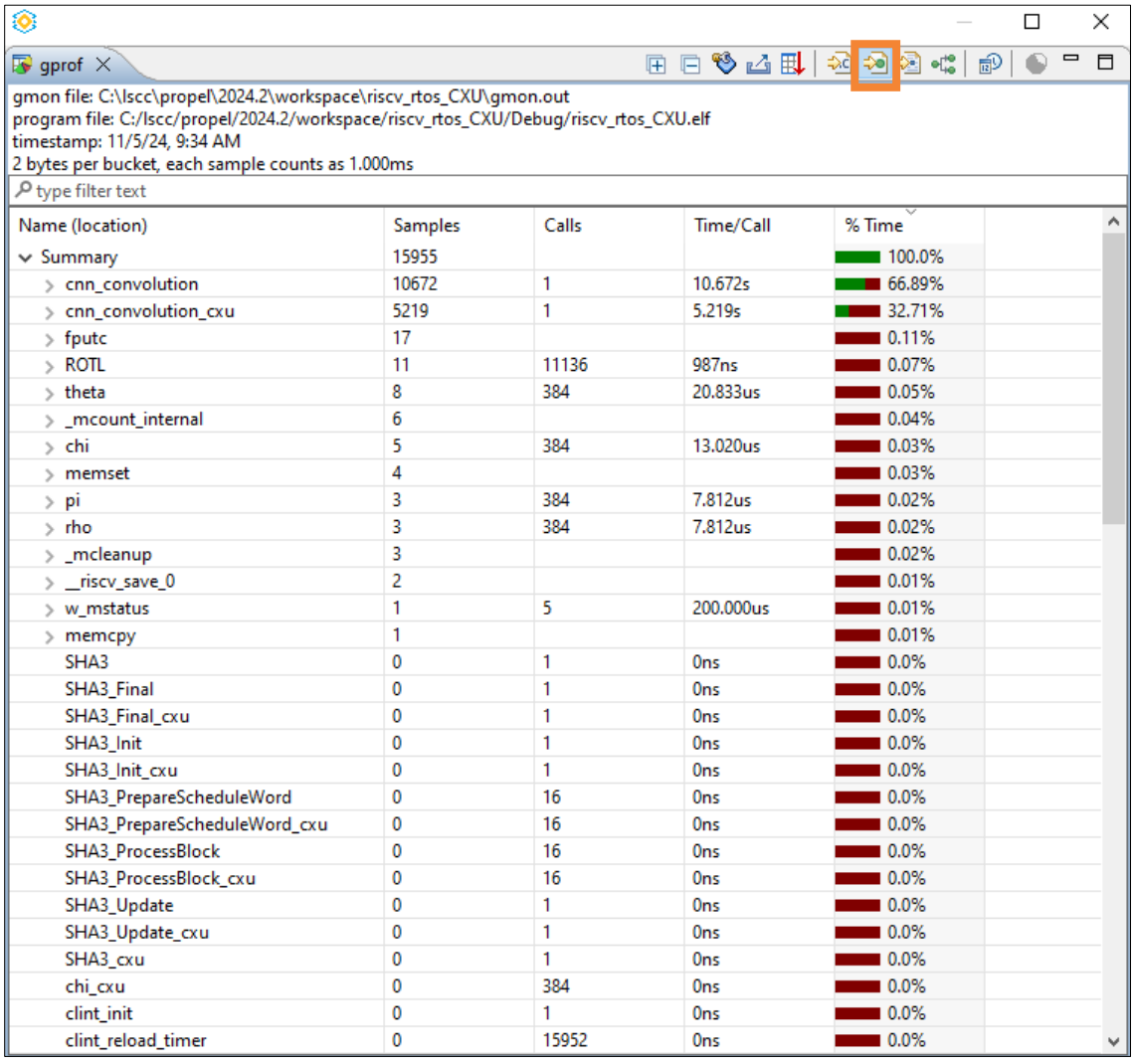


Figure 7.5. Console Logs 1



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 7.6. gprof Viewer

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

7.5. Using the Code Coverage Function

For the details of the timing profiling function, refer to section How to Add Code Coverage Function to an Existing C Project.

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 5.23).
4. Select **GNU RISC-V Cross C Compiler** > **Miscellaneous**. Add the compiler flag, -fprofile-arcs -ftest-coverage (Figure 5.24).
5. Select **GNU RISC-V Cross C Linker** > **Libraries**. Add the library, smallgcov (Figure 5.25).
6. Select **GNU RISC-V Cross C Linker** > **Miscellaneous**. Check link flags under the label, **Other linker flags**. Make sure the linker flag, --oslib=semihost, is supported (Figure 5.27).
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 7.7).
9. Double-click one of the coverage files to display the gcov Viewer, refer to the Code Coverage Project section.
10. Parse the gcov report (Figure 7.8).
11. Restore settings in step 3 to step 6.

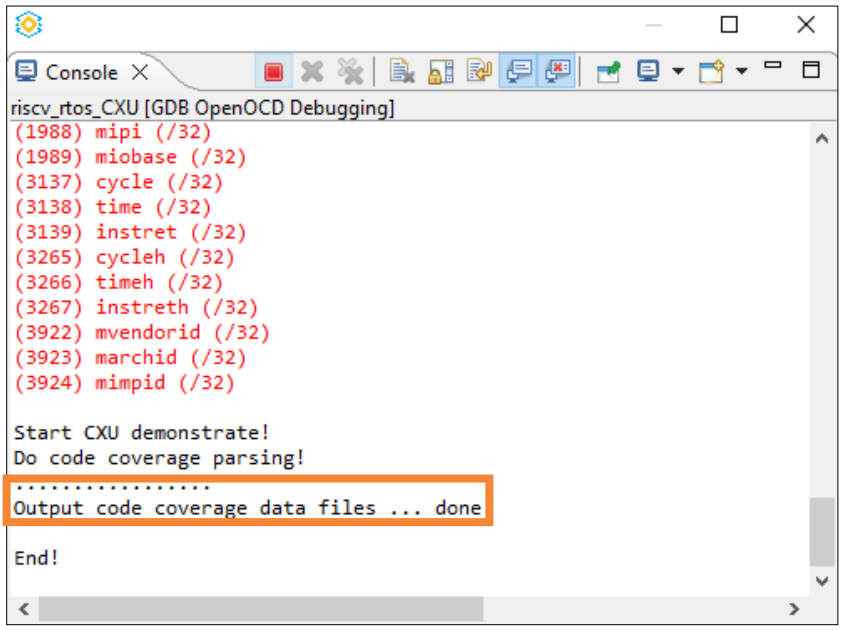


Figure 7.7. Console Logs 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

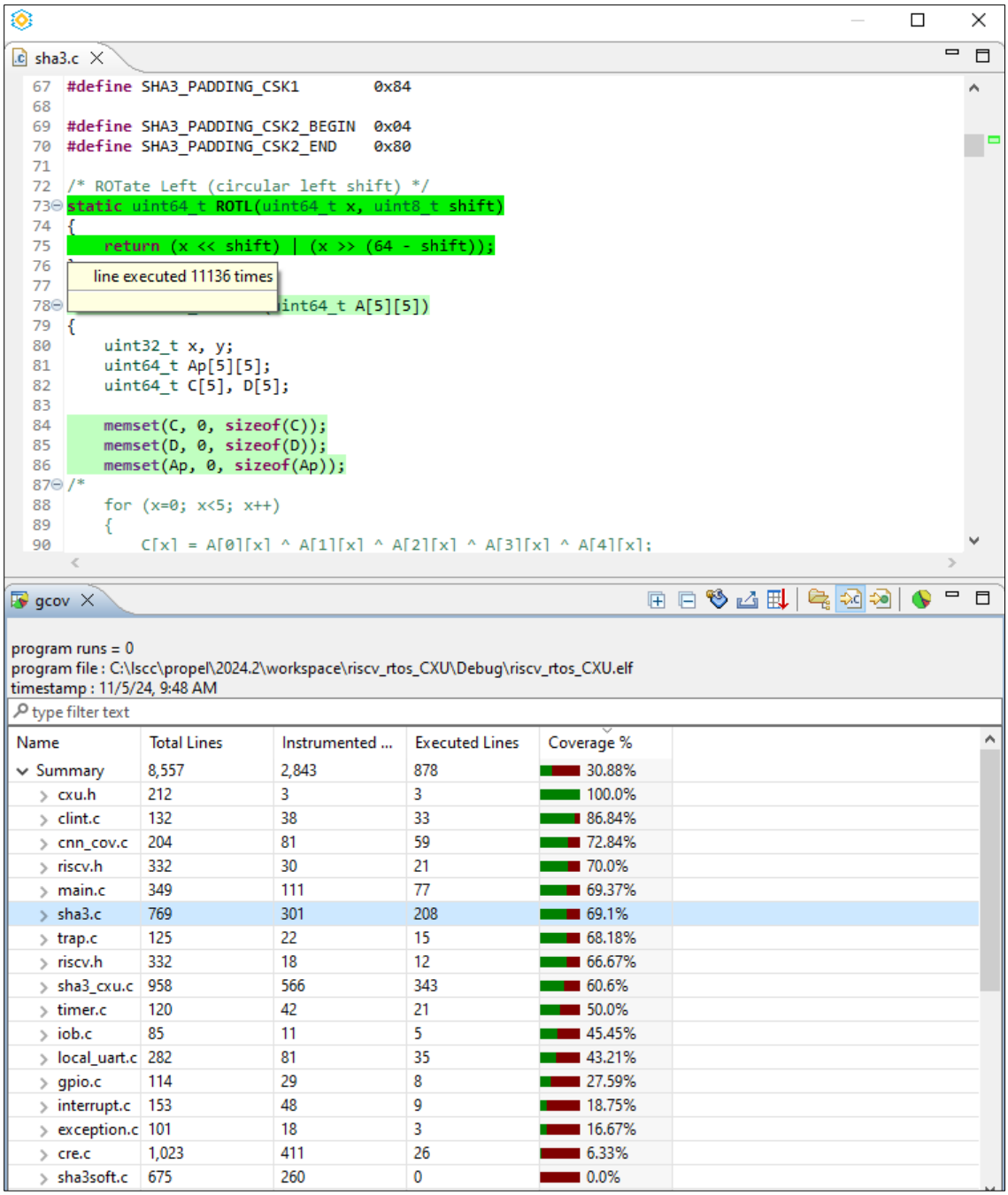


Figure 7.8. gcov Viewer

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

8. Lattice Propel Tutorial – QEMU

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

For the detailed manual of QEMU, refer to QEMU — System Emulation. Lattice Propel SDK includes a RISC-V QEMU simulator with no hardware required and a QEMU_helloworld template for demonstration.

8.1. 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 8.1).

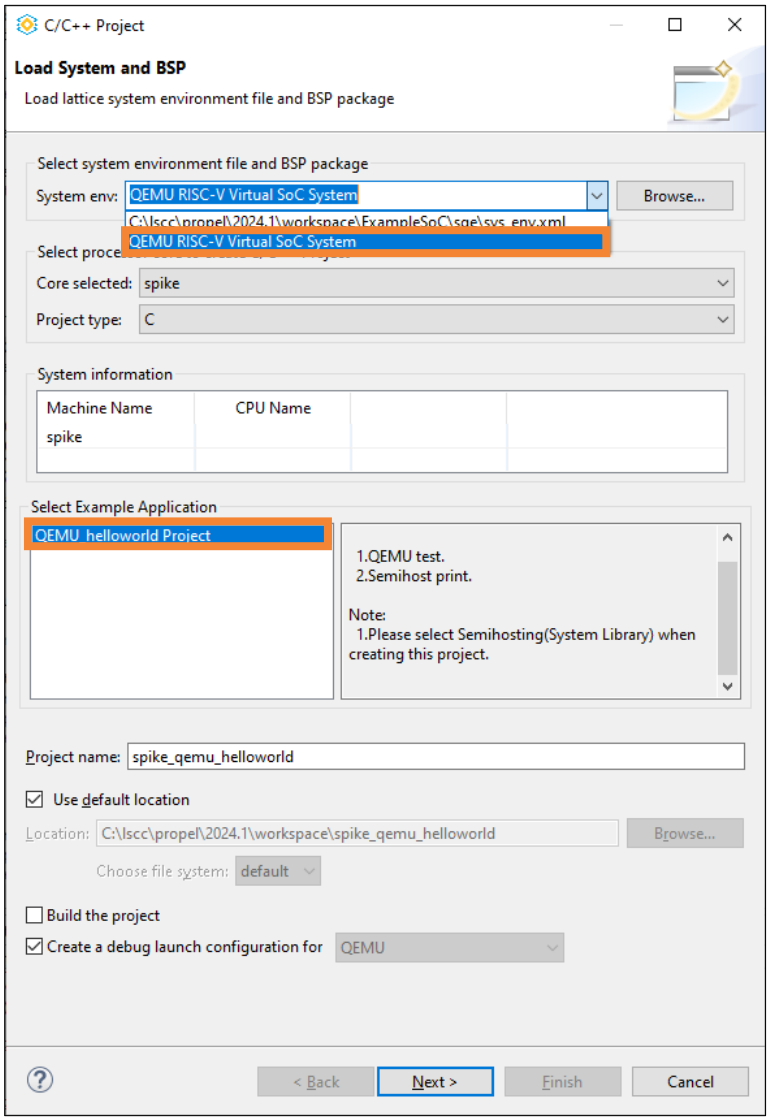


Figure 8.1. Load System and BSP Page 4

2. Select **QEMU RISC-V Virtual SoC System** using the drop-down menu of the **System env** field (Figure 8.1).
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.

   The C project is created and displayed in the workbench.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

5. In the **Project Explorer** view, select the C project just created, spike_qemu_helloworld.
6. Choose **Project** > **Build Project**.
7. Check the build result from the **Console** view (Figure 8.2).

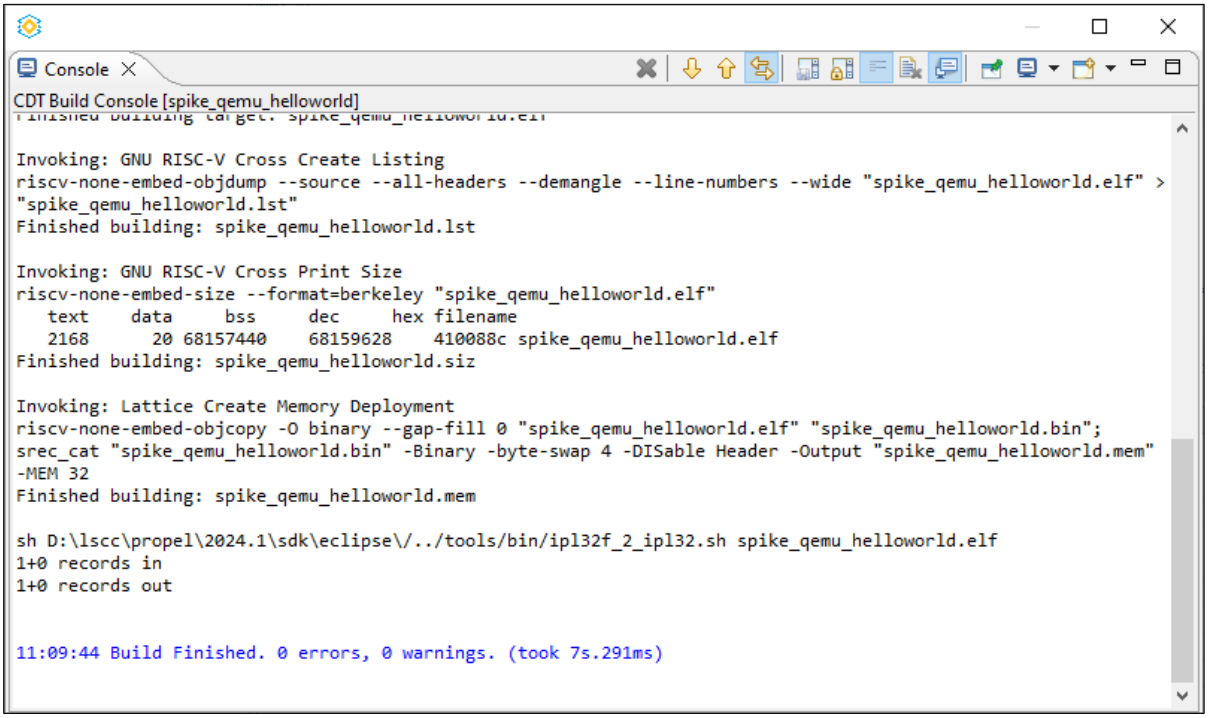


Figure 8.2. Build Console

8.2. Running QEMU C Project

1. In the **Project Explorer** view, select the C project just created, spike_qemu_helloworld.
1. Choose **Run** > **Debug Configurations…**.
2. Choose spike_qemu_helloworld in **GDB QEMU Debugging** (Figure 8.3).
3. 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.
4. Click the **Resume** icon on the toolbar. The Console outputs running logs (Figure 8.4).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

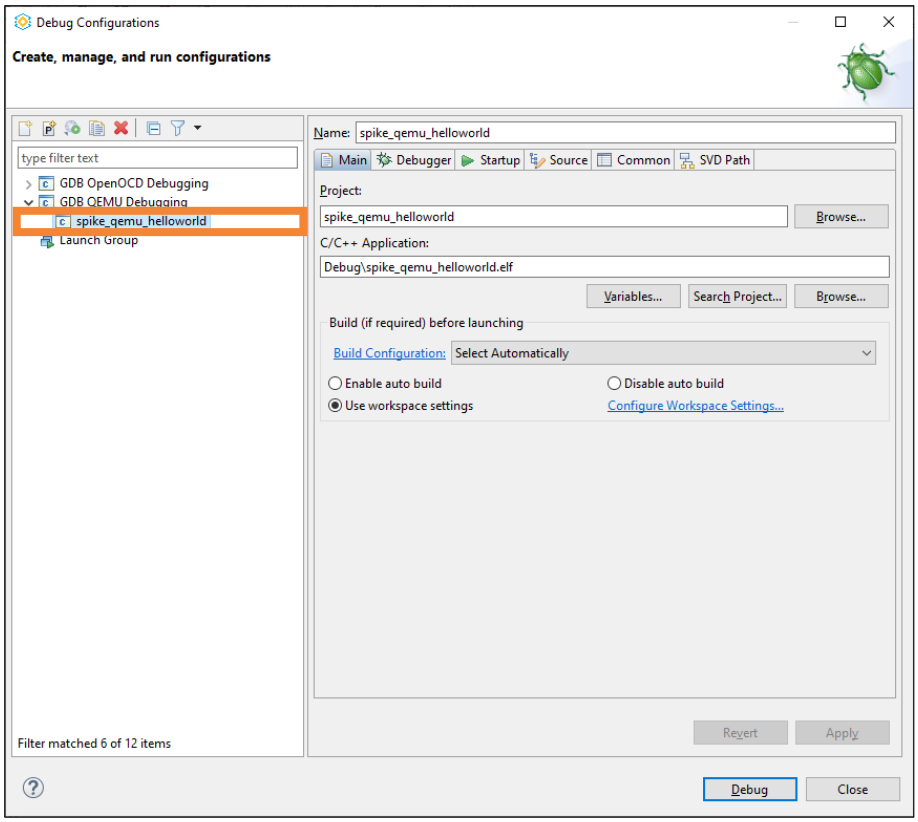


Figure 8.3. Debug Configurations Dialog 4

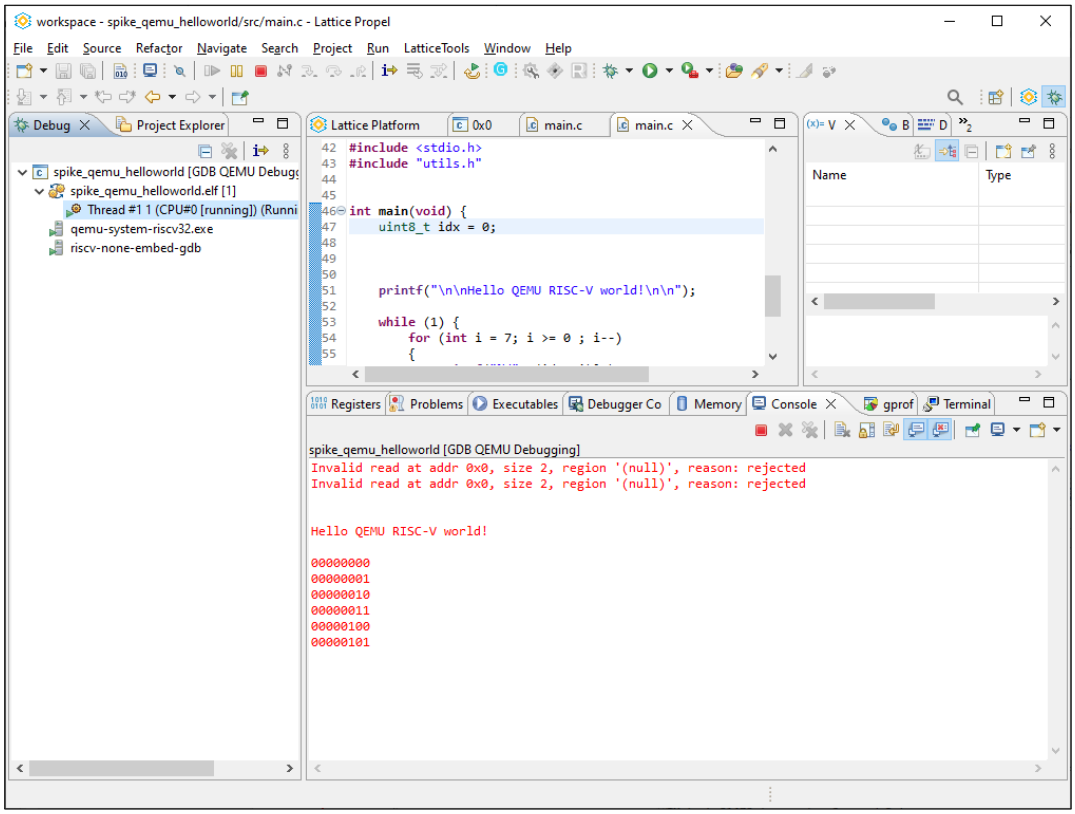


Figure 8.4. QUMU C Project Running Windows

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

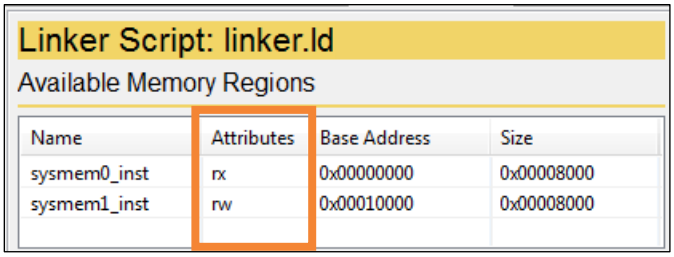


Figure A.1. Memory Regions in Linker Script

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.

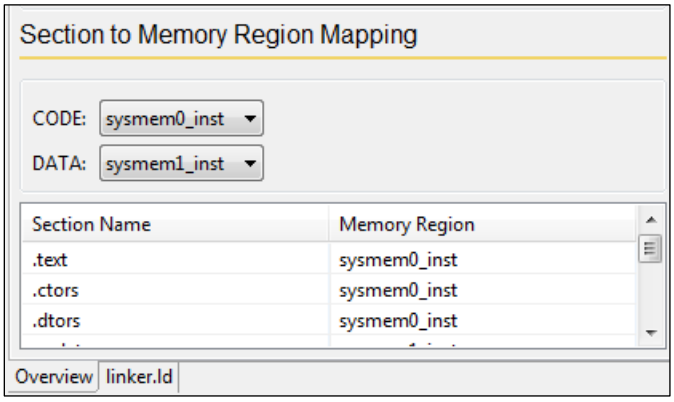


Figure A.2. Section to Memory Region Mapping

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

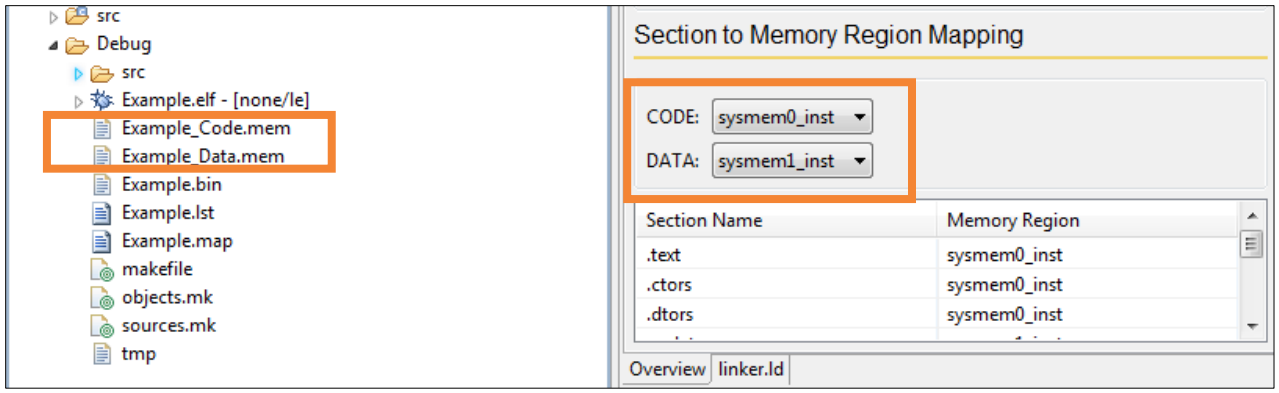


Figure A.3. Linker Script and Generated Memory Files

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 the **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**.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

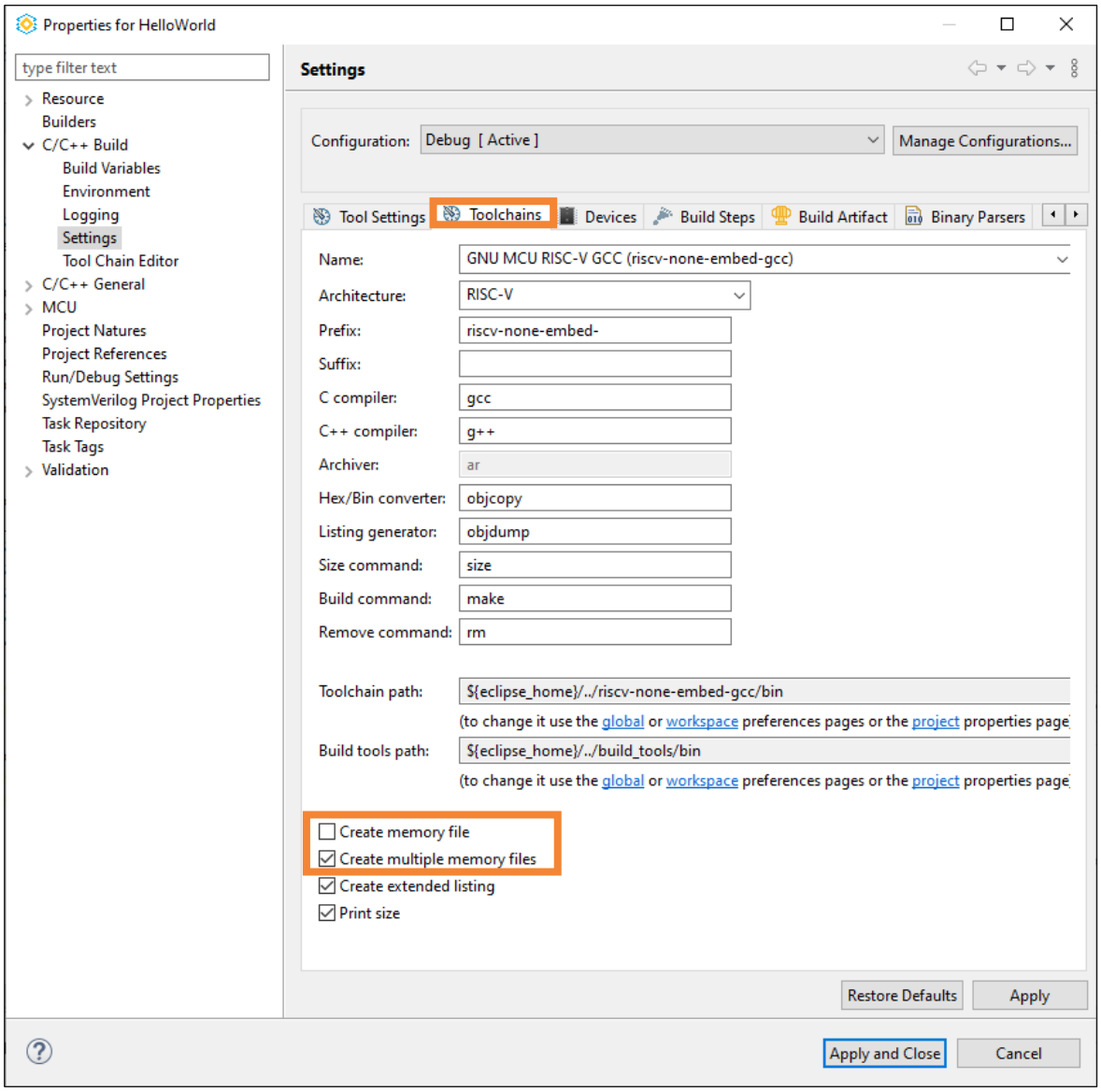


Figure A.4. Toolchains Tab of C/C++ Build Settings

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

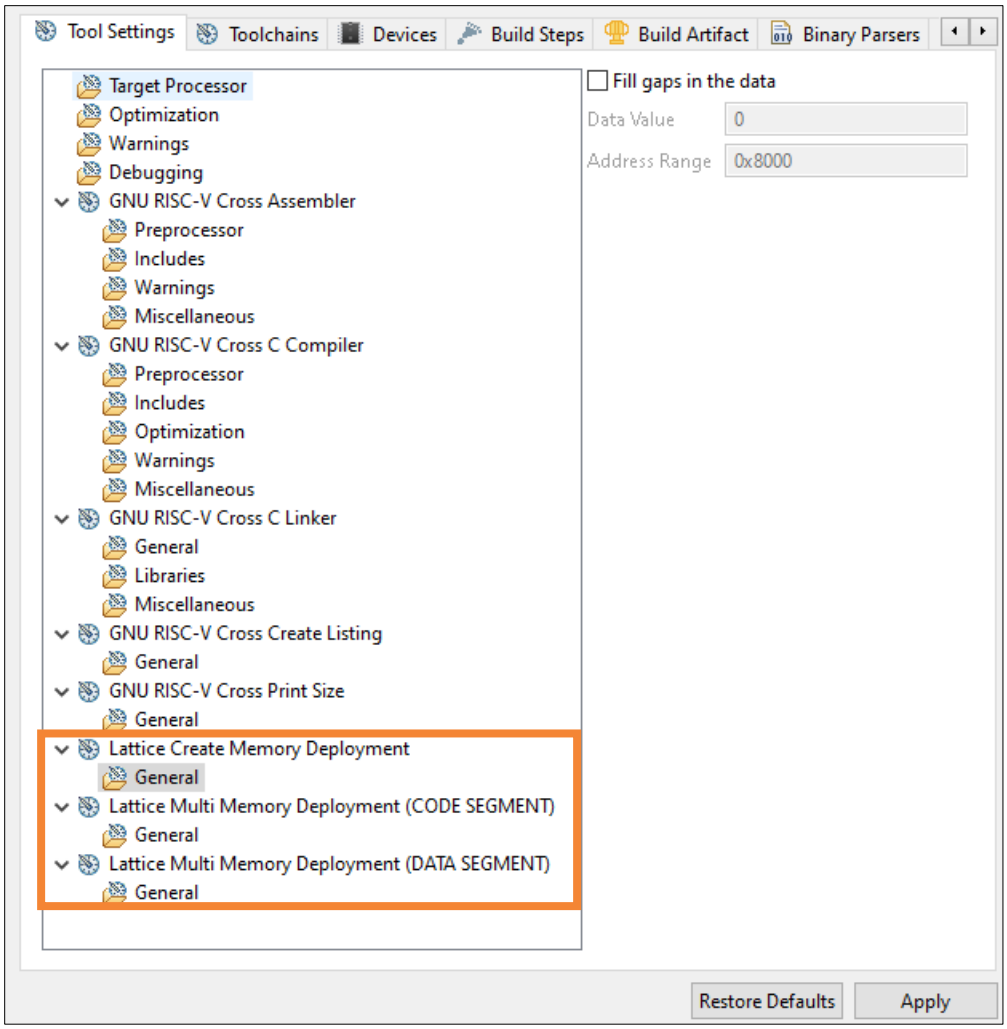


Figure A.5. Tool Settings Tab of C/C++ Build Settings

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.

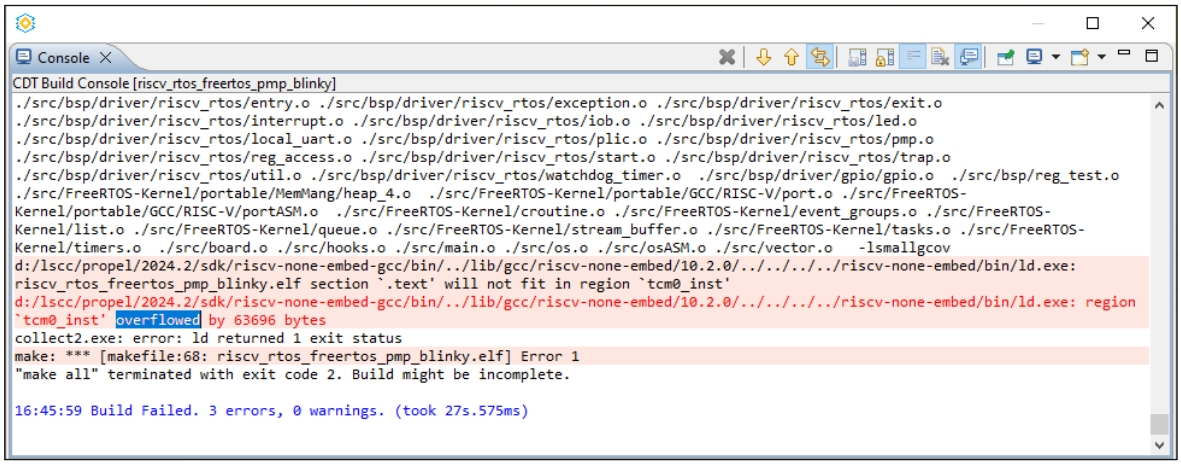


Figure A.6. Build Project Console 1

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

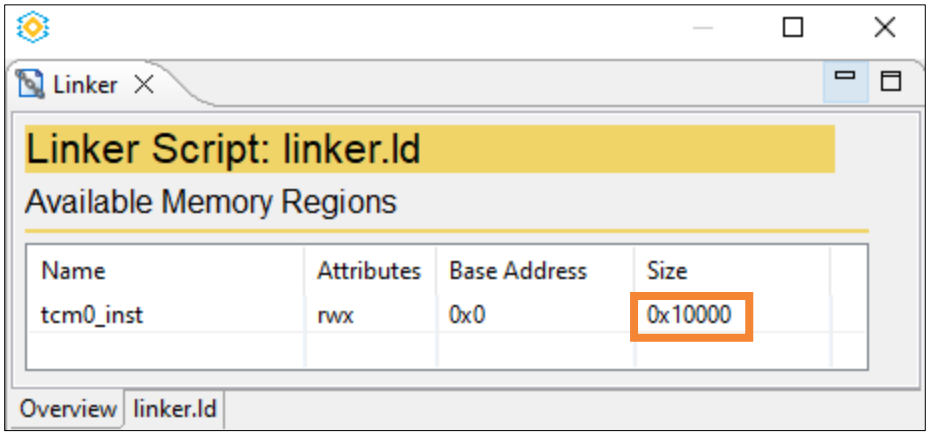


Figure A.7. Linker Script

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).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

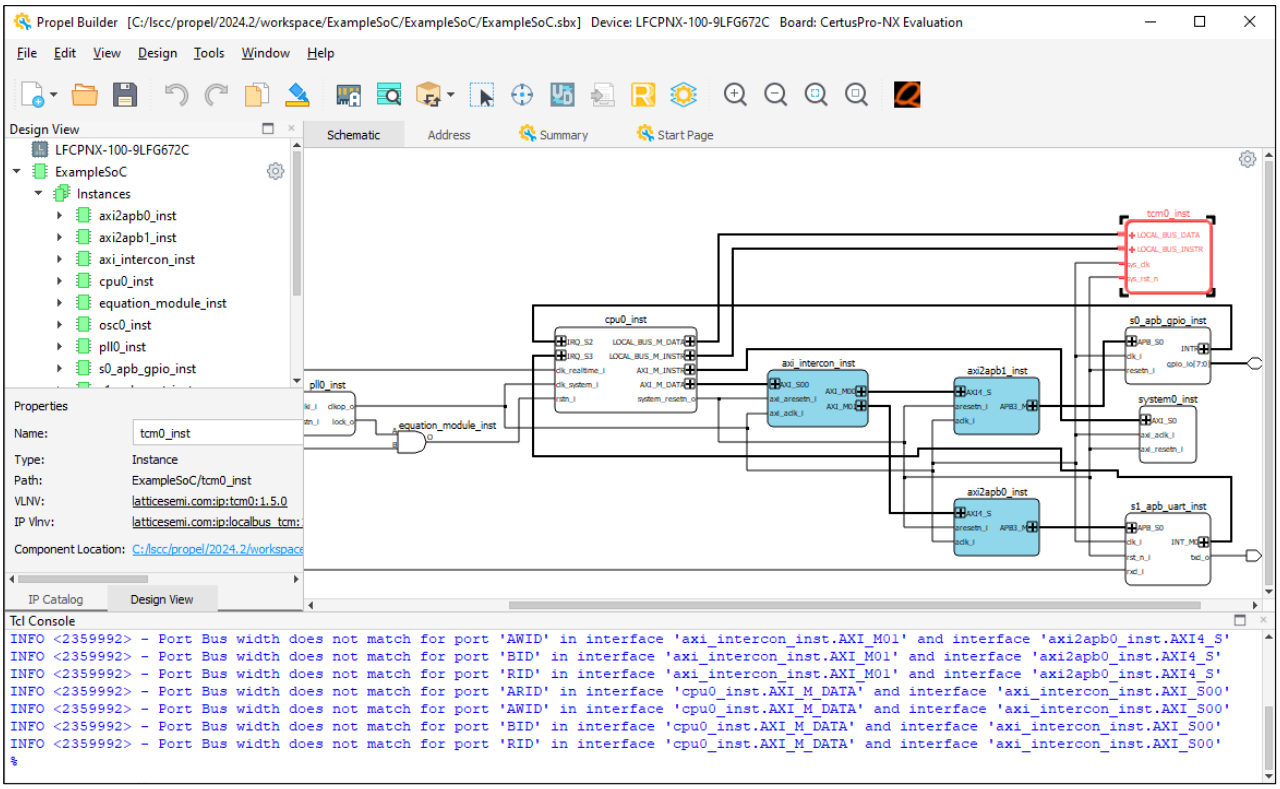


Figure A.8. Corresponding SoC Project

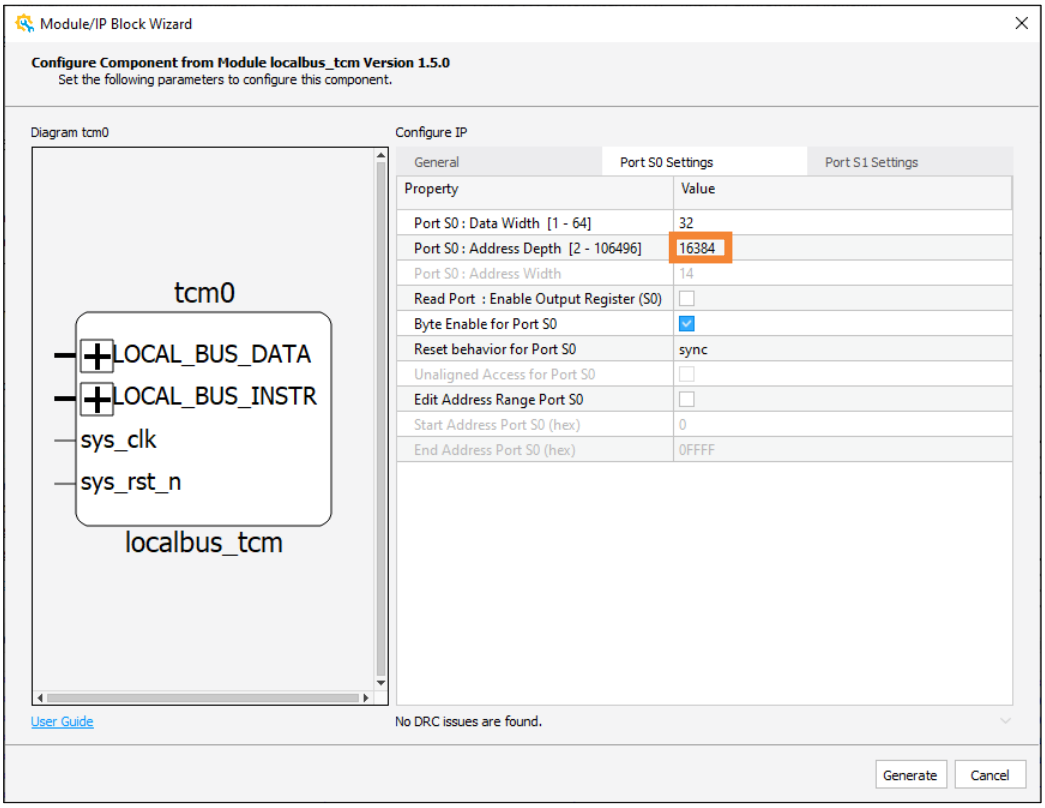


Figure A.9. tcm0_inst Port S0 Address Depth Settings

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

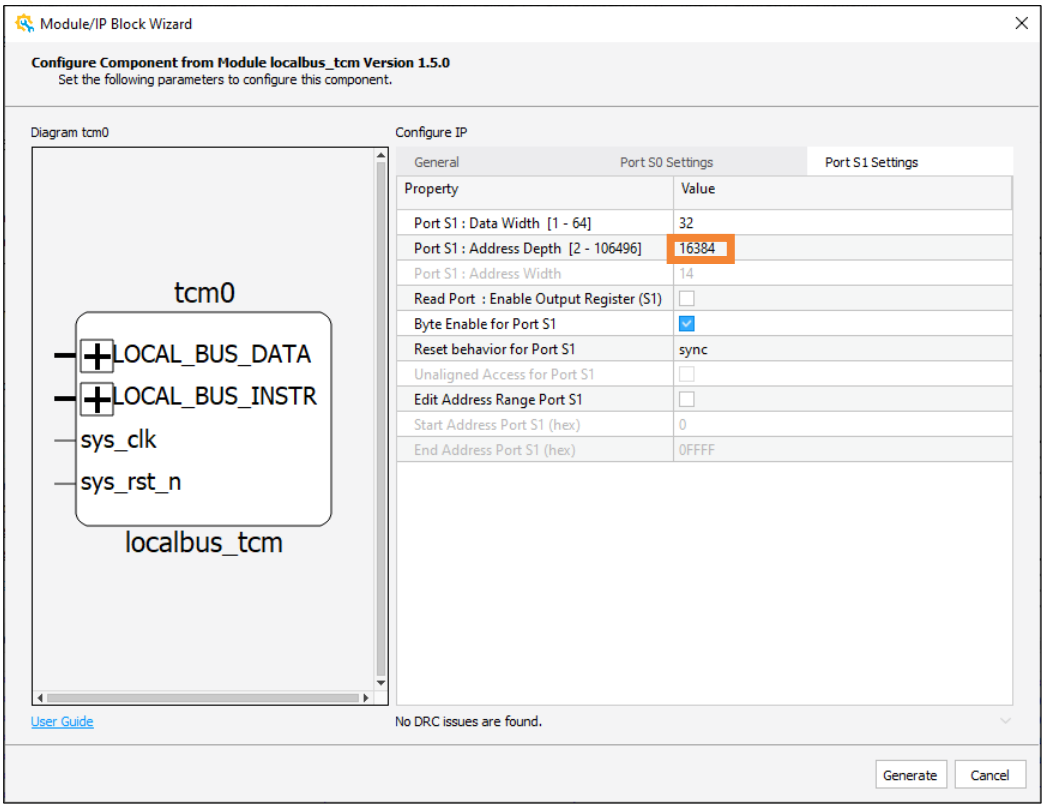


Figure A.10. tcm0_inst Port S1 Address Depth Settings

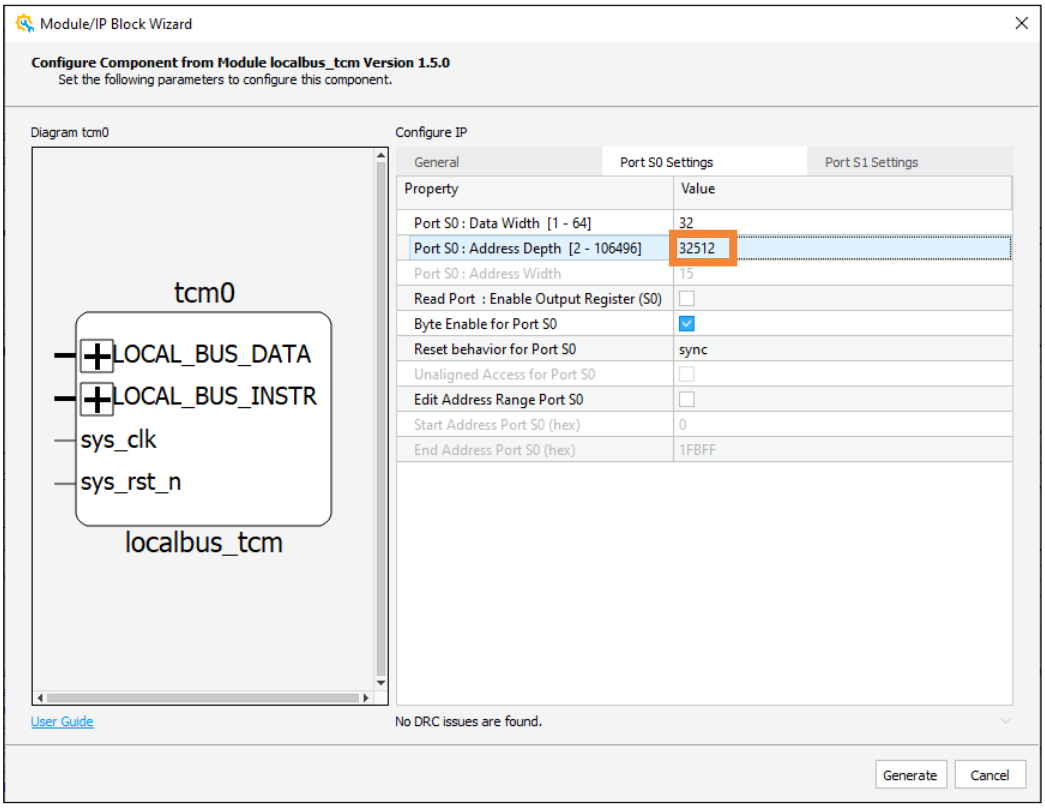


Figure A.11. Modifying tcm0_inst Port S0 Address Depth Settings

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

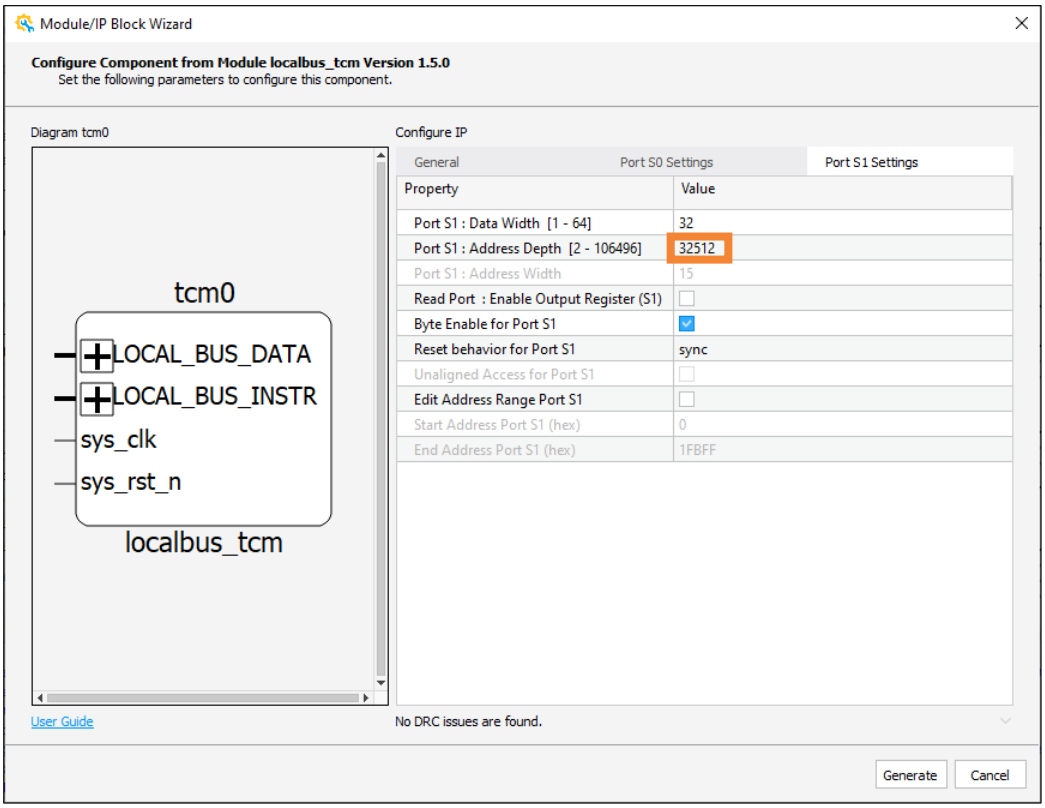


Figure A.12. Modifying tcm0_inst Port S1 Address Depth Settings

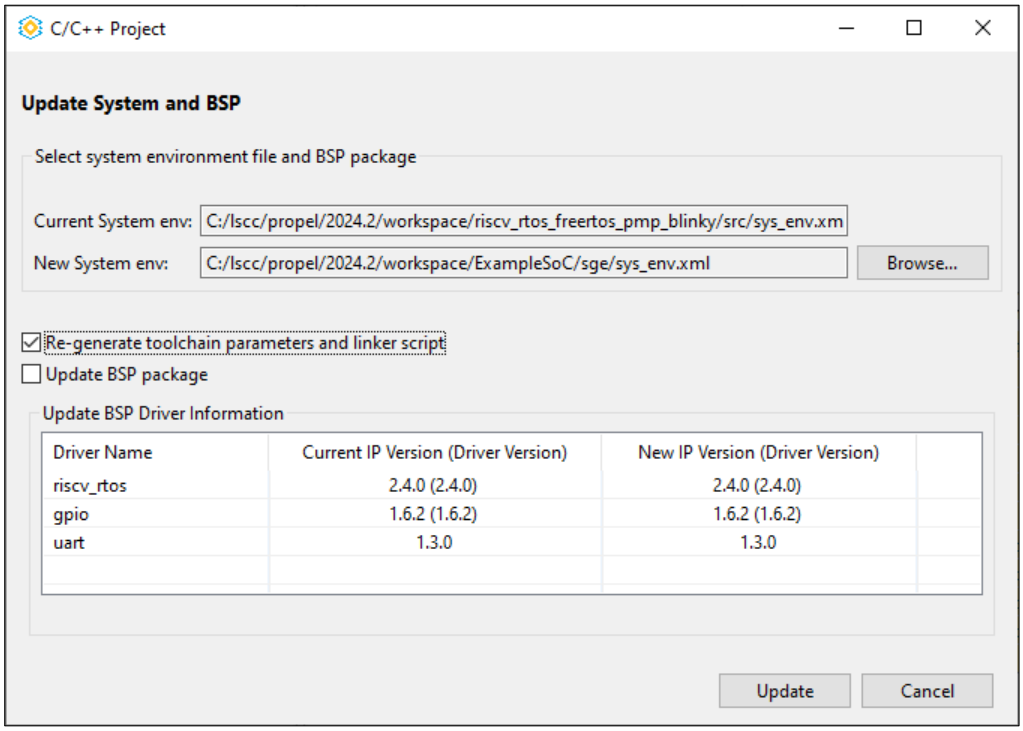


Figure A.13. Updating System and BSP

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

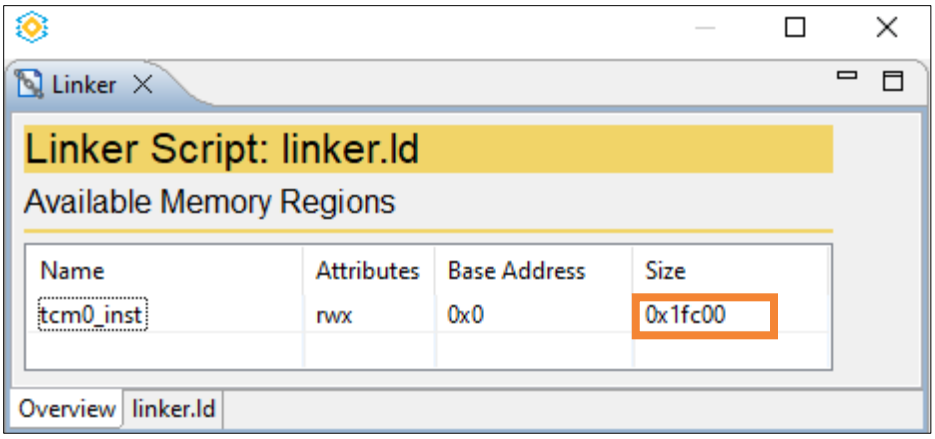


Figure A.14. Updated Linker Script

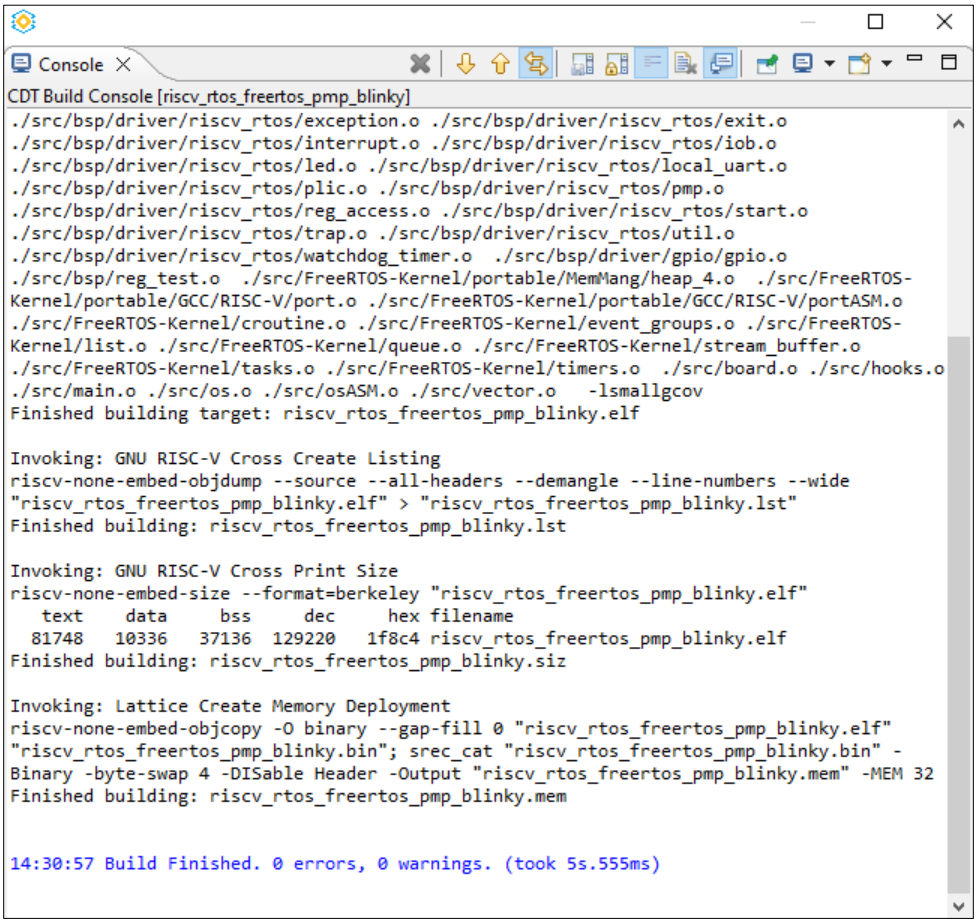


Figure A.15. Build Project Console 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

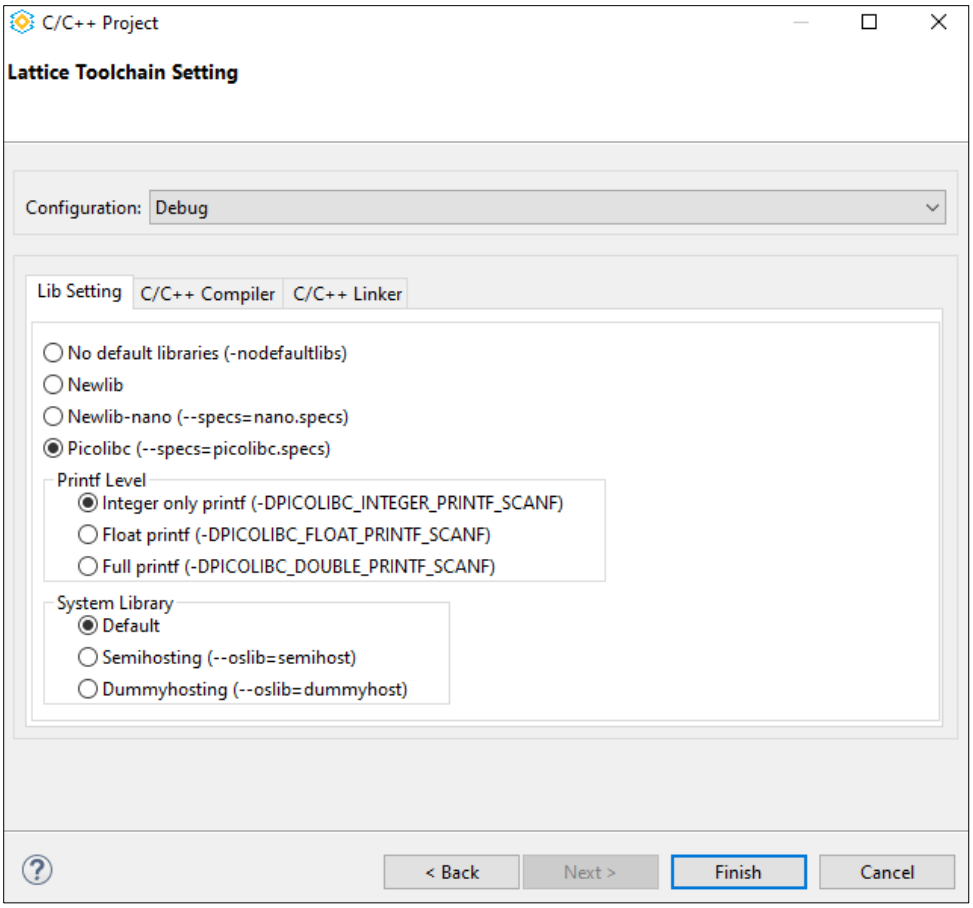


Figure B.1. Lattice Toolchain Setting Dialog 2

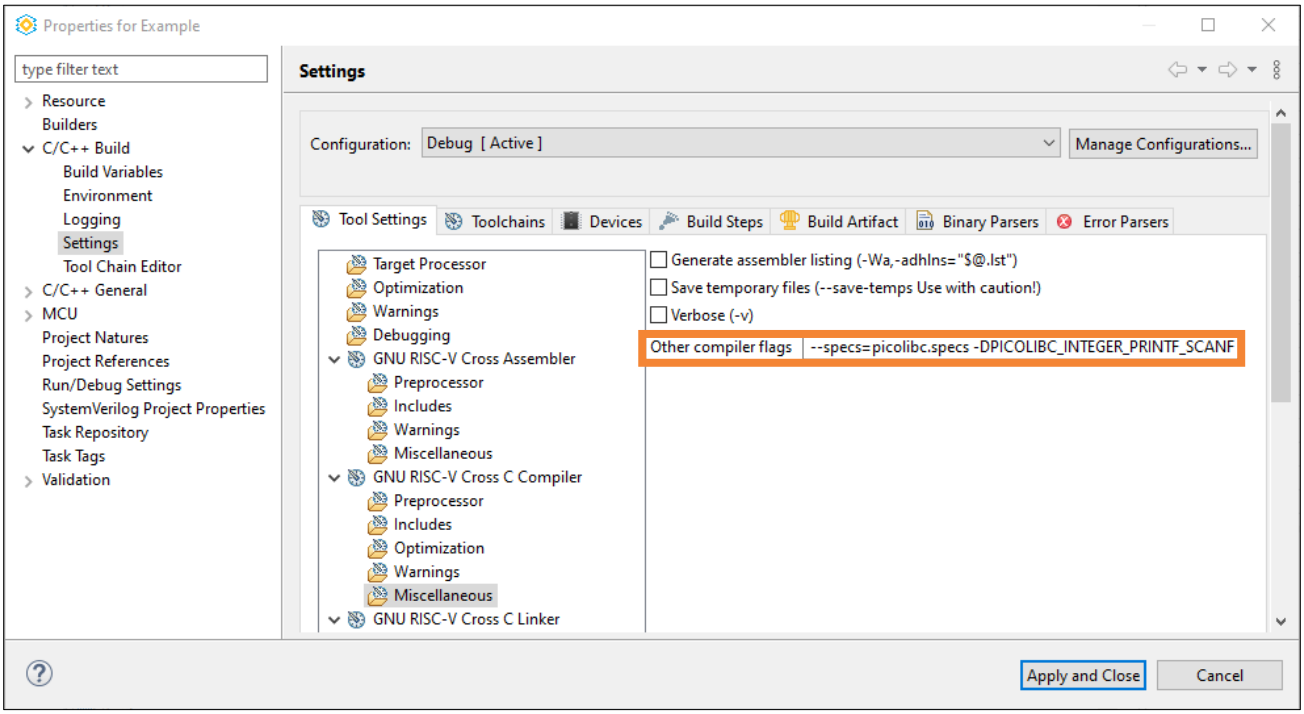


Figure B.2. Properties of C/C++ Project – Compiler Options

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

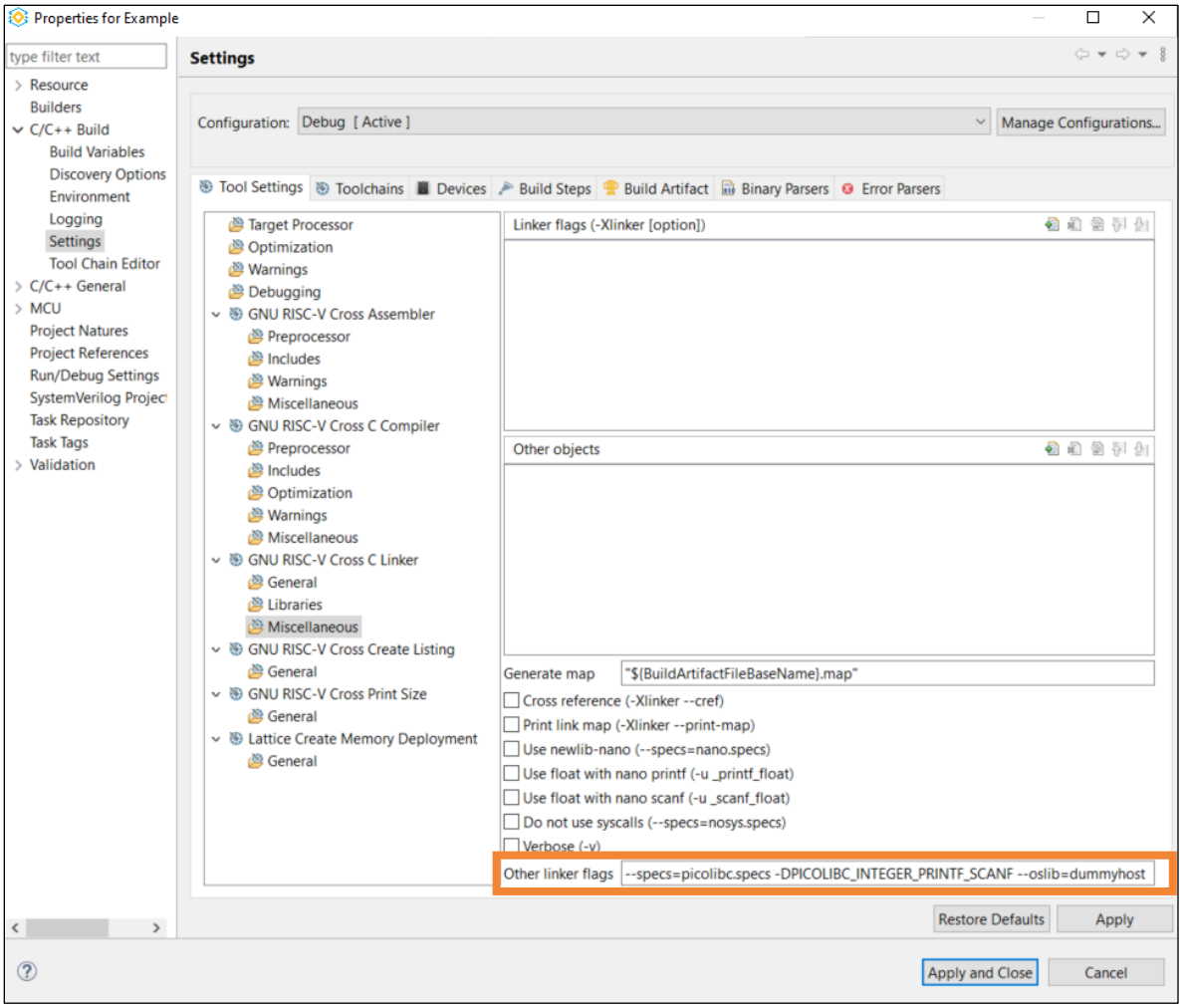


Figure B.3. Properties of C/C++ Project – Linker Options

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 channel 14" -c "set cmdlength 0" -c "set RISCV_SMALL_YAML
src/cpu.yaml" -f interface/lattice-cable.cfg -f target/riscv-small.cfg
```

   You can find the channel value and cmdlength value from the C project environment (Figure D.1).

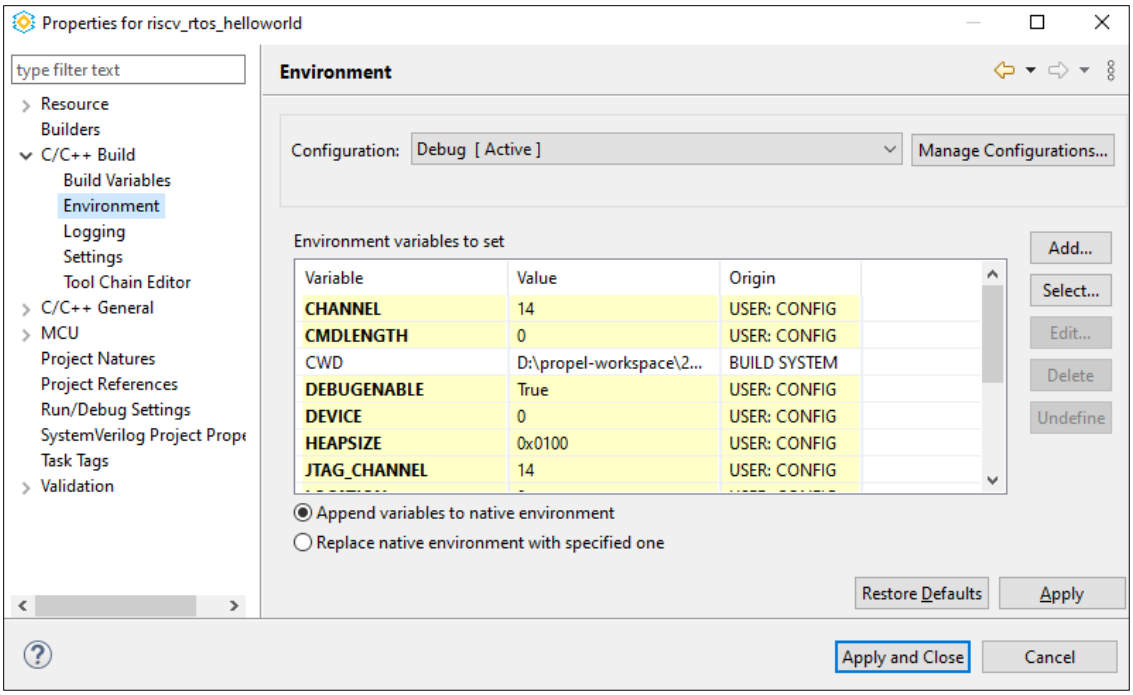


Figure D.1. Project Environment

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Example C: Launching gdb under command line.

```
$ cd <C project Location>/Debug
$ riscv-none-embed-gdb <C project name>.elf
(gdb)target extended-remote localhost:3333
(gdb)monitor reset init
(gdb)monitor arm semihosting enable
(gdb)monitor reset halt
(gdb)…
```

3. Run debug flow with the command line. Below is an example flow.
   a. Start openocd.

      Run propel_commandline.cmd (Windows) or propel_commandline.sh (Linux) to launch the command line window (Figure D.2).

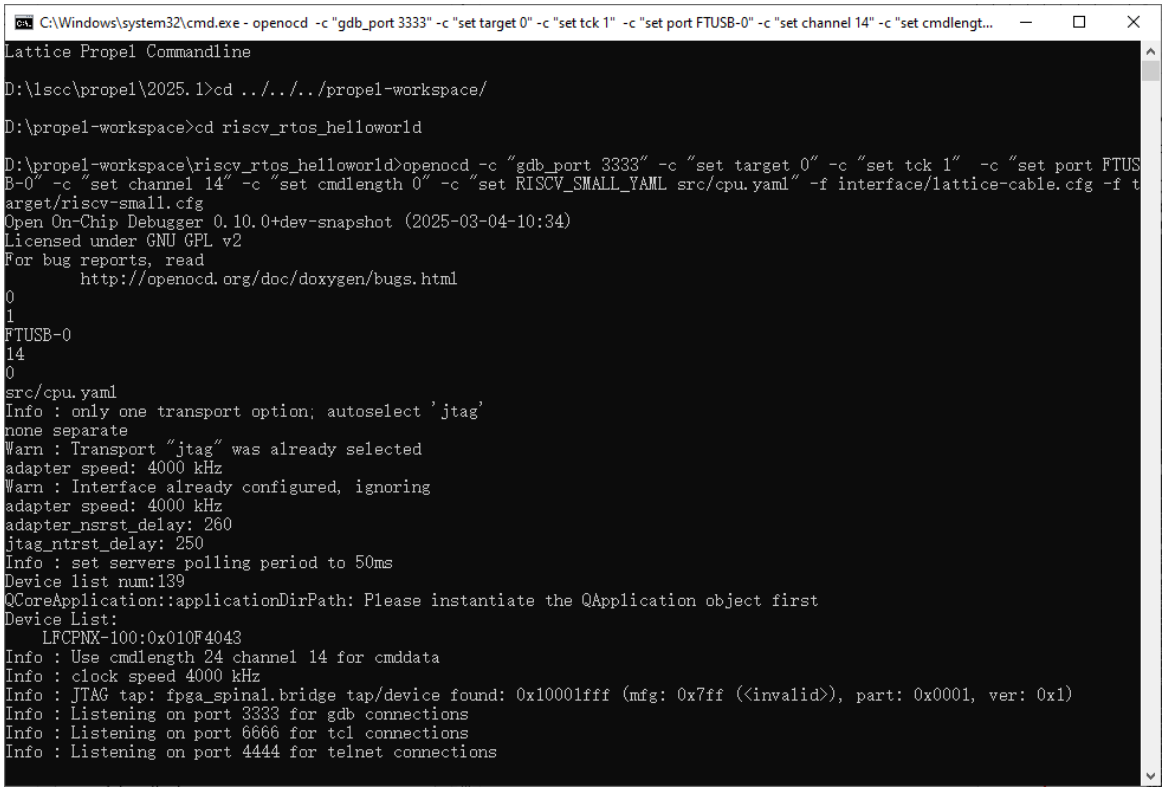


Figure D.2. Launching openOCD

   b. Start GDB.

      Run propel_commandline.cmd (Windows) or propel_commandline.sh (Linux) to launch another command line window (Figure D.3).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

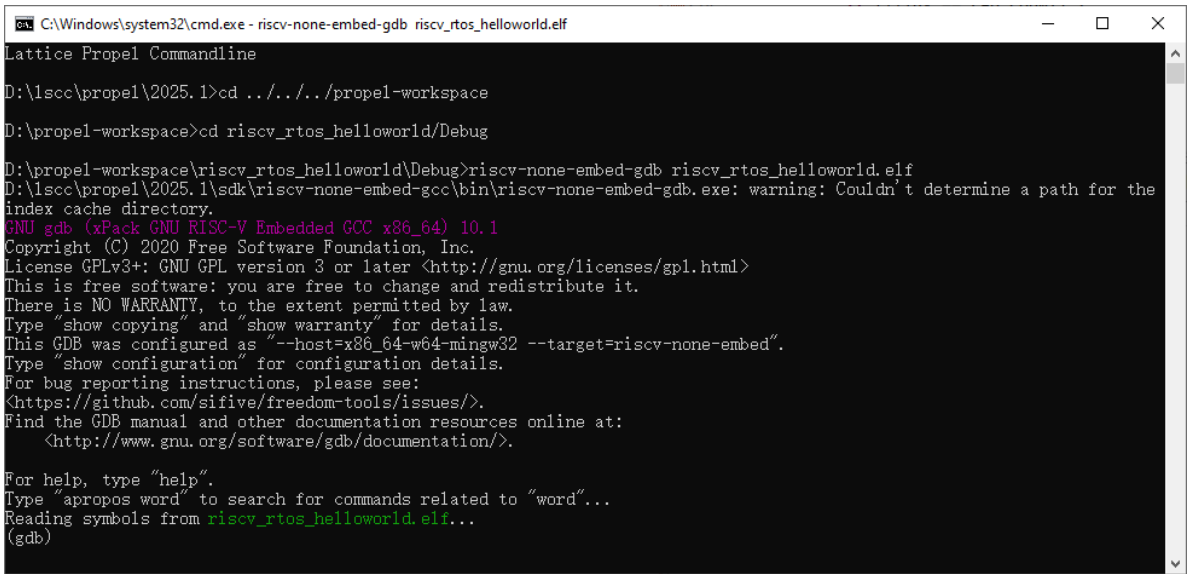


Figure D.3. Launching GDB

c. Start debugging.

Input GDB command in the GDB window (Figure D.4). You can refer to GDB Tutorial for GDB commands.

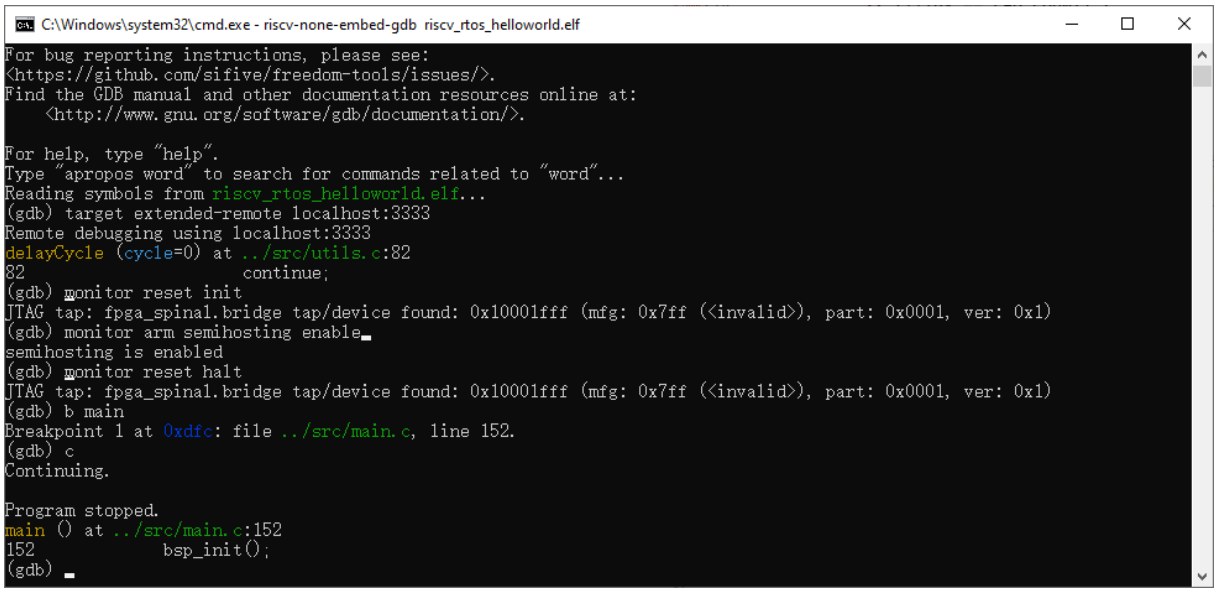


Figure D.4. GDB Debugging Window

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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 a preloaded memory.

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

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.

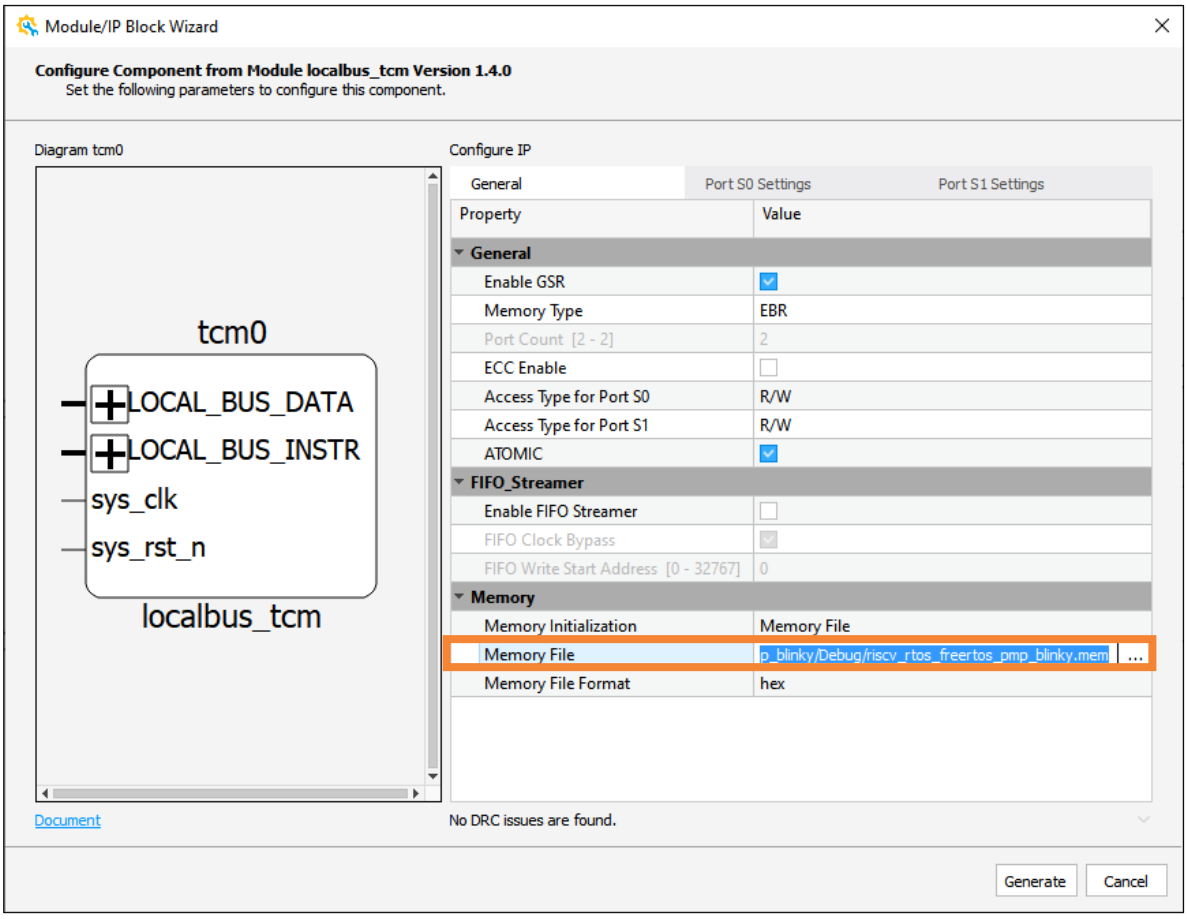


Figure E.1. Setting Memory Initialization File

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 the **Attach to running target** checkbox (Figure E.2). Click **Apply**. Click **Debug**.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

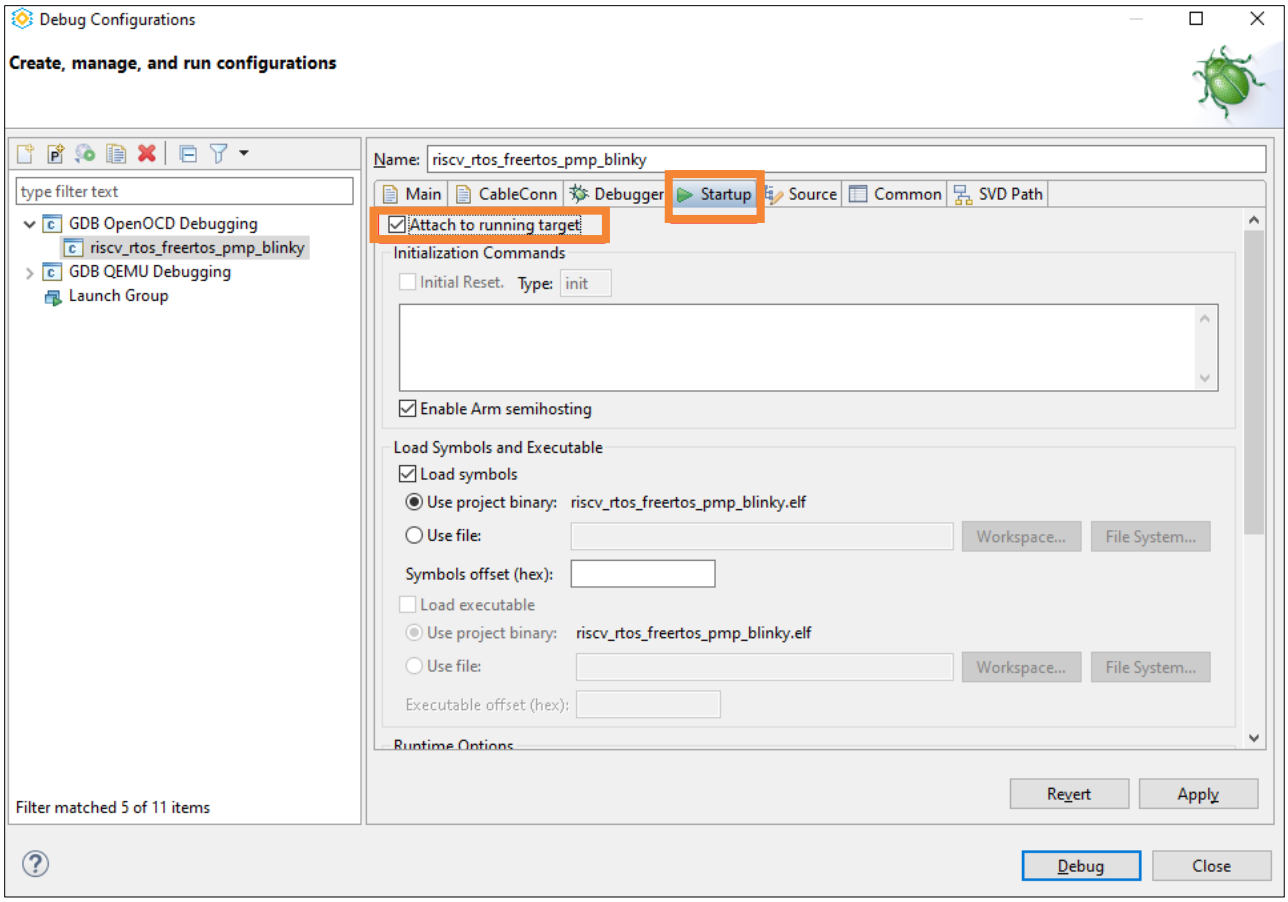


Figure E.2. Debug Configurations Dialog 5

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

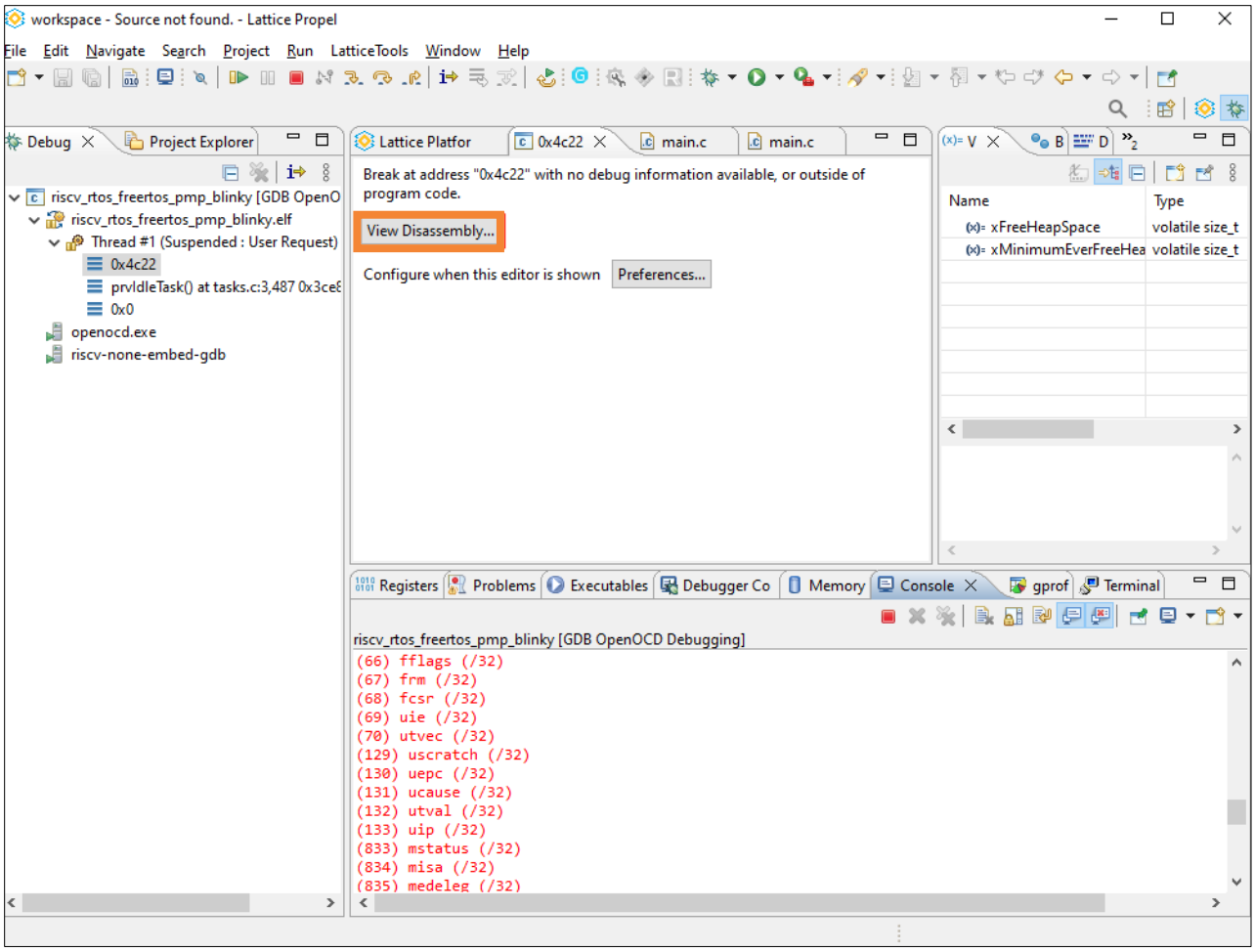


Figure E.3. Debug Perspective 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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).

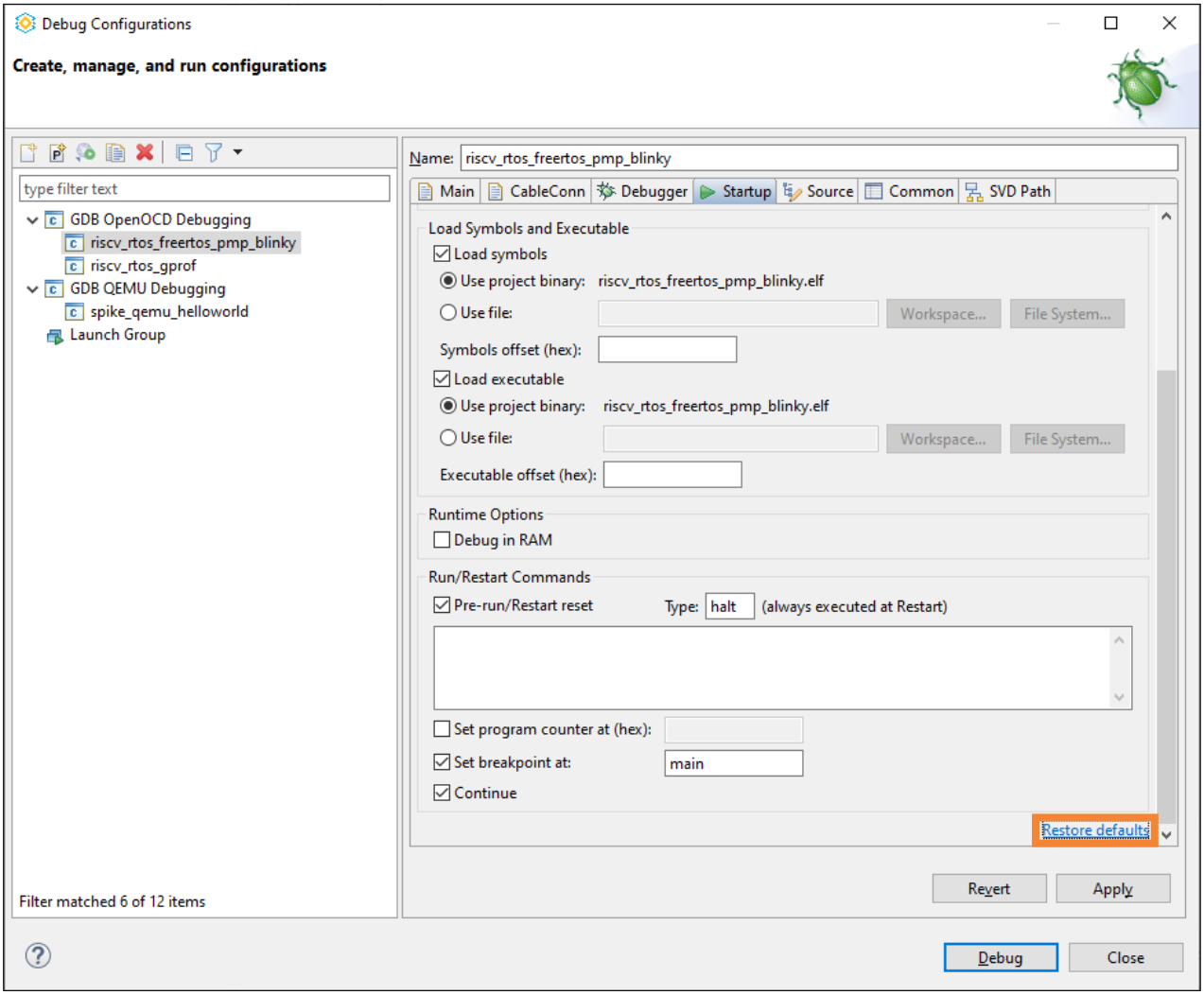


Figure E.4. Restore Defaults Operation

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

Generating Test Code

1. Create an SoC project. Refer to the Creating an SoC Design Project (Deprecated) 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. If you enable register access test, the code shown in Figure F.2 is used. This code is generated automatically and do not edit it. Lattice Propel SDK collects testable register address from used IP, tries to write and read back, and compares these data to get the result of this test.
3. Create a corresponding C project. Choose an application template that **Supports IP register access test** (Figure F.3).

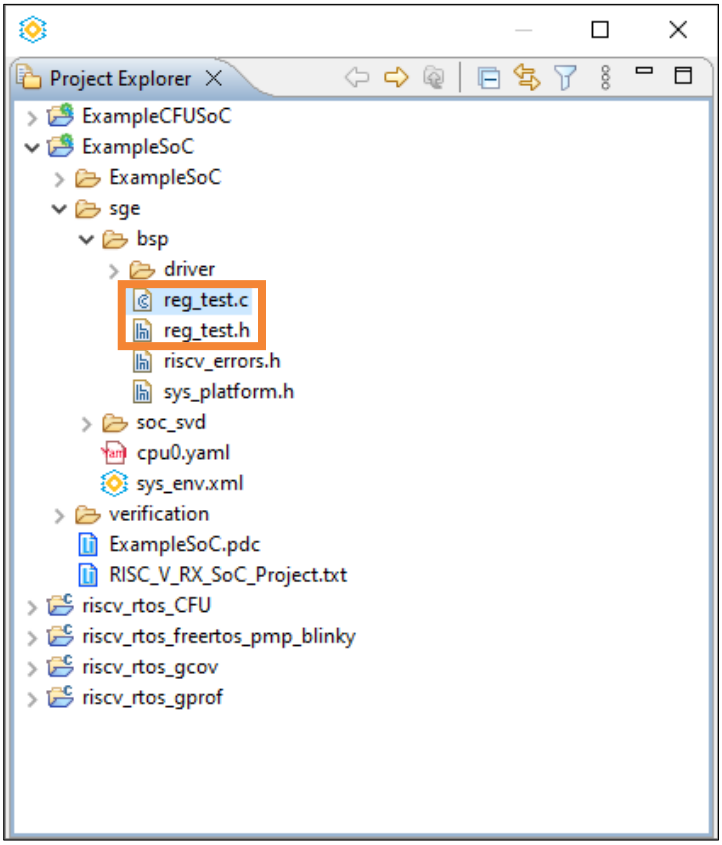


Figure F.1. Project Explorer

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

reg_test.c

```c
#include "stdio.h"
#include "stdint.h"
#include "reg_access.h"
#include "reg_test.h"
#include "sys_platform.h"

#if REG_TEST_ENABLE
uint8_t access_detect(uint32_t reg_addr,uint32_t offset,uint32_t mask,  char *inst_name, char *reg_name)
{
    uint8_t access_success = 0;
    uint32_t reg_32b_value = 0;

    reg_32b_write( reg_addr + offset, 0x5A5A5A5A & mask);
    reg_32b_read(reg_addr + offset, &reg_32b_value);

    if( reg_32b_value == (0x5A5A5A5A & mask))
    {
        reg_32b_write( reg_addr + offset, 0xA5A5A5A5 & mask);
        reg_32b_read(reg_addr + offset, &reg_32b_value);
        if( reg_32b_value == (0xA5A5A5A5 & mask))
            access_success = 1;
    }
    else
    {
        access_success = 0;
        printf("access_detect fail, inst: %s, register: %s, 0x%lx\n", inst_name, reg_name, (reg_addr + offset));
    }
    return access_success;
}

uint8_t mem_access_test(void)
{
    uint8_t ret = 1;

#ifdef I2C_CONTROLLER0_INST_BASE_ADDR

    ret *= access_detect(I2C_CONTROLLER0_INST_BASE_ADDR, OFFSET_I2C_CONTROLLER0_INST_TARGET_ADDRL_REG, MASK_I2C_CONTROLLER0_INST_TARGET
#endif

#ifdef I2C_TARGET0_INST_BASE_ADDR

    ret *= access_detect(I2C_TARGET0_INST_BASE_ADDR, OFFSET_I2C_TARGET0_INST_TARGET_ADDRL_REG, MASK_I2C_TARGET0_INST_TARGET_ADDRL_REG,
#endif

#ifdef UART0_INST_BASE_ADDR

    ret *= access_detect(UART0_INST_BASE_ADDR, OFFSET_UART0_INST_IER, MASK_UART0_INST_IER, "uart0_inst", "IER");
#endif

    return ret;
}
#endif
```

Figure F.2. Register Test Code

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

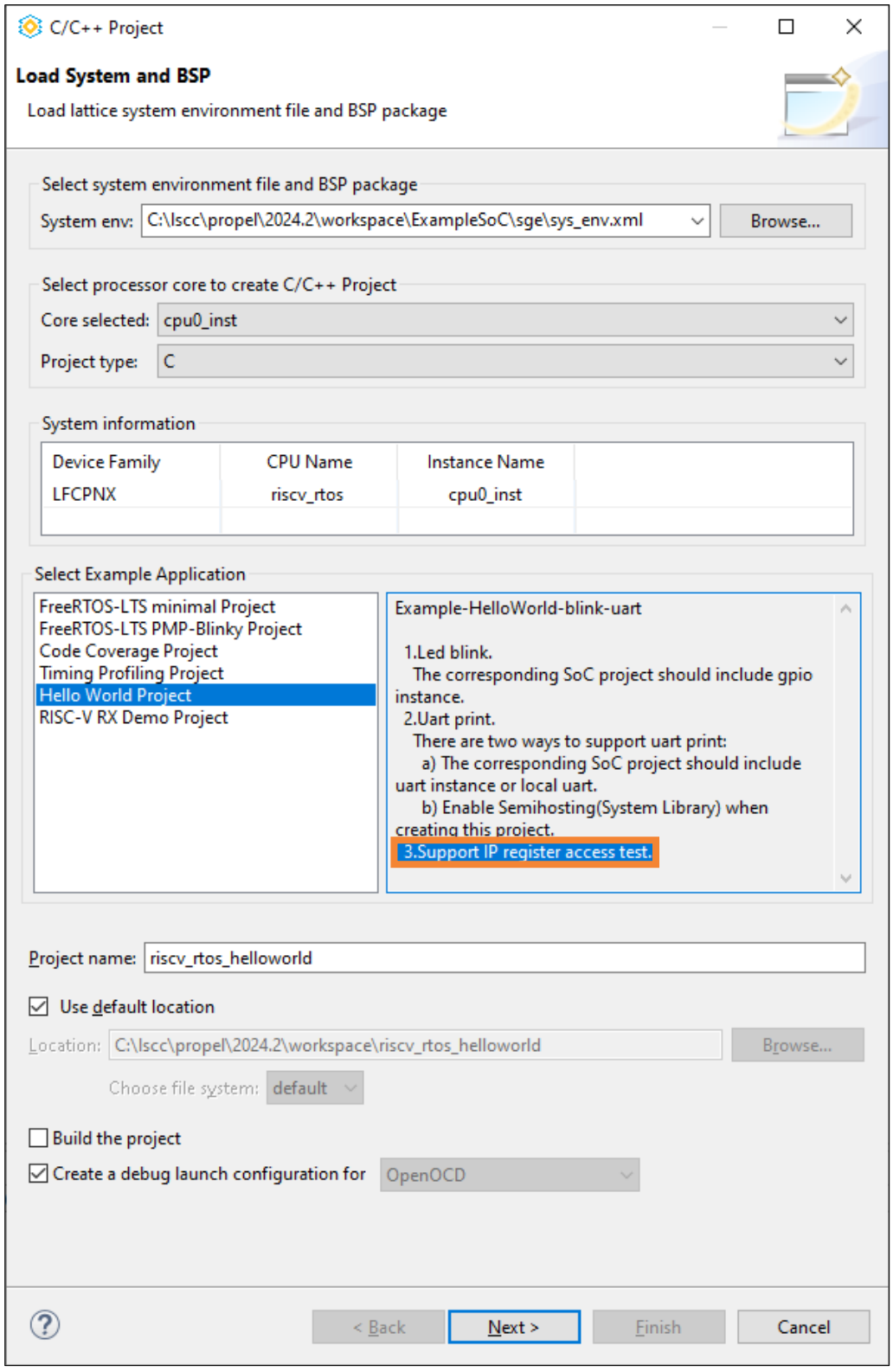


Figure F.3. Load System and BSP Page 5

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Enabling 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.4).
2. Set `define REG_TEST_ENABLE` to 1 in the file **sys_platform.h** (Figure F.5).

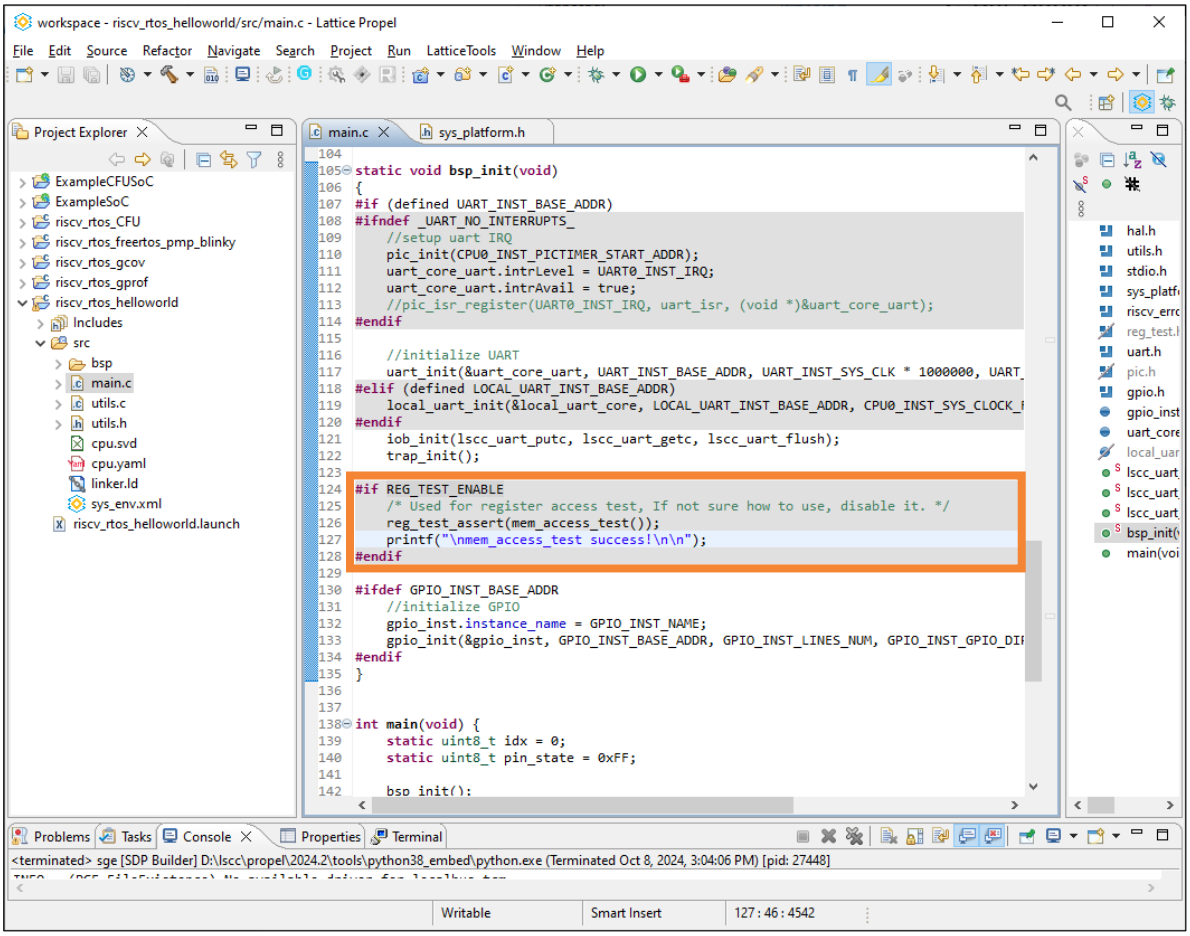


Figure F.4. Test Entrance

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

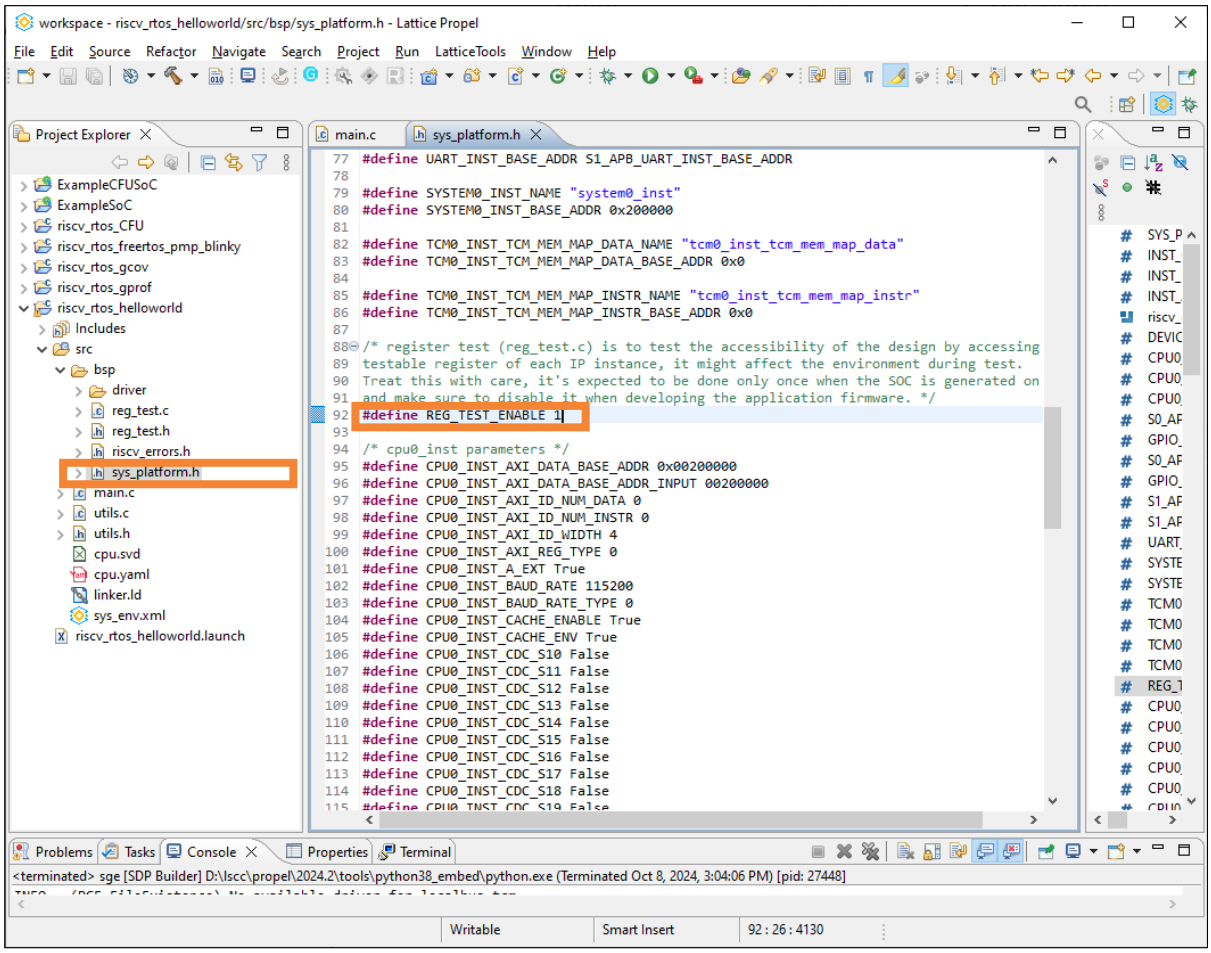


Figure F.5. Enabling Test Code

Running Test

1. Generate the bit file from the corresponding 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 print-out log.
4. Success log (Figure F.6) or failure log (Figure F.7) is shown.

Note: This function is an advance 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.6. Success Log

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

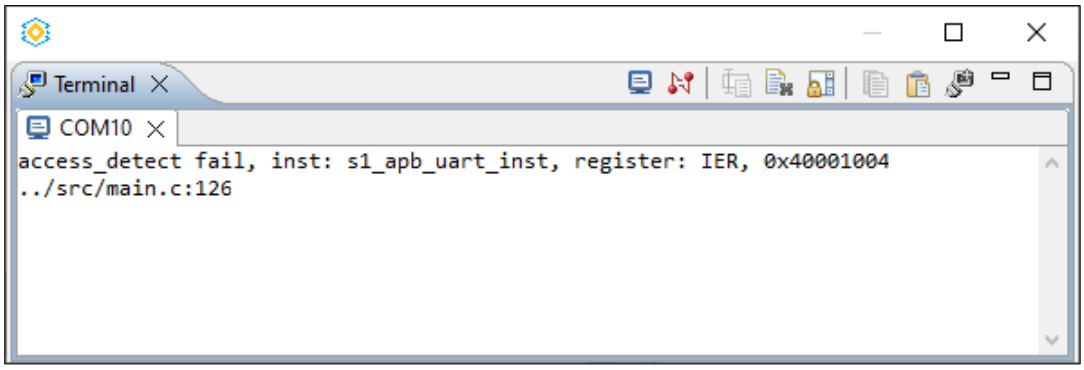


Figure F.7. Failure Log

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix G. Stack Overflowed Check

Introduction

In the example C project shown below, the file linker.ld defines the size of memory stack (Figure G.1).

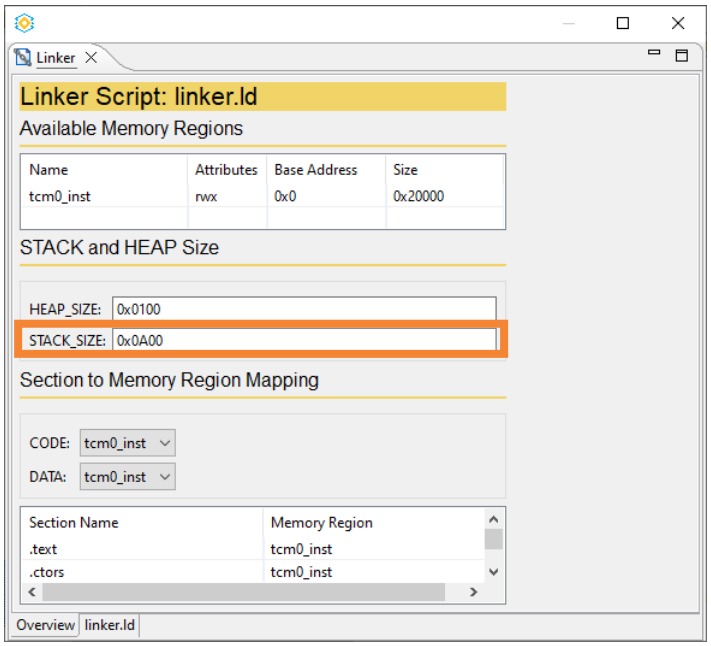


Figure G.1. linker.ld 1

In Figure G.1, STACK_SIZE is set to 0xA00, which means this C project's stack size is 0xA00 (2560) bytes. If you define a larger data array in some functions and use them, as shown in Figure G.2, some hard-to-detect errors might occur.

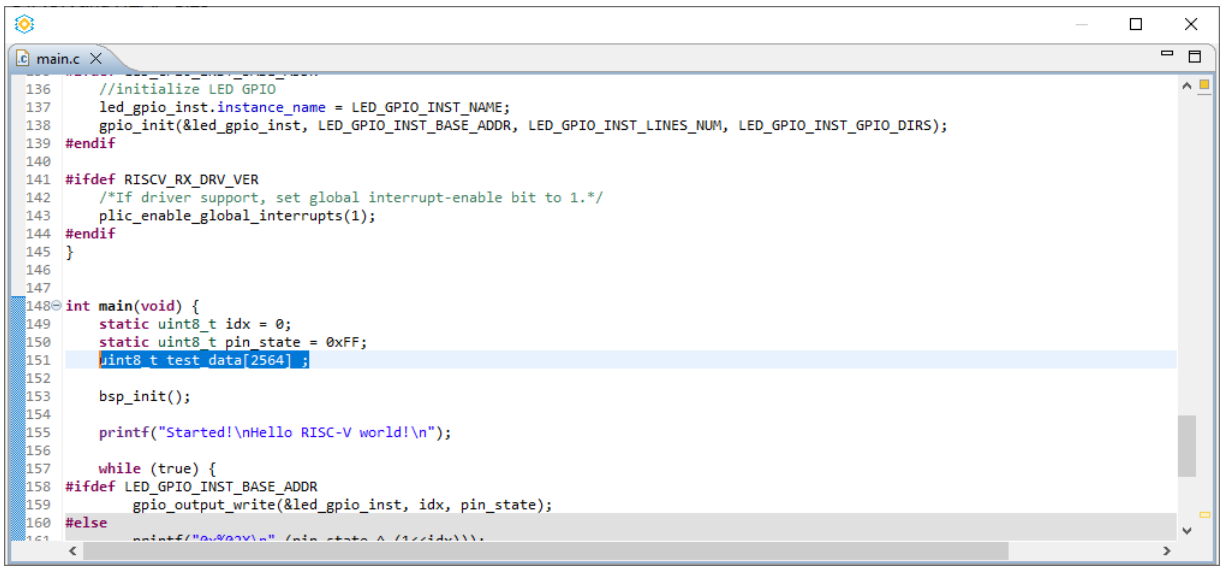


Figure G.2. C Code

To avoid these errors, Lattice Propel SDK provides an additional warning flag (Figure G.3).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

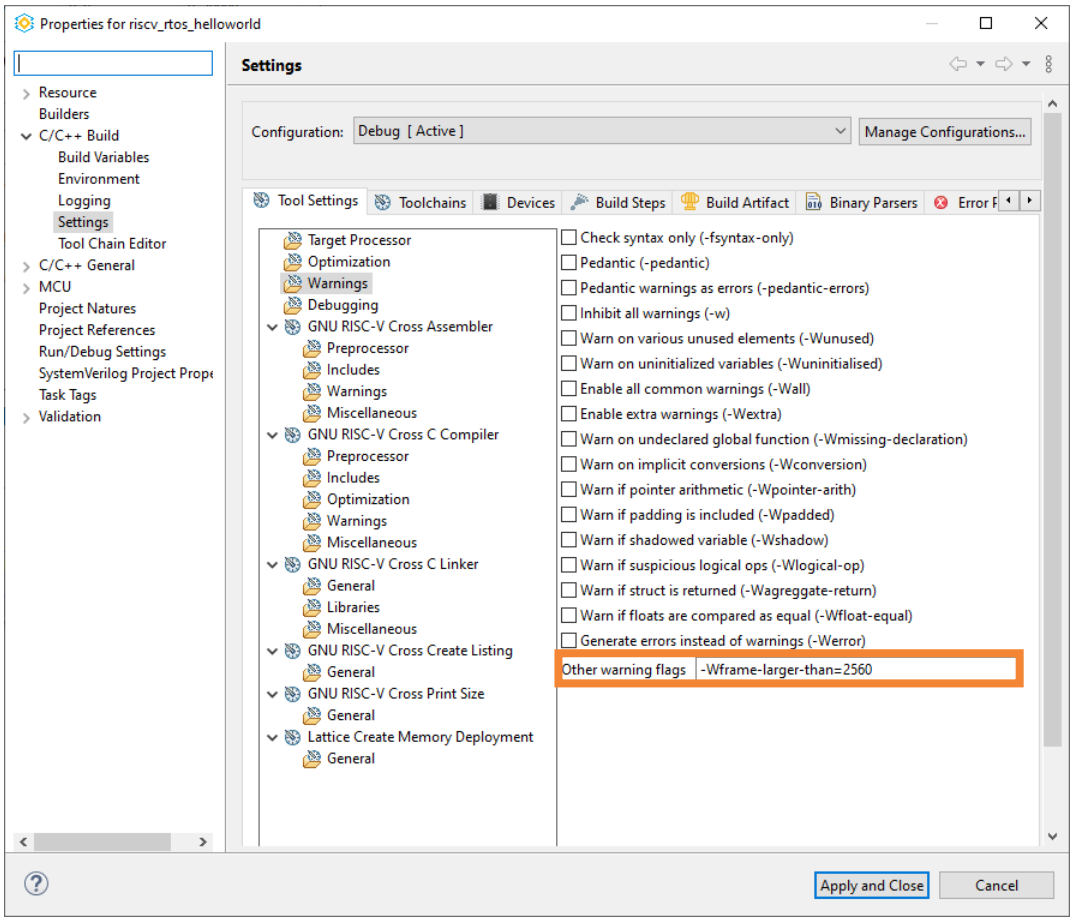


Figure G.3. Warnings Settings 1

Then we can find a warning message when building this project, Figure G.4.

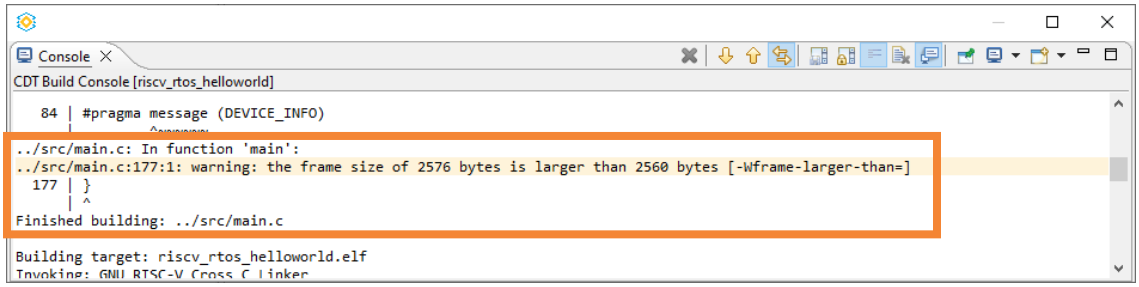


Figure G.4. Warning Message

How to Enable the Stack Overflowed Check Function

- For general templates shown in Table 5.1, their stack size is fixed and cannot be modified. During the C project creating flow, in the **Lattice Toolchain Setting** GUI (Figure G.5), the **Warn for stack frame larger than** item cannot be edit to make sure the template code runs correctly.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

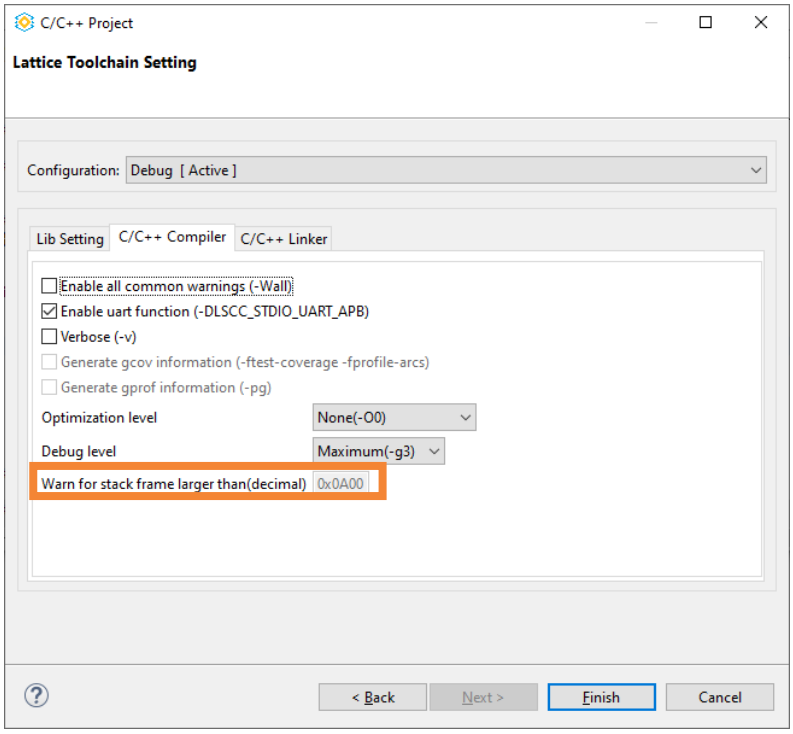


Figure G.5. Toolchain Setting GUI 1

- For solution templates and special templates, such as the CXU template, the **Warn for stack frame larger than** item in the **Lattice Toolchain Setting** GUI can be enabled. To enable this stack overflowed check function, fill in a decimal number in the text box for this item (Figure G.6). If you leave it as default, this function is disabled.
  Note: The number filled in must be equal to STACK_SIZE in the linker.ld file.

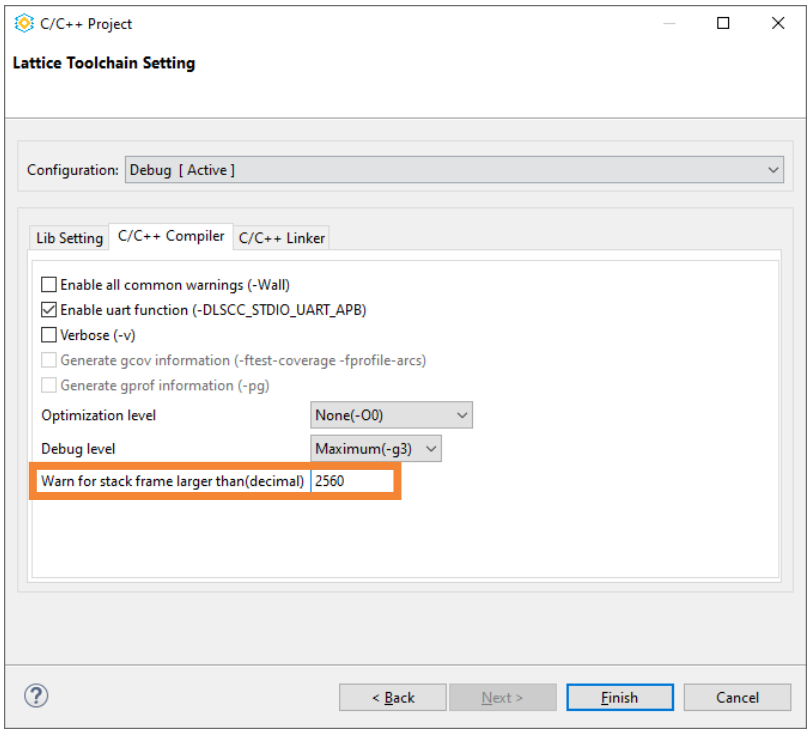


Figure G.6. Toolchain Setting GUI 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- For **Warning** Settings (Figure G.3), you can edit the warning flag manually. Keep the value in decimal format, as shown in Figure G.7.

  Note: If you change this value, set STACK_SIZE in the linker.ld file to the same value (Figure G.8).

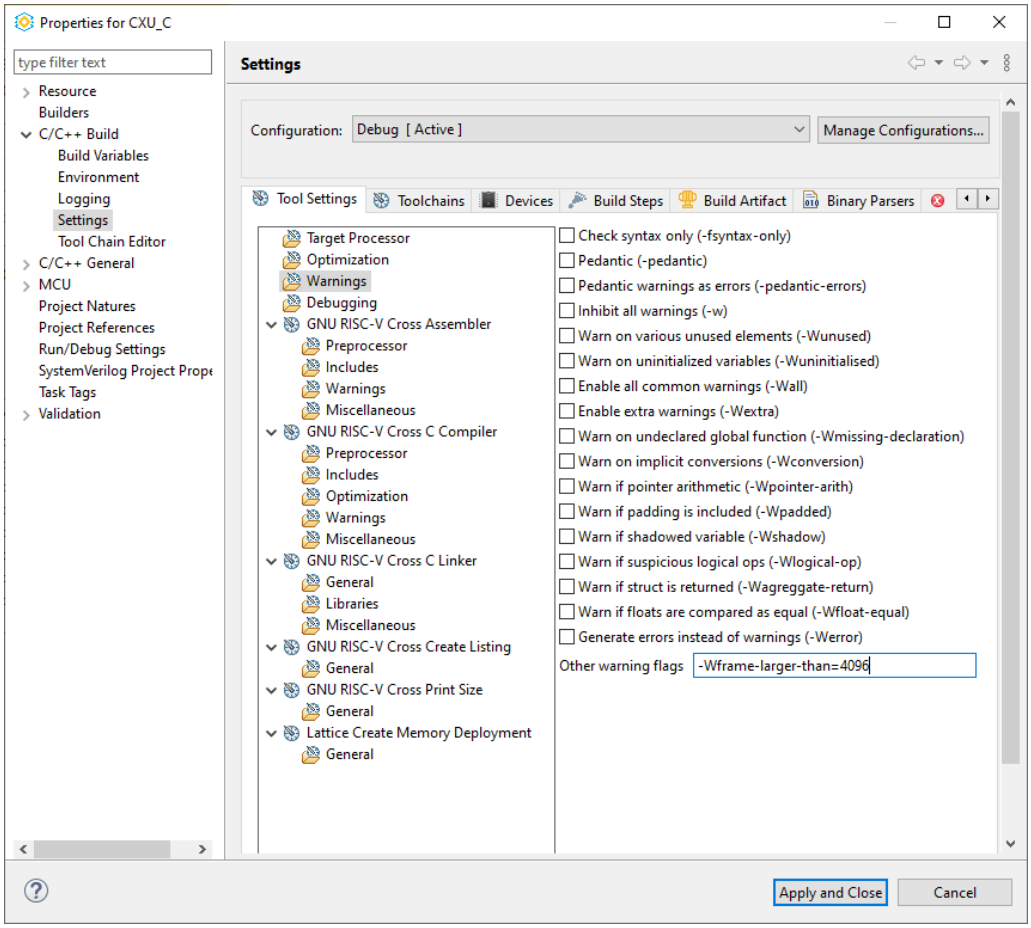


Figure G.7. Warnings Settings 2

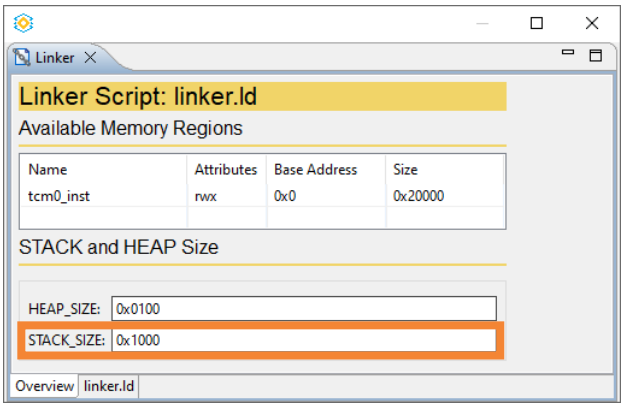


Figure G.8. linker.ld 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix H. Breakpoint and Watchpoint Introduction

We can use GDB commands to add and use breakpoints and watchpoints. Refer to GDB Tutorial for related tutorials.

How to Use Breakpoint

Lattice software processors have two hardware breakpoints. You can add only two hardware breakpoints and others must be of software types.

1. Create a C/C++ project, build it, and execute on-chip debugging. Here a HelloWorld project is taken as an example (Figure H.1).
2. Switch to Debugger Console, which is the GDB command line window.

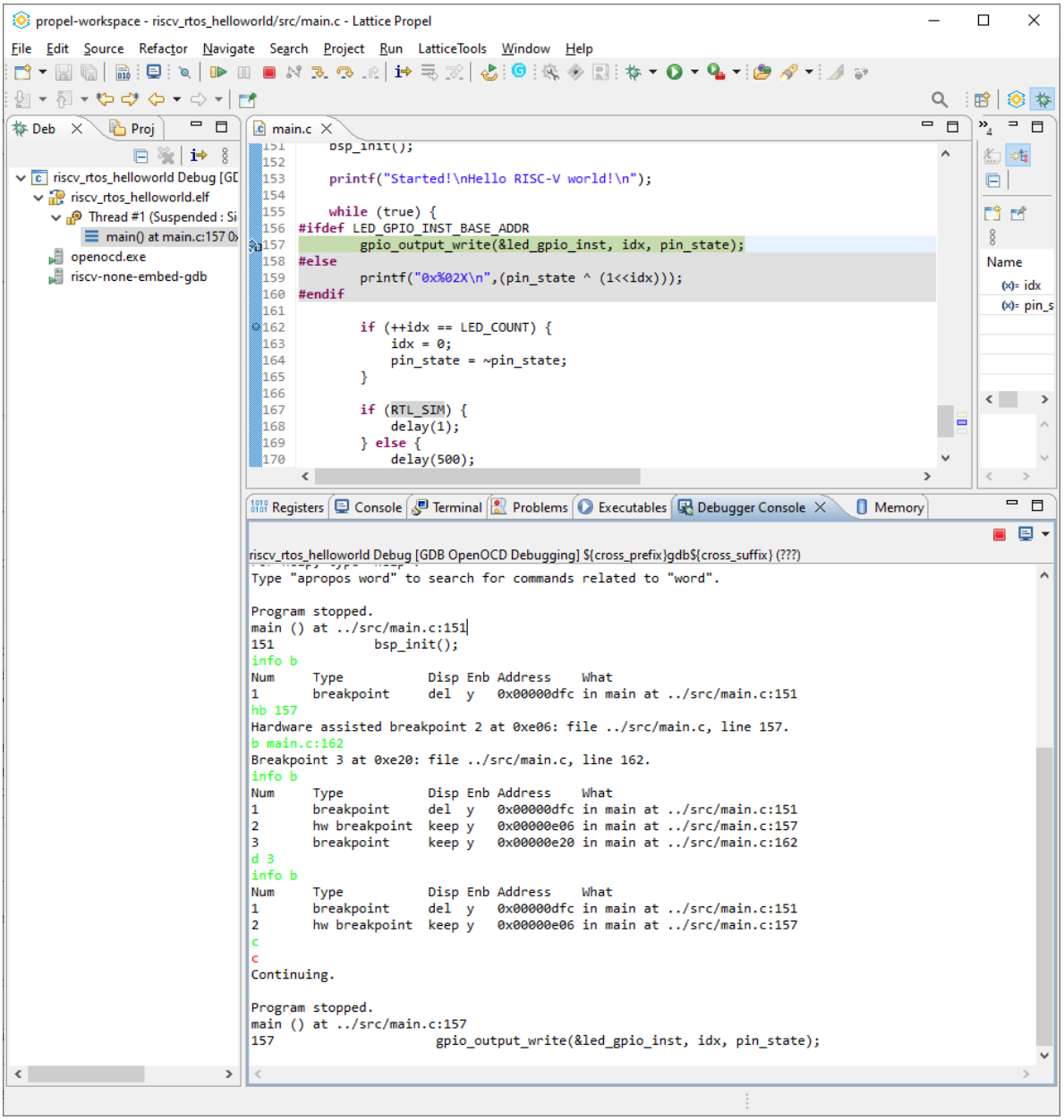


Figure H.1. GDB Console 1

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Input the following commands in sequence in the Debugger Console shown in Figure H.1.

```
info b       //display all of breakpoints
hb 157       //add a hardware breakpoint in line 157
b main.c:162 //add a software breakpoint in lime 162
info b       //display all of breakpoints
d 3          //delect breakpoint by index
info b       //display and check breakpoint list
c            //let program run and find breakpoint works.
```

How to Use Watchpoint

Watchpoint is a special kind of breakpoint.

Lattice software processor does not have any hardware watchpoint. If you need to use a watchpoint, set it as a software watchpoint.

1. Create a C/C++ project, build it, and execute on-chip debugging. A HelloWorld project is used as an example, as shown in Figure H.2.
2. Switch to Debugger Console, which is the GDB command line window.

Input the following commands in sequence in the GDB command line window shown in Figure H.2.

```
info b       //display all of breakpoints
b 157        //add a breakpoint in line 157
c            //let program run to line 157
watch idx    //add watchpoint for idx
c            //let program run and find watchpoint works.
info b       //display all of breakpoints
```

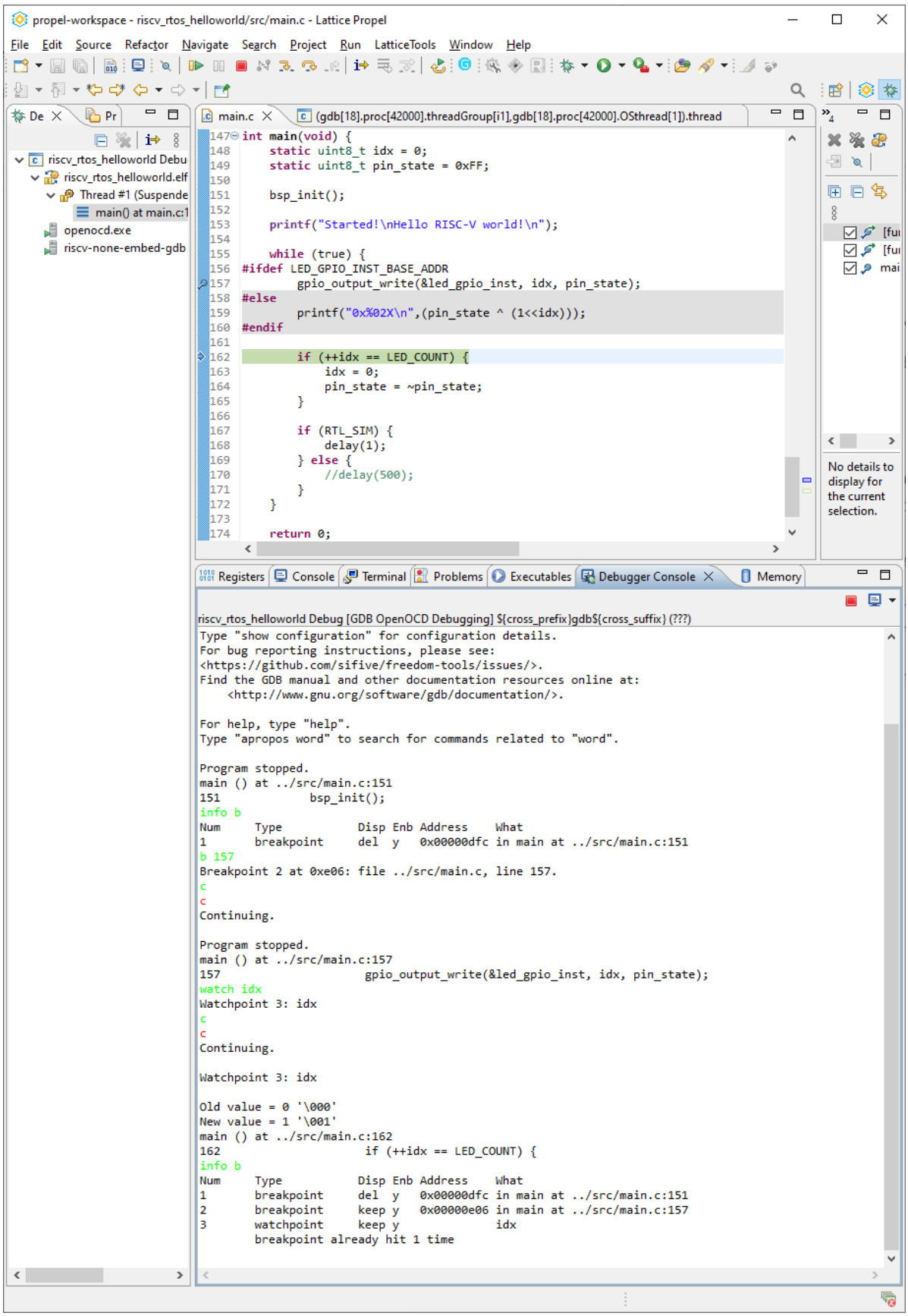
Limitation of Software Watchpoint

If you enable a software watchpoint, the processor checks this watchpoint's status in every instruction cycle, decreasing the program's efficiency. The most noticeable effect is that the program runs more slowly.

In Figure H.2 , line 170 originally includes billions of instruction cycle, and this line is commented out to avoid making the program running too slowly.

Note: It is not recommended to use software watchpoints to analyze code.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.




```
int main(void) {
    static uint8_t idx = 0;
    static uint8_t pin_state = 0xFF;

    bsp_init();

    printf("Started!\nHello RISC-V world!\n");

    while (true) {
#ifdef LED_GPIO_INST_BASE_ADDR
        gpio_output_write(&led_gpio_inst, idx, pin_state);
#else
        printf("0x%02X\n",(pin_state ^ (1<<idx)));
#endif

        if (++idx == LED_COUNT) {
            idx = 0;
            pin_state = ~pin_state;
        }

        if (RTL_SIM) {
            delay(1);
        } else {
            //delay(500);
        }
    }

    return 0;
```

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://github.com/sifive/freedom-tools/issues/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".

Program stopped.
main () at ../src/main.c:151
151             bsp_init();
info b
Num     Type           Disp Enb Address    What
1       breakpoint     del  y   0x00000dfc in main at ../src/main.c:151
b 157
Breakpoint 2 at 0xe06: file ../src/main.c, line 157.
c
c
Continuing.

Program stopped.
main () at ../src/main.c:157
157                     gpio_output_write(&led_gpio_inst, idx, pin_state);
watch idx
Watchpoint 3: idx
c
c
Continuing.

Watchpoint 3: idx

Old value = 0 '\000'
New value = 1 '\001'
main () at ../src/main.c:162
162                     if (++idx == LED_COUNT) {
info b
Num     Type           Disp Enb Address    What
1       breakpoint     del  y   0x00000dfc in main at ../src/main.c:151
2       breakpoint     keep y   0x00000e06 in main at ../src/main.c:157
3       watchpoint     keep y              idx
        breakpoint already hit 1 time
```

Figure H.2. GDB Console 2

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Tips for Using Breakpoint or Watchpoint

- Use software breakpoints or watchpoints as sparingly as possible.

  Every software breakpoint or watchpoint needs the debugging tool to check its status in every instruction cycle, this causes the program to run more slowly. With more breakpoints or watchpoints added, the program gets even slower.

- Clear breakpoints or watchpoints before debugging.

  Lattice Propel SDK workspace is usually used to manage projects. This workspace provides a project explorer to review all projects files. The breakpoints or watchpoints created for one project are also visible to the rest of the projects in the workspace.

  Therefore, if you debug a project, you might see some useless pending breakpoints or watchpoints in the breakpoint list and some error messages in the GDB console (Figure H.3).

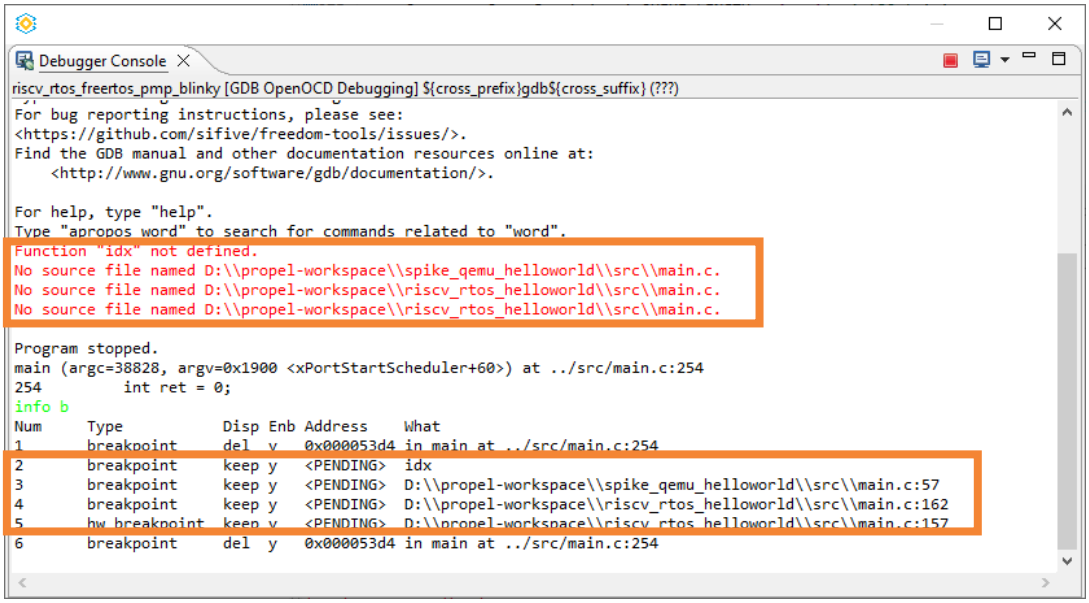


Figure H.3. GDB Console 3

There are two ways to clear these pending breakpoints:

- Select the debugging project, right click and select **Close Unrelated Project** (Figure H.4). Then, re-debug.

  In this way, breakpoints are cleared temporarily for this debugging project, while pending watchpoints are still visible. If you open a project in the Propel SDK workspace again, the breakpoints are still there.

- Select **Run** > **Remove All Breakpoints** from the menu to delete all breakpoints in the current workspace. Then, re-debug. If you want to keep these breakpoints, do not clear them in this way.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

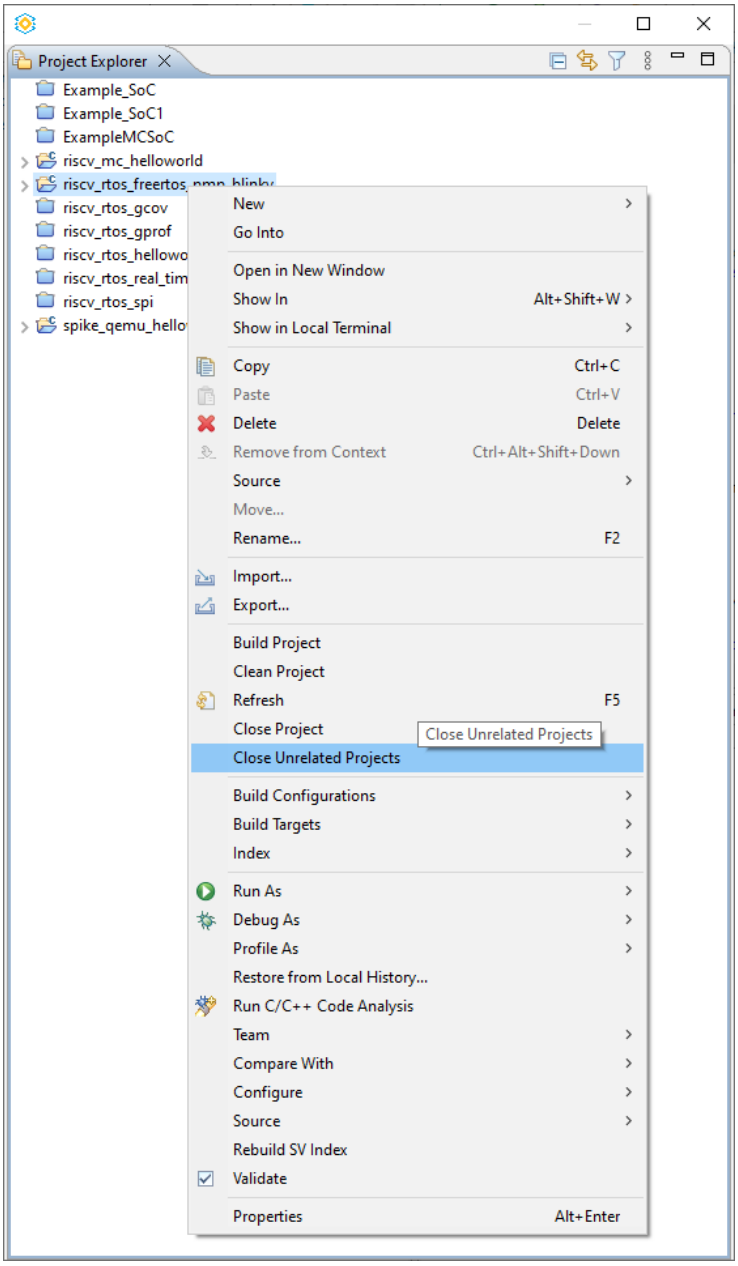


Figure H.4. Closing Unrelated Projects

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

References

- Lattice Propel 2025.1 Release Notes (FPGA-AN-02100)
- Lattice Propel 2025.1 Installation for Windows User Guide (FPGA-AN-02098)
- Lattice Propel 2025.1 Installation for Linux User Guide (FPGA-AN-02099)
- Lattice Propel Builder 2025.1 User Guide (FPGA-UG-02235)
- IP Packager 2025.1 User Guide (FPGA-UG-02236)
- Lattice Propel Revision Control User Guide (FPGA-UG-02237)
- 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

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

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.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Revision History

Revision 1.0, June 2025

Section	Change Summary
All	Production release.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

www.latticesemi.com