



# **A Step-By-Step Approach to Lattice Propel**

## **Application Note**

FPGA-AN-02052-1.1

January 2024

## Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. 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. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. 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.

## Contents

Acronyms in This Document .....	6
1. Introduction .....	7
1.1. Purpose and Overview .....	7
1.2. Audience.....	7
1.3. Lab Requirements .....	7
2. Overview of the Propel Project Flow .....	8
2.1. Project Development Flow .....	8
3. Propel Builder Flow.....	9
3.1. Creating and Managing a Project .....	9
3.2. Downloading, Installing, and Generating IP .....	12
3.2.1. Downloading IP from the IP Server .....	14
3.2.2. Generating and Instantiating the IP .....	15
3.3. Adding Logic and Connecting Components.....	26
3.3.1. Adding Glue Logic.....	26
3.3.2. Manually Connecting Components .....	28
3.3.3. Automatically Connecting Components .....	29
3.3.4. Making Additional Connections.....	31
3.3.5. Creating Top-Level Ports .....	31
3.3.6. Exporting Ports .....	32
3.3.7. Assigning Constant Values to a Port .....	34
3.4. Managing Memory Space.....	35
3.4.1. Propel Builder TCL Commands .....	36
4. Radiant Project Flow.....	37
4.1. Generating Wrapper and Exporting Project.....	37
4.2. Exporting Project to Lattice Radiant.....	37
4.3. Programming the Device .....	39
5. Propel SDK Flow.....	41
5.1. Launching Propel SDK.....	41
5.2. Managing the Workspace .....	41
5.2.1. Switching between Workspaces.....	42
5.3. Creating and Managing a Project .....	43
5.3.1. Creating a New Propel SDK Project .....	43
5.4. Compiling the Project.....	46
5.5. On-Chip Debugging .....	48
5.5.1. Setting up a Serial Terminal.....	48
5.5.2. Debugging with OpenOCD.....	49
5.6. Initializing the Memory .....	50
6. Verification Project Flow.....	52
6.1. Pre-Simulation Requirements .....	52
6.2. Generating a Simulation Environment .....	53
6.2.1. Setting up the Simulation Environment .....	54
6.2.2. Adding Verification IP – Clock and Reset Generator .....	54
6.2.3. Adding Verification IP – UART Model .....	55
6.2.4. Adding Verification IP – AHB-Lite Master Bus Function Model.....	56
6.3. Project Simulation .....	58
6.3.1. Modifying the Simulation Environment .....	58
7. References .....	59
Technical Support Assistance .....	60
Revision History .....	61

## Figures

Figure 3.1. Propel SDK Workspace Selection Window.....	9
Figure 3.2. Propel SDK Workspace Environment .....	10
Figure 3.3. SoC Project Creation Window and Settings for the Demo.....	10
Figure 3.4. New Empty SoC Project in Propel Builder .....	11
Figure 3.5. Recent Project List in Propel Builder Start Page .....	12
Figure 3.6. IP Catalog and Design View .....	12
Figure 3.7. IP on Local Tab of IP Catalog .....	13
Figure 3.8. Additional Information Icon for Oscillator IP v1.3.0 .....	13
Figure 3.9. Additional Information for Oscillator IP v1.3.0 .....	14
Figure 3.10. Downloaded and Installed IPs.....	15
Figure 3.11. RISC-V MC Component Parameter Configuration .....	15
Figure 3.12. UART Component Parameter Configuration .....	16
Figure 3.13. GPIO Component Parameter Configuration .....	17
Figure 3.14. Watchdog Timer Component Parameter Configuration.....	18
Figure 3.15. Oscillator Component Parameter Configuration .....	19
Figure 3.16. PLL Component Parameter Configuration .....	20
Figure 3.17. AHB-L interconnect Component Parameter Configuration .....	21
Figure 3.18. AHB Lite to APB Bridge Component Parameter Configuration .....	22
Figure 3.19. APB Interconnect Component Parameter Configuration .....	23
Figure 3.20. System Memory Component Parameter Configuration .....	24
Figure 3.21. AHB Lite Feedthrough Component Parameter Configuration .....	25
Figure 3.22. Glue Logic Section in IP on Local.....	26
Figure 3.23. Split Glue Logic Configuration.....	27
Figure 3.24. First Equation Glue Logic Configuration .....	27
Figure 3.25. First Equation Glue Logic Configuration .....	28
Figure 3.26. Schematic View after Connecting Glue Logic Components .....	29
Figure 3.27. Connect Ports Window to Select Components to Connect .....	29
Figure 3.28. Initial Interface Connections to Make.....	30
Figure 3.29. Initial Interface Connections to Make.....	31
Figure 3.30. Schematic View with Additional Connections .....	31
Figure 3.31. Create Port Settings for debug_o .....	32
Figure 3.32. Exported Port for GPIO Component .....	32
Figure 3.33. Component Name Field in Design View.....	33
Figure 3.34. Assigning Constant Value to Oscillator Input.....	34
Figure 3.35. Final SoC Design .....	34
Figure 3.36. Address Tab.....	35
Figure 4.1. Adding Physical Constraints in Lattice Radiant File List .....	37
Figure 4.2. Design Process Bar .....	38
Figure 4.3. Design Process Bar after Running Project Flow .....	38
Figure 4.4. Cable Setup to Detect Programming Cable .....	39
Figure 4.5. Device Properties Dialog Box .....	40
Figure 5.1. Propel SDK Workspace Selection .....	41
Figure 5.2. Switch Workspace Option.....	42
Figure 5.3. Load System and BSP Page of Project Creation Wizard .....	43
Figure 5.4. Initial C/C++ Project Creation Page .....	44
Figure 5.5. Lattice Toolchain Setting Page .....	44
Figure 5.6. Lattice Toolchain Setting Page .....	45
Figure 5.7. Compile Project Options .....	46
Figure 5.8. Project Settings Page .....	47
Figure 5.9. Terminal Tab in Propel SDK.....	48
Figure 5.10. Launch Terminal Dialog Box.....	48
Figure 5.11. Debug Configurations .....	49

Figure 5.12. Debug Configurations for GDB OpenOCD Debugging.....	49
Figure 5.13. ECO Editor Memory Initialization Tab.....	51
Figure 6.1. System Memory Parameters to Initialize Memory .....	52
Figure 6.2. RISC-V MC Parameters for Project Simulation.....	53
Figure 6.3. Initial Verification Project Schematic for Empty Projects .....	53
Figure 6.4. Regenerated DUT Component Expanded to View Contents .....	54
Figure 6.5. Clock and Reset Generator Component Parameter Configuration .....	54
Figure 6.6. UART Model Component Parameter Configuration .....	55
Figure 6.7. AHB-Lite Master BFM Component Parameter Configuration.....	56
Figure 6.8. Final Verification Project Design .....	57

## Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AHB-L	Advanced High-performance Bus-Lite
APB	Advanced Peripheral Bus
BSP	Board Support Package. The layer of software containing hardware-specific drivers and libraries to function in a particular hardware environment.
DUT	Design Under Test.
ECO	Engineering Change Order
FPGA	Field-Programmable Gate Array
IDE	Integrated Development Environment.
OCD	On-Chip Debugger
OEM	Original Equipment Manufacturer.
RISC-V	Reduced Instruction Set Computer-V
SDK	Embedded System Design and Development Kit. A set of software development tools that allows the creation of applications for software package on the Lattice embedded platform.
SoC	System-on-Chip
TCL	Tool Command Language
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

# 1. Introduction

## 1.1. Purpose and Overview

Embedded system solutions play an important role in FPGA system design, allowing the user to develop software for a processor in an FPGA device. It provides flexibility for the user to control various peripherals from a system bus. To develop an embedded system on an FPGA, the user needs to design the System-on-Chip (SoC) with an embedded processor and develop system software on the processor. Lattice Propel™ helps the user develop a system with a RISC-V processor, peripheral IP, and a set of tools.

The purpose of this document is to introduce the general project development flow for Propel Builder and Propel SDK projects. Some of the topics that are covered include: project creation, IP generation, block based SoC development, generating a bitstream, debugging firmware on an FPGA, verification project flow, and more.

## 1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice FPGA devices. This document is tutorial-like and is targeted towards those that are new to Lattice Propel or want to learn more about its design process and various features. The technical guidelines for this document assume that readers have at least some expertise with digital design, FPGAs, and embedded systems.

## 1.3. Lab Requirements

The main requirements to follow along with this document are Lattice Radiant™ version 3.2 and Lattice Propel version 2.2. Aside from these, download the set of files that are included with this document. Use this as a starting point in the project. A CrossLink™-NX Evaluation board is recommended for the demo. The user, however, can follow along for most of the lab without any device by skipping the portions that involve device programming and debugging.

The contents of the project .zip file are as follows:

- constraints.pdc – physical constraints for the project
- Project\_Ref – completed Propel Builder and SDK projects (for use as a reference)
- reconstruct.tcl – Propel Builder TCL script to regenerate the demo project from scratch using TCL commands

## 2. Overview of the Propel Project Flow

The Propel project development flow is broken into a few different parts, each corresponding to a different part of the Propel project flow. The list below indicates each part of the project development flow, as well as the topics that are discussed in this document. It is important to remember that the exact flow to follow depends on the user's role within the project development team. By going through the entire Propel project development flow, this demo seeks to improve the user's understanding of each part of the process.

### 2.1. Project Development Flow

- Propel Builder
  - Project creation
  - Downloading, generating, and installing IP
  - Connecting components
  - Creating ports and interfaces
  - Memory space management
  - Project simulation
  - Exporting project to Lattice Radiant or Lattice Diamond®
  - Verification projects
- Lattice Radiant/Lattice Diamond®
  - Synthesis, MAP, and PAR
  - Bitstream generation
  - Memory initialization
  - Device programming
- Propel SDK
  - Project creation
  - Compiling source files
  - Generating a memory file
  - On-chip debugging

**Important:** The user can complete the Propel project development flow using the three basic tools in any order. Typically, the Propel project development flow requires frequent switching between various Lattice software tools. For example, the user may need to switch often between Propel Builder, Propel SDK, and Lattice Radiant to make changes to the design/software and reprogram the device.



## 3. Propel Builder Flow

This section describes how to create and manage a new SoC project. It also provides procedures on downloading, instantiating, and generating IPs. Additional topics include how to add glue logic and connect components.

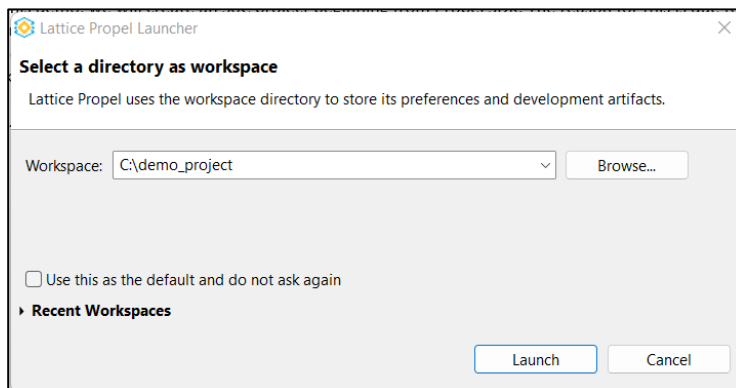
### 3.1. Creating and Managing a Project

The first step in the Propel Builder project flow is to create a new SoC project. There are two ways to create a new SoC project depending on whether the user begins with Propel Builder or Propel SDK. Both methods have essentially the same project creation options, with the main difference being how the SoC project is opened afterwards, as well as how the overall Propel project is organized.

In this demo, the SoC project is created beginning from Propel SDK. This method results in improved project organization. By creating the SoC project from Propel SDK, the Builder project files are saved to the same directory where the software project is located. This allows the user to easily track and manage the files within the project using the workspace containing the SoC and software projects.

To create a new SoC Project using Propel SDK:

1. Launch Propel SDK.
2. Select a directory to use for the project workspace by clicking **Browse**.  
The SoC and C/C++ software projects are saved in this location.
3. Click **Launch** to open Propel SDK.



**Figure 3.1. Propel SDK Workspace Selection Window**

4. Select **Create a new Lattice SoC Design Project**, as shown in [Figure 3.2](#).  
The SoC project creation window opens, as shown in [Figure 3.3](#).

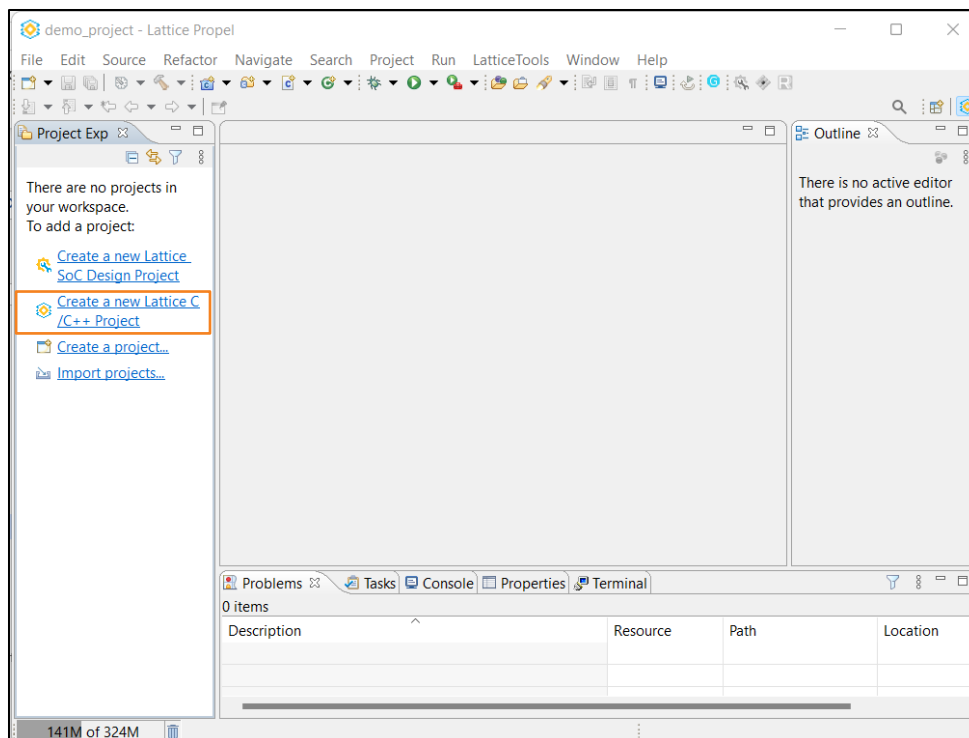


Figure 3.2. Propel SDK Workspace Environment

5. Enter a name for the SoC project in the **Project Name** field.

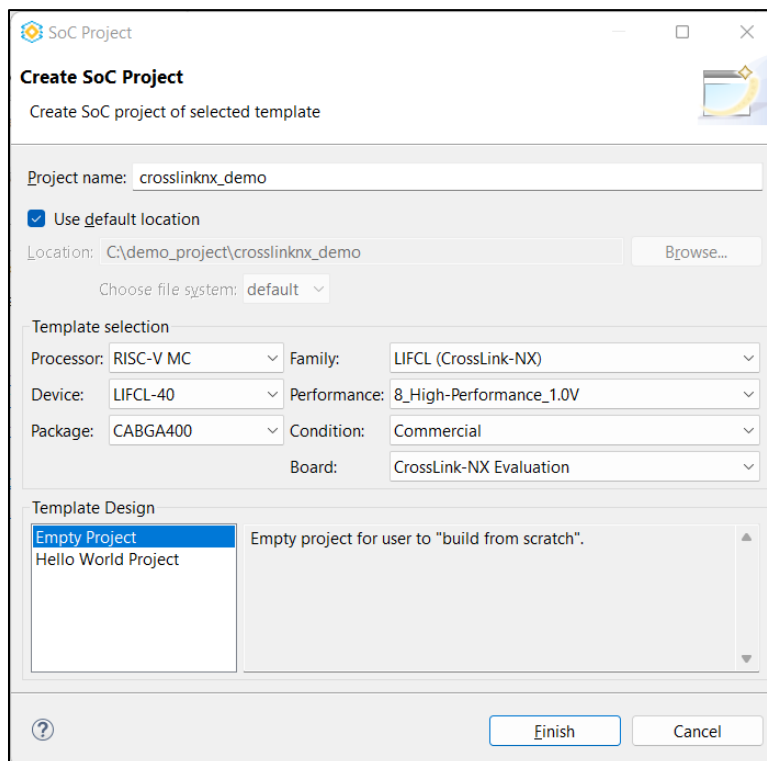


Figure 3.3. SoC Project Creation Window and Settings for the Demo

By default, the SoC project is created in the active workspace. To improve the organization of project files, select **Use default location** for this demo.

(Optional) The user can select a new location for the SoC project by disabling the **Use default location** checkbox and then clicking the **Browse** button to select a new location for the project.

6. Select the **Family**, **Package**, **Performance**, and **Condition** for the target device.

The **Board** option can be used to easily select the correct target device if targeting an evaluation, breakout, or demo board.

Board templates are evaluated on each device, come with constraints, and are guaranteed to work.

This demo targets the CrossLink-NX Evaluation Board (LIFCL-40-8BGBG400C). The user can also follow along with any Certus™-NX, CertusPro™-NX, or MachXO5™-NX board as long as user constraints are used later.

In **Processor**, select the **RISC-V MC** for the project. This controls the templates that are available for selection.

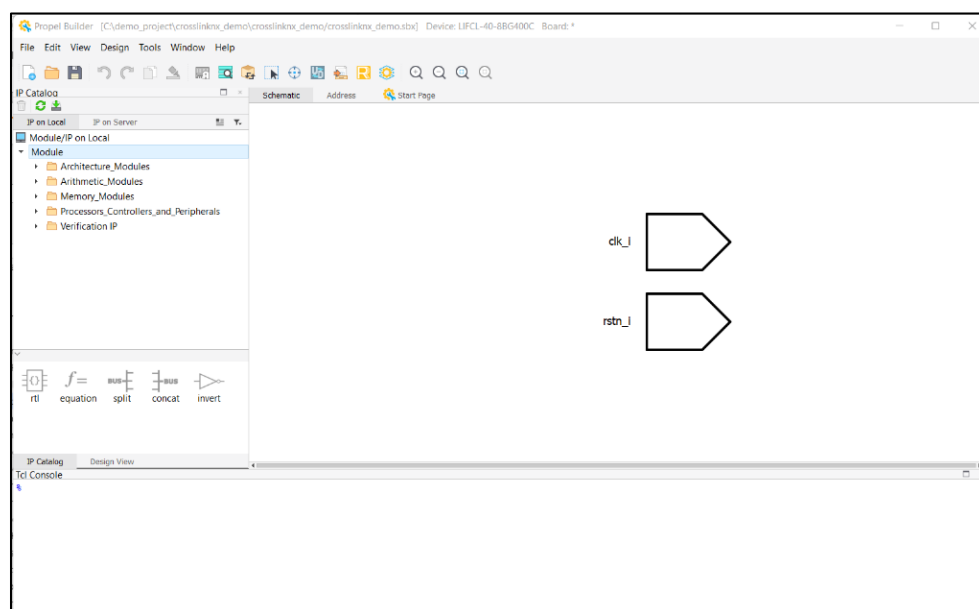
For example, if **RISC-V SM** is selected, only the templates containing that processor appear.

If creating a project from scratch, the processor selection does not matter. All **Empty Project** templates are functionally the same.

7. Select the **Empty Project** option as template under **Template Design**. The available options depend on the device and processor selections.

Some **Board** options have additional template options that are specific to that board. For example, the **Lattice Sentry™ RoT** template is only available for the Lattice Sentry Demo Board for Mach™-NX.

8. Click **Finish**. Propel Builder automatically opens the new SoC project, as shown in Figure 3.4.



**Figure 3.4. New Empty SoC Project in Propel Builder**

When the SoC project opens for the first time, it appears in the Recent Project List in the Start Page tab, as shown in Figure 3.5. The seven most recently opened projects in Propel Builder are listed regardless of the Propel version where these projects were last opened.

To quickly open a project from the Recent Project List, simply click the name of the project to open.

**Note:** Selecting File > Recent Designs > <Project Name> also works.

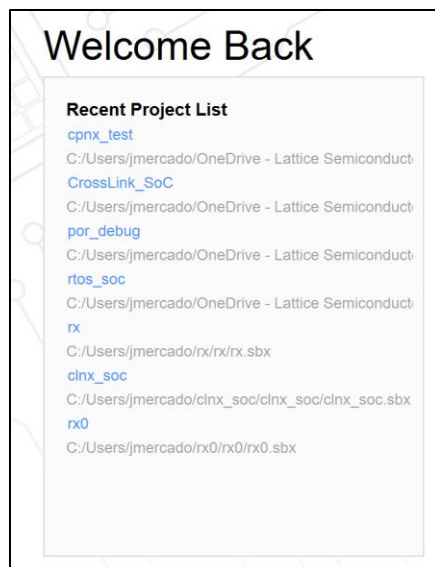


Figure 3.5. Recent Project List in Propel Builder Start Page

## 3.2. Downloading, Installing, and Generating IP

One of Propel Builder's most important features is its ability to easily integrate IP to develop projects. For this reason, it is important to understand Propel Builder's IP Catalog, as it controls the IP that can be used in projects.

If the IP Catalog is not open, switch to it by selecting the IP Catalog tab, which can be found next to the Design View tab, as shown in Figure 3.6.

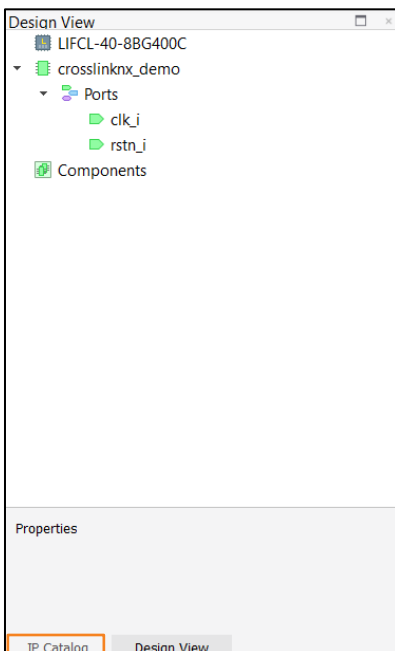
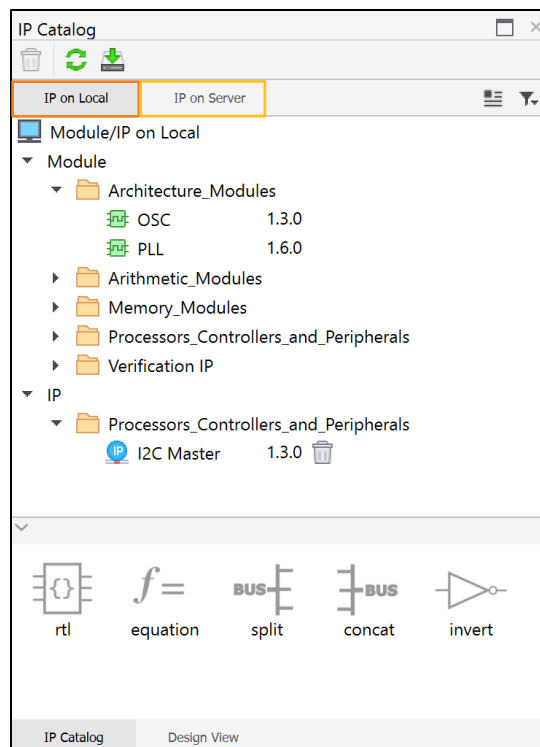


Figure 3.6. IP Catalog and Design View

Figure 3.7 shows the IP Catalog with two main tabs: IP on Local and IP on Server. The first tab, IP on Local, contains IP that is already installed and is ready for use.



**Figure 3.7. IP on Local Tab of IP Catalog**

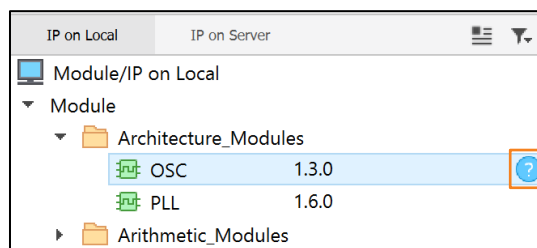
The IP within the IP on Local tab are divided into two categories: Module and IP. IP that are installed along with Propel, known as foundation IP, appear under Module. IP that are installed from the IP Catalog or using a custom *.ipk* IP package file appear under IP. Each of the IP under these two categories are sorted into different folders. To view the IP within a folder, expand the folder using the ▶ icon.

Each folder may contain several IPs. The name and version of each IP are indicated.

**Note:** Only IPs that are compatible with the target device of the active project appear in the IP on Local tab.

In the IP Catalog, an icon next to each IP name indicates if there are any issues with the IP. As shown in Figure 3.7, the blue circle next to I<sup>2</sup>C Master IP indicates that there are no issues with the IP and it can be used in the current project. In the IP on Server tab, an icon also indicates if the IP is not compatible with the current project's target device. An orange triangle with an exclamation point is used.

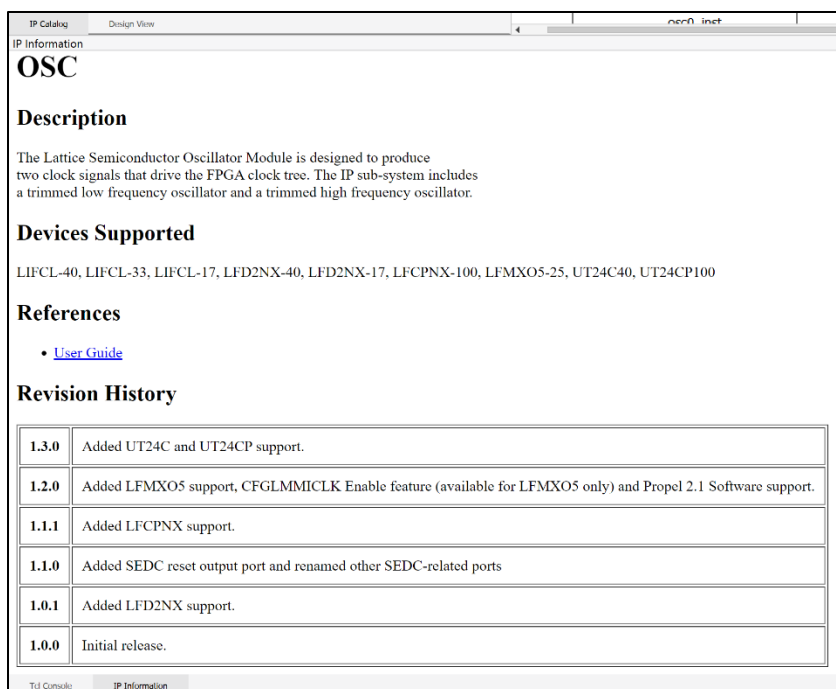
Additional information can be accessed by hovering the mouse towards the right side of an IP row to display a blue circle with a question mark, as shown in Figure 3.8.



**Figure 3.8. Additional Information Icon for Oscillator IP v1.3.0**

Click this icon to open the IP Information window at the bottom of the Propel Builder interface.

The IP Information window, as shown in Figure 3.9, provides additional information about an IP such as a brief description, supported devices, reference documents, and version history.



**Figure 3.9. Additional Information for Oscillator IP v1.3.0**

The IP on Server tab contains all the IPs available for download from Lattice’s IP Server. Similar to the IP on Local tab, this section of the IP Catalog also contains IPs that are sorted into folders depending on their type.


### 3.2.1. Downloading IP from the IP Server

For this demo, download, install, and generate the IPs for the project. If the user is familiar with the Lattice Radiant IP Catalog, the process is similar.

Download and install the most recent version of the following IPs:

- RISC-V MC
- UART
- GPIO
- Watchdog-Timer

To download and install an IP from the IP Server:

1. Click the  icon next to an IP’s name. A user license agreement prompt appears.
2. Click **Agree** to download and install the IP. Once an IP is installed, it is listed under the IP section in the **IP on Local** tab.
3. After installing all the IPs, select the **IP on Local** tab. The IPs are listed under the IP section, as shown in [Figure 3.10](#).

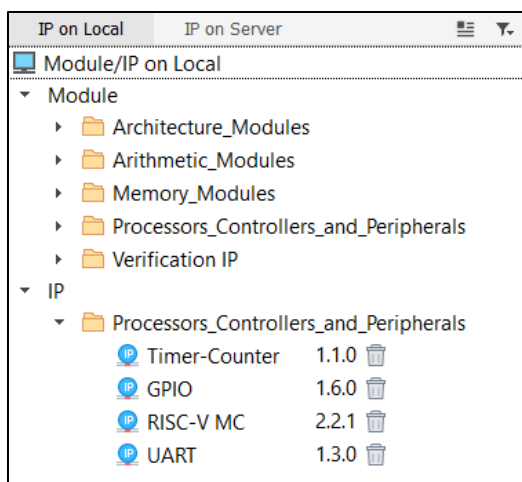


Figure 3.10. Downloaded and Installed IPs

### 3.2.2. Generating and Instantiating the IP

This section provides the procedure for generating the IPs required in this demo.

#### 3.2.2.1. Generating and Instantiating the RISC-V MC IP

To generate and instantiate the RISC-V MC IP:

1. In the IP on Local tab under the IP > Processors\_Controllers\_and\_Peripherals section, double-click RISC-V MC.
2. Enter a name for the IP. For this demo, enter **riscv\_mc0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in Figure 3.11.

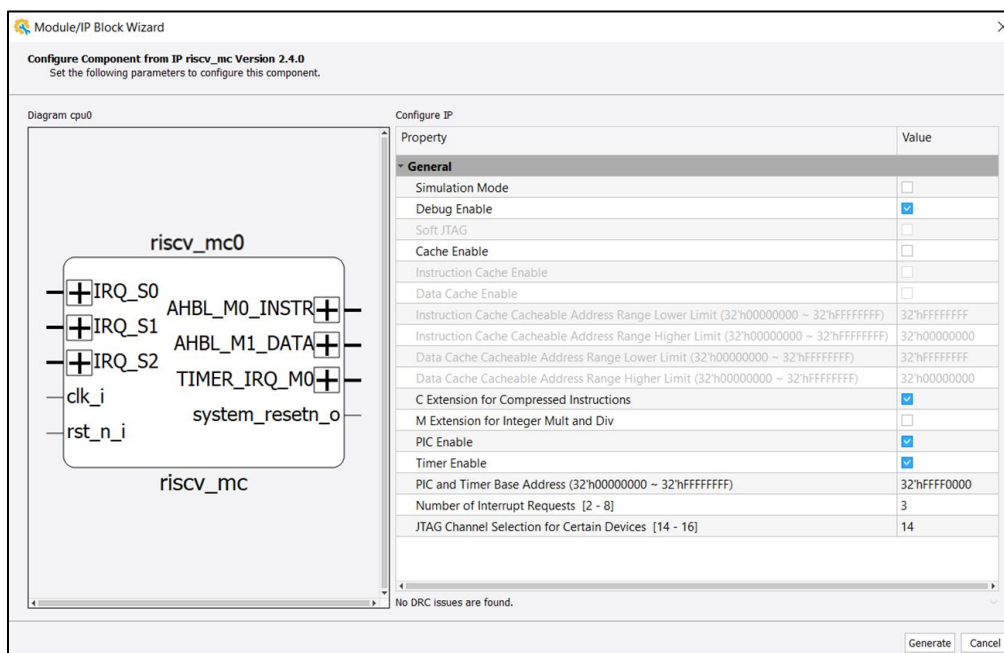


Figure 3.11. RISC-V MC Component Parameter Configuration

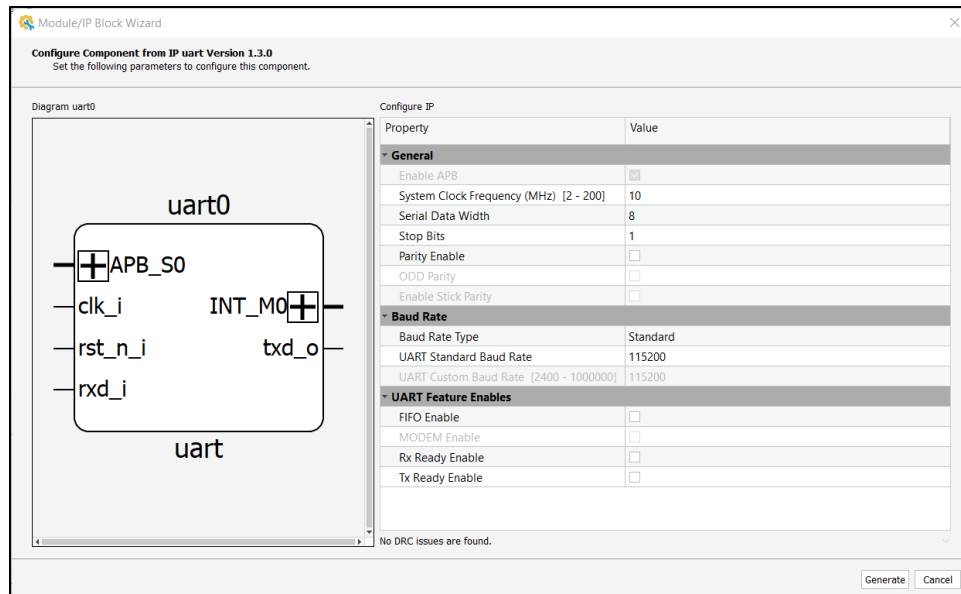
5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.

7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **riscv\_mc0\_inst**.
9. Click **OK**.

### 3.2.2.2. Generating and Instantiating the UART IP

To generate and instantiate the UART IP:

1. In the IP on Local tab under the IP > Processors\_Controllers\_and\_Peripherals section, double-click UART.
2. Enter a name for the IP. For this demo, enter **uart0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.12](#).



**Figure 3.12. UART Component Parameter Configuration**

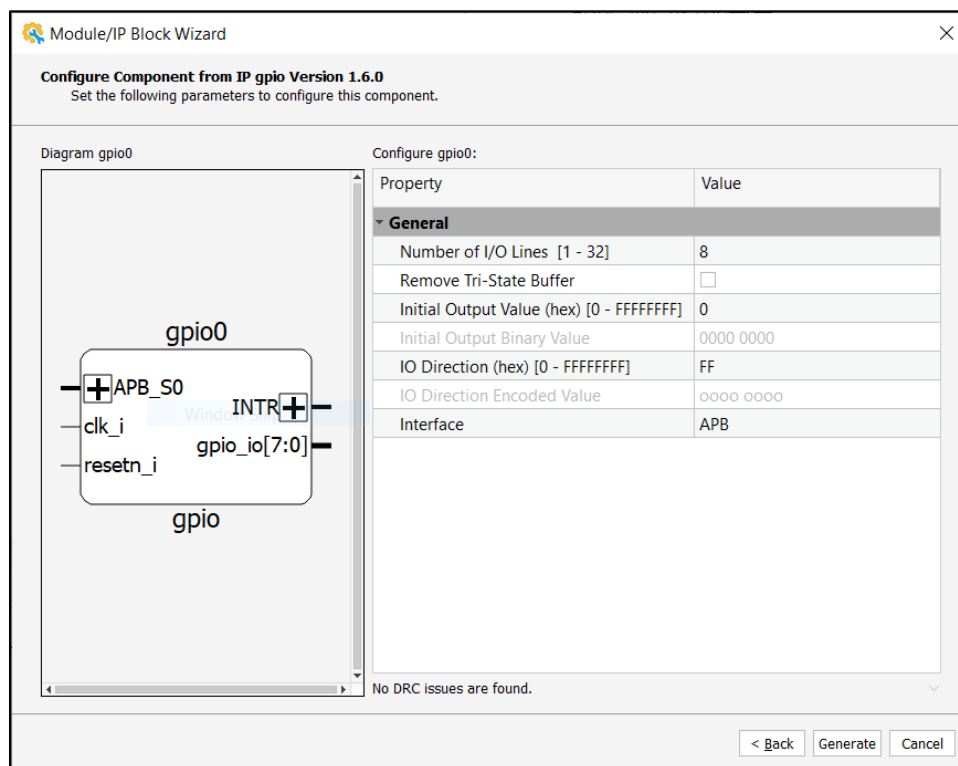
5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **uart0\_inst**.
9. Click **OK**.



### 3.2.2.3. Generating and Instantiating the GPIO IP

To generate and instantiate the GPIO IP:

1. In the IP on Local tab under the IP > Processors\_Controllers\_and\_Peripherals section, double-click GPIO.
2. Enter a name for the IP. For this demo, enter **gpio0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.13](#).



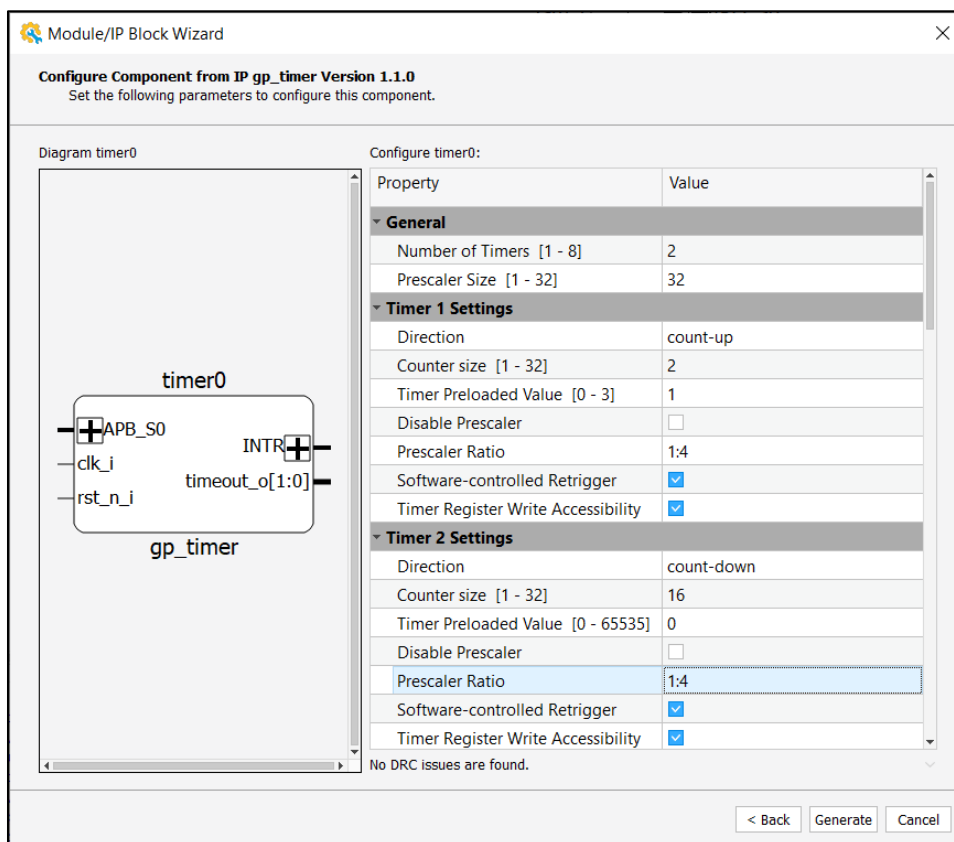
**Figure 3.13. GPIO Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **gpio0\_inst**.
9. Click **OK**.

### 3.2.2.4. Generating and Instantiating the Timer Counter IP

To generate and instantiate the Timer Counter IP:

1. In the IP on Local tab under the IP > Processors\_Controllers\_and\_Peripherals section, double-click Timer-Counter.
2. Enter a name for the IP. For this demo, enter **timer0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.14](#).



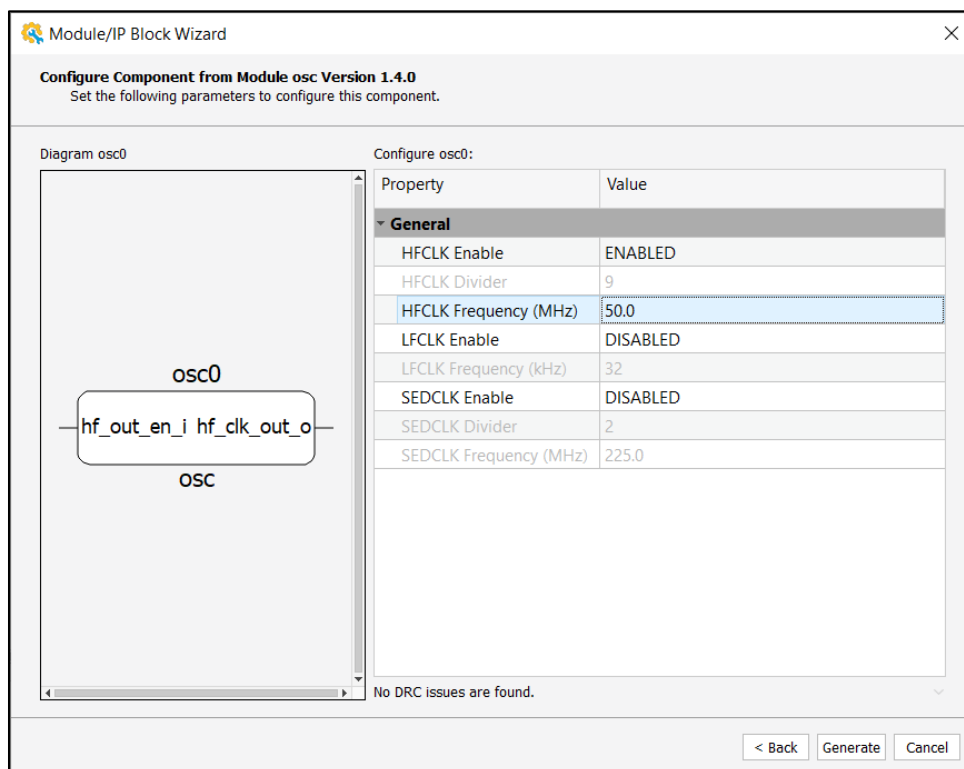
**Figure 3.14. Watchdog Timer Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **timer0\_inst**.
9. Click **OK**.

### 3.2.2.5. Generating and Instantiating the Oscillator IP

To generate and instantiate the Oscillator IP:

1. In the IP on Local tab under the Module > Architecture\_Modules section, double-click Oscillator.
2. Enter a name for the IP. For this demo, enter **osc0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.15](#).



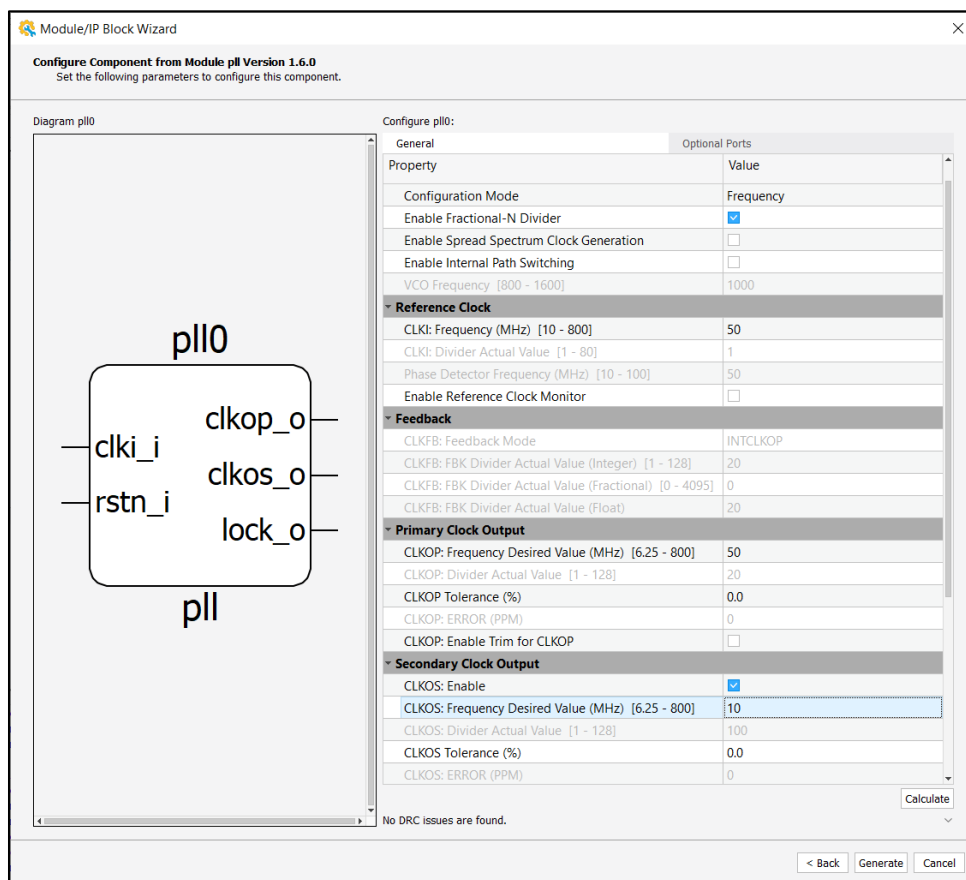
**Figure 3.15. Oscillator Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **osc0\_inst**.
9. Click **OK**.

### 3.2.2.6. Generating and Instantiating the PLL IP

To generate and instantiate the PLL IP:

1. In the **IP on Local** tab under the **Module > Architecture\_Modules** section, double-click **PLL**.
2. Enter a name for the IP. For this demo, enter **pll0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.16](#).



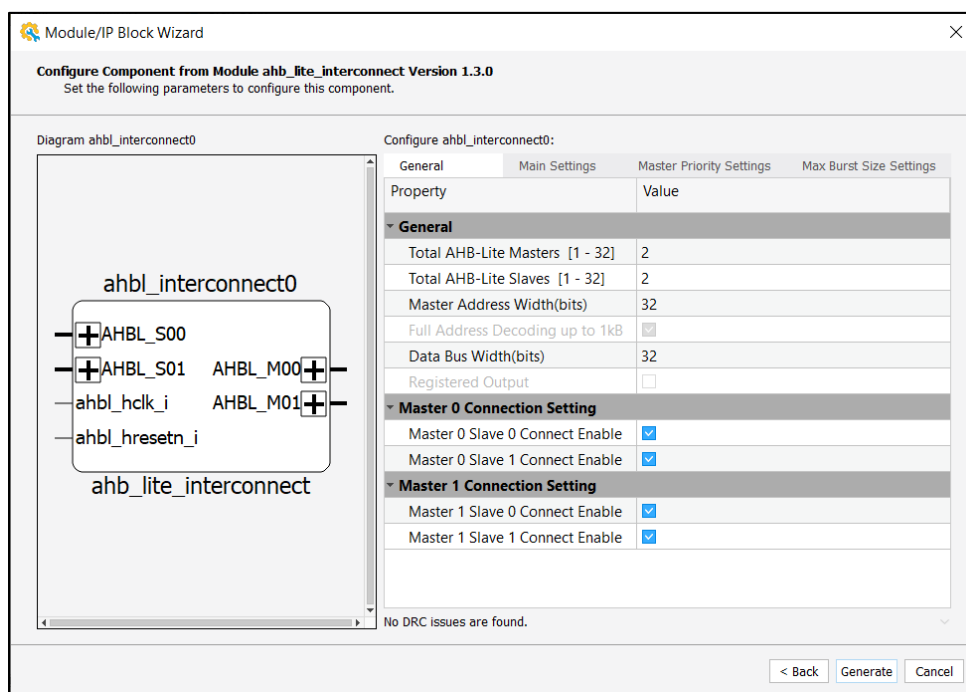
**Figure 3.16. PLL Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **pll0\_inst**.
9. Click **OK**.

### 3.2.2.7. Generating and Instantiating the AHB-Lite Interconnect IP

To generate and instantiate the UART IP:

1. In the IP on Local tab under the Module > Processors\_Controllers\_and\_Peripherals section, double-click AHB-Lite Interconnect.
2. Enter a name for the IP. For this demo, enter **ahbl\_interconnect0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.17](#).



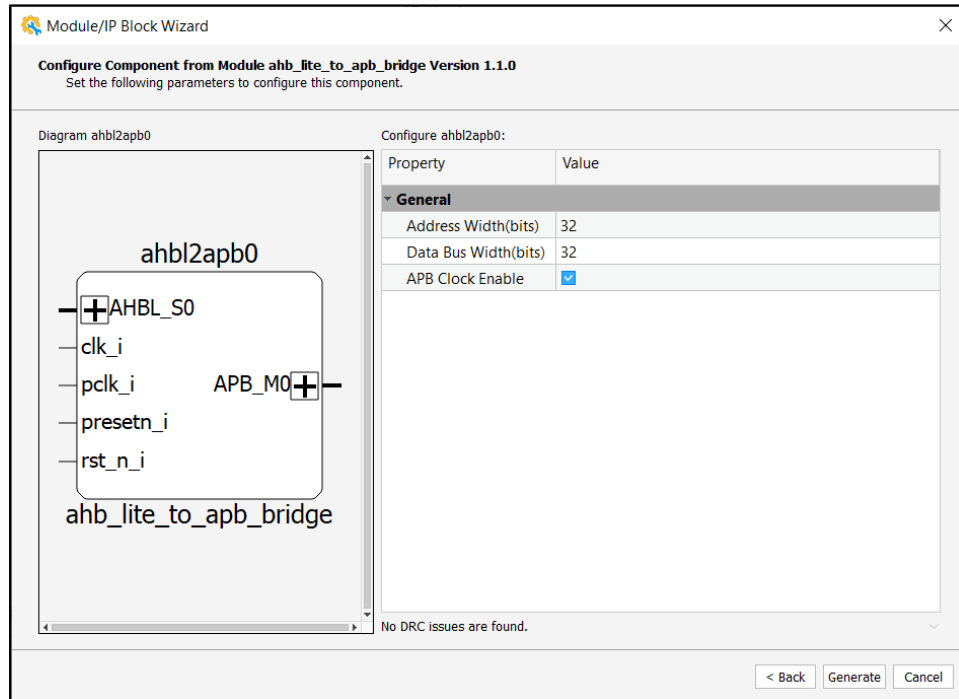
**Figure 3.17. AHB-L interconnect Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **ahbl\_interconnect0\_inst**.
9. Click **OK**.

### 3.2.2.8. Generating and Instantiating the AHB-L to APB Bridge IP

To generate and instantiate the AHB-L to APB Bridge IP:

1. In the IP on Local tab under the Module > Processors\_Controllers\_and\_Peripherals section, double-click AHB-Lite to APB Bridge.
2. Enter a name for the IP.  
For this demo, enter **ahbl2apb0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.18](#).



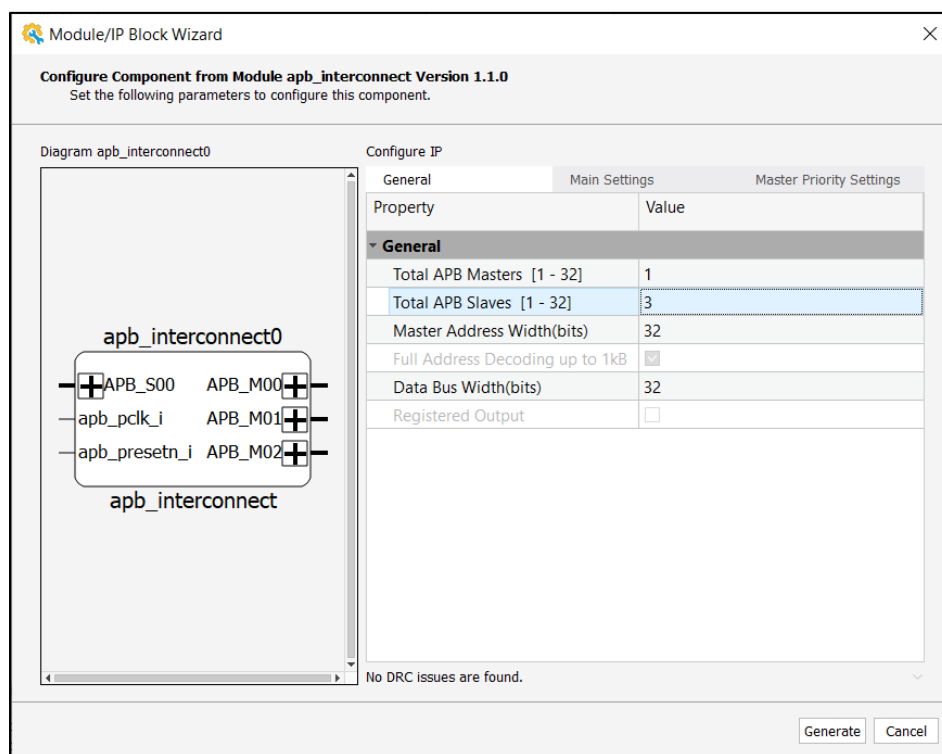
**Figure 3.18. AHB Lite to APB Bridge Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **ahbl2apb0\_inst**.
9. Click **OK**.

### 3.2.2.9. Generating and Instantiating the APB Interconnect IP

To generate and instantiate the APB Interconnect IP:

1. In the IP on Local tab under the Module > Processors\_Controllers\_and\_Peripherals section, double-click APB Interconnect.
2. Enter a name for the IP. For this demo, enter **apb\_interconnect0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.19](#).



**Figure 3.19. APB Interconnect Component Parameter Configuration**

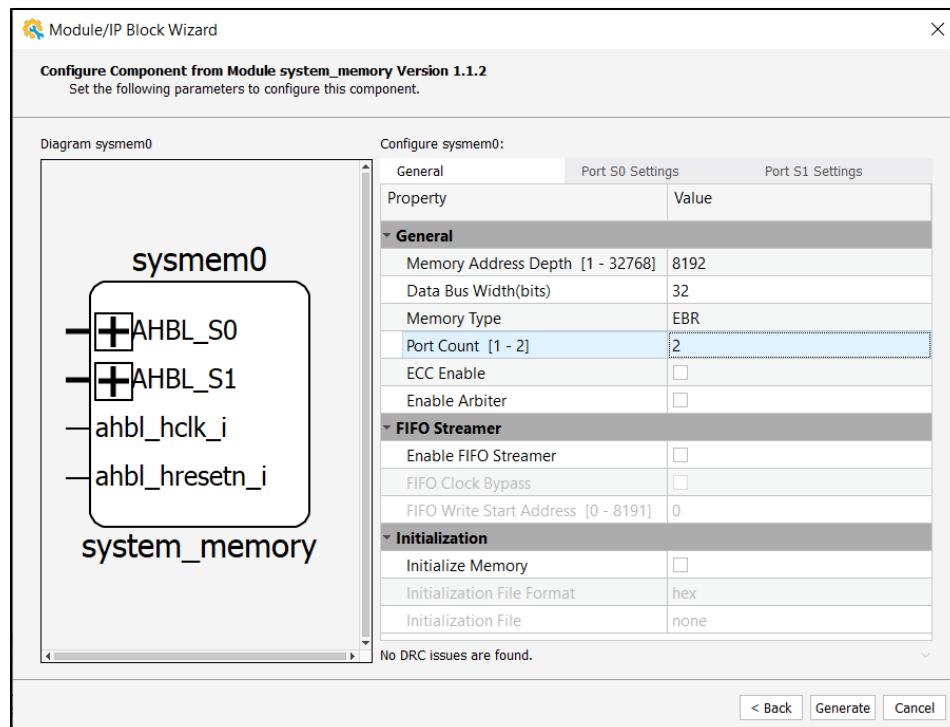
5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **apb\_interconnect0\_inst**.
9. Click **OK**.

### 3.2.2.10. Generating and Instantiating the System Memory IP

The system memory component is a fundamental building block for most Propel projects and is one method for initializing memory in the project.

To generate and instantiate the System Memory IP:

1. In the IP on Local tab under the Module > Processors\_Controllers\_and\_Peripherals section, double-click System Memory.
2. Enter a name for the IP. For this demo, enter **sysmem0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.20](#).



**Figure 3.20. System Memory Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **sysmem0\_inst**.
9. Click **OK**.

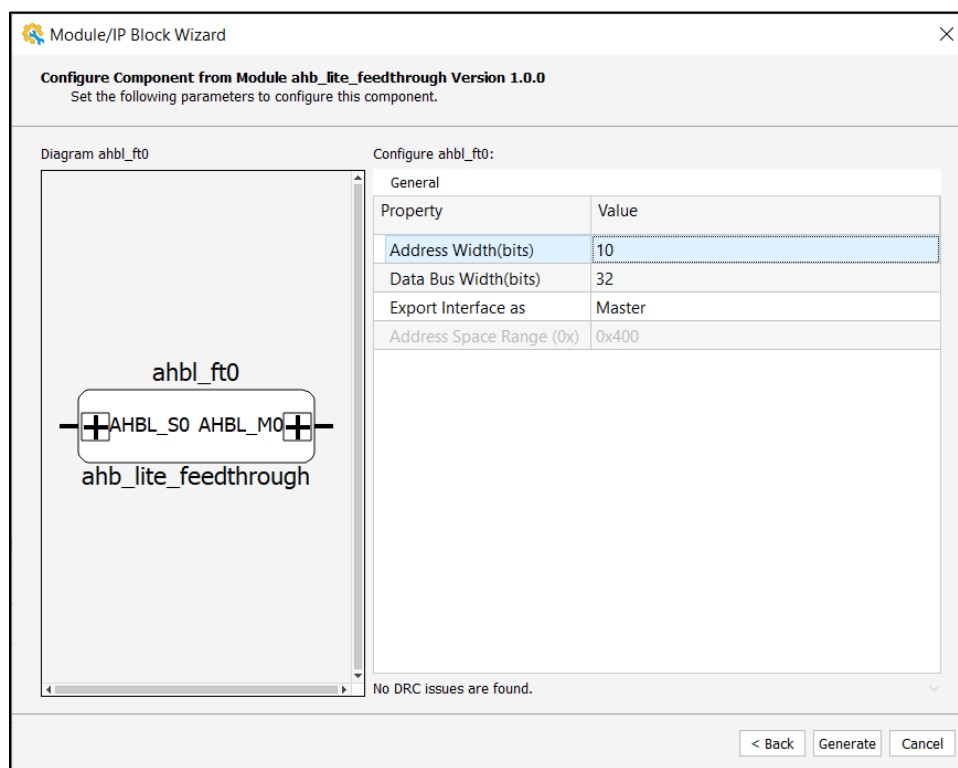


### 3.2.2.11. Generating and Instantiating the AHB-Lite Feedthrough IP

The final component to generate is the AHB-Lite Feedthrough IP. The purpose of the AHB-L feedthrough is to allow the user to connect to an external master or slave component of the same type. For this demo, the user connects this component to an external slave in the simulation environment, so the user needs to generate this component as a master.

To generate and instantiate the AHB-Lite Feedthrough IP:

1. In the IP on Local tab under the Module > Processors\_Controllers\_and\_Peripherals section, double-click AHB Lite Feedthrough.
2. Enter a name for the IP. For this demo, enter **ahbl\_ft0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 3.21](#).



**Figure 3.21. AHB Lite Feedthrough Component Parameter Configuration**

5. Click **Generate**.
6. Ensure that the **Insert to Project** option is enabled.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **ahbl\_ft0\_inst**.
9. Click **OK**.

## 3.3. Adding Logic and Connecting Components

### 3.3.1. Adding Glue Logic

Figure 3.22 shows the additional section in IP Catalog at the bottom of the IP on Local tab. This section contains the glue logic that can be added for Propel projects. To add a specific type of glue logic, double-click the icon corresponding to the type of glue logic to generate.

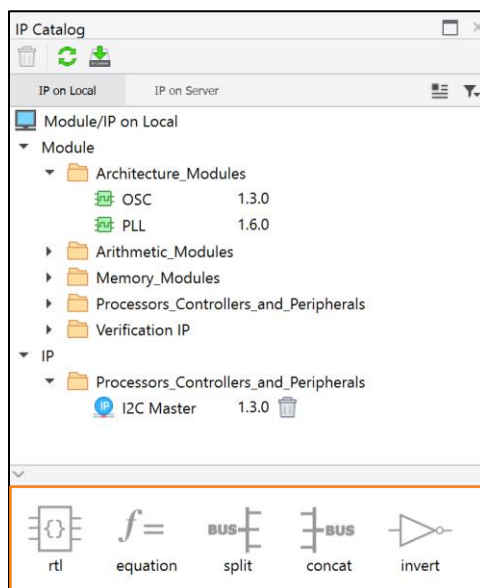


Figure 3.22. Glue Logic Section in IP on Local

There are five main types of glue logic, which can be used to add logic to the projects in different ways:

- RTL – add a Verilog or VHDL module to the project. The user can add the module's code directly or link the source file containing the module to add. Before adding a module to the project, ensure there are no syntax errors. Otherwise, the added component does not work correctly.
- Equation – add combinational logic to the project. The inputs to the combinational logic should be variables, which correspond to the ports that can be connected in the design.
- Split – allows the user to split a bus with multiple bits into several smaller output buses or wires that can be connected in the project.
- Concat – allows the user to combine signals of varying widths into a single larger bus.
- Invert – inverts a single 1-bit input.

For this demo, the split, invert, and equation glue logic components are used to add some additional functionality to the design.

To add glue logic to the project:

1. Double-click the **Split** icon in the glue logic area.
2. Set the **LHS** and **RHS** parameters as shown in Figure 3.23.
3. Click the **Remove** button to remove any additional rows from the RHS.

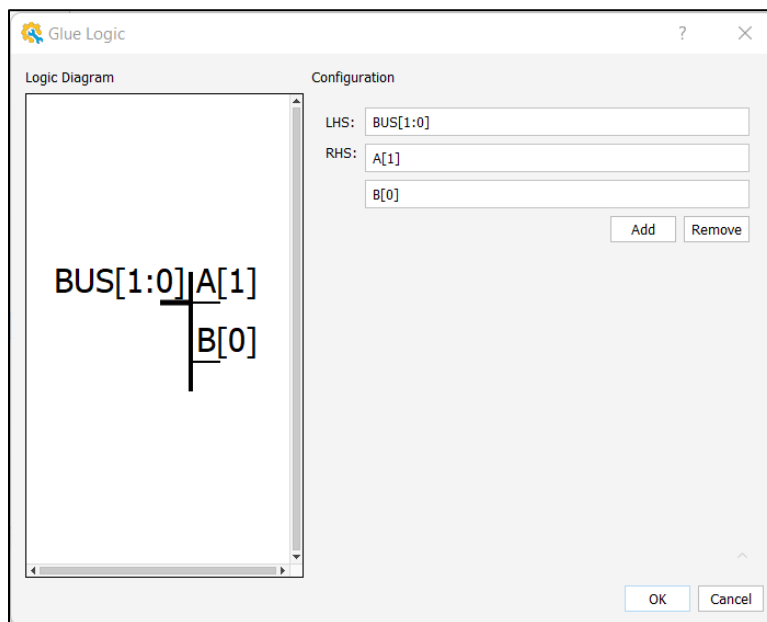


Figure 3.23. Split Glue Logic Configuration

4. Double-click the **Equation** icon in the glue logic area.
5. Set **Expression** as shown in Figure 3.24.

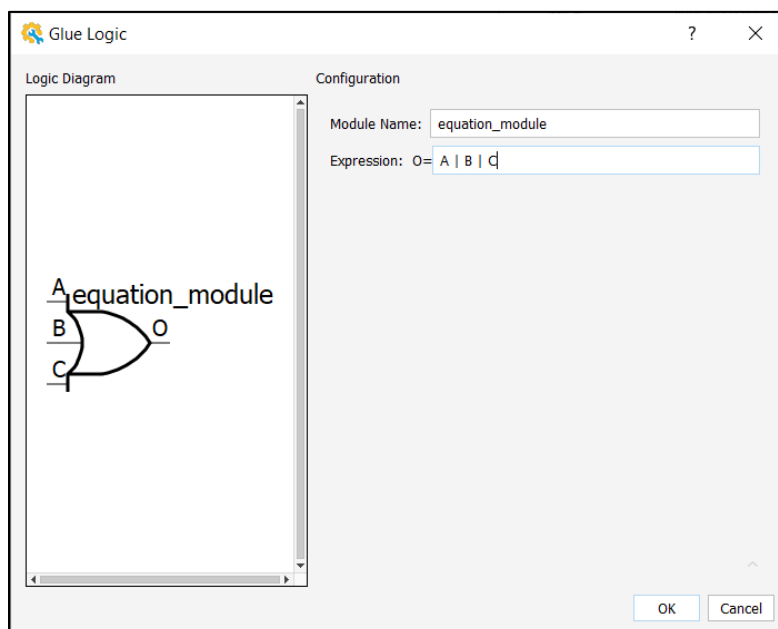
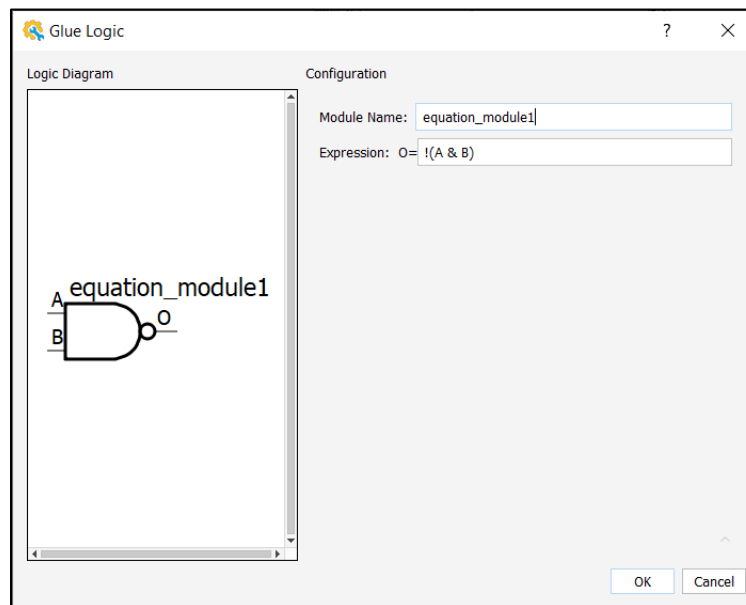


Figure 3.24. First Equation Glue Logic Configuration

6. Again, double-click the **Equation** icon in the glue logic area.

7. Set **Expression** as shown in Figure 3.25.



**Figure 3.25. First Equation Glue Logic Configuration**

8. Double-click the **Invert** icon from the glue logic area. An inverting glue logic component is automatically added to the schematic.
9. Click **OK**.

### 3.3.2. Manually Connecting Components

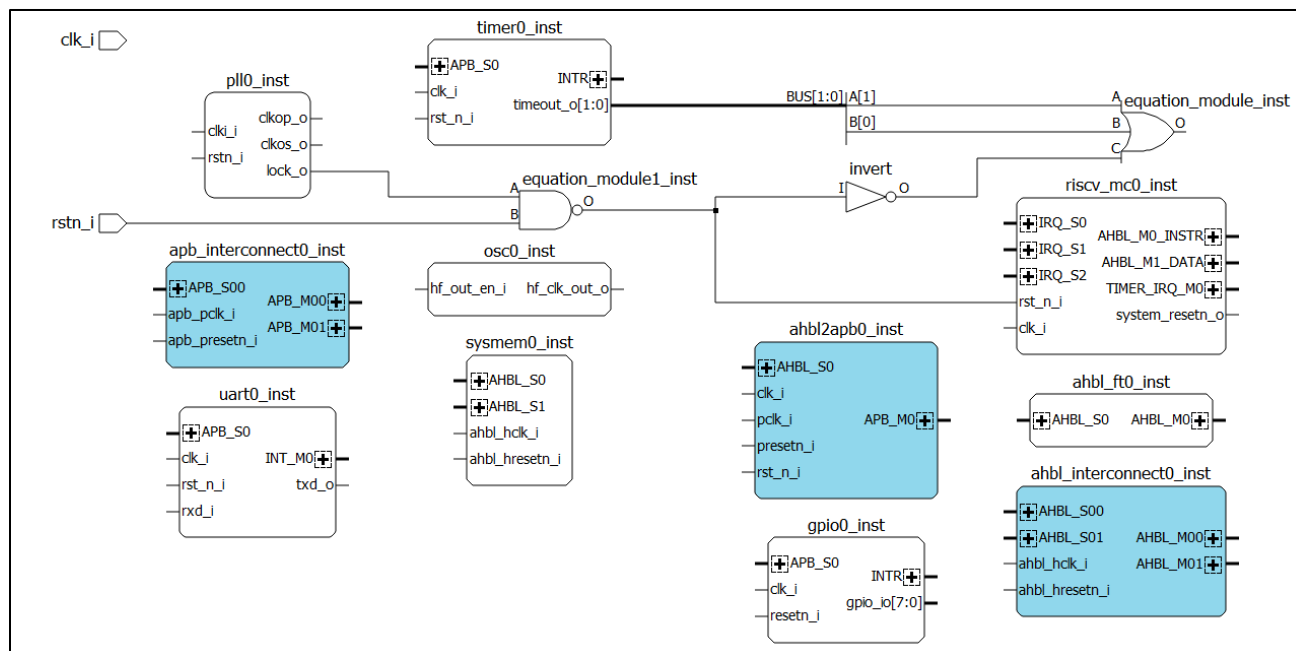
To manually connect two components in the design, hover the mouse over the port from which to connect and click when the pencil icon appears. Once a port is selected, the pencil icon should remain. Click the destination port.

Propel Builder's manual connection tool can be used to make multiple connections at the same time. Note that after making a connection, the pencil icon does not disappear until the user right-clicks or selects a blank location in the schematic view. Instead of clicking to remove the pencil icon, the user can continue making connections to the same type of ports by clicking the name of the destination port to connect.

**Important:** Only ports of the same type and bit width can be connected. In Propel, the two basic types of ports are data and interface. Only interfaces of the same type (AHB-L with AHB-L, 8-bit input with 8-bit output, and others) are valid connections. Any other component connections are not possible. Similarly, the user cannot split an interface signal to make connections with other regular ports. In this case, use a module to convert the interface signals to a regular signal data type. Before proceeding, make the following connections by manually clicking the source port (left side) with the destination port (right side):

- rstn\_i > equation\_module1/B
- pll0\_inst/lock\_o > equation\_module1/A
- equation\_module1/O > riscv\_mc0\_inst/rst\_n\_i
- equation\_module1/O > invert/I
- timer0\_inst/timeout\_o[1:0] > BUS[1:0]
- A[1] > equation\_module\_inst/A
- B[0] > equation\_module\_inst/B
- invert/O > equation\_module\_inst/C

Figure 3.26 shows the schematic view after connecting the components. Click the disconnected clk\_i port and press DEL on the keyboard to delete the unused port.

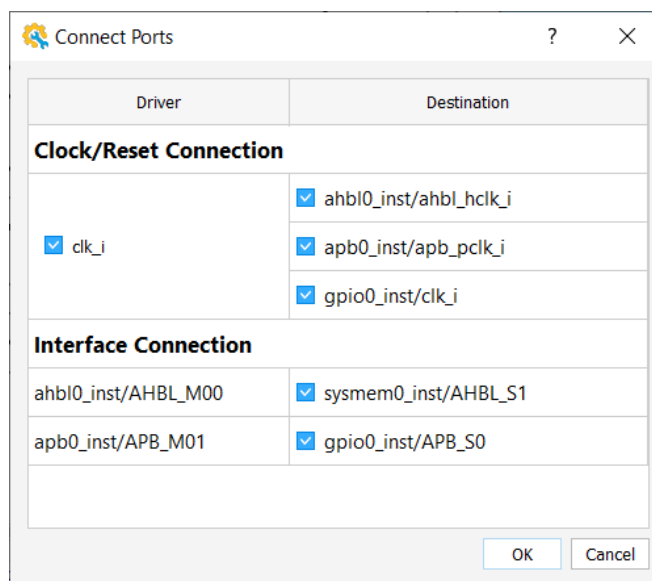


**Figure 3.26. Schematic View after Connecting Glue Logic Components**

### 3.3.3. Automatically Connecting Components

Propel Builder's auto-connect functionality is used to connect the clock, reset, and interface ports within a design.

Figure 3.27 shows the Connect Ports window divided by the types of connections that can be made: the Clock/Reset Connection and the Interface Connection.



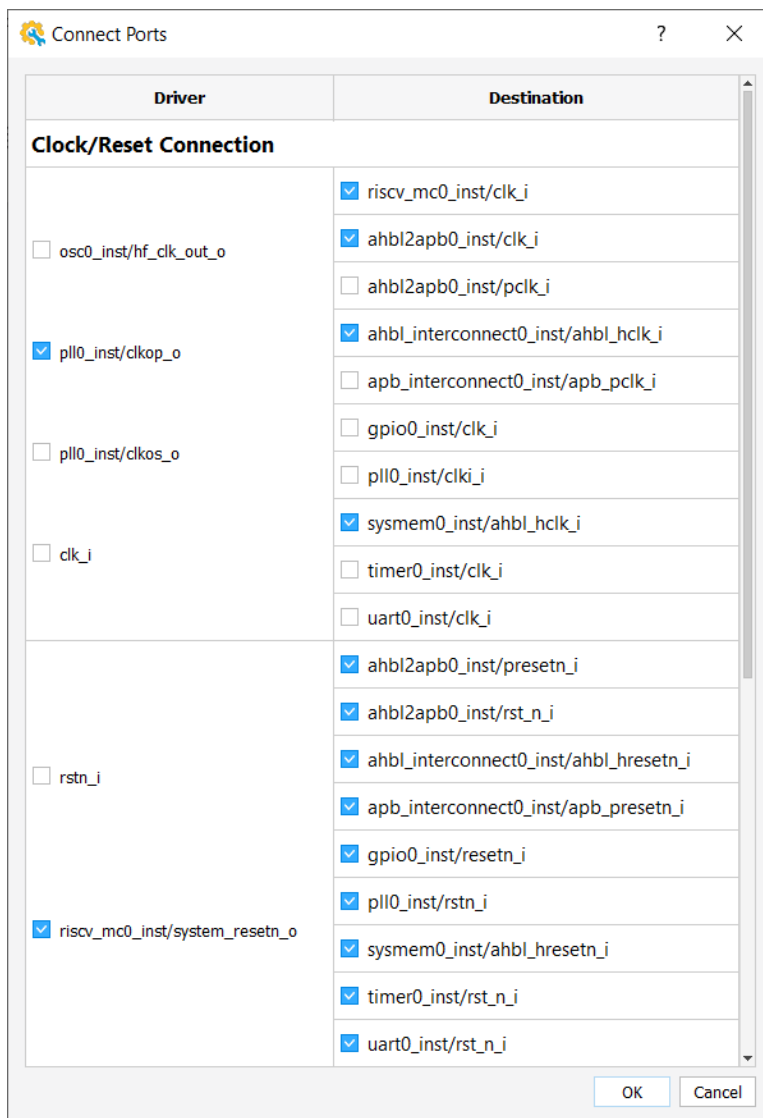
**Figure 3.27. Connect Ports Window to Select Components to Connect**

Depending on the unconnected ports in the design, auto-connect may recommend various connections of either clock/reset or interface types. On the left side of the Connect Ports window are the available drivers. Use the checkboxes next to each driver port's name to determine whether or not to make that type of connection. Similarly, if multiple destination ports are recommended, the checkboxes next to the names of each destination port should be used to select which connections to automate.

Since the recommended connections are automatically generated by auto-connect, it is important to confirm each connection before clicking OK. In some cases, auto-connect may not recognize all the correct connection to make or may not recommend any connections at all for some unconnected components. In such cases, the user needs to make these connections manually.

To use auto-connect to connect components:

1. Right-click anywhere in schematic view and select **Auto Connect**.
2. Compare the auto-connect window with [Figure 3.28](#) and [Figure 3.29](#).



**Figure 3.28. Initial Interface Connections to Make**

Interface Connection	
riscv_mc0_inst/AHBL_M1_DATA	<input checked="" type="checkbox"/> ahbl2apb0_inst/AHBL_S0
ahbl2apb0_inst/APB_M0	<input checked="" type="checkbox"/> apb_interconnect0_inst/APB_S00
ahbl_interconnect0_inst/AHBL_M00	<input type="checkbox"/> ahbl_ft0_inst/AHBL_S0
ahbl_interconnect0_inst/AHBL_M01	<input checked="" type="checkbox"/> sysmem0_inst/AHBL_S0
apb_interconnect0_inst/APB_M00	<input checked="" type="checkbox"/> gpio0_inst/APB_S0
apb_interconnect0_inst/APB_M01	<input checked="" type="checkbox"/> timer0_inst/APB_S0
apb_interconnect0_inst/APB_M02	<input checked="" type="checkbox"/> uart0_inst/APB_S0
gpio0_inst/INTR	<input checked="" type="checkbox"/> riscv_mc0_inst/IRQ_S0
timer0_inst/INTR	<input checked="" type="checkbox"/> riscv_mc0_inst/IRQ_S1
uart0_inst/INT_M0	<input checked="" type="checkbox"/> riscv_mc0_inst/IRQ_S2

Figure 3.29. Initial Interface Connections to Make

### 3.3.4. Making Additional Connections

Auto connect does not make all the project connections for the user. The following additional connections should be made.

- osc0\_inst/hf\_clk\_out\_o > pll0\_inst/clk\_i
- riscv\_mc0\_inst/AHBL\_M0\_INSTR > sysmem0\_inst/AHBL\_S1
- riscv\_mc0\_inst/AHBL\_M1\_DATA > ahbl\_interconnect0\_inst/AHBL\_S00
- ahbl\_ft0\_inst/AHBL\_M0 > ahbl\_interconnect0\_inst/AHBL\_S01
- ahbl\_interconnect0\_inst/M00 > ahbl2apb0\_inst/AHBL\_S0
- pll0\_inst/clkos\_o > remaining unconnected clocks for APB components:
  - gpio0\_inst/clk\_i
  - uart0\_inst/clk\_i
  - timer0\_inst/clk\_i
  - ahbl2apb0\_inst/apb\_pclk\_i
  - apb\_interconnect0\_inst/apb\_pclk\_i

Figure 3.26 shows the schematic view after making additional connections.

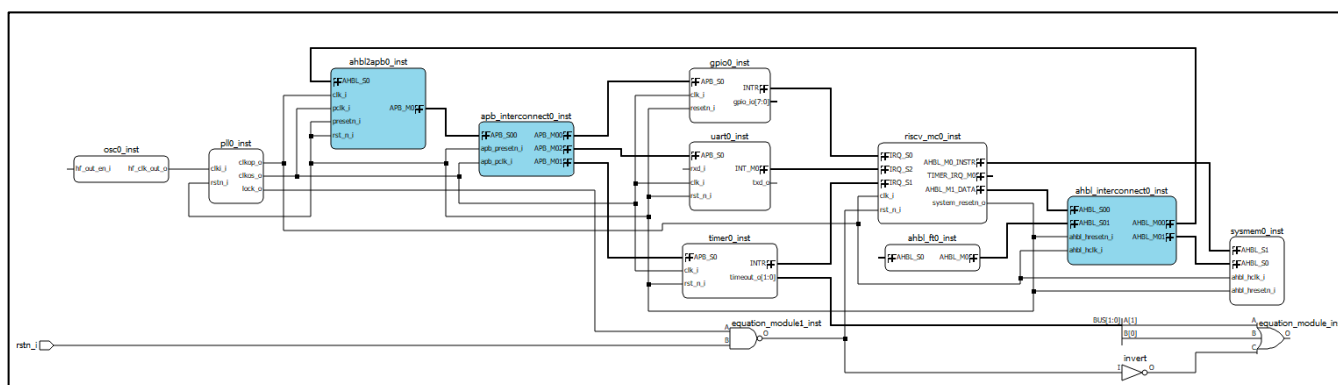


Figure 3.30. Schematic View with Additional Connections

### 3.3.5. Creating Top-Level Ports

After the basic components in the project are connected, the next step is to create the top-level ports. Ensure that the names of the generated ports match the ones from this demo, as the constraint file uses these names for reference.

To manually create a top-level port:

1. Right-click anywhere in the schematic view.
2. Select Create Port.
3. In the **Create Port** dialog box, define the basic properties for the new port.
  - Name – debug\_o
  - Direction – Output
  - Type – General

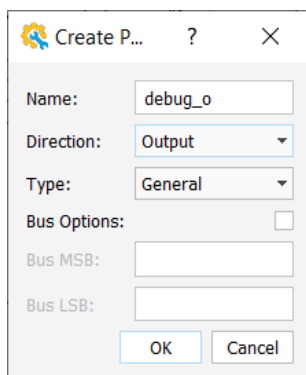


Figure 3.31. Create Port Settings for debug\_o

4. Click **OK**.
5. Repeat steps 1 to 4 to create two additional ports called **rx\_d\_i** and **tx\_d\_o**.
  - rx\_d\_i: 1-bit input
  - tx\_d\_o: 1-bit output
6. Connect the newly created ports:
  - equation\_module\_inst/O > debug\_o
  - uart0\_inst/tx\_d\_o > tx\_d\_o
  - rx\_d\_i > uart0\_inst/rx\_d\_i

### 3.3.6. Exporting Ports

In Propel Builder, the unconnected ports of any component can automatically be exported. The tool automatically generates top-level ports for the unconnected components in the design, and then connect those new ports with the actual ports of the source component.

To export component ports:

1. Right-click the **gpio0\_inst** component. A single port can be exported also by right-clicking the port instead of the component.
2. Select **Export**. The unconnected ports are exported. The name of the ports, however, need to be corrected, as shown in Figure 3.32.

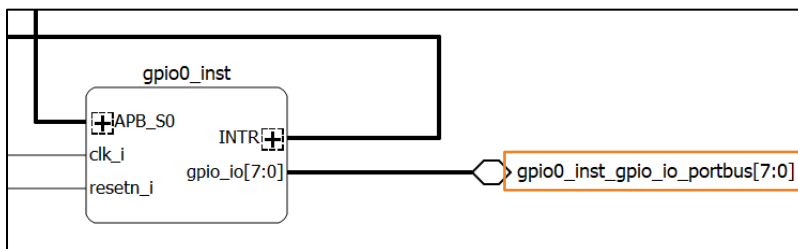
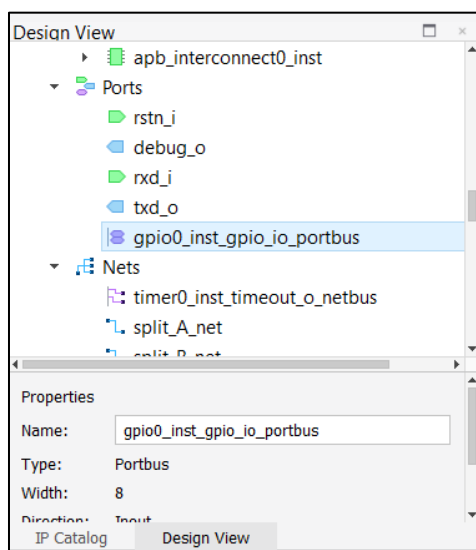


Figure 3.32. Exported Port for GPIO Component



3. To correct the name of the exported port, click the **Design View** tab.
4. Click the exported port in schematic view. The corresponding port is highlighted in **Design View** as shown in Figure 3.33.
5. In the **Name** field under **Properties** section, change the name to **led\_o**.



**Figure 3.33. Component Name Field in Design View**

6. Press **Enter** on the keyboard to confirm the port's new name.

Each port, net, and signal in the SoC design has a name, which is displayed in the Design View tab. These connection names are to be used in the generated Verilog wrapper for the Propel Builder project. Each net, port, or components name can be modified in the Design View tab as described in step 5.

The user can export a component interface so that it can be connected elsewhere, use the AHB-L or APB Feedthrough components.

To export a component interface:

1. Generate the correct feedthrough component.
2. Connect the master or slave feedthrough to the component with the interface to export.
3. Export the ports for the feedthrough component.

The exported ports appear in the project address space.

### 3.3.7. Assigning Constant Values to a Port

The final task for developing the SoC design is to assign a constant value of one to the input port for our Oscillator IP. To assign a constant value to the input port:

1. Right-click the **hf\_out\_en\_i** port of the Oscillator IP and select **Assign Constant Value**.
2. In the prompt, enter **1** to assign a constant value of 1 to the port input, as shown in Figure 3.34.

**Note:** Ensure that the oscillator is always enabled.

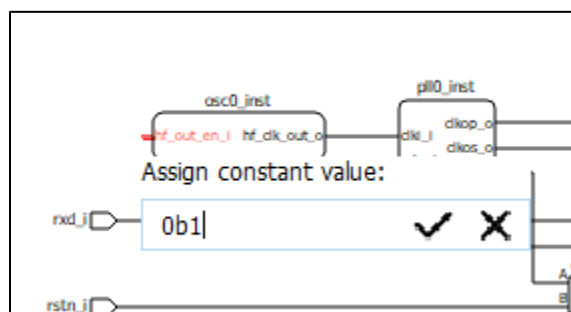


Figure 3.34. Assigning Constant Value to Oscillator Input

Figure 3.35 shows the final schematic view for the SoC project.

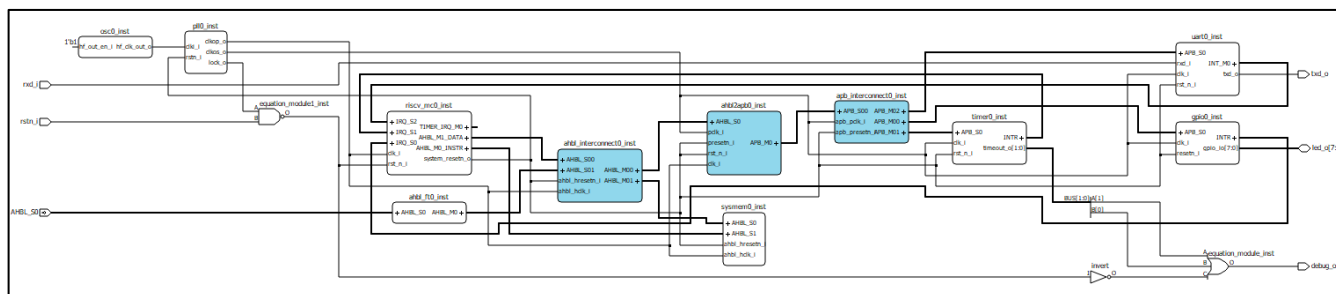


Figure 3.35. Final SoC Design

### 3.4. Managing Memory Space

After adding all the components and connections for the design, the final task in the Propel Builder project flow is to check the project's memory space allocation. As components and connections are added to the design, the memory space is automatically updated. This ensures that there are no overlapping memory spaces. The user can also manually manage memory spaces.

View the project's memory space by selecting the Address tab. Figure 3.36 shows the Address tab in Propel Builder.

Schematic		Address		Start Page	
Cell		Base Address	Range	End Address	Lock
▼ ahbl_ft0_inst					
▼ ahbl_ft0_inst/AHBL_M0(32 address bits: 4G)					
gpio0_inst/APB_S0		0x00008000	1K	0x000083FF	<input type="checkbox"/>
sysmem0_inst/AHBL_S0		0x00000000	32K	0x00007FFF	<input type="checkbox"/>
timer0_inst/APB_S0		0x00008400	1K	0x000087FF	<input type="checkbox"/>
uart0_inst/APB_S0		0x00008800	1K	0x00008BFF	<input type="checkbox"/>
▼ riscv_mc0_inst					
▼ LocalMemory					
riscv_mc0_inst/pic_timer_registers		0xFFFF0000	2K	0xFFFF07FF	
▼ riscv_mc0_inst/AHBL_M0_INSTR(32 address bits: 4G)					
sysmem0_inst/AHBL_S1		0x00000000	32K	0x00007FFF	<input type="checkbox"/>
▼ riscv_mc0_inst/AHBL_M1_DATA(32 address bits: 4G)					
gpio0_inst/APB_S0		0x00008000	1K	0x000083FF	<input type="checkbox"/>
sysmem0_inst/AHBL_S0		0x00000000	32K	0x00007FFF	<input type="checkbox"/>
timer0_inst/APB_S0		0x00008400	1K	0x000087FF	<input type="checkbox"/>
uart0_inst/APB_S0		0x00008800	1K	0x00008BFF	<input type="checkbox"/>

0x00000000

...em0\_inst/AHBL\_S0

0x00007FFF

0x00008000

0x000083FF

0x00008400

0x000087FF

0x00008800

0x00008BFF

gpio0\_inst/APB\_S0

timer0\_inst/APB\_S0

uart0\_inst/APB\_S0


Figure 3.36. Address Tab


The Address tab contains two main sections. On the right side of the screen is a visual representation for the address block allocation for a selected component address space. As the address space for an interface is modified, this interface is also updated accordingly.

The left section of the interface contains the data for the project address space allocation. This section contains all the master-slave interface connections. In this area, the user can modify the following settings:

- Base Address in the address block
- Lock of address space settings

Modifying a component's Base Address updates its End Address according to the component's range. Typically, a component's address range is determined by some parameter within the component or is pre-defined depending on the component. The second setting, *Lock*, can be used to lock a component's memory space allocation, so it cannot be manually or automatically updated.

To automatically assign the memory space for the project, click the  button in Propel Builder toolbar.

After allocating the project memory space, perform a DRC to confirm that there are no issues with the address space. To do this, enter sbp\_design drc into the Propel Builder TCL console. The user can also click the  button from the toolbar. Both methods create an error report if there are issues with the project memory space.

### 3.4.1. Propel Builder TCL Commands

While using Propel Builder, the user may have noted that almost every action in the Builder user interface outputs a TCL command to its interactive TCL console. These commands can be used in individual TCL scripts to replicate these various actions in Propel Builder. The sample TCL script in the demo project files is used to regenerate the entire SoC project from scratch.

To try out the script, change the `project_location` variable in line 2 to match the directory of this demo project. This step is crucial because the script uses the source files for each component in this project to regenerate components in the new Propel Builder project. Propel Builder has no TCL commands to generate IP from the IP Catalog. The tool can only be used to instantiate an existing IP.

The user can also modify the `new_proj_name` and `new_proj_location` variables in lines 3 and 4 to change the name and location of the new project.

The final modifications to make in the script are in the following 11 lines so that each variable matches the name of the component in the SoC design. For example, if the UART component is called `my_uart`, modify line 6 of the script to match the name of the UART component, so the script correctly instantiates the correct component.

After modifying the script, execute it using the `source` command, followed by the location of the TCL script in the Propel Builder TCL console. For example, if the `reconstruct.tcl` script is saved in the *Downloads* folder, enter `source C:/users/john/Downloads/reconstruct.tcl` in the Propel Builder's TCL console to execute the script.


The user is not required to use the interactive TCL console in the Propel Builder user interface to invoke TCL commands and scripts. To open a separate command line instance of the Propel Builder TCL console, double-click the `propelbld.exe` file located at `propel/2.2/builder/rtf/bin/nt64`. This opens a standalone TCL console for Propel Builder, allowing the user to use Propel Builder TCL commands separately from the Propel Builder user interface.


## 4. Radiant Project Flow

This section describes how to generate the Verilog wrapper file for the design and export it to either Lattice Radiant or Diamond. For this demo, the user exports the project to Lattice Radiant, since our target device is a CrossLink-NX evaluation board.

**Note:** The overall process flow is similar for Diamond projects.

### 4.1. Generating Wrapper and Exporting Project

To generate the Verilog wrapper file for the project, click the  button in the Propel Builder toolbar. This updates the Verilog wrapper file, which is found in the project base directory. When changes are made to the project, click this button to update the changes in the Verilog wrapper file.

After generating the Verilog wrapper file, export the Propel project to Lattice Radiant. To do this, click the  button in the Propel Builder toolbar (for projects using Diamond devices, this button is the Diamond icon). This launches Lattice Radiant and invokes the *radiant\_setup\_template.tcl* script in the project base directory, which generates a Lattice Radiant project and adds all the required files from the Propel project.

### 4.2. Exporting Project to Lattice Radiant

The specific steps in exporting to Lattice Radiant may vary depending on the project. For this demo, the basic steps of the Lattice Radiant flow is used.

To export a project to Lattice Radiant:

1. Add the *constraints.pdc* physical constraints file to the project from the project source directory.
2. In the Lattice Radiant file list, right-click **Post-Synthesis Constraint Files** and select **Add > Existing File**.
3. Locate the *constraints.pdc* file and click **Add**.

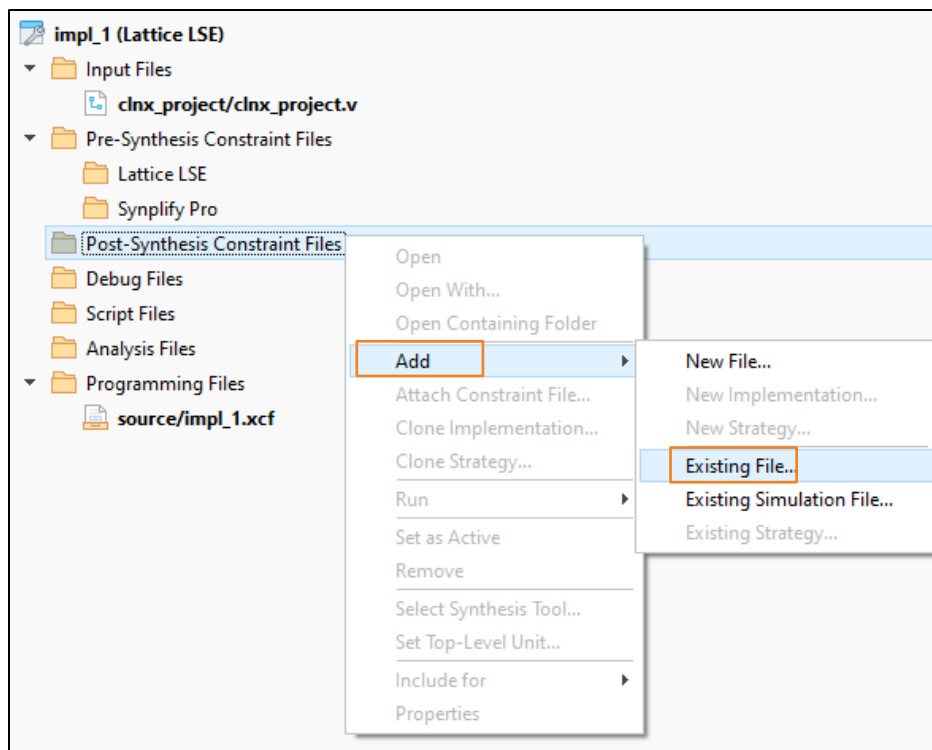


Figure 4.1. Adding Physical Constraints in Lattice Radiant File List

4. Use Lattice Radiant **Design Process Bar** to run through the project development flow (synthesis, map, PAR, and bitstream generation).

There are several ways to do this. For this demo, click the green arrow button. The user can also click the **Export Files** button.



Figure 4.2. Design Process Bar

5. After the bitstream is generated, all buttons in the **Design Process Bar** become green checkmarks, as shown in Figure 4.3.



Figure 4.3. Design Process Bar after Running Project Flow

It is important to know how Propel projects are managed along with Lattice Radiant or Diamond.

Clicking the Generate button in Propel Builder updates the top-level Verilog wrapper file that is generated for the Propel Project. When regenerating the Propel wrapper file, the rest of the files within the project also update accordingly. If the project is already exported to Lattice Radiant, the tool refreshes after changes are made to reflect the changes. Because of this, the user only needs to regenerate the Verilog wrapper file each time changes are made in Propel. There is no need to continuously relaunch Lattice Radiant from Propel to synchronize changes.

The recommended flow is to only launch Lattice Radiant directly from Propel the first time to generate a corresponding Lattice Radiant project. Each time after that, open the project directly from the Lattice Radiant user interface instead of launching it from Propel Builder toolbar. If any changes are made to the source Propel project, the changes are synchronized between the Propel and Lattice Radiant projects by regenerating the Verilog wrapper file as described in this section.

**Important:** Each time the Generate button is clicked in Propel Builder, all the files within the project RTL are updated. If any changes are made to the RTL files generated by Propel, these changes are overwritten each time the Propel project is regenerated. Keep a separate copy of any modified files to revert any unintentional changes.


The ECO Editor in Lattice Radiant and Diamond is used to make minor changes to a project once PAR is completed. Considering the time it takes to run through the Lattice Radiant project flow and generate a bitstream, this tool is especially useful in the Propel project development flow. The ECO Editor allows the user to initialize the memory within the design without having to go through the entire Lattice Radiant project flow. This feature and how it should be used in the Propel project development flow are discussed in more depth in the [Initializing the Memory](#) section.

### 4.3. Programming the Device

After completing the Lattice Radiant project flow and generating a bitstream file, the next task is to program the device.

**Note:** The overall process flow is similar in Lattice Radiant and Diamond.

To program the device after bitstream generation:

1. Connect the device to its power supply and to the host computer using USB.
2. Click the  button in the Lattice Radiant toolbar to open the **Programmer** tool.

The user can also click **Tools > Programmer** from the menu bar.

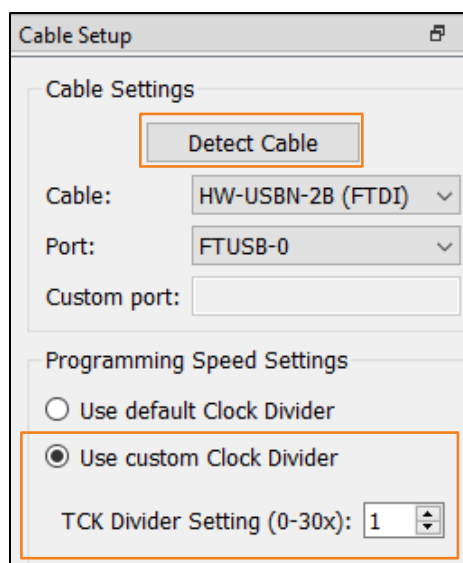
**Note:** Launching Programmer this way automatically generates a Programmer project and an .xcf chain configuration file.

The Programmer standalone version can also be launched from the Lattice Radiant directory; however, this requires additional steps to create and setup the Programmer project.


These additional steps are performed automatically when Programmer is launched directly from Lattice Radiant using the first two methods that were mentioned.

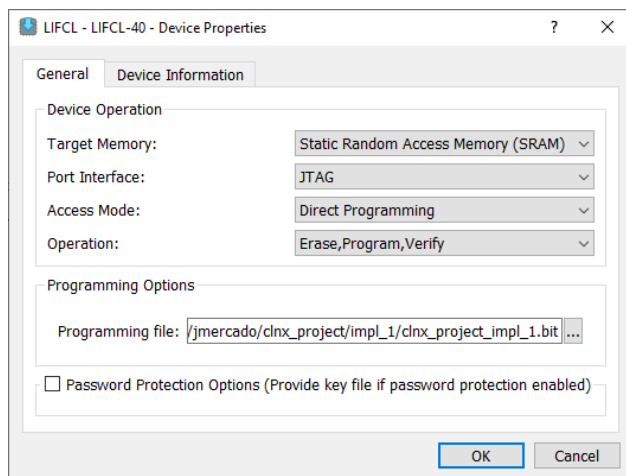
3. In the **Cable Setup** dialog box, click **Detect Cable** to detect the cable the device is using.
4. Select the **Port**.

**Note:** If unable to detect the device cable, select **Use custom Clock Divider** and set a higher **TCK Divider Setting** value.




**Figure 4.4. Cable Setup to Detect Programming Cable**

5. Scan for the device by clicking the  button. The user can also select **Run > Scan Device**.
6. Double-click the **Operation** column to configure the programming operation. This opens the **Device Properties** dialog box, as shown in [Figure 4.5](#).



**Figure 4.5. Device Properties Dialog Box**


7. In the **General** tab, under **Device Operation** select the options below.
  - Target Memory – SRAM
  - Port Interface – JTAG
  - Access Mode – Direct Programming
  - Operation – Erase, Program, Verify
8. Under **Programming Options**, in **Programming file**, select the bitstream file to program the device.
9. Click **OK**.
10. Click the  icon to program and load the bitstream file to the device. The user can also select **Run > Program Device**.



## 5. Propel SDK Flow

This section describes how firmware for the device is developed using Propel SDK.

### 5.1. Launching Propel SDK

The first step in the Propel SDK flow is to launch Propel SDK. Click the  button in the Propel Builder toolbar. The user can also select *Design > Run Propel* from the menu bar. Once the user selects a workspace and Propel SDK opens, a new Create Project dialog box opens with the required references to the Propel Builder project automatically populated.

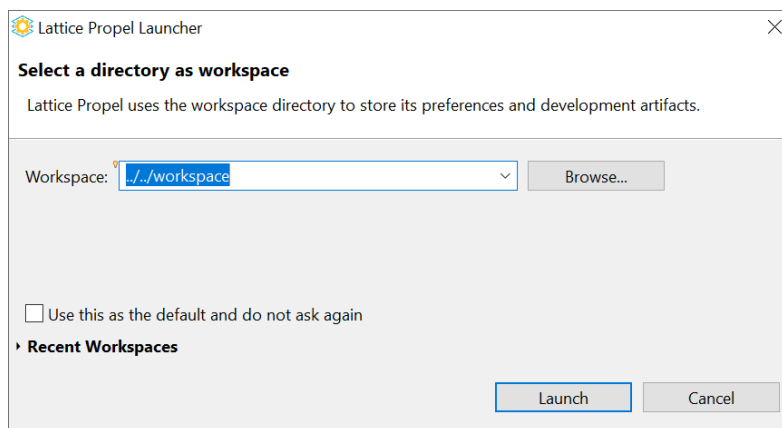
Alternatively, the user can also directly launch Propel SDK without using Propel Builder. This method, however, only launches Propel SDK and does not open the *Create Project* dialog box. To create a new project, use the Propel Builder toolbar or the menu bar to launch Propel SDK for the first time and then use any method each time after that.

### 5.2. Managing the Workspace

Once Propel SDK is launched, the Lattice Propel Launcher window opens, as shown in [Figure 5.1](#). Select a directory for the workspace. Creating the SDK workspace helps organize files in various software projects the user creates. By default, the active workspace is included with the Propel installation (*/lsc/propel/2.2/workspace*).

To select a different workspace location:

1. Click the **Browse** button.



**Figure 5.1. Propel SDK Workspace Selection**

2. For this demo, select the *demo\_workspace* directory created earlier.
3. Click **Launch** to open Propel SDK with the current workspace selection.

Selecting the *Use this as the default and do not ask again* option makes the current workspace selection always active each time Propel SDK is launched. As such, this Lattice Propel Launcher window is no longer displayed next time Propel SDK is opened.

Expanding the Recent Workspaces arrow opens a list of the most recently opened workspaces for the version of Propel SDK in use. The user can select a workspace from this list.

### 5.2.1. Switching between Workspaces

If the user accidentally opens the incorrect workspace directory or enabled the *Use this as the default and do not ask again* option and want to switch to a different workspace, the user can switch the active workspace directly from Propel SDK. To do this, select *File > Switch Workspace* from Propel SDK menu bar. This shows a list of the most recently opened workspaces. Either select a workspace from the list of recently opened workspaces or use the *Other* option to select an entirely new workspace.

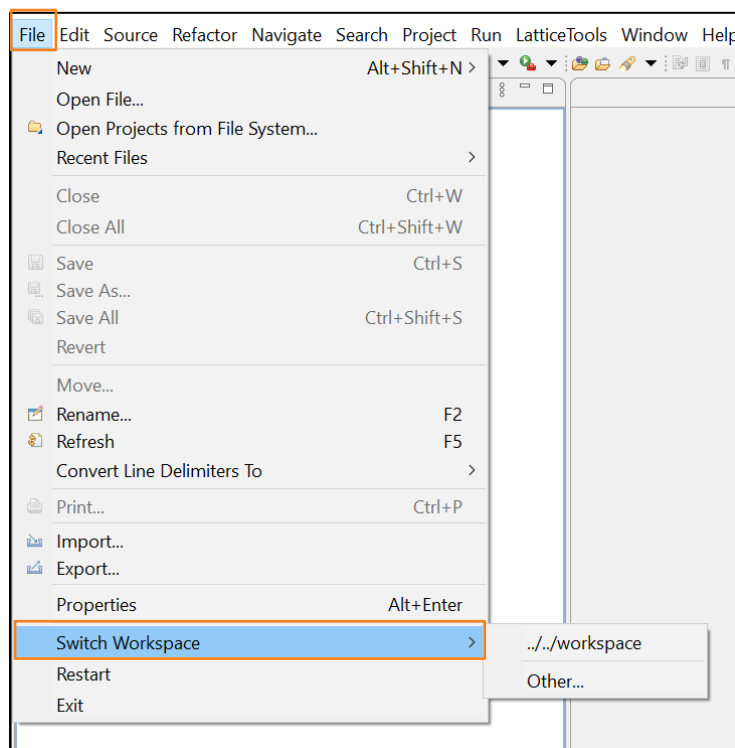


Figure 5.2. Switch Workspace Option

## 5.3. Creating and Managing a Project

### 5.3.1. Creating a New Propel SDK Project

To create a new Propel SDK project:

1. Select **File > New > Lattice C/C++ Project** from the Propel SDK menu bar. This step is optional if Propel SDK is launched directly from Propel Builder.
2. The **C/C++Project** wizard opens in the **Load System and BSP** page.

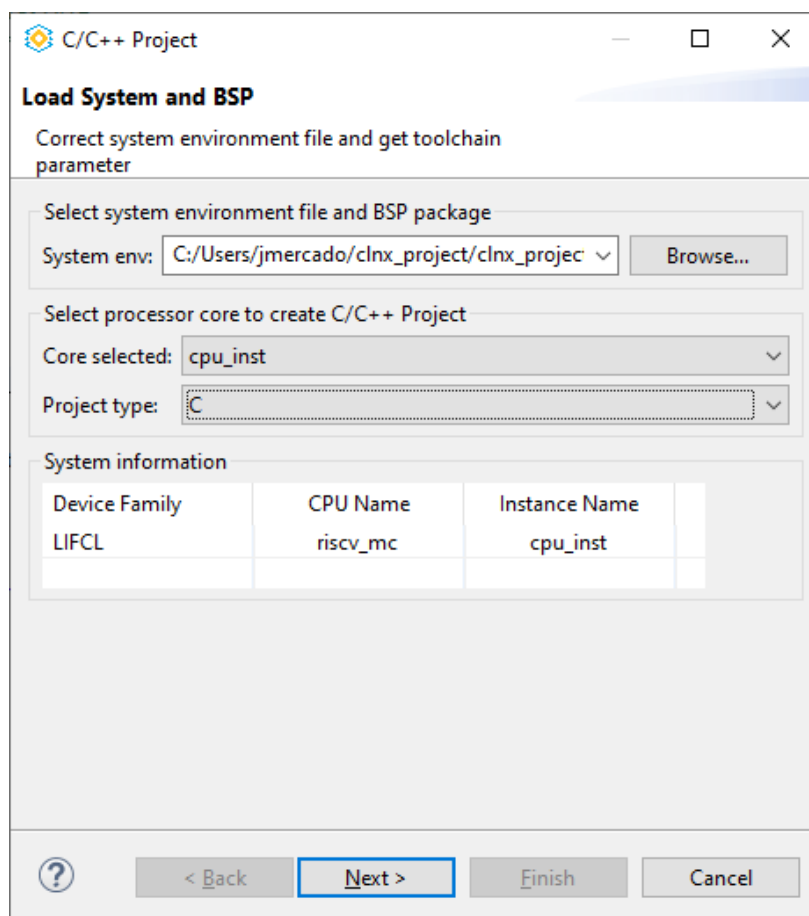


Figure 5.3. Load System and BSP Page of Project Creation Wizard

3. Select the system environment file from the Propel Builder project in **System env**. This file is located at `/crosslinknx_demo/sge/sys_env.xml`. This is also the file that needs to be updated each time changes are made to the SoC project. If Propel SDK is launched from a Propel Builder project, this field is automatically populated.
4. Select a processor for the project in **Core selected**. If the project contains multiple processors, ensure the correct processor is selected. For single processor projects, this step is not required since there is only one processor to select.
5. Select the language to use for the project in **Project type** (Figure 5.4).

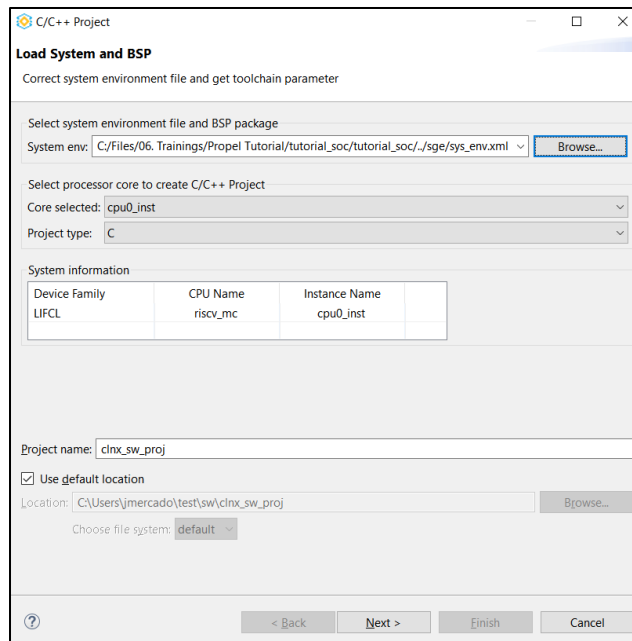


Figure 5.4. Initial C/C++ Project Creation Page

6. Enter a name for the project in the **Project name** field.
7. (Optional) Select a different location to save the project by disabling **Use default location** and selecting a new project directory. If **Use default location** is enabled, the project is saved to the directory of the active workspace. The project directory is indicated in the **Location** field.
8. Click **Next**. The **Lattice Toolchain Setting** page opens (Figure 5.5).

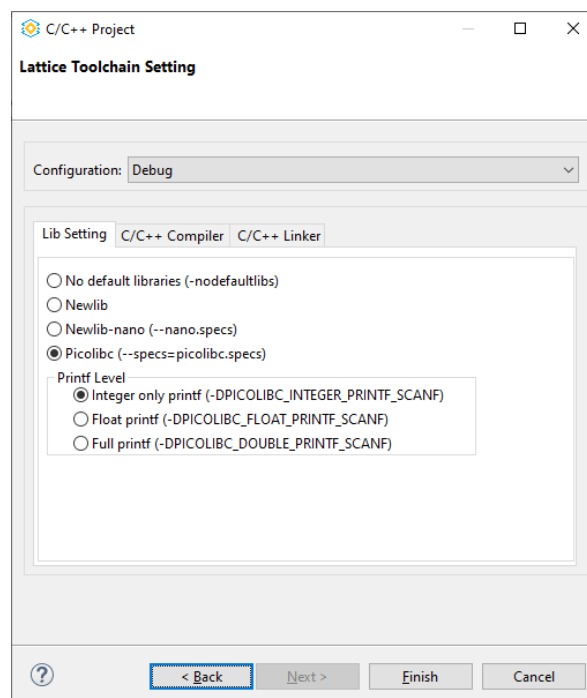


Figure 5.5. Lattice Toolchain Setting Page

9. Select **Debug** from the **Configuration** drop-down menu.

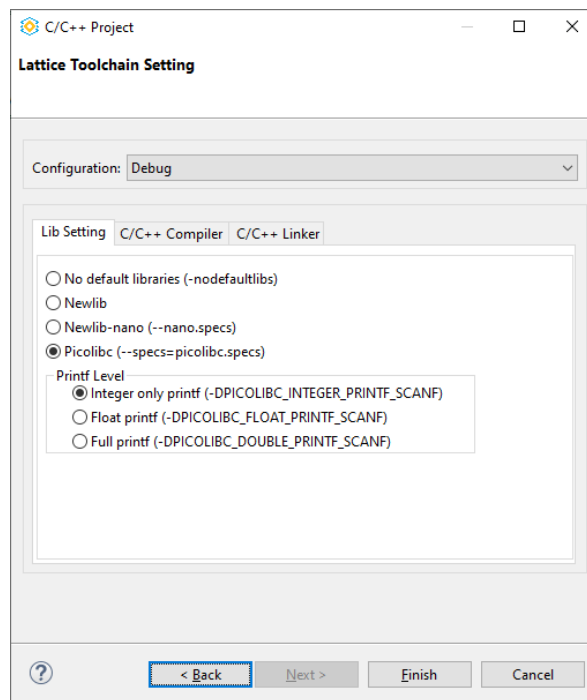
**Note:** Select **Debug** for testing and **Release** if the software is ready for release.

10. Configure Lib Setting, C/C++ Compiler, and C/C++ Linker settings.

11. Click **Finish** to complete the generation the software project.

The new project appears in the **Project Explorer** area if it is saved to the active workspace.

12. Click Next. The Lattice Toolchain Setting page opens.



**Figure 5.6. Lattice Toolchain Setting Page**

13. Select **Debug** from the **Configuration** drop-down menu.

**Note:** Select **Debug** for testing and **Release** if the software is ready for release.

14. Configure Lib Setting, C/C++ Compiler, and C/C++ Linker settings.

15. Click **Finish** to complete the generation the software project.

The new project appears in the **Project Explorer** area if it is saved to the active workspace.

## 5.4. Compiling the Project

Once the user starts working on the code for the software project, the next step in the Propel project development flow is to compile the files for testing and to check for errors. To compile the project, right-click the project name from the Project Explorer window and then select Build Project, as shown in Figure 5.7.

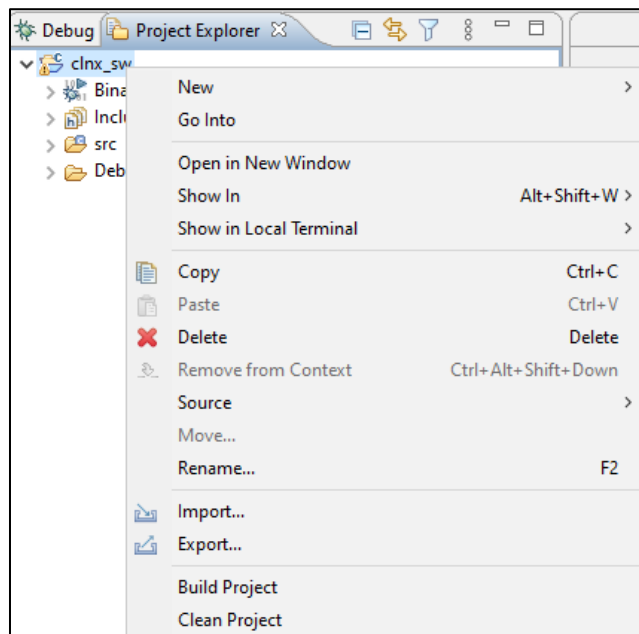


Figure 5.7. Compile Project Options

After the project is compiled, additional files are generated and added to the Debug folder in the project directory. The `<project name>.elf` file is an executable that is used for debugging. See the [On-Chip Debugging](#) section for details.

The following files are also generated:

- `<project name>.bin` binary file for flash programming
- `<project name>.map` memory linker map file
- `<project name>.lst` list file from the object tool dump
- `<project name>.mem` file, which is used to initialize the memory of the Propel projects.

**Important:** The user can also modify the toolchain and compilation settings. To do this, select *Project > Properties* in the Propel SDK menu bar. This opens the project Properties window, as shown in Figure 5.8. From this window, modify the toolchain and compilation settings by selecting the various tabs under the C/C++ Build dropdown options.

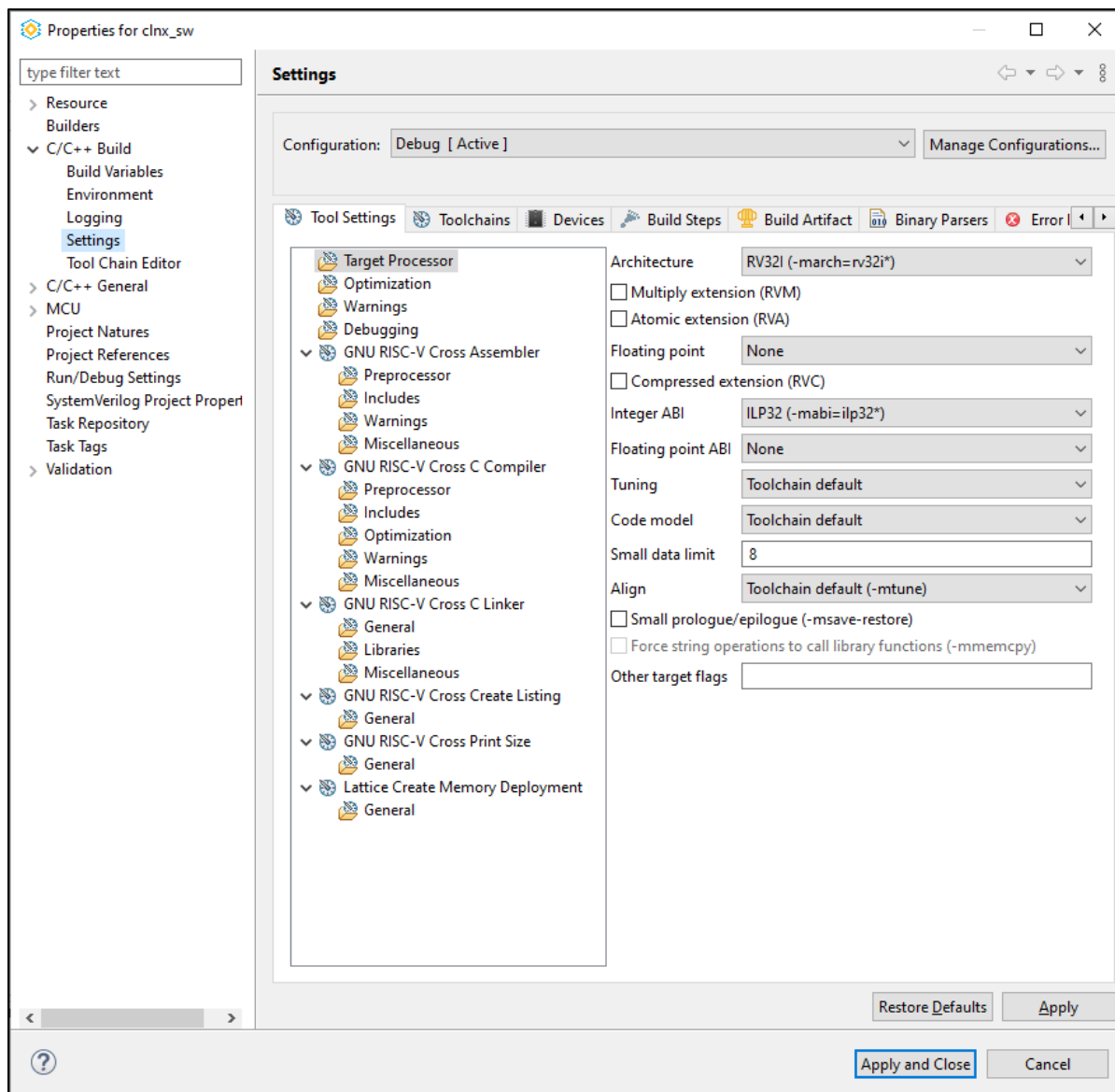


Figure 5.8. Project Settings Page

## 5.5. On-Chip Debugging

OpenOCD (On-Chip Debugging) is a tool for debugging the firmware on the device. OpenOCD allows the user to test the firmware without having to manually update the project memory and generate a new bitstream every time there is a change to the code.

The main mode of communication for OpenOCD is through a device com port. Typically, a serial terminal is connected to observe and test the output of firmware on the device. OpenOCD can also be used with Reveal Inserter and Reveal Analyzer/Controller to observe specific signals on the device while testing the firmware.

### 5.5.1. Setting up a Serial Terminal

To set up a serial terminal:

1. Click the **Terminal** tab.

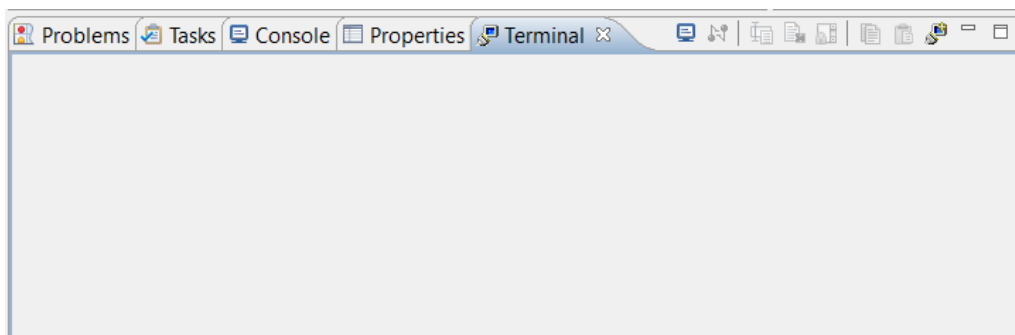


Figure 5.9. Terminal Tab in Propel SDK

2. Click the **Open a Terminal** button. The **Launch Terminal** dialog box opens.

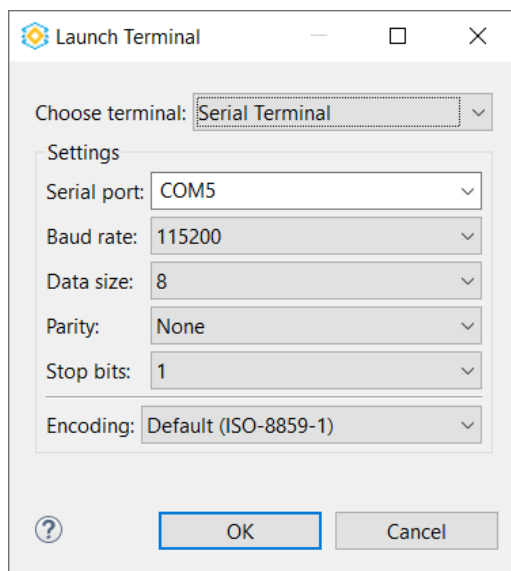


Figure 5.10. Launch Terminal Dialog Box

3. Select Serial Terminal in Choose terminal.
4. Select the **Serial port** to where the device is connected.
5. Click **OK**.



## 5.5.2. Debugging with OpenOCD

To debug using OpenOCD:

1. Select Run > Debug Configurations from the menu bar. The Debug Configurations window opens.

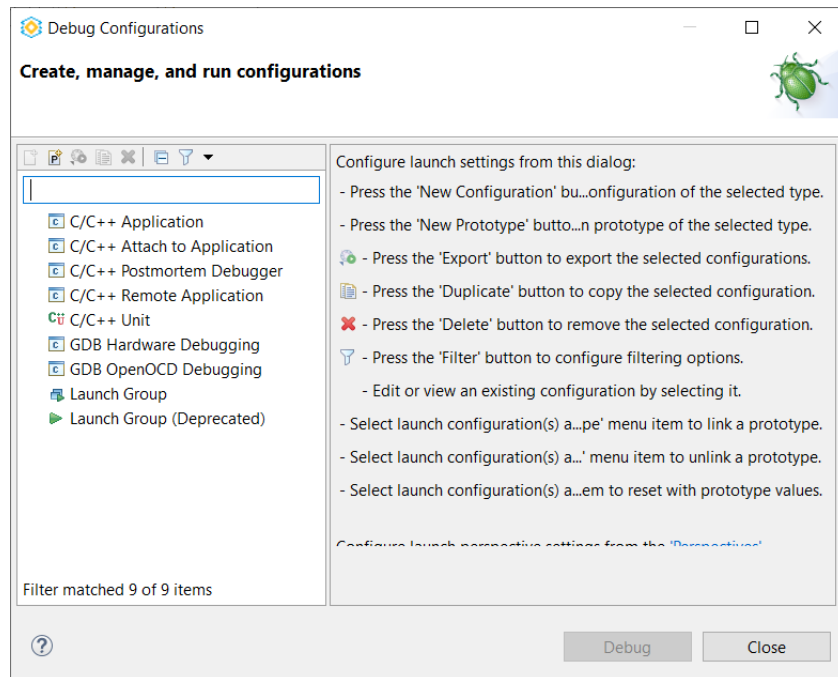


Figure 5.11. Debug Configurations

2. Double-click **GDB OpenOCD Debugging**. A new debug configuration is automatically generated for the active project, as shown in Figure 5.12.

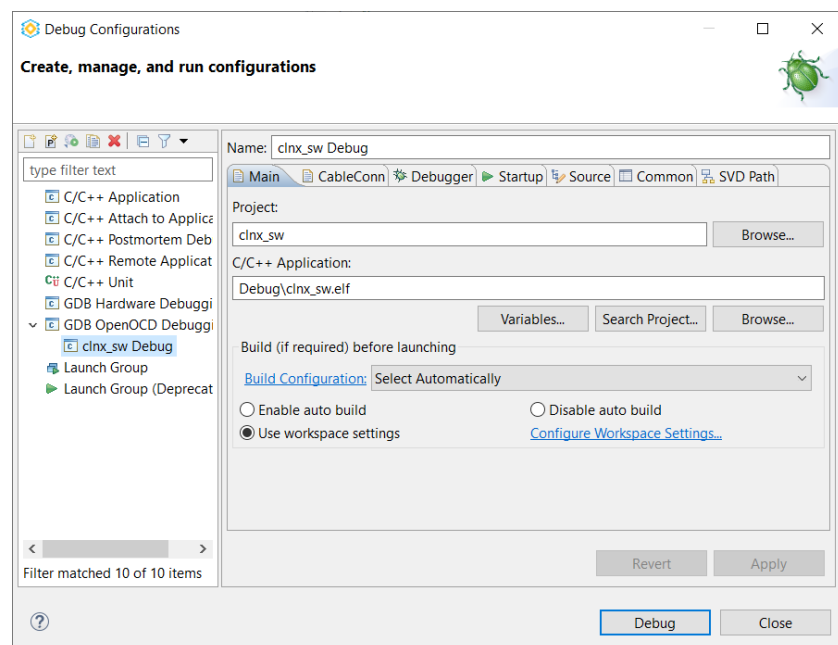


Figure 5.12. Debug Configurations for GDB OpenOCD Debugging

3. Ensure the correct project is selected. Otherwise, click the **Browse** button to select the correct project.

4. Click the **CableConn** tab.
5. Click the **Detect Cable** button to detect the programming cable connected to the device. If the selected cable is incorrect, click the drop-down arrow to select the correct cable.
6. Click the **Scan Device** button to scan for the project device.
7. Click **Debug** to begin the OpenOCD debugging operation. The console output is updated, indicating the status of the OpenOCD debugging operation. By default, the color of the OpenOCD console output is red. This does not indicate an error.
8. Switch to the **Terminal** tab again to observe the UART output. The device LED should also update continuously once the debug session begins.

## 5.6. Initializing the Memory

OpenOCD can be used to test software on the target device but it does not actually initialize the device system memory. After testing and debugging the software project, manual update of the system memory is still required for the changes to take place in the design. Typically, this is one of the last steps in the Propel project development flow, as both RTL and software should be finalized and working before going through this flow.

Depending on the software being used, there are two main ways to manually initialize project memory.

The first method is to double-click the component in the SoC design to modify some of its settings. From the component configuration window, locate the memory initialization settings, and configure the Format and Location settings to specify how to initialize system memory. By default, the *.mem* file generated by Propel SDK uses a hexadecimal format, so that is the format to select.

The drawback of this method is that it requires a rerun of synthesis, map, PAR, and bitstream generation, regardless of the amount of change. Running the entire flow again may be tedious and time consuming, especially for larger projects.


The second method is to use the Lattice Radiant or Diamond ECO Editor tool (the software used depends on the Propel project target device). ECO Editor can be used to make minor changes to a project after place and route, without having to rerun synthesis, map, and PAR.

The drawback of this method is that the changes do not affect the actual RTL or files within the project. To make larger changes to the project (such as adding another component), the original changes made using ECO Editor are not retained since a new UDB needs to be generated.

Another drawback of using the ECO Editor for memory initialization is that it does not work for Propel Verification projects. (Refer to the [Verification Project Flow](#) for details.) Verification projects only use the RTL and do not have anything to do with the UDB files generated by Lattice Radiant.



Because of these, the first method is the recommended option for initializing system memory to see the correct results in the simulations.

To initialize system memory using ECO Editor:

1. Open ECO Editor.
2. Select **Tools > ECO Editor** from the Lattice Radiant menu bar. Another method is to click the  icon from the toolbar.
3. Click the Memory Initialization tab.
4. Locate the target memory instance from the **Memory Instance** list. Depending on the project, several memory instances may appear. Check each instance name to ensure that the correct memory is update.

Memory Instance		Attribute		
▼ 1.0.4		Width:32	Depth:8192	Mode:EBR
Memory Initialization		Choose File ...		
▼ Components				
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R10C38		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R10C51		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R28C38		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R10C41		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R10C34		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R28C34		
mem_inst.lscs_sys_mem_inst.u...		EBR_CORE_R28C57		
sysIO Settings		Memory Initialization		

**Figure 5.13. ECO Editor Memory Initialization Tab**

5. In **Memory Initialization**, specify the *.mem* file to use for memory initialization.
6. Click the  button to perform a DRC check.
7. Click the  button to save the changes. Another option is to press CTRL + S on the keyboard. When prompted on how to proceed, select **Save** to update the active project's memory.

Exporting the changes as a TCL script allows the user to apply these changes to a different project for testing and also rerun the script to recreate the changes in case they are lost.

## 6. Verification Project Flow

Aside from SoC projects, Propel Builder also has a verification project mode, which can be used to generate a simulation environment to perform a behavioral simulation for our project. Each SoC project that is created automatically has a verification project that is generated alongside. The benefit of this is that the user does not need to create verification projects and can instead use the one that is automatically generated by Propel.

### 6.1. Pre-Simulation Requirements

To simulate the project:

1. Double-click **sysmem0\_inst** to modify the System Memory component.
2. Locate the Initialization parameter group to initialize the system memory.
3. Enable the **Initialize Memory** setting.
4. Set Initialization **File Format** to **Hex**.
5. Select the **Initialization File** option and locate the **.mem** file generated from the Propel SDK flow.

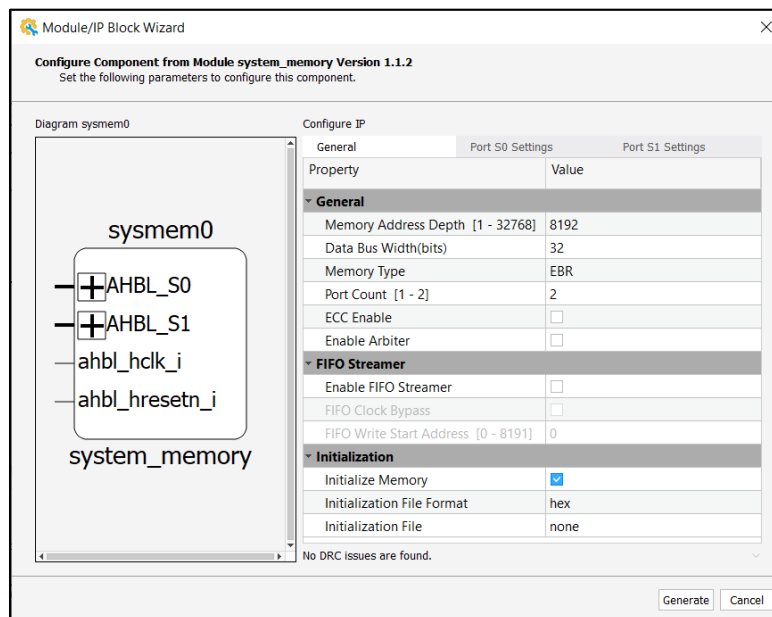


Figure 6.1. System Memory Parameters to Initialize Memory

6. Click **Generate** to update the component.
7. Double-click the **risvc\_mc0\_inst** component.
8. Set Simulation Mode to Enabled.
9. Set **Debug Mode** to **Disabled**. Skipping this step causes the simulation to fail, as there is no simulation model for the JTAG debug component that is added when this setting is enabled.

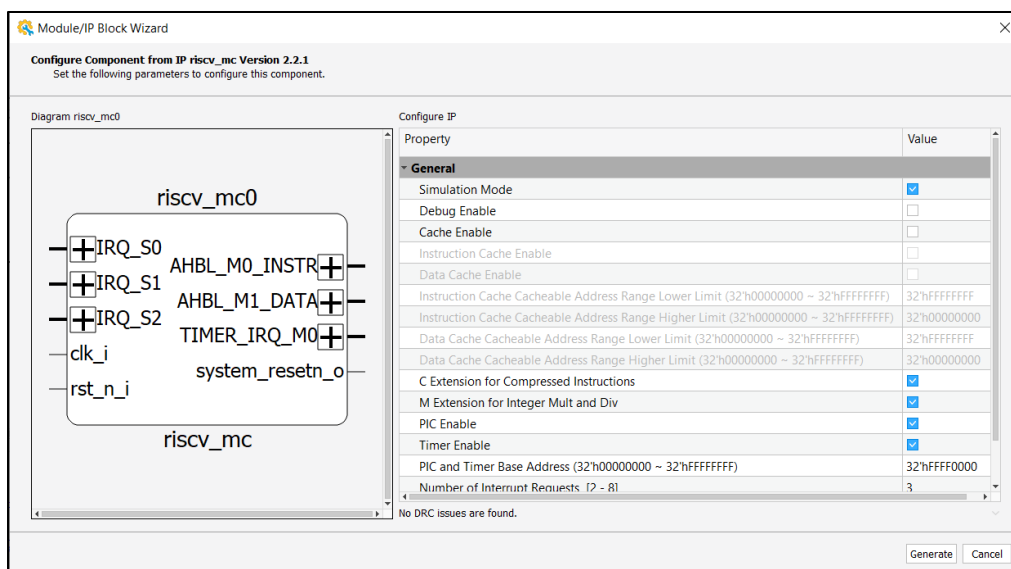


Figure 6.2. RISC-V MC Parameters for Project Simulation

10. Click **Generate** to update the component.

11. Switch to the Verification project for the design using the  icon.

This icon can be used to switch back and forth between the SoC and Verification projects.

## 6.2. Generating a Simulation Environment

For most project templates, a basic simulation environment is set up for the verification project using the Propel verification IP. For empty projects like this one, however, configure the basic simulation environment to provide stimulus and observe signals using the available verification IP like UART Model, Clock and Reset Generator, and AHBL Master BFM Edition.

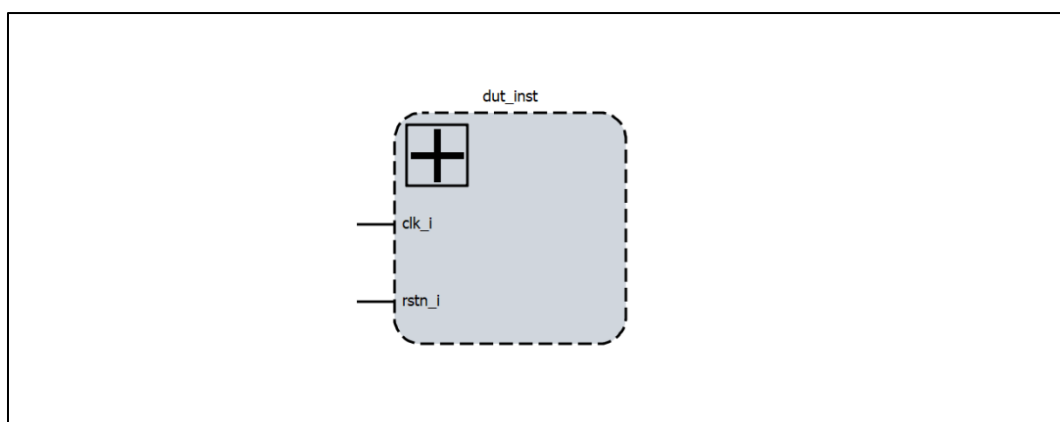


Figure 6.3. Initial Verification Project Schematic for Empty Projects

### 6.2.1. Setting up the Simulation Environment

Before simulating our SoC project, add additional verification IP to setup the simulation environment. Start by regenerating the DUT by double-clicking the dut\_inst component in [Figure 6.3](#). This reloads the DUT, so it is synchronized with the SoC project. This is important as it updates the top-level ports in which to connect.

In addition, clicking the plus icon on the top left of the dut\_inst component expands it, allowing the user to view the entire SoC project. This is another reason why it is important to update the DUT. The verification IP can be connected to signals within this project as well.

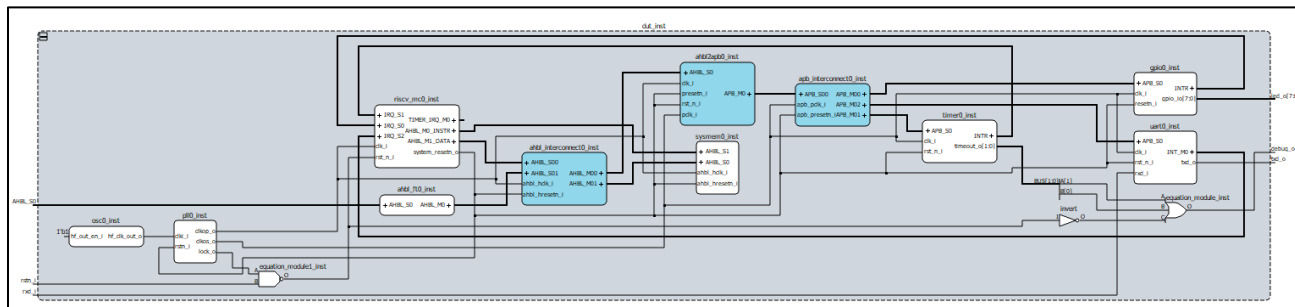


Figure 6.4. Regenerated DUT Component Expanded to View Contents

Propel Builder has several verification IPs that can be used to assist in debugging the SoC project in ModelSim. Each of the components added to the verification project is essentially its own instance in the generated simulation testbench, so this is especially useful depending on the verification IP in use.

### 6.2.2. Adding Verification IP – Clock and Reset Generator

1. Double-click the Clock and Reset Generator IP from Verification IP.
2. Enter a name for the IP. For this demo, use **clkrst0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 6.5](#).

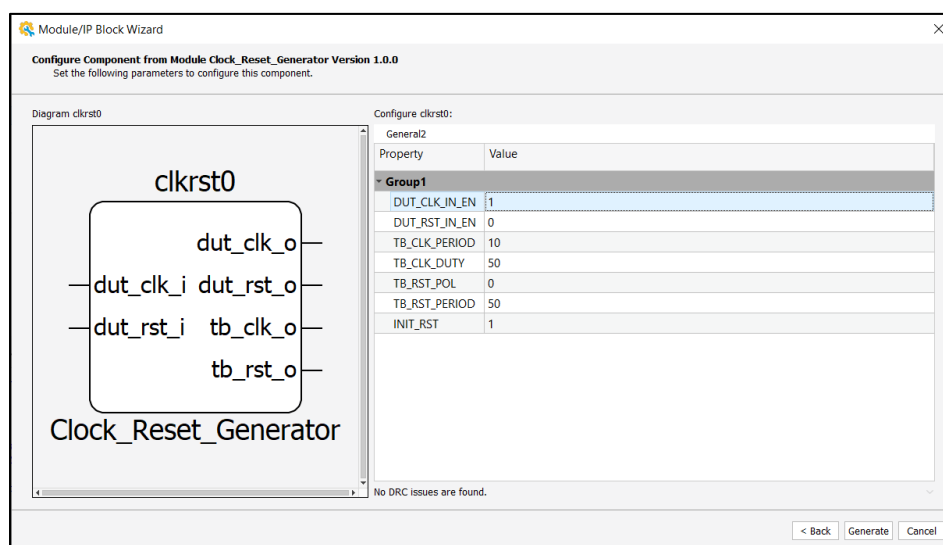


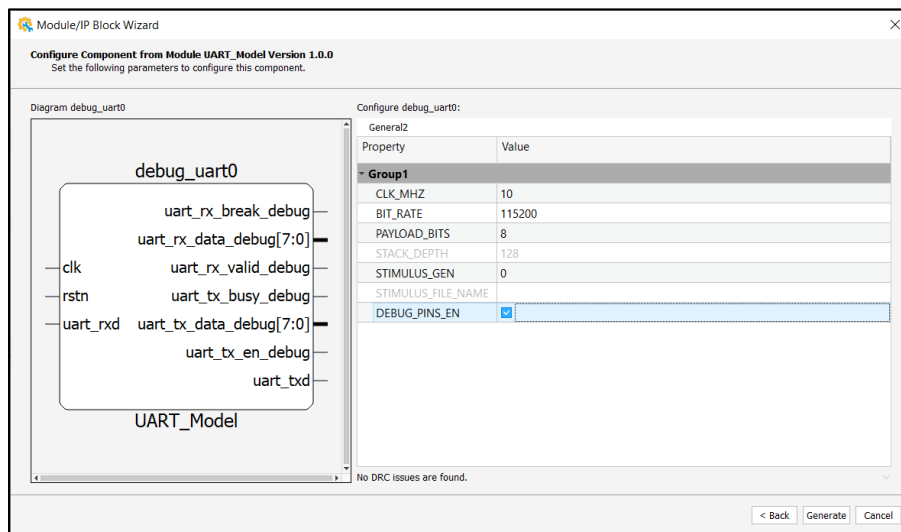
Figure 6.5. Clock and Reset Generator Component Parameter Configuration

5. Click **Generate**.
6. Enable **Insert to Project**.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.

8. Define an instance name. For this demo, the selected name is **clkrst0\_inst**.
9. Click **OK**.

### 6.2.3. Adding Verification IP – UART Model

1. Double-click the UART Model IP from Verification IP.
2. Enter a name for the IP. For this demo, use **debug\_uart0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 6.6](#).



**Figure 6.6. UART Model Component Parameter Configuration**

5. Click **Generate**.
6. Enable **Insert to Project**.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **debug\_uart0\_inst**.
9. Click **OK**.

## 6.2.4. Adding Verification IP – AHB-Lite Master Bus Function Model

1. Double-click the AHB-Lite Master BFM Lite Version IP from Verification IP.
2. Enter a name for the IP. For this demo, use **ahbl\_bfm0**. This is the name of the component.
3. Click **Next**.
4. In the **Module/IP Block Wizard**, set the parameters as shown in [Figure 6.7](#).

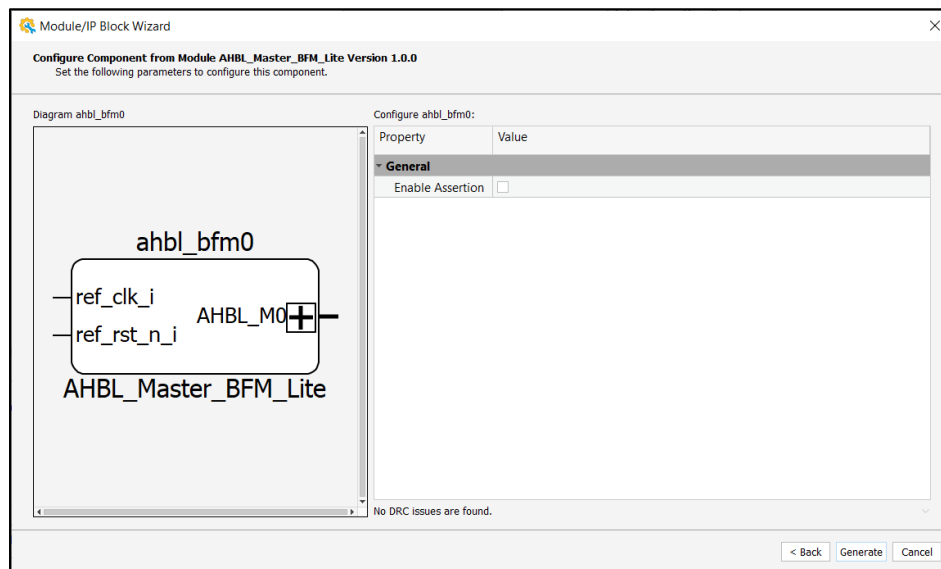
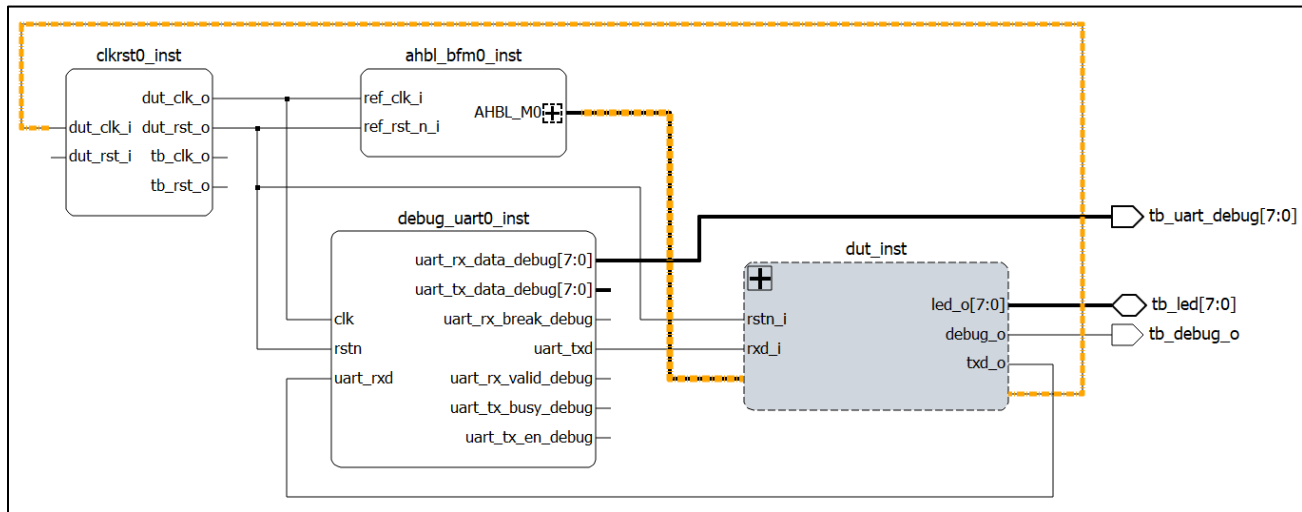


Figure 6.7. AHB-Lite Master BFM Component Parameter Configuration


5. Click **Generate**.
6. Enable **Insert to Project**.
7. Click **Finish**. A window to instantiate the new component in the SoC Design appears.
8. Define an instance name. For this demo, the selected name is **ahbl\_bfm0\_inst**.
9. Click **OK**.
10. Make the following connections in the verification project:
  - clkrst0\_inst/dut\_clk\_o > debug\_uart0\_inst/clk
  - clkrst0\_inst/dut\_clk\_o > ahbl\_bfm0\_inst/ref\_clk\_i
  - clkrst0\_inst/dut\_rst\_o > debug\_uart0\_inst/rstn
  - clkrst0\_inst/dut\_rst\_o > ahbl\_bfm0\_inst/ref\_rst\_n\_i
  - clkrst0\_inst/dut\_rst\_o > dut\_inst/rstn\_i
  - dut\_inst/txd\_o > debug\_uart0\_inst/uart\_rxd
  - debug\_uart0\_inst/uart\_txd > dut\_inst/rxd\_i
  - ahbl\_bfm0\_inst/AHBL\_M0 > dut\_inst/AHBL\_S0
11. Expand the **dut\_inst** component again by clicking the plus icon on the top left.
12. Connect pll0\_inst/clkop\_o > clkrst0\_inst/dut\_clk\_i.
13. Create three additional ports by right-clicking and selecting **Create Port**.
  - 8-bit inout port called tb\_led.
  - 8-bit output port called tb\_uart\_debug.
  - 1-bit output port called tb\_debug\_o.
14. Make the following last few connections:
  - dut\_inst/led\_o > tb\_led
  - dut\_inst/debug\_o > tb\_debug\_o
  - debug\_uart0\_inst/uart\_rx\_data\_debug > tb\_uart\_debug



15. The final verification project design is similar to [Figure 6.8](#).



**Figure 6.8. Final Verification Project Design**

16. Click the  icon to generate a simulation environment.

The main simulation environment files that are generated are:

- *msim.do* – script used to perform the entire simulation process (compile files, run simulation, add signals, and so on)
- *wave.do* – waveform do script that automatically adds all the signals from the DUT
- *flist.f* – file list used to specify the location of files included in the design for compilation
- *<project name>\_v.sv* – System Verilog test bench that is generated to provide the DUT stimulus. This file is extremely important. The Propel verification projects generate a minimalistic testbench, so it is likely that additional code are added to this file to simulate the project in a more detailed manner.

17. Click **Yes** if a prompt appears confirming that the simulation environment has changed.

- This warning is to prevent files from being overwritten, since existing files from previous simulations are updated and overwritten.
- If modifications are made to any of the generated simulation files, ensure that there is a separate local copy of these files to serve as backup.


## 6.3. Project Simulation


After the project simulation environment is generated, the final step is to launch ModelSim and invoke the simulation scripts to perform the behavioral simulation.

**Important:** Each Lattice software tool comes with a separate set of simulation libraries. For this reason, it is important to make sure that Propel simulation libraries are being used. Using Lattice Radiant or Diamond may result in simulation errors depending on the components in the project.

There are two main methods to perform project simulation.

The first method is to manually launch ModelSim from Propel, by selecting Tools and then ModelSim Lattice-Edition. This opens the Propel version of ModelSim and ensures that the correct simulation library is used. Once ModelSim is opened, invoke the *msim.do* script to perform the simulation.

The second method is to click the  icon from the Propel Builder toolbar. This opens ModelSim from Propel and automatically invokes the *msim.do* script. If invoking the simulation script for the first time, or do not plan on making any changes to the simulation environment then this method is recommended. However, if the user plans on editing any of the files from the generated simulation environment, the recommended flow is to launch ModelSim separately from the verification project. This ensures that no files are overwritten and allows for easier debugging.

If ModelSim is launched using the  icon from the Propel Builder toolbar, the simulation script that is generated for the verification project is automatically invoked. The simulation script is basic and it compiles the files in the project, invokes the VSIM simulator, adds signals to the waveform display, and advances the simulation.

Once the simulation script is executed, view the results for the first one millisecond of the simulation. By default, the generated simulation environment is run for one millisecond and includes all the top-level signals from the DUT. The reason for this is because the purpose of verification projects is to assist in generating a simulation environment for a Propel project. Aside from the simulation *.do* scripts, verification projects also generate a simple testbench used to provide the DUT with stimulus. The expectation is that this generated simulation environment should be used as a starting point to simulate a project, and that the user should modify the generated environment as desired to meet their project requirements.

### 6.3.1. Modifying the Simulation Environment

To modify the simulation environment:

1. Open the simulation *.do* script in a text editor.  
`/crosslinknx_demo/verification/sim/msim.do`
2. Replace line #10, **do wave.do**, with **add wave /\***. Instead of the top-level signals from the DUT, the signals from the testbench are added.
3. Change line #11 to **run 0.5 ms**.
4. Add another line afterwards at line #12, **wave zoom full**. This fits the simulation results to the waveform display once the simulation is finished.
5. Save the script and return to ModelSim.
6. Enter **quit -sim** into the TCL console at the bottom of the window.
7. Reinvoke the *msim.do* simulation script.
8. Select **File > Load > Macro File** from the menu bar and locate the *msim.do* script to invoke it.  
**Note:** Press the up arrow in the TCL console to reinvoke commands. This is another way to easily reinvoke the simulation script.
9. Once the simulation is finished, different signals are added to the waveform display and the simulation has run for a different amount of time.
10. Right-click the **tb\_uart\_debug** signal and select **Radix** and then **ASCII**. The UART debug output is updated. *Hello World* is displayed in the simulation.

## 7. References

- [Lattice Propel 2.1 Release Notes \(FPGA-AN-02044\)](#)
- [Lattice Propel 2.1 Installation for Windows User Guide \(FPGA-AN-02043\)](#)
- [Lattice Propel Builder 2.1 Usage Guide \(FPGA-UG-02143\)](#)
- [MachXO3D Family Data Sheet \(FPGA-DS-02026\)](#)
- [MachXO3D Programming and Configuration Usage Guide \(FPGA-TN-02069\)](#)
- [MachXO3D Breakout Board User Guide \(FPGA-UG-02084\)](#)
- [Lattice Propel Builder 2.2 Release Notes \(FPGA-AN-02051\)](#)
- [Lattice Propel 2.2 Builder Usage Guide \(FPGA-UG-02156\)](#)
- [Lattice Propel 2.2 SDK User Guide \(FPGA-UG-02155\)](#)
- [Lattice Propel 2.2 Installation for Windows User Guide \(FPGA-AN-02049\)](#)
- [Lattice Propel 2.2 Installation for Linux User Guide \(FPGA-AN-02050\)](#)
- [RISC-V MC CPU IP - Lattice Propel Builder 2.2 \(FPGA-IPUG-02181\)](#)
- [Clock Reset Generator VIP - Lattice Propel Builder \(FPGA-IPUG-02147\)](#)
- [Timer/Counter IP Core – Lattice Propel Builder \(FPGA-IPUG-02139\)](#)
- [UART Model VIP - Lattice Propel Builder \(FPGA-IPUG-02146\)](#)

## Technical Support Assistance

Submit a technical support case through [www.latticesemi.com/techsupport](http://www.latticesemi.com/techsupport).

For frequently asked questions, refer to the Lattice Answer Database at [www.latticesemi.com/Support/AnswerDatabase](http://www.latticesemi.com/Support/AnswerDatabase).

## Revision History

### Revision 1.1, January 2024

Section	Change Summary
Propel Builder Flow	<ul style="list-style-type: none"><li>• Made editorial modifications to some descriptions.</li><li>• Updated <a href="#">Figure 3.3. SoC Project Creation Window and Settings for the Demo</a>.</li><li>• Updated the section header to <a href="#">Generating and Instantiating the AHB-Lite Interconnect IP</a>.</li><li>• Updated <a href="#">Figure 3.11. RISC-V MC Component Parameter Configuration</a> reflecting the Number of Interrupt Requests change from 2 to 3.</li><li>• Updated <a href="#">Figure 3.12. UART Component Parameter Configuration</a>.</li></ul>
Propel SDK Flow	Updated the description of the <a href="#">Creating a New Propel SDK Project</a> section to reflect the most recent software changes.

### Revision 1.0, September 2022

Section	Change Summary
All	Production release.



[www.latticesemi.com](http://www.latticesemi.com)