Lattice Propel 2.0 Builder

User Guide

FPGA-UG-02126-1.0

April 2021

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 and with all faults, and all risk associated with such information is entirely with 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

Glossary 8
1. Introduction 9
1.1. Purpose 9
1.2. Audience 9
2. Lattice Propel Builder Design Flows 10
2.1. Builder Environment 10
2.2. Project Design Flow 11
2.2.1. Creating a New SoC Project 11
2.2.2. Creating Template SoC Project 16
2.2.3. Opening an SoC Existing Project 20
2.2.4. Adding Modules 21
2.2.5. Adding Glue Logic 24
2.2.6. Undo/Redo 35
2.2.7. Working with the Schematic View 35
2.2.8. Connecting Modules 45
2.2.9. Creating Top-Level Ports 48
2.2.10. Adjusting Address Spaces 49
2.2.11. Validating the Design 50
2.2.12. Generating the RTL file 50
2.2.13. Opening Project in Diamond or Radiant 51
2.2.14. Launching SDK 53
2.3. Verification Project Design Flow 56
2.3.1. Creating a Verification Project 56
2.3.2. Switching SoC Project to Verification Project 60
2.3.3. Opening a Verification Project 60
2.3.4. Adding Modules, IPs and VIPs 60
2.3.5. Working with the Schematic View 61
2.3.6. Connecting Modules 61
2.3.7. Monitoring DUT 61
2.3.8. Generating Simulation Environment 62
2.3.9. Launching Simulation 62
3. IP Packager 64
3.1. Launching IP Packager 65
3.2. Packing Custom IP Flow 65
3.2.1. Opening an IP Directory 65
3.2.2. Editing IP 68
3.2.3. Previewing IP 88
3.2.4. Packaging IP 89
3.3. Editing IP Package Files 89
3.3.1. Meta Data File 89
3.3.2. Implementation RTL Files 99
3.3.3. Python Script Plugin File 100
3.3.4. Memory Map CSV File 101
4. TCL Commands 102
4.1. sbp_design 102
4.1.1. Open 102
4.1.2. Close 102
4.1.3. New 102
4.1.4. Save 102
4.1.5. Drc 102
4.1.6. Generate 103
4.1.7. auto_assign_addresses 103

4.1.8. Launch SoC Verification Engine 103
4.1.9. Execute PGE Engine 103
4.1.10. Undo 104
4.1.11. Redo 104
4.2. Other TCL Commands 104
4.2.1. sbp_add_component 104
4.2.2. sbp_add_sbx_component 104
4.2.3. sbp_add_gluelogic 105
4.2.4. sbp_create_glue_logic 105
4.2.5. sbp_add port 105
4.2.6. sbp_connect_net 105
4.2.7. sbp_connect_interface_net 105
4.2.8. sbp_connect_constant 105
4.2.9. sbp_connect_whitebox 105
4.2.10. disconnect whitebox connection 106
4.2.11. sbp_disconnect_interface_net 106
4.2.12. sbp_disconnect_net 106
4.2.13. sbp_assign_addr_seg 106
4.2.14. sbp_unassign_addr_seg 106
4.2.15. sbp_assign_local_memory 106
4.2.16. sbp_export_pins 106
4.2.17. sbp_export_interface 107
4.2.18. sbp_rename 107
4.2.19. sbp_replace 107
4.2.20. sbp_copy 107
4.2.21. sbp_delete 107
4.2.22. sbp_get_pins 108
4.2.23. sbp_get_interface_pins 108
4.2.24. sbp_get_ports 108
4.2.25. sbp_get_interface_ports 108
4.2.26. sbp_get_nets 108
4.2.27. sbp_get_interface_nets 108
4.2.28. sbp_set_property 108
4.2.29. sbp_get_property 109
4.2.30. sbp_report_properties 109
4.2.1. sbp_get_components 109
Appendix A. metadata.xsd 110
References 118
Technical Support Assistance 119
Revision History 120

Figures

Figure 2.1. Propel Builder Workbench Window 10
Figure 2.2. Create System Design – Design Information Wizard 11
Figure 2.3. Create System Design - Propel Project Configure Wizard 12
Figure 2.4. Specify a Device for Project 13
Figure 2.5. Specify a Board for Project 14
Figure 2.6. Create System Design - Project Information Wizard 15
Figure 2.7. Propel Builder GUI Shows Empty Project 16
Figure 2.8. Create System Design – Design Information Wizard 17
Figure 2.9. Specify a Board for Project 18
Figure 2.10. Project Information Wizard 19

Figure 2.11. Propel Builder GUI shows Template Project 19
Figure 2.12. Open Sbx Dialog 20
Figure 2.13. Open HelloWorld Project 20
Figure 2.14. IP Catalog 21
Figure 2.15. Module/IP Block Wizard – Generate Component from Module gpio Version 1.4.1 22
Figure 2.16. Module/IP Block Wizard - Configure Component from IP Gpio Version 1.4.1 22
Figure 2.17. Module/IP Block Wizard - Check Generated Result 23
Figure 2.18. Design Instance Dialog Box 23
Figure 2.19. Propel Builder Schematic View Shows the Module Instance 24
Figure 2.20. IP Catalog 25
Figure 2.21. Glue Logic for Concat Module 26
Figure 2.22. Schematic View Shows a Concat Module 27
Figure 2.23. Glue Logic Wizard for Equation Module 28
Figure 2.24. Schematic View Shows an Equation Module 29
Figure 2.25. Schematic View Shows the Invert Module 30
Figure 2.26. Glue Logic Wizard for Rtl Module 31
Figure 2.27. Existing Rtl Module Configuration 32
Figure 2.28. Schematic View Shows the Custom Rtl Module 33
Figure 2.29. Glue Logic Wizard for Split Module 34
Figure 2.30. Schematic View Shows Split Module 35
Figure 2.31. Signal List of Modules 36
Figure 2.32. Select Object 37
Figure 2.33. Locate Objects 38
Figure 2.34. Duplicate a Module 38
Figure 2.35. Design View 39
Figure 2.36. Define Instance Dialog Box 39
Figure 2.37. Module/IP Block Wizard – Configure Component 40
Figure 2.38. Select Module 41
Figure 2.39. Options Dialog 42
Figure 2.40. Show Connectivity of the Module 43
Figure 2.41. Highlight an Object 43
Figure 2.42. Object Properties 44
Figure 2.43. Print Preview 44
Figure 2.44. Draw a Pin or a Port 45
Figure 2.45. Draw Nets 45
Figure 2.46. Select More than One Ports 46
Figure 2.47. Action Menu of Right-clicking an Input Pin 47
Figure 2.48. Dialog Box of Assigning Constant Value to an Input Pin 47
Figure 2.49. Create Port Dialog Box 48
Figure 2.50. Input Port 48
Figure 2.51. Output Port and Inout Port 48
Figure 2.52. Address View 49
Figure 2.53. Edit Base Address 50
Figure 2.54. Memory Report 50
Figure 2.55. Diamond Project 51
Figure 2.56. Generate Programming Files 52
Figure 2.57. Radiant Project 53
Figure 2.58. Lattice Propel Launcher Wizard 54
Figure 2.59. Propel SDK GUI and C/C++ Project Wizard 54
Figure 2.60. Create C/C++ Project 55
Figure 2.61. Lattice Toolchain Setting Dialog 56
Figure 2.62. Create System Design – Design Information Wizard 57
Figure 2.63. Create System Design – Propel Project Configure Wizard 58
Figure 2.64. Verification Project 59

Figure 2.65. Whole SoC Design 59
Figure 2.66. Propel Builder Dialog Box 60
Figure 2.67. Switching to Project Verification 60
Figure 2.68. Monitoring DUT 61
Figure 2.69. Testbench of the Verification Project 61
Figure 2.70. Testbench File Structure 62
Figure 2.71. Propel Builder – A Reminding Dialog Box 62
Figure 2.72. Simulation GUI 63
Figure 2.73. Builder Options Wizard 63
Figure 3.1. Example Directories and Files of an IP Package 64
Figure 3.2. Example Directories and Files of an IP Instance Package 64
Figure 3.3. IP Packager GUI 65
Figure 3.4. Select Folder Dialog 66
Figure 3.5. IP Packager with IP Project Details 67
Figure 3.6. IP Packager with Meta Data Details 68
Figure 3.7. Meta Data View 69
Figure 3.8. Configure Basic Info 69
Figure 3.9. Warning Message of the Invalid Radiant and Propel Version 70
Figure 3.10. Right-click Menu of Parameters in the Meta Data View 71
Figure 3.11. Group1 Added to Tab1 71
Figure 3.12. NewParam1 Added to Group1 72
Figure 3.13. Rename a Parameter 72
Figure 3.14. Configure a Parameter 73
Figure 3.15. Three Port Types 75
Figure 3.16. Right-click Menu of the IN Port 76
Figure 3.17. Add inferred ports to meta data Wizard 77
Figure 3.18. Rename an IN Port 77
Figure 3.19. Configure an IN Port 78
Figure 3.20. Right-click Menu of an Interface 79
Figure 3.21. Rename Interface 79
Figure 3.22. Configure an Interface 80
Figure 3.23. Right-click Menu of a Memory Map 81
Figure 3.24. Generate New Memory Map 82
Figure 3.25. IP Packager with Design File Details 83
Figure 3.26. IP Packager with Test Bench Details 84
Figure 3.27. IP Packager with Misc Details 85
Figure 3.28. IP Packager with Doc Assistant Details 86
Figure 3.29. Configure Doc Assistant 87
Figure 3.30. IP Preview Shows Configuration for IP Module 88
Figure 3.31. IP Packager Pops up Error Message 88
Figure 3.32. IP Packager Pops up Successful Packaging Message 89
Figure 3.33. Example XML of Metadata Layout 89
Figure 3.34. Example of busInterface Node 95
Figure 3.35. Example of addressSpace Node 96
Figure 3.36. Example of memoryMaps Node 97
Figure 3.37. Example of componentGenerator Node 98
Figure 3.38. Example of XInclude Usage 99
Figure 3.39. Example of Specifying Lib for VHDL 99
Figure 3.40. Template of Plugin File 100
Figure 3.41. Example of Memory Map CSV File 101
Figure 4.1. Tcl Console 102

Tables

Table 3.1. Details of Parameter Property 73
Table 3.2. Details of Port Property 78
Table 3.3. Child Nodes of General Node..90
Table 3.4. Attributes of Setting Nodes 91
Table 3.5. Attributes of Port Nodes..93
Table 3.6. Elements in estimatedResource Node..98

Glossary

A list of words or terms specialized in this document.

Glossary	Definition
BSP	Board Support Package, the layer of software containing hardware-specific drivers and libraries to function in a particular hardware environment.
CPU	Central Processing Unit.
CSV	Comma Separated Values file.
DRC	Design Rule Check.
DUT	Design Under Test.
ESI	Previous name of Propel.
FPGA	Field Programmable Gate Array.
GUI	Graphic User Interface.
HDL	Hardware Description Language.
HSM	Hardware Security Module.
IDE	Integrated Development Environment.
IP-XACT	An XML format that defines and describes electronic components and their designs.
LHS	Left Hand Side.
LSB	Least Significant Bit.
MSB	Most Significant Bit.
Programmer	A tool can program Lattice FPGA SRAM and external SPI Flash through various interfaces, such as JTAG, SPI, and I^2C.
Perspective	A group of views and editors in the Workbench window.
RHS	Right Hand Side.
RISC-V	A free and open instruction set architecture (ISA) enabling a new era of processor innovation through open standard collaboration.
SBX	The files that store the spatial index of the features
SDK	Embedded System Design and Develop 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. An integrated circuit that integrates all components of a computer or other electronic systems.
SRAM	Static Random Access Memory.
TCL	Tool Command Language.
UFM	User Flash Memory.
VIP	Verification IP.
Workspace	The directory where stores your work, it is used as the default content area for your projects as well as for holding any required metadata.
Workbench	Refers to the desktop development environment in Eclipse IDE platform.

1. Introduction

Lattice Propel™ 2.0 Builder is a graphical tool used to assemble complex System-on-Chip (SoC) modules which can be used in the supported Lattice FPGA devices. These modules and/or IPs can be assembled and connected easily by simply dragging and dropping the modules and/or IPs into the Schematic Window.

1.1. Purpose

Embedded system solutions play an important role in FPGA system design allowing you to develop the software for a processor in an FPGA device. It provides 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. Lattice Propel Builder helps you develop your system with a RISC-V processor, peripheral IP, and a set of tools by a simple drag-and-drop.

The purpose of this document is to introduce Lattice Propel 2.0 Builder tool and design flow to help you quickly get started to build a small demo system. You can also find the recommended flows of using Lattice Propel Builder in this document.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice MachXO3D, MachXO3L, MachXO3LF, MachXO2, LIFCL, LFD2NX and LFMNX devices. The technical guidelines assume readers have expertise in the embedded system area and FPGA technologies.

2. Lattice Propel Builder Design Flows

The Propel Builder design flow includes creating an SoC project design flow, and verification design flow, which are discussed in detail in the following sections.

2.1. Builder Environment

After Propel 2.0 is installed, you can launch the stand-alone Propel Builder by double-clicking the Builder icon to launch Builder. Refer to the Lattice Propel 2.0 Installation for Windows User Guide (FPGA-AN-02036) for details on the installation. After the Propel Builder is launched, a single workbench window is displayed. The workbench contains Menu, Toolbar, Design View, IP catalog, Schematic view, address mapping, Start Page and TCL console. Figure 2.1 shows the workbench with opening a project.

1. Menu bar
2. Toolbar
3. IP Catalog and Design View
4. Schematic View, Address Mapping, and Start Page
5. TCL Console

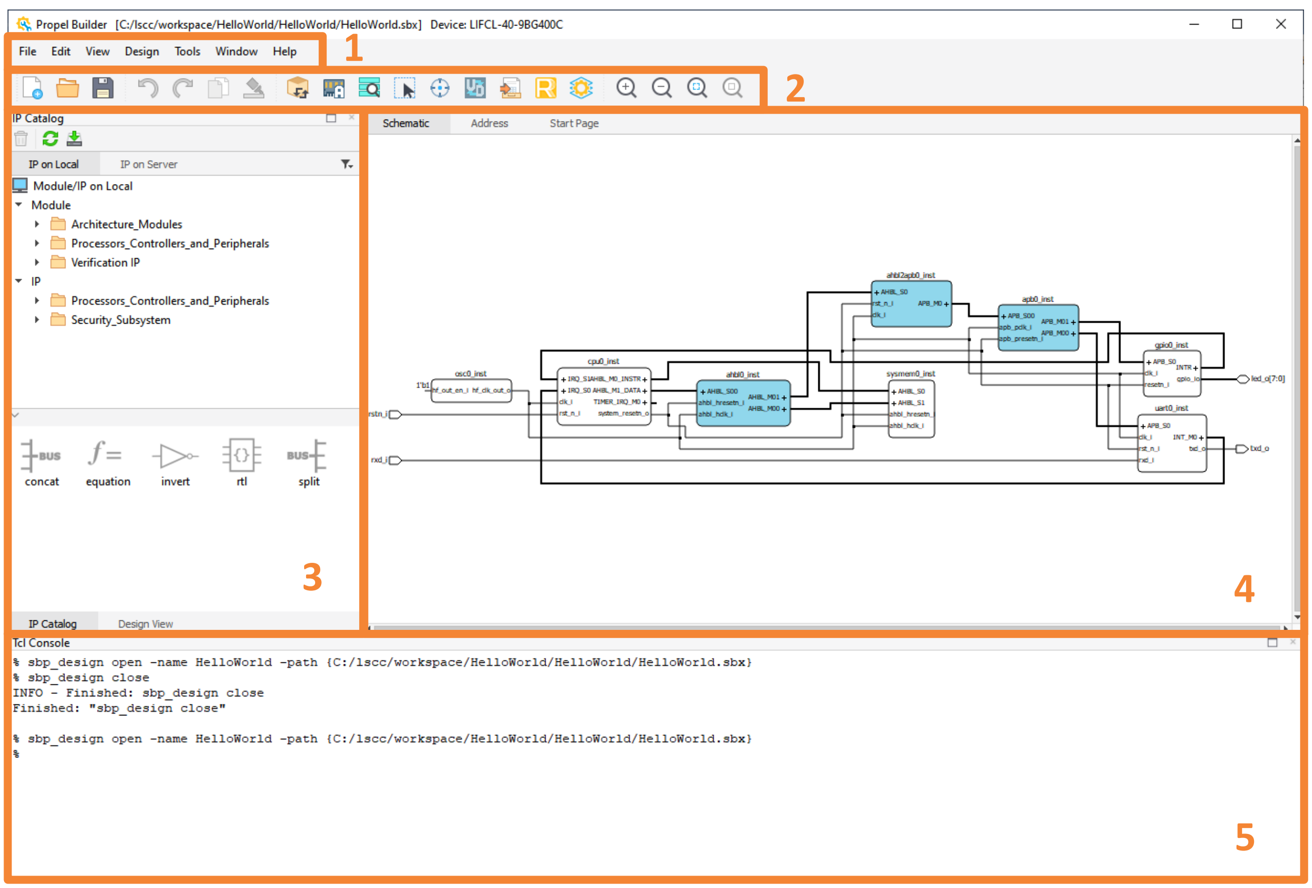


Figure 2.1. Propel Builder Workbench Window

2.2. Project Design Flow

2.2.1. Creating a New SoC Project

1. Choose **File >** **New Design** from the Lattice Propel Builder Menu bar. The Create System Design – Design Information wizard opens (Figure 2.2).

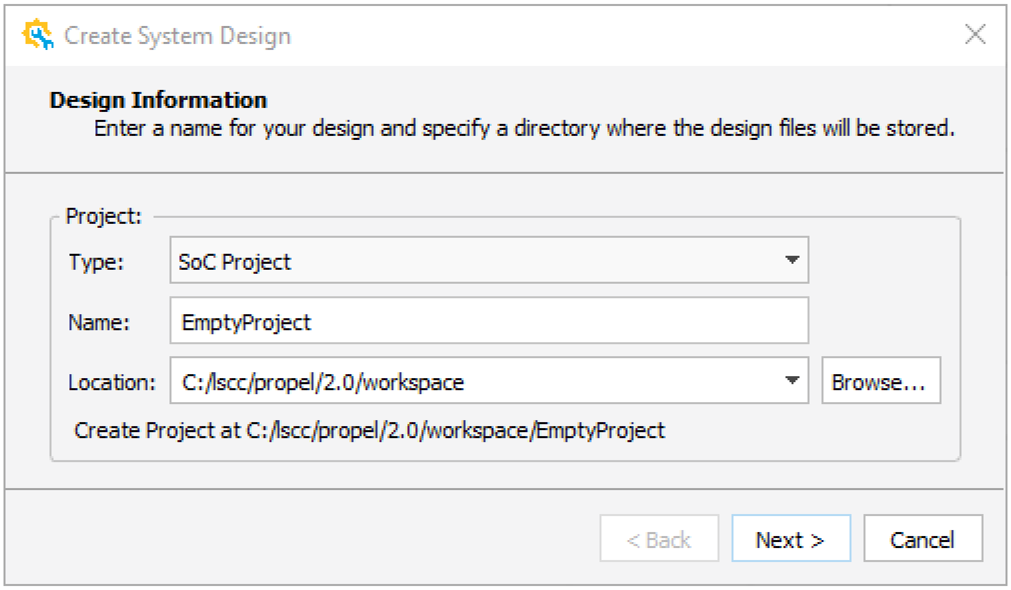


Figure 2.2. Create System Design – Design Information Wizard

2. The default Project Type is displayed in the **Type** field. Use the drop-down menu to choose SoC Verification if creating a verification project.
3. Enter the desired project name in the **Name** filed.
4. (Optional) The default location is shown in the **Location** field. Use the **Browse…** option to change the project workspace location.
5. Click **Next**. The Create System Design - Propel Project Configure wizard opens (Figure 2.3).

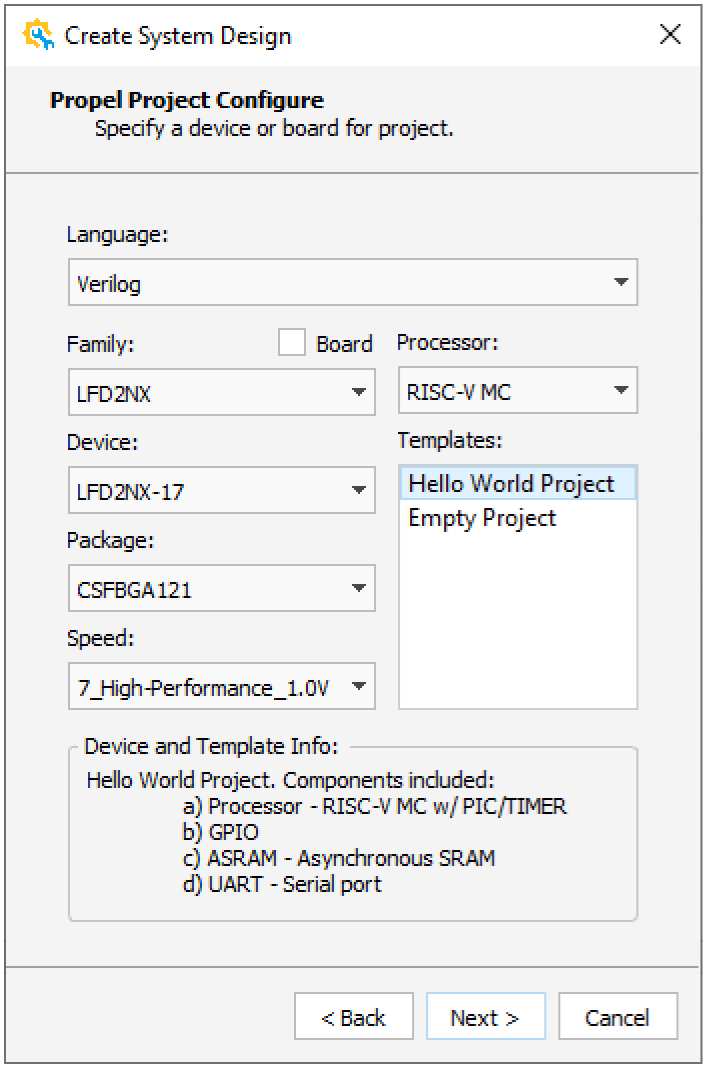


Figure 2.3. Create System Design - Propel Project Configure Wizard

6. (Optional) The default Hardware Description Language (HDL) is displayed in the **Language** field. Use the drop-down menu to change the default language.
7. Specify a device or board for project.
   - Use the drop-down menu to select desired device information (Family, Device, Package and Speed), and select Empty Project in the **Templates** section (Figure 2.4).

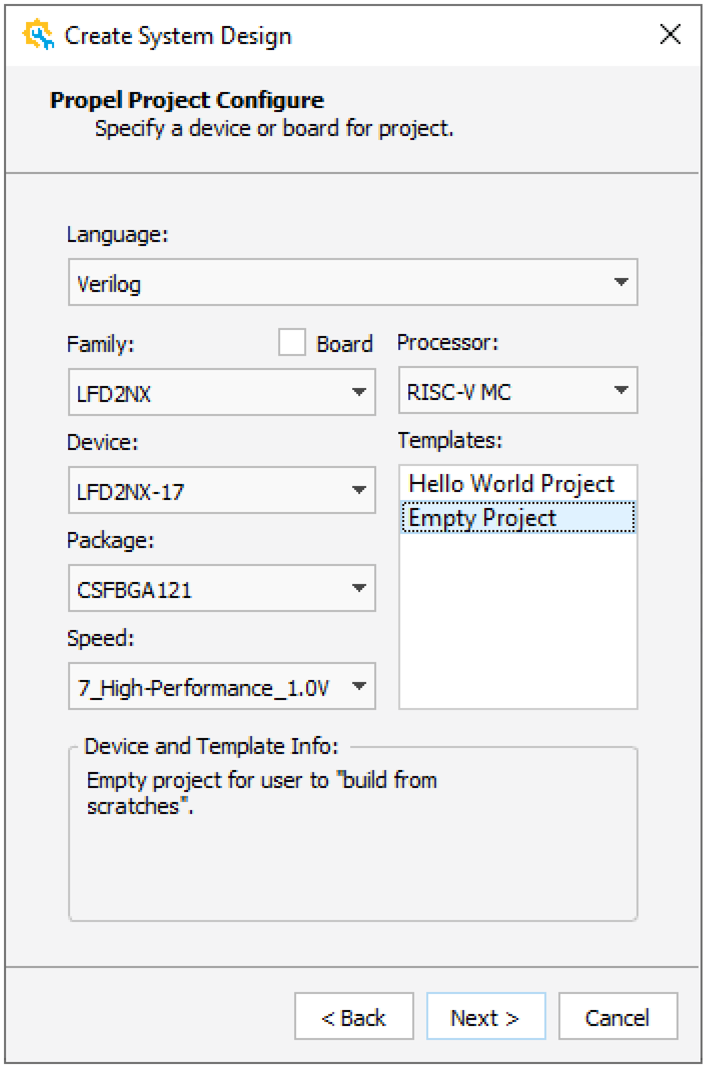


Figure 2.4. Specify a Device for Project

- Click **Board**. The Propel Project Configure wizard is shown in the **Board Select:** area (Figure 2.5). Select the desired board, such as Crosslink-NX Evaluation.

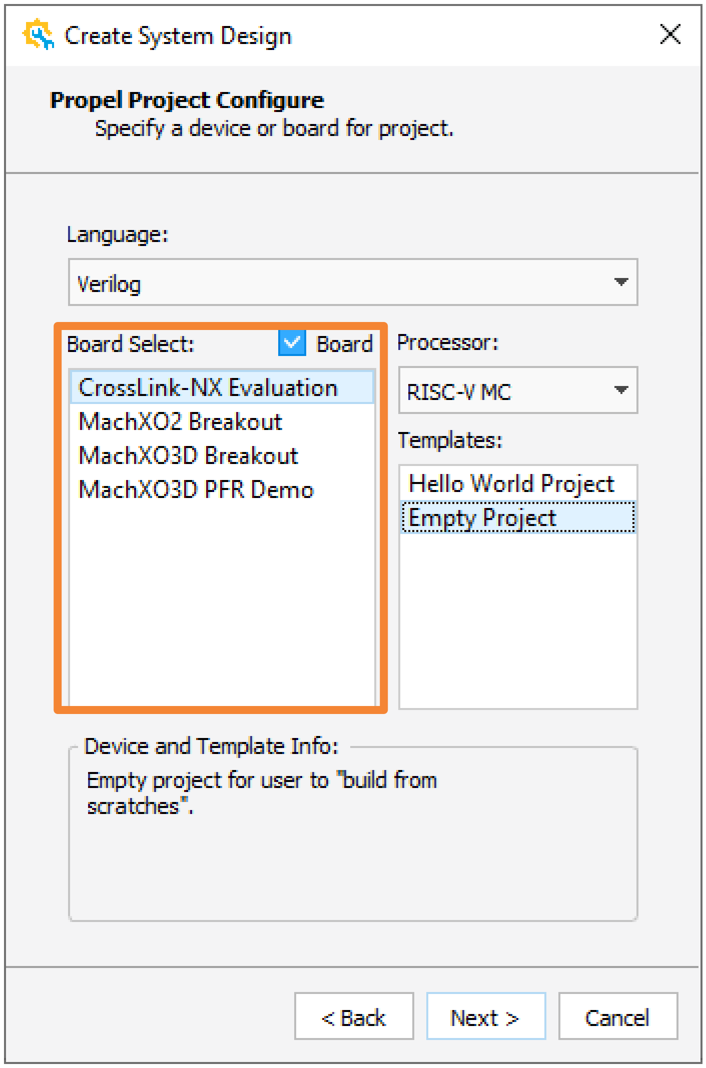


Figure 2.5. Specify a Board for Project

8. Click **Next**. The Project Information wizard opens (Figure 2.6).

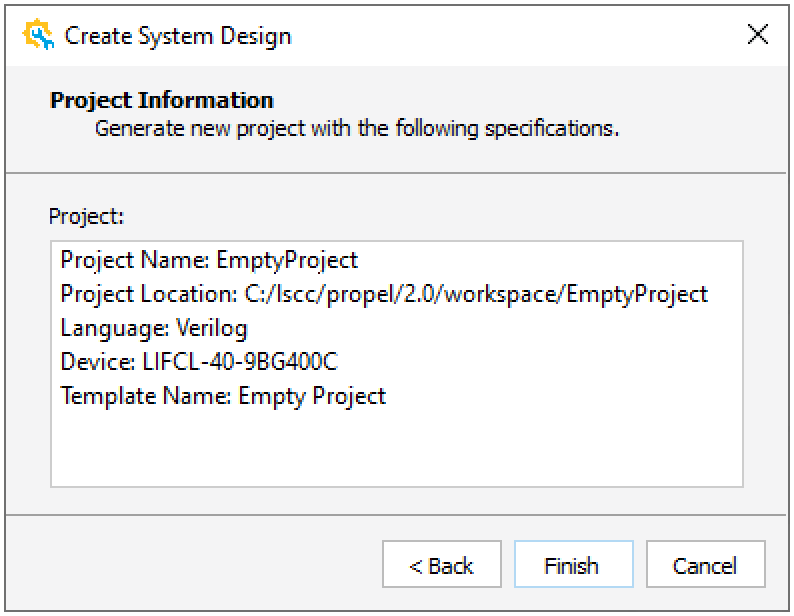


Figure 2.6. Create System Design - Project Information Wizard

9. Check and confirm the project information.
10. Click **Finish**. Propel Builder GUI opens (Figure 2.7).

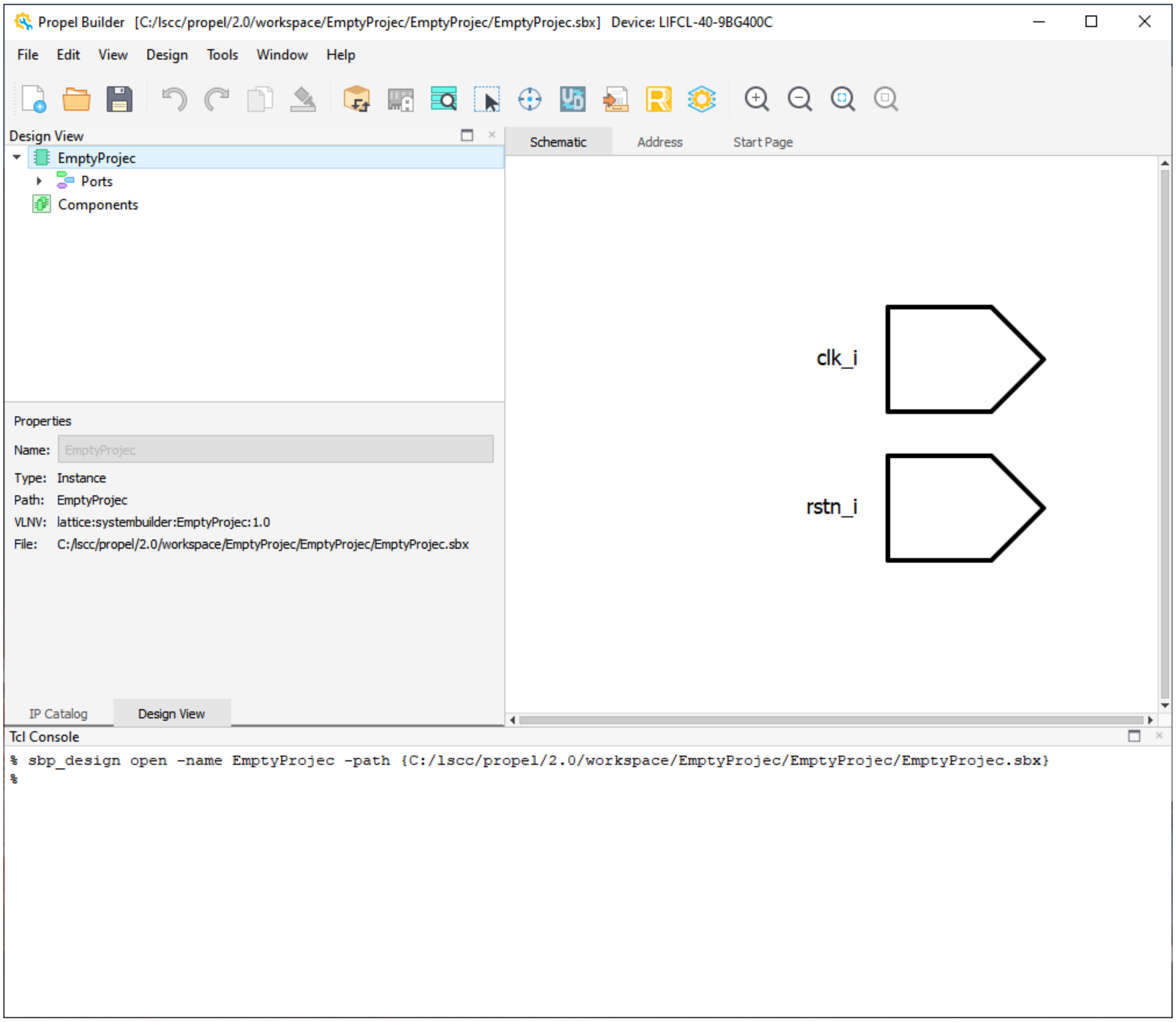


Figure 2.7. Propel Builder GUI Shows Empty Project

2.2.2. Creating Template SoC Project

1. Choose **File** >  **New Design** from the Lattice Propel Builder Menu bar. The Create System Design wizard opens (Figure 2.8).

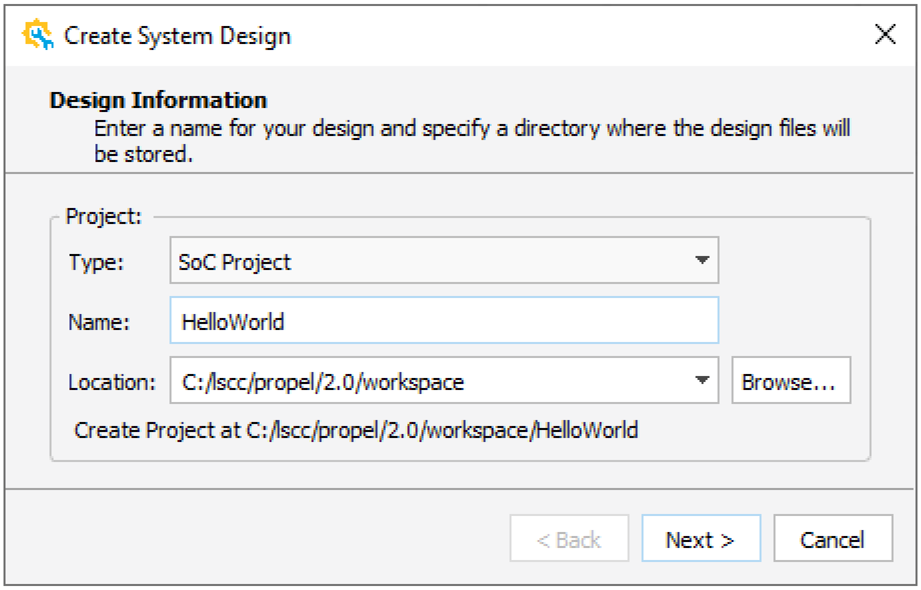


Figure 2.8. Create System Design – Design Information Wizard

2. The default Project Type is displayed in the **Type** field. Use the drop-down menu to choose SoC Verification if creating a verification project.
3. Enter a project name in the **Name** field, such as HelloWorld.
4. (Optional) The default location is shown in the **Location** field. Use the **Browse…** option to change the project workspace location.
5. Click **Next**. The Propel Project Configure wizard opens (Figure 2.3).
6. Click **Board**. The Propel Project Configure wizard is shown in the **Board Select** area. Select a desired Board, such as CrossLink-NX Evaluation.
7. Select a desired template from the **Templates** area, such as Hello World Project (Figure 2.9).

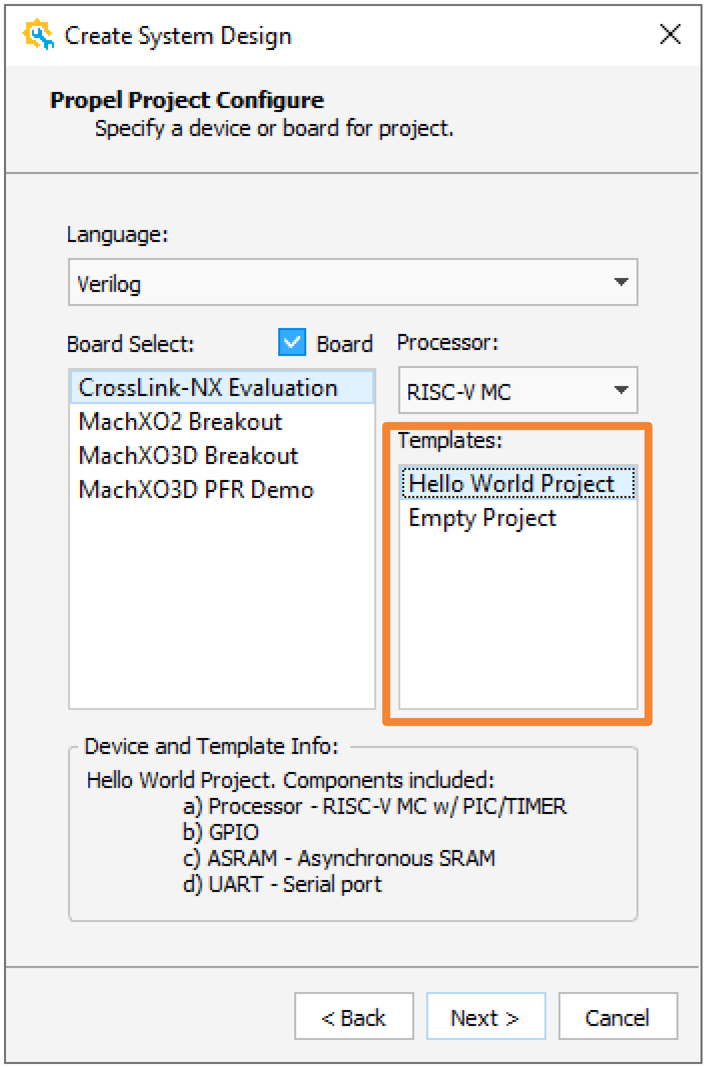


Figure 2.9. Specify a Board for Project

8. Click **Next**. The Project Information wizard opens (Figure 2.10).

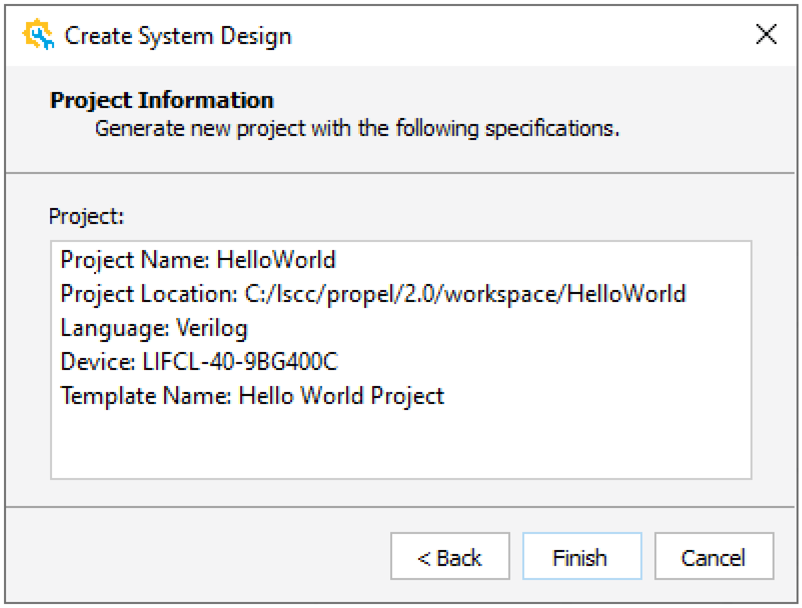


Figure 2.10. Project Information Wizard

9. Check and confirm the project information.
10. Click **Finish**. Propel Builder GUI opens (Figure 2.11).

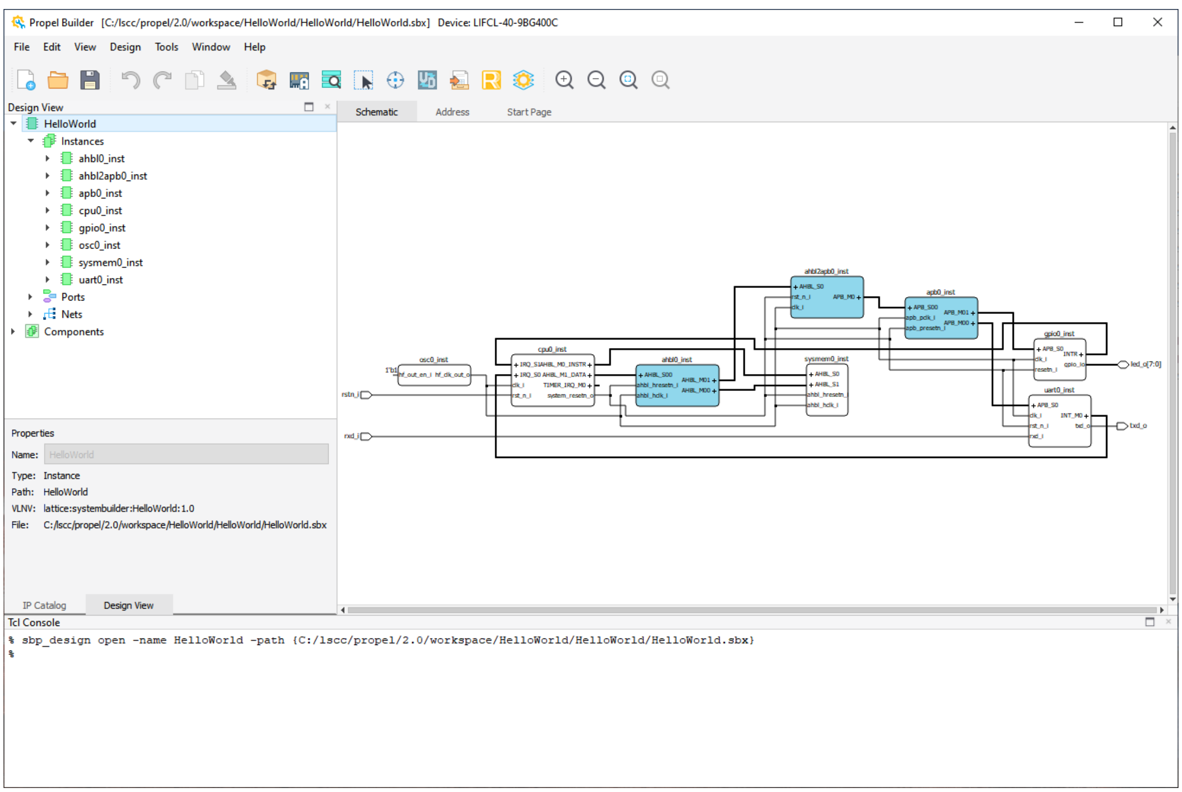


Figure 2.11. Propel Builder GUI shows Template Project

2.2.3. Opening an SoC Existing Project

1. Choose **File** > **Open Design** from Propel Builder GUI Menu, or Click **Open Design** icon from Propel Builder Toolbar. The Open sbx dialog opens (Figure 2.12).

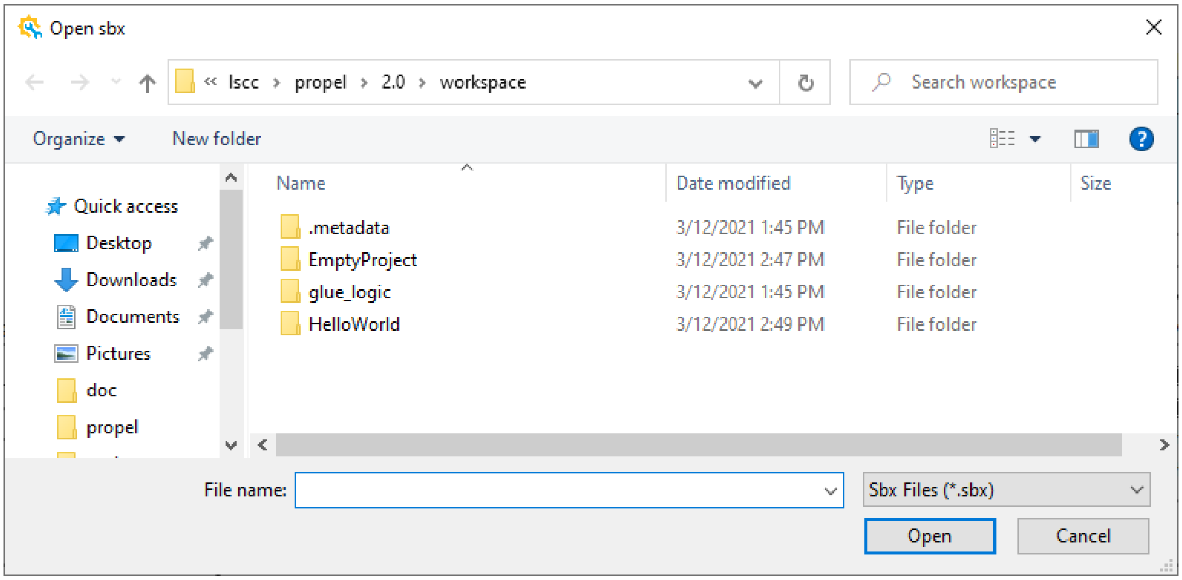


Figure 2.12. Open Sbx Dialog

2. Browse to find the default project workspace folder or your own project workspace folder. Choose the sbx file, such as HelloWorld.sbx (Figure 2.13).

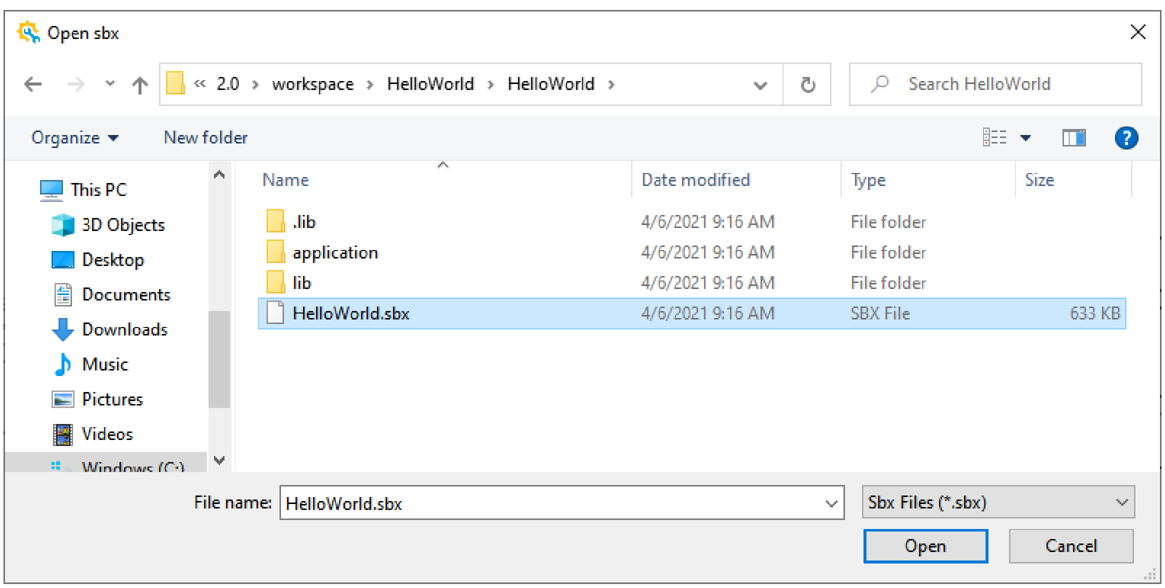


Figure 2.13. Open HelloWorld Project

3. Click **Open**. The Builder GUI shows the SoC design.

Note: Choose **File** > **Recent Designs** from Propel Builder GUI Menu. You can quickly open a most recently closed project.

2.2.4. Adding Modules

After starting a Propel Builder project, you can add modules by dragging them from the IP Catalog view to the Schematic view. The IP Catalog view comes with a large variety of modules for common use and some glue logic modules as well, which can be found from the **IP on Local** tab (Figure 2.14). Click the **IP on Server** tab to find and download more modules for specialized use.

1. (Optional) From the Propel Builder GUI **IP Catalog** area (Figure 2.14), choose the **IP on Server** tab. Select a desired IP. Click the  **Install** button. After the IP is installed successfully, the new IP can be shown in the **IP on Local** tab.

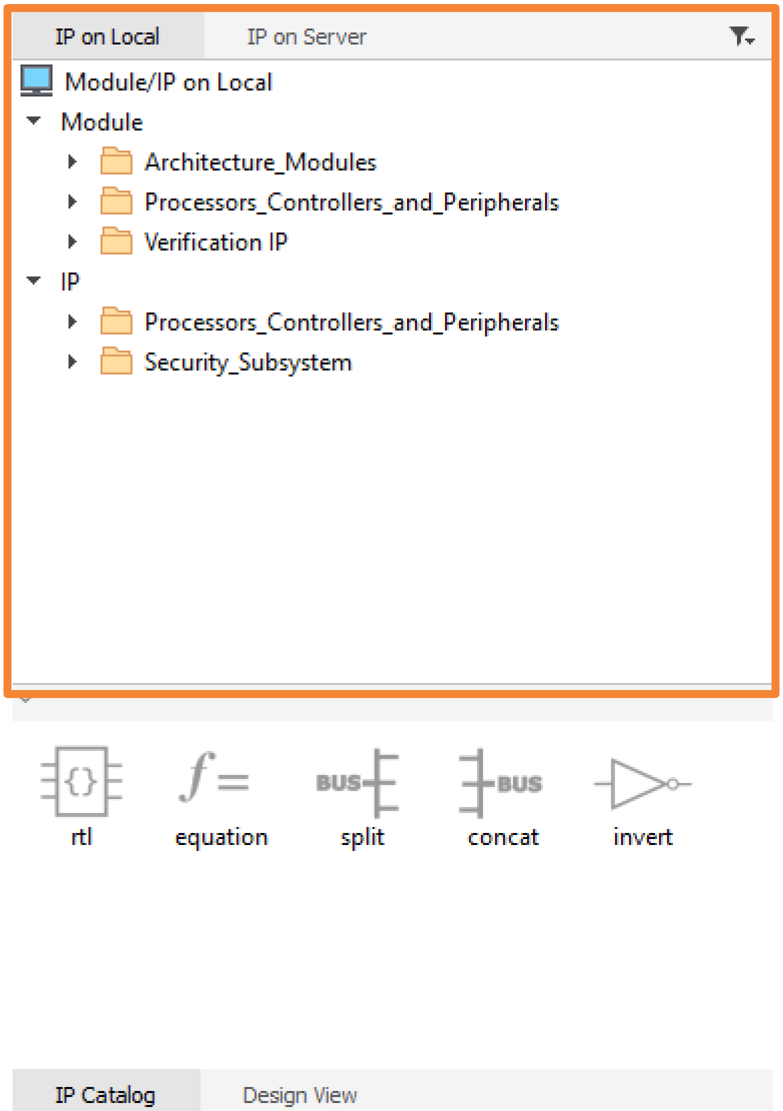


Figure 2.14. IP Catalog

2. From the **IP on Local** tab, select a desired IP, such as GPIO. Double-click the IP module or drag and drop the IP module to the **Schematic** view. A Module/IP Block wizard pops up (Figure 2.15).

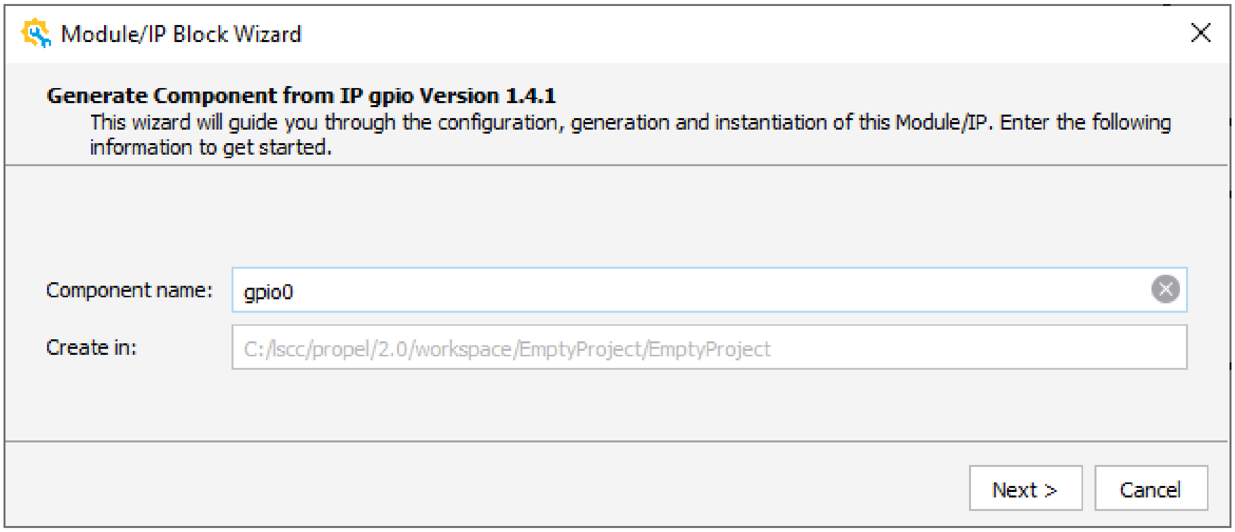


Figure 2.15. Module/IP Block Wizard – Generate Component from Module gpio Version 1.4.1

3. Enter a component name in the **Component name** area, such as gpio0. Click **Next**. Module/IP Block Wizard shows the **Configure Component from IP gpio** page (Figure 2.16).

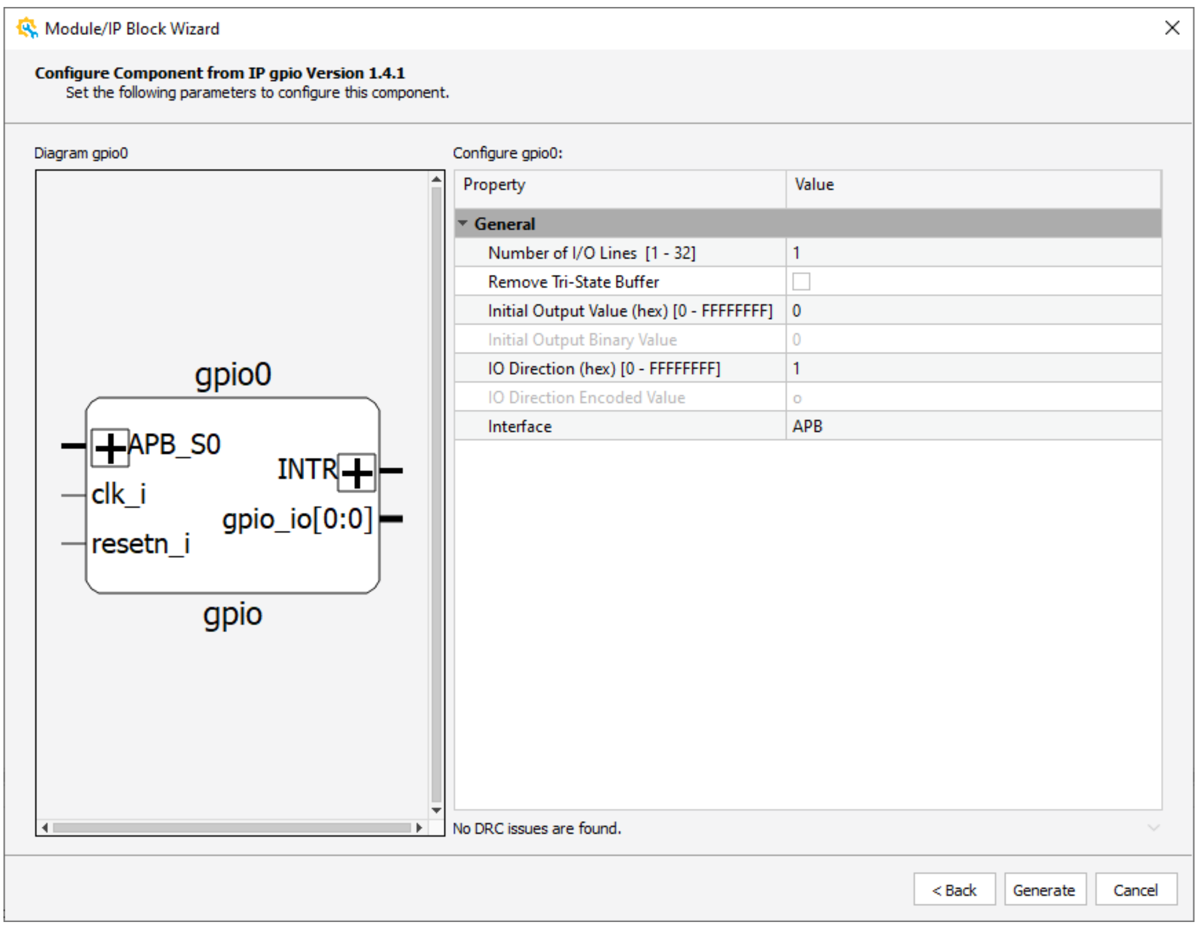


Figure 2.16. Module/IP Block Wizard - Configure Component from IP Gpio Version 1.4.1

4. (Optional) Click the **Value** field. You can enter a desired value for a property, check the checkbox to select a property, or select a desired value from the dropdown menu for a property in the **Value** area. By changing the value, the properties are thus configured. The property in gray is not configurable.
5. Click **Generate**. Module/IP Block wizard shows the Check Generated Result page (Figure 2.17).

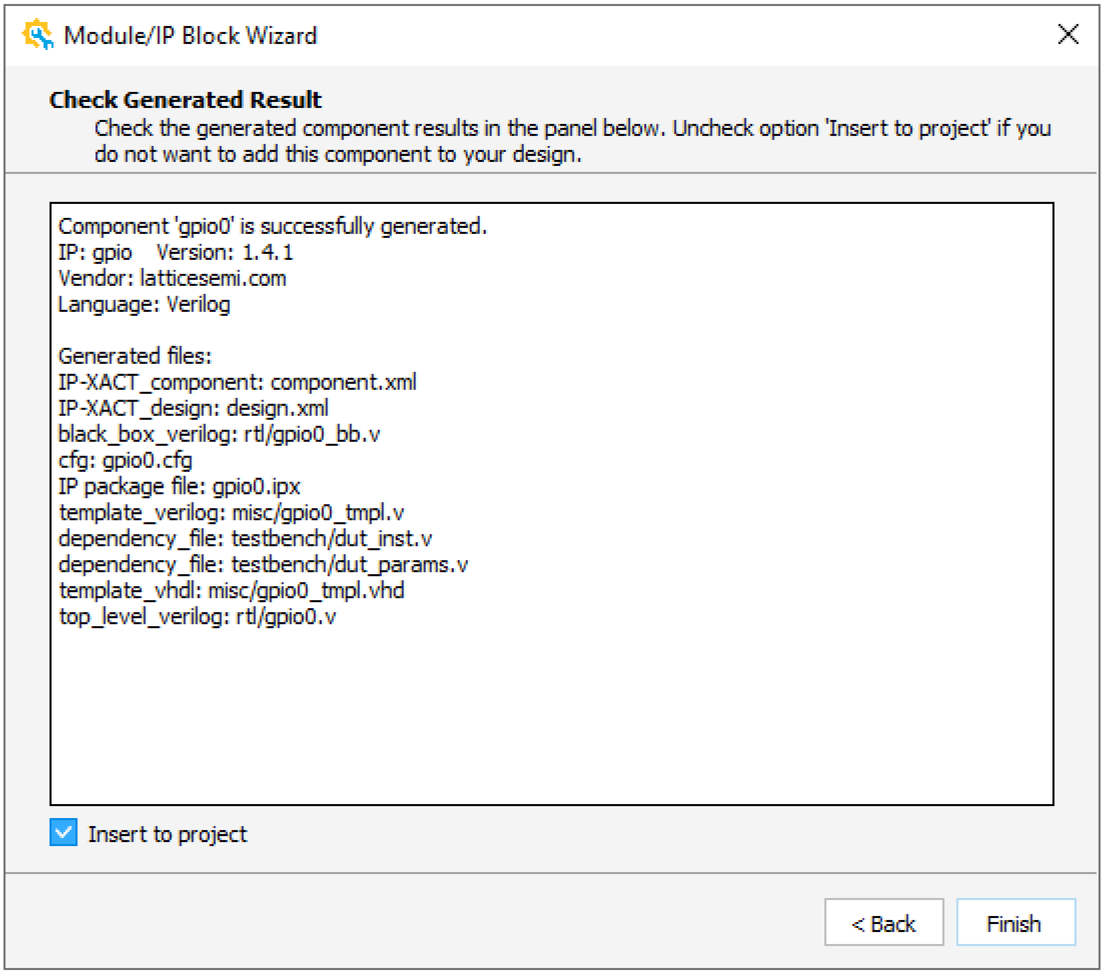


Figure 2.17. Module/IP Block Wizard - Check Generated Result

6. Select **Insert to project** (Figure 2.17) at the bottom of the Check Generate Result wizard.
7. Click **Finish**. The **Define Instance** dialog box opens (Figure 2.18).

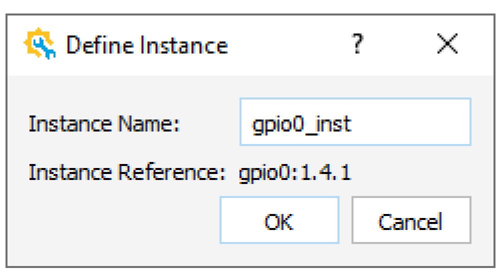


Figure 2.18. Design Instance Dialog Box

8. (Optional) Change the instance name, if desired. Space and special characters are not allowed.
9. Click **OK**. The schematic block for the module appears in the **Schematic** view (Figure 2.19).

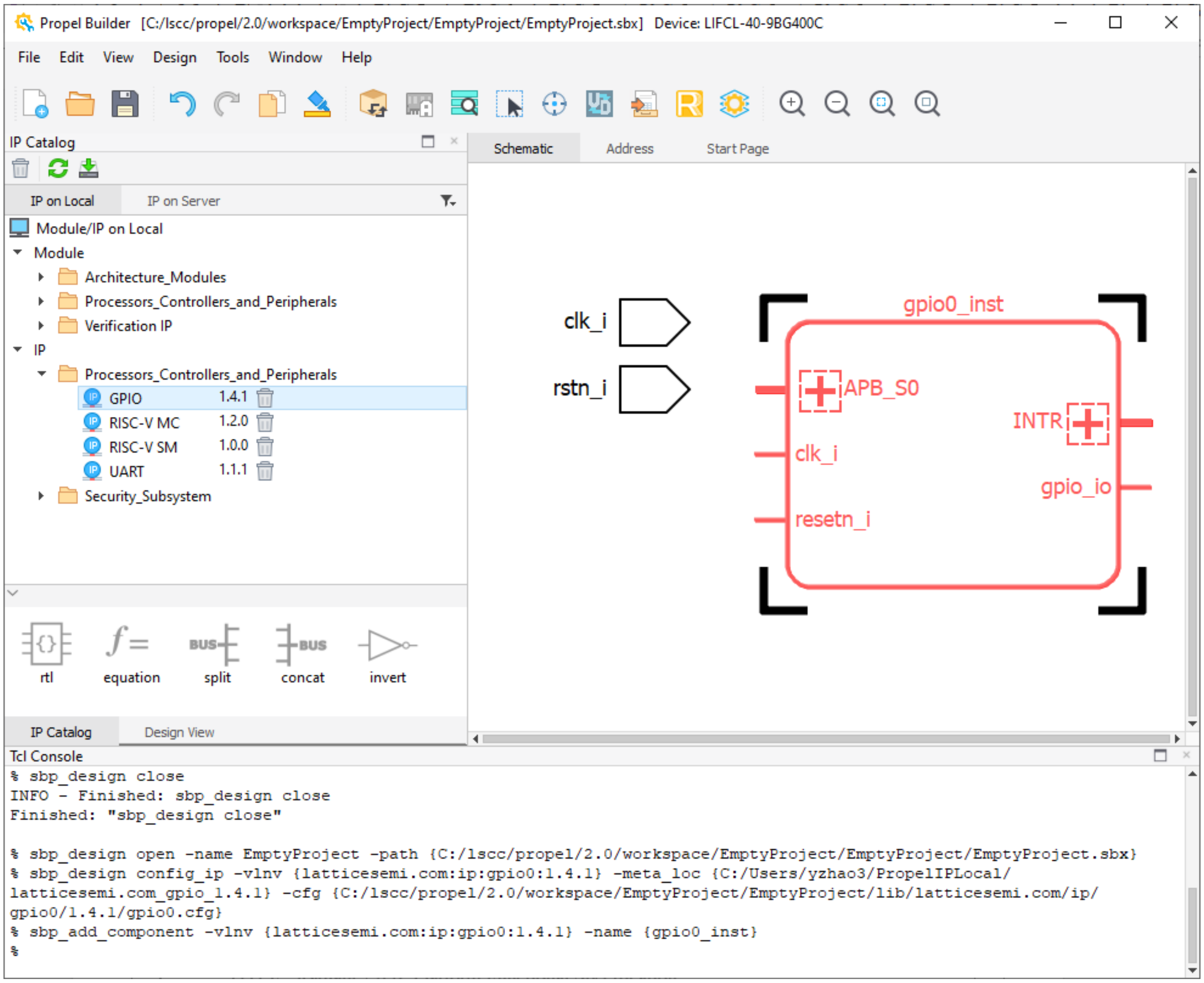
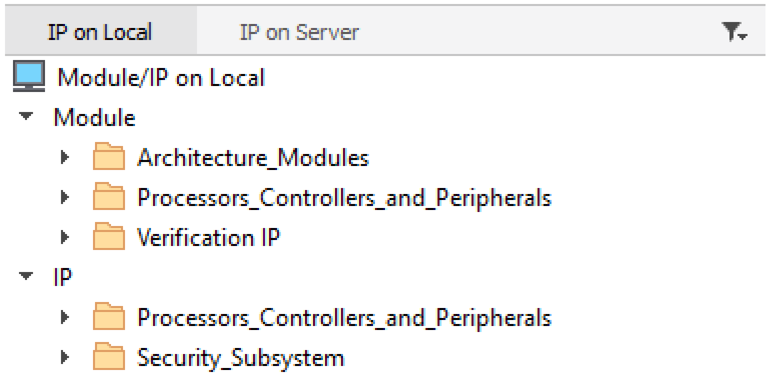


Figure 2.19. Propel Builder Schematic View Shows the Module Instance

2.2.5. Adding Glue Logic

Propel 2.0 builder supports glue logic modules as well as IP modules. You can add glue logic modules by dragging them from the IP Catalog view (Figure 2.20) to the Schematic view.



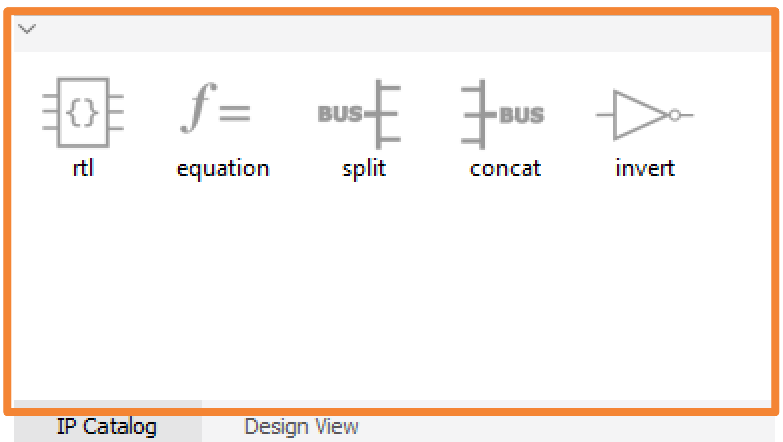


Figure 2.20. IP Catalog

- Concat
  a. From the IP Catalog, select the **concat** module. Double-click the concat module or drag and drop concat module to the Schematic view. A Glue Logic wizard (Figure 2.21) pops up.

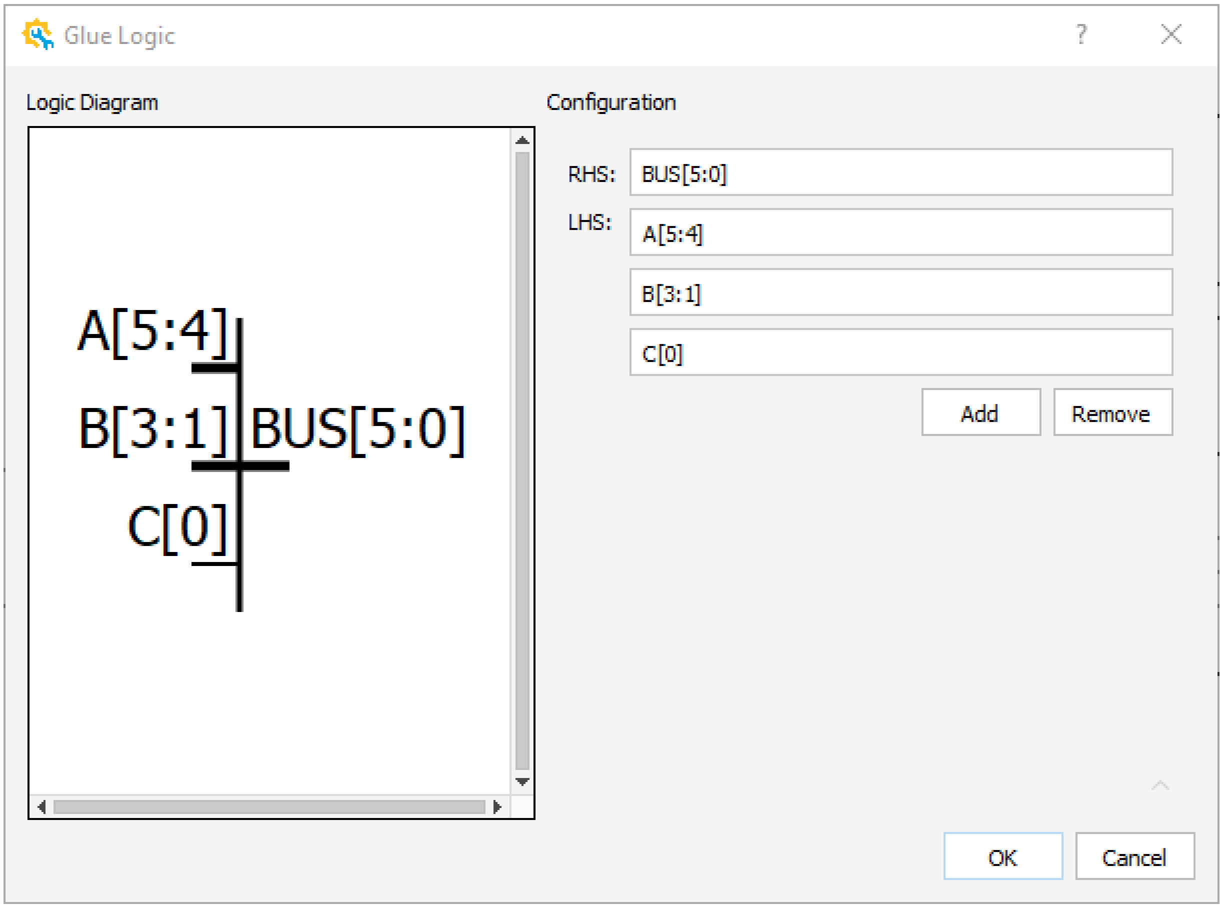


Figure 2.21. Glue Logic for Concat Module

b. Click the **RHS** field to change the default right-hand side bus width to a desired one. Click the **LHS** field to change the default left-hand side bus width to a desired one. Click the Add button to add LHS (Left-hand Side). Click the Remove button to remove LHS (Left-hand Side). You can only add or remove LHS but not RHS. LHS and RHS are thus configured.

c. Click **OK**. The schematic block for the concat module in red appears in the Schematic view (Figure 2.22).

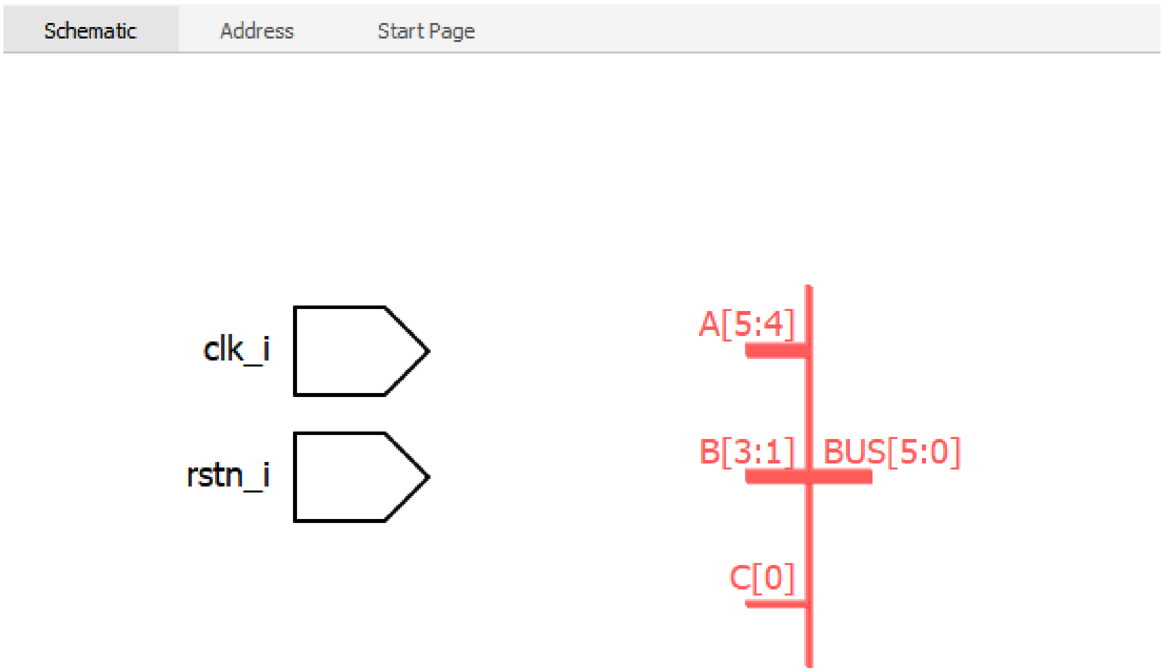


Figure 2.22. Schematic View Shows a Concat Module

- Equation
  a. From the IP Catalog, select the **equation** module. Double-click the equation module, or drag and drop the equation module to the Schematic view. A Glue Logic wizard (Figure 2.23) pops up.

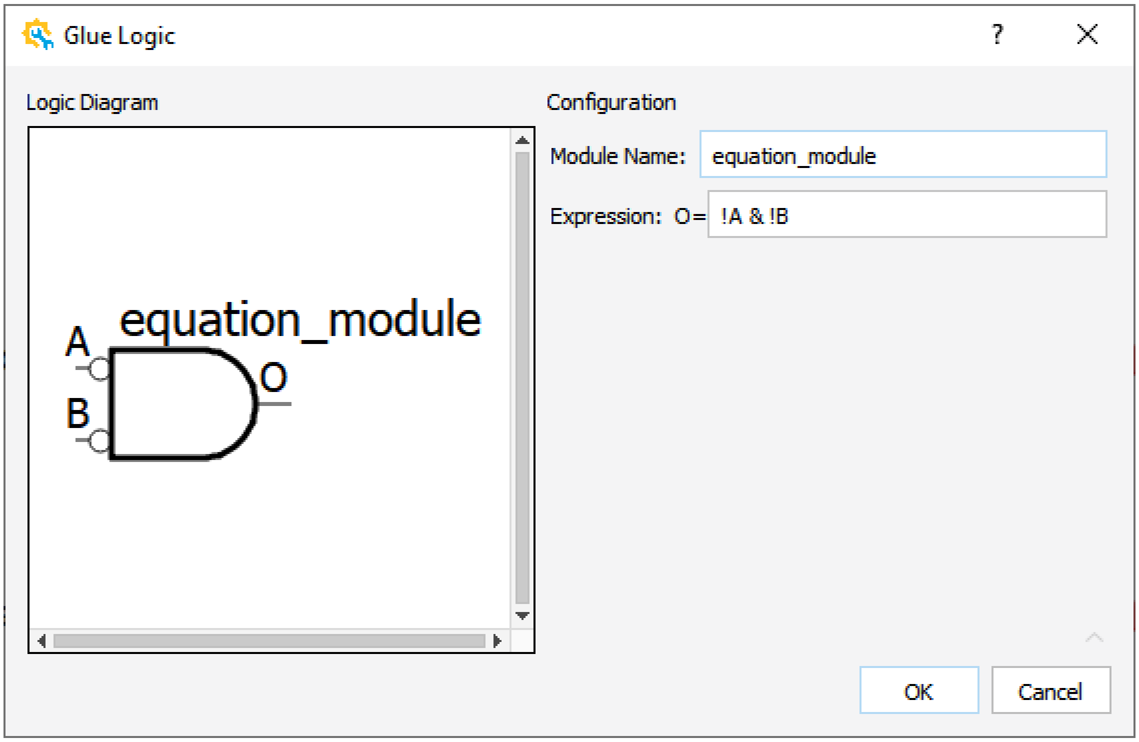


Figure 2.23. Glue Logic Wizard for Equation Module

b. Click the **Module Name** field to change the default value to a desired name. Click the **Expression** field to change the default expression to a desired one. The expression supports and (&), or (|), negation (!). The module name and expression are thus configured.

c. Click **OK.** The schematic block for the equation module in red appears in the Schematic view (Figure 2.24).

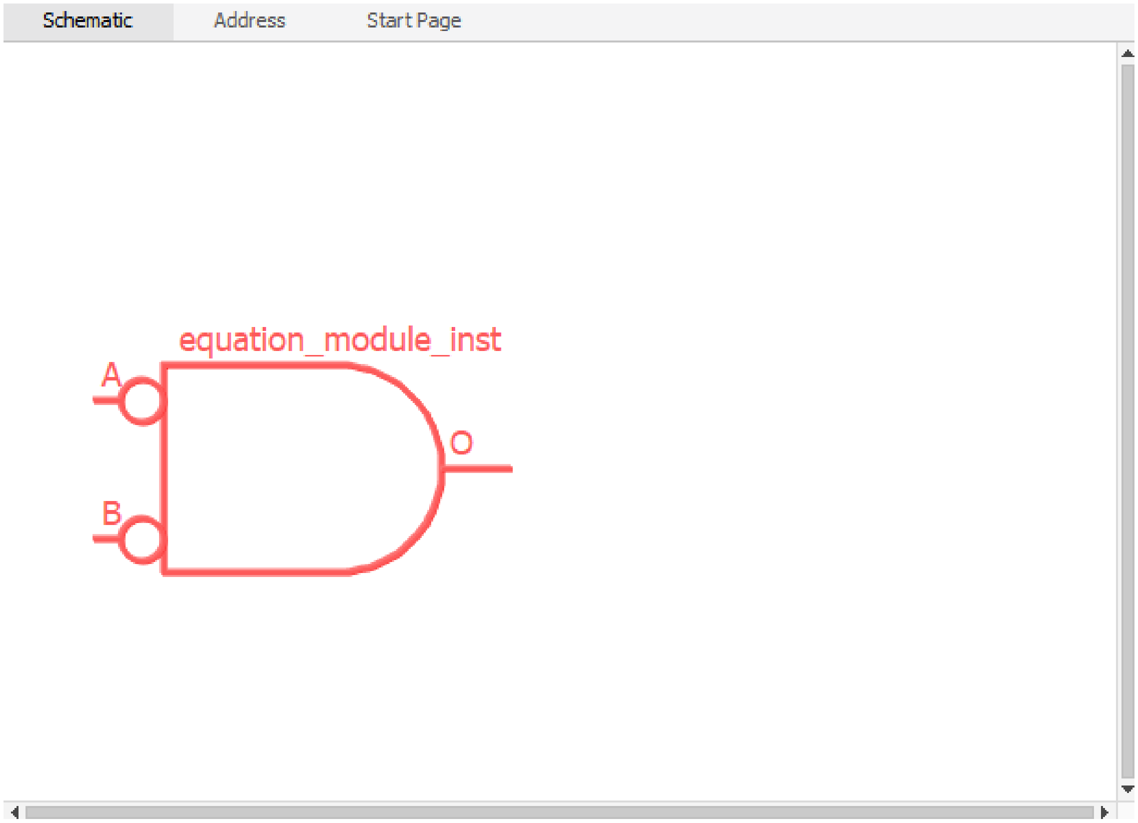


Figure 2.24. Schematic View Shows an Equation Module

- Invert
  a. From the IP Catalog, select the **invert** module. Double-click the invert module, or drag and drop invert module to the Schematic view. The schematic block for the invert module in red appears in the Schematic view (Figure 2.25).

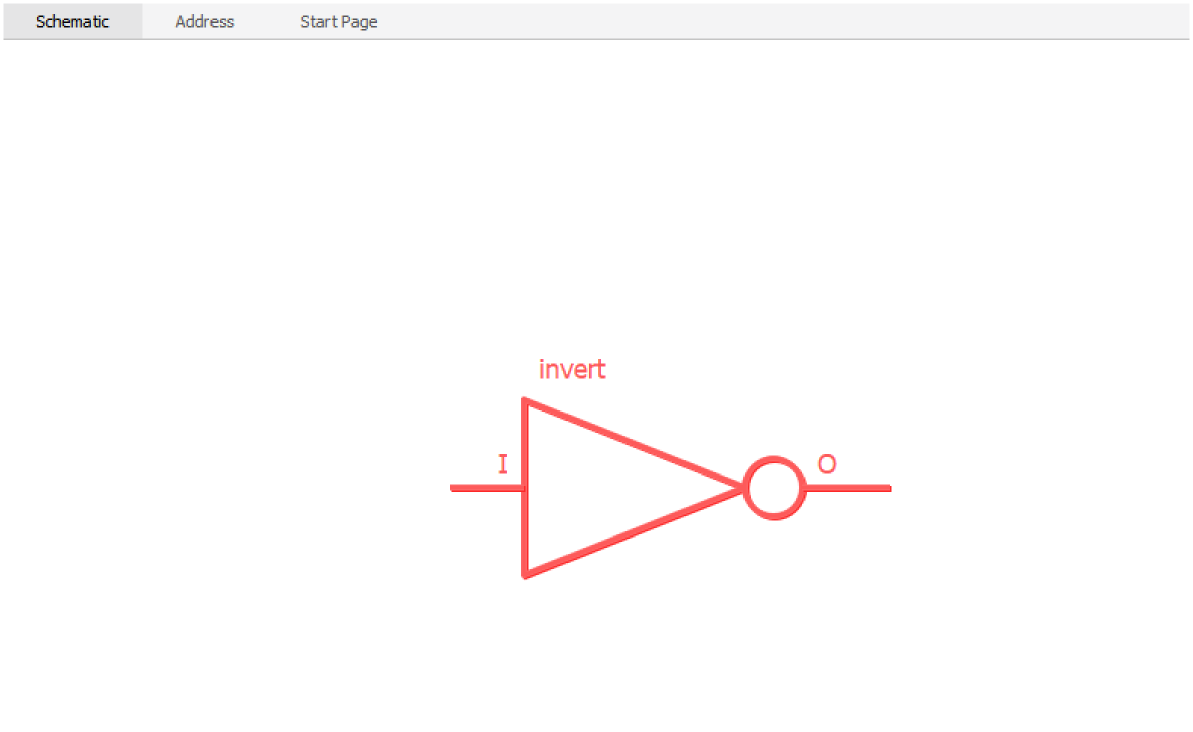


Figure 2.25. Schematic View Shows the Invert Module

- Rtl
  a. From the IP Catalog, select the **rtl** module. Double-click the rtl module or drag and drop the rtl module to the Schematic view. A Glue Logic wizard (Figure 2.26) pops up.

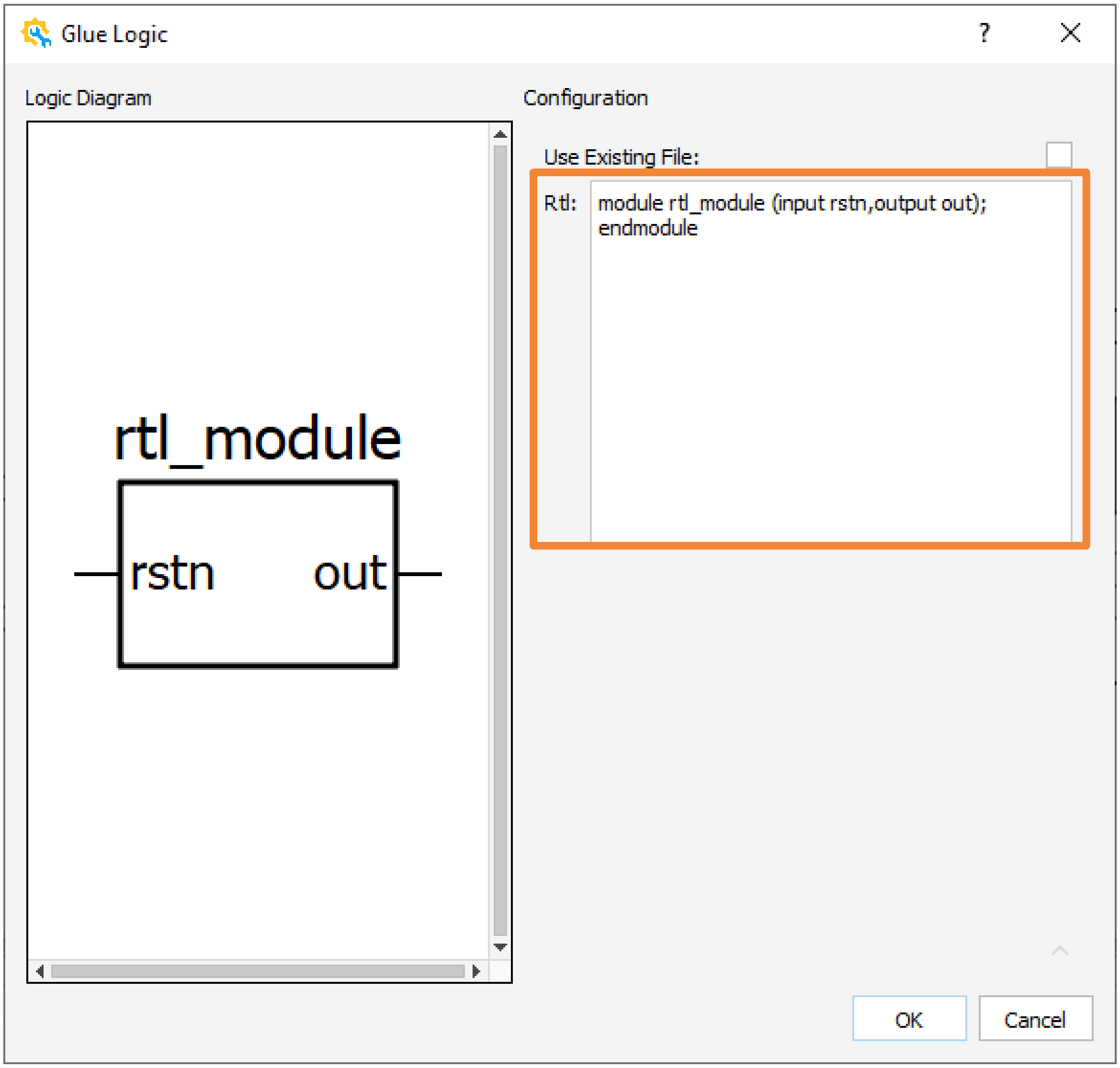


Figure 2.26. Glue Logic Wizard for Rtl Module

b. You can edit your own rtl module in the **Rtl** area by entering the rtl module function.

c. Or, you can use an existing rtl module by checking the **Use Existing File** checkbox. After checking the **Use Existing File** checkbox (Figure 2.27), browse to find the existing RTL module file path from the **Path** filed. And, check the **CopyToDesign** checkbox, if you want to copy the rtl module to the design.

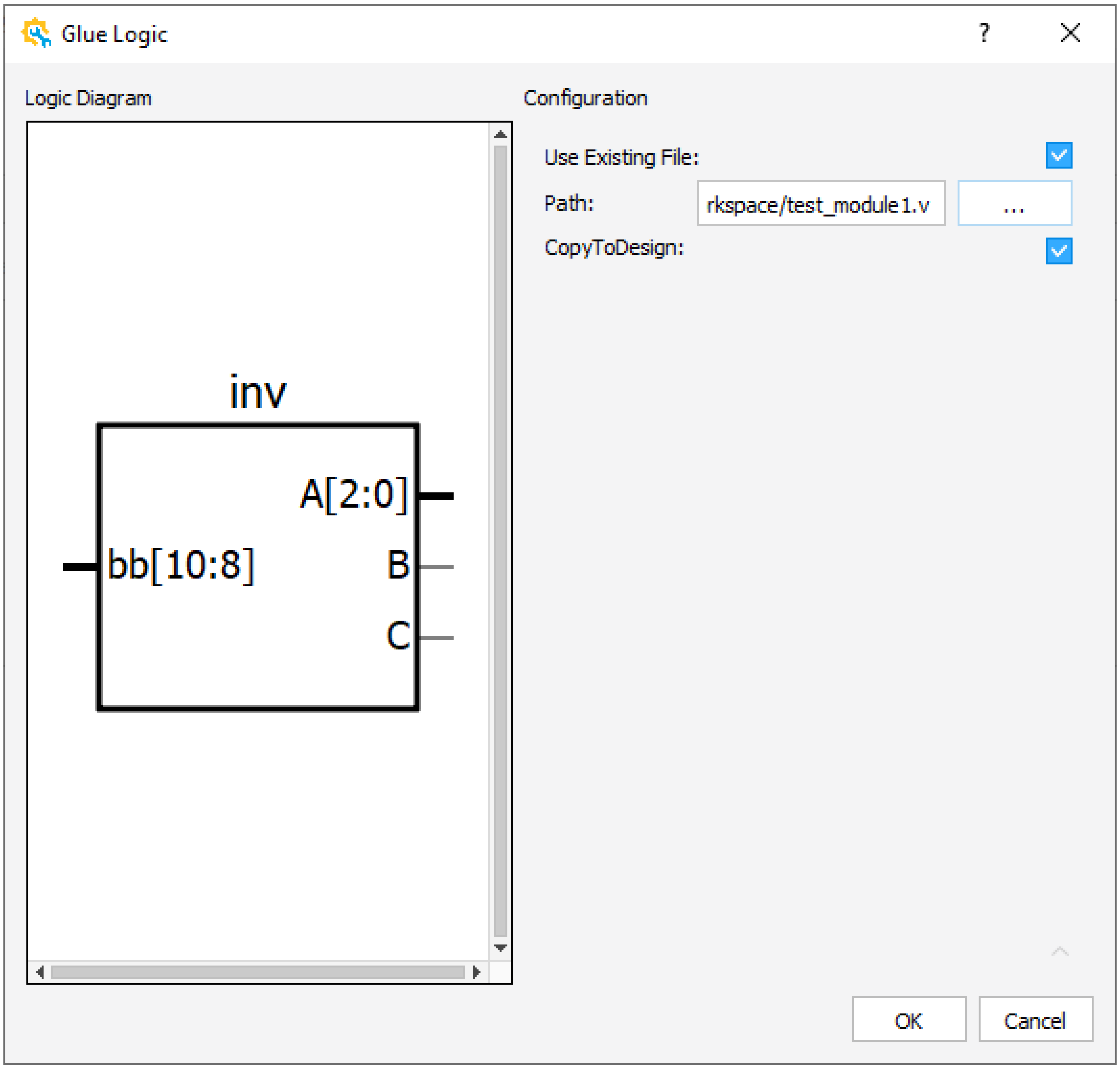


Figure 2.27. Existing Rtl Module Configuration

d. Click **OK**. The schematic block for the rtl module in red appears in the Schematic view (Figure 2.28).

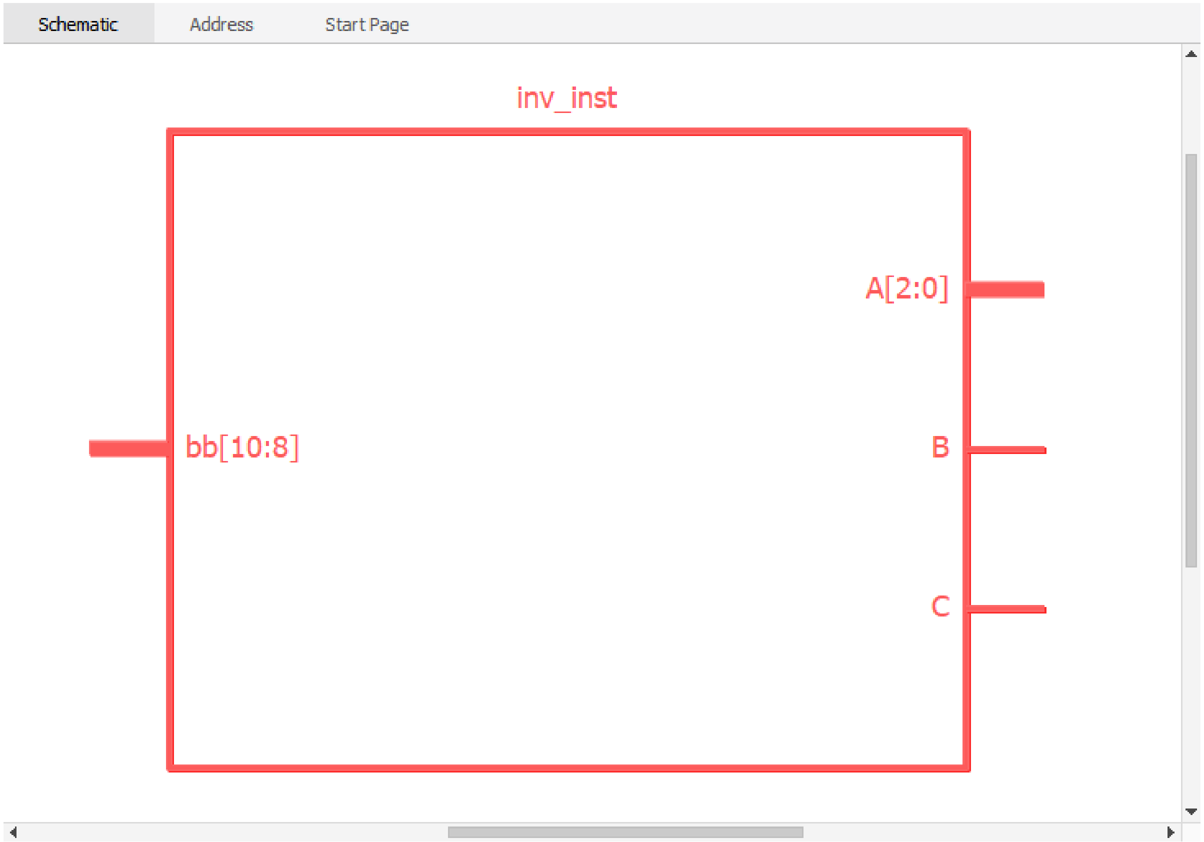


Figure 2.28. Schematic View Shows the Custom Rtl Module

Note: Currently, Rtl Module only supports Verilog HDL.

- Split
  a. From the IP Catalog, select the **split** module. Double-click the split module, or drag and drop the split module to the Schematic view. A Glue Logic wizard (Figure 2.29) pops up.

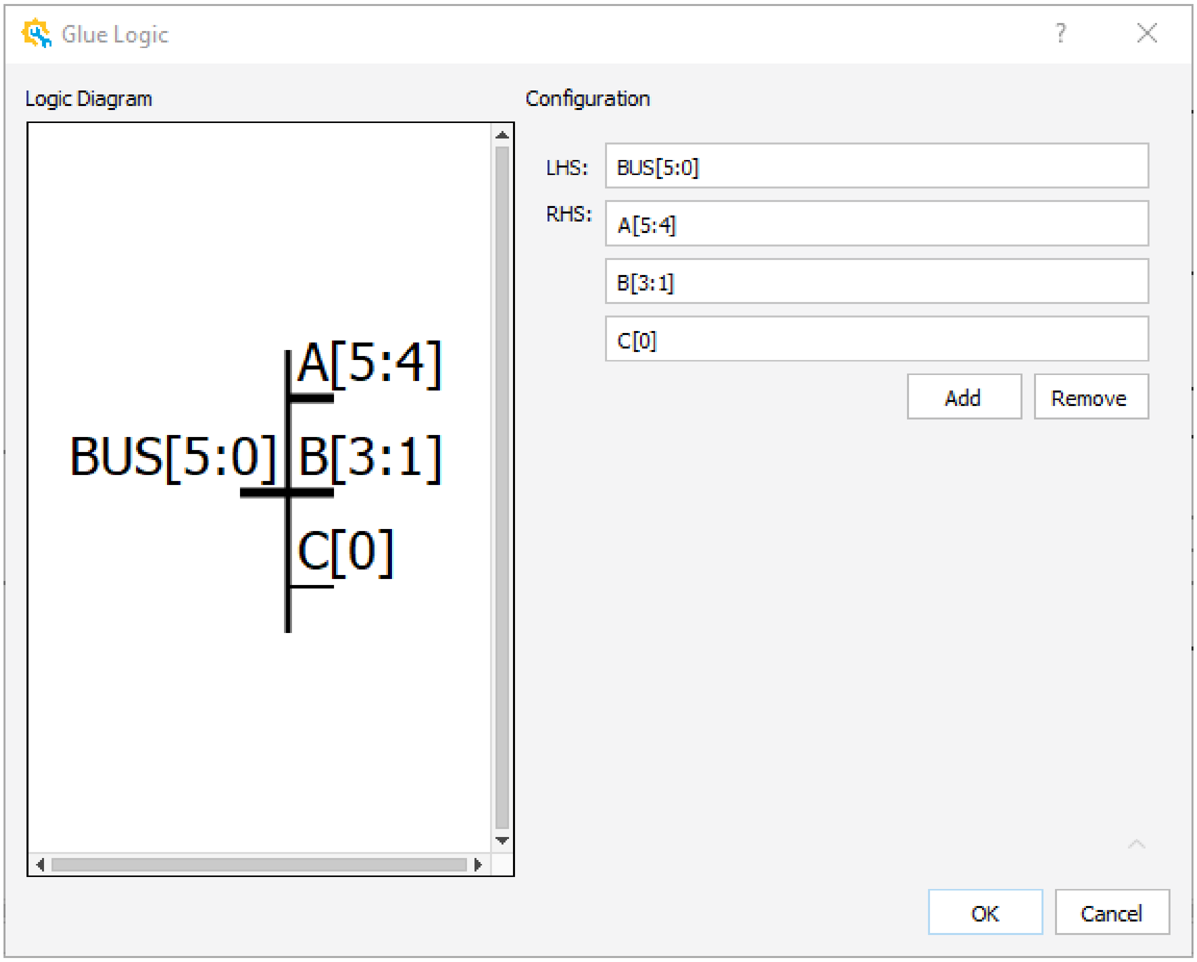


Figure 2.29. Glue Logic Wizard for Split Module

b. Click **LHS** field to change the default left-hand side bus widths to a desired one. Click **RHS** field to change the default right-hand side bit width to a desired one. Click the Add button to add an RHS. Click the Remove button to remove an RHS. You can only add or remove RHS but not LHS. LHS and RHS are thus configured.

c. Click **OK**. The schematic block for the split module in red appears in the Schematic view (Figure 2.30).

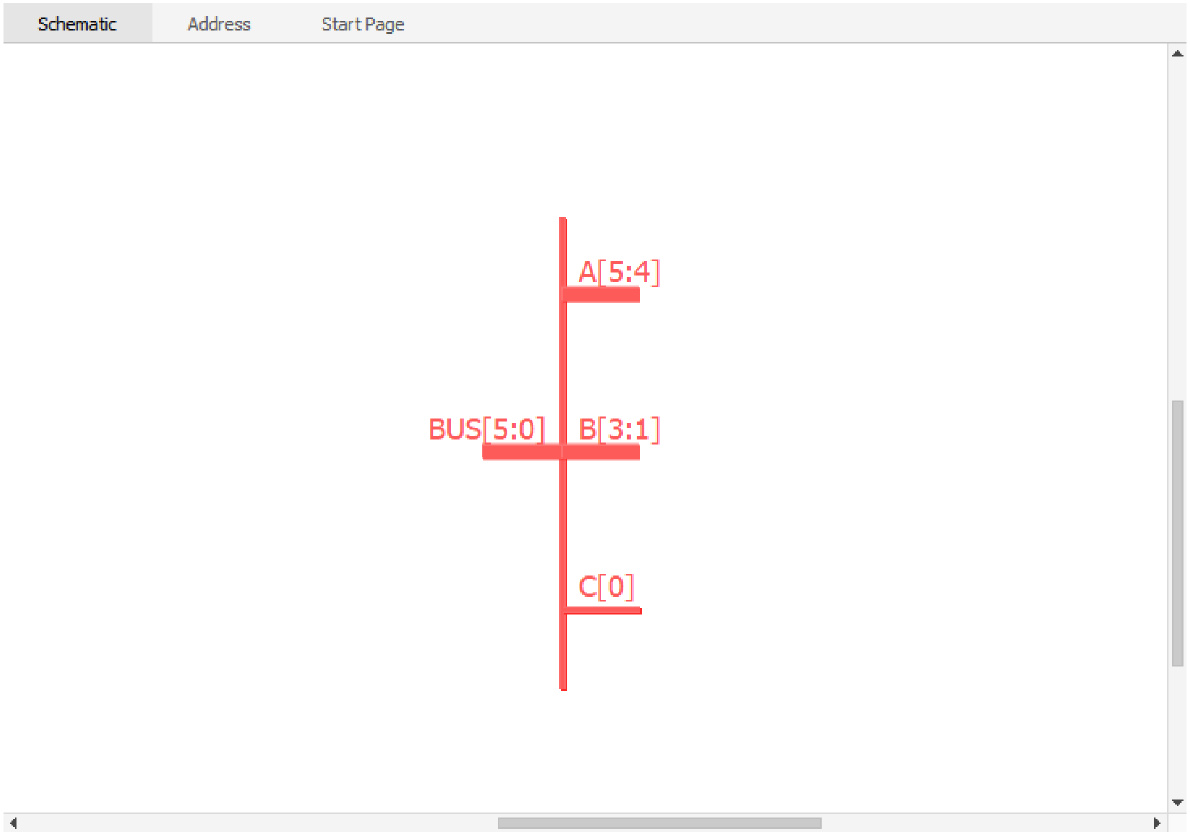


Figure 2.30. Schematic View Shows Split Module

2.2.6. Undo/Redo

Propel 2.0 Builder supports one-level undo/redo. You can click the Undo icon from Propel Builder Toolbar to go back to last action. Click the Redo icon from Propel Builder Toolbar to recover last undo action.

- Undo: Choose **Edit** > **Undo** from the Lattice Propel Builder Menu bar. Or, click the **Undo** icon from Propel Builder GUI Toolbar.
- Redo: Choose **Edit** > **Redo** from the Lattice Propel Builder Menu bar. Or, click the **Redo** icon from Propel Builder Toolbar.

Note: Currently, Propel 2.0 Builder only supports single undo/redo.

2.2.7. Working with the Schematic View

You can make changes in the Schematic view including automatic layout to clean up the display, moving, resizing, renaming blocks manually, highlighting objects and zooming the display in and out.

- To view signal list of block:

  Click the plus sign of the desired block to see what signals it contains. The plus sign changes to a negative sign and shows the signal list as shown in Figure 2.31. Click the negative sign to close the expanded bus. The schematic returns to the previous form.

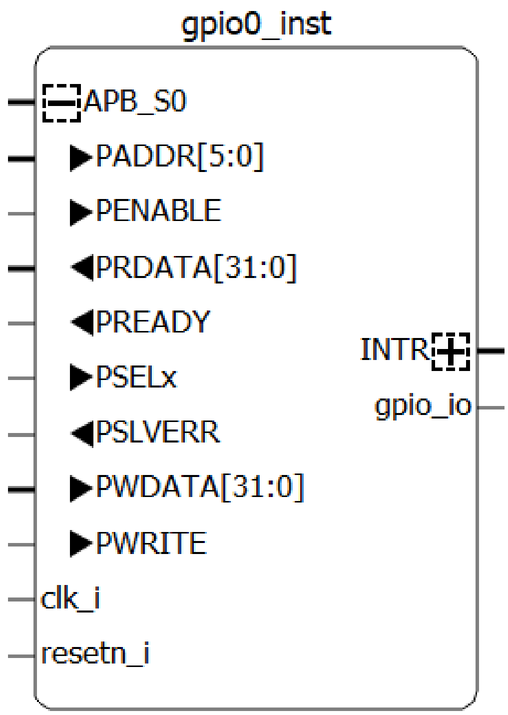


Figure 2.31. Signal List of Modules

- To select one or more objects:

  Select one object or more objects in one of the following ways:

  - Click on the object in the Schematic view. The selected object turns to red (Figure 2.32).

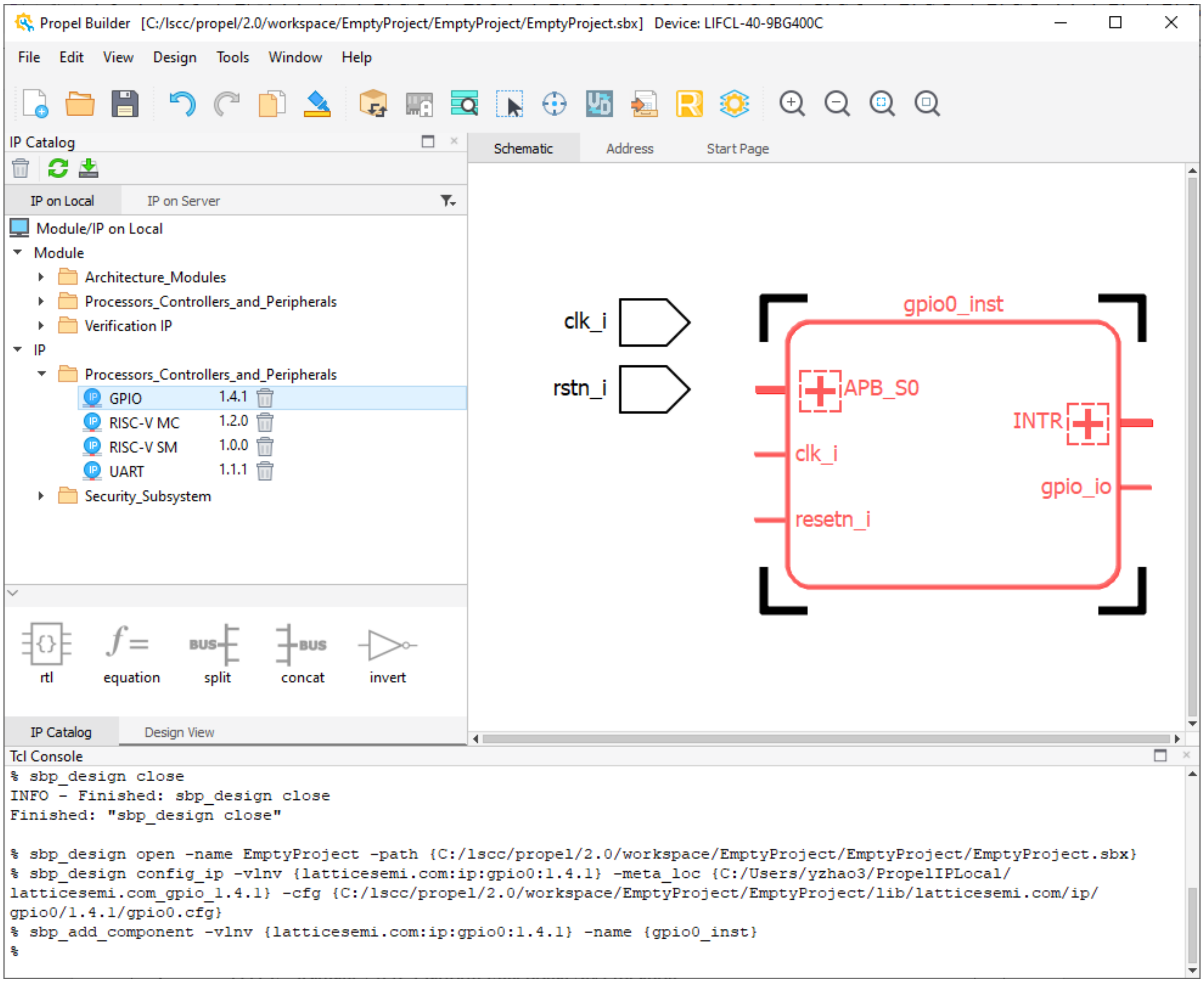


Figure 2.32. Select Object

- Ctrl-click or Shift-click in the Schematic view to select more than one objects.
- Click the Area_select icon  from the Propel Builder Toolbar. Click and drag to draw a selection rectangle around the modules and ports in the Schematic view. Click the icon again to turn off the Area_select mode.
- From the Schematic view, Right-click and choose Select All or press Ctrl-A to select all of the objects.

Note: Ctrl-click or Shift-click on the object in the Schematic view can also de-select the object while leaving the others selected.

- To re-arrange the schematic:

  Propel Builder allows re-arranging the objects in the schematic view. You can re-arrange modules and the ports. Drag objects to re-arrange the schematic. Propel Builder has rules for placing objects so as to adjust the schematic in an organized arrangement.

  a. Select the desired modules (one or more modules can be dragged at the same time) or ports (one or more ports can be dragged at the same time).
  b. Click on the selected items and drag it to the desired location.
  c. Release the mouse button.

  Note: The selected objects can be moved to the specified location, or as near as the rules allow. Other objects in the schematic can also be moved to accommodate the selected objects' new location.

- To bring selected objects to the center of the Schematic view using the Locate Object mode:
  a. Click the **Locate Object** icon from the Propel Builder Toolbar. The background turns to dark gray.
  b. Select the object in the List view of the Design View. The selected object is in the center of the Schematic View (Figure 2.33).

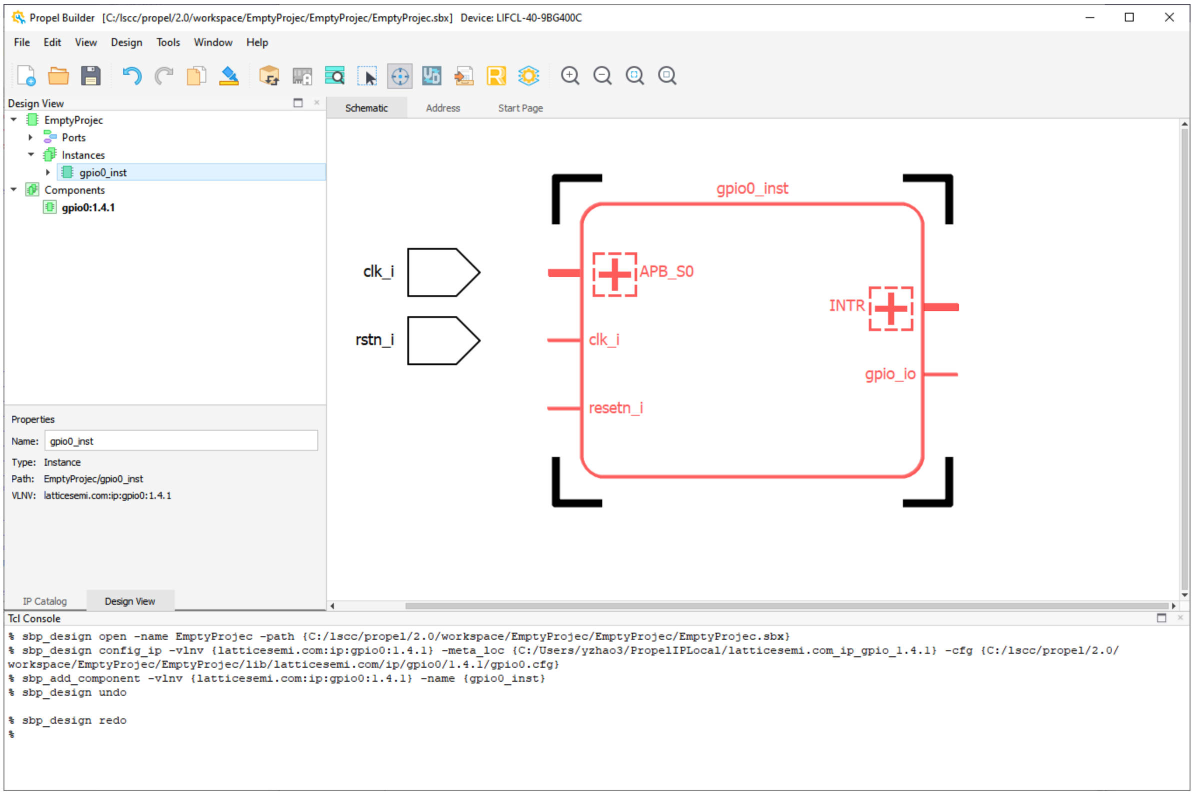


Figure 2.33. Locate Objects

- To automatically simplify the layout:
  - Right-click anywhere in the Schematic view and choose **Relayout**.
- To duplicate a module:
  - Select the desired module and click **Clone**. Or, right-click on the module and choose **Clone**. A copy of the schematic block appears with a new instance name (Figure 2.34).

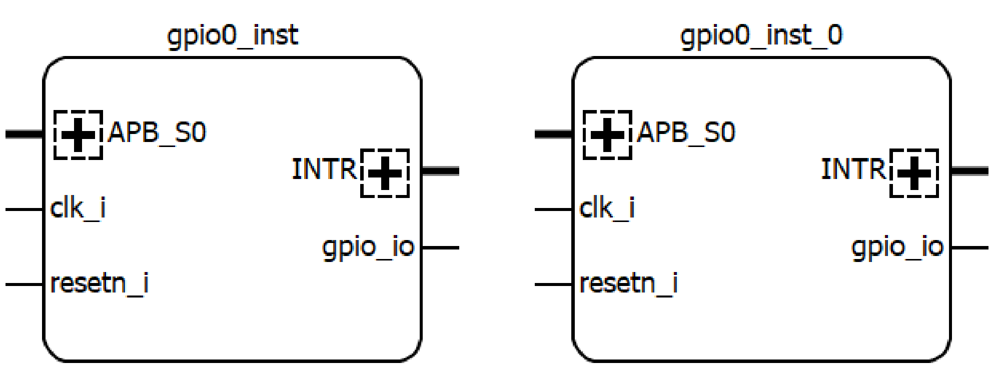


Figure 2.34. Duplicate a Module

- To restore a deleted module:

a. In the List view of the Design View (Figure 2.35), go to the Components folder. The deleted module can still be found in the List view in plain text. Those not deleted are shown in bold-faced text.

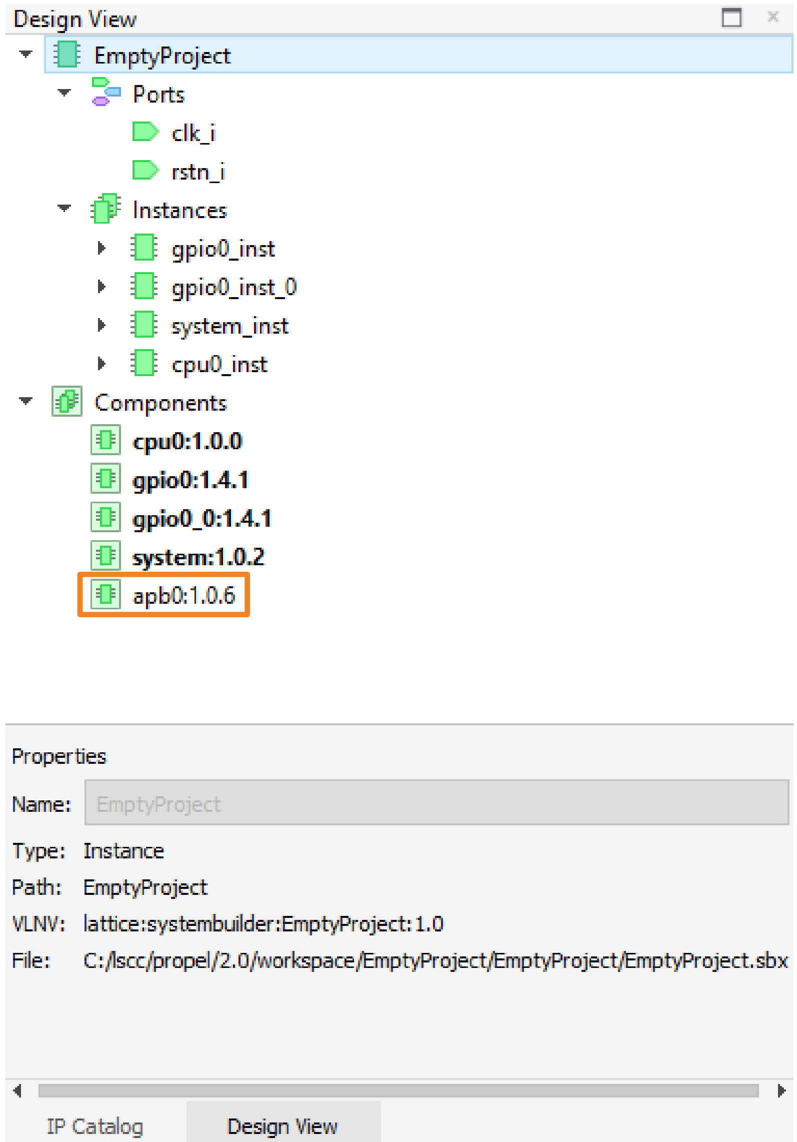


Figure 2.35. Design View

b. Right-click on the desired component and choose **Instantiate**. The Define Instance dialog box (Figure 2.36) open.

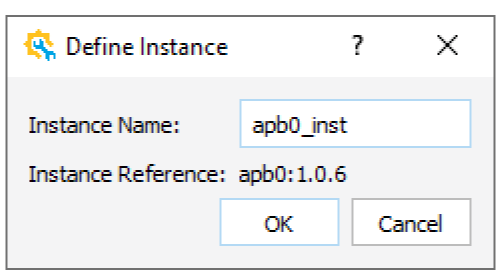


Figure 2.36. Define Instance Dialog Box

c. Enter a name for the new instance.

d. Click **OK**. The module appears in the Schematic view and the List view of Design Info, and the component name is bold-faced.

- To reconfigure a module:

a. Double-click the module, or right-click the module. Choose **Reconfig**. The Module/IP Block wizard (Figure 2.37) opens.

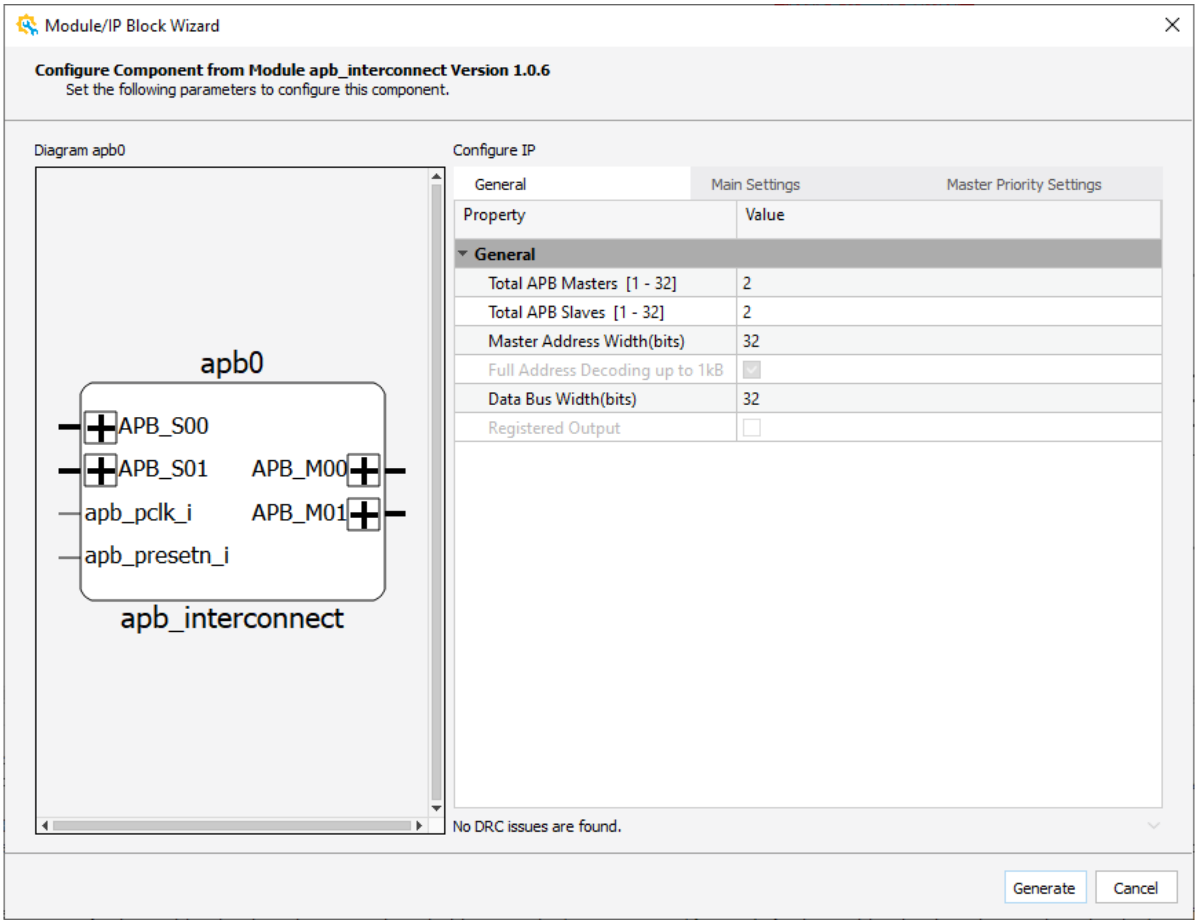


Figure 2.37. Module/IP Block Wizard – Configure Component

b. Configure component (including General property, Main Settings and Mater Priority Settings) at Configure IP table in the Module/IP Block Wizard. Click **Generate** to generate the module as usual. The schematic block for the module changes to match the new configuration.

Note: For an SoC design, if the init file (mem file) of System Memory module instance is updated in a C project, the initialization section of the system memory module instance should be re-configured. Or, use the ECO flow to update the information. Refer to the Diamond online help for more on how to use an ECO flow.

- To resize module blocks:

a. Select the desired module block. The block is highlighted in red with black corners (Figure 2.38).

Figure 2.38. Select Module

b. Click and drag one of the corners to change the size and shape of the block.

c. Release the mouse button. All the other objects move to make room for this block.

Note: Right-click the module block and choose **Unresize Instance** to restore the size of the module block.

- There are a variety of methods to zoom within the Schematic view, including toolbar commands and dragging in the Schematic view.

The following commands are available from the Propel Builder Toolbar.

- Zoom In (Ctrl++) — enlarges the view of the entire layout.
- Zoom Out (Ctrl+-) — reduces the view of the entire layout.
- Zoom Fit — reduces or enlarges the entire layout so that it fits inside the window.
- Zoom To — enlarges the size of one or more selected objects on the layout and fills the window with selection.

Note: The mouse wheel provides a finer zoom control by rolling the mouse wheel forward to zoom in and backward to zoom out while pressing the Ctrl key.

To zoom by holding the mouse button and dragging.

- To zoom to fit the window, drag up and to the left. The image adjusts to fill the window.
- To zoom out, drag up and to the right. The dragging distance determines the amount of zoom. The image is reduced and centered in the window.
- To zoom in, drag down and to the left. The dragging distance determines the amount of zoom. The image is enlarged and centered in the window.
- To zoom in in a specific area, start at the upper-left corner of the area and drag to the lower-right corner of the area. The area that dragging across is adjusted to fill the window.

Note: Make sure that the **Area_select** icon is not selected in the Toolbar. If you need to use Area_select and zoom by dragging frequently, choose **Design** > **Options** from Propel Builder menu bar. The Options Dialog opens (Figure 2.39). Select **Use right mouse button for zooming actions** and click **OK**. Then you can select an instance using the left mouse button, zoom in or zoom out using the right mouse button, and the **Area_select** icon disappears from the Propel Builder Toolbar.

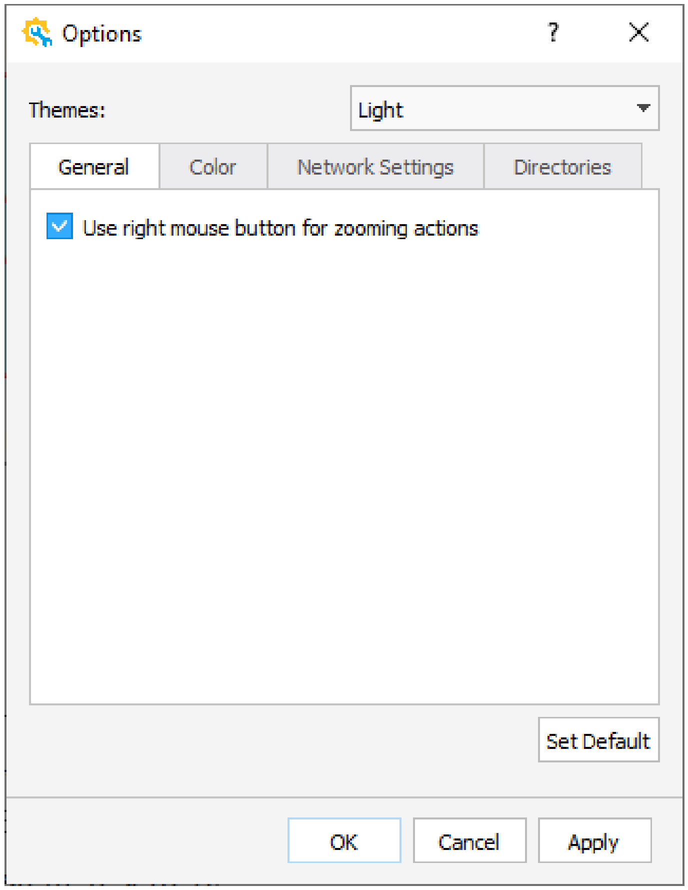


Figure 2.39. Options Dialog

- To move the schematic image within the Schematic view by panning and scrolling.
  - To pan the image: Hold down the Ctrl key and the left mouse button while dragging the image.
  - To scroll vertically: Rotate the mouse wheel. Or, click in the vertical scroll bar.
  - To scroll horizontally: Hold down the Shift key and rotate the mouse wheel. Or, click in the horizontal scroll bar.
- To show the connectivity of a module: right-click the module and choose **Show connectivity**. All nets connected to the module, all pins and ports connected to the nets are highlighted (Figure 2.40).

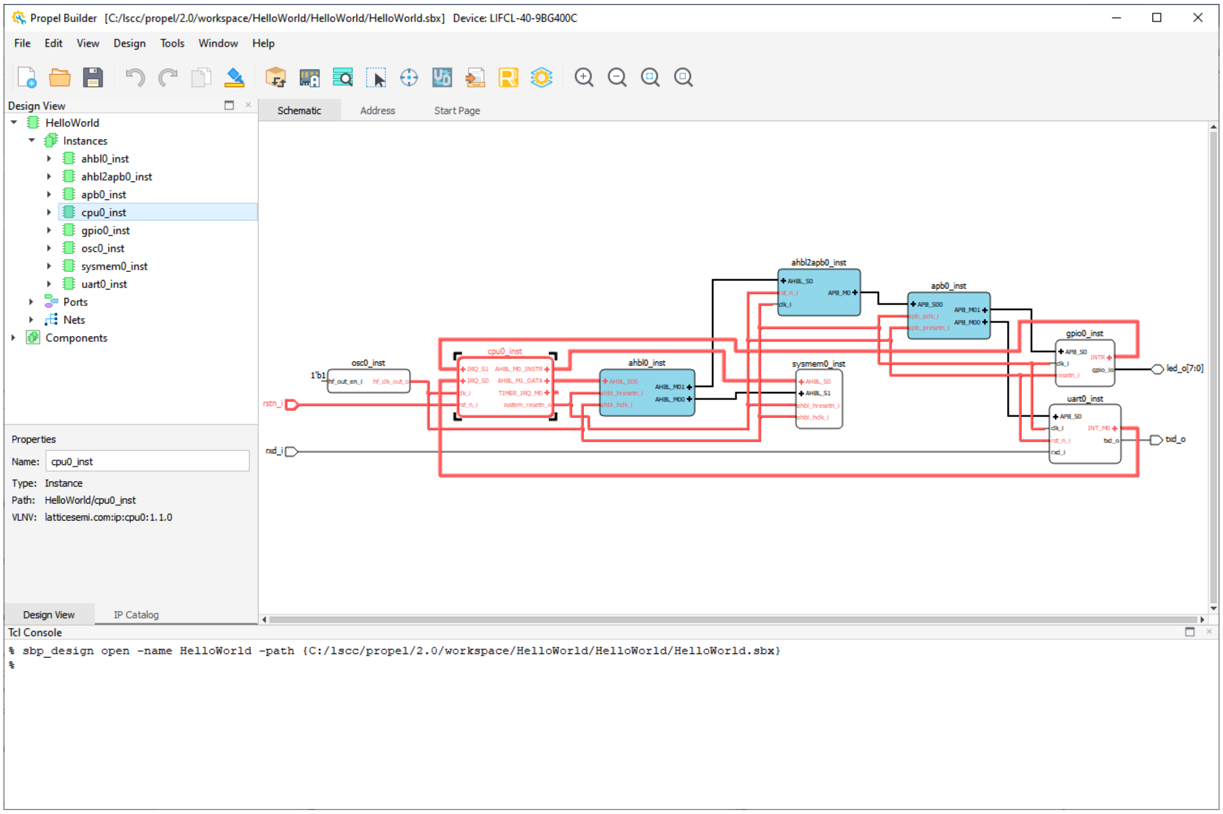


Figure 2.40. Show Connectivity of the Module

- To highlight an object: select an object and click **Highlight** , or, right-click the object and choose **Highlight** . The object is highlighted in blue (Figure 2.41). Click **Highlight**  again. You can remove highlighting.

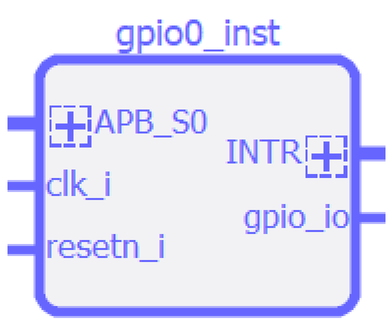


Figure 2.41. Highlight an Object

- To change the name of an object:
  a. Select the object from the List view of Design View. Information of the selected object is shown in the Properties area (Figure 2.42).

Figure 2.42. Object Properties

b. Change the name and click **Enter.** The name changes in the Schematic view and List view of Design Info.

- To print a schematic:

a. Choose **File** > **Print Preview**. The Print Preview window (Figure 2.43) opens.

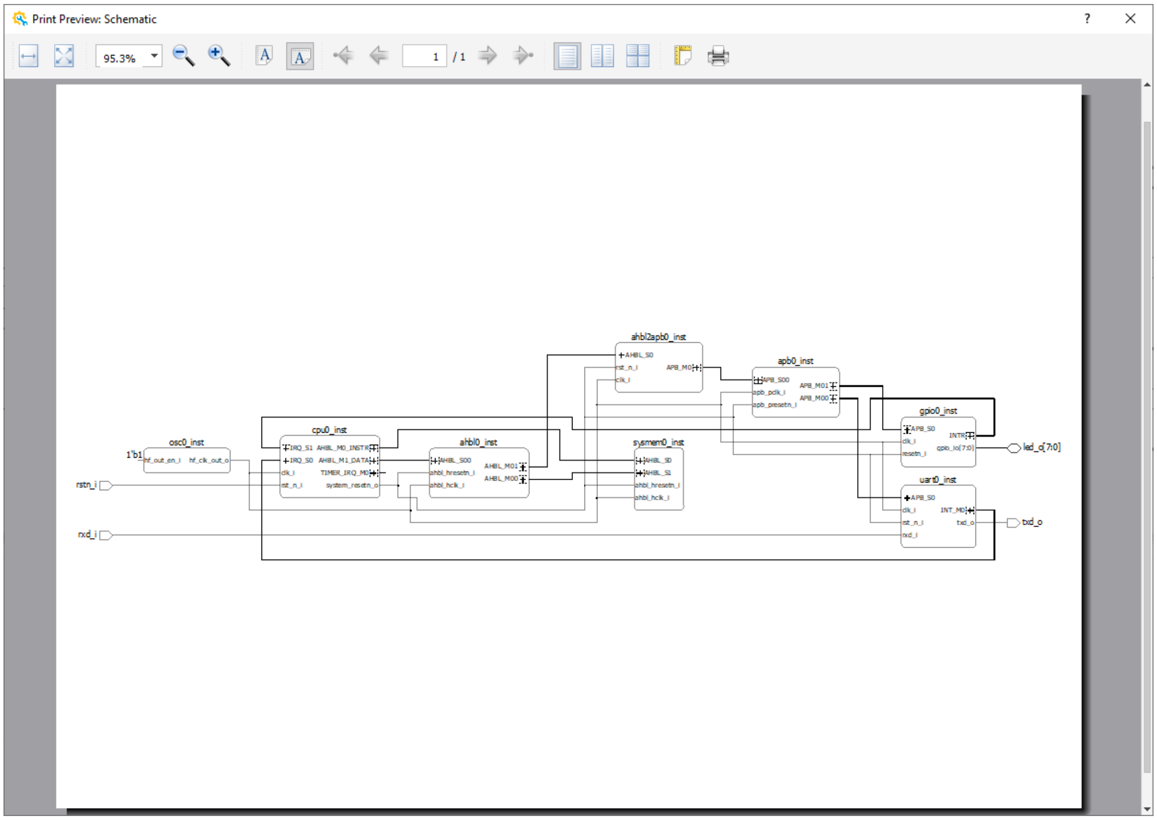


Figure 2.43. Print Preview

b. Expand the Print Preview window to the desired size.

c. Click the **Page Setup** button  and adjust the paper size and margins, if necessary.

d. Click the **Print** button .

e. Adjust the printer settings, if necessary. Click **Print**.

2.2.8. Connecting Modules

You can connect the pins of modules to other modules or to top-level ports by dragging a line between them or by selecting connection points, or by assigning a constant value to an input pin or bus. Propel Builder does not allow obvious inappropriate connections, such as a connection between two output pins or mismatched buses.

- To connect modules by drawing:
  a. Move the cursor to a pin or port. The cursor changes to a pencil icon. Click and hold while dragging to another pin, port, or net. An allowed pin or port shows a green check-mark when you hover over it (Figure 2.44). An allowed net becomes bold when you hover over it (Figure 2.45).

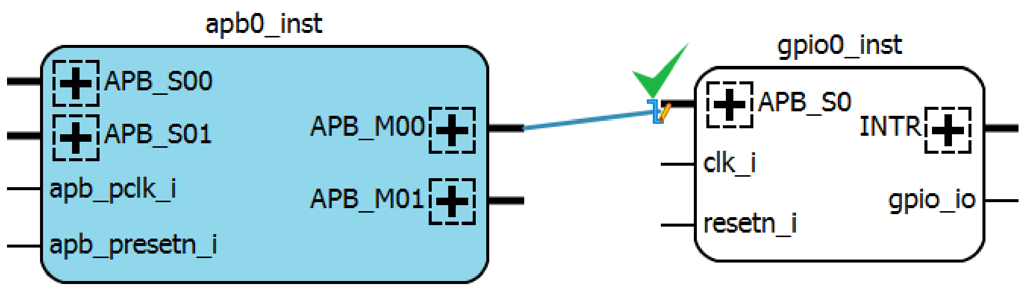


Figure 2.44. Draw a Pin or a Port

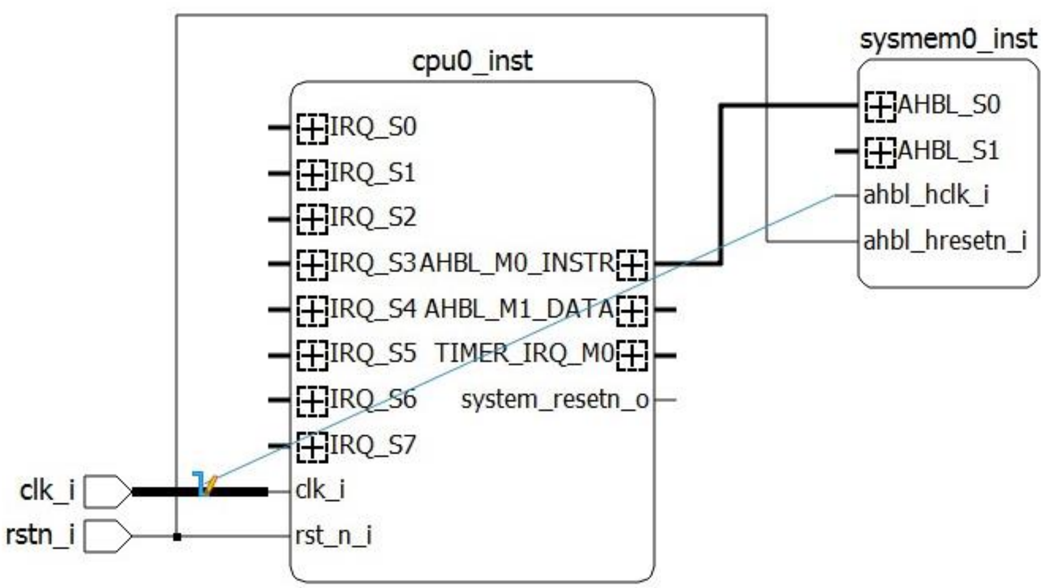


Figure 2.45. Draw Nets

  b. Click on the pin, port, or net that you want to connect to. If the connection is allowed, a line appears connecting the two objects. Propel Builder creates a path around other objects.

  c. You can connect multiple ports once. When more than one port are selected (Figure 2.46), click **connect** from the right-click menu to implement it.

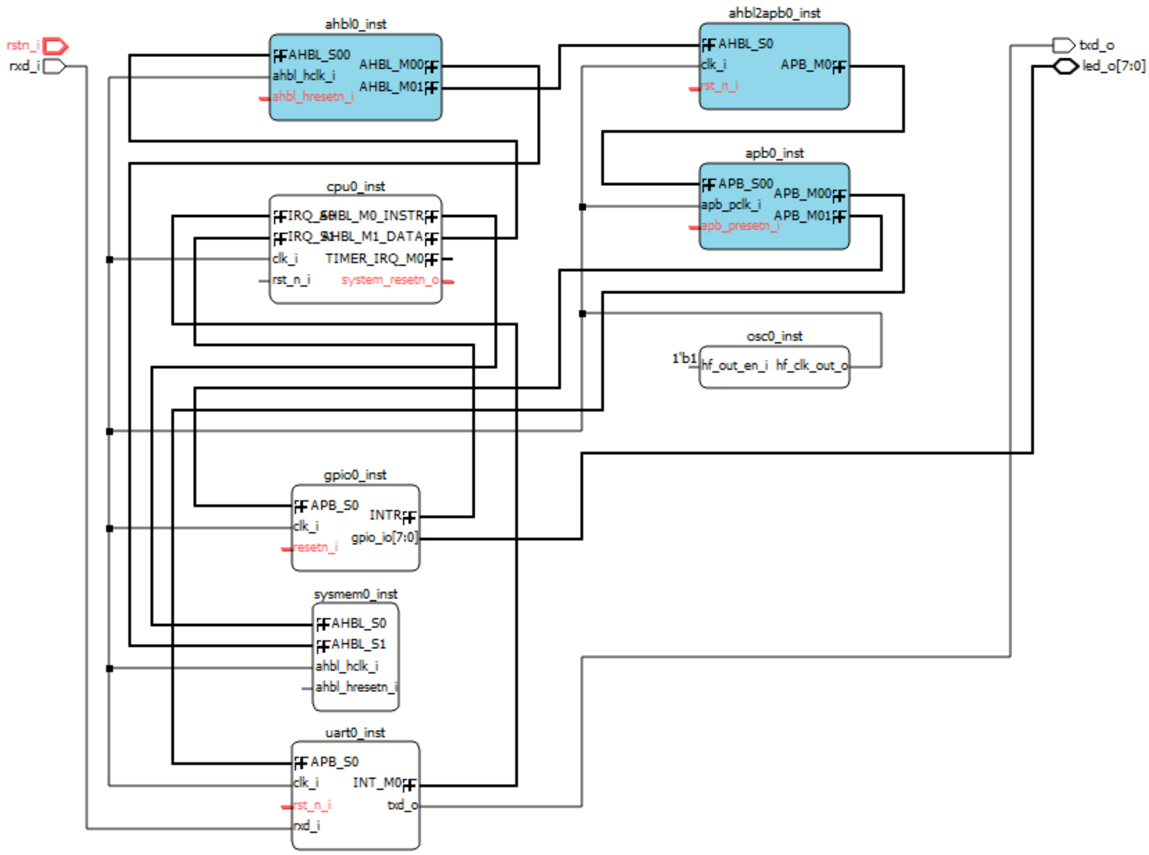


Figure 2.46. Select More than One Ports

d. When the net is complete, right-click to leave the drawing mode.

- To connect modules by selecting points:
  a. Select the pins, ports, and nets that you want to connect.
  b. Right-click one of the selected objects. Choose **Connect** .
- To assign a constant value to an input pin:
  a. Select the desired pin and right-click the pin. Choose **Assign Constant Value** (Figure 2.47). A small dialog box appears (Figure 2.48).

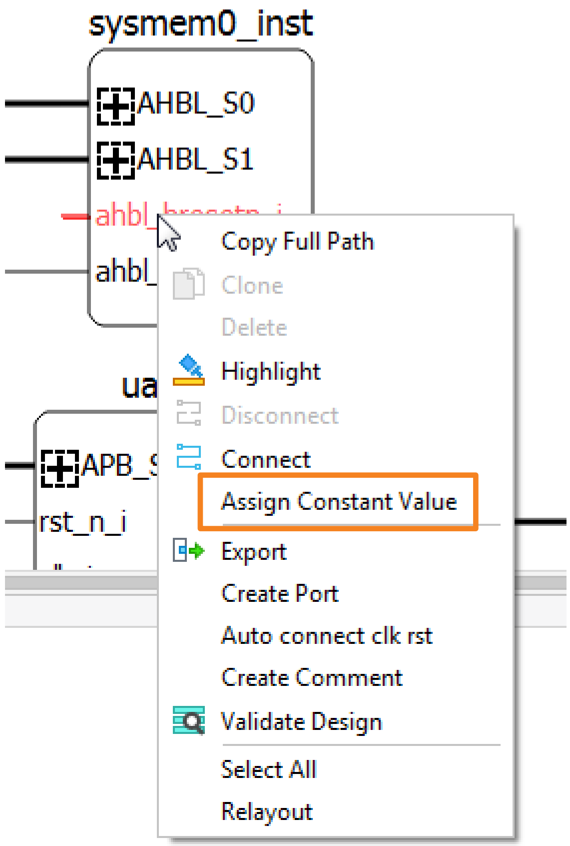


Figure 2.47. Action Menu of Right-clicking an Input Pin

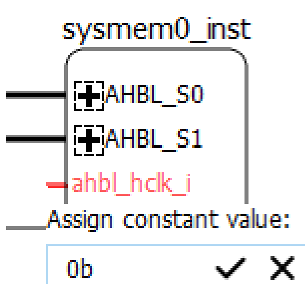


Figure 2.48. Dialog Box of Assigning Constant Value to an Input Pin

b. Enter the desired value. To erase the value and start over, click X. For all buses (more than one pin), the format is hexadecimal. Make sure the value fits the number of pins in the bus. For example, a 3-pin bus can accept 0-7, but not 8.

- Usually, disconnecting modules just means deleting a net. If the net has multiple branches, just delete one branch, leaving the rest of the nets intact.
  - To disconnect one branch of a net, right-click the pin of that branch and choose **Disconnect**. The branch going to that pin disappears.
  - To disconnect a whole net, right-click the net and choose **Delete**. The whole net is deleted.

2.2.9. Creating Top-Level Ports

With multiple modules in a Propel Builder project, top-level ports need to be created for a complete Propel Builder module. You can create a top-level port either manually or automatically.

For input ports that connect to more than one pin, such as for clock and reset signals, the manual way is more convenient than the automatic way. If the port names automatically-generated need to be changed, the manual way is better.

For other pins, the automatic way is usually preferred. The automatic way enables you to create the appropriate port type and connect net simultaneously. The automatic way can also create multiple ports at the same time.

The automatic way of creating ports is the most effective. If all the modules are selected, Propel Builder can automatically create ports for all the remaining unconnected pins for selected modules in one step.

- To create a port for a pin or bus manually:
  a. Right-click in the Schematic view, and choose **Create Port**. The Create Port dialog box pops up (Figure 2.49).

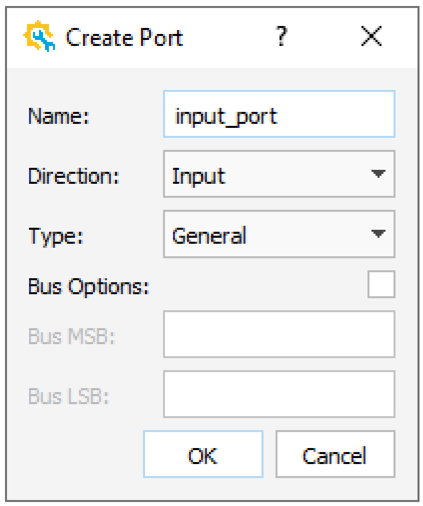


Figure 2.49. Create Port Dialog Box

  b. Enter a name for the port in the **Name** field.
  c. Choose a direction for the port, such as Input in the **Direction** field.
  d. Choose the port type from the **Type** field. The default type is General.
  e. (Optional) Select the **Bus Options**. Enter the number for the most significant bit (MSB) and the least significant bit (LSB). These two options define the bus width.
  f. Click **OK**. The port appears. Figure 2.50 shows an input port, port name of which is on the left. Figure 2.51 shows an output port and an inout port, port names of which are on the right.

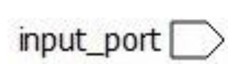


Figure 2.50. Input Port

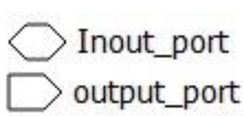


Figure 2.51. Output Port and Inout Port

  g. Connect the port to the module pins. Refer to the Connecting Modules section for more details.

- To create ports automatically:
    a. Select the pins that are to be connected to the top-level ports. All the pins in a module can be selected by clicking its block. Any pins that are already connected to a net or a constant value are skipped.

    b. Right-click on one of the selected pins, interfaces or modules, and choose **Export**. The selected pins are extended by lines to new top-level port symbols. The names of the ports and nets are added to the List view. Zoom out or scroll the image in the Schematic view to see the new ports.

    c. Port or net names can be changed, if needed.

2.2.10. Adjusting Address Spaces

The Address view shows the base address, size of the address segment, and the end address for each leaf memory-mapped slave connection in the Propel Builder project. Propel Builder can automatically assign address values, while the base address value can be changed manually. The ranges are set when the modules are configured. The end addresses are calculated.

The Lock option on each address space prevents Auto Assign from changing the base address value. The Lock option is selected automatically when you manually change the address value. To reset the address space, clear the Lock option before clicking the Auto Assign icon.

Note: There is no Lock option on LocalMemory. The value of the base address can always be changed, while Auto Assign does not reset it to its original value.

1. In the Propel Builder main window, click the Address tab. The Address view (Figure 2.52) shows.

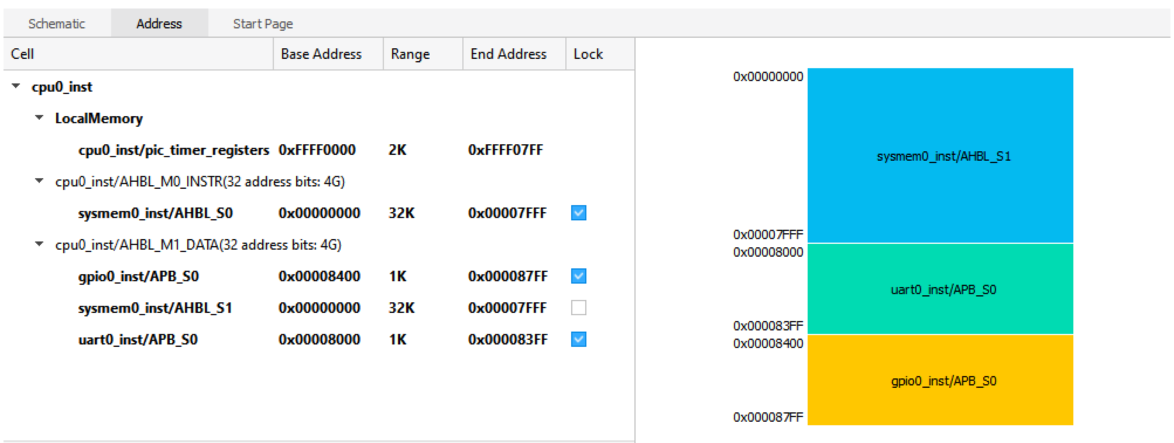


Figure 2.52. Address View

2. (Optional) Set or clear the Lock options as desired in Address view.

3. In the Propel Builder Toolbar, click **Auto Assign** . Default address values appear in Address view.

4. (Optional) Double-click the base address in Address view (Figure 2.53). Type the new value and press **Enter**. Values must align with 1K boundaries, such as 0x00000400, 0x00000800, or 0x00000C00. The end address value changes based on the new value. The Lock option is selected automatically at this time.

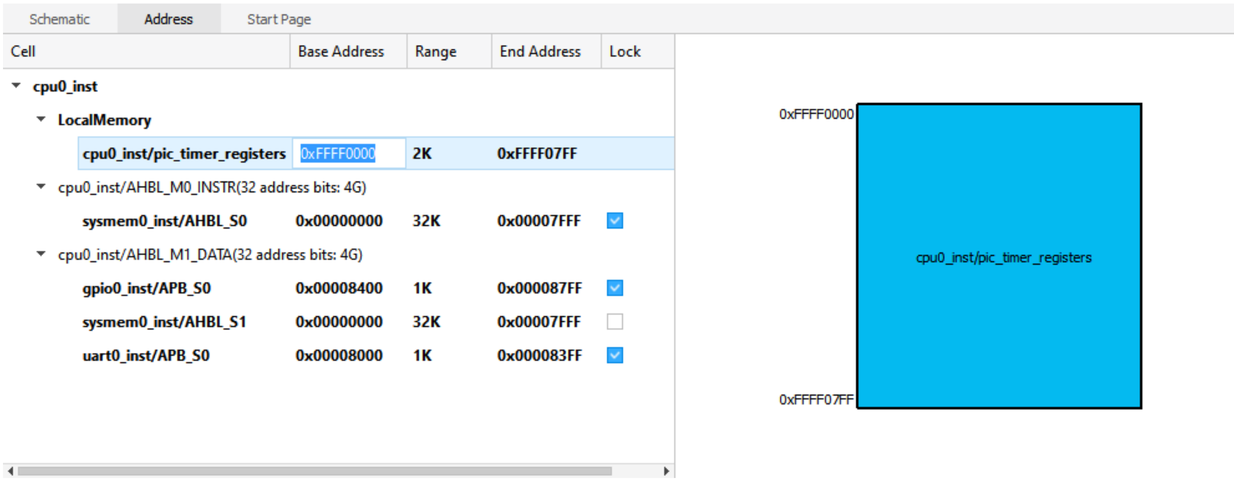


Figure 2.53. Edit Base Address

Note: If there is a conflict of a related address space, it is shown in red in the graphic.

2.2.11. Validating the Design

The design rule check (DRC) can be run at any time. This checks whether or not there is any illegal connection or overlapping address space.

To perform the design rule check:

1. From the toolbar of the Propel Builder main window, click the **Validate Design** icon . The DRC results appear in the Tcl Console.

2.2.12. Generating the RTL file

The final step for creating a Propel Builder module is to generate a .sbx file, which defines a Propel Builder project, an RTL file, and the instantiation templates. The RTL file includes the Verilog code for the module. The instantiation templates have Verilog and VHDL code to help instantiate the Propel Builder module in a design.

- From the toolbar of the Propel Builder main window, click the **Generate** icon . This step also saves the Propel Builder design and runs the design rules check (DRC).
- From the toolbar of the Propel Builder main window, click the **Generate Memory Report** icon . The SoC design memory information is generated. The memory report folder (mem_report) is generated. All design memory information is shown in detail in the memory report files. The Memory Report files are in HTML format (Figure 2.54).

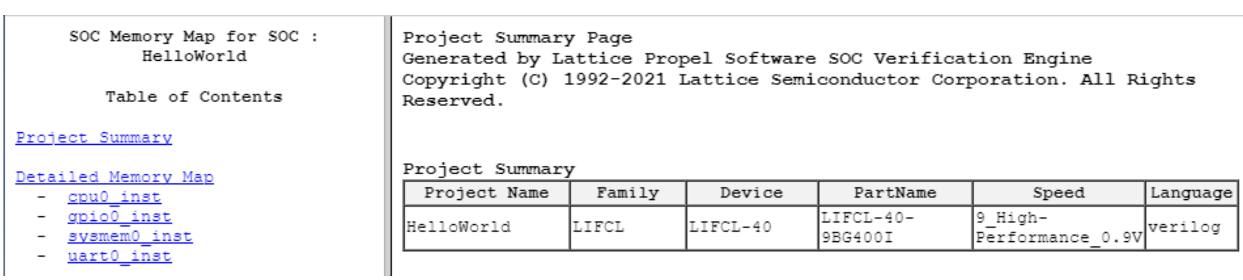

SOC Memory Map for SOC : HelloWorld

Table of Contents

Project Summary

Detailed Memory Map
- cpu0_inst
- gpio0_inst
- sysmem0_inst
- uart0_inst

Project Summary Page
Generated by Lattice Propel Software SOC Verification Engine
Copyright (C) 1992-2021 Lattice Semiconductor Corporation. All Rights Reserved.

Project Summary

Project Name	Family	Device	PartName	Speed	Language
HelloWorld	LIFCL	LIFCL-40	LIFCL-40-9BG400I	9_High-Performance_0.9V	verilog

Figure 2.54. Memory Report

2.2.13. Opening Project in Diamond or Radiant

In a complete SoC project, a Diamond or Radiant project can be created including a Propel Builder design, and after that the SoC project can be opened in Diamond or Radiant software. According to device in the SoC project, Propel Builder can launch Diamond or Radiant software accordingly. If the device is LCMXO3D, MachXO3L, MachXO3LF, MachXO2 LFMNX, the **Diamond** icon is shown in the Builder GUI Toolbar. If the device is LIFCL or LFD2NX, the **Radiant** icon is shown in the Builder GUI Toolbar. For a template SoC project, you can launch **Diamond** or **Radiant** software directly after template SoC project is created.

- To launch Diamond software:
  a. Click the **Diamond** icon from the Propel Builder toolbar.
  b. Lattice Diamond is launched with a Diamond project generated for SoC at background (Figure 2.55).

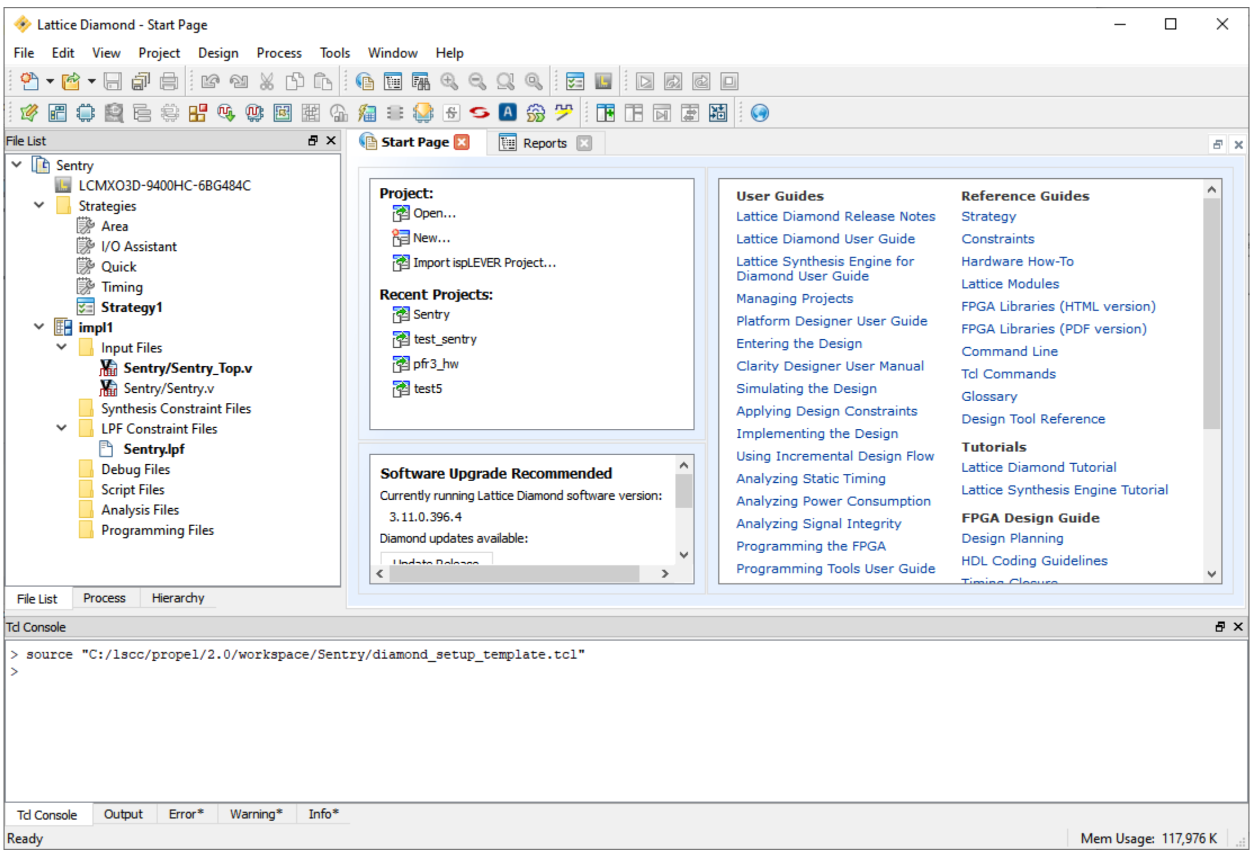


Figure 2.55. Diamond Project

  c. (Optional) From the **File List** view of Diamond:
     - 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.
     - 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.
  d. (Optional) Modify LPF constraint file (<*proj_name*>.lpf) to match the SoC design, if you have modified the SoC design. This step is a must to the SoC design that is created from Empty Project template.
  e. Switch to **Process** view of the Diamond project (Figure 2.56). Make sure at least one programming file (IBIS Model, Verilog Simulation File, VHDL Simulation File, Bitstream File, or JEDEC file) is selected in the **Export Files** section. Available programming files can be different upon specific device included.

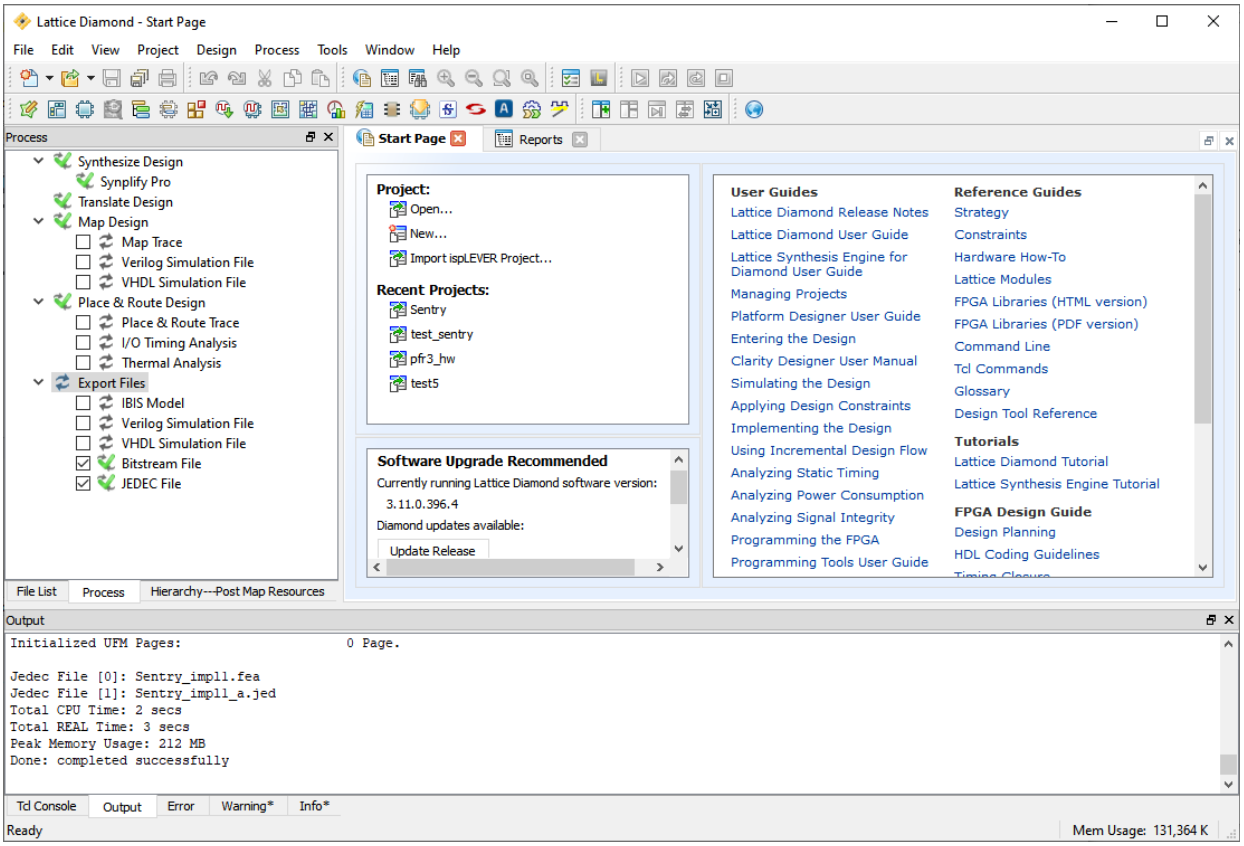


Figure 2.56. Generate Programming Files

f. Right-click **Export Files**, and choose **Run** . The programming file is generated. The programming file can be used in the Diamond Programmer.

Note: Refer to the Diamond online help for more details of the Diamond project.

- To launch Radiant software:

a. Click the **Radiant** icon  from the Propel Builder toolbar.

b. Lattice Radiant is launched with a Radiant project generated for SoC at background (Figure 2.57).

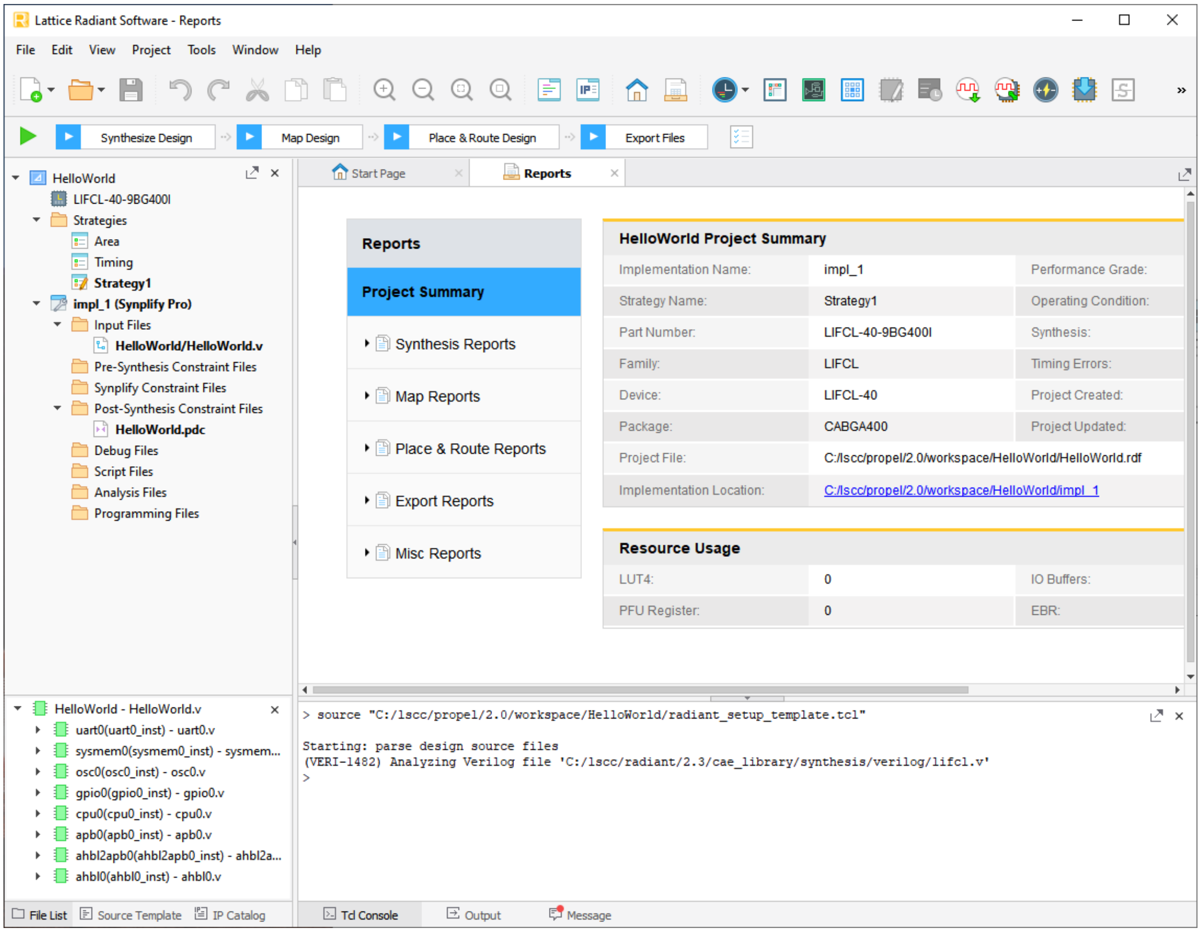


Figure 2.57. Radiant Project

c. (Optional) From the **File List** view of Radiant:

- 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.

d. (Optional) Modify pdc constraint file (<*proj_name*>.pdc) to match the SoC design, if you have modified the SoC design. This step is a must to the SoC design that is created from Empty Project template.

e. Click  **Run all**. The Programming file is generated. It can be used in the Radiant Programmer.

Note: Refer to the Radiant online help for more details of the Radiant project.

2.2.14. Launching SDK

After an SoC design is completed, Propel SDK can be launched in Propel Builder for software development. For template SoC project, you can launch SDK directly by clicking **Run Propel** icon from Propel Builder toolbar after template SoC project created.

1. Click the **Run Propel** icon  from the Propel Builder GUI Toolbar. Lattice Propel Launcher (Figure 2.58) opens.

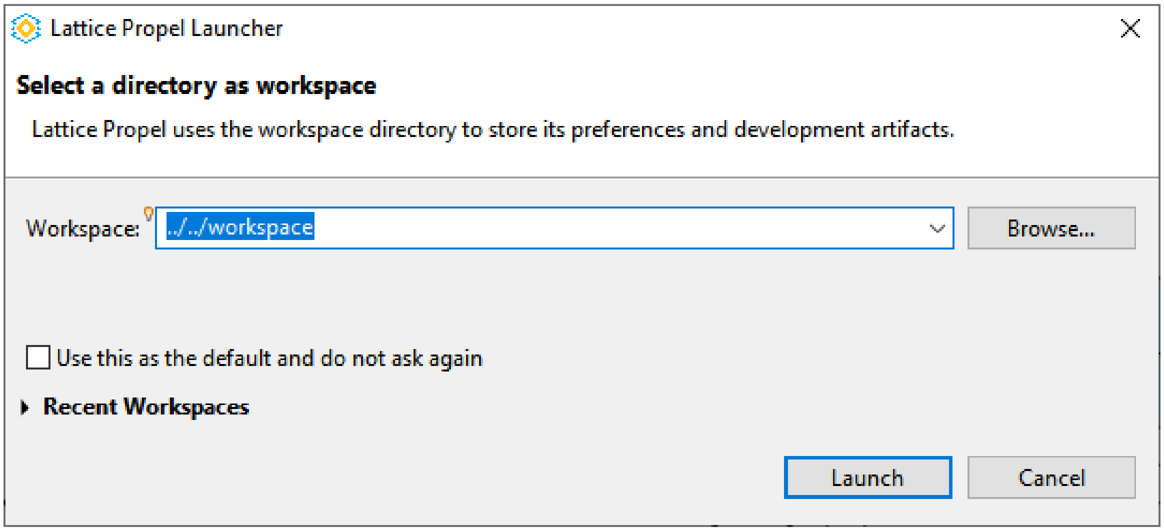


Figure 2.58. Lattice Propel Launcher Wizard

2. Click **Launch**. Propel SDK GUI opens and the C Project wizard (Figure 2.59) pops up for loading the system and the board support package (BSP) to create a C/C++ project.

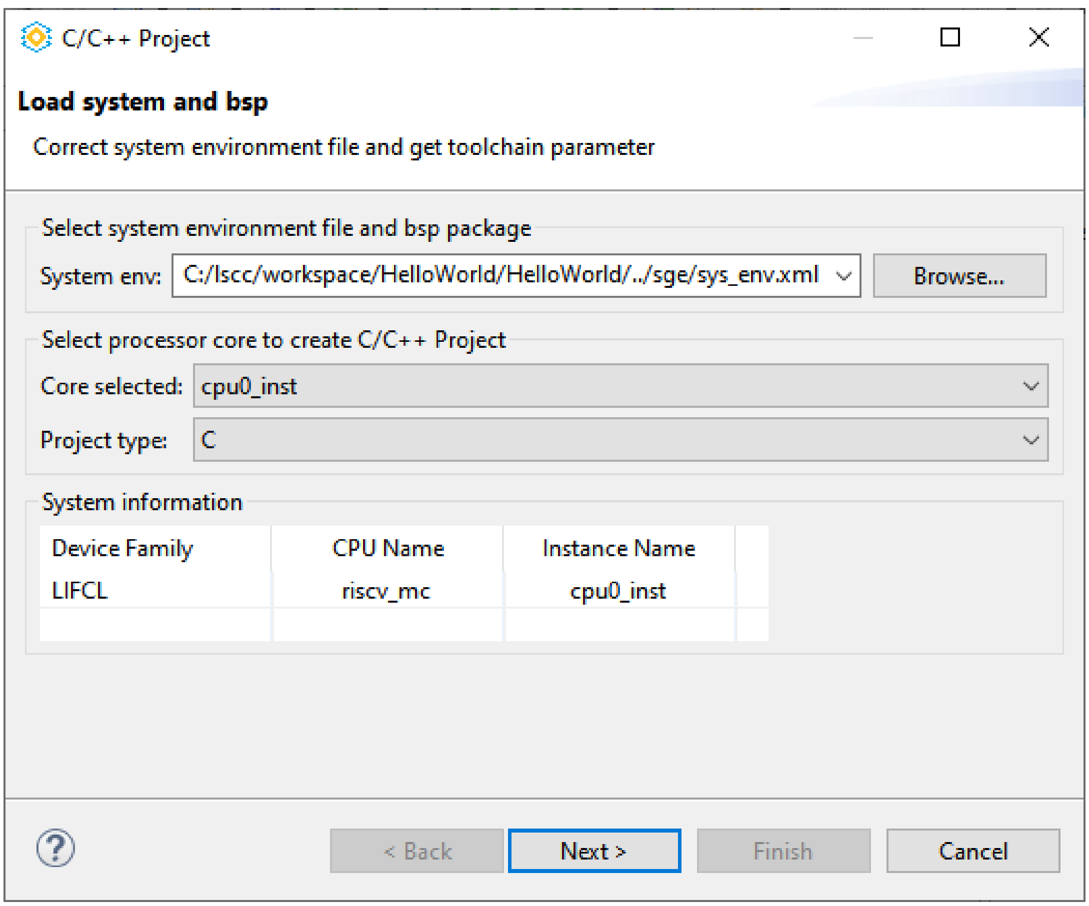


Figure 2.59. Propel SDK GUI and C/C++ Project Wizard

3. Click **Next**. The C/C++ project dialog (Figure 2.60) opens.

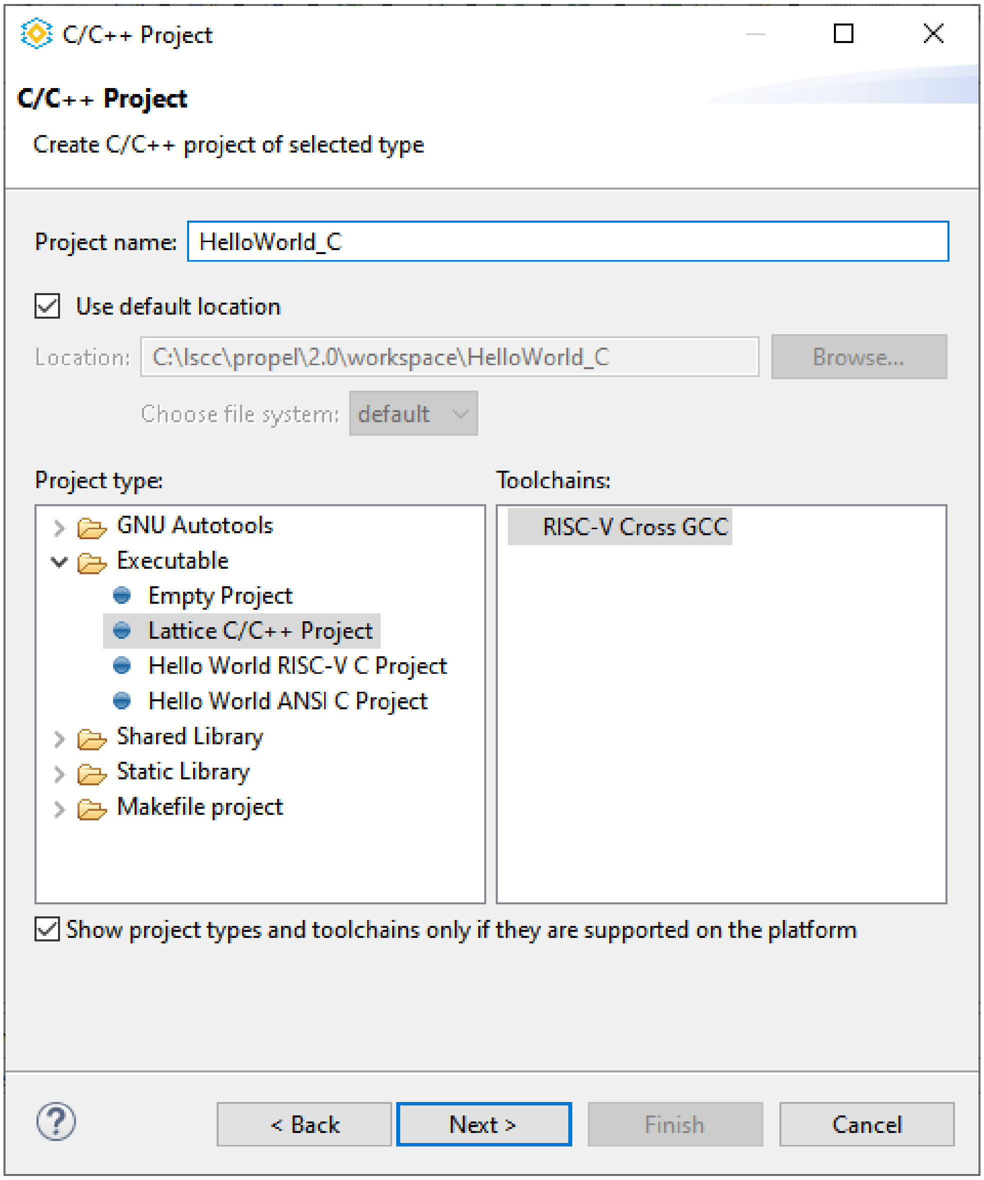


Figure 2.60. Create C/C++ Project

4. Enter a project name in the **Project Name** field, such as HelloWorld_C. Click **Next**. The Lattice toolchain setting dialog opens (Figure 2.61).

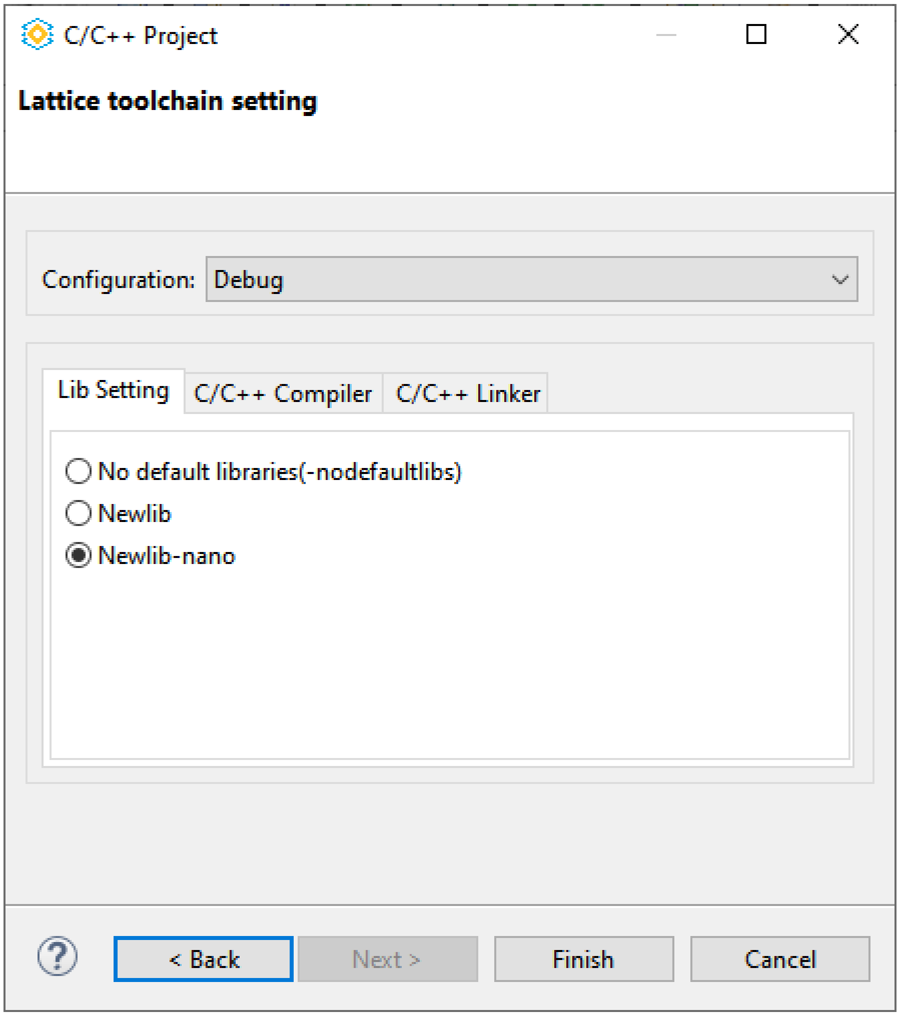


Figure 2.61. Lattice Toolchain Setting Dialog

5. Click **Finish**. The C/C++ project is created and is displayed using the C/C++ perspective. A perspective is a collection of tool views for a particular purpose. The C/C++ perspective is for creating C/C++ programs.

Note: Refer to Lattice Propel 2.0 SDK User Guide (FPGA-UG-02127) for more details on how to create a C/C++ project and develop the C/C++ project in Propel SDK.

2.3. Verification Project Design Flow

2.3.1. Creating a Verification Project

1. Choose **File** > **New Design** from the Lattice Propel Builder Menu Bar. The Create System Design wizard opens (Figure 2.62).

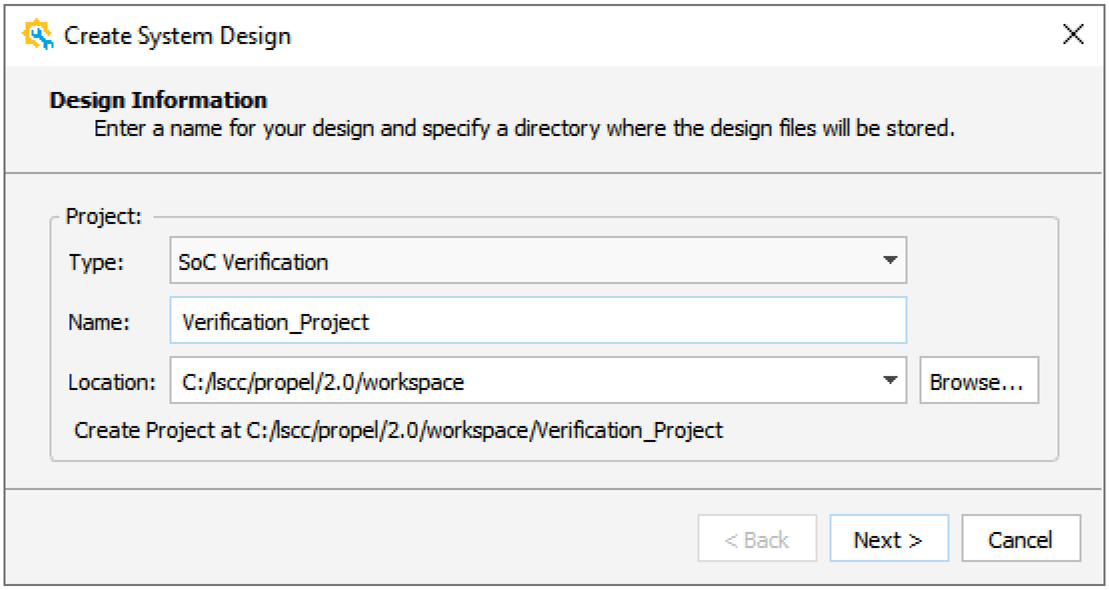


Figure 2.62. Create System Design – Design Information Wizard

2. Choose SoC Verification from the **Type** field.
3. Enter a project name in the **Name** field, such as Verification_Project.
4. (Optional) Use the "**Browse…**" option to change the project location in the **Location** field, if needed.
5. Click **Next**. The Propel Project Configure wizard is shown as Figure 2.63.

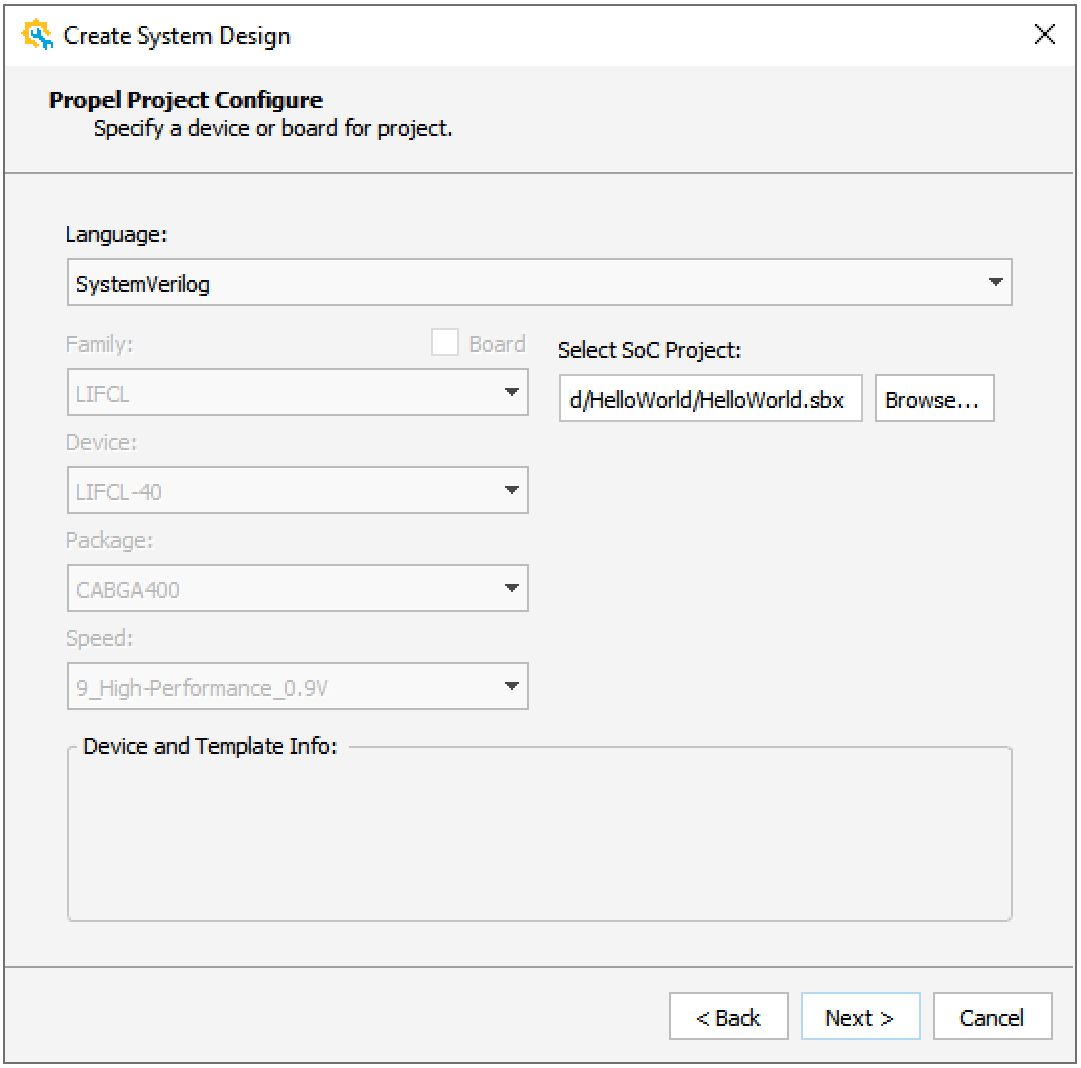


Figure 2.63. Create System Design – Propel Project Configure Wizard

6. (Optional) The default Hardware Description Language (HDL) is displayed in the Language area. Use the drop-down menu to change the default language.
7. Select an existing SBX design by using **Browse…** to choose an SBX file, such as HelloWorld.
8. Click **Next**. The Project Information wizard opens.
9. Click **Finish**. A dut_inst of SoC project can be seen in the Schematic view (Figure 2.64).

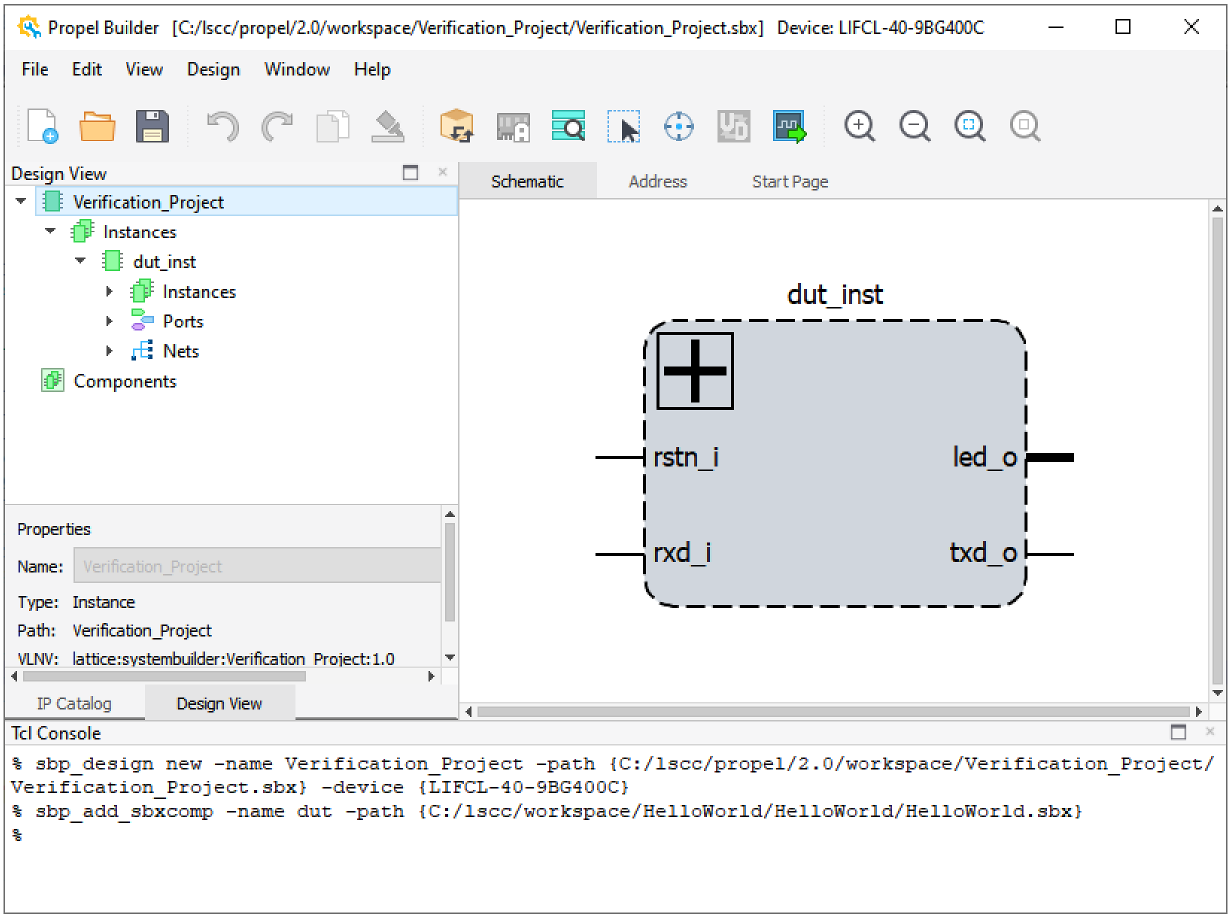


Figure 2.64. Verification Project

Note: The default instance name of an imported Design Under Test (DUT) is dut_inst. By default, its boundary is shown with dotted line, and the filled-in color is gray. If there is any change in the SOC design, the dut_inst DUT block can be updated accordingly by double-clicking this dut_inst DUT block, or by right-clicking this dut_inst DUT block and choosing Reconfig.

10. Click the plus sign of this dut_inst DUT block. You can see the whole SoC Design (Figure 2.65). Click the negative sign to close the expanded bus.

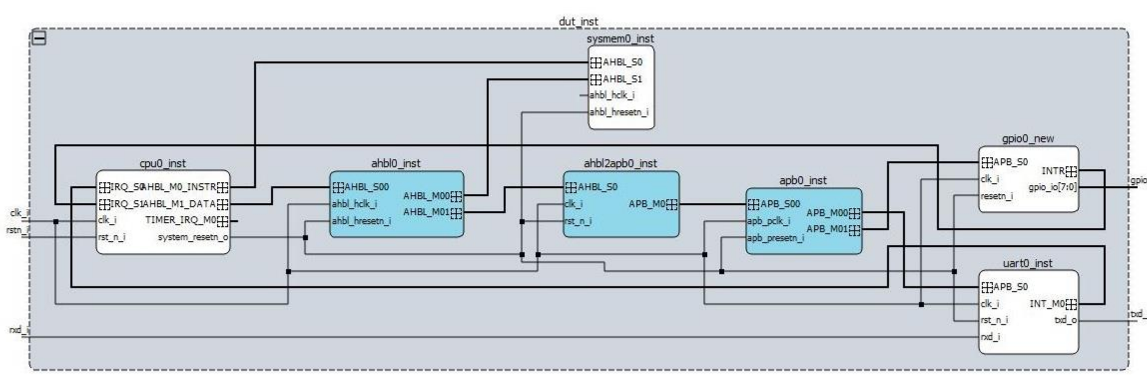


Figure 2.65. Whole SoC Design

2.3.2. Switching SoC Project to Verification Project

Propel Builder supports switching SoC project to verification project manually after creating an SoC project

1. Create an SoC Project, such as HelloWorld project. Refer to the Creating Template SoC Project section for more details.

Note: Corresponding Verification project is created automatically when creating a new empty SOC project. Hello World Templates can have pre-developed Verification projects.

2. Click **Switch Verification and Soc Design** icon  from Propel Builder GUI Toolbar. Propel Builder dialog box opens (Figure 2.66), if a project is not saved.

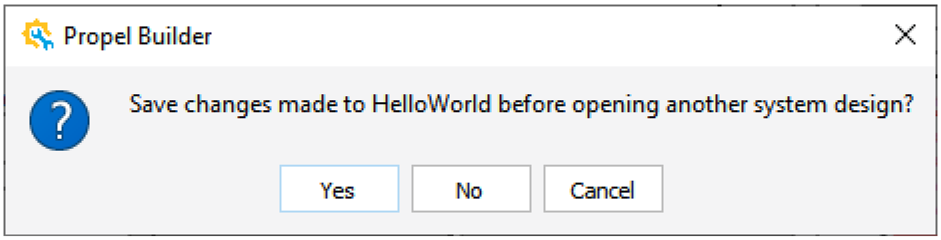


Figure 2.66. Propel Builder Dialog Box

3. Click **Yes** in the Propel Builder dialog box. The Propel Builder GUI switches to verify the project (Figure 2.67).

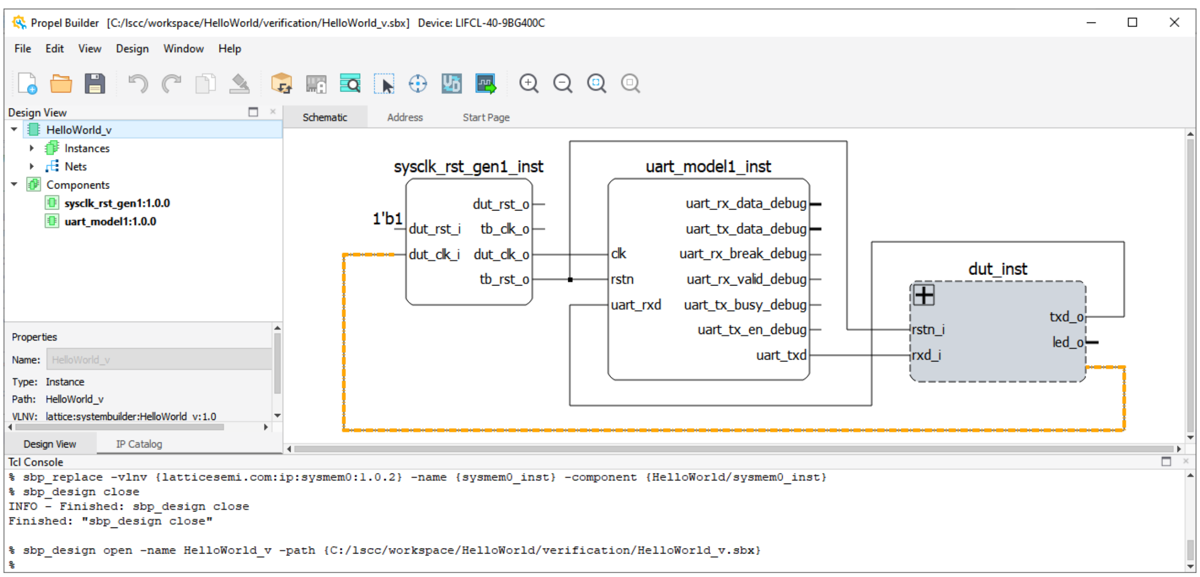


Figure 2.67. Switching to Project Verification

Note: If the SoC project needs to be re-configured, click **Switch Verification and Soc Design**  again to switch the verification project to the SoC project.

2.3.3. Opening a Verification Project

Refer to the Opening an SoC Existing Project section for procedure in detail.

2.3.4. Adding Modules, IPs and VIPs

Refer to the Adding Modules section for procedure in detail.

2.3.5. Working with the Schematic View

Refer to the previous Working with the Schematic View section for procedure in detail.

2.3.6. Connecting Modules

Refer to the previous Connecting Modules section for procedure in detail.

2.3.7. Monitoring DUT

This feature is available to an SOC Verification project only. In Propel Builder, the pin inside DUT can be connected to the input pin of a VIP. The connected pins can be found at both ends of the orange line, as shown in Figure 2.68. Only pin, pin bus are supported in the current release of Propel Builder.

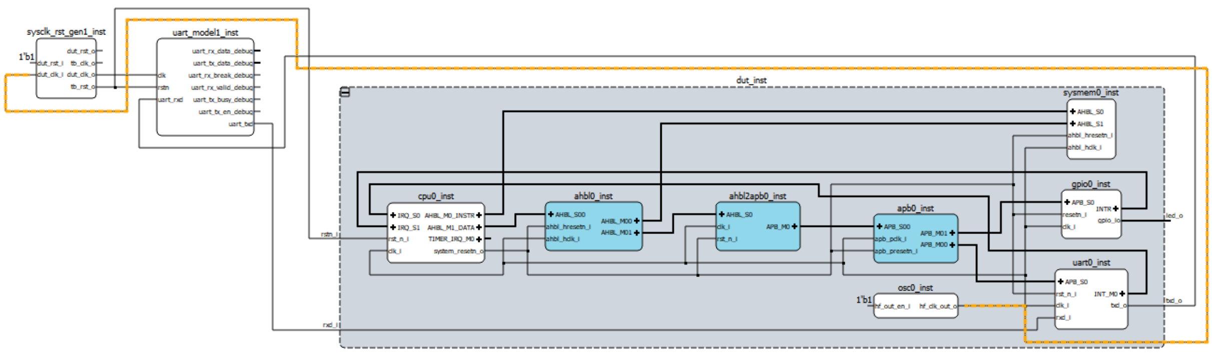


Figure 2.68. Monitoring DUT

The testbench generated for this Verification project is shown in Figure 2.69. This testbench file can be used for simulation.

```
logic sysclk_rst_gen1_inst_dut_clk_i_net;

/*------------------------Connection Block------------------------*/
/* This section cover all the connections that related to internal*/
/* signals of DUT, or interface(e.g, AHBL Master BFM).            */
/*----------------------------------------------------------------*/

assign sysclk_rst_gen1_inst_dut_clk_i_net = dut_inst.osc0_inst.hf_clk_out_o;

/*---------------------------BFM Block---------------------------*/
/* This section is reserved for user to create stimulus using BFM */
/* APIs.                                                          */
/*----------------------------------------------------------------*/


....

sysclk_rst_gen1
sysclk_rst_gen1_inst
(
   .dut_clk_i(sysclk_rst_gen1_inst_dut_clk_i_net),
   .dut_clk_o(sysclk_rst_gen1_inst_dut_clk_o_net),
   .tb_rst_o(sysclk_rst_gen1_inst_tb_rst_o_net)
);
```

Figure 2.69. Testbench of the Verification Project

2.3.8. Generating Simulation Environment

1. Click the **Generate** icon from the Propel Builder Toolbar to generate a testbench file including the scripts for the chosen simulator, file list for HDLs, and some other files. The testbench file structure is shown in Figure 2.70.

```
[sim]                        -- generated simulation environment
    [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 will be copied here
    flist.f                  -- file list for HDLs
    msim.do                  -- do script for simulator, it can be qsim for Questasim
    wave.do                  -- do script for adding signals in waveform window
    <project_name>.sv        -- top testbench, SystemVerilog based
```

Figure 2.70. Testbench File Structure

Unlike those HDLs generated in the SOC design, the testbench generation in the Verification project is just a start point for you to work with. Make sure that you generate a new testbench, if there is an existing simulation environment. A dialog box pops up prompting you to make a choice of Yes or No (Figure 2.71).

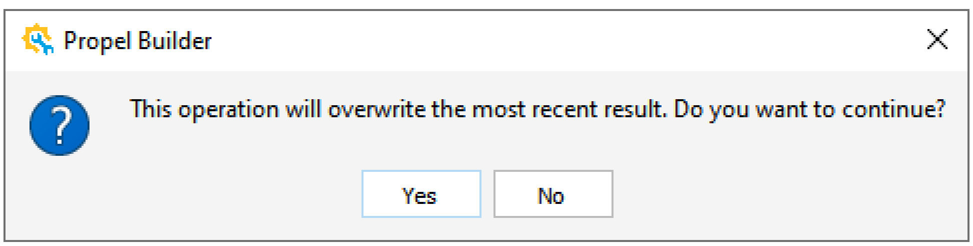


Figure 2.71. Propel Builder – A Reminding Dialog Box

2.3.9. Launching Simulation

1. Click **Launch Simulation** icon from the Propel Builder Toolbar to launch simulation.
2. The default simulation tool, ModelSim, OEM version, opens (Figure 2.72).

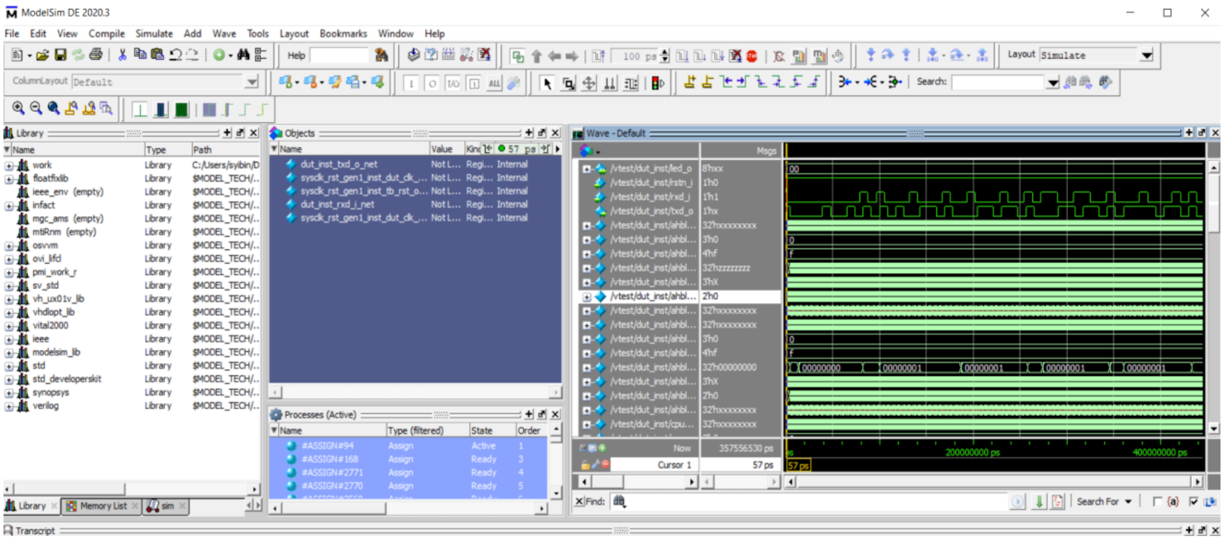


Figure 2.72. Simulation GUI

Note: Before launching simulation, you can change Simulator to Questasim by clicking **Design** > **Options** from the Builder Menu bar. Builder Options Wizard opens (Figure 2.73). Click **Directories** to set a desired Questasim Location.

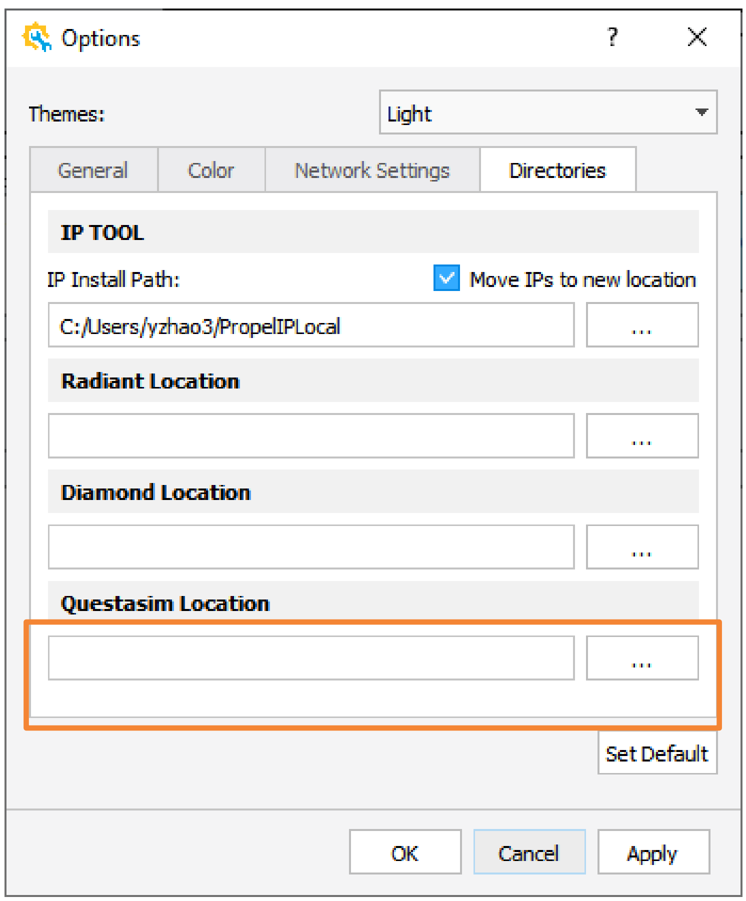


Figure 2.73. Builder Options Wizard

3. IP Packager

IP Packager is a tool for you to create an IP package easily. You can edit port, file, parameter, and memory in the IP Packager, and pack a customized IP directly.

IP Package is a collection of all required files related to an IP. All the related files are organized in different directories (Figure 3.1).

```
- [IP Package]
  - metadata.xml
  - [rtl]
  - [testbench]
  - [ldc]
    - constraint.ldc
  - [driver]
    - [ip_eval]
  - [plugin]
    - plugin.py
  - [doc]
    - introduction.html
    - EULA.txt
```

Figure 3.1. Example Directories and Files of an IP Package

- metadata.xml [mandatory]: XML metadata file which mainly describes legal usage and interface of an IP.
- rtl [mandatory]: Directory for parameterized HDL source files. HDL source files contain configurable parameters for you to configure.
- testbench [optional]: Directory for test bench files.
- ldc [optional]: Directory template constraint file. The file name should be "constraint.ldc".
- driver [optional]: Directory for driver source code files.
- plugin [optional]: Directory for Python script to implement internal logic of the soft IP. The file name of Python script should be "plugin.py".
- doc[mandatory]: Directory for documentation files. It should contain one mandatory introduction file, one mandatory license agreement file, and other optional document.

The custom IP package can be used in Propel Builder for SoC design. IP instance package (Figure 3.2) is generated when user configures IP in Propel Builder.

```
- <instance_name>
  - <instance_name>.cfg
  - <instance_name>.ipx
  - component.xml
  - design.xml
  - [rtl]
    - <instance_name>.v
    - <instance_name>_bb.v
  - [constraints]
    - <instance_name>.ldc
  - [driver]
  - [ip_eval]
  - [misc]
    - <instance_name>_tmpl.v
    - <instance_name>_tmpl.vhd
```

Figure 3.2. Example Directories and Files of an IP Instance Package

3.1. Launching IP Packager

1. Choose **Tools** > **IP Packager** from the Lattice Propel Builder Menu Bar.
2. IP Packager is launched (Figure 3.3)

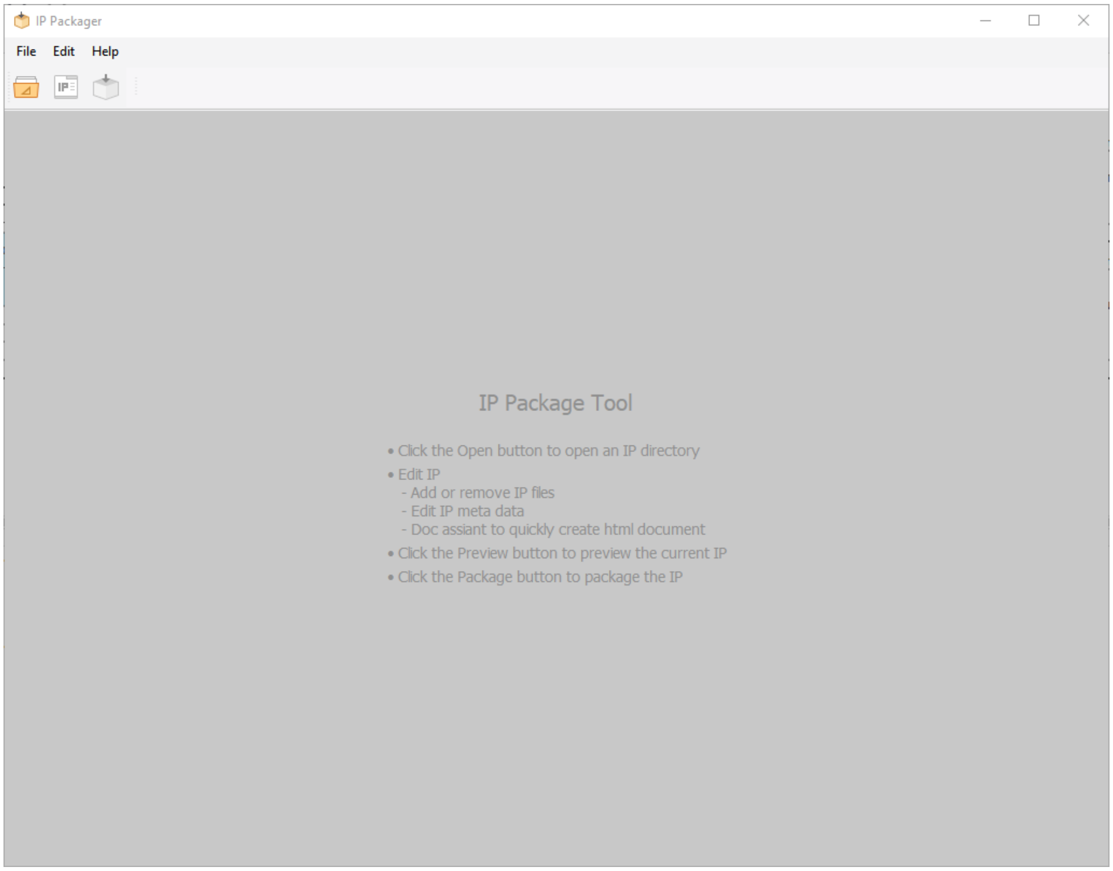


Figure 3.3. IP Packager GUI

3.2. Packing Custom IP Flow

3.2.1. Opening an IP Directory

1. Choose **File** > **Open IP Directory** from the Propel Builder Menu, or Click the **Open Design** icon from Propel Builder Toolbar. The Select Folder dialog opens (Figure 3.4).

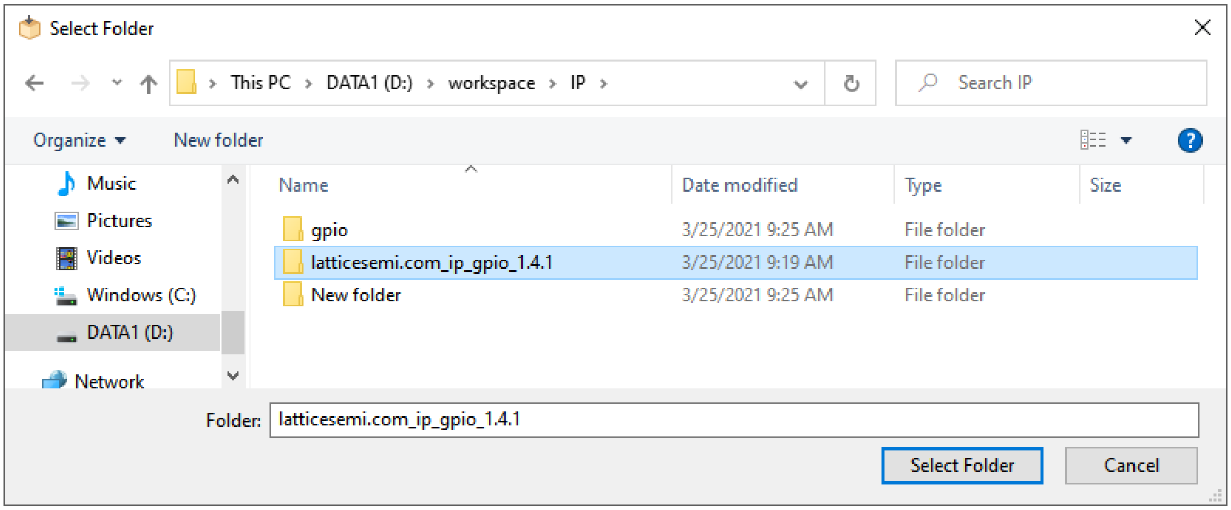


Figure 3.4. Select Folder Dialog

2. Choose a desired IP directory.
3. Click **Select Folder**. The IP Packager GUI shows the IP Package project (Figure 3.5). IP package project includes three mandatory parts (Meta Data, Design File, and Doc) and two optional parts (Test Bench and Misc) that need to be edited. Refer to the Editing IP section for more details.

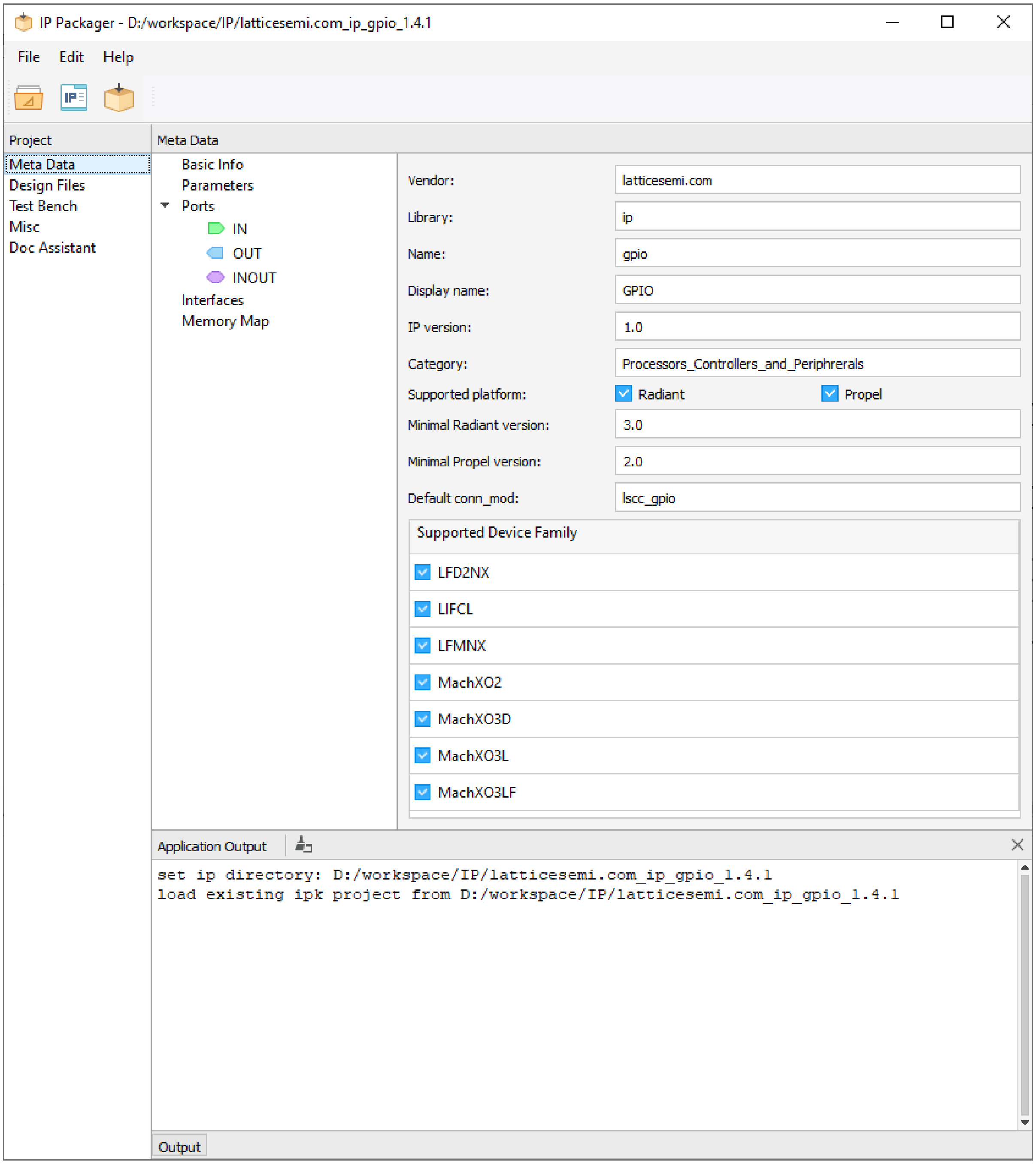


Figure 3.5. IP Packager with IP Project Details

Note: If you prepared all directories and files of an IP module in advance, the IP Packager can load the information. If you do not want to update the IP module information, you can skip the Editing IP section below.

3.2.2. Editing IP

1. From the IP Packager Project area, click **Mata Data**. The IP Packager GUI shows Meta Data information in detail (Figure 3.6).

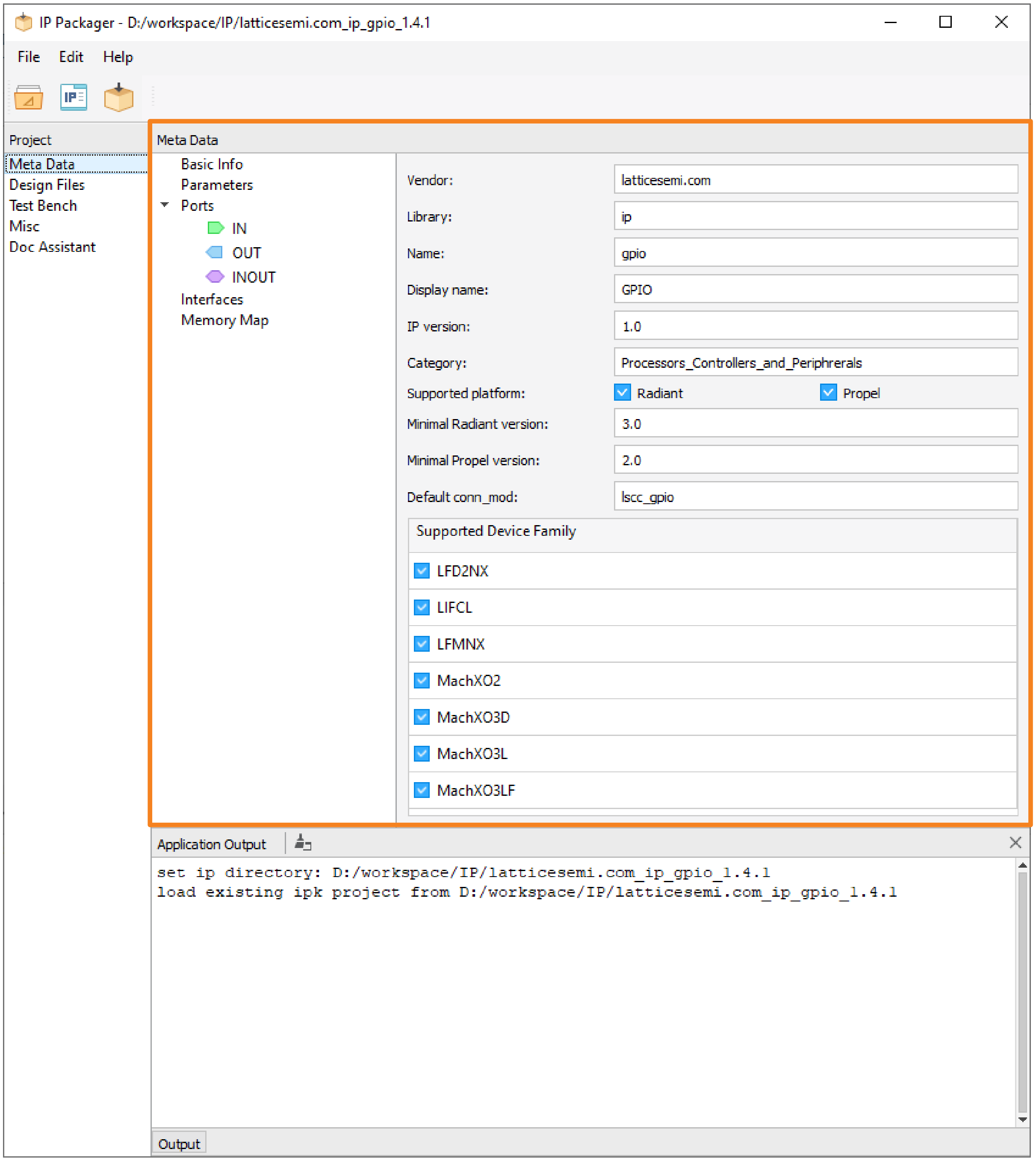


Figure 3.6. IP Packager with Meta Data Details

a. To configure basic information:
   - Click **Basic Info** from Meta Data view (Figure 3.7).

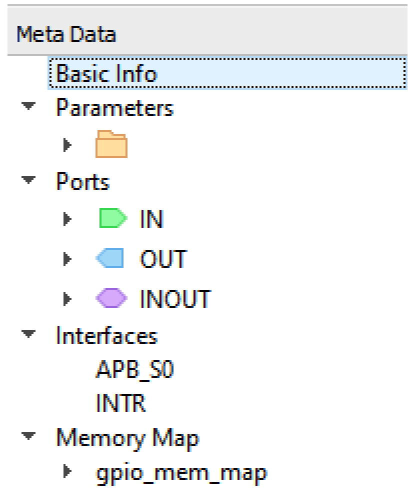


Figure 3.7. Meta Data View

- You can see the Basic Info properties in detail as shown in Figure 3.8.

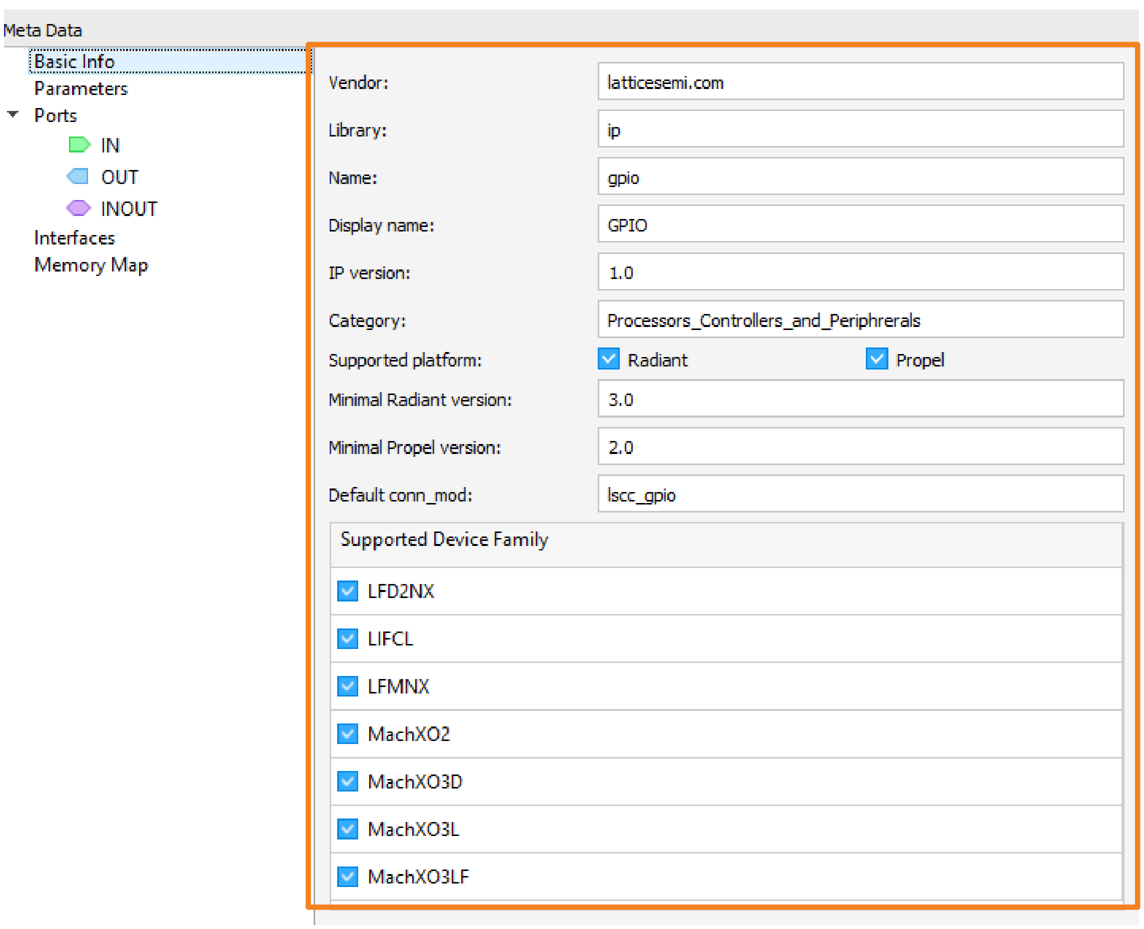


Figure 3.8. Configure Basic Info

- Configure the basic information by entering desired value for a property or checking the checkbox for the property in the property field. For example, click the **Vendor** filed to enter a desired vendor value.

  Note: IP Packager supports both Radiant and Propel software. IP Packager is based on Radiant 3.0, but does not support Radiant version of which is lower than 3.0. “Minimal Radiant version” should be set to 3.0 or 3.0+. For Propel, IP Packager is based on Propel 2.0, not support Propel version of which is lower than 2.0. “Minimal Propel version” should be set to 2.0 or 2.0+. If the version of Radiant or Propel is invalid, the warning message is shown (Figure 3.9).

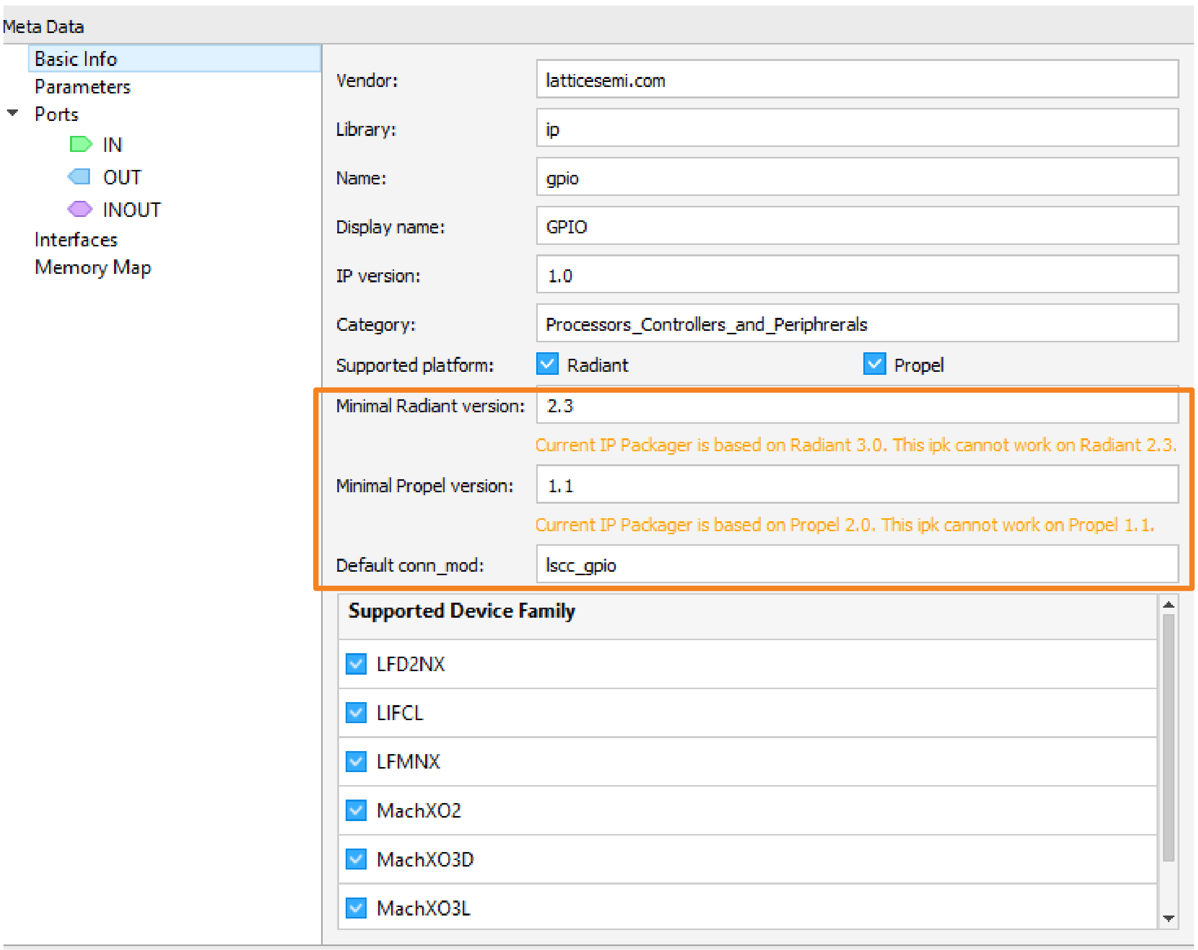


Figure 3.9. Warning Message of the Invalid Radiant and Propel Version

b. To configure parameters:

- Right-click **Parameters** from Meta Data view, and choose **Add Parameter Tab** (Figure 3.10). Parameters create Tab1 items.

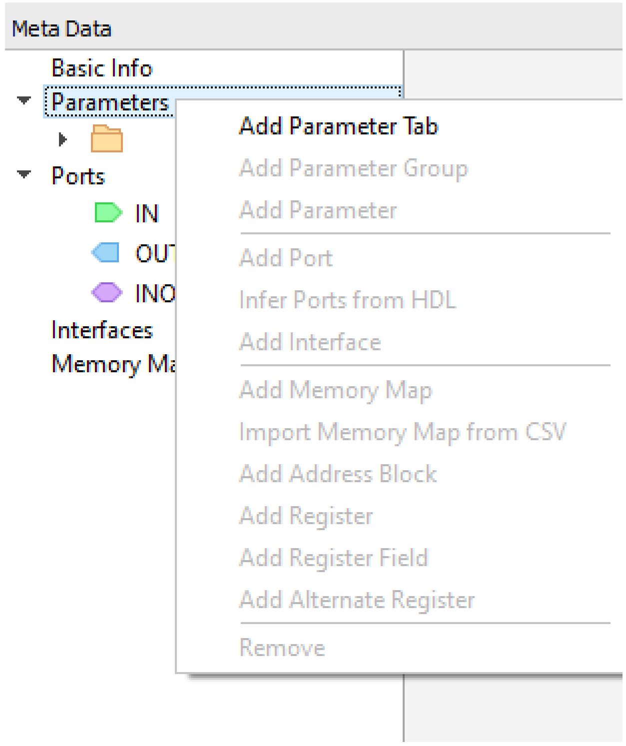


Figure 3.10. Right-click Menu of Parameters in the Meta Data View

- Right-click **Tab1** under **Parameters** from Meta Data view, and choose **Add Parameter Group**. **Group1** is newly added under **Tab1** (Figure 3.11). Group1 is used to group the settings. Settings of the same "group1" are displayed as sub-items under a group item on GUI. The settings with the same "group1" should be written continuously if you want GUI to group them under one group item, otherwise you can see multiple groups with the same name in GUI. Tab1 is used to Group the "group1" groups. "group1" groups of the same "Tab1" are displayed in a separate page on GUI. So, two level hierarchy is supported in GUI display. Unlike 'group1', you need not write setting nodes with the same 'Tab1' continuously but you must follow the rule for 'group1' that all the settings with the same 'group1' must be in the same 'Tab1'.

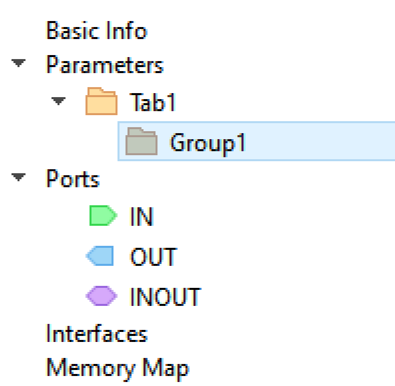


Figure 3.11. Group1 Added to Tab1

- Right-click **Group1** under **Tab1** from the Meta Data view, and choose **Add Parameter**. **NewParam1** is newly added under **Group1** (Figure 3.12).

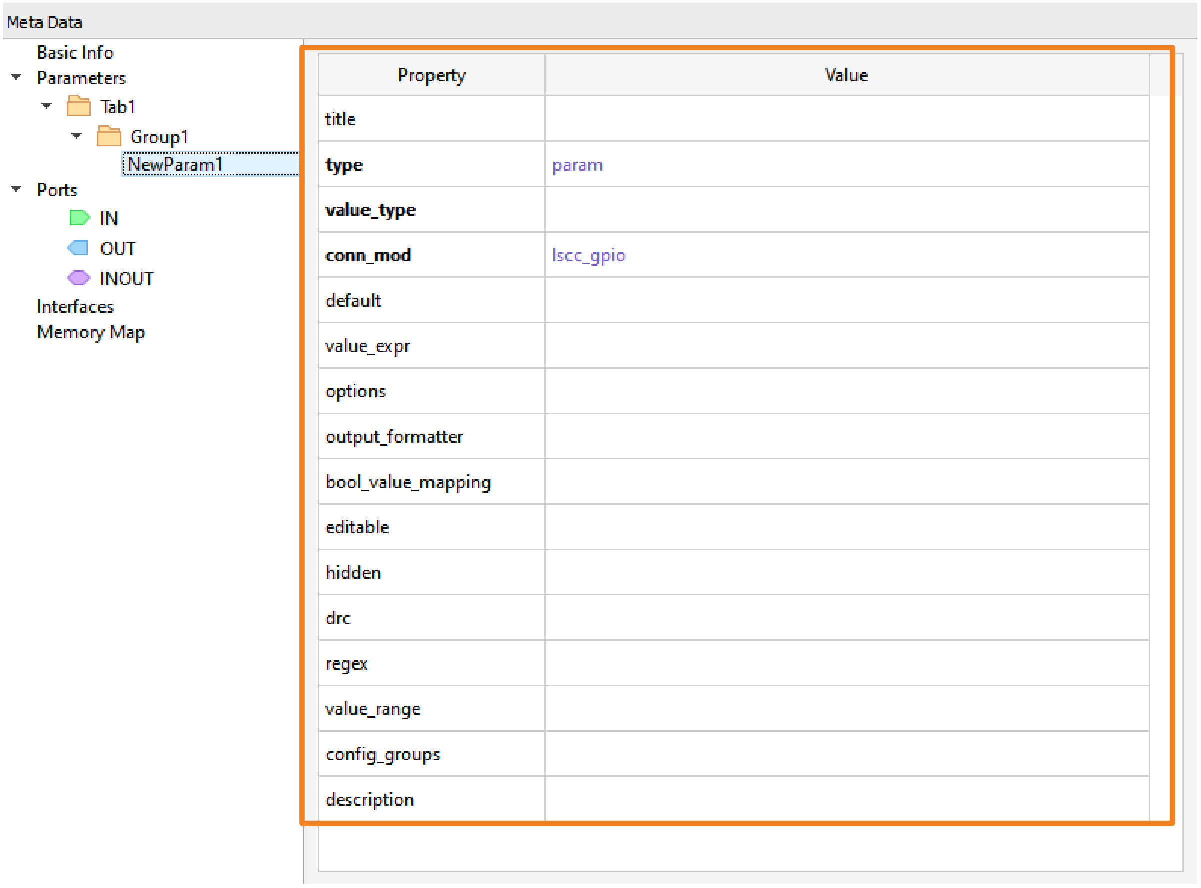


Figure 3.12. NewParam1 Added to Group1

- Double-click **NewParam1** to rename the parameter (Figure 3.13). The parameter name is also used as the unique ID of the parameter setting.

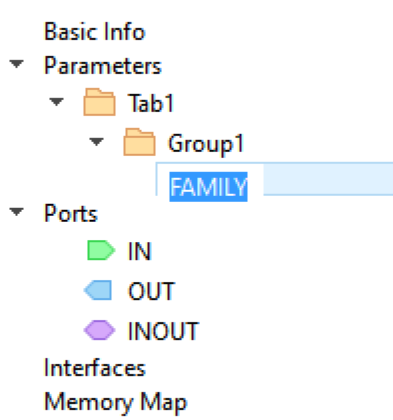


Figure 3.13. Rename a Parameter

- Double-click the property value field to enter the desired parameter value, or select the desired value from drop-down menu (Figure 3.14). The properties are thus configured. Property listed in bold must be configured. The details of each property is shown in Table 3.1

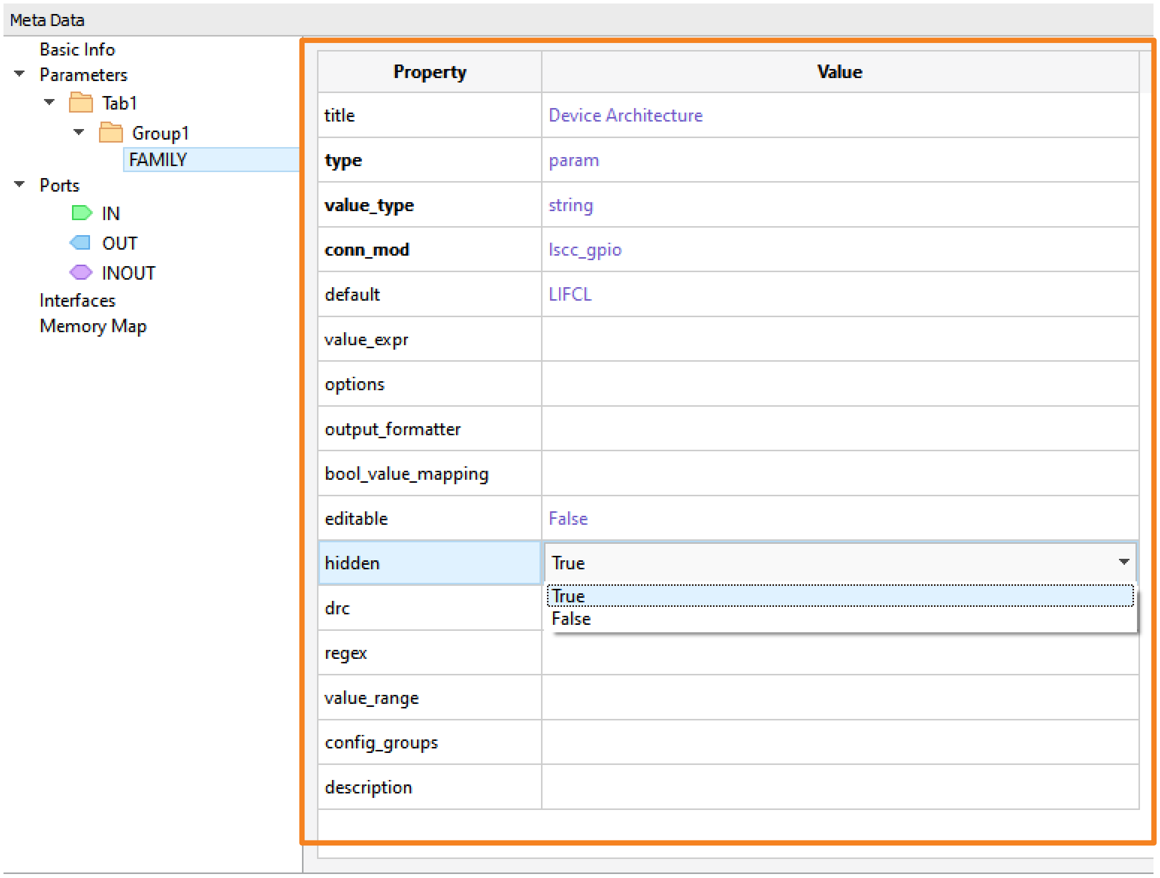


Figure 3.14. Configure a Parameter

Table 3.1. Details of Parameter Property

Property	Value	Mandatory	Description
title	String	No	Short title of the setting. If "title" is not specified, value of "id" is used. Example: title="Device Architecture"
type	param, input, command	Yes	A setting can be a Verilog parameter, Verilog macro definition or user input. Param, verilog_macro and input settings can be used to compute values of other param and input settings. They only differ in generated files. param is written out as a Verilog parameter value pf the IP module, verilog_macro is translated to a Verilog macro definition by the define compiler directives, but input is not. Command shows as a button. Example: type="param"
value_type	bool, string, int, float, path	Yes	Type of the value. Supported types are bool, int, float, string and path. The int type supports unlimited precision. The float type supports the precision of float type of C programming language. Refer to the Characteristics of Floating Types section of the 1999 ISO/IEC C Standard for details. The path type indicates a string which represents a path. "/" is used as separator. Example: value_type="string"
conn_mod	String	Yes	Name of soft IP module to which this setting applies to. Example: conn_mod="lscc_gpio"

Property	Value	Mandatory	Description
default	Python expression	No	Default value of the setting. If the setting has no "default" attribute but has "options" attribute, the first option is picked as default value. If the setting has neither "default" attribute nor "options" attribute, the initial value of setting is set to 0 for int, 0.0 for float, "" for string and False for bool. By default, it is not allowed to reference to other setting values. Use value_expr as reference values. Example: default="LIFCL"
value_expr	Python expression	No	Python expression to compute the value of the setting. The result is used as the parameter value if the setting is not editable. For example, divider is calculated by frequencies. Example: value_expr=" int(round((sys_clock_freq * 250.0) / i2c_left_desired_frequency))-1"
options	Python list or list of tuples	No	Candidate options for the setting, which is used by GUI to display a dropdown selector. It can be set to a simple list or a list of tuples. If it is a simple list, elements are displayed and written. If it is a list of tuples, the first item in tuple is displayed and the second item in tuple is written. Example: options="[0.1, 0.2, 0.5, 1.0]"
output_formatter	`str`	No	Control how parameter values are written in output RTL files. Following formatters are supported. str: parameter values are written as strings nostr: quotation marks of strings are removed Example: output_formatter="str"
bool_value_mapping	Python tuple or list with 2 string elements	No	The map-to-map bool values to dedicated strings. By default, bool values are written as 1, 0. Example: bool_value_mapping="('True', 'False')"
editable	Python expression	No	Python expression to determine if the setting is editable. When a setting is not editable, it is grayed out in GUI display and its value is computed by value_expr. Otherwise, user input is used. Example: editable="(FEEDBACK_PATH == 'PHASE_AND_DELAY') (FEEDBACK_PATH is a setting ID in metadata.xml)
hidden	True	No	Python expression to determine whether or not the setting is hidden in GUI. If hidden is set to True (default is False), the item is hidden in GUI. The expression is resolved to boolean value after the user setting is changed. Example: hidden="True"
drc	Python expression	No	Python expression to do DRC on the setting. True means DRC pass. Example: drc="check_valid_addr_pre(I2C_LEFT_ADDRESSING_PRE,i2c_left_addressing_width)" (check_valid_addr_pre is defined in plugin.py setting ID: I2C_LEFT_ADDRESSING_PRE i2c_left_addressing_width)
regex	Regular expression	No	Regular expression to do DRC on the value. For example, the value should be prefixed by "0b". Example: regex="0b[01]+"
value_rang	Python tuple	No	Valid range of setting value, which is used to do DRC on the setting. The

Property	Value	Mandatory	Description
e	or list with 2 comparable elements		maximum value can be infinity “float(‘inf’)”. Example: value_range="(0, 1023) if(i2c_right_enable) else (-9999, 9999)"
config_gro ups	Python expression	No	A name or names to group settings together. It is copied from IP-XACT configGroups attribute, and only supports “SystemBuilder” value. If it is defined, related RTL parameter is brought out to IP instance top module, so that System Builder can re-define its value.
description	String	No	Detailed description of the setting.
macro_na me	id, value	No	Specify how to name the Verilog macro in ‘verilog_macro’ type setting item, the ID or value of this setting item. The default value is ‘id’, which means the setting’s ID will be defined as a Verilog macro. If it is set as ‘value’, the evaluated setting value is the Verilog macro. Example: maro_name= "value" Result: `define <setting value> Note: This is only be considered in the setting item whose ‘value_type’ attribute is set to ‘string’.

- Repeat steps above to configure all parameters as desired.

c. To configure Ports:

- Click **Ports** from the Meta Data Tree, three port types (IN, OUT, INOUT) (Figure 3.15) are listed.

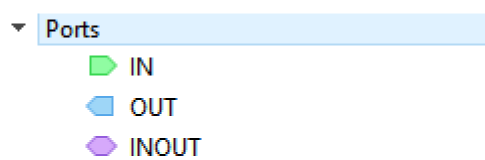


Figure 3.15. Three Port Types

- Right-click **IN** from the Ports area, and choose **Add Port** (Figure 3.16).

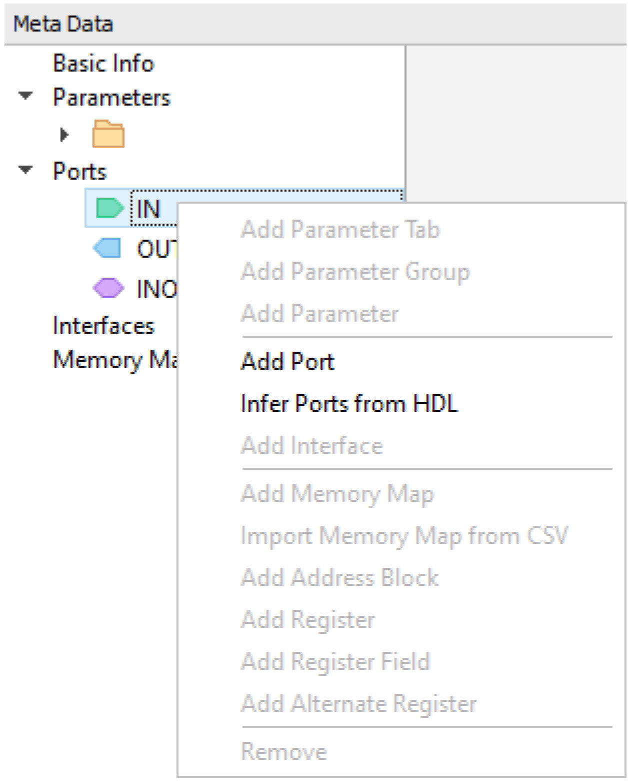


Figure 3.16. Right-click Menu of the IN Port

Note: You can add all ports by using **Infer Ports from HDL**, if you prepare valid RTL files and add RTL files in Design Files section. Click **Infer Ports from HDL**, the Add inferred ports to meta data wizard pops up (Figure 3.17). If you have added some ports and then choose **Infer Ports from HDL**, all the previously-added ports are removed and only ports from the HDL file can be added.

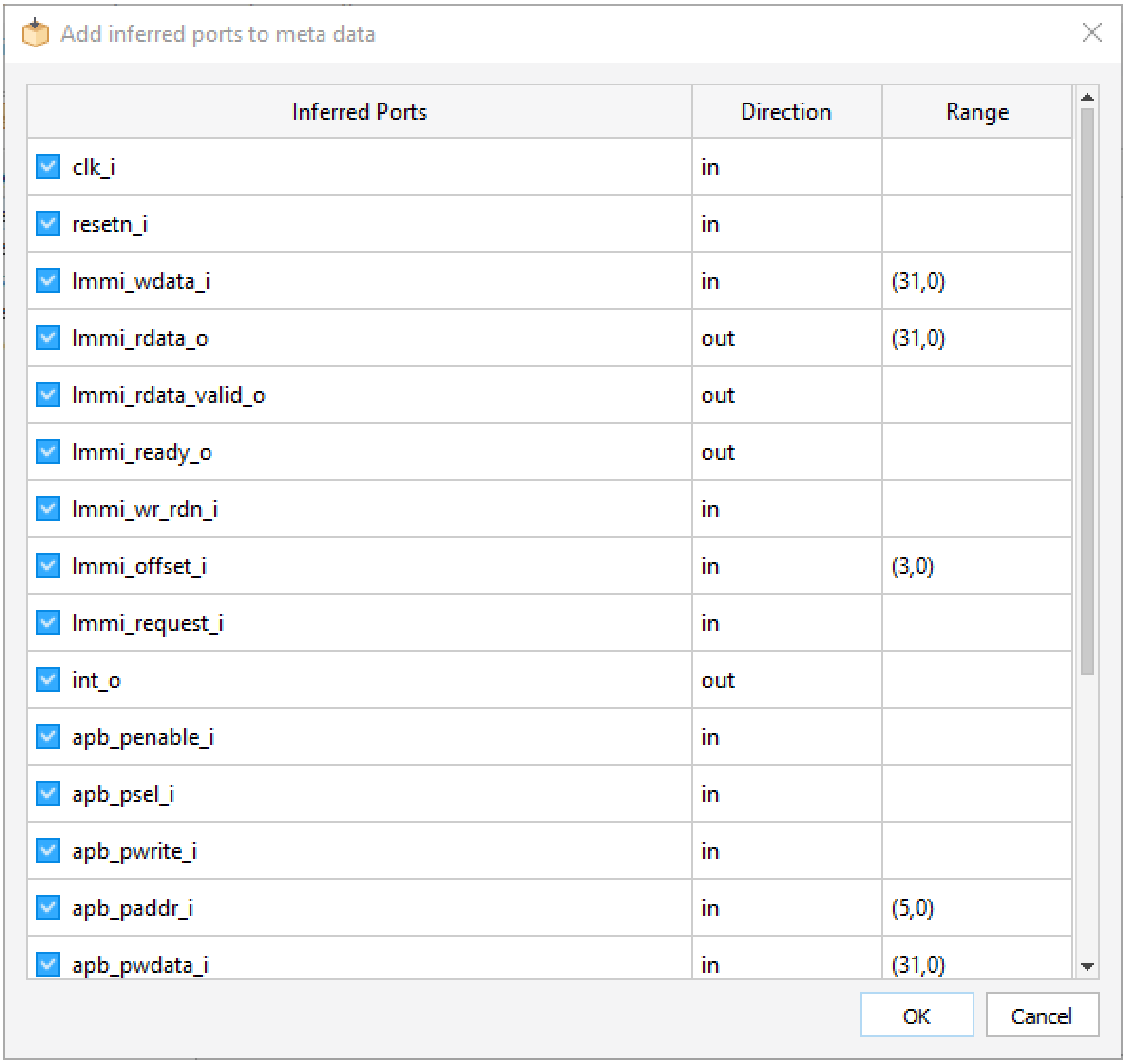


Figure 3.17. Add inferred ports to meta data Wizard

- **NewPort1** is created. Double click NewPort1 to rename the port (Figure 3.18).

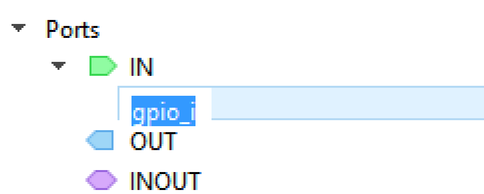


Figure 3.18. Rename an IN Port

- Configure all properties of the IN port as desired (Figure 3.19). Property listed in bold must be configured. The details of each property is shown in Table 3.2.

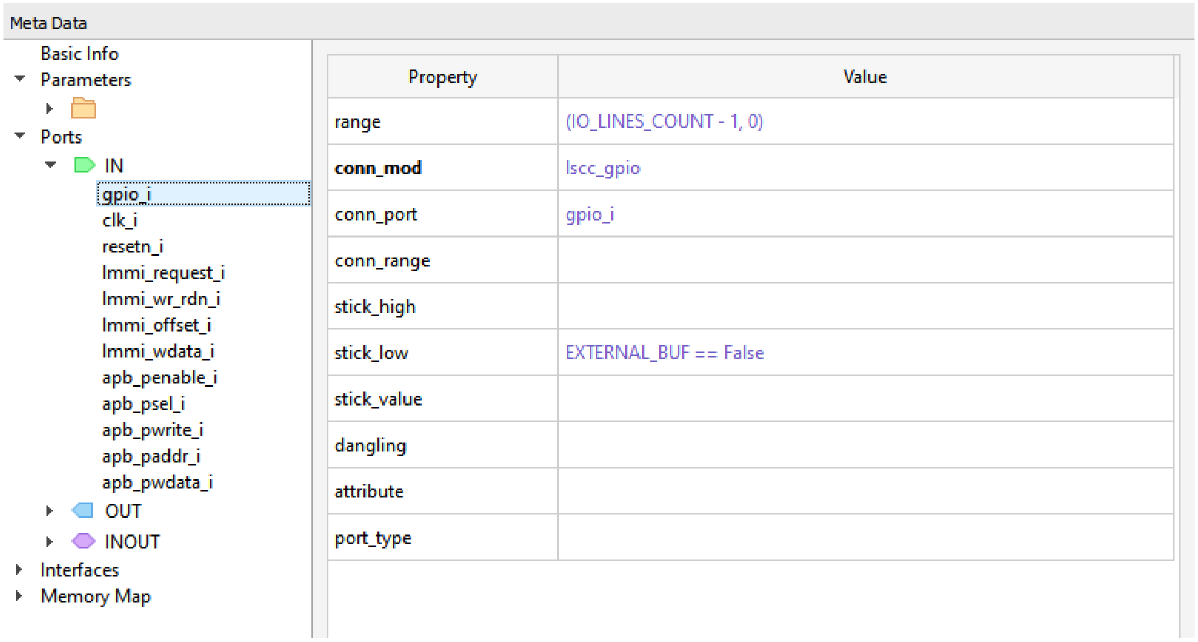


Figure 3.19. Configure an IN Port

Table 3.2. Details of Port Property

Property	Value	Mandatory	Description
range	Python tuple or list with two-integer elements	No	Range of this port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: range="(A_WDT-1, 0)" (A_WDT is a setting ID)
conn_mod	Valid Verilog module name	Yes	Name of the IP core module to which this port connects. Example: conn_mod="counter"
conn_port	Valid Verilog module name	No	Name of port of IP core module to which this port connects. Value of "name" is used if "conn_port" is not specified. Example: conn_port="Clk"
conn_range	Python tuple or list with two-integer elements	No	Range of conn_port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: conn_range="(A_WDT-1, 0)" (A_WDT is a setting ID)
stick_high	Python expression	No	Python script. True: tie this port to 1. Example: stick_high="True"
stick_low	Python expression	No	Python script. True: tie this port to 0. Example: stick_low="no_seq_pins()" (no_seq_pins is defined in plugin.py)
stick_value	Python expression	No	Python script. Tie port to the evaluation result of this attribute.
dangling	Python expression	No	Python script. True: keep this port unconnected. Example: dangling="not USE_COUT" (USE_COUT is a setting ID)
attribute	Python expression	No	Python script. The value can be written to the .v file as the

Property	Value	Mandatory	Description
			attribute of the port. Example: attribute="'(* AAA, BBB=1 *)'" => (* AAA, BBB=1*) input PORT;
port_type	String	No	'data', 'reset' and 'clock' are valid values. The default value is 'data'. port_type can be passed to the IPXact component.xml as an lsccip:isClk or lsscip:isRst node in venderExtensions in component/model/ports/port.

- Repeat steps above to configure all ports as desired.

d. To configure Interfaces:

- Right-click **Interfaces** from the Meta Data view, and choose **Add Interface** (Figure 3.20). A new interface **NewInterface1** is created.

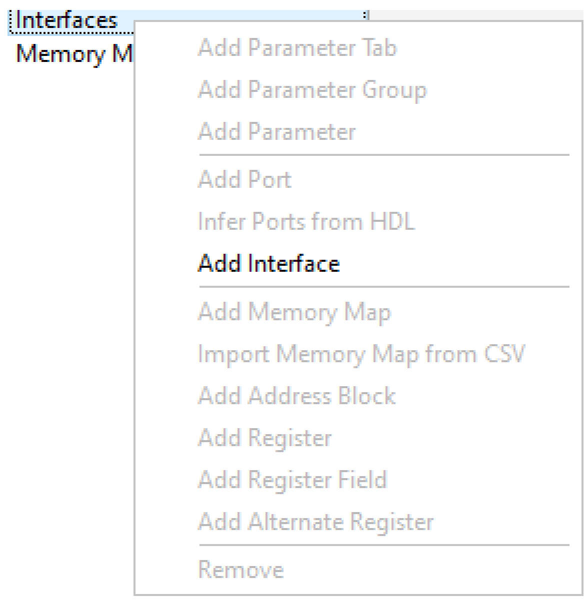


Figure 3.20. Right-click Menu of an Interface

- Double click **NewInterface1** to rename interface (Figure 3.21).

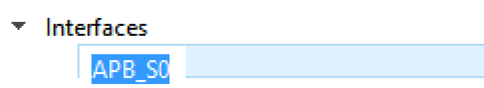


Figure 3.21. Rename Interface

- Configure the interface as desired (Figure 3.22).

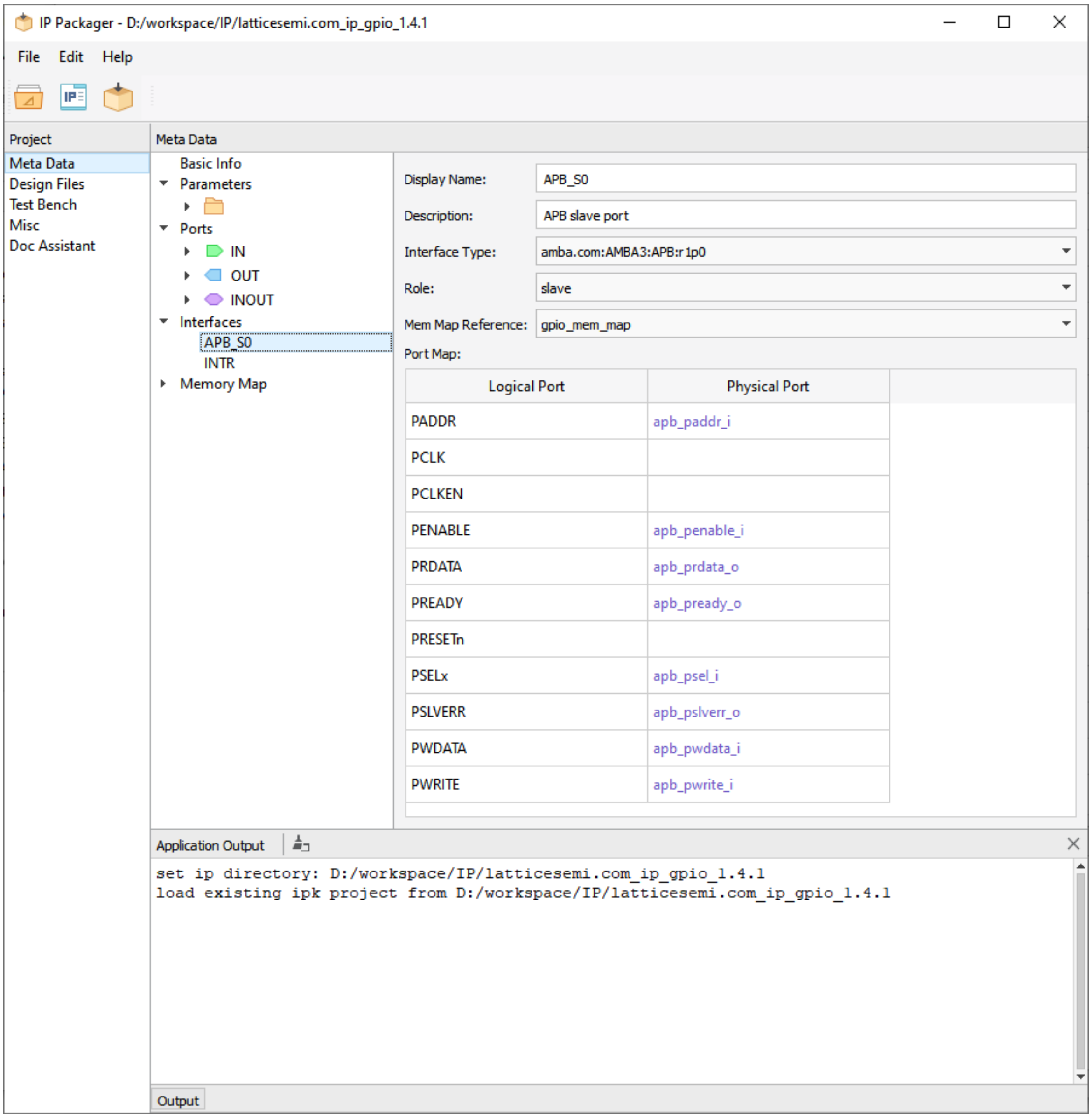


Figure 3.22. Configure an Interface

- Repeat steps above to configure all interfaces as desired.

e. To configure Memory Map:
- Right-click **Memory Map** from the Meta Data Tree, and choose **Import Memory Map from CSV** (Figure 3.23). New memory map is created (Figure 3.24).
  Note: Propel 2.0 Builder only supports using **Import Memory Map from CSV** to create memory map. There are some issues if you use **Add Memory Map**.

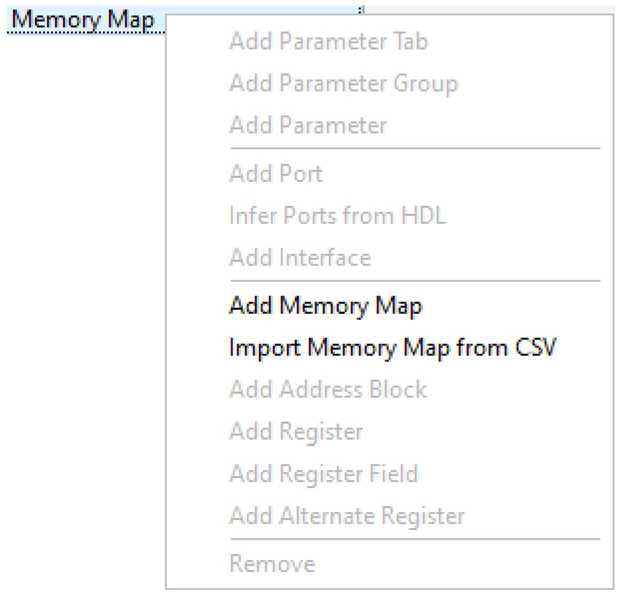


Figure 3.23. Right-click Menu of a Memory Map

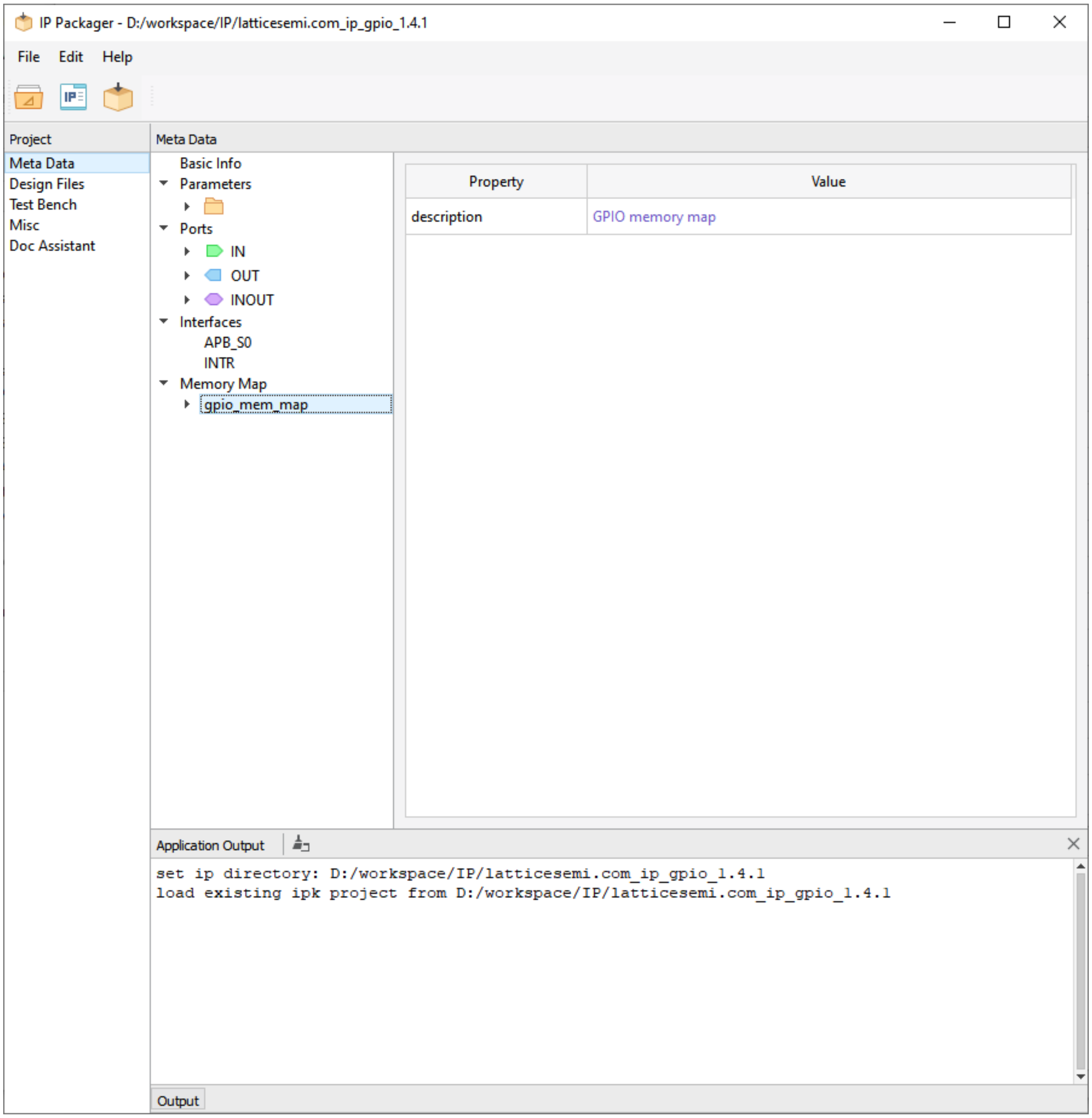


Figure 3.24. Generate New Memory Map

2. From the IP Packager Project area, click **Design Files**. IP Packager GUI shows the Design Files information (Figure 3.25).

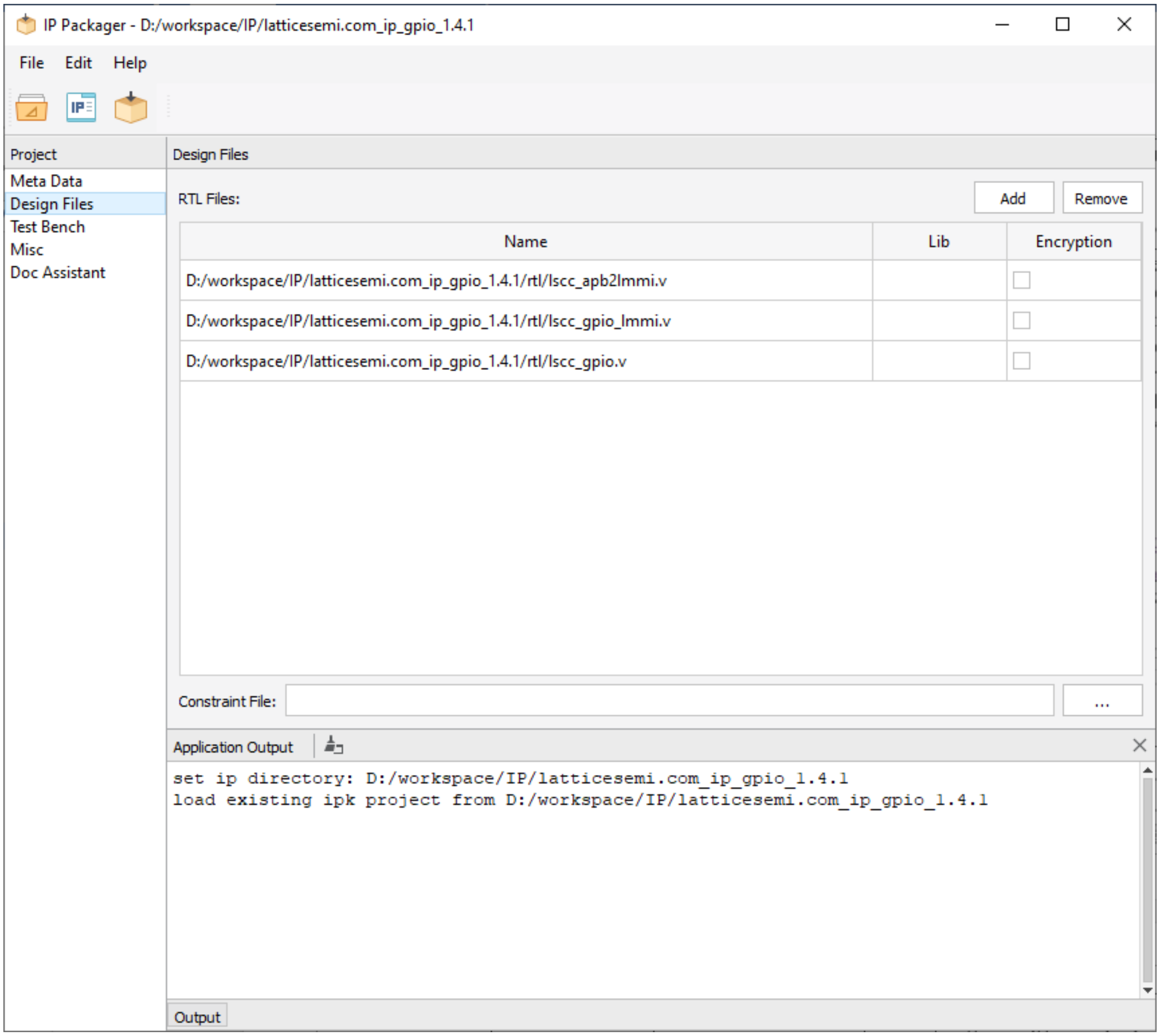


Figure 3.25. IP Packager with Design File Details

- Click the Add button to select a desired rtl file in the Design Files field. Click the Remove button to remove an RTL file. Use the "**…**" option to add a desired constraint file.

  Note: Design file configuration is mandatory. Specify at least one RTL file. If the RTL file needs to be encrypted, check the **Encryption** option for it.

3. From the IP Packager Project area, click **Test Bench**. The Test Bench information is shown in detail (Figure 3.26).

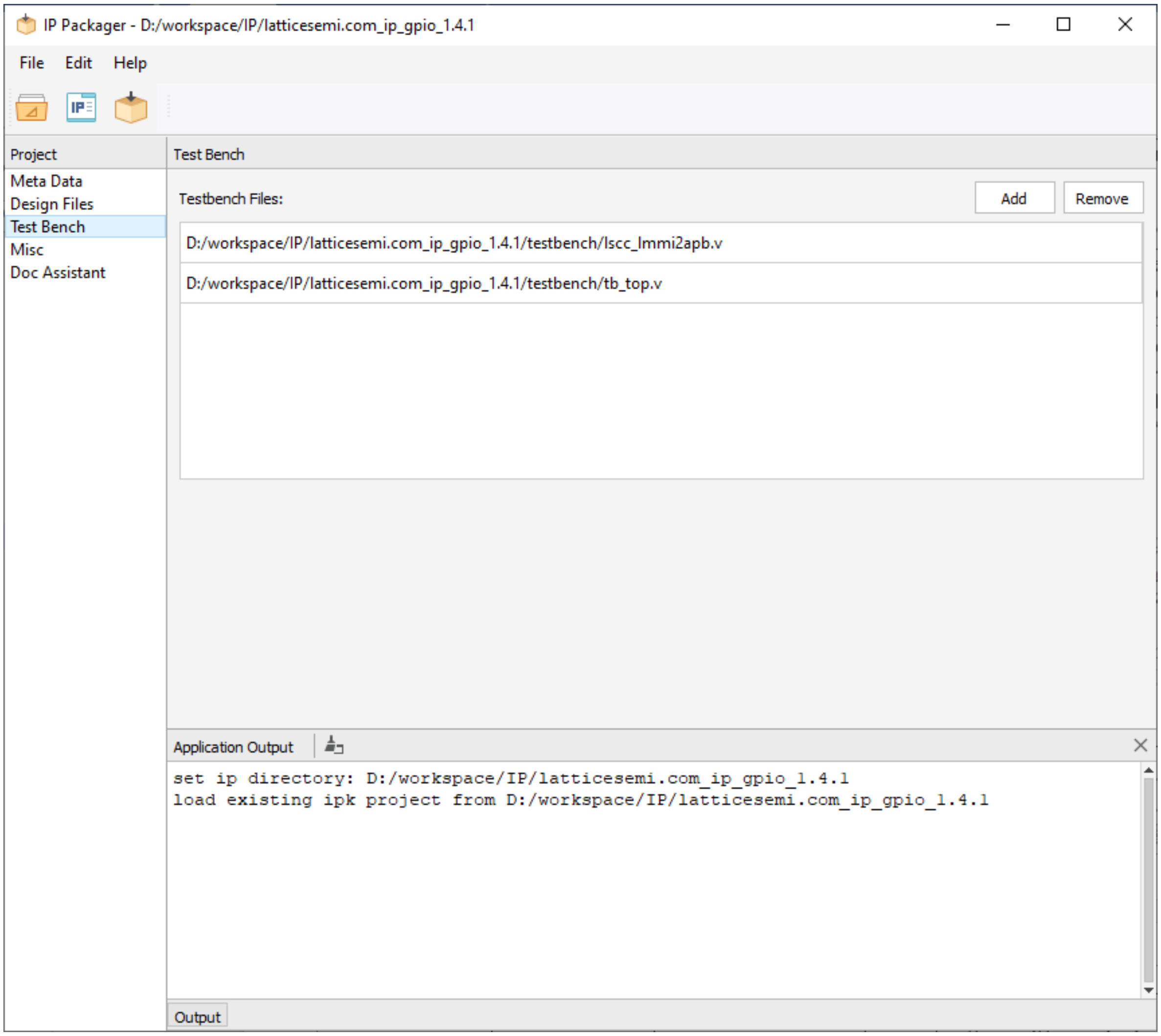


Figure 3.26. IP Packager with Test Bench Details

Follow the steps as those for configuring the Design Files to configure the Test Bench. The Test Bench files configuration is not mandatory.

4. From the IP Packager Project area, click **Misc**. The Misc related information is shown in detail (Figure 3.27). Configure Misc information as desired.

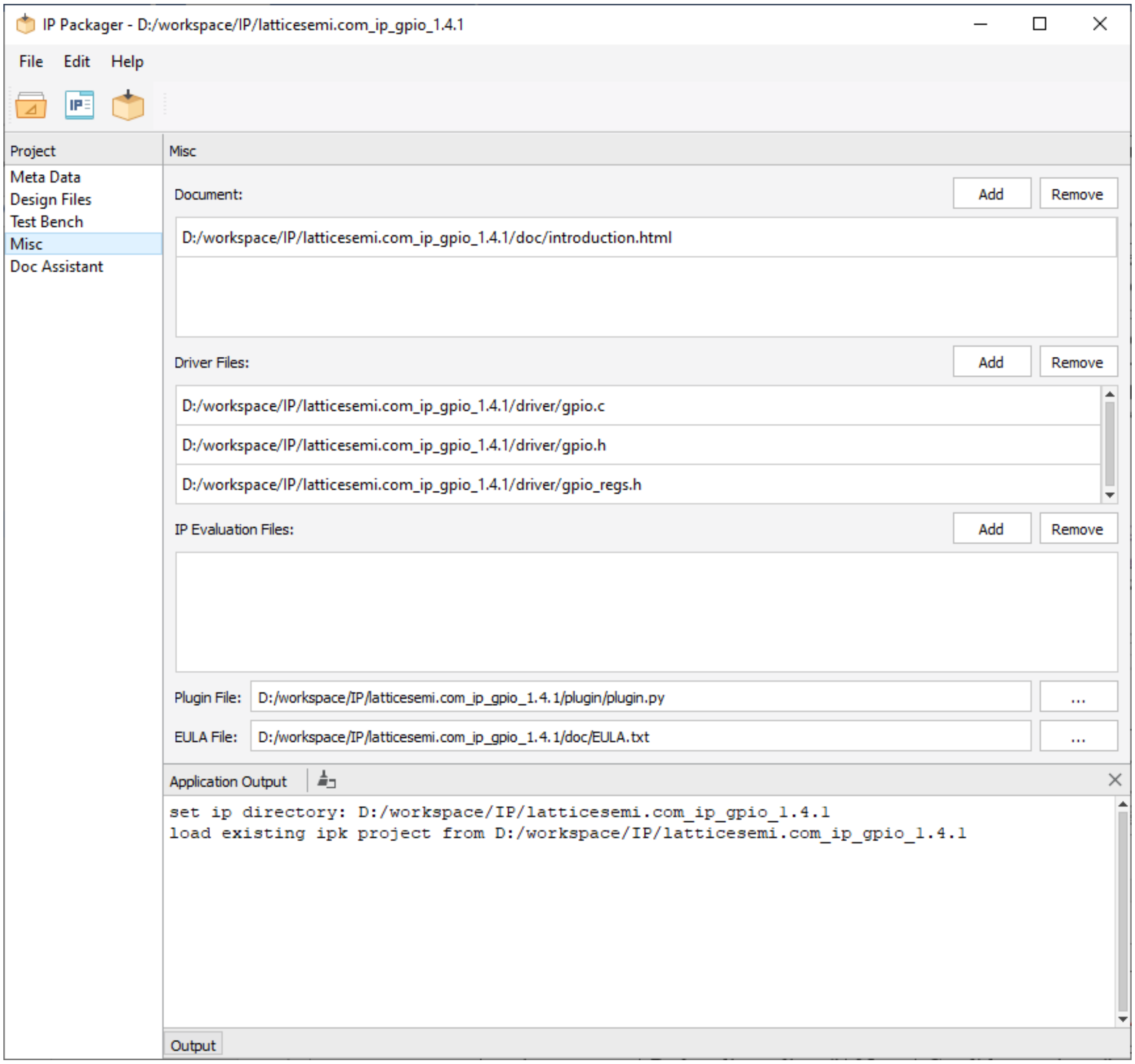


Figure 3.27. IP Packager with Misc Details

5. From the IP Packager Project area, click **Doc Assistant**. The Doc Assistant information is shown in detail (Figure 3.28).

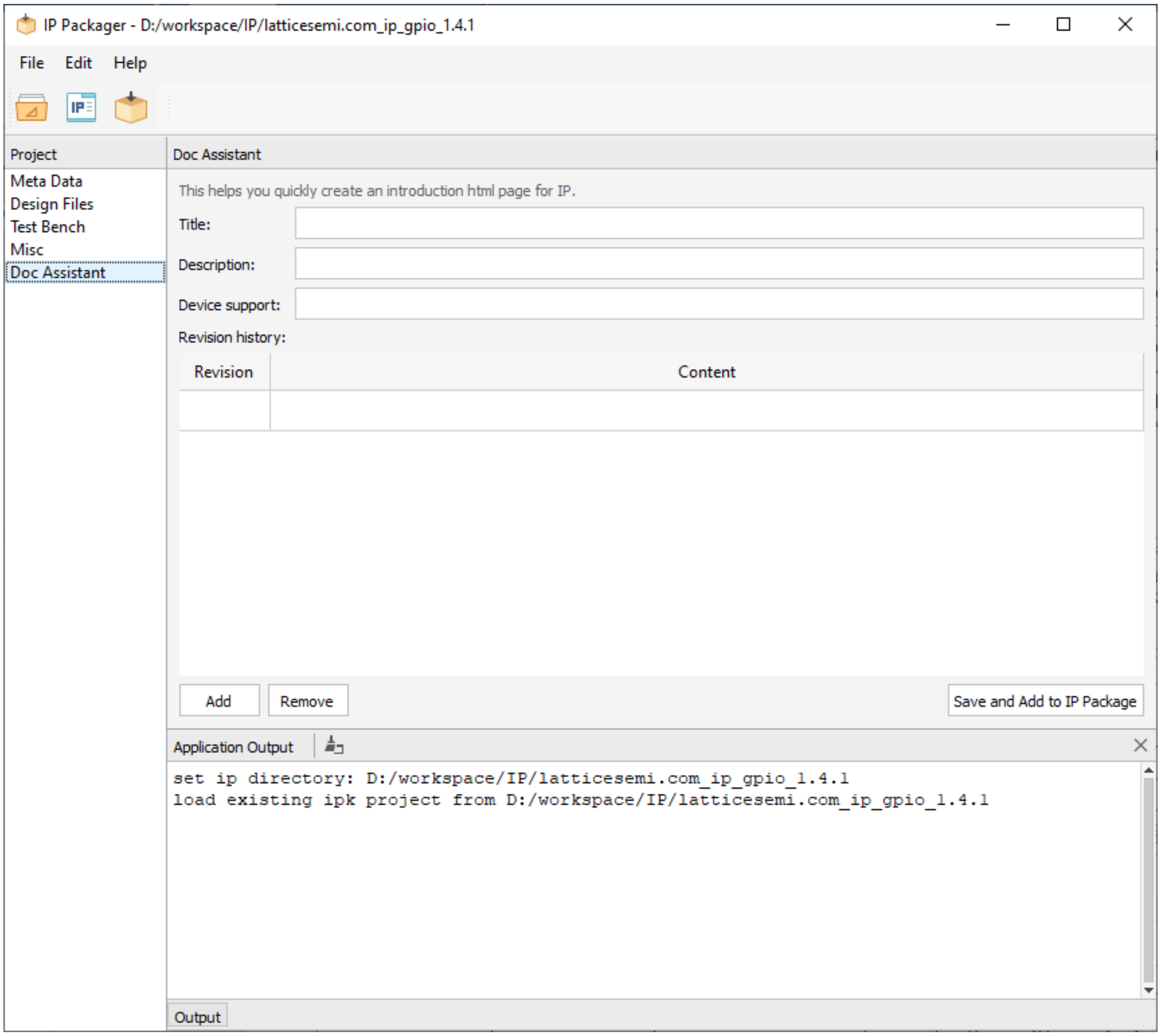


Figure 3.28. IP Packager with Doc Assistant Details

- Enter the desired title, description, and device in the **Title**, **Description**, and **Device support** fields accordingly.
- Double-click and enter a revision in **Revision** field. Double-click and enter revision contents in **Content** field. Refer to Figure 3.29 as an example. Click the Add button. A new blank row is added to the Revision History area. You can add desired revision and contents accordingly. Select the revision you want to delete, and click the Remove button to remove a Revision.
- Click the Save and Add to IP Package button to create an introduction html page and save this file to the IP package.

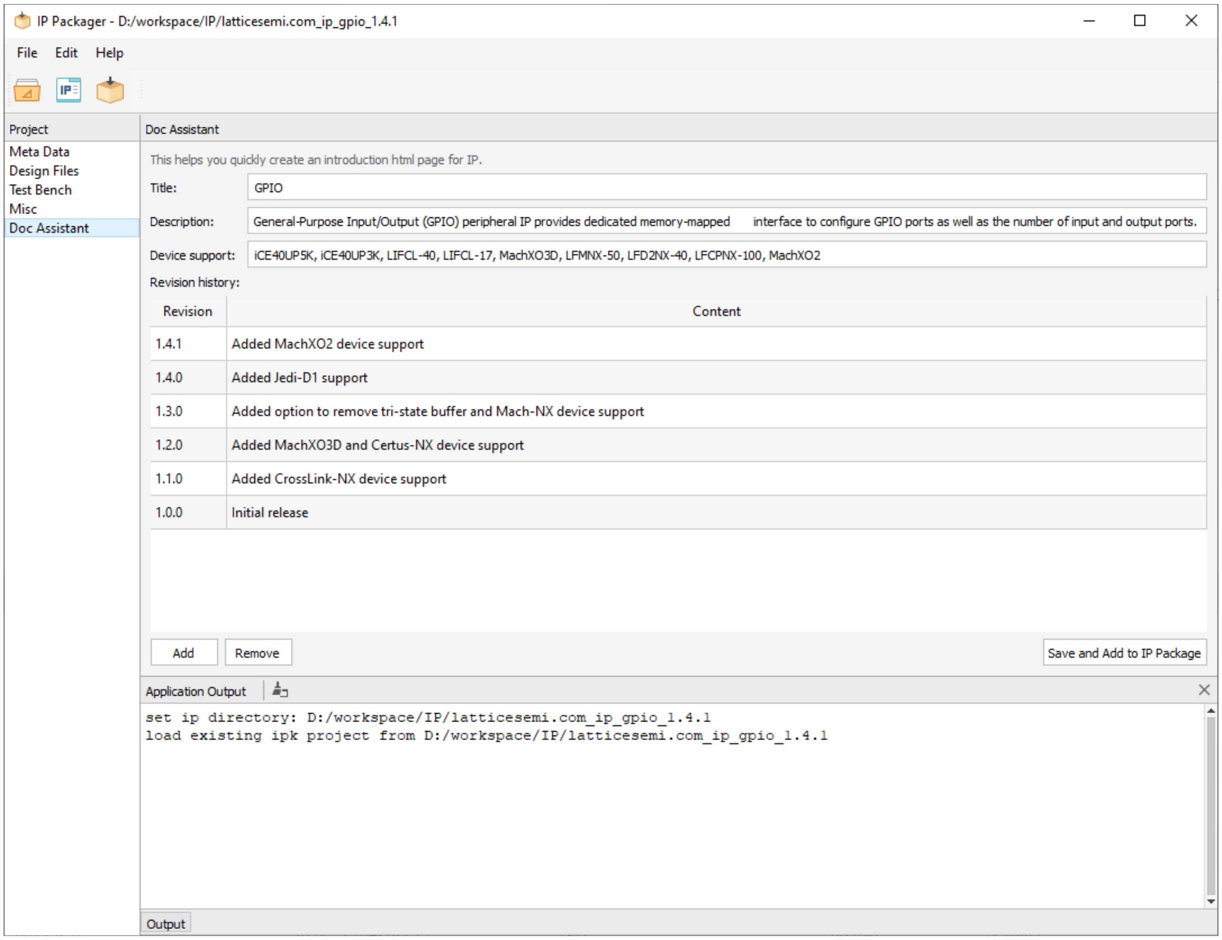


Figure 3.29. Configure Doc Assistant

3.2.3. Previewing IP

1. Choose **Edit** > **IP Preview** from Propel Builder menu. Or, click the **IP Preview** icon from Propel Builder Toolbar.
2. IP Preview pops up showing the configuration component for the IP module (Figure 3.30), if the IP project configuration is correct. Otherwise, the IP Packager pops up an error message (Figure 3.31).

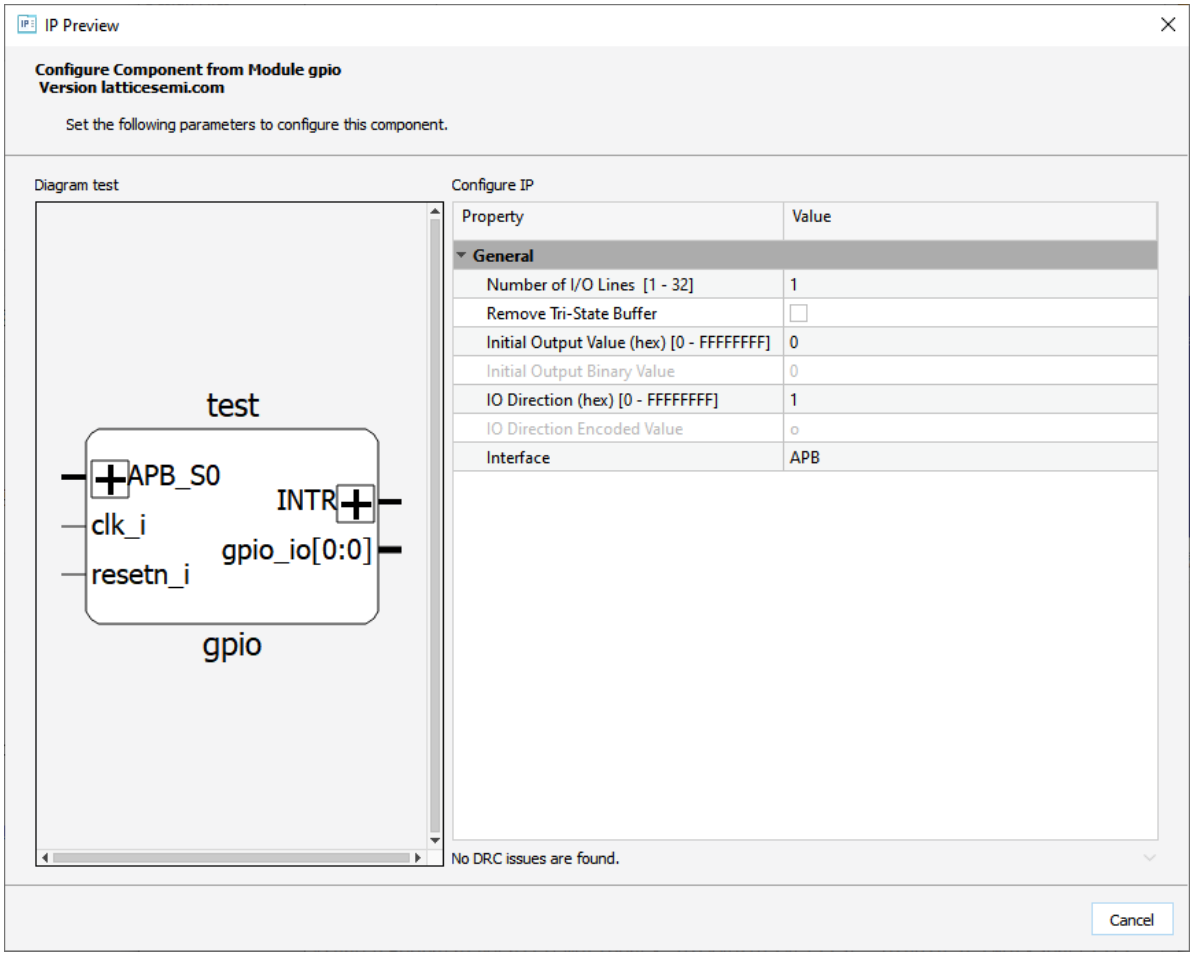


Figure 3.30. IP Preview Shows Configuration for IP Module

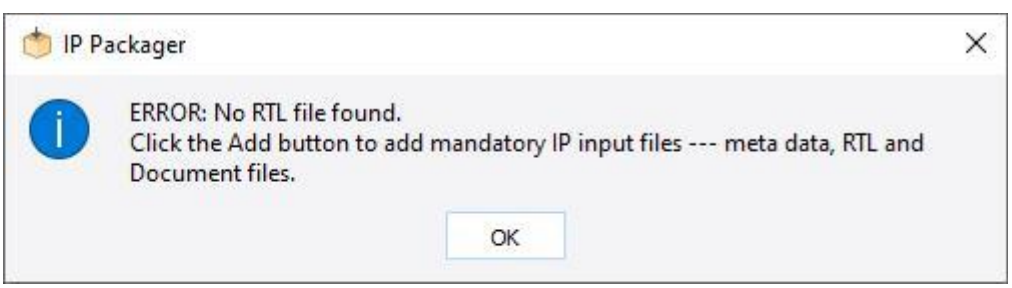


Figure 3.31. IP Packager Pops up Error Message

3.2.4. Packaging IP

1. Choose **Edit** > **Package IP** from Propel Builder menu. Or, click the **Package IP** icon from Propel Builder Toolbar.
2. IP Packager pops up a message showing the packaging is completed successfully (Figure 3.32).

Figure 3.32. IP Packager Pops up Successful Packaging Message

3.3. Editing IP Package Files

3.3.1. Meta Data File

Metadata.xml can be generated after editing IP. You also can manually edit the file. IP Platform uses XML file to describe metadata of a soft IP. Namespace "lsccip" is used in XML file. The content of the XML file consists of three mandatory nodes including <general>, <settings> and <ports>, five optional nodes including <busInterfaces>, <addressSpaces>, <memoryMaps>, <componentGenerators> and <estimatedResources>. One optional new child node <outFileConfigs> is added to support the customized IP generation flow in Propel (Figure 3.33).

```
<lsccip:ip
xmlns:lsccip="http://www.latticesemi.com/XMLSchema/Radiant/ip"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.0">
  <lsccip:general>
    ......
  </lsccip:general>
  <lsccip:settings>
    ......
  </lsccip:settings>
  <lsccip:ports>
    ......
  </lsccip:ports>
</lsccip:ip>
```

Figure 3.33. Example XML of Metadata Layout

1. <general> node: describes general information about a soft IP, for example, name, version, and category. Table 3.3 shows the child nodes of <general> node.

Table 3.3. Child Nodes of General Node

Child Node	Mandatory	Description
vendor	Yes	Soft IP vendor. Official soft IPs should have vendor name. Example: “latticesemi.com”
library	Yes	Library of the soft IP. If “library” is not set, the default value “ip” is used. Example: “ip”, “interface”
name	Yes	Name of the soft IP. The name must be unique among soft IPs of the same vendor and library. Example: "adder"
display_name	No	Name to be displayed in the software. If “display_name” is not set, software displays “name” directly. Example: “Adder”
version	Yes	Version of the soft IP. Example: 1.0.0
category	Yes	Category of the soft IP. Category can be hierarchical. Levels are separated by “,”. Example: Memory_Modules,Distributed_RAM
keywords	No	Keywords of the soft IP. Multiple keywords are separated by “,”. Example: BusType_AHB,BusType_APB
min_radiant_version	Yes	The minimal Radiant version, which supports the soft IP. Example: 1.0 (no service pack), 1.0.1 (with service pack)
max_radiant_version	No	The maximal Radiant version, which supports the soft IP. Example: 2.0.
supported_products	No	FPGA products supported by the soft IP.
type	No	Enhancement for cpu IP.
min_esi_version	No	The minimal ESI version, which supports the soft IP. Enhancement for esi.
max_esi_version	No	The maximal ESI version, which supports the soft IP. Enhancement for esi.
supported_platforms	No	Default is Radiant, specific for esi.

2. <settings> node: describes parameters information that should contain one or more <setting> nodes. In an IP instance package, Verilog parameters are used to configure the soft IP. All user configurable parameters should be added to the <settings> section as <setting> nodes. Beside parameters, you can add <setting> nodes for user-input only. Table 3.4 shows attributes of <setting> nodes.

Table 3.4. Attributes of Setting Nodes

Attribute	Value	Mandatory	Description
id	String	Yes	The unique ID of the setting, which is also be referred as: Parameter name in RTL codes Python variable name Tcl variable name To make value of the “id” valid in Verilog HDL, Python and Tcl, it should consist of only letters, digits and underscore. The first character should be letter. Example: id=”num_outputs”
title	String	No	Short title of the setting. If “title” is not specified, value of “id” is used. Example: title=”Number of Output”
type	param, input, command	Yes	A setting can be a Verilog parameter, Verilog macro definition or user input. Param, verilog_macro and input settings can be used to compute values of other param and input settings. They only differ in generated files. param is written out as a Verilog parameter value pf the IP module, verilog_macro is translated to a Verilog macro definition by the define compiler directives, but input is not. Command shows as a button. Example: type=”param”
value_type	bool, string, int, float, path	Yes	Type of the value. Supported types are bool, int, float, string and path. The int type supports unlimited precision. The float type supports the precision of float type of C programming language. Refer to the Characteristics of Floating Types section of the 1999 ISO/IEC C Standard for details. The path type indicates a string which represents a path. “/” is used as separator. Example: value_type=”int”
conn_mod	String	Yes	Name of soft IP module to which this setting applies to. Example: conn_mod=”pll”
default	Python expression	No	Default value of the setting. If the setting has no “default” attribute but has "options" attribute, the first option is picked as default value. If the setting has neither “default” attribute nor “options” attribute, the initial value of setting is set to 0 for int, 0.0 for float, “” for string and False for bool. Example: default=”1.0”
value_expr	Python expression	No	Python expression to compute the value of the setting. The result is used as parameter value if the setting is not editable. For example, divider is calculated by frequencies. Example: value_expr=" int(round((sys_clock_freq * 250.0) / i2c_left_desired_frequency))-1"
options	Python list or list of tuples	No	Candidate options for the setting, which is used by GUI to display a dropdown selector. It can be set to a simple list or a list of tuples. If it is a simple list, elements are displayed and written. If it is a list of tuples, the

Attribute	Value	Mandatory	Description
			first item in tuple is displayed and the second item in tuple is written. Example: options="[0.1, 0.2, 0.5, 1.0]"
output_formatter	str	No	Control how parameter values are written in output RTL files. Following formatters are supported. str: parameter values are written as strings nostr: quotation marks of strings are removed Example: output_formatter="str"
bool_value_mapping	Python tuple or list with 2 string elements	No	The map-to-map bool values to dedicated strings. By default, bool values are written as 1, 0. Example: bool_value_mapping="('True', 'False')"
editable	Python expression	No	Python expression to determine if the setting is editable. When a setting is not editable, it is grayed out in GUI display and its value is computed by value_expr. Otherwise, user input is used. Example: editable="(FEEDBACK_PATH == 'PHASE_AND_DELAY') (FEEDBACK_PATH is a setting ID in metadata.xml)
hidden	True	No	Python expression to determine whether the setting is hidden in GUI. If hidden is set to True (default is False), the item is hidden in GUI. The expression is resolved to boolean value after the user setting is changed. Example: hidden="True"
drc	Python expression	No	Python expression to do DRC on the setting. True means DRC pass. Example: drc="check_valid_addr_pre(I2C_LEFT_ADDRESSING_PRE,i2c_left_addressing_width)" (check_valid_addr_pre is defined in plugin.py setting ID: I2C_LEFT_ADDRESSING_PRE i2c_left_addressing_width)
regex	Regular expression	No	Regular expression to do DRC on the value. For example, the value should be prefixed by "0b". Example: regex="0b[01]+"
value_range	Python tuple or list with 2 comparable elements	No	Valid range of setting value, which is used to do DRC on the setting. The maximum value can be infinity "float('inf')". Example: value_range="(0, 1023) if(i2c_right_enable) else (-9999, 9999)"
config_groups	Python expression	No	A name or names to group settings together. It is copied from IP-XACT configGroups attribute, and only supports "SystemBuilder" value. If it is defined, related RTL parameter is brought out to IP instance top module, so that System Builder can re-define its value.
description	String	No	Detailed description of the setting.
group1	String	No	Group the settings. Settings of the same "group1" is displayed as sub-items under a group item on GUI. The settings with the same "group1" should be written continuously if you want GUI to group them under one group item. Otherwise, you can see multiple groups with the same name in GUI. Example: group1= "Output Setting"
group2	String	No	Group the "group1" groups. "group1" groups of the same "group2" is displayed in a separate page on GUI. So, two-level hierarchy is supported in GUI display.

Attribute	Value	Mandatory	Description
			Unlike 'group1', you need not write setting nodes with the same 'group2' continuously. You must follow the rule for 'group1' that all the settings with the same 'group1' must be in the same 'group2'.
macro_name	id, value	No	Specify how to name the Verilog macro in 'verilog_macro' type setting item, the ID or value of this setting item. The default value is 'id', which means the setting ID is defined as a Verilog macro. If it is set as 'value', the evaluated setting value is the Verilog macro. Example: maro_name= "value" Result: `define <setting value> Note: This is only be considered in the setting item whose 'value_type' attribute is set to 'string'.

3. <ports> node: IP module package has some ports in its implementation. These ports should be described in <ports> section as <port> child nodes. If an input port is stick to fixed value, or an output port is dangling, the port is not used and is invisible after generation. Table 3.5 shows the attributes of <port> nodes.

Table 3.5. Attributes of Port Nodes

Attribute	Value	Mandatory	Description
name	Valid Verilog port name	Yes	Name of a port. Example: name="Clk"
dir	in, out, inout	Yes	Direction of a port. Example: dir="in"
range	Python tuple or list with 2 int elements	No	Range of this port. It should be a Python expression whose evaluation result is a tuple or array with 2 elements. Example: range="(A_WDT-1, 0)" (A_WDT is a setting ID)
conn_mod	Valid Verilog module name	Yes	Name of an IP core module to which this port connects. Example: conn_mod="counter"
conn_port	Valid Verilog module name	No	Name of port of an IP core module to which this port connects. Value of "name" is used, if "conn_port" is not specified. Example: conn_port="Clk"
conn_range	Python tuple or list with 2 int elements	No	Range of conn_port. It should be a Python expression whose evaluation result is a tuple or array with 2 elements. Example: conn_range="(A_WDT-1, 0)" (A_WDT is a setting ID)
stick_high	Python expression	No	Python script. True: tie this port to 1. Example: stick_high="True"
stick_low	Python expression	No	Python script. True: tie this port to 0. Example: stick_low="no_seq_pins()" (no_seq_pins is defined in plugin.py)
stick_value	Python expression	No	Python script. Tie port to the evaluation result of this attribute.
dangling	Python expression	No	Python script. True: keep this port unconnected. Example: dangling="not USE_COUT"

Attribute	Value	Mandatory	Description
			(USE_COUT is a setting ID)
bus_interface	Valid bus interface name	No	Bus interface name defined in <busInterfaces> node. Example: bus_interface=" ahb_slave_0"
attribute	Python expression	No	Python script. The value is written to the .v file as the attribute of the port. Example: attribute="'(* AAA, BBB=1 *)'" => (* AAA, BBB=1*) input PORT;
port_type	String	No	'data', 'reset' and 'clock' are valid values. The default value is 'data' port_type will be passed to the IPXact component.xml as an lsccip:isClk or lsscip:isRst node in venderExtensions in component/model/ports/port

4. <busInterfaces> node: contains a list of all interfaces ports of the soft IP (Figure 3.34). The description follows IP-XACT format. Refer to IEEE 1685-2014: IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows for more details. The busInterface has a list of portMap, which defines logicalPort name and physicalPort name. In <ports> nodes, one port can have optional "bus_interface" attribute, which binds the port with a busInterface. If one port is bound to the busInterface, its name should match physicalPort name of one portMap of the busInterface, so that port and busInterface/portMap is bound together. Whether or not a port can be used is based on user configuration. If the port is used, the corresponding portMap in busInterface is written to the output IP-XACT file. Otherwise, the corresponding portMap in busInterface is not written to the output IP-XACT file.

```
<lsccip:busInterfaces>
    <lsccip:busInterface>
        <lsccip:name>AHBL_S00</lsccip:name>
        <lsccip:displayName>AHBL_S00</lsccip:displayName>
        <lsccip:description>AHB-Lite slave
port</lsccip:description>
        <lsccip:busType library="interface" name="ahblite"
vendor="lattice" version="1.0"/>
        <lsccip:abstractionTypes>
            <lsccip:abstractionType>
                <lsccip:abstractionRef library="interface"
name="ahblite_rtl" vendor="lattice" version="1.0"/>
                <lsccip:portMaps>
                    <lsccip:portMap>
                        <lsccip:logicalPort>
                            <lsccip:name>HSEL</lsccip:name>
                        </lsccip:logicalPort>
                        <lsccip:physicalPort>

<lsccip:name>ahbl_s00_hsel_slv_i</lsccip:name>
                        </lsccip:physicalPort>
                    </lsccip:portMap>
                    <lsccip:portMap>
                        <lsccip:logicalPort>
                            <lsccip:name>HADDR</lsccip:name>
                        </lsccip:logicalPort>
                        <lsccip:physicalPort>

<lsccip:name>ahbl_s00_haddr_slv_i</lsccip:name>
                        </lsccip:physicalPort>
                    </lsccip:portMap>
                </lsccip:portMaps>
            </lsccip:abstractionType>
        </lsccip:abstractionTypes>
        <lsccip:slave>
            <lsccip:memoryMapRef memoryMapRef="ahbs_mem_map"/>
        </lsccip:slave>
    </lsccip:busInterface>
</lsccip:busInterfaces>
```

Figure 3.34. Example of busInterface Node

5. <addressSpaces> node: specifies the addressable area seen by bus interfaces of type master (Figure 3.35). The description follows IP-XACT format. Refer to IEEE 1685-2014: IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows for more details.

```
<lsccip:addressSpaces>
   <lsccip:addressSpace>
      <lsccip:name>ahbm_addr_space</spirit:name>
      <lsccip:range>4k</lsccip:range>
      <lsccip:width>32</lsccip:width>
   </lsccip:addressSpace>
</lsccip:addressSpaces>
```

Figure 3.35. Example of addressSpace Node

6. <memoryMaps> node: specifies the information about the range of registers, memory, or other address blocks accessible through slave interface (Figure 3.36). The description follows IP-XACT format. Refer to IEEE 1685-2014: IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows for more details and examples.
   To represent dynamic availability of some elements, <register> and <field> node are extended with an optional isPresent element (introduced in IEEE Std 1685-2014), which defines whether the enclosing element is present or not. The value of isPresent element is a Python expression. Parameters defined in <settings> node can be used in the expression.
   If a register is used based on user configuration, the register is written to output IP-XACT file. Otherwise, the register is not written to output IP-XACT file.

```
<lsccip:memoryMaps>
    <lsccip:memoryMap>
        <lsccip:name>ahbs_mem_map</lsccip:name>
        <lsccip:description>AHB-Lite Slave 0 memory
map</lsccip:description>
        <lsccip:addressBlock>
            <lsccip:name>registers</lsccip:name>
            <lsccip:displayName>registers</lsccip:displayName>
            <lsccip:description>Register Block</lsccip:description>
            <lsccip:baseAddress>0</lsccip:baseAddress>
            <lsccip:range>4096</lsccip:range>
            <lsccip:width>32</lsccip:width>
            <lsccip:usage>register</lsccip:usage>
            <lsccip:access>read-write</lsccip:access>
            <lsccip:register>
                <lsccip:name>Status</lsccip:name>
                <lsccip:displayName>Status Register</lsccip:displayName>
                <lsccip:description>Status Register</lsccip:description>
                <lsccip:addressOffset>0x10</lsccip:addressOffset>
                <lsccip:size>4</lsccip:size>
                <lsccip:volatile>true</lsccip:volatile>
                <lsccip:access>read-only</lsccip:access>
                <lsccip:field>
                    <lsccip:name>FIFO_Empty</lsccip:name>
                    <lsccip:displayName>FIFO_Empty</lsccip:displayName>
                    <lsccip:description>Indicates current status of the
interface in the receive direction:  0 - There is data available.  1 - The
FIFO is empty.</lsccip:description>
                    <lsccip:isPresent> 1 != int_setting
</lsccip:isPresent>
                    <lsccip:bitOffset>0</lsccip:bitOffset>
                    <lsccip:bitWidth>1</lsccip:bitWidth>
                    <lsccip:volatile>true</lsccip:volatile>
                    <lsccip:access>read-only</lsccip:access>
                    <lsccip:writeValueConstraint>
                        <lsccip:minimum>0</lsccip:minimum>
                        <lsccip:maximum>0</lsccip:maximum>
                    </lsccip:writeValueConstraint>
                    <lsccip:testable
testConstraint="unConstrained">false</lsccip:testable>
                </lsccip:field>
            </lsccip:register>
        </lsccip:addressBlock>
</lsccip:memoryMap>
</lsccip:memoryMaps>
```

Figure 3.36. Example of memoryMaps Node

7. <componentGenerators> node: contains a list of componentGenerator elements. Each componentGenerator element defines a generator that is run on generated IP instance package. Each generator is called after other IP instance package files are generated. For example, a component generator can generate memory initialization file for a memory IP (Figure 3.37).

```
<lsccip:componentGenerators>
  <lsccip:componentGenerator>
    <lsccip:name>memGenerator</lsccip:name>
    <lsccip:generatorExe>script/mem_gen.py</lsccip:generatorExe>
  </lsccip:Generator>
</lsccip:Generators>
```

Figure 3.37. Example of componentGenerator Node

8. <estimatedResources> node: contains a list of estimatedResource element. Each estimatedResource element defines the formula to calculate one type of resource used in the IP instance package. Table 3.6 shows the element in < estimatedResource> node.

Table 3.6. Elements in estimatedResource Node

Element	Value	Mandatory	Description
name	String	Yes	Name of the resource.
number	Python expression	Yes	Python script to calculate the number of the resource.

9. <outFileConfigs> node: specifies all customized output file configuration nodes < fileConfig > for whole customized flow. fileConfig node contains a group of attributes to specify a specific file or directory generation.

A full description of a soft IP might be large. Metadata.xml supports to build a large XML file from small manageable chunks. The approach is implemented by XInclude (Figure 3.38).

metadata.xml

```
<lsccip:ip version="1.0"
xmlns:lsccip="http://www.latticesemi.com/XMLSchema/Radiant/ip"
xmlns:xi="http://www.w3.org/2001/XInclude">
    <lsccip:general> … </lsccip:general>
<xi:include href="setting.xml" parse="xml" />
<xi:include href="memory_map.xml" parse="xml" />
<xi:include href="address_space.xml" parse="xml" />
<xi:include href="bus_interface.xml" parse="xml" />
<lsccip:ports> … </lsccip:ports>
</lsccip:ip>
```

setting.xml

```
<lsccip:settings
xmlns:lsccip="http://www.latticesemi.com/XMLSchema/Radiant/ip">
<lsccip:setting id="int_setting" type="input" value_type="int"
conn_mod="example" default="1"   title="Integer Setting" />
<lsccip:setting id="bool_with_value_mapping" type="param"
value_type="bool" conn_mod="example"
               default="False" bool_value_mapping="('Enabled',
'Disabled')" />
</lsccip:settings>
```

Figure 3.38. Example of XInclude Usage

You can use metadata.xsd in Appendix A to check the metadata file. For those elements borrowed from IPXact, refer to the related xsd files of IPXact.

3.3.2. Implementation RTL Files

You should put all IP implementation RTL files in 'rtl' sub directory of whole IP package. Lattice Builder platform does not support hierarchical rtl directories. All RTL sources can be put in one directory level. Three file suffix names ('.v', '.sv' and '.vhd') are supported for RTL files to represent 'Verilog', 'System Verilog' and 'Vhdl' language types accordingly.

IP Platform always generates a wrapper module by default for the whole IP generation. This wrapper is in Verilog HDL no matter what the language type of the IP implementation RTLs is. Providing a Verilog-HDL top to handle the interface translation among different language types is strongly recommended.

If the IP is implemented in Verilog HDL (including System Verilog) only, all the contents are copied in one .v/.sv RTL file and modules are defined in the wrapper top module. All module names have unified naming rule to avoid name collision such as Top module name, PMI module name, Primitive module name, Blackbox module name.

If some portions of an IP module in the RTL file are encrypted, the corresponding portions generated in Verilog file are also encrypted.

All the RTLs are kept and copied to an IP instance directory, if there is VHDL in IP implementation RTLs. Therefore, the module-naming rule does not need to be unified. You can specify different lib (default is work) for VHDL source that is specified in "wrapper" file generator (Figure 3.39).

```
<lsccip:outFileConfigs>
      <lsccip:fileConfig name="wrapper" lib='test2.vhd=libA;test3.vhd=libB' />
</lsccip:outFileConfigs>
```

Figure 3.39. Example of Specifying Lib for VHDL

3.3.3. Python Script Plugin File

Python expressions can be used in metadata file to implement complex logic. To support complex logic, you can add any python functions in plugin.py file (Figure 3.40) of an IP package, and then call the functions in Python expressions in metadata file.

Each setting item value can be referred to in a Python expression in metadata.xml by its ID as a Python variable. An expression is evaluated by demand. The plugin has its individual namespace and all the setting items values are exported to plugin as a global map variable IP_SETTINGS. You can use IP_SETTINGS [item ID] to refer to an item or set its value in plugin code.

A global variable '__PLUGIN_VER' is defined with value '2.0'. You can check this variable if you want to provide a plugin that can work on both current and legacy IP platforms.

```
def cntr_opt():
  #return {"Down"  :0,
  #         "Up"    :1,
  #         "UpDown":2}
  return [("Down"  , 0),
          ("Up"    , 1),
          ("UpDown", 2)]


def cntr_ldir():
  return ((CNTR_DIR == 0) | (CNTR_DIR == 1))


def cntr_wdt():
  if (CNTR_WIDTH < 2):
    return 2
  else:
    return CNTR_WIDTH


def cntr_hval_check():
  if (CNTR_HVALUE <= CNTR_LVALUE):
    ret = 0
    PluginUtil.post_error("Higher count value should be greater than the Lower count value.")
  else:
    ret = 1
  return ret


def get_device_name(value):
    x = runtime_info.device_info.architecture(value)
    return x
```

Figure 3.40. Template of Plugin File

3.3.4. Memory Map CSV File

Propel 2.0 Builder supports importing memory map from a CSV file to create memory map. You can edit a CSV file. Figure 3.41 shows the CSV file template. Each row to be imported is with one of the keywords among MEMORYMAP, REGISTER, and FIELD. Row with no keyword is ignored, such as the header row (orange squared part in Figure 3.41). The header row is to help identify the column.

	A	B	C	D	E	F	G	H	I	J
1	0	1	2	3	4	5		7	8	9
2		name	description	baseAddress	range	width				
3	MEMORYMAP	ABC	desc1	0x0	32	64				
4		name	displayName	description	addressOffset	size	volatile	access		
5	REGISTER	QSPI_CTRL	QSPI_CTRL	QSPI Control Register	0x00	32	TRUE	read-write		
6		field	displayName	description	bitOffset	bitWidth	volatile	access	default	testable
7	FIELD	spi_mode	spi_mode	Sets the SPI mode	0	2	TRUE	read-write	DEFAULT_SPI_MODE	TRUE
8	FIELD	sck_div	sck_div	Sets the SPI clock divider	2	3	TRUE	read-write	DEFAULT_SCK_DIV	TRUE
9	FIELD	reserved	reserved	Reserved bits	5	26	TRUE	read-write	0	TRUE
10	FIELD	soft_reset	soft_reset	Resets internal soft logic	31	1	TRUE	read-write	0	TRUE
11		name	displayName	description	addressOffset	size	volatile	access		
12	REGISTER	CMD_DATA	CMD_DATA	Command Data Register	0x04	32	TRUE	read-write		
13		field	displayName	description	bitOffset	bitWidth	volatile	access	default	testable
14	FIELD	cmd_data	cmd_data	Command data to transmit in transaction phase 1 (always big endian)	0	32	TRUE	read-write	0	TRUE
15		name	displayName	description	addressOffset	size	volatile	access		
16	REGISTER	TX_FIFO_DATA	TX_FIFO_DATA	Tx FIFO Data Register	0x08	32	TRUE	write-only		
17		name	displayName	description	bitOffset	bitWidth	volatile	access	default	testable
18	FIELD	tx_fifo_data	tx_fifo_data	Data to transmit in transaction phase 2.	0	32	TRUE	write-only	NA	FALSE

Figure 3.41. Example of Memory Map CSV File

4. TCL Commands

Propel Builder provides TCL commands to execute actions. You can manually enter TCL commands in Tcl Console (Figure 4.1), if you prefer using command lines rather than using the GUI.

Figure 4.1. Tcl Console

4.1. sbp_design

The sbp_design command is used as one of the high-level management commands such as opening, closing, of the design files created by the Propel Builder.

4.1.1. Open

Opens an existing Propel Builder design for modification.

Usage	sbp_design open -name <design name> -path <design path> [-device <device name>]
Example	sbp_design open -name project1 -path project1.sbx

4.1.2. Close

Closes a Propel Builder design currently opened.

Usage	sbp_design close

4.1.3. New

Creates a new Propel Builder design.

Usage	sbp_design new -name <new design name> -path <new design path>

4.1.4. Save

Saves the current Propel Builder design to file on disk, or save it as a new file. You can choose to save the design to the project location (Example 1) or to a specific path (Example 2).

Usage	sbp_design save [-path <new design path>]
Example 1	sbp_design save
Example 2	sbp_design save -path new_design.sbx

4.1.5. Drc

Runs the design rule check for the Propel Builder design file.

Usage	sbp_design drc

4.1.6. Generate

Generates RTL code to instantiate and connect the IP cores specified in the Propel Builder design file.

Usage	sbp_design generate

4.1.7. auto_assign_addresses

Automatically assigns memory mapped addresses to all the slaves in the system. These addresses should be chosen to avoid slaves with multiple non-contiguous address ranges.

Usage	sbp_design auto_assign_addresses

4.1.8. Launch SoC Verification Engine

Launches the verification engine to verify design.

Usage	sbp_design verify [-h] [--workfolder <workfolder_path>] [--sbx_file <sbx_file_path>] {sim_gen,pfr_pack,regmap_gen,auto_run} positional arguments: {sim_gen,pfr_pack,regmap_gen,auto_run} specify the application sim_gen Simulation Generation Flow pfr_pack PFR Security Engine Packager Flow regmap_gen Memory Map Generation Flow auto_run Automation Flow optional arguments: -h, --help show this help message and exit --workfolder WORKFOLDER, -wf WORKFOLDER assign the working folder. Otherwise, the default one (current working folder) will be used --sbx_file SBX_FILE, -sf SBX_FILE specify the sbx file
Example	sbp_design verify --workfolder ./mem_map --sbx_file D:/ /XO3D_Initial/XO3D_Initial.sbx regmap_gen

4.1.9. Execute PGE Engine

Runs command with different parameters to call SGE function, DGE function, and Signature.

- Runs command to call SGE to generate files for SDK.

Usage	sbp_design *pge sge –i <input path> [-o <output path>]* -i [Required] the top-level SoC project sbx file full path. -o [Optional] output full path, if the parameter is not specified, output path is the same as SoC project path. sge folder is generated at output path, at present sge folder under the SoC project root directory.

- Runs command to call DGE to generate TCL script that can be used to generate a Diamond or Radiant project.

Usage	sbp_design *pge dge –i <input path> [-o <output path>] [-diamond|-radiant]* -i [Required] the top level SoC project sbx file full path. -o [Optional] output full path, if the parameter is not specified, output path is the same as SoC project path. For DGE, tcl script and related files are generated at output path. At present, Diamond or Radiant project share the same workspace with SoC project. -diamond [Optional] flag to execute DGE for diamond project. -radiant [Optional] flag to execute DGE for Radiant project.

- Run command to generate signature.

  PGE gets partial UFM3 contents including version packet (C), Sentry PFR configure data (D), and call Flash Address Tool to sign with private key from Factory HSM.

Usage	sbp_design *pge* gen_signature -cfile <c binary file> -dfile <d binary file> -output <output binary file>

4.1.10. Undo

Undo operation. Currently, software only supports single undo.

Usage	undo

4.1.11. Redo

Redo operation. Currently, software only supports single redo.

Usage	redo

4.2. Other TCL Commands

The following commands are used for specifying connectivity and IP instantiation to the Propel Builder backend.

4.2.1. sbp_add_component

Instantiates an IP component into the system. Must specify the component VLNV identifier. This corresponds to a component instance in the IP-XACT design. For example, the command below can instantiate an AHB-Lite interconnect component.

Usage	sbp_add_component -vlnv <VLNV> -name <instance_name>
Example	sbp_add_component -vlnv lattice:ip:ahbl_interconnect:1.0 -name ahblite_interconnect

4.2.2. sbp_add_sbx_component

Instantiates a hierarchical component from sbx file into the system.

Usage	sbp_add_sbxcomp -name <hname> -path <sbx_file_absolute_path_name>
Example	sbp_add_sbxcomp -name sim_comp -path “C:/test/test.sbx”

4.2.3. sbp_add_gluelogic

Instantiates an gluelogic component into the system.

Usage	sbp_add_gluelogic –name <instance_name> -logicinfo <logic json info from sbp_create_glue_logic>
Example	sbp_add_gluelogic -name equation_module_inst -logicinfo [sbp_create_glue_logic equation equation_module {} { {"expr":"!A & !B","module_name":"equation_module"}}]

4.2.4. sbp_create_glue_logic

Create gluelogic information when adding gluelogic component. rtl_path must be a file path if gluelogic comes from a file. Else be empty.

Usage	sbp_create_glue_logic <type> <module_name> <rtl_path> <json cfg data>
Example	sbp_add_gluelogic -name equation_module_inst -logicinfo [sbp_create_glue_logic equation equation_module {} { {"expr":"!A & !B","module_name":"equation_module"}}]

4.2.5. sbp_add port

Creates a top-level I/O port. Must specify the direction.

Usage	sbp_add_port [-from <bit number>] [-to <bit number>] -direction <in/out/inout> <port_name>

4.2.6. sbp_connect_net

Connects all of the specified pins and/or ports to the same net. The arguments can be pins or ports in the system design. Only one of the arguments can be the driver (output pin/port), driving all other input pin/ports. Example below connects the clk port to all components, assuming component pins are all named clk.

Usage	sbp_connect_net [-name <net name>] <pin/port> <pin/port>
Example	sbp_connect_net [sbp_get_pins clk] {CLOCK_IN}

4.2.7. sbp_connect_interface_net

Connects a bus interface pin/port to another interface pin/port. This corresponds to the interconnection element in the IP-XACT design.

Usage	sbp_connect_interface_net <pin/port> <pin/port>

4.2.8. sbp_connect_constant

Connects a constant integer to a pin/pinbus/port/portbus. To assign to a pin/pinbus, the object must be an input pin/pinbus. To assign to a port/portbus, the object must be an output port/portbus. If the integer requires multiple bits, not 0 or 1, then the object must be a bus. The tcl command can be used to assign the same constant to multiple pin/pinbus/port/portbus at the same time.

Usage	sbp_connect_constant -constant <integer> <pin/pin bus/ port/portbus> <pin/pin bus/ port/portbus>…
Example	sbp_connect_constant -constant 1 {test/i2c_mst_apb/rst_n_i} {test/riscv/clk_i}

4.2.9. sbp_connect_whitebox

User can do connection cross the hierarchy boundary for verification purpose.

Usage	sbp_connect_whitebox <design_name/vip_inst_name/pin_name> <design_name/uit_inst_name/port_name>

4.2.10. disconnect whitebox connection

Removes the connection cross the hierarchy boundary.

Usage	sbp_disconnect_whitebox <design_name/vip_inst_name/pin_name> <design_name/uit_inst_name/port_name>

4.2.11. sbp_disconnect_interface_net

Disconnects an interface pin/port from the interface nets they attached to. Note that any interface pin or interface port can attach to one interface net at the most.

Usage	sbp_disconnect_interface_net <pin/port> <pin/port>

4.2.12. sbp_disconnect_net

Disconnects all of the specified input pins and/or ports from the nets they attached to. Note that any pin or port can attach to one net at the most. Can also be used to disconnect a constant that is connected to a pin/port with connect_constant.

Usage	sbp_disconnect_net <pin0> <pin1> <port2>

4.2.13. sbp_assign_addr_seg

Assigns a memory map between a pair of master and slave interfaces. Range specifies the range of the segment, for example, 32‘h0000400, 32‘h0001000. Offset specifies the base offset of the range, for example, 32‘h0000400

Usage	sbp_assign_addr_seg -offset <offset> <slave connection name>
Example	sbp_assign_addr_seg -offset 32‘h00001000 simple/riscv/AHBL_S00

4.2.14. sbp_unassign_addr_seg

The Tcl command unsets the fixed offset flag for a memory map allowing the auto_assign Tcl command to assign the memory map offset.

Usage	sbp_unassign_addr_seg <slave interface name>
Example	sbp_unassign_addr_seg simple/spi/AHB_S00

4.2.15. sbp_assign_local_memory

Assigns a base address to a local memory map of a master address space.

Usage	sbp_assign_local_memory -offset <offset> <master_addr_space>
Example	sbp_assign_local_memory -offset ‘h0050000 Foundation_SoC/riscv/ahbl_m_data_Address_Space

4.2.16. sbp_export_pins

Exports a list of pins or interface pins, or all not-yet-connected pins of the components to the top-level port list in the design. The function detects whether or not the argument is pin(s) or component(s). In the two examples below, Example 1 demonstrates a Tcl command to export the pin init_done, while Example 2 demonstrates a Tcl command to export all pins and interfaces of the ddr3 component.

Usage	sbp_export_pins <pin/component> <pin/component>
Example 1	sbp_export_pins {ddr3/init_done}
Example 2	sbp_export_pins {ddr3}

4.2.17. sbp_export_interface

Exports bus interfaces that are passed as arguments to the Tcl command from the component to the top-level component. Example below exports the AHBL_MASTER bus interface of the RISC-V component to the top-level component.

Usage	sbp_export_interfaces <interface> <interface> <interface>
Example	sbp_export_interfaces simple/riscv/AHBL_MASTER

4.2.18. sbp_rename

The rename Tcl command renames objects with in the design. The new name of the object and the current hierarchical name of the given object are used as the parameter value. The object can be an interface connection, connection, port, interface, or component. The example below demonstrates the changing of the name of a port in milestone project from CLK to CLOCK.

Usage	sbp_rename -name <new name> <object name>
Example	sbp_rename -name CLOCK milestone/CLK

4.2.19. sbp_replace

The Replace (Re-Config) Tcl command replaces a component with a new configuration for itself. VLNV refers to the newly-generated IP, component name refers to the existing component that is to be replaced, and instance refers to the instance name of the component with new configuration.

Usage	sbp_replace -vlnv <VLNV> -name <instance> -component <component name>
Example	sbp_replace -vlnv lattice:ip:ahblite_bus_0:1.1 -name ahbl_bus_0 -component simple/ahblite

4.2.20. sbp_copy

Copies objects, IP instances and nets, from the current or other open sbp designs to the current sbp design. All objects are post-fixed with a postfix string. For example, you can write TCL code below to duplicate the components and connections with new instance names post fixed with X, by calling copy on all the components, ports and nets. Pins are automatically duplicated while the components are duplicated.

Usage	sbp_copy -postfix <postfixString> objects
Example	sbp_copy –postfix X $selected_objs

4.2.21. sbp_delete

Deletes objects, IP instances and nets, from the current sbp design. In the examples below, Example 1 demonstrates a Tcl command to delete a port, while Example 2 demonstrates a Tcl command to delete ddr3 component.

Usage	Sbp_delete objects -type <type name>
Example 1	sbp_delete [sbp_get_ports <clock>] -type port
Example 2	sbp_delete {ddr3} -type component

4.2.22. sbp_get_pins

Gets a list of pin names that match a pattern string, and/or the pins that are associated with an object. The object in the [-from <objectName>] option can be a net or a component. The example below gets the clk pin from the interconnect IP.

Usage	sbp_get_pins [-from <object Name>] [pattern]
Example	sbp_get_pins -from ahblite_interconnect clk

4.2.23. sbp_get_interface_pins

Gets a list of interface names that match a pattern string, and/or the interfaces that are associated with an object. The object in the [-from <objectName>] option can be an interface net or a component. The example below can get all AHB-Lite slave interface pins from the interconnect IP.

Usage:	sbp_get_interface_pins [-from <objectName>] [pattern]
Example:	sbp_get_interface_pins -from ahblite_interconnect S*_AHB

4.2.24. sbp_get_ports

Get a list of the names of ports that match a pattern string, and/or the ports that are associated with an object. The object in the [-from <objectName>] option can be a net.

Usage	sbp_get_ports [-from <objectName>] [pattern]

4.2.25. sbp_get_interface_ports

Gets a list of interface names that match a pattern string, and/or the interface ports that are associated with an object. The object in the [-from <objectName>] option can be an interface net.

Usage	sbp_get_interface_ports [-from <objectName>] [pattern]

4.2.26. sbp_get_nets

Gets a list of net names that match a pattern string, and/or the nets that are associated with an object. The object in the [-from <objectName>] option can be a pin or a port.

Usage	sbp_get_nets [-from <objectName>] [pattern]

4.2.27. sbp_get_interface_nets

Gets a list of interface net names that match a pattern string, and/or the interface nets that are associated with an object. The object in the [-from <objectName>] option can be an interface pin or an interface port.

Usage	sbp_get_interface_nets [-from <objectName>] [pattern]

4.2.28. sbp_set_property

Sets the properties of an input object. The first argument is a list of name value pairs. We have a list of parameters because the GUI IP configuration dialog submits changes to many parameters of an IP component at the same time when the dialog is closed. In the examples below, Example 1 changes the data width of the RAM block named “ebr_0”, while Example 2 changes the number of master interfaces to two and number of slave interface to three on AHB-Lite interconnect block named “ahbl_interconnect”.

Usage	sbp_set_property <name0 value0 name1 value1 …> object
Example 1	sbp_set_property {datawidth 32} {test/ebr_0}
Example 2	sbp_set_property {NUM_MI 2 NUM_SI 3} test/ahbl_interconnect

4.2.29. sbp_get_property

Gets the property of the object. The example below is to get the number of slave interfaces of AHB-Lite interconnect block named "ahbl_interconnect_0".

Usage	sbp_get_property <parameter name> <object>
Example	sbp_get_property NUM_SI ahbl_interconnect_0

4.2.30. sbp_report_properties

Prints all the properties and values associated to the type of the object.

Usage	sbp_report_properties object
Example	sbp_report_properties ahbl_interconnect
Outputs	Name: ahbl_interconnect NUM_SI: 1 NUM_MI: 2 DATA_WIDTH: 32 ADDRESS_WIDTH: 32

4.2.1. sbp_get_components

Gets a list of component names that match a pattern string, and/or the components that are associated with an object. The default pattern string is a wildcard "*" that matches all components. The pattern string may consist of string segments and wildcards. The example below returns all the component names that contain "interconnect". The command returns an empty string if no match is found.

Usage	sbp_get_components <component name>
Example	sbp_get_components {*interconnect*}

Appendix A. metadata.xsd

```
<?xml version="1.0" encoding="gb2312"?>
<!-- IP Metadata Schema 0.0.4 -->
<!--
Change Log:
    0.0.0  21-Jul-2016 initial revision.
    0.0.1  22-Jul-2016 1. removed enum_value from <setting> 2. change
value_mapping to scriptType
    0.0.2  29-Jul-2016 1. removed GUI out of metadata scope
    0.0.3  30-Jul-2016 1. add value list support
    0.0.4  12-Dec-2020 1. all new features for radiant 3.0.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.latticesemi.com/XMLSchema/Radiant/ip"
           xmlns:lsccip="http://www.latticesemi.com/XMLSchema/Radiant/ip"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
  <xs:include schemaLocation="busInterface.xsd"/>
  <xs:include schemaLocation="memoryMap.xsd"/>
  <xs:include schemaLocation="file.xsd"/>
  <xs:include schemaLocation="generator.xsd"/>
  <xs:include schemaLocation="design.xsd"/>

  <xs:attributeGroup name="ipany.att">
    <xs:anyAttribute processContents="lax"/>
  </xs:attributeGroup>
  <!-- version string type definition-->
  <xs:simpleType name="threeFigureVersionType">
    <xs:annotation>
      <xs:documentation>
        The syntax for version string.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+\.[0-9]+\.[0-9]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="twoFigureVersionType">
    <xs:annotation>
      <xs:documentation>
        The syntax for Radiant version string.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+\.[0-9]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="scriptType">
    <xs:annotation>
      <xs:documentation>
```

```
        Any python expressions. Could be a simple const variable like
&quot;1&quot;,
        or a tuple &quot;(0, 10)&quot;, or a function call
&quot;user_func1()&quot;,
        or a complex expression &quot;a==12&quot;
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="settingTypeType">
    <xs:annotation>
      <xs:documentation>
       Is this setting variable for parameter of IP core module, or just for
user input?
       Valid values are &quot;input&quot; and &quot;param&quot;.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="param"/>
      <xs:enumeration value="input"/>
      <xs:enumeration value="command"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="settingValueType">
    <xs:annotation>
      <xs:documentation>
        Value type of setting variable. Valid types are &quot;bool&quot;,
&quot;string&quot; and &quot;int&quot;
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="bool"/>
      <xs:enumeration value="string"/>
      <xs:enumeration value="int"/>
      <xs:enumeration value="float"/>
      <xs:enumeration value="path"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="settingDefaultType">
    <xs:annotation>
      <xs:documentation>
        Default value of setting variable.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="settingOutputFormatterType">
    <xs:annotation>
      <xs:documentation>
        Formatter of output.
```

```
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="str"/>
      <xs:enumeration value="nostr"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- ip.settings.setting-->
  <xs:complexType name="settingElementType">
      <xs:annotation>
            <xs:documentation>
                  Setting variable definition.
            </xs:documentation>
      </xs:annotation>

      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="type" type="lsccip:settingTypeType"
            use="required" />
      <xs:attribute name="value_type" type="lsccip:settingValueType"
            use="required" />
      <xs:attribute name="conn_mod" type="xs:string" use="required" />
      <xs:attribute name="default" type="lsccip:settingDefaultType"
            use="optional" />
      <xs:attribute name="value_expr" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="drc" type="lsccip:scriptType" use="optional" />
      <xs:attribute name="editable" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="description" type="xs:string" use="optional" />
      <xs:attribute name="title" type="xs:string" use="optional" />
      <xs:attribute name="hidden" type="xs:string" use="optional" />
      <xs:attribute name="regex" type="xs:string" use="optional" />
      <xs:attribute name="options" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="value_range" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="group1" type="xs:string" use="optional" />
      <xs:attribute name="group2" type="xs:string" use="optional" />
      <xs:attribute name="bool_value_mapping" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="output_formatter"
            type="lsccip:settingOutputFormatterType" use="optional" />
      <xs:attribute name="config_groups" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="process_path" type="xs:boolean" use="optional"
default="true"></xs:attribute>
  </xs:complexType>

  <!-- ip.settings -->
  <xs:complexType name="settingsType">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="setting"
type="lsccip:settingElementType" />
```

```
    </xs:sequence>
  </xs:complexType>

  <!-- ip.ports.port-->
  <xs:complexType name="portElementType">
      <xs:annotation>
            <xs:documentation>IP port definition.</xs:documentation>
      </xs:annotation>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="dir" type="xs:string" use="required" />
      <xs:attribute name="conn_mod" type="xs:string" use="required" />
      <xs:attribute name="conn_port" type="xs:string" use="optional" />
      <xs:attribute name="conn_range" type="xs:string" use="optional" />
      <xs:attribute name="range" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="stick_high" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="stick_low" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="stick_value" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="dangling" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="bus_interface" type="xs:string"
            use="optional" />
      <xs:attribute name="attribute" type="lsccip:scriptType"
            use="optional" />
      <xs:attribute name="port_type" use="optional" default="data">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="data"></xs:enumeration>
                        <xs:enumeration value="reset"></xs:enumeration>
                        <xs:enumeration value="clock"></xs:enumeration>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
  </xs:complexType>
  <!-- ip.outFileConfigs -->
  <xs:complexType name="outFileConfigsType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="fileConfig"
type="lsccip:fileConfigType" />
    </xs:sequence>
  </xs:complexType>
  <!-- ip.outFileConfigs.fileConfig -->
  <xs:complexType name="fileConfigType">
      <xs:annotation>
            <xs:documentation>Configure output file</xs:documentation>
      </xs:annotation>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="use" use="optional" default="merge">
            <xs:annotation>
                  <xs:documentation>
                        Merge the new attributes to existing configuration or
```

```
                        replace all
                  </xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="merge"></xs:enumeration>
                        <xs:enumeration value="replace"></xs:enumeration>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="type" use="optional" default="file">
            <xs:annotation>
                  <xs:documentation>
                        Indicate the item is generate a file or directory
                  </xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="file"></xs:enumeration>
                        <xs:enumeration value="directory"></xs:enumeration>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="description" type="xs:string"
            use="optional">
      </xs:attribute>
      <xs:attribute name="enable_output" type="xs:string" use="optional"
            default="True">
            <xs:annotation>
                  <xs:documentation>
                        Python expression represent a boolean value to
                        indicate the output is enabled or disabled
                  </xs:documentation>
            </xs:annotation>
      </xs:attribute>
      <xs:attribute name="phase" type="xs:int" use="optional"
            default="0">
      </xs:attribute>
      <xs:attribute name="file_base_name" type="xs:string"
            use="optional">
      </xs:attribute>
      <xs:attribute name="file_suffix" type="xs:string"
            use="optional">
      </xs:attribute>
      <xs:attribute name="sub_dir" type="xs:string"
use="optional"></xs:attribute>
      <xs:attribute name="file_generator" type="xs:string"
            use="optional">
      </xs:attribute>
      <xs:attribute name="src_dir" type="xs:string"
          use="optional">
      </xs:attribute>
      <xs:attribute name="dest_dir" type="xs:string"
        use="optional">
```

```
    </xs:attribute>
    <xs:attribute name="src_file" type="xs:string"
        use="optional">
    </xs:attribute>
    <xs:attribute name="export" use="optional" default="input_only">
        <xs:annotation>
            <xs:documentation>
                Indicate export all setting items or input only
            </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="all"></xs:enumeration>
                <xs:enumeration value="input_only"></xs:enumeration>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
      <xs:attributeGroup ref="lsccip:ipany.att"/>
  </xs:complexType>
  <!-- ip.interface-->
  <xs:complexType name="portsType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="port"
type="lsccip:portElementType" />
    </xs:sequence>
  </xs:complexType>

  <!-- ip.interface-->
  <xs:complexType name="generalType">
    <xs:sequence>
      <xs:element maxOccurs="1" minOccurs="0" name="vendor"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="0" name="library"
            type="xs:Name" />
      <xs:element maxOccurs="1" minOccurs="1" name="name"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="0" name="display_name"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="1" name="version"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="1" name="category"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="0" name="keywords"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="0" name="type"
            type="xs:string" />
      <xs:element maxOccurs="1" minOccurs="0" name="instantiatedOnce"
            type="xs:boolean" />
      <xs:element maxOccurs="1" minOccurs="1"
            name="min_radiant_version" type="lsccip:twoFigureVersionType" />
      <xs:element maxOccurs="1" minOccurs="0"
            name="max_radiant_version" type="lsccip:twoFigureVersionType" />
      <xs:element maxOccurs="1" minOccurs="0" name="min_esi_version"
            type="lsccip:twoFigureVersionType" />
```

```
      <xs:element maxOccurs="1" minOccurs="0" name="max_esi_version"
            type="lsccip:twoFigureVersionType" />
      <xs:element maxOccurs="1" minOccurs="1"
            name="supported_products" type="lsccip:supportedProductsType" />
      <xs:element maxOccurs="1" minOccurs="0"
            name="supported_platforms" type="lsccip:supportedPlatformsType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="supportedPlatformsType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="supported_platform"
type="lsccip:supportedPlatformType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="supportedPlatformType">
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="supportedProductsType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="supported_family"
type="lsccip:supportedFamilyType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="supportedFamilyType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="supported_device"
type="lsccip:supportedDeviceType" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="supportedDeviceType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0"
name="supported_speed_grade" type="lsccip:supportedSpeedType" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="supportedSpeedType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="supported_package"
type="lsccip:supportedPackageType" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="supportedPackageType">
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
```

```
  <!-- ip.estimatedResources -->
  <xs:complexType name="estimatedResourcesType">
    <xs:annotation>
      <xs:documentation>
          IP estimated resources definition.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="estimatedResource" type="lsccip:estimatedResourceType"
minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="estimatedResourceType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="number" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <!-- ip -->
  <xs:element name="ip">
    <xs:complexType>
      <xs:sequence>
            <xs:element name="general" type="lsccip:generalType" />
            <xs:element name="settings" type="lsccip:settingsType" />
            <xs:element name="ports" type="lsccip:portsType" />
            <xs:element name="outFileConfigs" type="lsccip:outFileConfigsType"
maxOccurs="1" minOccurs="0">
          <xs:unique name="fileCfgKey">
            <xs:selector xpath="lsccip:fileConfig"/>
            <xs:field xpath="@name"/>
          </xs:unique>
            </xs:element>
            <xs:element ref="lsccip:busInterfaces" minOccurs="0" />
            <xs:element ref="lsccip:addressSpaces" minOccurs="0" />
            <xs:element ref="lsccip:memoryMaps" minOccurs="0" />
            <xs:element ref="lsccip:componentGenerators" minOccurs="0" />
            <xs:element ref="lsccip:choices" minOccurs="0" />
            <xs:element ref="lsccip:fileSets" minOccurs="0" />
            <xs:element ref="lsccip:design" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="estimatedResources"
                  type="lsccip:estimatedResourcesType" minOccurs="0" />
            <xs:element ref="lsccip:parameters" minOccurs="0" />
      </xs:sequence>
      <xs:attribute name="version" type="lsccip:twoFigureVersionType"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

References

- Lattice Propel 2.0 SDK Usage Guide (FPGA-UG-02127)
- Lattice Propel 2.0 Installation for Windows Usage Guide (FPGA-AN-02036)
- Lattice Diamond Online Help
- Lattice Radiant Online Help

Technical Support Assistance

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

Revision History

Revision 1.0, April 2021

Section	Change Summary
All	Initial release.

www.latticesemi.com