



Lattice IP Packager 2023.1

User Guide

FPGA-UG-02187-1.0

May 2023

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications, or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property, or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Glossary	5
1. Introduction	6
1.1. Purpose	6
1.2. Audience	6
2. IP Packager	7
2.1. Launching IP Packager	8
2.2. Packing Custom IP Flow	9
2.2.1. Opening an IP Directory	9
2.2.2. Editing IP	11
2.2.3. Previewing IP	35
2.2.4. Packaging IP	36
2.3. Editing IP Package Files	36
2.3.1. Metadata File	36
2.3.2. Implementation RTL Files	45
2.3.3. Python Script Plugin File	45
2.3.4. Memory Map CSV File	47
Appendix A. metadata.xsd	48
References	58
Technical Support Assistance	59
Revision History	60

Figures

Figure 2.1. Example Directories and Files of an IP Package	7
Figure 2.2. Example Directories and Files of an IP Instance Package	7
Figure 2.3 Launch IP Packager from Lattice Propel Builder	8
Figure 2.4. IP Packager GUI	8
Figure 2.5. Select Folder Dialog	9
Figure 2.6. IP Packager with IP Project Details	10
Figure 2.7. IP Packager with Meta Data Details	11
Figure 2.8. Meta Data View	12
Figure 2.9. Configure Basic Info	12
Figure 2.10. Warning Message of Invalid Radiant and Propel Version	13
Figure 2.11. Preview on Parameters	14
Figure 2.12. Two-Level Hierarchy to Categorize Parameters	15
Figure 2.13. IP Preview Showing Two-level Hierarchy in Parameters	16
Figure 2.14. Right-click Menu of Parameters in the Meta Data View	17
Figure 2.15. Group1 Added to Tab1	17
Figure 2.16. NewParam1 Added to Group1	18
Figure 2.17. Rename a Parameter	18
Figure 2.18. Enter Desired Parameter Value	19
Figure 2.19. Select Desired Parameter Value from Drop-down Menu	19
Figure 2.20. Three Port Types	21
Figure 2.21. Right-click Menu of the IN Port	22
Figure 2.22. Add inferred ports to meta data Wizard	23
Figure 2.23. Rename an IN Port	23
Figure 2.24. Configure an IN Port	24
Figure 2.25. Right-click Menu of an Interface	25
Figure 2.26. Rename Interface	25
Figure 2.27. Configure an Interface	26
Figure 2.28. Right-click Menu of a Memory Map	27

Figure 2.29. Generate New Memory Map	27
Figure 2.30. IP Packager with Design File Details	28
Figure 2.31. Set IP Rtl Library Path in Lattice Propel Builder	29
Figure 2.32. Set IP RTL Library Path in Lattice Radiant Software	30
Figure 2.33. IP Packager with Test Bench Details	31
Figure 2.34. IP Packager with Misc Details	32
Figure 2.35. IP Packager with Doc Assistant Details	33
Figure 2.36. Configure Doc Assistant	34
Figure 2.37. IP Preview Shows Configuration for IP Module	35
Figure 2.38. Error Message Pops Up in IP Packager	35
Figure 2.39. IP Packager Pops up Successful Packaging Message	36
Figure 2.40. Example XML of Metadata Layout	36
Figure 2.41. Example of busInterface Node	41
Figure 2.42. Example of addressSpaces Node	42
Figure 2.43. Example of memoryMaps Node	43
Figure 2.44. Example of componentGenerators Node	44
Figure 2.45. Example of Xinclude Usage	44
Figure 2.46. Example of Specifying Lib for VHDL	45
Figure 2.47. Template of Plugin File	46
Figure 2.48. Example of Memory Map CSV File	47

Tables

Table 2.1. Details of Parameter Property	20
Table 2.2. Details of Port Property	24
Table 2.3. Child Nodes of General Node	37
Table 2.4. Attributes of Setting Nodes	37
Table 2.5. Attributes of Port Nodes	40
Table 2.6. Elements in estimatedResources Node	44

Glossary

A list of words or terms specialized in this document.

Glossary	Definition
CPU	Central Processing Unit
CSV	Comma Separated Values file
DRC	Design Rule Check
ESI	Previous name of Propel
FPGA	Field Programmable Gate Array
GUI	Graphic User Interface
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
IP-XACT	An XML format that defines and describes electronic components and their designs.
SoC	System on Chip. An integrated circuit that integrates all components of a computer or other electronic systems.
TCL	Tool Command Language

1. Introduction

An IP package is a collection of all required files related to an IP core. These related files of an IP are organized in different directories. The graphical tool Lattice IP Packager 2023.1 is used to create an IP package easily for Lattice Radiant™ software and Lattice Propel™ Builder.

1.1. Purpose

The purpose of this document is to introduce Lattice IP Packager 2023.1 to help you quickly get started to create a package for a customized IP module. A variety of IP modules that can be customized are available for you in Lattice Radiant Software and Lattice Propel Builder. These modules cover a variety of common functions, and can assist your design work.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice MachXO2™, MachXO3D™, MachXO3L™, MachXO3LF™, CrossLink™-NX, Certus™-NX, CertusPro™-NX, Mach™-NX, MachXO5™-NX, and Lattice Avant™ devices. The technical guidelines assume readers have expertise in the embedded system area and FPGA technologies.

2. IP Packager

IP Packager helps you create an IP package easily. You can edit port, file, parameter, and memory in the IP Packager, and pack a customized IP directly. [Figure 2.1](#) shows an example of directories and files of an IP package.

```

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

Figure 2.1. 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 documents.

The custom IP package can be used in Propel Builder for SoC design. IP instance package ([Figure 2.2](#)) is generated when you configure an 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 2.2. Example Directories and Files of an IP Instance Package

2.1. Launching IP Packager

IP Packager can be launched from Lattice Propel Builder or Lattice Radiant Software.

To launch IP Packager from Lattice Propel Builder:

1. Choose **Tools** >  **IP Packager** from the Propel Builder Menu Bar, as shown in [Figure 2.3](#).

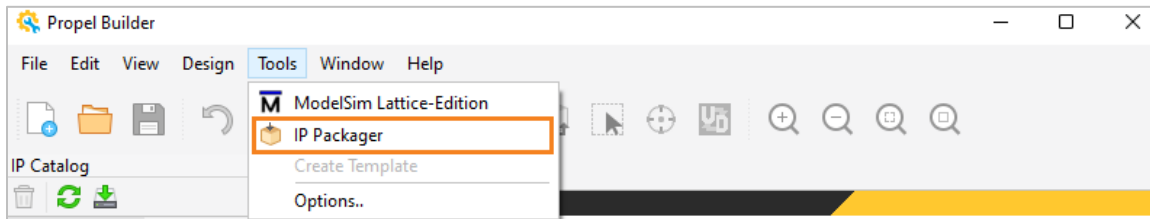


Figure 2.3 Launch IP Packager from Lattice Propel Builder

2. The IP Packager window pops up, as shown in [Figure 2.4](#).

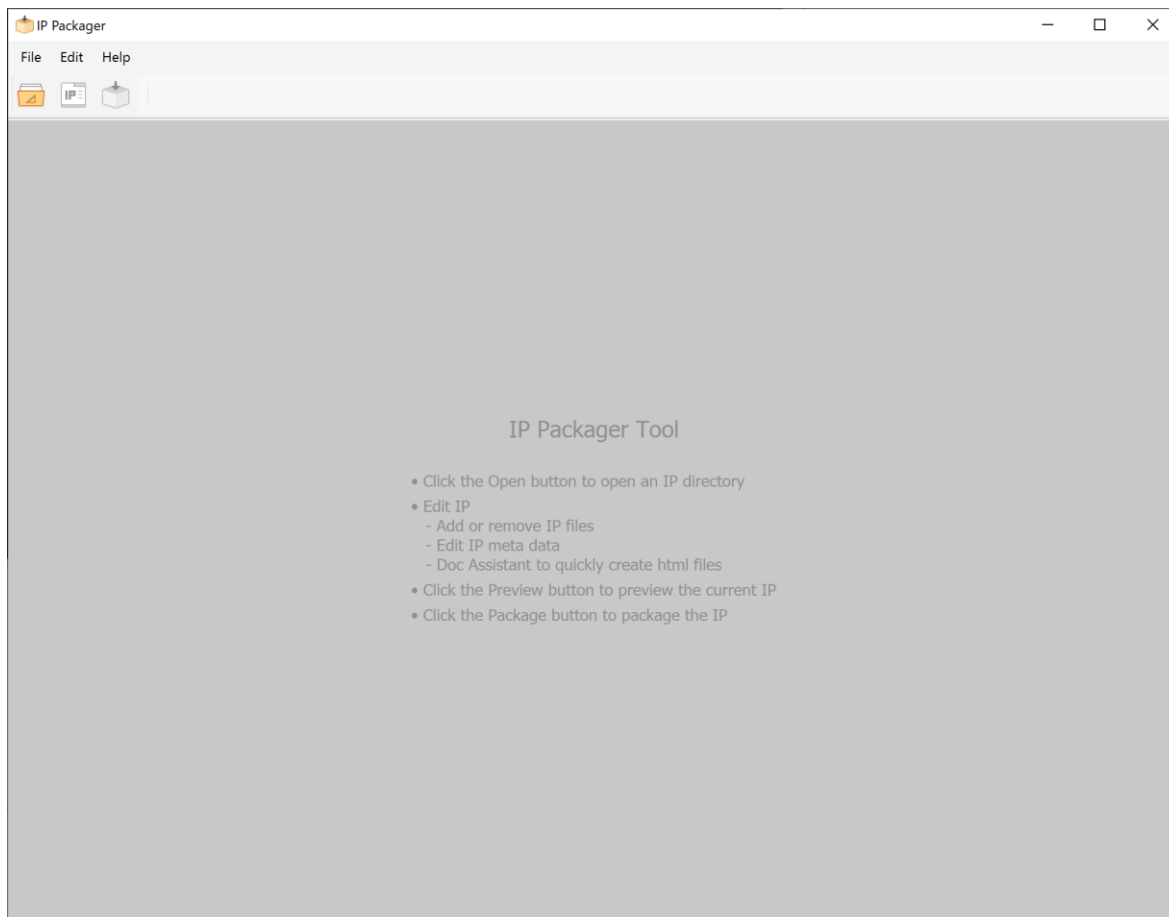




Figure 2.4. IP Packager GUI

To launch IP Packager from Lattice Radiant Software:

1. Go to the Windows Start menu and choose **Programs** > **Lattice Radiant Software** > **Accessories** > **IP Packager**.
2. The **IP Packager** window pops out ([Figure 2.4](#)).

2.2. Packing Custom IP Flow

2.2.1. Opening an IP Directory

1. In Lattice IP Packager, choose **File** >  **Open IP Directory** from the Menu or Click the **Open Design** icon  from the Toolbar. The Select Folder dialog opens (Figure 2.5).

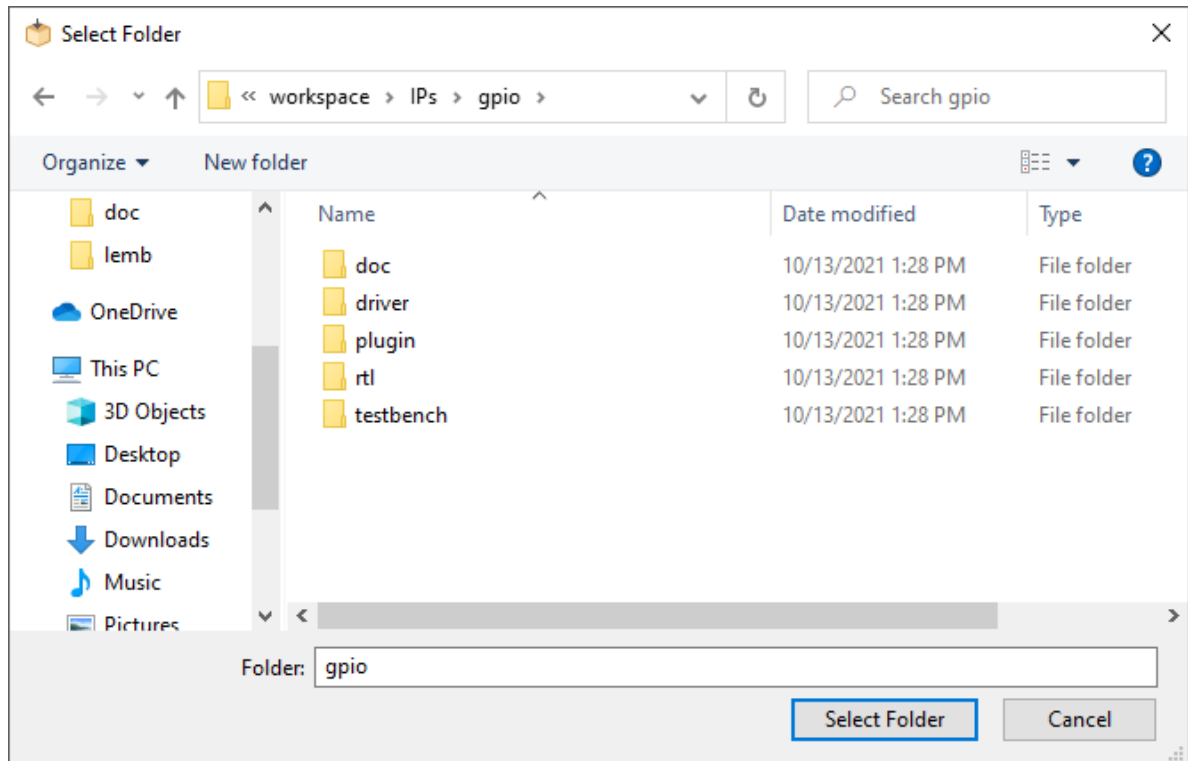


Figure 2.5. Select Folder Dialog

2. Choose a desired IP directory.
3. Click **Select Folder**. The IP Packager GUI shows the IP Package project (Figure 2.6). An IP package project may include the following parts.

The three listed below are mandatory:

- Meta Data
- Design File
- Doc

The two listed below are optional:

- Test Bench
- Misc

These mandatory and optional parts need to be edited. Refer to the [Editing IP](#) section for more details.

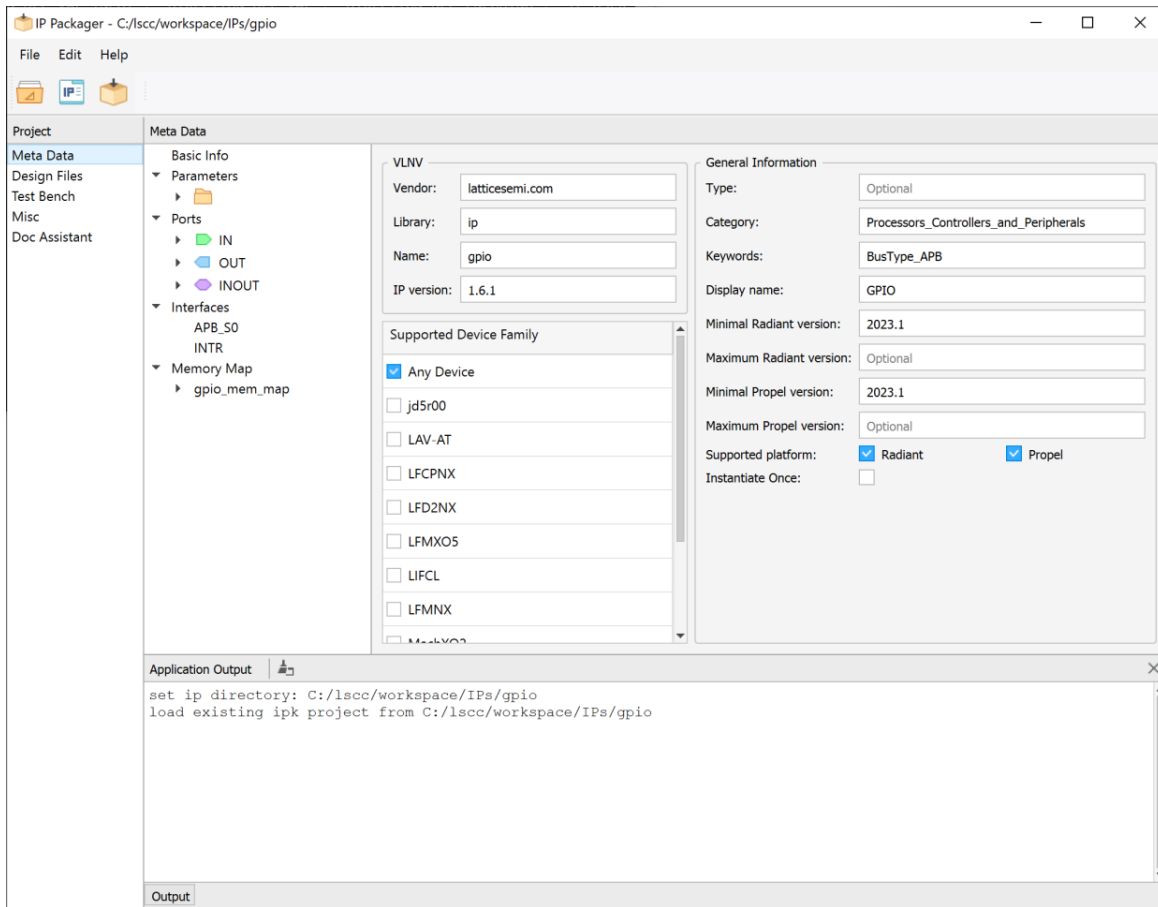


Figure 2.6. IP Packager with IP Project Details

Note: If you prepare all directories and files of an IP module in advance, the IP Packager can load the information. You can use IP Packager to open an empty folder for creating a new IP, and add your own RTL files into this empty IP. If you do not want to update the IP module information, you can skip the [Editing IP](#) section below.

2.2.2. Editing IP

- From the IP Packager Project area, click **Meta Data**. The IP Packager GUI shows the Meta Data information in detail (Figure 2.7).

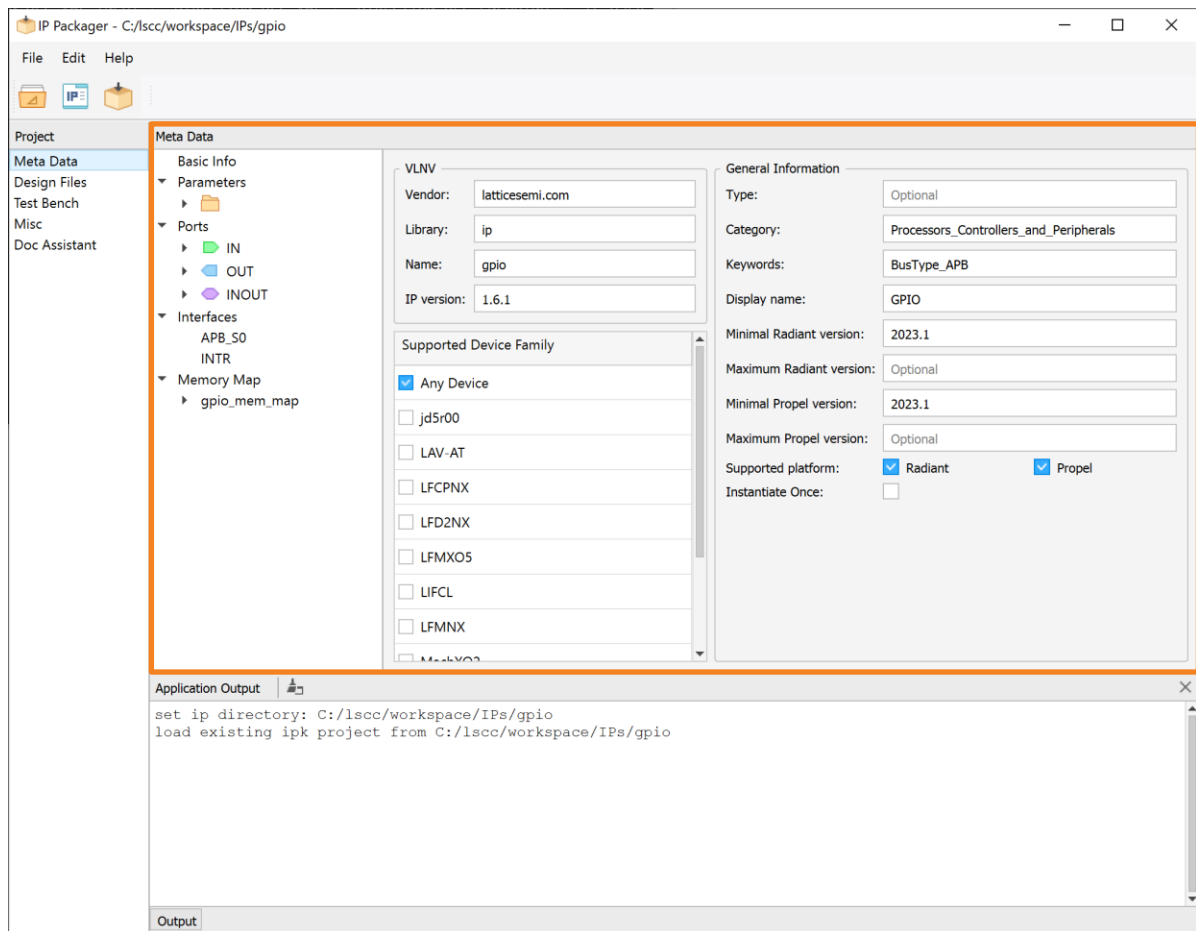


Figure 2.7. IP Packager with Meta Data Details

- To configure basic information:
 - Click **Basic Info** from the **Meta Data** view (Figure 2.8).

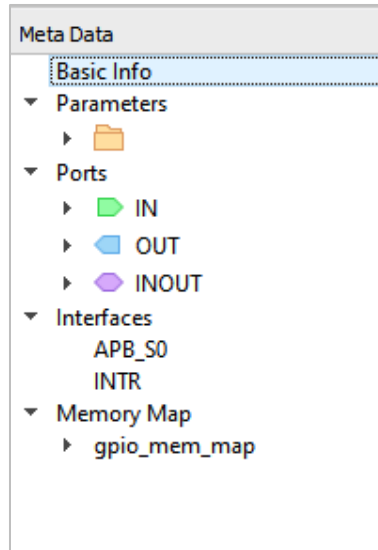


Figure 2.8. Meta Data View

- You can see the Basic Info properties in detail, as shown in Figure 2.9.

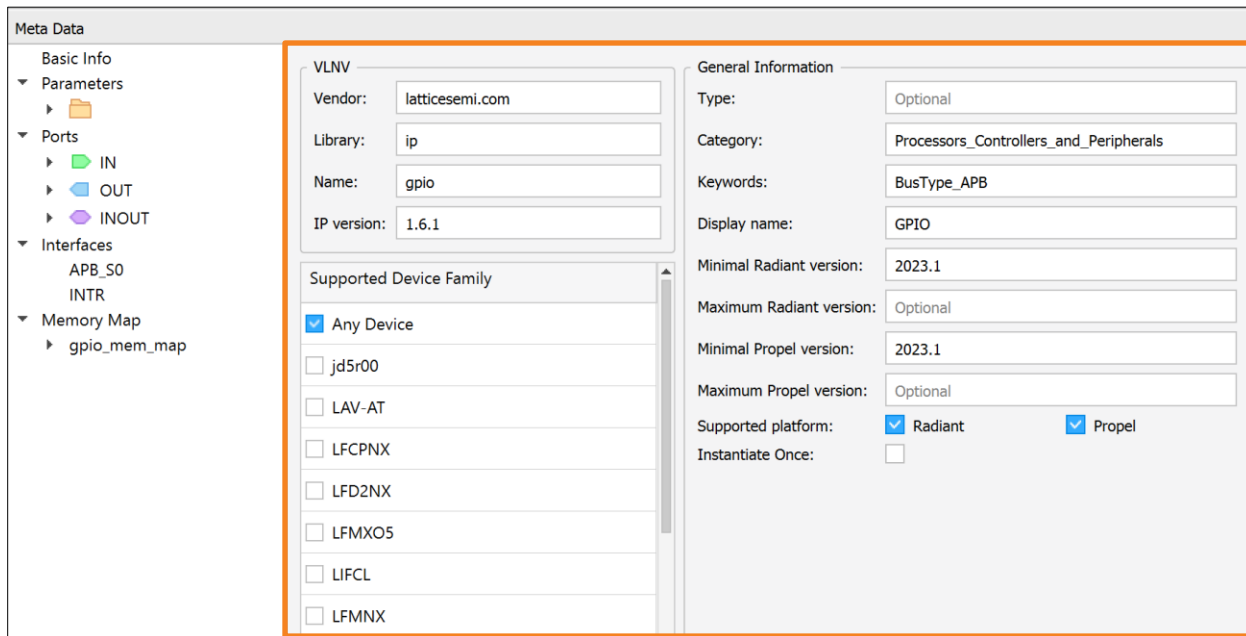


Figure 2.9. Configure Basic Info

- 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** field to enter a desired vendor value.
Note: IP Packager supports both Lattice Radiant Software and Lattice Propel Builder. IP Packager currently is based on Radiant 2023.1. It does not support Radiant version of which is lower than 2023.1. Therefore, **Minimal Radiant version** should be set to 2023.1 or 2023.1+. For Propel, IP Packager currently is based on Propel 2023.1. It does not support Propel version of which is lower than 2023.1. So, **Minimal Propel version** should be set to 2023.1 or 2023.1+. If the version of Radiant or Propel is invalid, a warning message is shown (Figure 2.10).

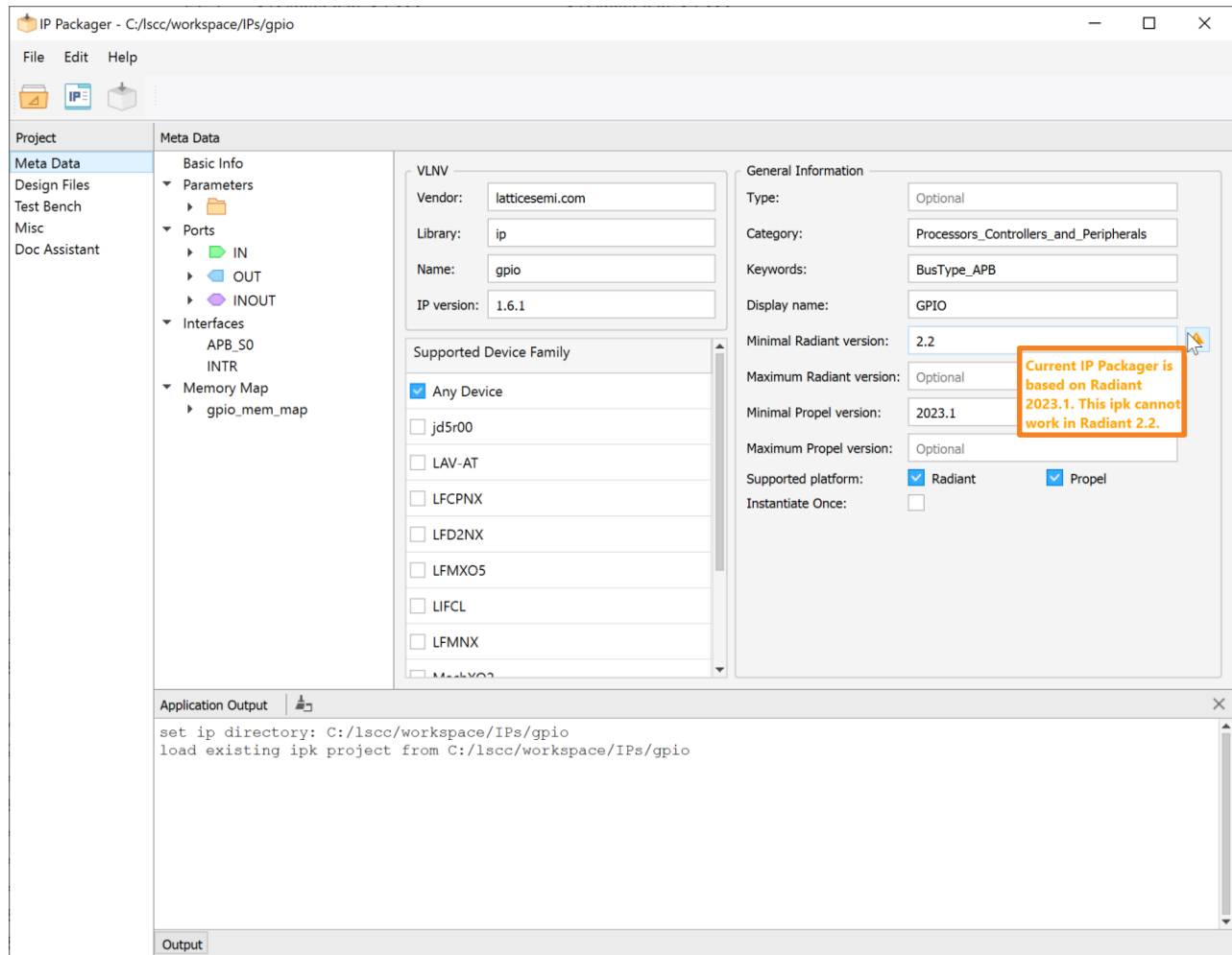


Figure 2.10. Warning Message of Invalid Radiant and Propel Version

b. To configure parameters:

- Parameters are settings that can be used in Ports/Interfaces/Memory map configuration. You can think of them as macros. For example, click on the Port gpio_i, you can see its configuration. In gpio_i, there is a value using EXTERNAL_BUF that is defined in Parameters (Figure 2.11).

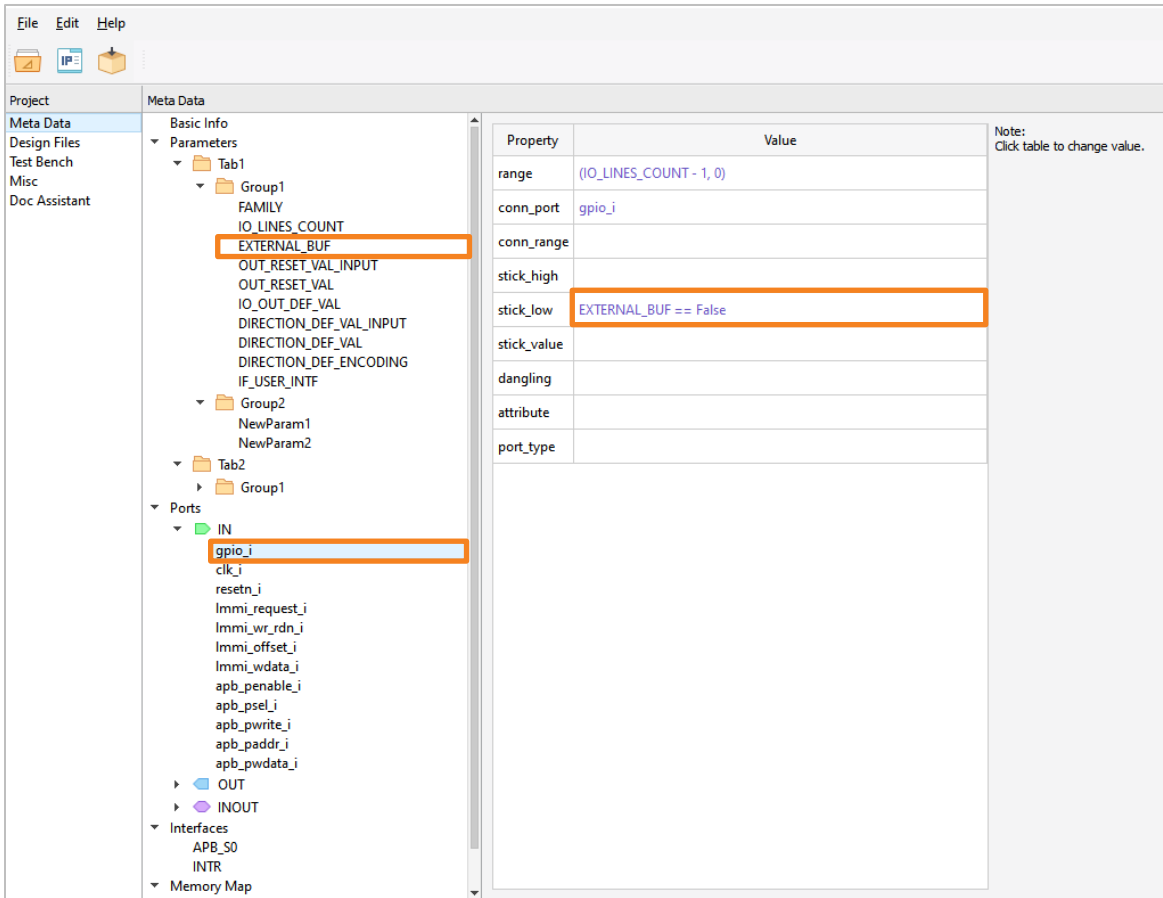


Figure 2.11. Preview on Parameters

- Two-level hierarchy (Tab/Group) is used to categorize the parameters. Click on the **IP Preview** icon from the toolbar (Figure 2.12), you can see the hierarchy relations (Figure 2.13).

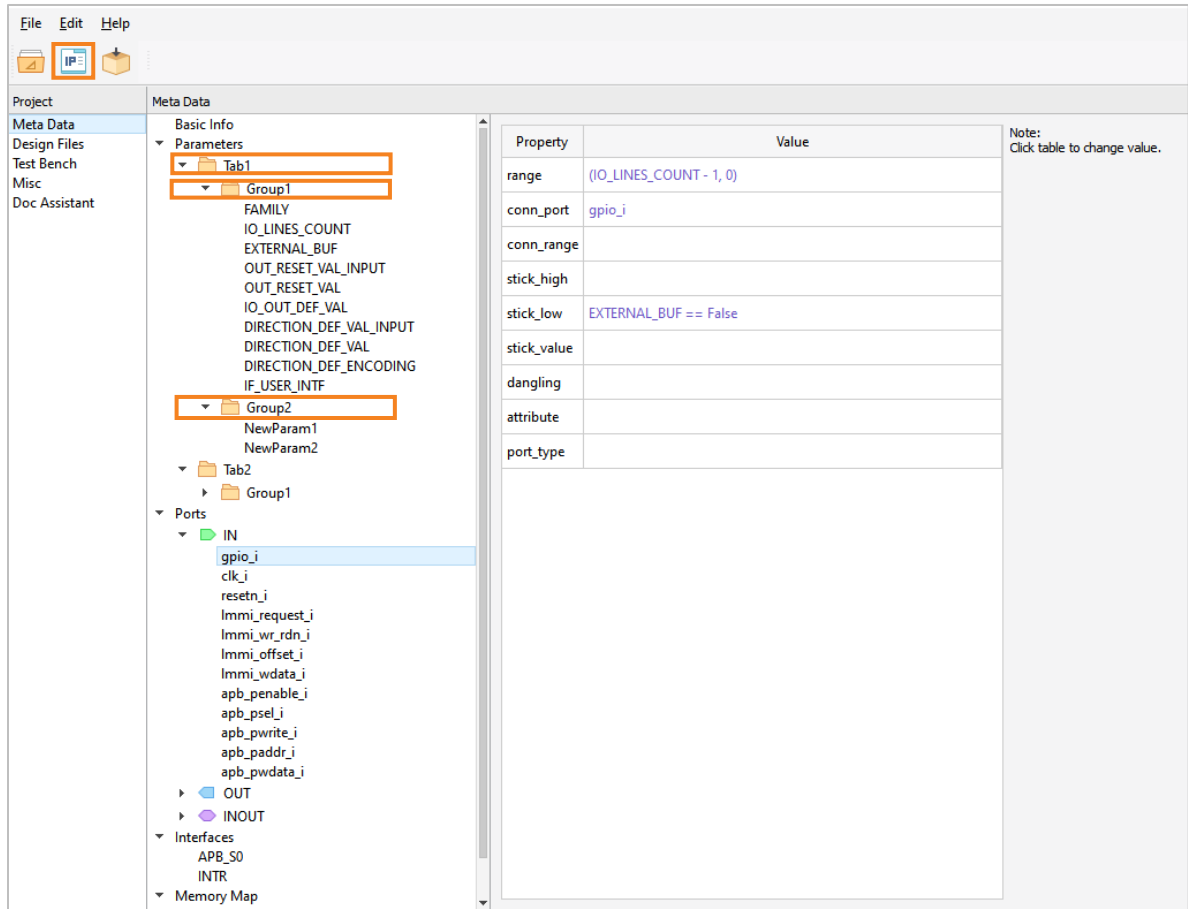


Figure 2.12. Two-Level Hierarchy to Categorize Parameters

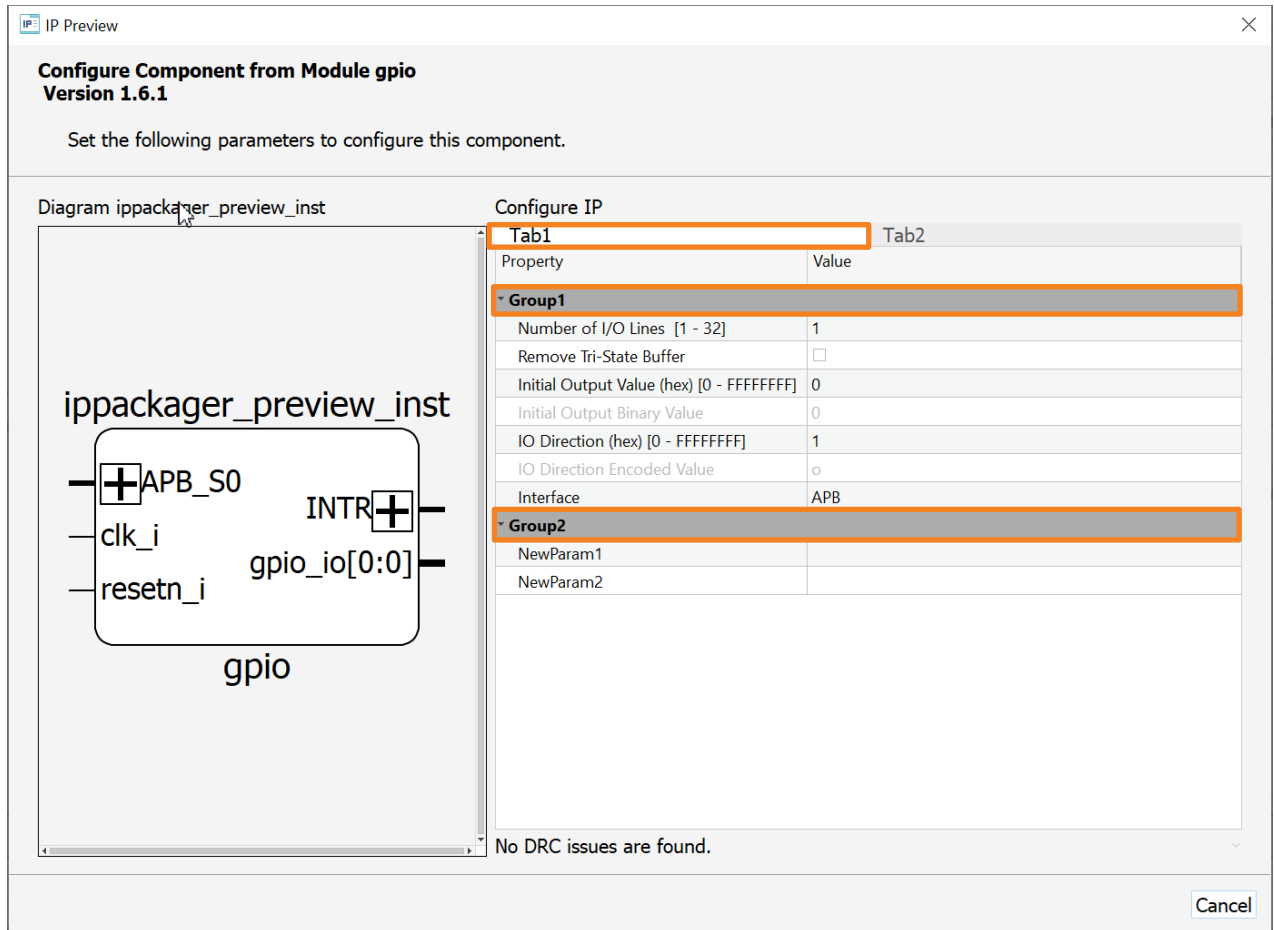


Figure 2.13. IP Preview Showing Two-level Hierarchy in Parameters

- Naming rules for parameters:
 - Parameters in all groups should not have the same name.
 - Characters supported in naming are letters with mixed case, numbers and underscore (_).
- To add a parameter, right-click Parameters from the **Meta Data** view and choose **Add Parameter Tab** (Figure 2.14). **Tab1** is created under **Parameters**.

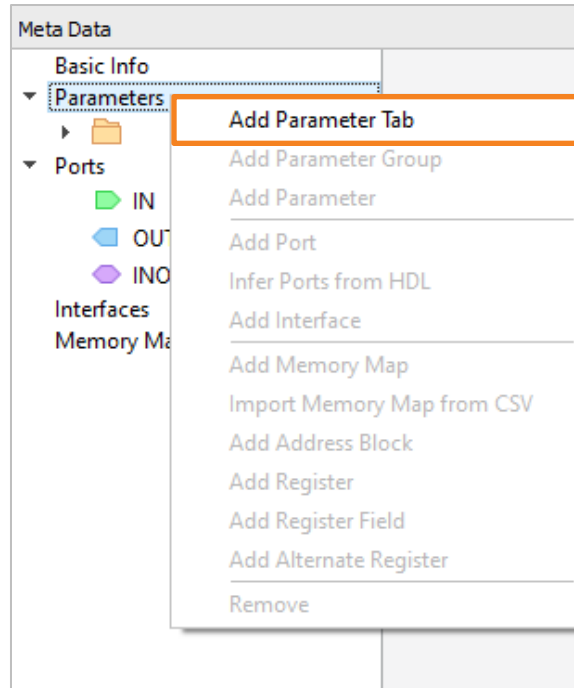


Figure 2.14. Right-click Menu of Parameters in the Meta Data View

- Right-click Tab1 under **Parameters** from the **Meta Data** view, and choose **Add Parameter Group**. Group1 is newly added under Tab1 (Figure 2.15).

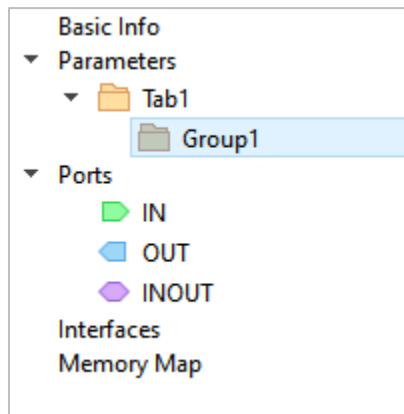


Figure 2.15. Group1 Added to Tab1

- Right-click Group1 under Tab1 from the **Meta Data** view, and choose **Add Parameter**. NewParam1 is newly added under Group1 (Figure 2.16).

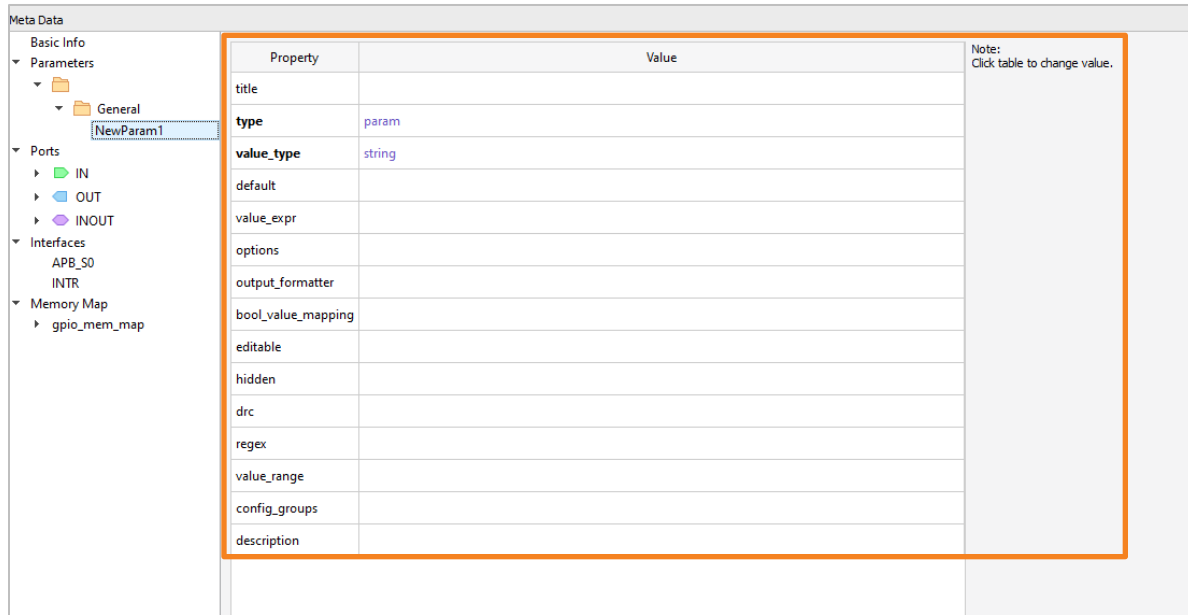


Figure 2.16. NewParam1 Added to Group1

- Double-click NewParam1 to rename the parameter (Figure 2.17). The parameter name is also used as the unique ID of the parameter setting.

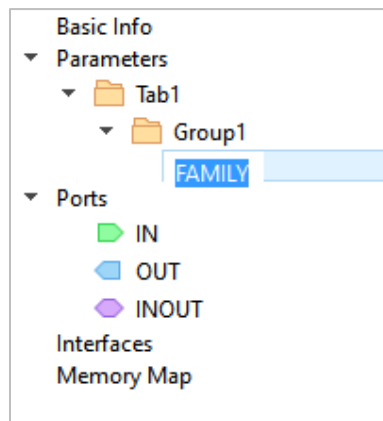


Figure 2.17. Rename a Parameter

- Double-click the property value field to enter the desired parameter value (Figure 2.18), or select the desired value from the drop-down menu (Figure 2.19). The properties are thus configured. Property listed in bold must be configured. The details of each property is shown in Table 2.1.

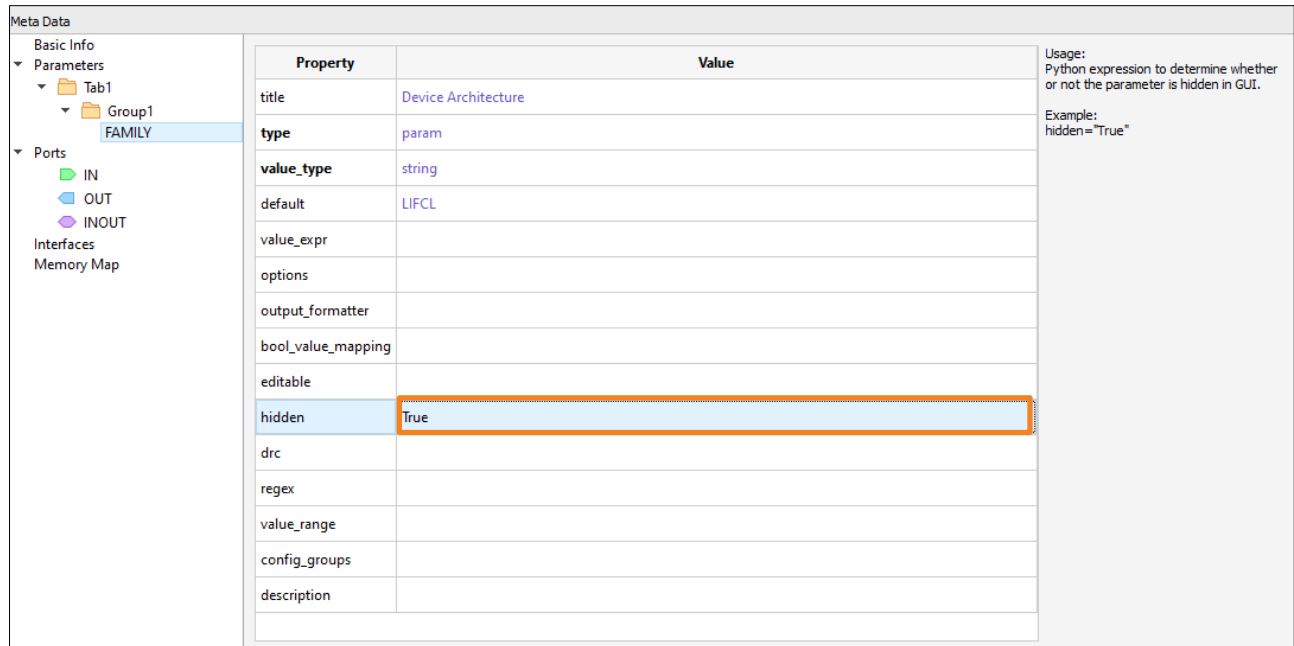


Figure 2.18. Enter Desired Parameter Value

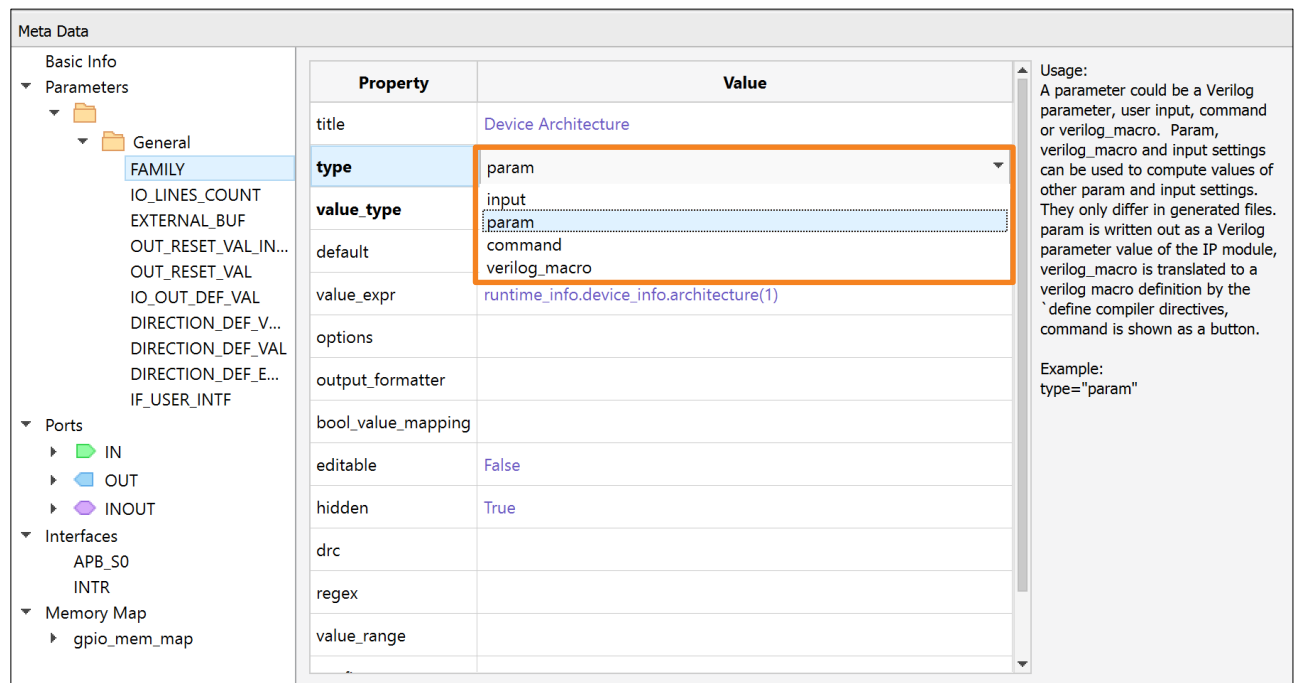


Figure 2.19. Select Desired Parameter Value from Drop-down Menu

Table 2.1. Details of Parameter Property

Property	Value	Mandatory	Description
title	String	No	Short title of the setting. If <i>title</i> is not specified, value of the setting is used. Example: title="Device Architecture"
type	param, input, command, verilog_macro	Yes	Type of the setting. A parameter could be a Verilog parameter, user input, command or verilog_macro. <i>Param</i> , <i>input</i> and <i>verilog_macro</i> settings can be used to compute values of other param and input settings. They only differ in generated files. <i>Param</i> is written out as a Verilog parameter value of the IP module; <i>verilog_macro</i> is translated to a Verilog macro definition by the define compiler directives; while a parameter with the type <i>command</i> is shown as a button. Example: type="param"
value_type	bool, string, int, float, path	Yes	Type of the value. Supported types are <i>bool</i> , <i>int</i> , <i>float</i> , <i>string</i> , and <i>path</i> . The <i>int</i> type supports unlimited precision. The float type supports the precision of float type of C programming language. Refer to the Characteristics of the Floating Types section of the 1999 ISO/IEC C Standard for details. The <i>path</i> type indicates a string which represents a path. "/" is used as separator. Example: value_type="string"
default	Python expression	No	Default value of the setting. If the setting has no <i>default</i> attribute but has the <i>options</i> attribute, the first option is picked as the default value. If the setting has neither <i>default</i> attribute nor <i>options</i> attribute, the initial value of the setting is set to 0 for <i>int</i> , 0.0 for <i>float</i> , "" for <i>string</i> and False for <i>bool</i> . By default, it is not allowed to reference to other setting values. Use <i>value_expr</i> 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 greyed out in GUI display and its value is computed by <i>value_expr</i> . Otherwise, the user input is used.

Property	Value	Mandatory	Description
			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 the GUI. If <i>hidden</i> is set to <i>True</i> (default is <i>False</i>), the item is hidden in the 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 <i>0b</i> . 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, <i>float("inf")</i> . 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 the <i>IP-XACT configGroups</i> attribute, and only supports the <i>SystemBuilder</i> value. If it is defined, related RTL parameter is brought out to IP instance top module, so that the System Builder can re-define its value.
Description	String	No	Detailed description of the setting.
Macro_name	id, value	No	Specify how to name the Verilog macro in <i>verilog_macro</i> type setting item, the ID or value of this setting item. The default value is <i>setting</i> , which means the setting ID is defined as a Verilog macro. If it is set as <i>value</i> , the evaluated setting value is the Verilog macro. Example: maro_name= "value" Result: `define <setting value> Note: This is only applicable to the setting item whose <i>value_type</i> attribute is set to <i>string</i> .

- 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 2.20) are listed.

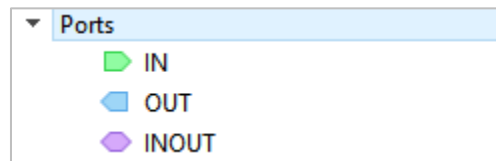


Figure 2.20. Three Port Types

- Right-click **IN** from the Ports area, and choose **Add Port** (Figure 2.21).

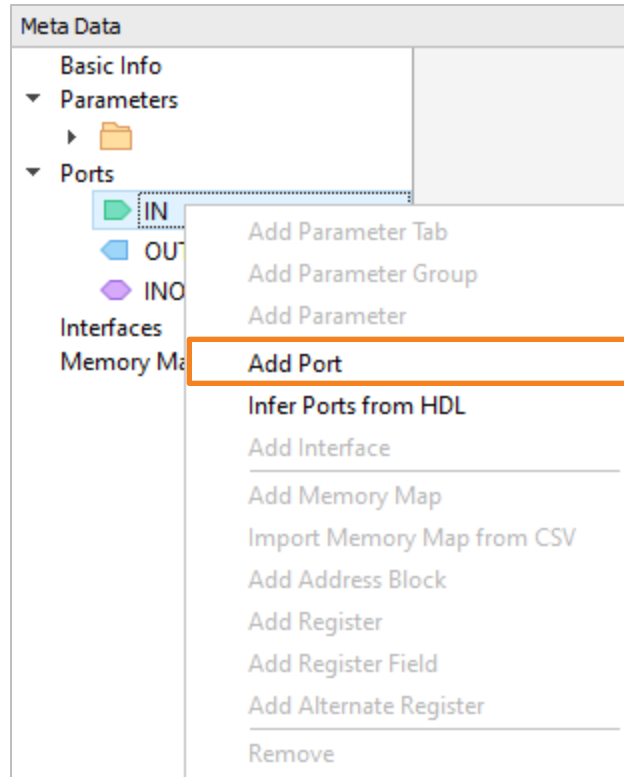


Figure 2.21. Right-click Menu of the IN Port

Note: You can add all ports by using **Infer Ports from HDL**, if you prepare valid RTL files and add RTL files in the **Design Files** section. Click **Infer Ports from HDL**, the **Add inferred ports to meta data** wizard pops up (Figure 2.22). 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.

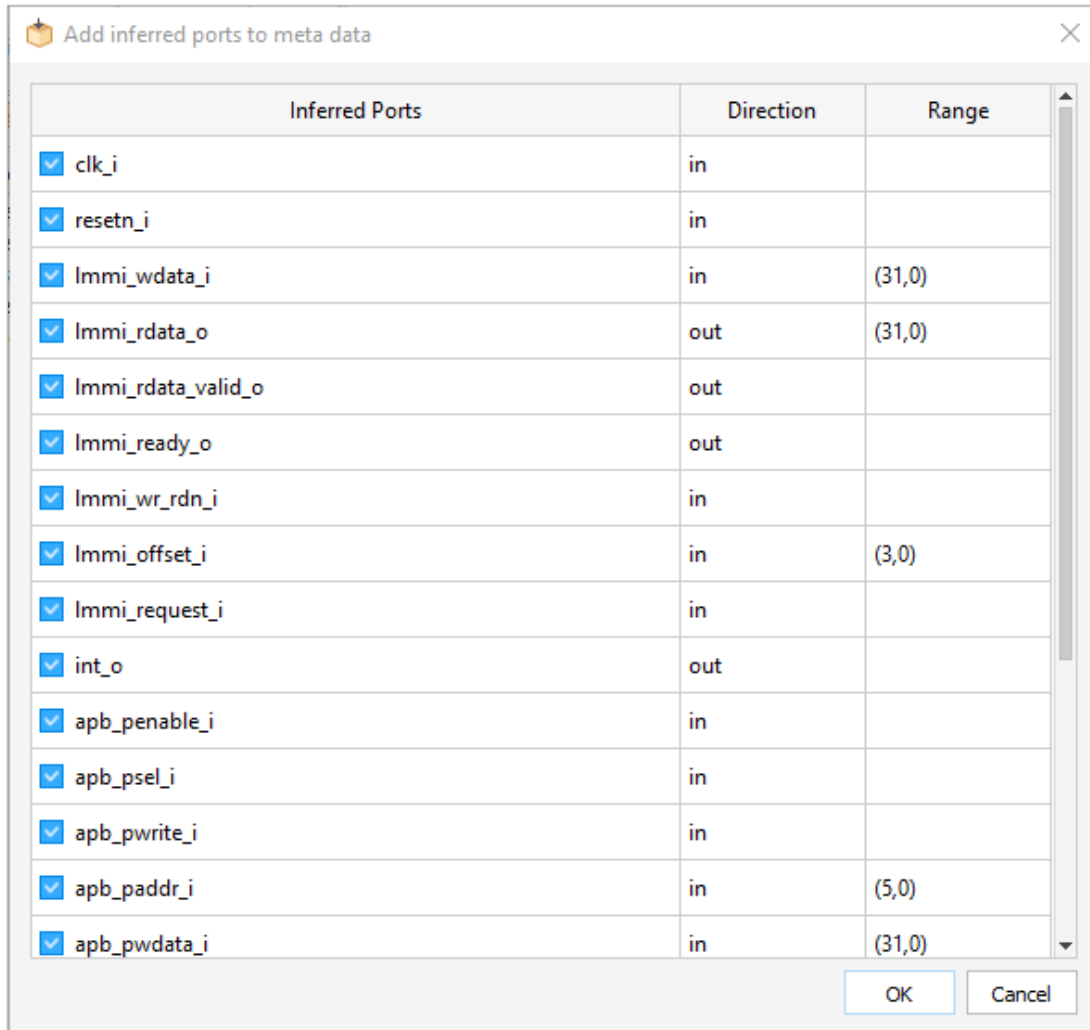


Figure 2.22. Add inferred ports to meta data Wizard

- **NewPort1** is created. Double click NewPort1 to rename the port (Figure 2.23).

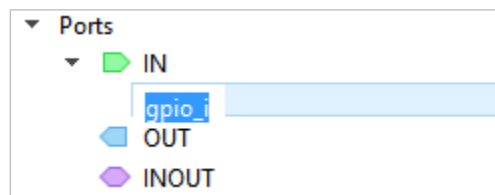


Figure 2.23. Rename an IN Port

- Configure all properties of the IN port as desired (Figure 2.24). Properties listed in bold must be configured. The details of each property is shown in Table 2.2.

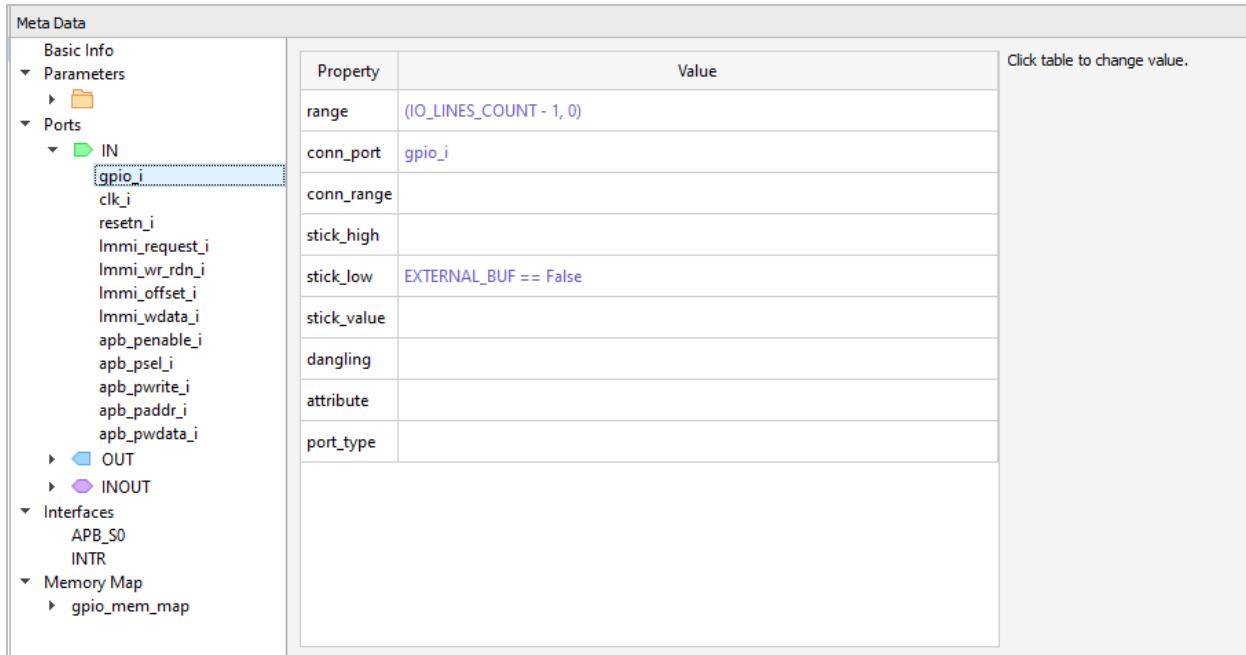


Figure 2.24. Configure an IN Port

Table 2.2. 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_port	Valid Verilog module name	No	Name of port of IP core module to which this port connects. Value of name is used if <i>conn_port</i> is not specified. Example: conn_port="Clk"
conn_range	Python tuple or list with two-integer elements	No	Range of <i>conn_port</i> . 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 attribute of the port.

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

- 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 2.25). A new interface **NewInterface1** is created.

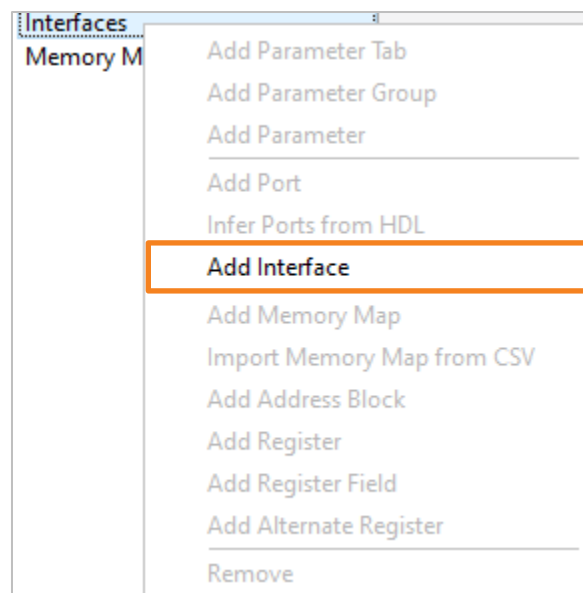


Figure 2.25. Right-click Menu of an Interface

- Double click **NewInterface1** to rename interface (Figure 2.26).

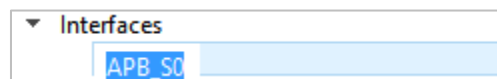


Figure 2.26. Rename Interface

- Configure the interface as desired (Figure 2.27).

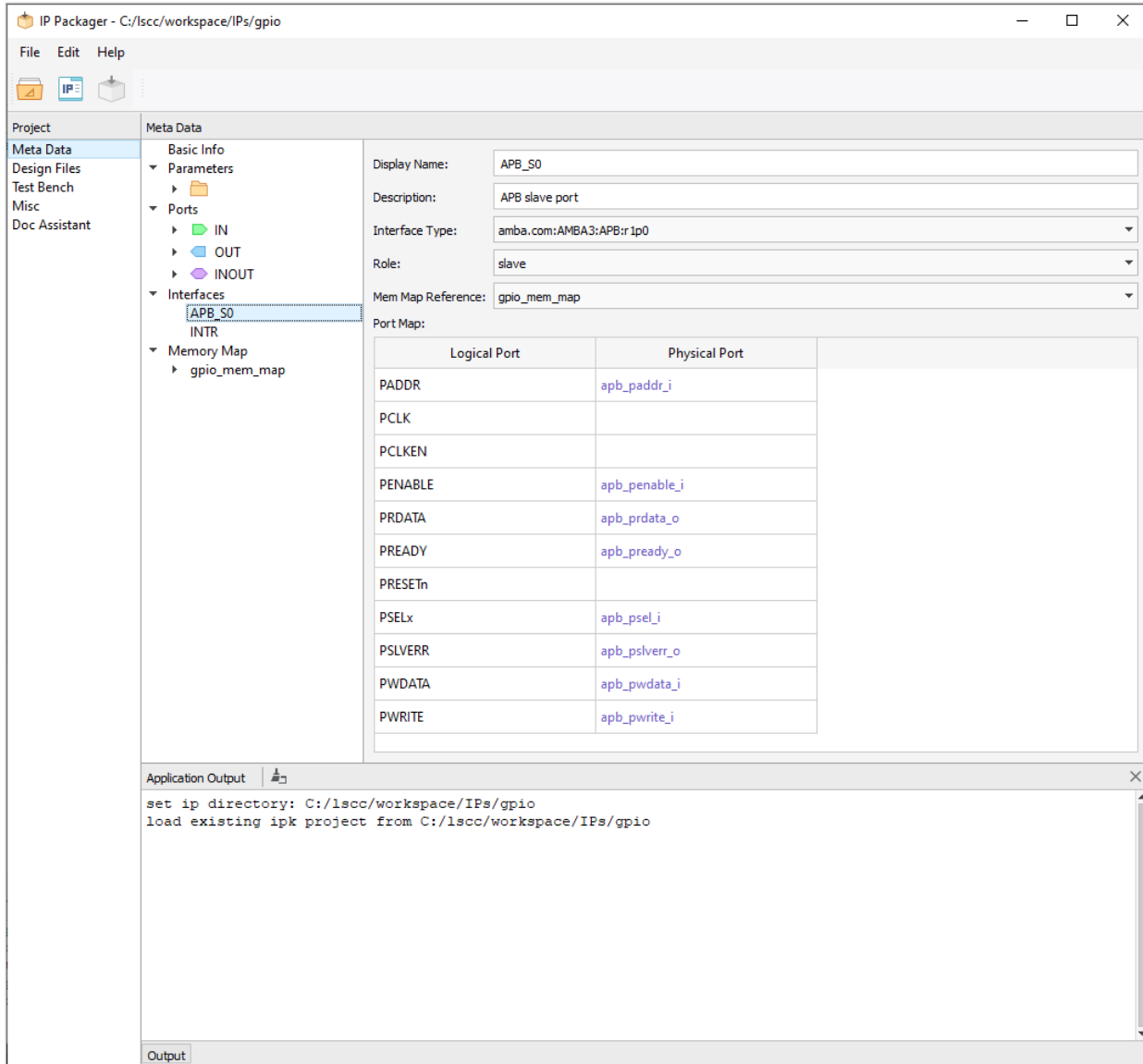


Figure 2.27. Configure an Interface

- 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 2.28). New memory map is created (Figure 2.29).

Note: Lattice IP Packager 2023.1 only supports using **Import Memory Map from CSV** to create memory map. There are some issues if you use **Add Memory Map**.

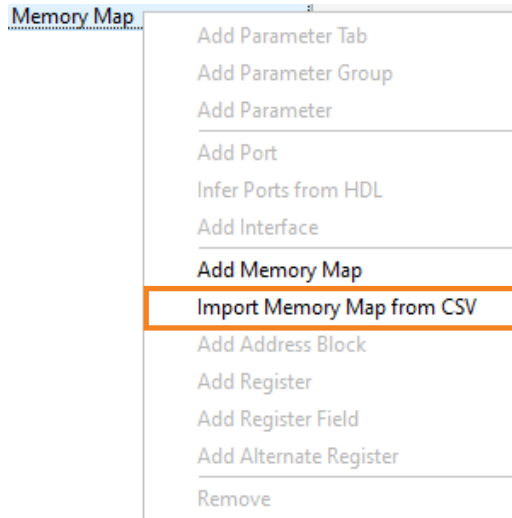


Figure 2.28. Right-click Menu of a Memory Map

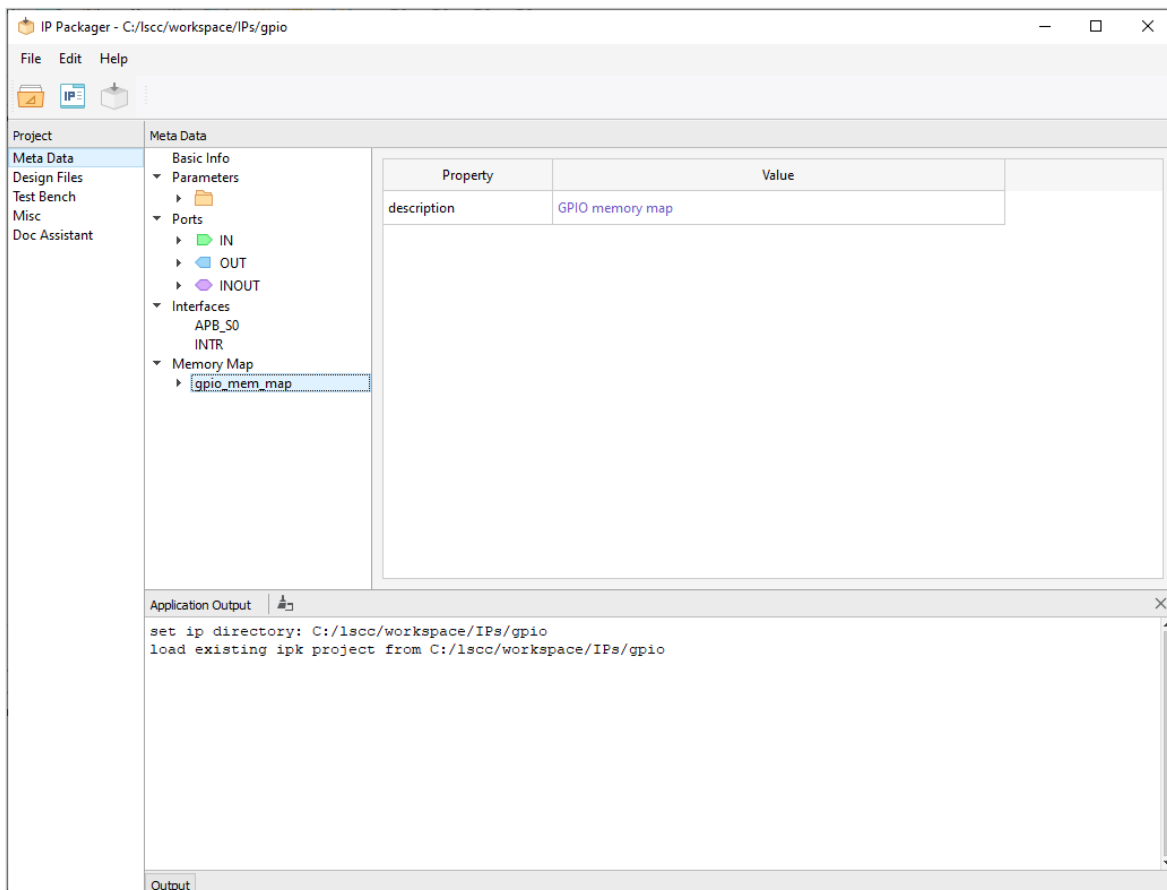


Figure 2.29. Generate New Memory Map

- From the IP Packager Project area, click **Design Files**. The IP Packager GUI shows the Design Files information (Figure 2.30).

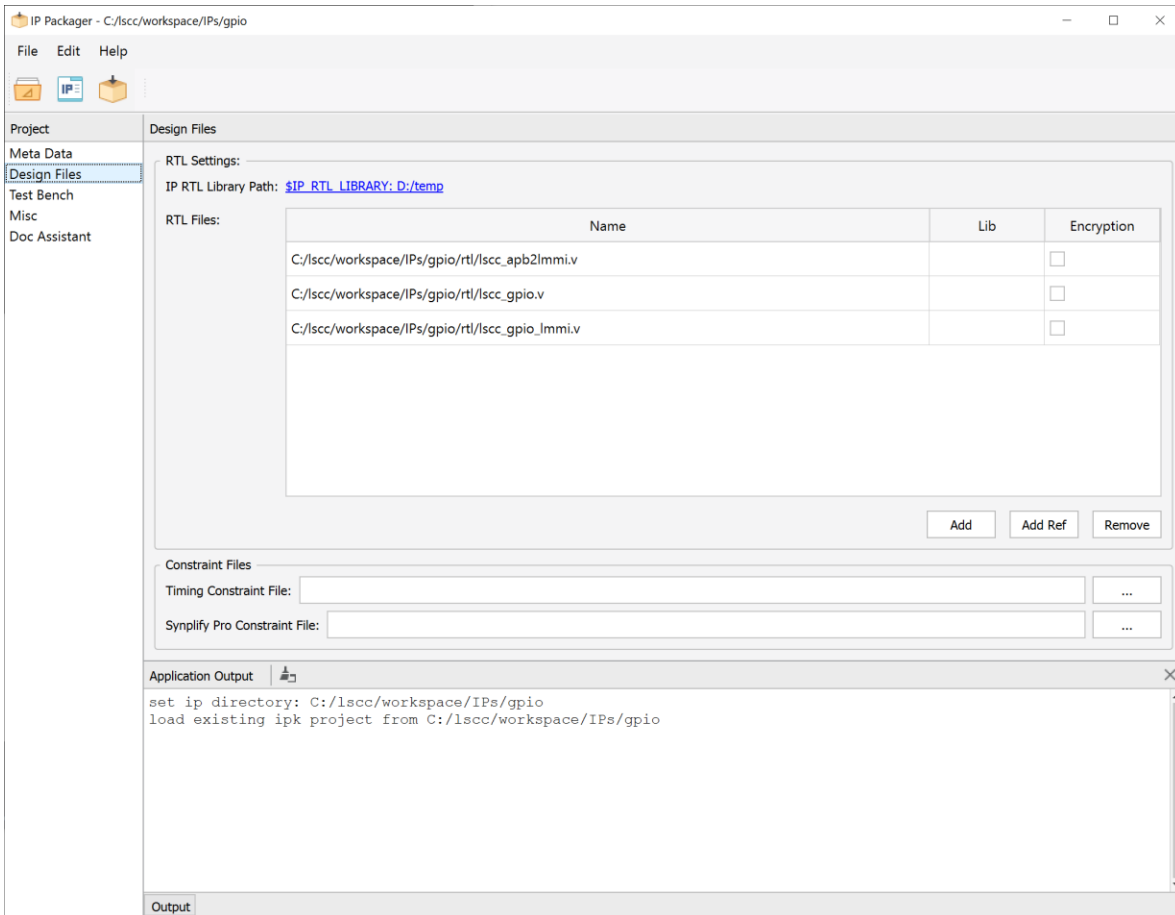


Figure 2.30. IP Packager with Design File Details

- Click the **Add** button to select a desired RTL file in the Design Files field, the added file is packaged into the IP.
- Click the **Add Ref** button to select a reference RTL file in the IP RTL Library Path, this file is not packaged into the IP.
- Click the **Remove** button to remove an RTL file.
- Use the “...” option to add a desired timing constraint file and synplify Pro 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.

IP Packager supports importing RTL files from the library path without saving them locally. When files change, you can always get the latest files from the library path without updating the IP version or regenerating the IP. This help avoid maintaining the RTL files in multiple locations.

- To add reference RTL file into the IP:
 - Set the path of RTL library.
 - Add reference RTL file from library by clicking the **Add Ref** button.

- To set IP Rtl Library path in Propel Builder:
If you want to use an IP that contains a reference RTL file in Propel Builder, you must set the value of IP Rtl Library in **Menu > Tools > Options > Directories**, as shown in [Figure 2.31](#). Alternatively, you can set the value to the environment variable "LATTICESEMI_IP_RTL_LIB_PATH".

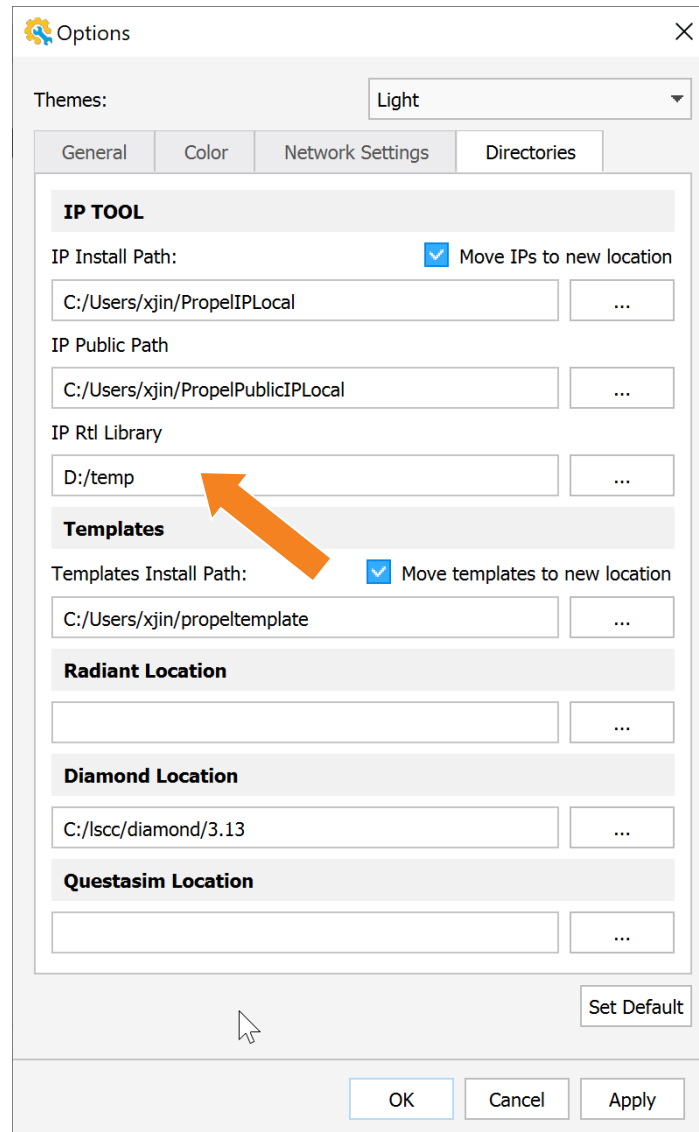


Figure 2.31. Set IP Rtl Library Path in Lattice Propel Builder

- To Set IP RTL Library path in Radiant:
If you want to use an IP that contains a reference RTL file in Radiant, you must set the value of IP RTL Library in **Menu > Tools > Options > General > Directories**, as shown in [Figure 2.32](#). Alternatively, you can set the value to the environment variable "LATTICESEMI_IP_RTL_LIB_PATH".

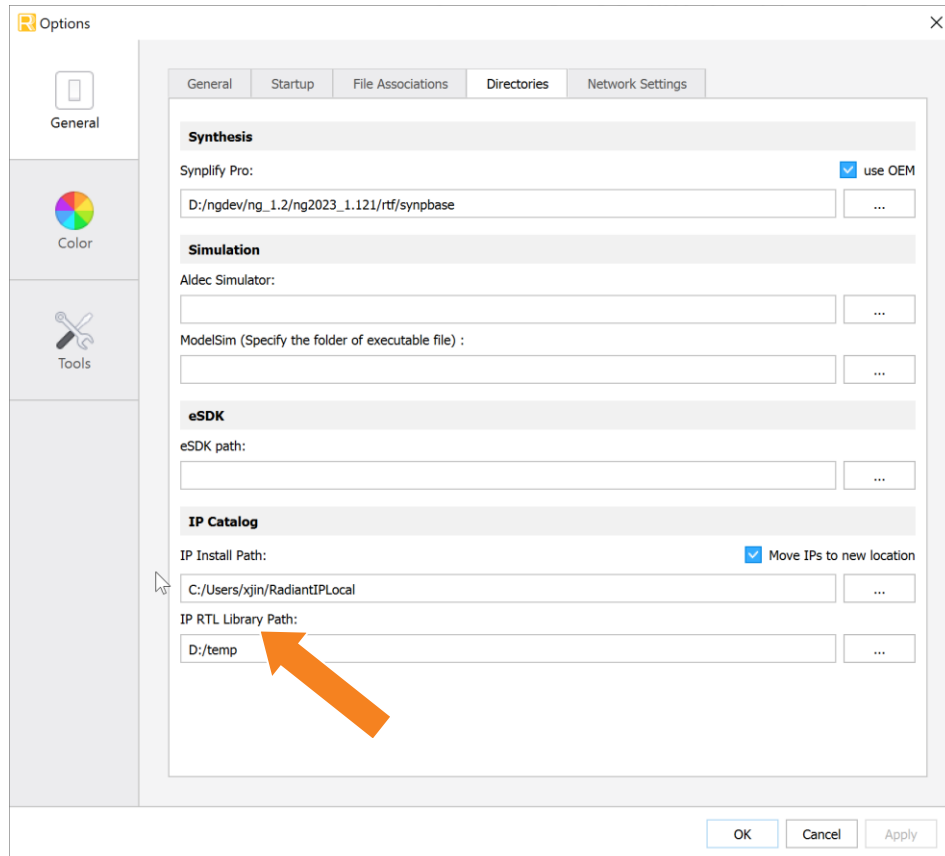


Figure 2.32. Set IP RTL Library Path in Lattice Radiant Software

3. From the IP Packager Project area, click **Test Bench**. The Test Bench information is shown in detail ([Figure 2.33](#)).

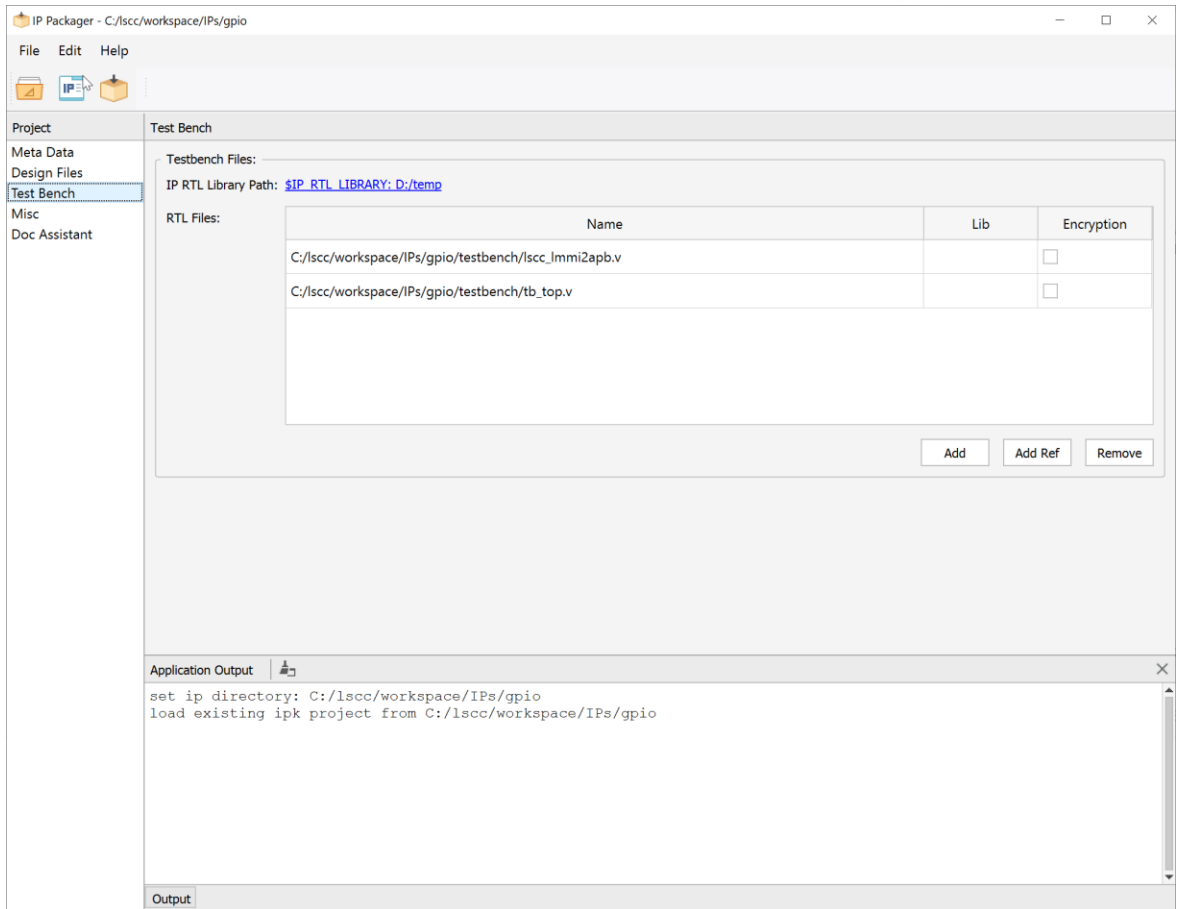


Figure 2.33. IP Packager with Test Bench Details

- Follow the steps as those for configuring the Design Files to configure the Test Bench.

Note: The Test Bench files configuration is not mandatory.

- From the IP Packager Project area, click **Misc**. The Misc related information is shown in detail (Figure 2.34). Configure Misc information as desired.

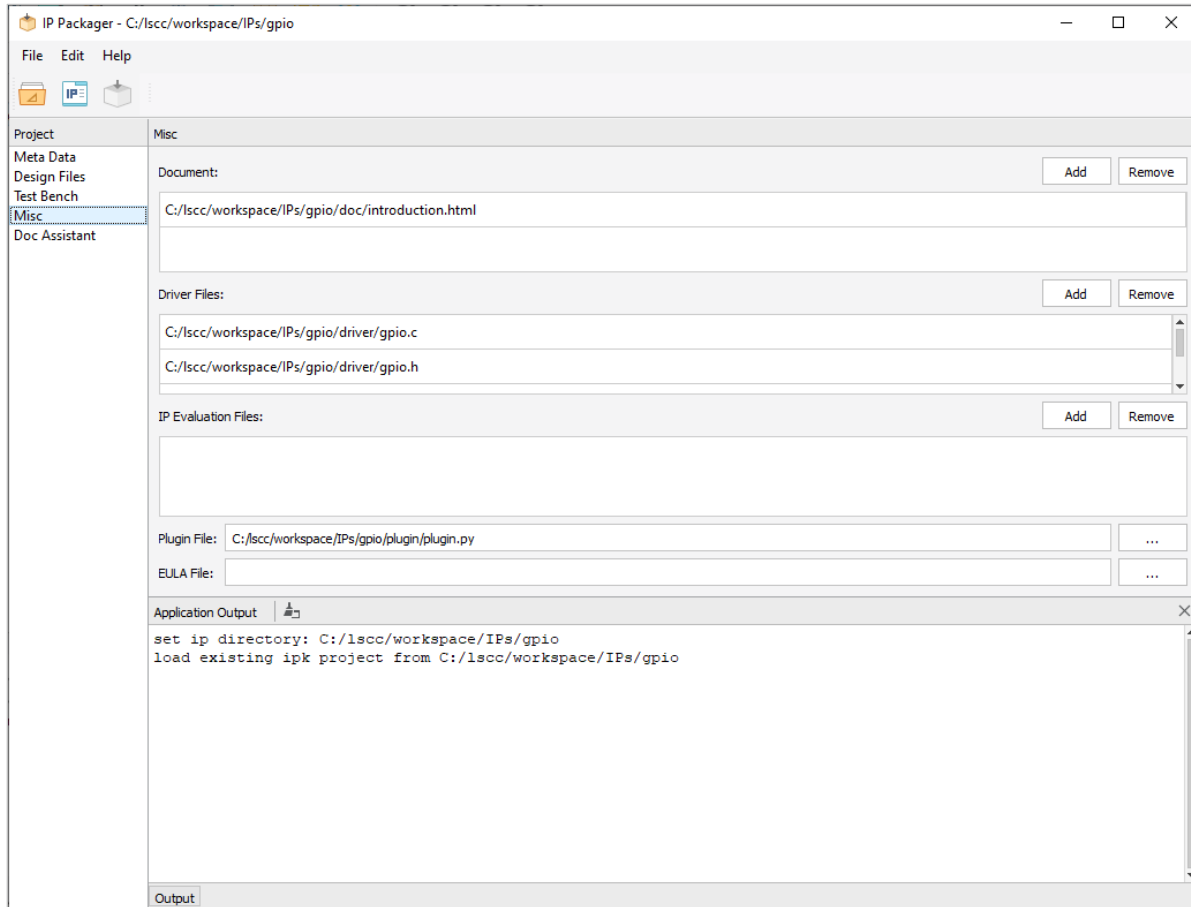


Figure 2.34. IP Packager with Misc Details

- From the IP Packager Project area, click **Doc Assistant**. The Doc Assistant information is shown in detail (Figure 2.35).

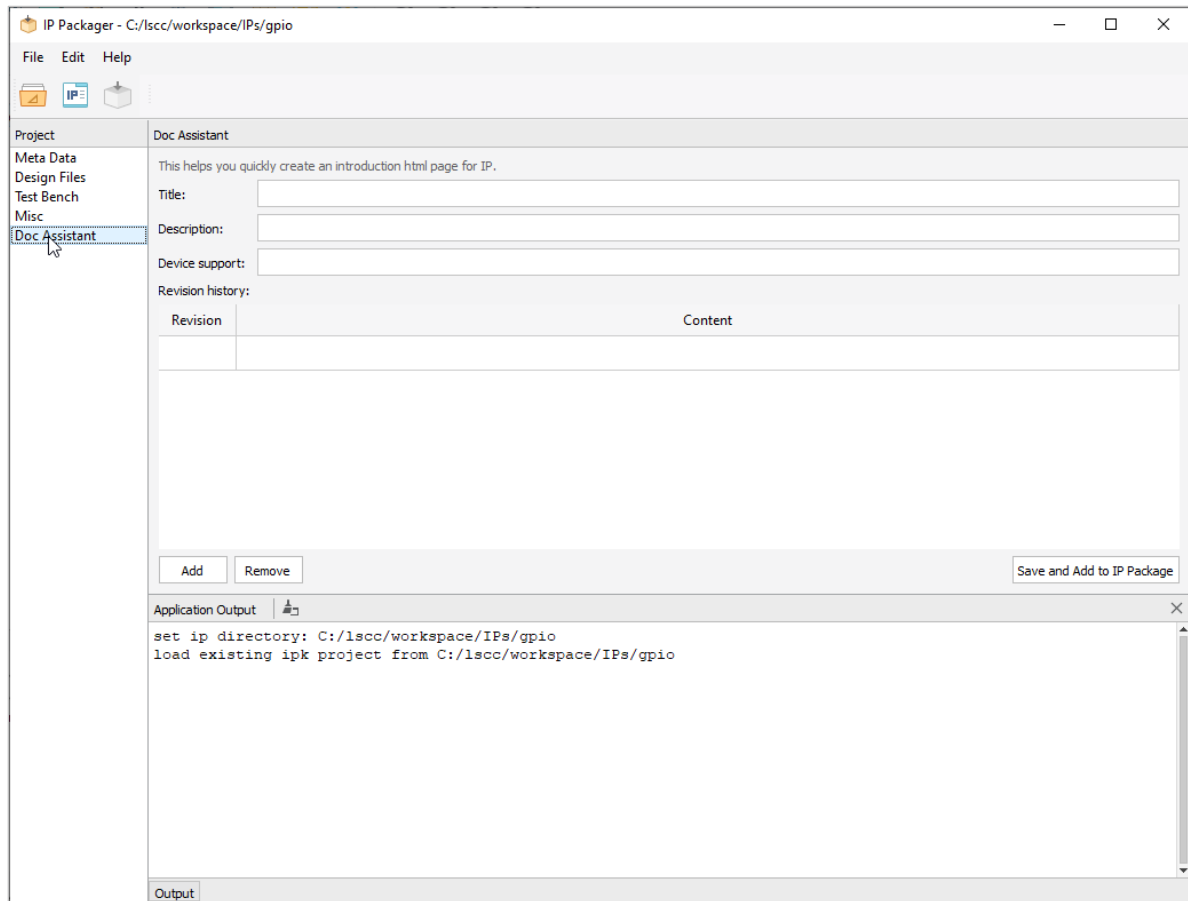


Figure 2.35. IP Packager with Doc Assistant Details

- 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 2.36 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.

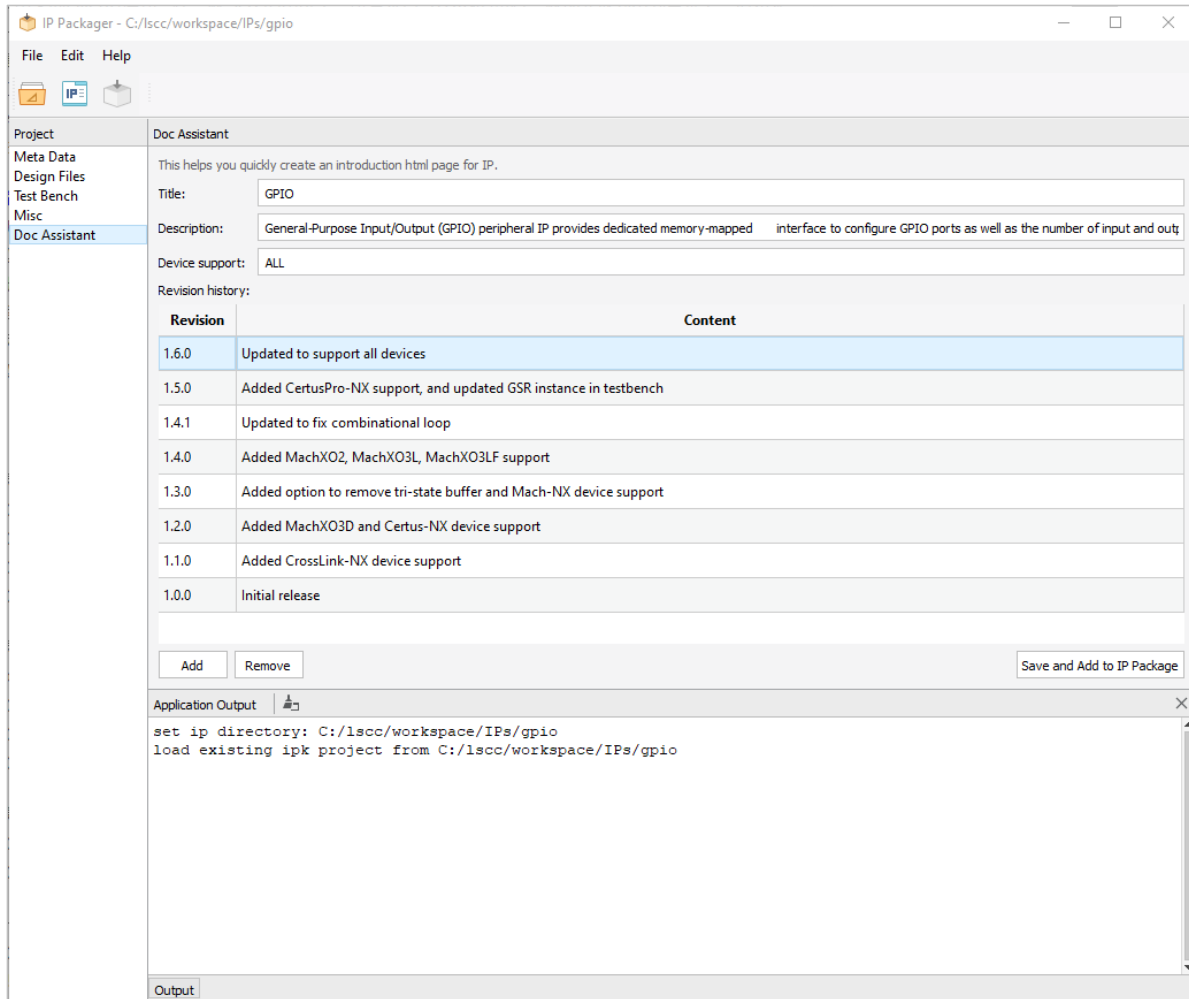




Figure 2.36. Configure Doc Assistant

2.2.3. Previewing IP

1. Choose **Edit** >  **IP Preview** from the IP Packager menu. Or, click the **IP Preview** icon  from the IP Packager toolbar.
2. **IP Preview** pops up showing the configuration component for the IP module (Figure 2.37), if the IP project configuration is correct. Otherwise, an error message pops up (Figure 2.38).

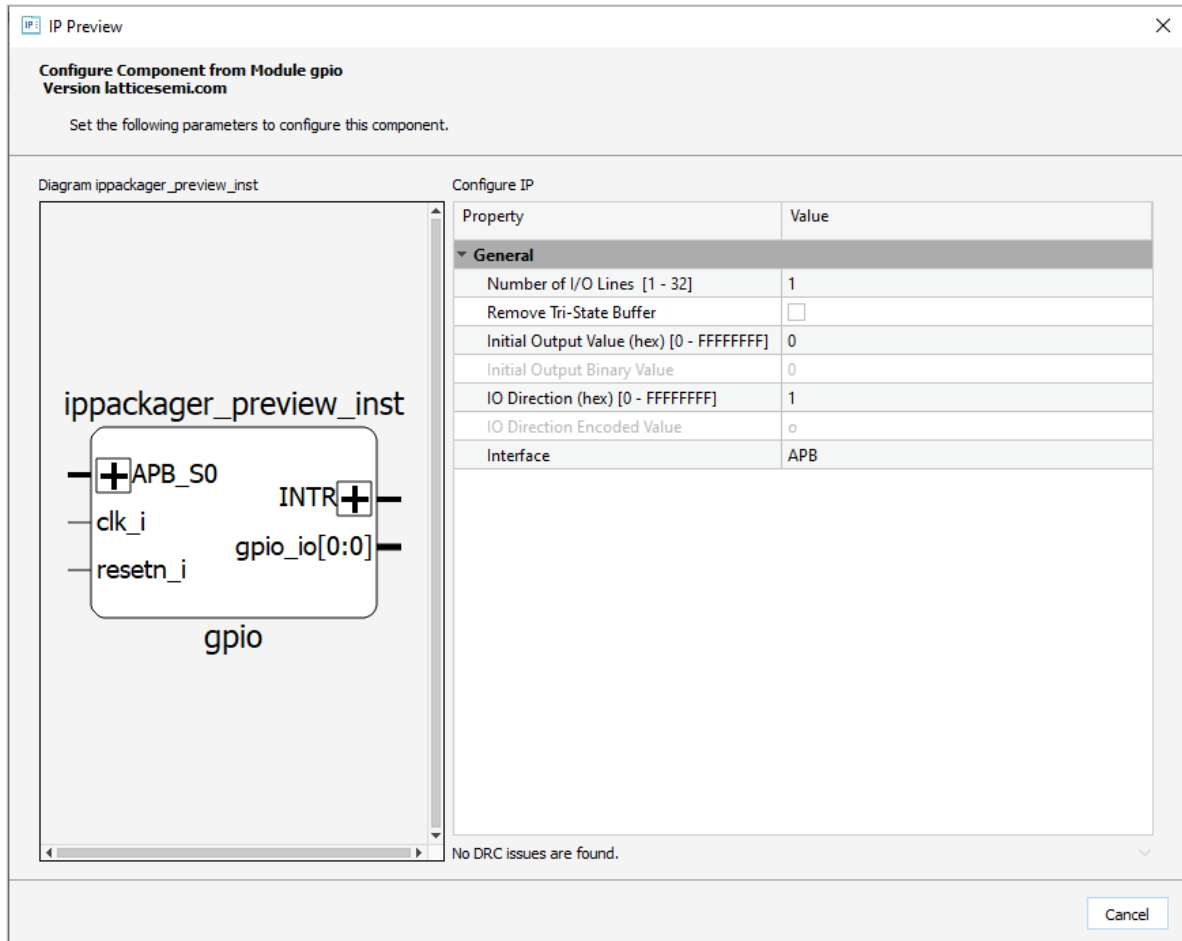


Figure 2.37. IP Preview Shows Configuration for IP Module

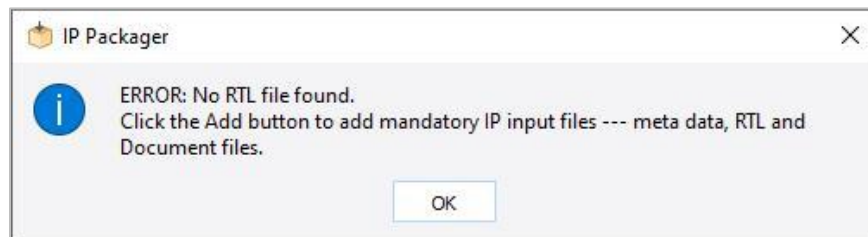




Figure 2.38. Error Message Pops Up in IP Packager

2.2.4. Packaging IP

1. Choose **Edit** >  **Package IP** from the IP Packager menu. Or, click the **Package IP** icon  from the Toolbar.
2. A message pops up showing the packaging is completed successfully (Figure 2.39).

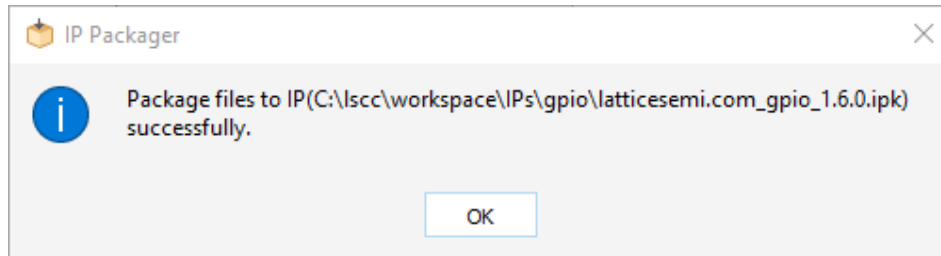


Figure 2.39. IP Packager Pops up Successful Packaging Message

Note: When the setting value is changed, the packaging operation is disabled. You must perform a preview operation before packaging. If the preview is successful, the packaging operation is enabled. Otherwise, error messages are displayed in the output widget.

2.3. Editing IP Package Files

2.3.1. Metadata File

Metadata.xml can be generated after editing IP. You can also manually edit the file. IP Platform uses XML file to describe metadata of a soft IP. Namespace "lscip" 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 2.40).

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

Figure 2.40. Example XML of Metadata Layout

1. <general> node: describes the general information about a soft IP, for example, its name, version, and category. Table 2.3 shows the child nodes of <general> node.

Table 2.3. Child Nodes of General Node

Child Node	Mandatory	Description
vendor	Yes	Soft IP vendor. An official soft IP should have the vendor name. Example: "latticesemi.com"
library	Yes	Library of the soft IP. If <i>library</i> is not set, the default value <i>ip</i> is used. Example: ip, interface
name	Yes	Name of the soft IP. The <i>name</i> must be unique among soft IP of the same vendor and library. Example: adder
display_name	No	Name to be displayed in the software. If <i>display_name</i> is not set, software displays <i>name</i> directly. Example: Adder
version	Yes	Version of the soft IP. Example: 1.0.0
category	Yes	Category of the soft IP. <i>Category</i> 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 2.4 shows attributes of <setting> nodes.

Table 2.4. Attributes of Setting Nodes

Attribute	Value	Mandatory	Description
id	String	Yes	The unique ID of the setting, which is also referred to as: Parameter name in RTL codes Python variable name Tcl variable name To make the 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 a

Attribute	Value	Mandatory	Description
			letter. Example: id="num_outputs"
title	String	No	Short title of the setting. If <i>title</i> is not specified, the value of <i>setting</i> is used. Example: title="Number of Output"
type	param, input, command, verilog_macro	Yes	Type of the setting. A parameter could be a Verilog parameter, user input, command or verilog_macro. <i>Param</i> , <i>input</i> and <i>verilog_macro</i> settings can be used to compute values of other param and input settings. They only differ in generated files. <i>Param</i> is written out as a Verilog parameter value of the IP module, <i>verilog_macro</i> is translated to a Verilog macro definition by the define compiler directives, while a parameter with the type of <i>command</i> is shown as a button. Example: type="param"
value_type	bool, string, int, float, path	Yes	Type of the value. Supported types are <i>bool</i> , <i>int</i> , <i>float</i> , <i>string</i> , and <i>path</i> . The <i>int</i> type supports unlimited precision. The <i>float</i> 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 <i>path</i> type indicates a string which represents a path. "/" is used as separator. Example: value_type="int"
default	Python expression	No	Default value of the setting. If the setting has no <i>default</i> attribute but has <i>options</i> attribute, the first option is picked as default value. If the setting has neither <i>default</i> attribute nor <i>options</i> attribute, the initial value of setting is set to 0 for <i>int</i> , 0.0 for <i>float</i> , "" for <i>string</i> and <i>False</i> for <i>bool</i> . 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 in the GUI to display a drop-down 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	No	Python expression to determine if the setting is editable. When a setting is not editable, it is greyed out in the GUI display and its value is computed by

Attribute	Value	Mandatory	Description
	expression		value_expr. Otherwise, the 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 <i>True</i> (default is <i>False</i>), 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 <i>0b</i> . 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, <i>float("inf")</i> . 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 the <i>IP-XACT configGroups</i> attribute, and only supports the <i>SystemBuilder</i> 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 <i>group1</i> is displayed as sub-items under a <i>group1</i> item on GUI. The settings with the same <i>group1</i> should be written continuously if you want GUI to group them under one <i>group1</i> item. Otherwise, you can see multiple groups with the same name in GUI. Example: group1= "Output Setting"
group2	String	No	Group the <i>group1</i> groups. <i>Group1</i> groups of the same <i>group2</i> is displayed in a separate page in the GUI. So, a two-level hierarchy is supported in GUI display. Unlike <i>group1</i> , you need not write setting nodes with the same <i>group2</i> continuously. You must follow the rule for <i>group1</i> that all the settings with the same <i>group1</i> must be in the same <i>group2</i> .
Macro_name	id, value	No	Specify how to name the Verilog macro in <i>verilog_macro</i> type setting item, the ID or value of this setting item. The default value is <i>setting</i> , which means the setting ID is defined as a Verilog macro. If it is set as <i>value</i> , 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 <i>value_type</i> attribute is set to <i>string</i> .

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 stuck to fixed value, or an output port is dangling, the port is not used and is invisible after generation. Table 2.5 shows the attributes of <port> nodes.

Table 2.5. 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 two 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 <i>name</i> is used, if <i>conn_port</i> is not specified. Example: conn_port="Clk"
conn_range	Python tuple or list with 2 int elements	No	Range of <i>conn_port</i> . 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)
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	<i>Data</i> , <i>reset</i> and <i>clock</i> are valid values. The default value is <i>data</i> . <i>Port_type</i> is passed to the IPXact component.xml as a <i>lscsip:isClk</i> or <i>lscscip:isRst</i> node in <i>venderExtensions</i> in <i>component/model/ports/port</i> .

4. <busInterfaces> node: contains a list of all interface ports of the soft IP (Figure 2.41). 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.

```

<lscsip:busInterfaces>
  <lscsip:busInterface>
    <lscsip:name>AHBL_S00</lscsip:name>
    <lscsip:displayName>AHBL_S00</lscsip:displayName>
    <lscsip:description>AHB-Lite slave port</lscsip:description>
    <lscsip:busType library="interface" name="ahblite"
vendor="lattice" version="1.0"/>
    <lscsip:abstractionTypes>
      <lscsip:abstractionType>
        <lscsip:abstractionRef library="interface"
name="ahblite_rtl" vendor="lattice" version="1.0"/>
        <lscsip:portMaps>
          <lscsip:portMap>
            <lscsip:logicalPort>
              <lscsip:name>HSEL</lscsip:name>
            </lscsip:logicalPort>
            <lscsip:physicalPort>
              <lscsip:name>ahbl_s00_hsel_slv_i</lscsip:name>
            </lscsip:physicalPort>
          </lscsip:portMap>
          <lscsip:portMap>
            <lscsip:logicalPort>
              <lscsip:name>HADDR</lscsip:name>
            </lscsip:logicalPort>
            <lscsip:physicalPort>
              <lscsip:name>ahbl_s00_haddr_slv_i</lscsip:name>
            </lscsip:physicalPort>
          </lscsip:portMap>
        </lscsip:portMaps>
      </lscsip:abstractionType>
    </lscsip:abstractionTypes>
    <lscsip:slave>
      <lscsip:memoryMapRef memoryMapRef="ahbs_mem_map"/>
    </lscsip:slave>
  </lscsip:busInterface>
</lscsip:busInterfaces>
    
```

Figure 2.41. Example of busInterface Node

5. <addressSpaces> node: specifies the addressable area seen by bus interfaces of type master (Figure 2.42). 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.

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

Figure 2.42. Example of addressSpaces Node

6. <memoryMaps> node: specifies the information about the range of registers, memory, or other address blocks accessible through the slave interface (Figure 2.43). 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.

```

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

Figure 2.43. Example of memoryMaps Node

- <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 2.44).

```

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

```

Figure 2.44. Example of componentGenerators Node

- <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 2.6 shows the element in < estimatedResource> node.

Table 2.6. Elements in estimatedResources 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.

- <outFileConfigs> node: specifies all customized output file configuration nodes <fileConfig> for the 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 2.45).

```

metadata.xml
<lscsip:ip version="1.0"
xmlns:lscsip="http://www.latticesemi.com/XMLSchema/Radiant/ip"
xmlns:xi="http://www.w3.org/2001/XInclude">
  <lscsip:general> ... </lscsip: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" />
  <lscsip:ports> ... </lscsip:ports>
</lscsip:ip>

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

```

Figure 2.45. 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.

2.3.2. Implementation RTL Files

You should put all IP implementation RTL files in the `rtl` sub-directory of the 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. The 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 2.46](#)).

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

Figure 2.46. Example of Specifying Lib for VHDL

2.3.3. 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 2.47](#)) 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 item 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 2.47. Template of Plugin File

2.3.4. Memory Map CSV File

Lattice IP Packager 2023.1 supports importing memory map from a CSV file to create memory map. You can edit a CSV file. Figure 2.48 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 2.48). 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_S PI_MODE
8	FIELD	sck_div	sck_div	Sets the SPI clock divider		2	3	TRUE	read-write	DEFAULT_S CK_DIV
9	FIELD	reserved	reserved	Reserved bits		5	26	TRUE	read-write	0
10	FIELD	soft_reset	soft_reset	Resets internal soft logic		31	1	TRUE	read-write	0
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
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

Figure 2.48. Example of Memory Map CSV File

Appendix A. metadata.xsd

```
<?xml version="1.0" encoding="gb2312"?>
<!-- IP Metadata Schema 0.0.5 -->
<!--
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 07-Jan-2021 1. new features in Radiant 3.0
    0.0.5 06-Apr-2021 1. fully support Verilog macro
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.latticesemi.com/XMLSchema/Radiant/ip"
    xmlns:lscip="http://www.latticesemi.com/XMLSchema/Radiant/ip"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:import schemaLocation="xml.xsd" namespace="http://www.w3.org/XML/1998/namespace"/>
    <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, verilog macro, or just
      for user input?
      Valid values are &quot;input&quot;;, &quot;verilog_macro&quot;;,
      &quot;param&quot;;, and &quot;command&quot;;.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="param"/>
    <xs:enumeration value="input"/>
    <xs:enumeration value="verilog_macro"/>
    <xs:enumeration value="command"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="settingMacroNameType">
  <xs:annotation>
    <xs:documentation>
      Specify what is used to name the Verilog macro, ID or Value?
      Valid values are &quot;id&quot;; and &quot;value&quot;;.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="id"/>
    <xs:enumeration value="value"/>
  </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:restriction>
</xs:simpleType>

```

```
    <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="lscip:settingTypeType"
    use="required" />
  <xs:attribute name="value_type" type="lscip:settingValueType"
    use="required" />
  <xs:attribute name="conn_mod" type="xs:string" use="required" />
  <xs:attribute name="domain" type="xs:string" use="optional" />
  <xs:attribute name="default" type="lscip:settingDefaultType"
    use="optional" />
  <xs:attribute name="value_expr" type="lscip:scriptType"
    use="optional" />
  <xs:attribute name="drc" type="lscip:scriptType" use="optional" />
  <xs:attribute name="editable" type="lscip: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="lscip:scriptType"
```

```

        use="optional" />
        <xs:attribute name="value_range" type="lscip: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="lscip:scriptType"
            use="optional" />
        <xs:attribute name="output_formatter"
            type="lscip:settingOutputFormatterType" use="optional" />
        <xs:attribute name="config_groups" type="lscip:scriptType"
            use="optional" />
        <xs:attribute name="process_path" type="xs:boolean" use="optional"
            default="true">
        </xs:attribute>
        <xs:attribute name="macro_name" type="lscip:settingMacroNameType"
            use="optional"
            default="id">
        </xs:attribute>
        <xs:attribute ref="xml:base" use="optional"/>
        <xs:attribute name="no_dependency" type="xs:string" use="optional" />
    </xs:complexType>

    <!-- ip.settings -->
    <xs:complexType name="settingsType">
        <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" name="setting"
                type="lscip:settingElementType" />
        </xs:sequence>
        <xs:attribute ref="xml:base" use="optional"/>
    </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="lscip:scriptType"
            use="optional" />
        <xs:attribute name="stick_high" type="lscip:scriptType"
            use="optional" />
        <xs:attribute name="stick_low" type="lscip:scriptType"
            use="optional" />
        <xs:attribute name="stick_value" type="lscip:scriptType"
            use="optional" />
        <xs:attribute name="dangling" type="lscip:scriptType"
            use="optional" />
        <xs:attribute name="bus_interface" type="xs:string"

```

```

        use="optional" />
<xs:attribute name="attribute" type="lscip: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:attribute ref="xml:base" use="optional"/>
</xs:complexType>
<!-- ip.outFileConfigs -->
<xs:complexType name="outFileConfigsType">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" name="fileConfig" type="lscip: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="lscip:ipany.att"/>
</xs:complexType>
<!-- ip.interface-->
    
```

```

<xs:complexType name="portsType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="1" name="port"
type="lscsip:portElementType" />
  </xs:sequence>
  <xs:attribute ref="xml:base" use="optional"/>
</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="lscsip:twoFigureVersionType" />
    <xs:element maxOccurs="1" minOccurs="0"
      name="max_radiant_version" type="lscsip:twoFigureVersionType" />
    <xs:element maxOccurs="1" minOccurs="0" name="min_esi_version"
      type="lscsip:twoFigureVersionType" />
    <xs:element maxOccurs="1" minOccurs="0" name="max_esi_version"
      type="lscsip:twoFigureVersionType" />
    <xs:element maxOccurs="1" minOccurs="1"
      name="supported_products" type="lscsip:supportedProductsType" />
    <xs:element maxOccurs="1" minOccurs="0"
      name="supported_platforms" type="lscsip:supportedPlatformsType" />
  </xs:sequence>
</xs:complexType>

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

<xs:complexType name="supportedPlatformType">

```

```
<xs:sequence>
  <xs:element maxOccurs="unbounded" minOccurs="0" name="supported_templates"
type="lscip:supportedTemplatesType" />
</xs:sequence>
<xs:attribute name="name" type="xs:string"/>
</xs:complexType>

<xs:complexType name="supportedTemplatesType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="supported_template"
type="lscip:supportedTemplateType" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="supportedTemplateType">
  <xs:attribute name="processor" type="xs:string"/>
  <xs:attribute name="family" type="xs:string"/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="version" type="xs:string" use="optional"/>
</xs:complexType>

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

<xs:complexType name="supportedFamilyType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="supported_device"
type="lscip: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="lscip: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="lscip: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="lscchip: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="lscchip:generalType" />
      <xs:element name="settings" type="lscchip:settingsType" />
      <xs:element name="ports" type="lscchip:portsType" />
      <xs:element name="outFileConfigs" type="lscchip:outFileConfigsType" maxOccurs="1"
minOccurs="0">
        <xs:unique name="fileCfgKey">
          <xs:selector xpath="lscchip:fileConfig"/>
          <xs:field xpath="@name"/>
        </xs:unique>
      </xs:element>
      <xs:element ref="lscchip:busInterfaces" minOccurs="0" />
      <xs:element ref="lscchip:addressSpaces" minOccurs="0" />
      <xs:element ref="lscchip:memoryMaps" minOccurs="0" />
      <xs:element ref="lscchip:componentGenerators" minOccurs="0" />
      <xs:element ref="lscchip:choices" minOccurs="0" />
      <xs:element ref="lscchip:fileSets" minOccurs="0" />
      <xs:element ref="lscchip:design" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="estimatedResources"
        type="lscchip:estimatedResourcesType" minOccurs="0" />
      <xs:element ref="lscchip:parameters" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="version" type="lscchip:twoFigureVersionType"/>
    <xs:attribute name="platform" type="xs:string" use="optional"/>
  </xs:complexType>

```



```
        <xs:attribute name="platform_version" type="lscip:twoFigureVersionType"  
use="optional"/>  
    </xs:complexType>  
</xs:element>  
  
</xs:schema>
```

References

[Lattice Propel 2023.1 Builder User Guide \(FPGA-UG-02185\)](#)

[Lattice Diamond Online Help](#)

[Lattice Radiant Online Help](#)

Technical Support Assistance

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

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

Revision History

Revision 1.0, May 2023

Section	Change Summary
All	Production release.



www.latticesemi.com