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
Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1.</td>
<td>Details of Parameter Property</td>
<td>20</td>
</tr>
<tr>
<td>Table 2.2.</td>
<td>Details of Port Property</td>
<td>24</td>
</tr>
<tr>
<td>Table 2.3.</td>
<td>Child Nodes of General Node</td>
<td>37</td>
</tr>
<tr>
<td>Table 2.4.</td>
<td>Attributes of Setting Nodes</td>
<td>37</td>
</tr>
<tr>
<td>Table 2.5.</td>
<td>Attributes of Port Nodes</td>
<td>40</td>
</tr>
<tr>
<td>Table 2.6.</td>
<td>Elements in estimatedResources Node</td>
<td>44</td>
</tr>
</tbody>
</table>

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 Prop 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
## Glossary

A list of words or terms specialized in this document.

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

```plaintext
- [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.

```plaintext
- <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](image)

2. The IP Packager window pops up, as shown in Figure 2.4.

![Figure 2.4. IP Packager GUI](image)

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).

![Select Folder Dialog]

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

1. 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](image)

a. To configure basic information:
   - Click Basic Info from the Meta Data view (Figure 2.8).
You can see the Basic Info properties in detail, as shown in Figure 2.9.

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).
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).
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](image)

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

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

![Parameter Value Interface](image)

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

![Parameter Selection Interface](image)
### Table 2.1. Details of Parameter Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>String</td>
<td>No</td>
<td>Short title of the setting. If title is not specified, value of the setting is used. Example: title=&quot;Device Architecture&quot;</td>
</tr>
<tr>
<td>type</td>
<td>param, input,</td>
<td>Yes</td>
<td>Type of the setting. A parameter could be a Verilog parameter, user input, command or verilog_macro. Param, input and verilog_macro 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 of the IP module; verilog_macro is translated to a Verilog macro definition by the define compiler directives; while a parameter with the type command is shown as a button. Example: type=&quot;param&quot;</td>
</tr>
<tr>
<td>value_type</td>
<td>bool, string,</td>
<td>Yes</td>
<td>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 the 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=&quot;string&quot;</td>
</tr>
<tr>
<td>default</td>
<td>Python expression</td>
<td>No</td>
<td>Default value of the setting. If the setting has no default attribute but has the options attribute, the first option is picked as the default value. If the setting has neither default attribute nor options attribute, the initial value of the 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=&quot;LIFCL&quot;</td>
</tr>
<tr>
<td>value_expr</td>
<td>Python expression</td>
<td>No</td>
<td>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=&quot;int(round((sys_clock_freq * 250.0) / i2c_left_desired_frequency))-1&quot;</td>
</tr>
<tr>
<td>options</td>
<td>Python list or</td>
<td>No</td>
<td>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=&quot;[0.1, 0.2, 0.5, 1.0]&quot;</td>
</tr>
<tr>
<td></td>
<td>list of tuples</td>
<td></td>
<td></td>
</tr>
<tr>
<td>output_formatter</td>
<td>str</td>
<td>No</td>
<td>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=&quot;str&quot;</td>
</tr>
<tr>
<td>bool_value_mapping</td>
<td>Python tuple or</td>
<td>No</td>
<td>The map-to-map bool values to dedicated strings. By default, bool values are written as 1, 0. Example: bool_value_mapping=&quot;('True', 'False')&quot;</td>
</tr>
<tr>
<td></td>
<td>list with 2 string elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>editable</td>
<td>Python expression</td>
<td>No</td>
<td>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 value_expr. Otherwise, the user input is used.</td>
</tr>
</tbody>
</table>

© 2023 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden</td>
<td>True</td>
<td>No</td>
<td>Python expression to determine whether the setting is hidden in the GUI. If hidden is set to True (default is False), the item is hidden in the GUI. The expression is resolved to boolean value after the user setting is changed. Example: hidden=&quot;True&quot;</td>
</tr>
<tr>
<td>drc</td>
<td>Python expression</td>
<td>No</td>
<td>Python expression to do DRC on the setting. True means DRC pass. Example: drc=&quot;check_valid_addr_pre(I2C_LEFT_ADDRESSING_PRE,i2c_left_addressing_width)&quot; (check_valid_addr_pre is defined in plugin.py setting ID: I2C_LEFT_ADDRESSING_PRE i2c_left_addressing_width)</td>
</tr>
<tr>
<td>regex</td>
<td>Regular expression</td>
<td>No</td>
<td>Regular expression to do DRC on the value. For example, the value should be prefixed by 0b. Example: regex=&quot;0b[01]+&quot;</td>
</tr>
<tr>
<td>value_range</td>
<td>Python tuple or list with 2 comparable elements</td>
<td>No</td>
<td>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=&quot;(0, 1023) if(i2c_right_enable) else (-9999, 9999)&quot;</td>
</tr>
<tr>
<td>config_groups</td>
<td>Python expression</td>
<td>No</td>
<td>A name or names to group settings together. It is copied from the IP-XACT configGroups attribute, and only supports the SystemBuilder 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.</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>No</td>
<td>Detailed description of the setting.</td>
</tr>
<tr>
<td>Macro_name</td>
<td>id, value</td>
<td>No</td>
<td>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 setting, 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= &quot;value&quot; Result: `define &lt;setting value&gt; Note: This is only applicable to the setting item whose value_type attribute is set to string.</td>
</tr>
</tbody>
</table>

- 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](image)

  - 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](image)

- 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.
**Table 2.2. Details of Port Property**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>Python tuple or list with two-integer elements</td>
<td>No</td>
<td>Range of this port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: range=&quot;(A_WDT-1, 0)&quot; (A_WDT is a setting ID)</td>
</tr>
<tr>
<td>conn_port</td>
<td>Valid Verilog module name</td>
<td>No</td>
<td>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=&quot;Clk&quot;</td>
</tr>
<tr>
<td>conn_range</td>
<td>Python tuple or list with two-integer elements</td>
<td>No</td>
<td>Range of conn_port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: conn_range=&quot;(A_WDT-1, 0)&quot; (A_WDT is a setting ID)</td>
</tr>
<tr>
<td>stick_high</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: tie this port to 1. Example: stick_high=&quot;True&quot;</td>
</tr>
<tr>
<td>stick_low</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: tie this port to 0. Example: stick_low=&quot;no_seq_pins()&quot; (no_seq_pins is defined in plugin.py)</td>
</tr>
<tr>
<td>stick_value</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. Tie port to the evaluation result of this attribute.</td>
</tr>
<tr>
<td>Dangling</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: keep this port unconnected. Example: dangling=&quot;not USE_COUT&quot; (USE_COUT is a setting ID)</td>
</tr>
<tr>
<td>attribute</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. The value can be written to the .v file as the attribute of the port.</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
<td>Mandatory</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example: attribute=&quot;&quot;(* AAA, BBB=1 <em>)&quot;&quot; =&gt; (</em> AAA, BBB=1*) input PORT;</td>
</tr>
<tr>
<td>port_type</td>
<td>String</td>
<td>No</td>
<td>Data, reset and clock are valid values. The default value is data. Port_type can be passed to the IPXact component.xml as a lsscip:isClk or lsscip:isRst node in vendorExtensions in component/model/ports/port.</td>
</tr>
</tbody>
</table>

- 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](image)

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

![Figure 2.26. Rename Interface](image)

- 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](image)

![Figure 2.29. Generate New Memory Map](image)
2. 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](image)

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

**Note:** 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 2.34). Configure Misc information as desired.

Figure 2.34. IP Packager with Misc Details
5. From the IP Packager Project area, click **Doc Assistant**. The Doc Assistant information is shown in detail (Figure 2.35).

![IP Packager with Doc Assistant Details](image)

**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](image)

![Figure 2.38. Error Message Pops Up in IP Packager](image)
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](image)

**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 “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 2.40).

```
    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 2.40. Example XML of Metadata Layout](image)

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

<table>
<thead>
<tr>
<th>Child Node</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor</td>
<td>Yes</td>
<td>Soft IP vendor. An official soft IP should have the vendor name. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“latticesemi.com”</td>
</tr>
<tr>
<td>library</td>
<td>Yes</td>
<td>Library of the soft IP. If library is not set, the default value ip is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: ip, interface</td>
</tr>
<tr>
<td>name</td>
<td>Yes</td>
<td>Name of the soft IP. The name must be unique among soft IP of the same vendor and library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: adder</td>
</tr>
<tr>
<td>display_name</td>
<td>No</td>
<td>Name to be displayed in the software. If display_name is not set, software displays name directly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: Adder</td>
</tr>
<tr>
<td>version</td>
<td>Yes</td>
<td>Version of the soft IP. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0.0</td>
</tr>
<tr>
<td>category</td>
<td>Yes</td>
<td>Category of the soft IP. Category can be hierarchical. Levels are separated by “.”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: Memory_Modules,Distributed_RAM</td>
</tr>
<tr>
<td>keywords</td>
<td>No</td>
<td>Keywords of the soft IP. Multiple keywords are separated by “;”. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BusType_AHB,BusType_APB</td>
</tr>
<tr>
<td>min_radiant_version</td>
<td>Yes</td>
<td>The minimal Radiant version, which supports the soft IP. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0 (no service pack), 1.0.1 (with service pack)</td>
</tr>
<tr>
<td>max_radiant_version</td>
<td>No</td>
<td>The maximal Radiant version, which supports the soft IP. Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0</td>
</tr>
<tr>
<td>supported_products</td>
<td>No</td>
<td>FPGA products supported by the soft IP.</td>
</tr>
<tr>
<td>Type</td>
<td>No</td>
<td>Enhancement for cpu IP</td>
</tr>
<tr>
<td>Min esi_version</td>
<td>No</td>
<td>The minimal ESI version, which supports the soft IP. Enhancement for esi.</td>
</tr>
<tr>
<td>Max esi_version</td>
<td>No</td>
<td>The maximal ESI version, which supports the soft IP. Enhancement for esi.</td>
</tr>
<tr>
<td>Supported Platforms</td>
<td>No</td>
<td>Default is Radiant, specific for esi.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Yes</td>
<td>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</td>
</tr>
</tbody>
</table>

© 2023 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal. All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>letter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: id=&quot;num_outputs&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>String</td>
<td>No</td>
<td>Short title of the setting. If title is not specified, the value of setting is used.</td>
</tr>
<tr>
<td></td>
<td>Example: title=&quot;Number of Output&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>param, input, command, verilog_macro</td>
<td>Yes</td>
<td>Type of the setting. A parameter could be a Verilog parameter, user input, command or verilog_macro. Param, input and verilog_macro 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 of the IP module, verilog_macro is translated to a Verilog macro definition by the define compiler directives, while a parameter with the type of command is shown as a button. Example: type=&quot;param&quot;</td>
</tr>
<tr>
<td>value_type</td>
<td>bool, string, int, float, path</td>
<td>Yes</td>
<td>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=&quot;int&quot;</td>
</tr>
<tr>
<td>default</td>
<td>Python expression</td>
<td>No</td>
<td>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=&quot;1.0&quot;</td>
</tr>
<tr>
<td>value_expr</td>
<td>Python expression</td>
<td>No</td>
<td>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=&quot;int(round((sys_clock_freq * 250.0) / i2c_left_desired_frequency))-1&quot;</td>
</tr>
<tr>
<td>options</td>
<td>Python list or list of tuples</td>
<td>No</td>
<td>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=&quot;[0.1, 0.2, 0.5, 1.0]&quot;</td>
</tr>
<tr>
<td>output_formatter</td>
<td>str</td>
<td>No</td>
<td>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=&quot;str&quot;</td>
</tr>
<tr>
<td>bool_value_mapping</td>
<td>Python tuple or list with 2 string elements</td>
<td>No</td>
<td>The map-to-map bool values to dedicated strings. By default, bool values are written as 1, 0. Example: bool_value_mapping=&quot;('True', 'False')&quot;</td>
</tr>
<tr>
<td>editable</td>
<td>Python</td>
<td>No</td>
<td>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</td>
</tr>
<tr>
<td>Attribute</td>
<td>Value</td>
<td>Mandatory</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>attribute</td>
<td>value_expr</td>
<td>Yes</td>
<td>value_expr. Otherwise, the user input is used. Example: editable=&quot;(FEEDBACK_PATH == 'PHASE_AND_DELAY')  (FEEDBACK_PATH is a setting ID in metadata.xml)</td>
</tr>
<tr>
<td>hidden</td>
<td>True</td>
<td>No</td>
<td>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=&quot;True&quot;</td>
</tr>
<tr>
<td>drc</td>
<td>Python</td>
<td>No</td>
<td>Python expression to do DRC on the setting. True means DRC pass. Example: drc=&quot;check_valid_addr_pre(I2C_LEFT_ADDRESSING_PRE,i2c_left_addressing_width)&quot;(check_valid_addr_pre is defined in plugin.py setting ID: I2C_LEFT_ADDRESSING_PRE i2c_left_addressing_width)</td>
</tr>
<tr>
<td>regex</td>
<td>Regular</td>
<td>No</td>
<td>Regular expression to do DRC on the value. For example, the value should be prefixed by 0b. Example: regex=&quot;0b[01]+&quot;</td>
</tr>
<tr>
<td>value_range</td>
<td>Python tuple</td>
<td>No</td>
<td>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=&quot;(0, 1023) if(i2c_right_enable) else (-9999, 9999)&quot;</td>
</tr>
<tr>
<td>config_groups</td>
<td>Python</td>
<td>No</td>
<td>A name or names to group settings together. It is copied from the IP-XACT configGroups attribute, and only supports the 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.</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>No</td>
<td>Detailed description of the setting.</td>
</tr>
<tr>
<td>Group1</td>
<td>String</td>
<td>No</td>
<td>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= &quot;Output Setting&quot;</td>
</tr>
<tr>
<td>group2</td>
<td>String</td>
<td>No</td>
<td>Group the group1 groups. Group1 groups of the same group2 is displayed in a separate page in the GUI. So, a two-level hierarchy is supported in GUI display. 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.</td>
</tr>
<tr>
<td>Macro_name</td>
<td>id, value</td>
<td>No</td>
<td>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 setting, 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=&quot;value&quot; Result: `define &lt;setting value&gt; Note: This is only be considered in the setting item whose value_type attribute is set to string.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Valid Verilog port name</td>
<td>Yes</td>
<td>Name of a port. Example: name=&quot;Clk&quot;</td>
</tr>
<tr>
<td>dir</td>
<td>in, out, inout</td>
<td>Yes</td>
<td>Direction of a port. Example: dir=&quot;in&quot;</td>
</tr>
<tr>
<td>range</td>
<td>Python tuple or list with 2 int elements</td>
<td>No</td>
<td>Range of this port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: range=&quot;(A_WDT-1, 0)&quot; (A_WDT is a setting ID)</td>
</tr>
<tr>
<td>conn_mod</td>
<td>Valid Verilog module name</td>
<td>Yes</td>
<td>Name of an IP core module to which this port connects. Example: conn_mod=&quot;counter&quot;</td>
</tr>
<tr>
<td>conn_port</td>
<td>Valid Verilog module name</td>
<td>No</td>
<td>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=&quot;Clk&quot;</td>
</tr>
<tr>
<td>conn_range</td>
<td>Python tuple or list with 2 int elements</td>
<td>No</td>
<td>Range of conn_port. It should be a Python expression whose evaluation result is a tuple or array with two elements. Example: conn_range=&quot;(A_WDT-1, 0)&quot; (A_WDT is a setting ID)</td>
</tr>
<tr>
<td>stick_high</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: tie this port to 1. Example: stick_high=&quot;True&quot;</td>
</tr>
<tr>
<td>stick_low</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: tie this port to 0. Example: stick_low=&quot;no_seq_pins()&quot; (no_seq_pins is defined in plugin.py)</td>
</tr>
<tr>
<td>stick_value</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. Tie port to the evaluation result of this attribute.</td>
</tr>
<tr>
<td>Dangling</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. True: keep this port unconnected. Example: dangling=&quot;not USE_COUT&quot; (USE_COUT is a setting ID)</td>
</tr>
<tr>
<td>bus_interface</td>
<td>Valid bus interface name</td>
<td>No</td>
<td>Bus interface name defined in &lt;busInterfaces&gt; node. Example: bus_interface=&quot;ahb_slave_0&quot;</td>
</tr>
<tr>
<td>attribute</td>
<td>Python expression</td>
<td>No</td>
<td>Python script. The value is written to the .v file as the attribute of the port. Example: attribute=&quot;&quot;(* AAA, BBB=1 <em>)&quot;&quot; =&gt; (</em> AAA, BBB=1*) input PORT;</td>
</tr>
<tr>
<td>port_type</td>
<td>String</td>
<td>No</td>
<td>Data, reset and clock are valid values. The default value is data. Port_type is passed to the IPXact component.xml as a lsccip:isClk or lssccip:isRst node in vendorExtensions in component/model/ports/port.</td>
</tr>
</tbody>
</table>
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.

```
<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 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.

```
<lsccip:addressSpaces>
  <lsccip:addressSpace>
    <lsccip:name>ahbm_addr_space</lsccip:name>
    <lsccip:range>4k</lsccip:range>
    <lsccip:width>32</lsccip:width>
  </lsccip:addressSpace>
</lsccip: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.
<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:field>
      </lsccip:register>
    </lsccip:addressBlock>
  </lsccip:memoryMap>
</lsccip:memoryMaps>

Figure 2.43. 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 2.44).

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

**Figure 2.44. Example of componentGenerators 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 2.6 shows the element in `<estimatedResource>` node.

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Yes</td>
<td>Name of the resource.</td>
</tr>
<tr>
<td>Number</td>
<td>Python expression</td>
<td>Yes</td>
<td>Python script to calculate the number of the resource.</td>
</tr>
</tbody>
</table>

9. `< 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
<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: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 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).

```xml
<lscip:outFileConfigs>
  <lsccip:fileConfig name="wrapper" lib='test2.vhd=libA;test3.vhd=libB' />
</lscip: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.
```python
def cntr_opt():
    return {"Down": 0,
            "Up": 1,
            "UpDown": 2}

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

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

def cntr_bval_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.

![Example of Memory Map CSV File](image)

Figure 2.48. Example of Memory Map CSV File
Appendix A. metadata.xsd

```xml
<?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: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: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="scriptIdType">
    <xs:annotation>
        <xs:documentation>
            Any python expressions. Could be a simple const variable like "1", 
or a tuple 
            "(0, 10)", or a function call "user_func1()",
or a complex expression "a==12";
        </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 "input", "verilog_macro", and "command".
        </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 "id" and "value".
        </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 "bool", 
            "string" and "int";
        </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="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="domain" type="xs:string" use="optional" />
  <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: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:complexType>
<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>
      Merge the new attributes to existing configuration or replace all
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="merge"/>
        <xs:enumeration value="replace"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="type" use="optional" default="file">
    <xs:annotation>
      Indicate the item is generate a file or directory
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="file"/>
        <xs:enumeration value="directory"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```xml
<xs:simpleType>
    <xs:attribute name="description" type="xs:string" use="optional"/>
    <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:integer" use="optional" default="0"/>
    <xs:attribute name="file_base_name" type="xs:string" use="optional"/>
    <xs:attribute name="file_suffix" type="xs:string" use="optional"/>
    <xs:attribute name="sub_dir" type="xs:string" use="optional"/>
    <xs:attribute name="file_generator" type="xs:string" use="optional"/>
    <xs:attribute name="src_dir" type="xs:string" use="optional"/>
    <xs:attribute name="dest_dir" type="xs:string" use="optional"/>
    <xs:attribute name="src_file" type="xs:string" use="optional"/>
    <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:attribute>
</xs:simpleType>
```

<xs:complexType name="portsType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="1" name="port" type="lsccip: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="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="0" 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:sequence>
  </xs:sequence>
</xs:complexType>
<xs:sequence>
  <xs:element minOccurs="0" maxOccurs="unbounded" name="supported_templates" type="lsccip:supportedTemplatesType" />
</xs:sequence>
<xs:attribute name="name" type="xs:string" />
</xs:complexType>
<xs:complexType name="supportedTemplatesType">
  <xs:element minOccurs="0" maxOccurs="unbounded" name="supported_template" type="lsccip: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 minOccurs="1" maxOccurs="unbounded" name="supported_family" type="lsccip:supportedFamilyType" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="supportedFamilyType">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" 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 minOccurs="0" maxOccurs="unbounded" 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 minOccurs="0" maxOccurs="unbounded" 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" use="optional"/>
    <xs:attribute name="platform" type="xs:string" use="optional"/>
  </xs:complexType>
<xs:attribute name="platform_version" type="lsccip: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**

<table>
<thead>
<tr>
<th>Section</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Production release.</td>
</tr>
</tbody>
</table>