Lattice Radiant Software IP User Guide

April 23, 2019

Copyright

Trademarks

Disclaimers

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
`.` `.` `.`	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

Lattice Radiant Software IP User Guide

Lattice provides pre-tested, reusable intellectual property (IP) functions. These proven IP cores are optimized for Lattice device architecture, resulting in fast, small cores that use the latest Lattice architectures to their fullest.

Lattice Radiant software provides enhanced IP functionality, which is described in this user guide. Major topics include:

Designing with IP and PMI

IP are functional bits of design that can be re-used wherever that function is needed. Creating such components with hardware design languages is common practice. To help your design along, the Radiant software provides a variety of components for common functions. They are optimized for Radiant software device architectures and can be customized. Use these components to speed up your design work and to get the most effective results.

Radiant software components come in a variety of forms:

- Modules: These basic, configurable blocks come with IP Catalog. They provide a variety of functions including I/O, arithmetic, memory, and more. Open IP Catalog to see the full list of what's available.
- IP: Intellectual property (IP) are more complex, configurable blocks. They are accessible through IP Catalog, but they do not come with the tool. They must first be downloaded and installed as a separate step before they can be accessed from IP Catalog. To see all that's available and to learn about licensing and other vendors of IP, go to the Lattice website: www.latticesemi.com/ip.

- PMI (Parameterized Module Instantiation) is an alternate way to use some of the components that come with IP Catalog. Instead of using IP Catalog, PMI can directly instantiate a component into your HDL and customize it by setting parameters in the HDL. You may find this easier than using IP Catalog if your design requires many variations of the same component. To decide which method to use, see "PMI or IP Catalog?" on page 6.
- Reference designs provide you with a starting point on creating your own components. Lattice Reference Designs are available in Verilog and VHDL, and can be downloaded from the Lattice website: www.latticesemi.com/ip.
- Lattice library primitives are very basic functions, such as logic gates and flip-flops. They can be directly instantiated as HDL into designs. But this is an advanced technique and should usually be avoided.

IP Catalog provides a variety of functions ranging from the most basic, such as arithmetic and memory, to much more complex functions. With IP Catalog these components can be extensively customized and created as part of a specific project or as a library for multiple projects. See "Creating IP Catalog Components" on page 7.

However, many of these components can also be used with PMI (see next item). To decide which method to use, see "PMI or IP Catalog?" on page 6.

PMI or IP Catalog?

Many IP Catalog components are also available as PMI components, but PMI doesn't offer error checking. This and the next two sections discuss the pros and cons of both options to help you decide which is best for your project.

PMI is a convenient way to use components of the same type but that vary from instance to instance. This eliminates the need to create a separate component for each instance using IP Catalog.

For example, a design might require dozens of FIFOs that are functionally the same but require different address depths. With PMI, you could insert the same FIFO instantiation command wherever it's needed in the HDL and just change the address depth parameter as you go. The alternative would be to use IP Catalog to generate different components for each variation and then insert the instantiation template for each of them. In this situation, you might find the PMI method easier and faster.

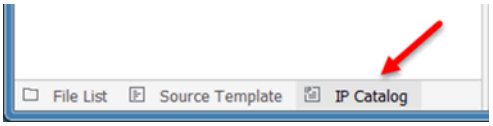
Before deciding whether to use a PMI component, compare it to the equivalent IP Catalog component. Often IP Catalog provides more options than PMI does. IP Catalog also assures that all of your option selections and parameter settings are legal. PMI offers no error checking.

Creating IP Catalog Components

IP Catalog is an easy way to use a collection of functional blocks from the Radiant software. There are two types of components available through IP Catalog: modules and IP. IP Catalog enables you to extensively customize these components. They can be created as part of a specific project or as a library for multiple projects.

Below are the basic steps of using IP Catalog IP. For details of performing these steps, see the following topics.

1. Open IP Catalog. IP Catalog is accessed via a tab at the lower left of the Radiant software. Click the tab, or click in the tool bar, to view the list of available components.



2. Customize the component. These IP can be extensively customized for your design. The options range from setting the width of a data bus to selecting features in a communications protocol. At a minimum you need to specify the design language to use for the output.
3. Generate the component and bring its .ipx file into your project. Prior to generating the component, select the option “Insert to project.” This will automatically bring the .ipx file into your project after the generation step completes. If you do not select this option, add the .ipx file to your project as you would with any other source file (such as a Verilog or VHDL file) after the generation is complete.
4. Instantiate the component into the project’s design. An HDL instance template is generated during the generation step to simplify this step.
5. IP Catalog IP can be further modified or updated later. After the .ipx file has been added to the Radiant software project, it is visible in the project’s file list. Double-clicking the .ipx file brings up the component’s configuration dialog box where changes can be made and the generation process repeated.

Downloading IP from IP on Server

You can use the IP on Server tab of IP Catalog to download and install the latest IP available from Lattice Semiconductor. Before you can download any IP, you need to set up an Internet connection. If you haven’t already, choose **Tools > Options > General > Network Settings** and fill in the dialog box.

The website also has links to other vendors of IP. To see all that’s available and to learn about licensing and other vendors of IP, go to the Lattice website: www.latticesemi.com/ip.

To download Lattice IP on Server:

1. In IP Catalog, click the **IP on Server** tab, located at the upper-left of the tool. The software connects to the Lattice Semiconductor website.

2. Click the refresh icon to update the list.
3. Expand the folder tree and select the IP you want to download.

   Information about the IP appears in the right pane including links for additional information.

   IP that are compatible with your selected device have icons highlighted in dark blue. IP that are not compatible with your version of Radiant software are have blue icons with a yellow triangle. Look for device support information in the right pane.
4. To download the IP, click the IP and choose to the right of the IP name in the list.
5. The downloaded IP is now added to the IP list in the **IP on Local** tab.

Generating a Component with IP Catalog

Use IP Catalog to generate a customized functional block from any component or installed IP.

To generate a component:

1. In the Module/IP tree, select the component that you want to generate.
2. To get more information about the component, click the **IP Information** tab.
3. Double-click the Module/IP or click the Generate Module/IP Instance button, located at the upper-left of the tool the In the Configuration tab. Specify general project information and the base file name for the component.
   - Instance Name is the base name for the component's files (that is, with no extension).
   - Create In is the location for the customized component's files.
4. Click **Next.**

   The component's dialog box opens.
5. In the dialog box, select your desired options.Click **Generate**.
6. To automatically import the .ipx file into your project when the component is generated, select **Insert to project** in the Check Generating Result dialog box.
7. Click **Finish**.

IP Catalog Output Files for Components

IP Catalog creates the following output files for components under the specified Project Path. The <*file_name*> comes from the File Name specified in the Configuration tab.

IP Catalog creates some different files for IP. These are documented in the IP's associated user guide.

Table 1: Output Files

File Name	Description
<*file_name*>.ipx	Manifest file. This file is loaded into the design project so that modifications can be made to the component.
<*file_name*>_tmpl.v	Instantiation template for Verilog netlist.
file_name>_tmpl.vhd	Instantiation template for VHDL netlist
<*file_name*>.v	Verilog HDL file for both synthesis and simulation. Verilog output files declare implicit wire types.
tb_top.v	Testbench for associated component.

Importing an IP Catalog Component into a Project

After generating component source files using IP Catalog, you can import the component by importing the IP Catalog manifest file (.ipx). Components have several files of different types, but you only need to import the .ipx. The .ipx file identifies the components needed to make up the component.

Importing the files may not be necessary if the "Import IPX to project" option was selected when the component was generated.

To import a component:

1. In the File List view, right-click the implementation folder ( ) and choose **Add > Existing File**.
2. Browse for the customized component's .ipx file, <*file_name*>**.ipx**, and select it.
3. Click **Add**.

   The .ipx file is added to the File List view.
4. Check the Output Panel for error messages. If the component is not targeted for the current device, try double-clicking the file to regenerate the component.

After importing the component, you can further customize it for your design project by changing options specific to the component. You can also update older components or IP to the latest version.

Instantiating an IP Catalog Component

IP Catalog components are instantiated the same way other components are in your HDL. When you generate components in IP Catalog, the tool also generates Verilog and VHDL reference templates. You can instantiate the reference templates for implementation or simulation flow of any design, as needed.

The instantiation file is located in the folder in which the component was created. The file name is based on the component's name: <*component_name*>_tmpl.v or <*component_name*>_tmpl.vhd.

You can instantiate the IP Catalog component in one of the following ways:

- If you selected the option of not inserting the generated IP into your project, browse to the <*component_name*>_tmpl.v or <*component_name*>_tmpl.vhd, open the file in a text editor, copy the text, and paste into your source file. Then, add an instance and signal names to the component ports.
- If you selected the option of inserting the generated IP into your project, the IP component configuration file (.ipx) appears in your design implementation.

  In the File List, right-click the .ipx file, and choose the instantiation command:

  - For Verilog, choose **Copy Verilog Instantiation**, and paste it into your source file. Then, add an instance and signal names to the component ports.
  - For VHDL, choose **Copy VHDL Component**, and paste it into your source file; then choose **Copy VHDL Instantiation**, and paste it into your source file. Then, add an instance and signal names to the component ports.

Instantiating a PMI Component

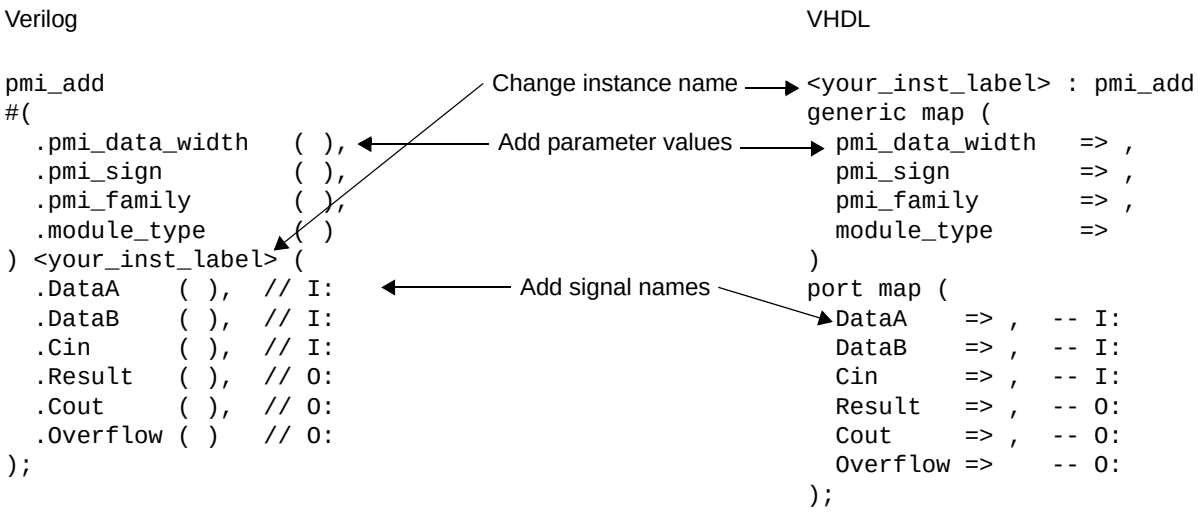
PMI components are instantiated the same way other components are in your HDL. The Radiant software provides a template for the Verilog or VHDL instantiation command that specifies the customized component's ports and parameters. Customize the component by changing its parameters.

To instantiate a PMI component:

1. With Source Editor, open the source file that will receive the component.
2. Place the cursor in the desired spot for the instantiation command.
3. In the Source Template tab, open the **Verilog > PMI Templates** or **VHDL > PMI Templates** folder and select the component.

   The code of the template appears in the lower box of the Source Template tab.
4. Right-click the template code and choose **Select All**.
5. Drag and drop the text into your source file.
6. Add an instance name, set parameter values, and add signal names to the corresponding component ports in the instantiation command, as shown in Figure 1 on page 11.

Figure 1: PMI Instantiation Command Example



7. If using stand-alone synthesis and simulation tools, add the PMI soft IP from the <*install_path*>/ip/pmi/ directory.
8. Save and close the source file.

Encrypting IPs

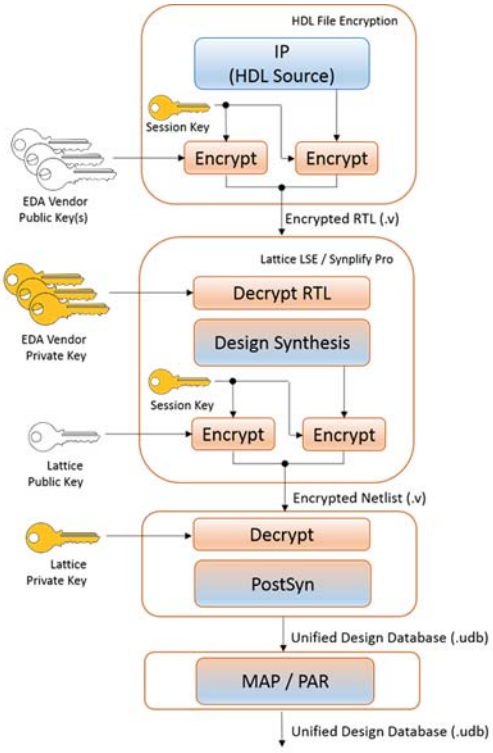
The Radiant software provides the IP security based on the Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP) of the IEEE 1735-2014 version 1 standard.

The implementation enables securing IP while ensuring the design flow and the interoperability across different EDA tools. Upon entering the design into the Radiant software, you select the desired level of security by encrypting the HDL source files. The set security level propagates through the design and impacts the visibility of secured objects in various Radiant tools. The current Radiant software supports encryption of VHDL and Verilog HDL files. The encrypted IP can be processed by other third-party EDA tools including Aldec Active-HDL, Cadence NCSim, Mentor Graphics ModelSim, Synopsys Synplify Pro, and Synopsys VCS simulator.

IP Encryption Flow

IP encryption flow enables you to protect your IP design. Following the industry standard, the Radiant software, through the IP encryption flow, allows the partnership between the IP Vendor, supported EDA vendor, and Lattice.

Figure 2: IP Encryption Flow



The encryption flow uses symmetric and asymmetric encryption methods to maximize the design security. The symmetric method only involves a single symmetric key for both, encryption and decryption. The asymmetric method involves the public-private key pair. The public key is published by a vendor and is used by the Radiant software. The private key is never revealed to the public.

The Radiant software supports these cryptographic algorithms:

- AES-128/AES-256: symmetric algorithm used to encrypt the content of the HDL source file.
- RSA-2048: asymmetric algorithm used to obfuscate a key used in HDL file encryption. The RSA is defined by the public-private key pair. You must be familiar with both keys in order to perform RSA decryption.

HDL File Encryption Flow

The overall HDL file encryption flow is summarized in these steps:

- The source file of the IP design is AES encrypted using a symmetric session key. The AES encryption uses the CBC-128 or CBC-256 algorithm. In the source files, this section is referred to as a data block.
- The session key is RSA encrypted using the vendor's public key. In the source files, this section is referred to as a key block. Multiple key blocks may be present in the source file.
- The encrypted session key and the encrypted design are merged to file generally named the Encrypted RTL

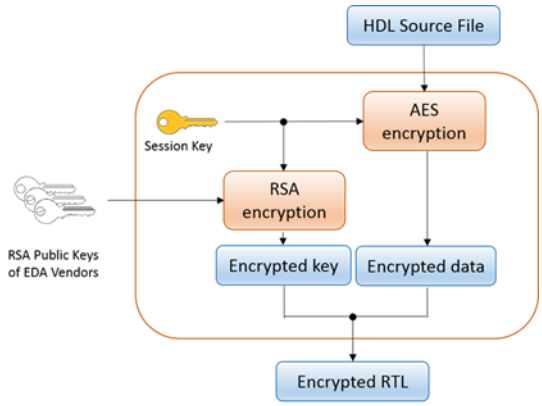
Each encrypted source file contains a single data block and one or more key blocks. The number of key blocks depends on the number of provided public keys.

NOTE

To decrypt an encrypted source file, you must perform the IP encryption flow steps in the reverse order.

During the next step in the design flow, typically synthesis, the Encrypted RTL is decrypted to access the original IP design, as shown in the following figure.

Figure 3: HDL Encryption Flow



By separating the encryption of data and key, you can use public keys from different vendors to encrypt the same HDL file.

For specific steps and information on how to encrypt HDL files in the Radiant software, refer to the following section in the Radiant software online help: **User Guides > Securing the Design**.

Packaging IP Using Radiant IP Packager

Radiant IP Packager allows external Intellectual Property (IP) developers — including third-party IP providers and customers — to prepare and package IP in the Radiant IP format.

IP packages must contain other file types, including:

- Metadata (.xml)
- RTL (<*IP_name*>.v and v.)
- Documentation (introduction.html, user guide, etc).

IP packages can also contain certain non-mandatory file types, including:

- Plugin (.py)
- LDC (.ldc)
- Testbench (.v)

- License Agreement (.txt)

A typical IP packaging procedure consists of the following steps:

1. The Radiant IP Packager encrypts the RTL files automatically if an IEEE P1735-2014 pragma is specified in RTL files.
2. The Radiant IP Packager provides design rule check (DRC) on the files. For example, the DRC will check whether or not the metadata.xml syntax is valid.
3. The Radiant IP Packager inserts a default license agreement file if you don't specify one.
4. The Radiant IP Packager tool allows developers to combine multiple files associated with the IP into a single deployable .IPK file. This file can then be used in the Radiant software environment to install the IP package in the user design environment.

Preparing IP Files for Packaging

IP Packager supports the following file types:

- Metadata file (.xml)
- RTL files (.v)
- LDC file (.ldc)
- Plugin file (.py)
- Document Files (.htm, .txt)
- License Agreement (.txt)
- Key (.txt)

The following describes each file type.

Metadata file (.xml) The metadata.xml file contains key information on the IP. For more information on the structure of the metadata.xml file, refer to Appendix A: "Metadata (.xml) File Structure" on page 21.

RTL files (.v) This file is the RTL implementation of the IP. The following is an example of the contents of a typical .v file, written in Verilog:

```
module module_01
#(
  parameter integer in_a_width = 8,
  parameter integer in_b_width = 8,
  parameter in_b_en = "True",
  parameter integer out_width = 16//,
) (
  clk,
  rst_n,
  in_a,
  in_b,
  result
);

localparam integer A_WDT = in_a_width;
localparam integer B_WDT = in_b_width;
localparam integer O_WDT = out_width;
localparam IB_EN = in_b_en;

input     clk;
input     rst_n;
input[A_WDT - 1:0]in_a;
input[B_WDT - 1:0]in_b;
input[O_WDT - 1:0]result;

generate
  if (IB_EN == "True")
  begin
    assign result - in_a;
  end
  else
  begin
    assign result = (in_a, in_b);
  end
endgenerate

endmodule // module_01
```

The IP component package has an optional parameterized testbench, which shares parameters used in IP implementation.

The IP Packager copies testbench files from the IP component package and creates two files based on the user configuration:

- dut_params.v

  Contains the parameters used in IP implementation.

  The name “dut_params.v” is fixed.

The following is an example of dut_params.v:

```
localparam ENABLE_A = 1;
localparam ENABLE_B = 0;
localparam BUS_WIDTH = 2;
```

- dut_inst.v

  Contains the IP instance instantiation. The connected wire name is the same as the port name.

  The name "dut_inst.v" is fixed.

  The following is an example of dut_inst.v.

```
INST_NAME u_INST_NAME
(
  .PORT_NAME   (PORT_NAME)
)
```

LDC file (.ldc) The template constraint file is written in Tcl. Settings of the IP can be used as variables in the template file, so that it can be further customized.

The IP generation engine will add additional information in the header of the template constraint file. The additional information includes:

- Architecture, device, and package names
- User configurations

The following is an example template constraint file. Availability of the create_clock constraint is controlled by variable $i2c_left_enable, and the period value of the constraint is controlled by variable $i2c_left_period.

```
if { $i2c_left_enable == 1 } {
  create_clock -name {clk0} -period $i2c_left_period [get_nets
SBCLKi_c]
}
```

The following is an example of a generated constraint file.

```
$architecture = "iCE40UP"
$device = "iCE40UP5K"
$package = "UWG30"
$i2c_left_enable = 1
$i2c_left_period = 1000

if { $i2c_left_enable == 1 } {
  create_clock -name {clk0} -period $i2c_left_period [get_nets
SBCLKi_c]
}
```

Plugin file (.py) Python expressions can be used in the metadata file to implement complex logic or calculations. However, Python expressions have limited capability if restricted to a single line.

To support complex logic, you can define any additional Python functions in the plugin.py file of the IP package, and then call the functions in Python expressions in the metadata file.

The following Python modules can be used in plugin.py:

- json

- xml
- textwrap
- collections
- os
- re
- traceback
- functools
- warnings
- shutil
- math
- itertools
- operator
- time
- weakref
- StringIO
- keyword
- copy
- codecs
- stat
- types
- string
- lxml

The following is an example of the contents of a plugin.py file:

```
def calc_out_width():
    ret_val = in_a_width
    if in_b_en:
        ret_val += in_b_width
    return ret_val
```

Document Files (.htm, .txt) The mandatory /doc directory contains:

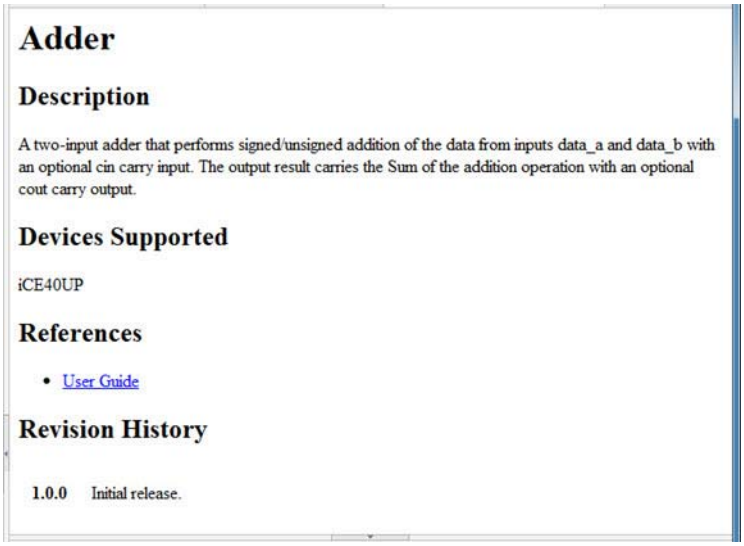
Mandatory introduction file. The file name should be “introduction.html”. The file should include the following information:

- Description
- Devices Supported
- Reference Documents
- Revision History

The following is an example of a typical “introduction.html” file. This example is for the Adder component:.

```
<HEAD>
  <TITLE>Adder</TITLE>
</HEAD>
<BODY>
  <H1>Adder</H1>
  <H2>Description</H2>
  <P>A two-input adder that performs signed/unsigned addition
of the data from inputs data_a and data_b with an optional cin
carry input. The output result carries the Sum of the addition
operation with an optional cout carry output.</P>
  <H2>Devices Supported</H2>
  <P>iCE40UP</P>
  <H2>References</H2>
  <UL>
    <P>
     <LI><A HREF="http://www.latticesemi.com/
view_document?document_id=52235" CLASS="URL">User Guide</A>
  </UL>
  <H2>Revision History</H2>
  <TABLE cellpadding="10">
    <TR>
<TD><B>1.0.0</B></TD> <TD>Initial release.</TD>
    </TR>
  </TABLE>
</BODY>
```

The following shows how a typical “introduction.html” page appears when displayed in the IP Information tab of the Radiant IP Catalog tool.



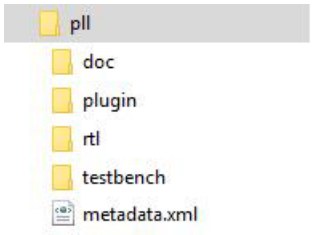
The /doc directory can also contain other optional documents.

License Agreement (.txt) Specify a license agreement in a text file. The Radiant IP Packager inserts a default license agreement file if you don’t specify one.

Directory Structure

This section describes the typical directory structure of a Radiant IP package.

The example below represents the folder tree for a “PLL” IP package. As shown, the top-level folder has the same name as the IP. The metadata.xml file does not reside in a subfolder.



There are four subfolders and one file inside the top folder, namely:

- The doc subfolder contains the user guide for the IP.
- The plugin subfolder contains a Python file embedding all necessary functions for parameter validation and checking.
- The rtl subfolder contains a parameterized RTL file of the IP. During IP generation, this file is instantiated into the wrapper with the parameters settings selected by the user.
- The testbench subfolder contains an optional parameterized testbench for the generated IP.

Running Radiant IP Packager

Radiant IP Packager is a stand-alone tool. Radiant IP Packager can be run from both Windows and Linux.

To run IP Packager:

- In Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Software > Accessories > IP Packager**.
- In Linux: from the ./<*Radiant_software_install_path*>/bin/lin64 directory, enter the following on a command line:

  ```
  ./ippackager
  ```

The IP Packager dialog box opens.

Generating an IPK File with IP Packager

IP Packager provides a user interface for you to select files and pack them into an IPK file. The IP Packager engine encrypts RTL files if IEEE P1735 V1 pragmas are specified in RTL source.

To generate an IPK file with IP Packager:

1. In the IP Packager dialog box, click **Add**.
2. In the Open dialog box, choose the file type that you wish to add from the dropdown menu. File types include:
   - Metadata file (.xml)
   - RTL files (.v)
   - Constraint files (.ldc)
   - Plugin file (.py)
   - Document files (.htm, .txt)
   - License Agreement (.txt)

   To remove any unwanted file, highlight the file in the Input files box and click **Remove**.
3. When all IP files are listed in the Input files box, in the **Output Directory** box, browse to the location where you want the IPK file to be generated. The default location is:

   C:\Users\<*username*>\RadiantIPLocal\<*IP_name*>
4. Click **Generate.** The IPK file is generated in the default location.

Installing IP Created with IP Packager into IP Catalog

You can download and add IP created with IP Packager into IP Catalog.

To download and add a User IP:

1. In the Radiant IP Catalog, click on the **Install a User IP** button.
2. In the **Select user IP package file to install** dialog box, browse to the Radiant Software IP Package (.ipk) file, and click **Open**.

- The IP will be installed into a folder in the your personal directory. For example: C:/Users/<*username*>/RadiantIPLocal/<*IP_name*>.
- The IP will be added into IP Catalog.

Metadata (.xml) File Structure

This appendix describes the structure and syntax of the metadata.xml file.

The metadata.xml file contains information on the IP including:

- Ports
- Parameters
- Attributes and functions for validating their intervals and relationships
- How the IP parameters will appear in the GUI. For example, some parameters might be visible or hidden, editable or fixed depending on other parameters values selected by the user.

The metadata.xml name for this file is mandatory for all IPs.

Metadata is provided in an XML file. Metadata serves the following functions:

- Contains the definitions of IPs ports.
- Contains the definitions of parameters, alongside their valid intervals and relationships.
- Controls creating an appropriate dialog box, such as specifying the way in which each parameter is presented graphically.
- Controls other aspects of how the IP is generated and synthesized.

When you make the necessary parameter selections and generate the IP, the generated RTL will contain the instance of IP specific to the specified parameters (such as appropriately set ports). There is a wrapper generated by the Python script which instantiates the IP's RTL while applying the selected parameters. The same Python script checks whether the selected combination of parameters is legal, and performs all necessary calculations. Only one Python script is present in the directory structure of each IP, and it contains all necessary functions.

The root node of the XML file is <ip>, which consists of:

- Attribute: "version"
- Three mandatory child nodes: <general>, <settings> and <ports>
- Two optional child nodes: <componentGenerators> and <estimatedResources>

The following is an example of Metadata layout:

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

The attribute “version” records the version of the XML metadata format. Current value is 1.0.

The following table lists child nodes of the root node.

Child Node	Description
general	General information of the soft IP
settings	Configurable settings of the soft IP
ports	Ports of the soft IP

The <general> node describes general information about a soft IP, for example, name, version, category, etc. The following table describes child nodes of the <general> node.

Child Node	Mandatory	Description
vendor	No	Soft IP vendor. Official soft IPs should have the vendor’s name. For example, “latticesemi.com”.
name	Yes	Name of the soft IP. The name is used to identify the soft IP, so it should be unique. For example, "adder"
display_name	No	Name to be displayed in the software. If “display_name” is not set, the software displays “name” directly. For example, “Adder”

library	No	Library of the soft IP. If “library” is not set, the default value “ip” will be used. For example, “ip”, “interface”.
version	Yes	Version of the soft IP. For example, 1.0.0.
category	Yes	Category of the soft IP. Category could be hierarchical. Levels are separated by “,”. For example, Memory_Modules,Distributed_RAM
min_radiant_version	Yes	The minimal Radiant version, which supports the soft IP. For example, 1.0.
supported_products	No	FPGA products supported by the soft IP.

The <settings> node should contain one or more <setting> nodes. In an IP instance package, Verilog parameters are used to customize the soft IP. All user configurable parameters should be added to the <settings> section as <setting> nodes.

Beside parameters, you can also add <settings> nodes for user input only. Both parameters and inputs could be used in Python expressions to do dynamic evaluation.

The following table lists attributes of the <settings> nodes.

Attribute	Value	Mandatory	Description
id	valid Python identity	Yes	The unique ID of the setting, which can also be referred as: ▶ variable name in Python expressions ▶ variable name in parameterized template constraint file For example, id=”num_outputs”
title	String	No	Short title of the setting. If title is not specified, id will be used. For example, title=”Number of Output”
type	param, input	Yes	A setting could be a Verilog parameter or user input. Both `param` and `input` settings can be used to compute values of other `param` and `input` settings. `param` and `input` only differ in generated file. `param` is written out but `input` is ignored. For example, type=”param”
value_type	bool, string, int, float, path	Yes	Type of the value. Supported types are `bool`, `int`, `float`, `string` and `path`. The `path` type indicates a string which represents a path. For example, value_type=”int”

Attribute	Value	Mandatory	Description
conn_mod	String	Yes	Name of the IP that this setting applies to. For example, conn_mod=”pll”
default	Python expression	No	Default value of the setting. Without default attribute, the first item from the options attribute is used. If there are no options, the initial value of setting will be set to 0 for `int`, 0.0 for `float`, “” for `string` and False for `bool`. For example, default=”1.0”
value_expr	Python expression	No	Python expression to compute the value of the setting. This attribute takes effects only when the setting is disabled. For example, value_expr="calc_extdiv_val_add(extdiv_en, extdiv_port_sel)" (calc_extdiv_val_add is defined in plugin.py extdiv_en and extdiv_port_sel are the setting IDs in metadata.xml)
options	Python list or list of tuples	No	Candidate options for the setting, which is used by GUI to display a dropdown selector. It could be set to a simple list or a list of tuples. If it’s a simple list, elements are displayed and written. If it’s a list of tuples, the 1st item in tuple is displayed and the 2nd item in tuple is written. For example, options=”[0.1, 0.2, 0.5, 1.0]”
output_formatter	str	No	Control how parameter values are written in output RTL files. Currently, only `str` is supported. `str`: parameter values are written as strings For 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. For 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 will be grayed out in GUI display and its value will be computed from `value_expr`. Otherwise user input will be used. For example, editable="(FEEDBACK_PATH == 'PHASE_AND_DELAY') (FEEDBACK_PATH is a setting ID in metadata.xml)
hidden	True	No	Python expression to determine whether the setting is hidden in GUI. If hidden is set to True (default is False), the item is hidden in GUI. Current GUI only support hidden=“True”, which cannot be changed dynamically. For example, hidden=”True”

Attribute	Value	Mandatory	Description
drc	Python expression	No	Python expression to do DRC on the setting. True means DRC pass. For 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 and i2c_left_addressing_width are IDs of other settings)
regex	Regular expression	No	Regular expression to do DRC on the setting.
value_range	Python tuple or list with 2 comparable elements	No	Valid range of setting value, which is used to do DRC on the setting. The maximum value can be infinity “float(‘inf’)”. For example, value_range="(0, 1023) if(i2c_right_enable) else (-9999, 9999)"
description	String	No	Detailed description of the setting.
group1	String	No	Group the settings, and be used to display on GUI For example, group1= "Output Setting"
group2	String	No	Group the settings, and be used to display on GUI

The <ports> node. IP module package has some ports in its implementation. These ports should be described in the <ports> section as <port> child nodes. The following table lists attributes of <port> nodes.

Attribute	Value	Mandatory	Description
name	Valid Verilog port name	Yes	Name of port. For example, name=”Clk”
dir	`in`, `out`, `inout`	Yes	Direction of port. For example, dir=”in”
range	Python tuple or list with 2 int elements	No	Range of this port. It should be a Python expression whose evaluation result is a tuple or array with 2 elements. For example, range="(A_WDT-1, 0)" (A_WDT is a setting ID)
conn_mod	Valid Verilog module name	Yes	Name of IP core module to which this port connects. For example, conn_mod=”counter”
conn_port	Valid Verilog port name	No	Name of port of IP core module to which this port connects. name will be used if conn_port is not specified. For example, conn_port=”Clk”

stick_high	Python expression	No	Python script. True: tie this port to 1. For example, stick_high="True"
stick_low	Python expression	No	Python script. True: tie this port to 0. For 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. For 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. For example, bus_interface=" ahb_slave_0"

The <componentGenerators> node contains a list of componentGenerator elements. Each componentGenerator element defines a generator that is run on generated IP instance package. The description of componentGenerators follows IP-XACT format.

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

The <estimatedResources> node contains a list of estimatedResource elements. Each estimatedResource element defines the formula to calculate one type of resource used in the IP instance package.

```
<lsccip:estimatedResources>
  <lsccip:estimatedResource>
    <lsccip:name>LUT4</lsccip:name>
    <lsccip:number>WIDTH * 4</lsccip:number>
  </lsccip:Resource>
</lsccip:Resources>
```

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.

Revision History

The following table gives the revision history for this document.

Date	Version	Description
04/23/2019	1.0	Initial Release.