



Creating Custom IP with IP Packager

Application Note

FPGA-AN-02102-1.0

April 2025

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 all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. 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. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. 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.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language [FAQ 6878](#) for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents	3
Abbreviations in This Document.....	5
1. Introduction	6
1.1. Lattice IP Packager	6
1.2. Lattice IP File Structure	7
2. Custom IP Pre-Requisites.....	9
2.1. Mandatory Files	9
2.1.1. RTL Design Files	9
2.1.2. Introduction.html File	9
2.2. Optional Files	9
2.2.1. Constraint Files	9
2.2.2. Plugin Script.....	10
2.2.3. Testbench Files	10
2.2.4. C/C++ Drivers.....	10
2.2.5. User License Agreement.....	10
2.3. Preparing RTL for IP Packaging.....	11
3. Creating a Custom IP	13
3.1. Defining IP Metadata	13
3.2. Adding Required Files.....	14
3.3. Defining Top-Level Ports, Interfaces, and Memory Map	15
3.4. Creating Custom Parameters	20
3.5. Further Customization of the IP.....	26
3.6. Packaging and Installing the IP.....	28
References.....	30
Technical Support Assistance	31
Revision History.....	32

Figures

Figure 1.1. Location of the IP Packager Application within the Windows Start Menu	6
Figure 1.2. Location of the IP Packager within the Propel Builder Software	7
Figure 1.3. Lattice IP File Structure Before and After IP Generation	7
Figure 2.1. introduction.html Page in Lattice IP Packages.....	9
Figure 2.2. Example of Python Plugin Script Function in an IP Package	10
Figure 2.3. Default License Agreement for Custom IP if None are Included	11
Figure 2.4. IP Generation GUI	11
Figure 2.5. IP-Generated RTL	12
Figure 3.1. Open IP Directory Icon.....	13
Figure 3.1. Basic Metadata Settings for the IP.....	14
Figure 3.1. Adding the Required Files	14
Figure 3.1. Doc Assistant Page.....	15
Figure 3.5. Infer Ports from HDL	16
Figure 3.6. Keep the Default Port Settings	16
Figure 3.7. IP Preview Page.....	17
Figure 3.8. Add Memory Map.....	17
Figure 3.9. Add Address Block	18
Figure 3.10. Address Block Settings.....	18
Figure 3.11. Interface Settings.....	19
Figure 3.12. A Preview of the Current IP	19
Figure 3.13. Add Parameter Tab.....	21

Figure 3.14. Parameter Tabs and Groups in a Completed IP Package.....	21
Figure 3.15. Add Parameter Group.....	22
Figure 3.16. APB Address Width Parameter Settings	22
Figure 3.17. Device Family Parameter Settings	23
Figure 3.18. Number of LEDs Parameter Settings	23
Figure 3.19. User Interface Parameter Settings	24
Figure 3.20. Enable Interrupt Parameter Settings	24
Figure 3.21. IP Package Structure	25
Figure 3.22. IP Configuration on the Preview Page	25
Figure 3.23. Stick Low Setting for Ports Beginning with APB.....	26
Figure 3.24. Stick Low Setting for Ports Beginning with LMMI.....	27
Figure 3.25. Stick Low Setting for int_o Signal.....	27
Figure 3.26. IP Diagram Updates According to the IP Configurations	28
Figure 3.27. Package IP Icon	28
Figure 3.28. Location of the Package Files.....	29
Figure 3.29. Install Custom IP Icon	29
Figure 3.30. Location of the Custom IP Package.....	29

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
FDC	Synplify Design Constraints
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
IP	Intellectual Property
LDC	Lattice Design Constraints
LSE	Lattice Synthesis Engine
OS	Operating System
RTL	Register Transfer Level
SDC	Synopsys Design Constraints
TCL	Tool Command Language
XML	Extensible Markup Language

1. Introduction

Intellectual property (IP) cores are a key part of many designer's development flows, allowing them to use existing IP without having to develop new register transfer level (RTL) design for existing functions each time they work on a new project. Lattice provides several IP for commonly used design blocks, such as memory interfaces, general purpose input/output (GPIO), arithmetic modules, and much more. All these IP are available in Lattice's IP Catalog in both the Radiant™ software and Propel™ software, however it is also possible to create custom IP that can be shared with others and reused in different projects and design environments.

There are many advantages in creating a custom IP. For example, if there is a parameterizable RTL module which is used in several assorted designs, creating a custom IP may be a good idea to enable easier reuse later. The most important thing to consider when deciding whether to create a custom IP is the time vs return tradeoff. Creating a custom IP requires additional work and effort so it may not be best to create a custom IP in all situations if there will not be much reuse.

1.1. Lattice IP Packager

The Lattice IP Packager is the tool used to create custom IP cores. The IP Packager is an integrated tool that is automatically installed whenever the Radiant software or Propel software is installed. There is no standalone IP Packager so you must use one of the aforementioned tools to install the IP Packager and create custom IP packages. The location and how the IP Packager is launched varies slightly between the Radiant software and Propel software. Within the Lattice Radiant software, you can launch the IP Packager in the Windows environments by opening the Windows navigation menu and locating the folder called *Lattice Radiant Software*. From the dropdown list of applications within that folder, click **IP Packager** to launch the tool.

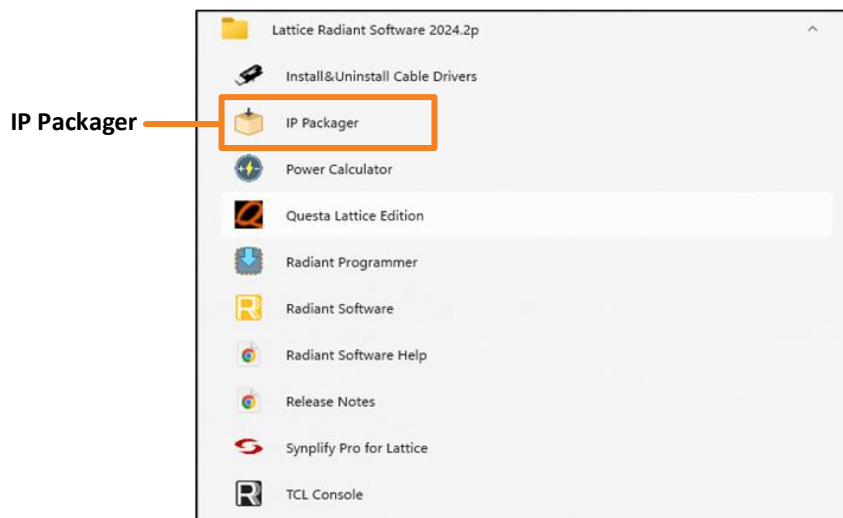


Figure 1.1. Location of the IP Packager Application within the Windows Start Menu

Aside from that, you can find and directly launch the IP Packager executable file (*ippack.exe*) by double-clicking within the */radiant/<version number>/bin/nt64/* directory in Windows, or */radiant/<version number>/bin/lin64/* for Linux operating system (OS). Additionally, you can also use the *ippackc.exe* file to launch the TCL console-only mode of IP Packager.

On the Propel Software side of things, you can launch the IP Packager directly from the Lattice Propel Builder software by selecting **Tools** → **IP Packager** from the toolbar.

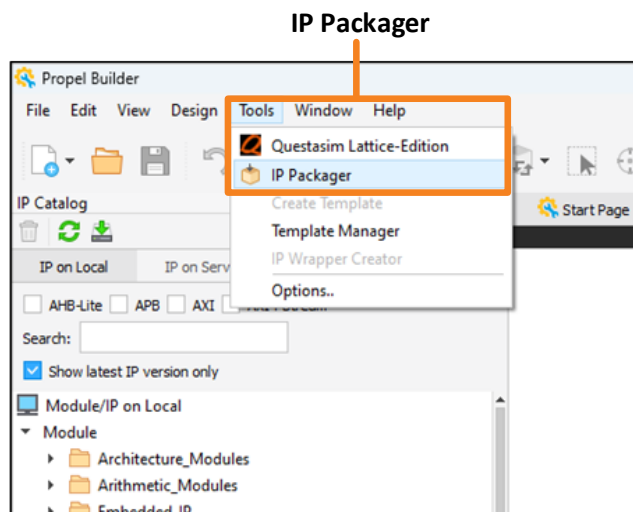


Figure 1.2. Location of the IP Packager within the Propel Builder Software

Similar to the Radiant software, you can find the IP Packager executable file (*ippack.exe*) within */propel/<version number>/builder/rtf/bin/nt64* for Windows, or */propel/<version number>/builder/rtf/bin/linux64* for Linux OS. Aside from that, you can also use the *ippackc.exe* file to launch the TCL console-only mode of IP Packager.

1.2. Lattice IP File Structure

One important distinction to remember when developing an IP package is its structure and files change slightly after IP generation. This section describes the differences between IP packages before and after IP generation.

The following figure outlines the structure of Lattice IP packages before and after IP generation. The text highlighted in green are folders and files that are unchanged, while text in orange are files that are new or modified during IP generation.

Before IP Generation	After IP Generation
constraints	constraints
doc	doc
EULA.txt	EULA.txt
introduction.html	introduction.html
ldc	ldc
plugin	plugin
plugin.py	plugin.py
rtl	rtl
top.v	<instance name>.v
submodule.v	<instance name>_bb.v
testbench	testbench
bus_interface.xml	component.xml
memory_map.xml	design.xml
metadata.xml	<instance name>.cfg
	<instance name>.ipx

Figure 1.3. Lattice IP File Structure Before and After IP Generation

As shown in the figure above, the main files that are changed during IP generation are RTL and XML files bundled with an IP package. One key behavior to keep in mind with IP RTL files is that after IP generation all the RTL modules are concatenated into a single file matching the name of the IP instance that was generated called *<instance name>.v*. This file contains the complete design and matches the language of the original source files used as an input to the IP package. If an IP package is mixed language, then additional RTL files are generated corresponding to the language of

the sub-modules within the IP package. Aside from that, there will also be a new file called *<instance name>_bb.v* that is a closed-box instantiation template for the IP that was generated.

Aside from that, another key difference has to do with the XML files in an IP package. Pre-IP generation of an IP package contains the following three main XML files, which define the IP package interfaces, address spaces, and settings:

- *bus_interface.xml*
- *memory_map.xml*
- *metadata.xml*

After IP generation, use the three original XML files to generate a *component.xml* and *design.xml*, which has the same information but narrowed down depending on the settings you select during IP generation. At a high-level, the three original files contain the settings for all possible definitions of an IP package, while the resultant two XML files after generation correlate to the specific settings you select during IP generation. For a typical development flow, you do not need to worry about any XML files, as the IP Packager automatically generate and populate these files.

Lastly, post-IP generation, there is also a *<instance name>.cfg* file that contains all the parameters that you select during IP generation as well as a *<instance name>.ipx* that is the IP instance file which is used as the IP source in the Radiant software and Propel software.

2. Custom IP Pre-Requisites

Custom IP packages can contain several types of files, some of which are optional while others are mandatory. The following section describes those files, and which file must be prepared ahead of time before creating a custom IP package.

2.1. Mandatory Files

2.1.1. RTL Design Files

These are the source files for a custom IP package. The IP Packager supports Verilog, System Verilog, and VHDL files, including mixed-language combinations. Before creating a custom IP, you must ensure that the RTL in your IP package can compile and is free of syntax errors. For more information on how to prepare RTL for custom IP packaging, refer to the [Preparing RTL for IP Packaging](#) section.

Prepare ahead of time: Yes.

2.1.2. Introduction.html File

The *introduction.html* file of an IP package is the basic Help information page that is displayed when an IP package Help info is selected. This file can be prepared ahead of time or generated within the IP Packager using the Doc Assistant feature. The following figure shows how this file appears after you install a custom IP package.

Prepare ahead of time: Optional.

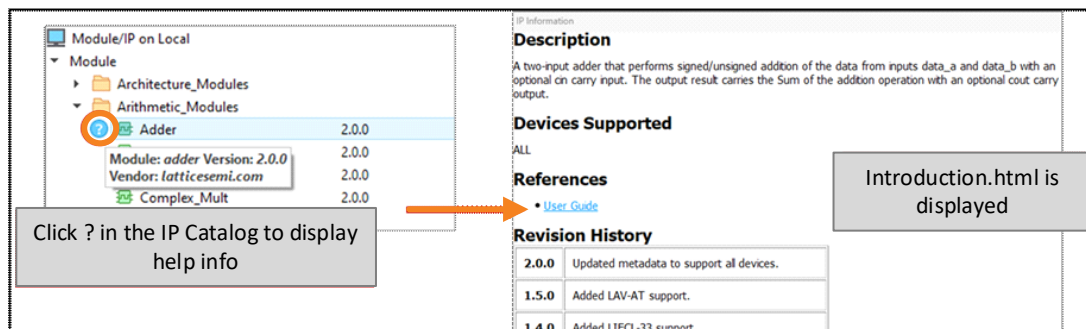


Figure 2.1. introduction.html Page in Lattice IP Packages

2.2. Optional Files

All the files in this section are optional and do not need to be created ahead of time.

2.2.1. Constraint Files

The IP Packager supports the creation of Synopsys® Design Constraints (SDC), Lattice Design Constraints (LDC), and Synplify® Design Constraints (FDC) files for use in custom IP packages. The most commonly used constraint type is SDC, which can be applied at either the pre-synthesis or pre-map stage depending on how those settings are configured in the IP Packager. Additionally, you can customize the behavior of these constraint files based on the synthesis tool you are using.

Aside from that, the IP Packager also supports entry of LDC constraint files for Lattice Synthesis Engine (LSE), as well as FDC constraint files for use with the Synplify Pro software.

2.2.2. Plugin Script

You can include a custom Python plugin script with IP packages to enable more complex IP generation. The Python script must be free of any syntax errors before usage in a custom IP package. The custom plugin script can consist of several functions that can be referenced from within a custom IP package to implement more complex logical conditions or design rule checks (DRCs) as shown in the figure below.

Property	Value
title	End Address Port S0
type	param
value_type	int
default	16383
value_expr	convHex(s0_end)
options	
output_formatter	
bool_value_mapping	
editable	False
hidden	True
drc	
regex	
value_range	
config_groups	
description	

```
def convHex(val):
    v_len = len(val)
    ret_val = 0
    for i1 in range(0, v_len):
        v_buff = (val[v_len - (1 + i1)]).capitalize()
        if(checkIfHex(v_buff)):
            ret_val = ret_val + convToDec(v_buff)*hex_exp(i1)
    return ret_val
```

Figure 2.2. Example of Python Plugin Script Function in an IP Package

2.2.3. Testbench Files

The IP Packager also supports entry of simulation and testbench files for custom IP packages. These files are not syntax checked during IP generation and do not need to be 100% free of syntax errors for custom IP packaging. Any testbench files are marked for simulation only when an IP package is generated and will not be used for synthesis.

2.2.4. C/C++ Drivers

It is also possible to add optional C or C++ driver files with an IP package. These files are not syntax checked and do not have a direct impact on how an IP package is generated. After packaging an IP and installing it in the Radiant software or Propel software, these driver files are added to a *driver* directory within the IP package. If a custom IP with drivers is used in a Propel software design, the driver files for that IP will be added automatically to the Lattice Propel SDK C/C++ project when it is created.

2.2.5. User License Agreement

You can add a user license agreement file to an IP package. This license agreement is the final text popup you see and must agree to before installing a custom IP. If a custom license agreement is not included, the IP Packager uses a generic license agreement as shown in the following figure.

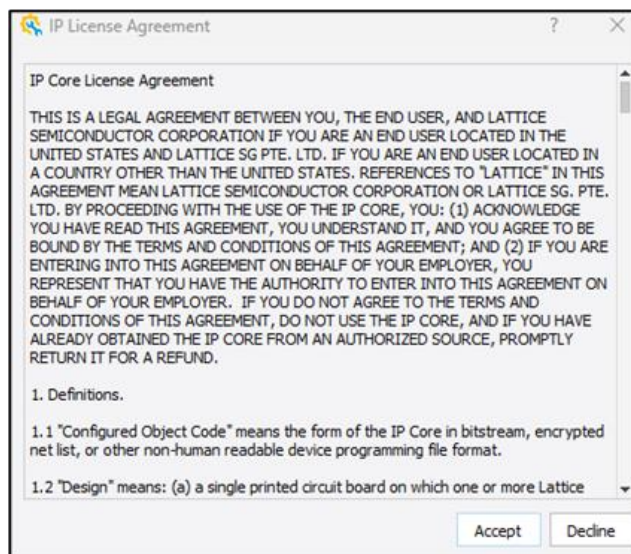


Figure 2.3. Default License Agreement for Custom IP if None are Included

2.3. Preparing RTL for IP Packaging

The most important thing to consider before creating a custom IP package is the structure of the RTL files within your design. First, an IP package RTL must be free of syntax errors and able to compile before IP packaging. Aside from that, another thing to keep in mind is how IP packages are generated when they are added to a design.

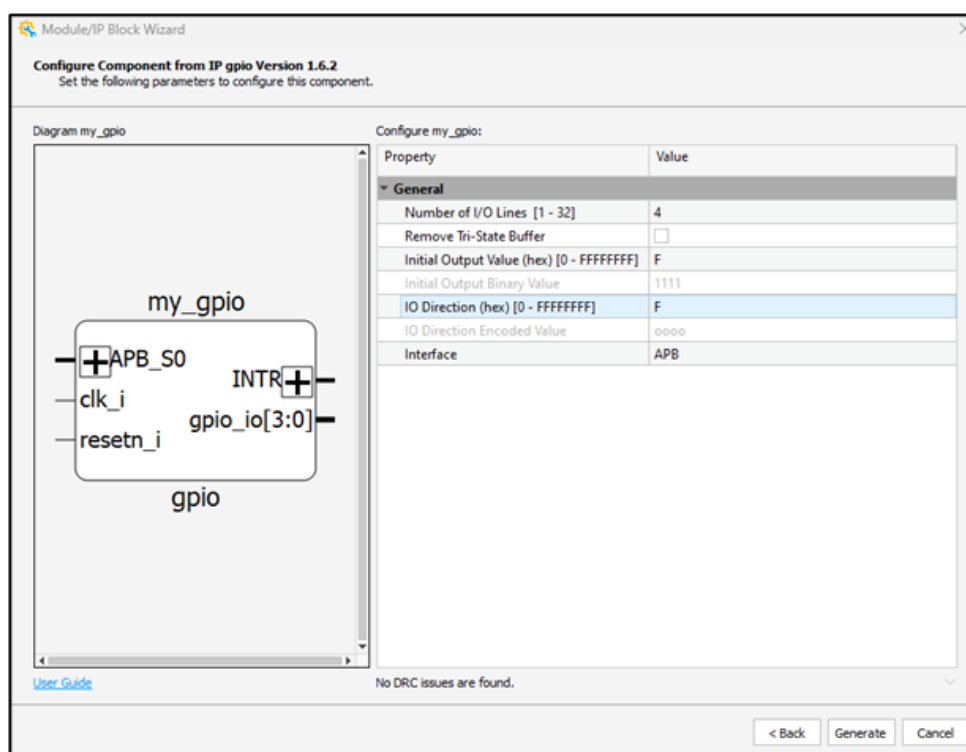


Figure 2.4. IP Generation GUI

When generating an IP in the Radiant software or Propel software, you must configure the IP settings in the Module/IP block wizard. Whatever settings are selected at this point are written out at the top level and passed directly to the

original RTL top module. Because of this, it is important that IP packages are set up in a way that all parameters that are intended to be user configurable are at the top-level module for parameter passing to work as intended.

As can be seen from the example in Figure 2.5, which is a continuation of the example in Figure 2.4, all parameters which were selected in the IP generation GUI were written in the top-level of the IP generated RTL and passed down through module instantiation to the original RTL. Any parameters that are not modified during IP generation are not passed down to the original RTL module, meaning the default parameter values will be used instead.

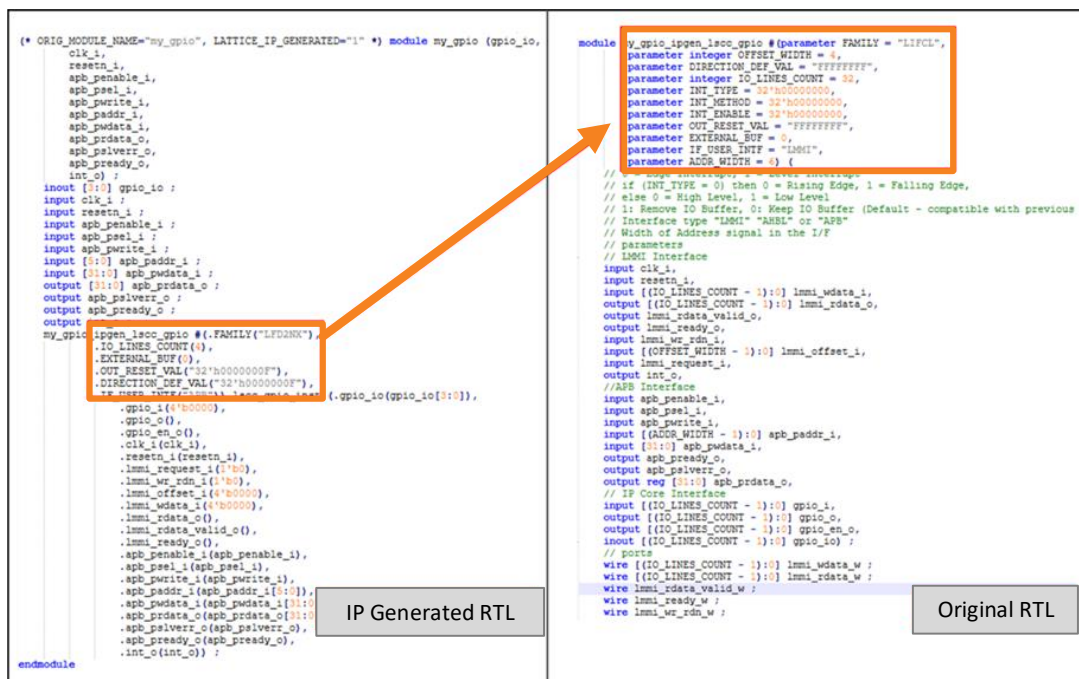


Figure 2.5. IP-Generated RTL

Because of this, it is important to always consider what parameters you want to be configurable when developing RTL for a custom IP to ensure IP cores are generated as expected. Parameters used in conjunction with generate statements enable very powerful and highly customizable IP generation.

3. Creating a Custom IP

This section describes the main process of creating a custom IP package. Aside from that, there are additional IP Packager settings that are not covered in this section. For more information on those settings and other IP Packager related questions, refer to the [Lattice IP Packager User Guide](#).

3.1. Defining IP Metadata

The first thing to do when creating a custom IP package is to define the basic metadata for the IP. This includes settings such as the IP name, library, version, compatible Radiant or Propel software versions, and device support.

To set the basic metadata for the IP, follow these steps:

1. Open the IP Packager.
2. Select the **Open IP Directory** icon, or select **Files** → **Open IP Directory**.

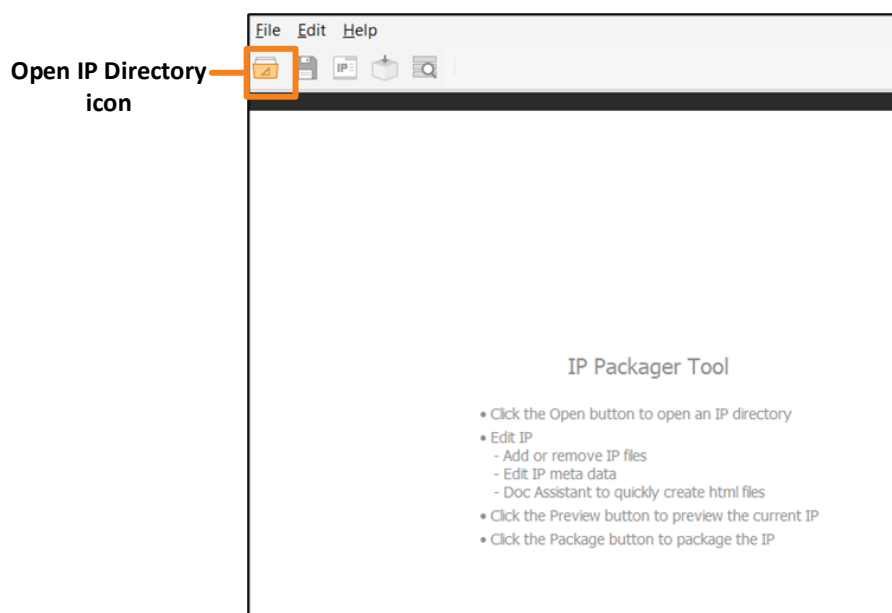


Figure 3.1. Open IP Directory Icon

3. Select the *driver_ip* directory.

This directory is completely empty. The IP Packager automatically generates required files for an IP package as settings are selected during IP development.

4. In the Basic Info portion of the **Meta Data** page, populate the following settings:

- Name: led_driver
- IP version: 1.0.1
- Supported device: LAV-AT, LFCPNX, LFD2NX, LFMXO5, LIFCL
- Category: custom
- Display name: APB LED Driver
- Supported platform: Lattice Radiant software and Lattice Propel software.
These are the Lattice tools that support the IP.

Figure 3.2. Basic Metadata Settings for the IP

3.2. Adding Required Files

After defining the basic metadata settings for a custom IP, the next steps are to add all the required and optional files.

To add the required files, follow these steps:

1. Navigate to the **Design Files** page.
2. Click the **Add** button to include the following files:
 - `top_driver.v`
 - `apb2lmmi.v`
 - `lmmi_driver.v`

Figure 3.3. Adding the Required Files

3. Use the up arrow key in the bottom left to move the `top_driver.v` file to the top of the file list.
The IP Packager automatically parses all source files to determine the top module.
4. Switch to the **Doc Assistant** page to create a custom *introduction.html* file.
5. Populate the **Title**, **Description**, **Device Support**, and **Revision** fields with any information.
Because this HTML file does not have any other impact on the IP package, the exact contents are not important.
You can also leave all the fields blank.

Revision history:	
Revision	Content
1.0.1	Added support for APB.
1.0.0	Initial release.

Figure 3.4. Doc Assistant Page

6. Click **Save and Add to IP Package**.
The *introduction.html* file is generated and automatically added to the IP package in the Misc page under *Document*.

3.3. Defining Top-Level Ports, Interfaces, and Memory Map

After the required files for an IP have been added, the next step is to define the top-level ports and interfaces for an IP. To define the top-level ports and interfaces, follow these steps:

1. Navigate back to the **Meta Data** page.
2. Right-click on **Ports** and select **Infer Ports from HDL**. This feature forces the IP Packager to extract the ports from the top-level RTL module in an IP package, and can only be used if RTL files are already added.

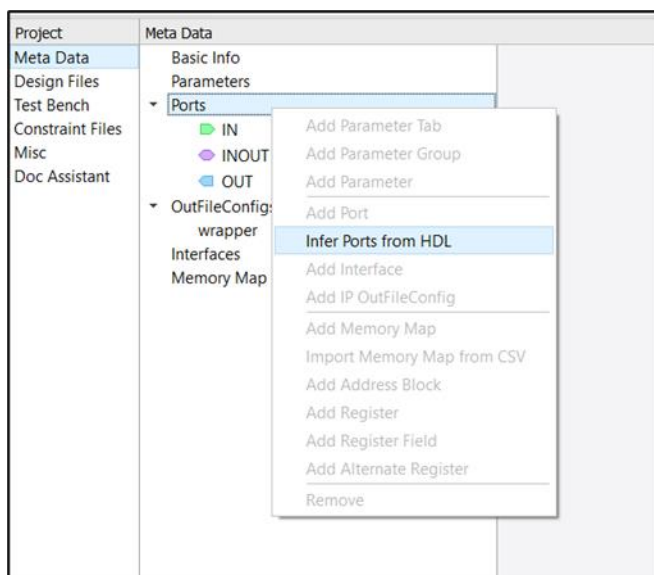


Figure 3.5. Infer Ports from HDL

3. Leave all port settings as default and click **OK**.

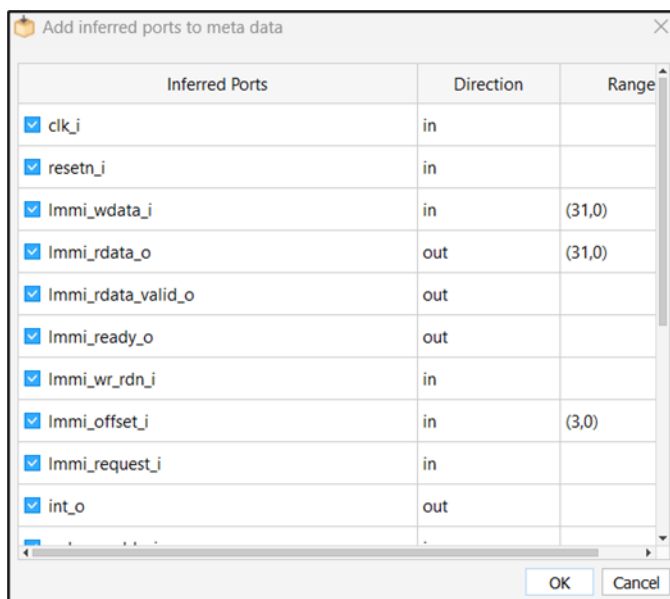


Figure 3.6. Keep the Default Port Settings

4. Click the **Preview IP** button, or select **Edit → IP Preview** from the dropdown toolbar.
 - A block-level preview of the current IP must appear with all the top-level ports added. At this point in the tutorial, all ports are considered regular signals, and additional setup must be done to identify which are part of an interface.
 - You can use IP Preview at any point during IP development to get an idea how your IP looks as you modify various settings.

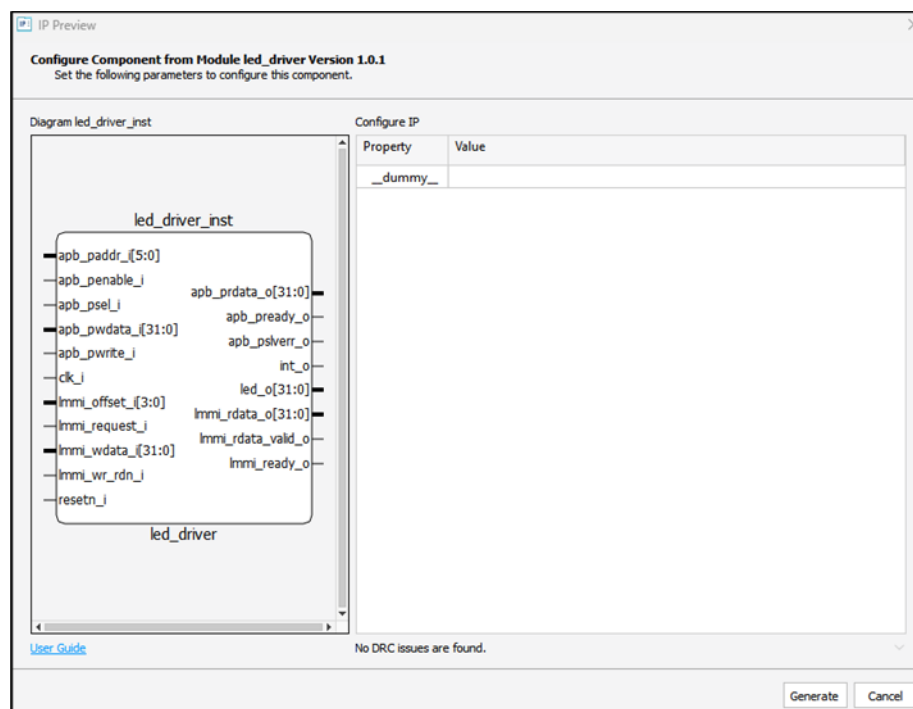


Figure 3.7. IP Preview Page

5. In the **Meta Data** page, right-click Memory Map, and select **Add Memory Map**.

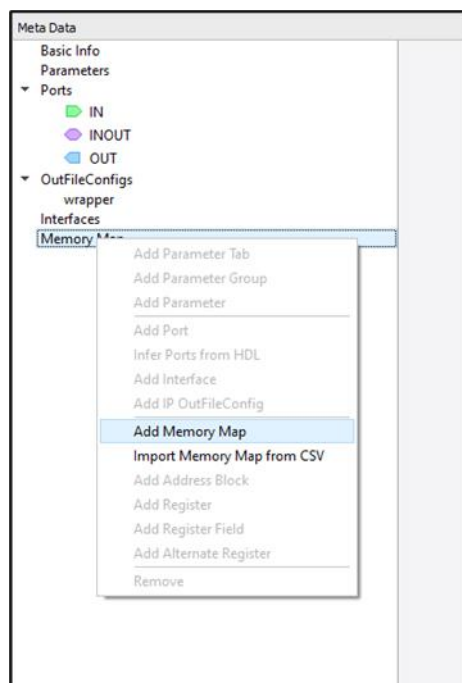


Figure 3.8. Add Memory Map

6. Double-click the new memory map, and rename it to LED_map.
7. Right-click the memory map LED_map and select **Add Address Block**.

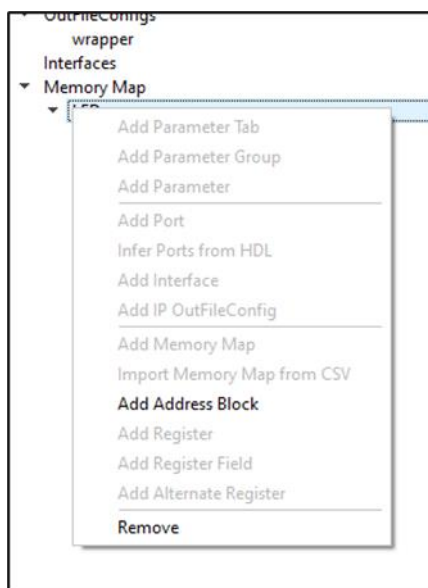


Figure 3.9. Add Address Block

8. Change the settings for NewAddressBlock1 as follows:

- Base address: 0
- Width: 32
- Range: 32

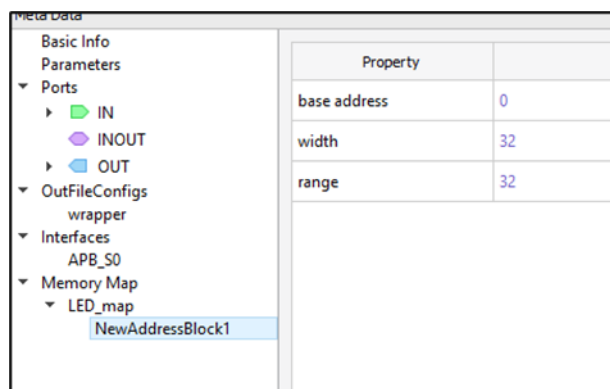


Figure 3.10. Address Block Settings

9. On the **Meta Data** page, right-click **Interfaces** and select **Add Interface**.

10. Double-click to rename the new interface APB_S0 and change its settings as follows:

- Display name: APB_S0
- Interface type: AMBA APB
- Role: Target
- Mem map reference: LED_map
- Port mappings:
 - PADDR: apb_paddr_i
 - PENABLE: apb_penable_i
 - PRDATA: apb_prdata_o
 - PREADY: apb_pready_o
 - PSELx: apb_psel_i
 - PSLVERR: apb_pslverr_o

- PWDATA: apb_pwdata_i
- PWRITE: apb_pwrite_i

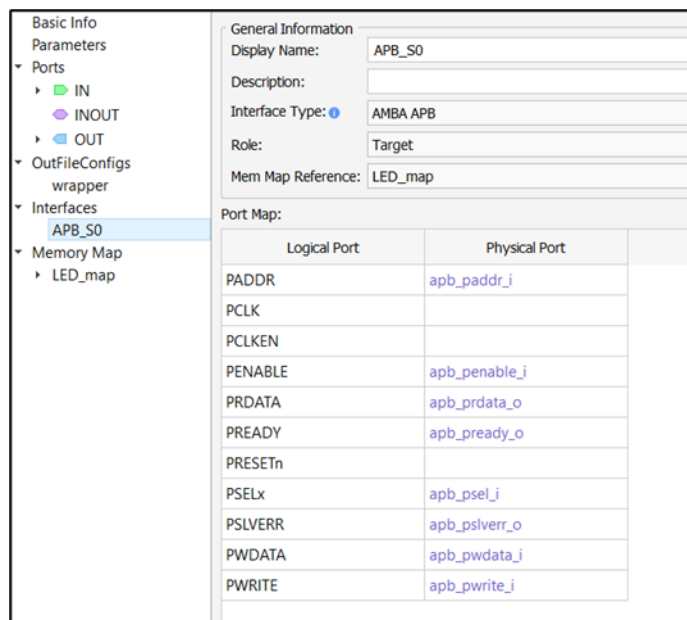


Figure 3.11. Interface Settings

- Right-click **Interfaces** to add another new interface, rename it to IRQ0, then change its settings as follows:
 - Display name: IRQ0
 - Interface type: Interrupt
 - Role: Controller
 - Port mappings:
 - IRQ: int_o
- Save the IP package, then click the **Preview IP** icon to view the current IP progress.

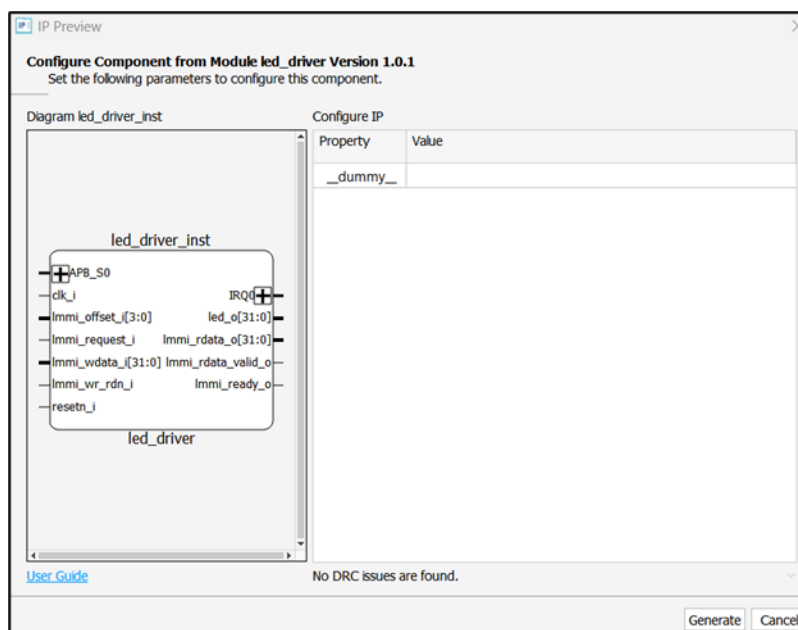


Figure 3.12. A Preview of the Current IP

3.4. Creating Custom Parameters

Parameters enable customization and configurability for custom IP packages. Before adding parameters and settings to an IP package, you must set up your RTL in a way that supports parameter passing at the top-level RTL module as described in the [Preparing RTL for IP Packaging](#) section.

Some important concepts to understand before creating custom parameters for an IP package are the types of parameters you can add, and what settings are required.

Types of Parameters:

- **Input:** These settings do not necessarily have a direct correlation with user RTL. These settings are often used to calculate values for parameters that are written to user RTL, or for DRC purposes. For example, a custom IP can contain a setting that takes a decimal input for a clock frequency and then converted to binary parameter that is written to the actual user RTL.
- **Param:** These settings directly relate to parameters in user RTL. When an IP is generated, these parameters are passed down to the original RTL through module instantiation. Parameters are case sensitive and must exactly match the parameter names in RTL.
- **Command:** A click-button that can be configured to run some function when it is pressed. For example, a command can be set up so a file is generated whenever a button in the IP generation GUI is clicked.
- **Verilog Macro:** This setting enables you to write Verilog-defined macros directly to your RTL.

Aside from the types of parameter settings, there are also a few other key settings you must understand before creating custom parameters for an IP package.

Additional Settings:

- **Value Type:** The type of value that is returned by a parameter or setting after IP generation. Supported types are bool, integer, float, and string. When configuring a custom parameter, it is important to ensure the value type is correct to avoid potential type mismatching issues.
- **Value_expr:** A single line of Python code. It can be any Python expression, or even a function call to a module within a Python plugin script. This setting is mutually exclusive with *options*.
- **Options:** A Python list of values that are supported for the current parameter or setting. Whatever is input into this list will appear as an option in a dropdown list from which you can select. This setting is mutually exclusive with *value_expr* and must contain a default value.
- **Editable:** A single line of Python code that returns Boolean True or False (can also set directly as True or False). If this setting is evaluated to be true, then the setting is available for you to modify. If this setting is evaluated to be false, the setting is grayed out and unavailable for you to directly edit. However, this setting can still be modified by other parameters or settings.
- **Hidden:** A single line of Python code that returns Boolean True or False (can also set directly as True or False). If this setting is evaluated to be true, the parameter is hidden from you. This setting is purely visual and does not impact IP generation in any way.
- **DRC:** A single line of Python code that returns Boolean True or False. If a DRC is evaluated to be true, a message, warning, or error is displayed depending on how the DRC was configured.

To create custom parameters for an IP package, follow these steps:

1. Within the **Meta Data** page, right click **Parameters** and select **Add Parameter tab**.

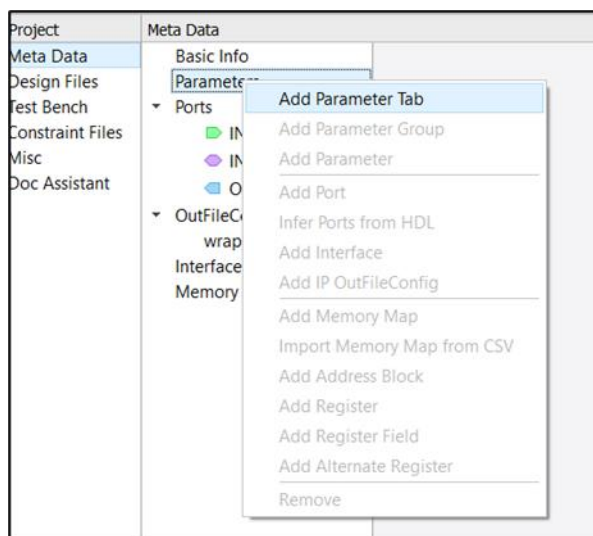


Figure 3.13. Add Parameter Tab

2. Double click the parameter tab **Tab1** to edit it, and change it to **General**, then press Enter.
 - Parameter tabs and groups can be renamed to anything.
 - Parameter tabs and groups are purely for organizational purposes:
 - Parameter tabs are separate pages within an IP that contains different parameters or settings.
 - Parameter groups allows control over how settings and parameters are grouped together within a page.

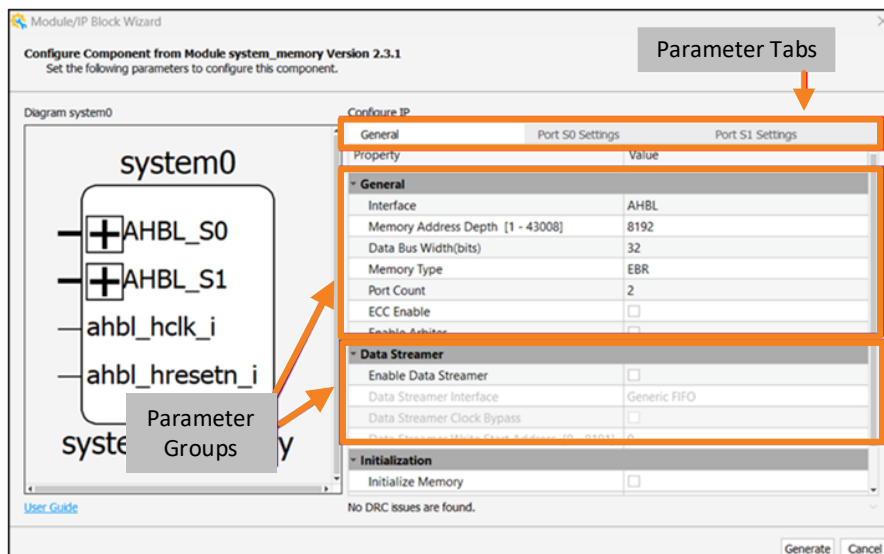


Figure 3.14. Parameter Tabs and Groups in a Completed IP Package

3. Right-click the parameter group **General** created in step 1, and select **Add Parameter Group**.

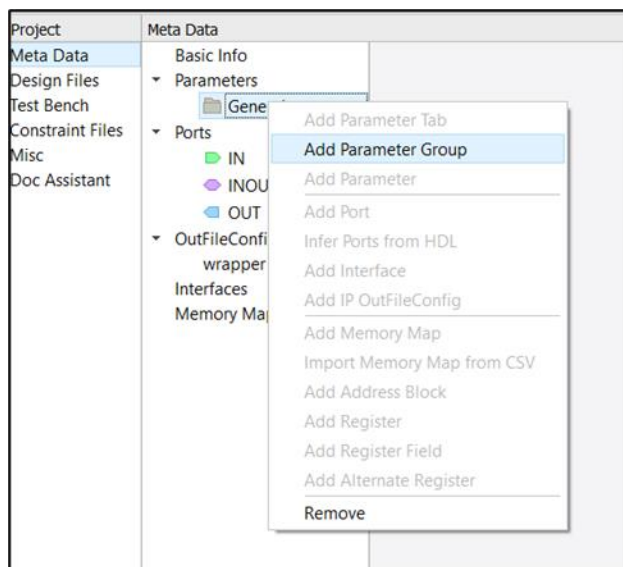


Figure 3.15. Add Parameter Group

4. Double-click the new parameter group, and rename it to *Parameters*.
5. Right-click the parameter group *Parameters*, and select **Add Parameter**.
6. Add a new parameter and rename it to *ADDR_WIDTH*.
 - Populate the following settings:
 - Title: APB Address Width
 - Type: param
 - Value_type: int
 - Default: 6
 - Options: $[(i+1)*2 \text{ for } i \text{ in range } (0,16)]$

Property	
title	APB Address Width
type	param
value_type	int
default	6
value_expr	
options	$[(i+1)*2 \text{ for } i \text{ in range}(0,16)]$
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.16. APB Address Width Parameter Settings

Note: When adding parameters, the parameter name in the IP Packager must exactly match the parameter name in RTL.

7. Add a new parameter and rename it to *FAMILY*.
 - Populate the following settings:
 - Title: Device Family
 - Type: param
 - Value_type: string
 - Default: Nexus
 - Options: ["Nexus", "Avant", "iCE40UP"]

Property	Value
title	Device Family
type	param
value_type	string
default	Nexus
value_expr	
options	["Nexus", "Avant", "iCE40UP"]
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.17. Device Family Parameter Settings

8. Add a new parameter and rename it to *IO_COUNT*.
 - Populate the following settings:
 - Title: Number of LEDs
 - Type: param
 - Value_type: int
 - Default: 32
 - Options: [1, 2, 4, 8, 16, 32]

Property	Value
title	Number of LEDs
type	param
value_type	int
default	32
value_expr	
options	[1, 2, 4, 8, 16, 32]
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.18. Number of LEDs Parameter Settings

9. Add a new parameter and rename it to *USER_INTF*.

- Populate the following settings:
 - Title: User Interface
 - Type: param
 - Value_type: string
 - Default: APB
 - Options: ["APB", "LMMI"]

Property	Value
title	User Interface
type	param
value_type	string
default	APB
value_expr	
options	["APB", "LMMI"]
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.19. User Interface Parameter Settings

10. Create a parameter group and rename it to *Settings*.

11. Add a new parameter and rename it to *INT_EN*.

- Populate the following settings:
 - Title: Enable Interrupt
 - Type: input
 - Value_type: bool

Property	Value
title	Enable Interrupt
type	input
value_type	bool
default	
value_expr	
options	
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.20. Enable Interrupt Parameter Settings

Meta Data	
Basic Info	
Parameters	
General	
Parameters	
ADDR_WIDTH	
FAMILY	
IO_COUNT	
USER_INTF	
Settings	
INT_EN	
Ports	
IN	
INOUT	
OUT	
OutFileConfigs	
wrapper	
Interfaces	
APB_S0	
IRQ0	
Memory Map	
LED_map	

Property	Value
title	APB Address Width
type	param
value_type	int
default	6
value_expr	
options	[(i+1)*2 for i in range(0,16)]
output_formatter	
bool_value_mapping	
editable	
hidden	
drc	
regex	
value_range	
config_groups	
description	

Figure 3.21. IP Package Structure

- Save the IP Package, then click the **Preview IP** icon to view the current IP progress.
You can modify all parameters and settings during IP preview to check whether they were implemented as expected.

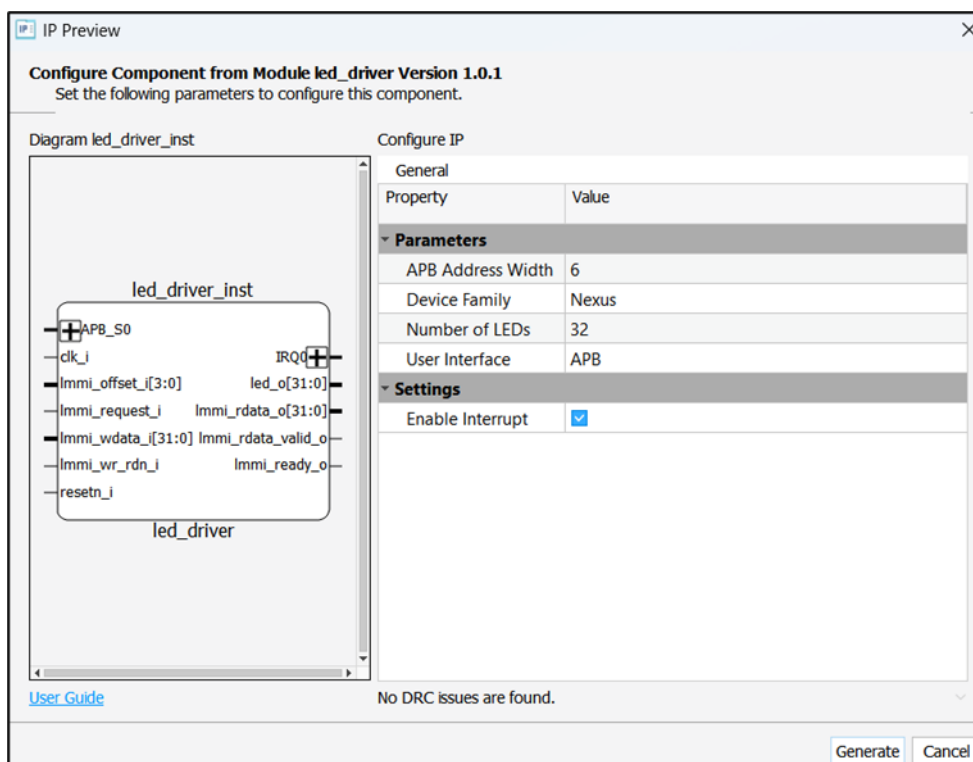


Figure 3.22. IP Configuration on the Preview Page

3.5. Further Customization of the IP

Now that you have finished setting up the parameters and settings for your custom IP package, you can use those parameters to control how other parts of your IP behave during IP generation. A useful feature of the IP Packager is that all of its parameters and settings can be referenced by name and used elsewhere in other settings.

To further customize and control other parts of the IP, follow these steps:

1. On the **Meta Data** page, locate the **Ports** section and add the following *stick_low* setting for all ports beginning with *apb*:
 - *Stick_low* ties ports to 0 whenever its expression is evaluated to be true. Using this setting enables us to control which user interface is active depending on what is selected during IP generation.
 - *Stick_low*: `USER_INTF != 'APB'`.
 - Complete list of signals:
 - `apb_paddr_i`
 - `apb_penable_i`
 - `apb_psel_i`
 - `apb_pwdata_i`
 - `apb_pwrite_i`
 - `apb_prdata_o`
 - `apb_pready_o`
 - `apb_pslverr_o`

Property	
range	(5,0)
conn_port	
conn_range	
stick_high	
stick_low	USER_INTF != 'APB'
stick_value	
dangling	
attribute	
port_type	

Figure 3.23. Stick Low Setting for Ports Beginning with APB

2. Add the following *stick_low* setting for all ports beginning with *Immi*:
 - *Stick_low*: `USER_INTF == "APB"`.
 - Complete list of signals:
 - `Immi_offset_i`
 - `Immi_request_i`
 - `Immi_wdata_i`
 - `Immi_wr_rdn_i`
 - `Immi_rdata_o`
 - `Immi_rdata_valid_o`
 - `Immi_ready_o`

Property	Value
range	
conn_port	
conn_range	
stick_high	
stick_low	USER_INTF == "APB"
stick_value	
dangling	
attribute	
port_type	

Figure 3.24. Stick Low Setting for Ports Beginning with LMMI

3. Locate the *int_o* signal and add the following *stick_low* setting.

Property	Value
range	
conn_port	
conn_range	
stick_high	
stick_low	INT_EN != True
stick_value	
dangling	
attribute	
port_type	

Figure 3.25. Stick Low Setting for int_o Signal

4. Click **Preview IP** to view the current IP development progress.
Try modifying some of the parameters that you configured to see how it impacts IP generation. The exact appearance of the IP updates based on the user interface and enable interrupt selections.

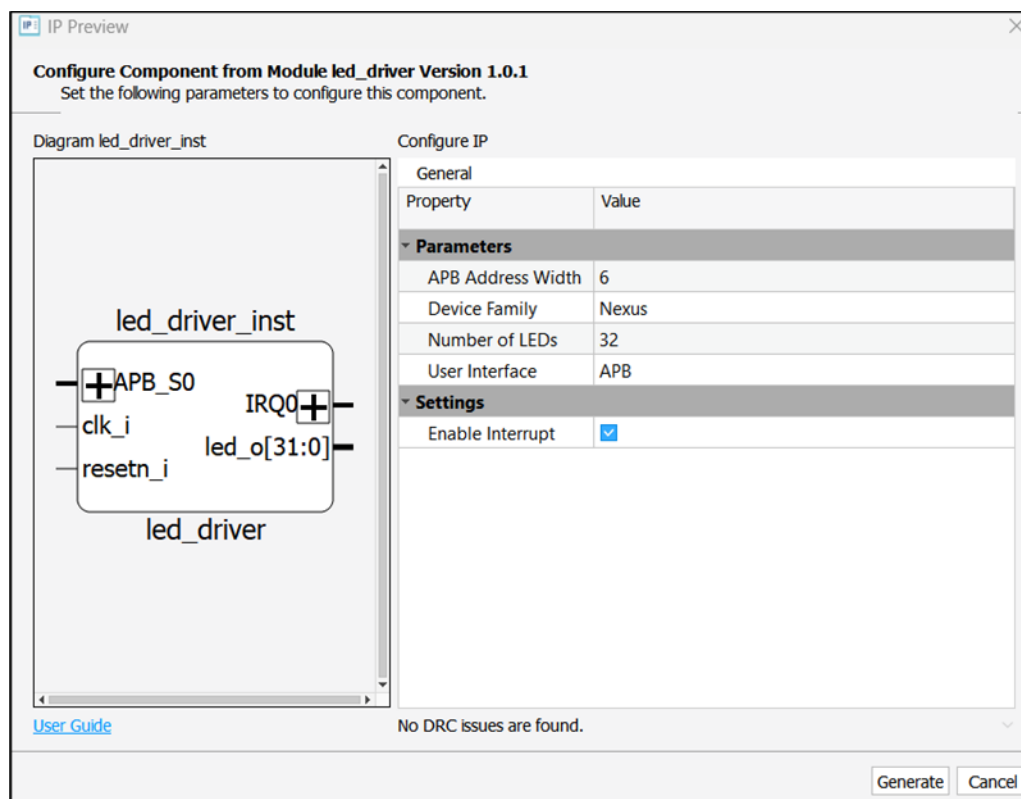


Figure 3.26. IP Diagram Updates According to the IP Configurations

3.6. Packaging and Installing the IP

After setting up all the required and optional files, ports, interfaces, memory map, and parameters or settings, the IP is ready for packaging. The next step is to package the IP and create a .IPK IP installation file that is used to install the IP in the Radiant software or Propel software.

To package and install the IP, follow these steps:

1. Click the **Package IP** icon, or select **Edit** → **Package IP** from the toolbar.

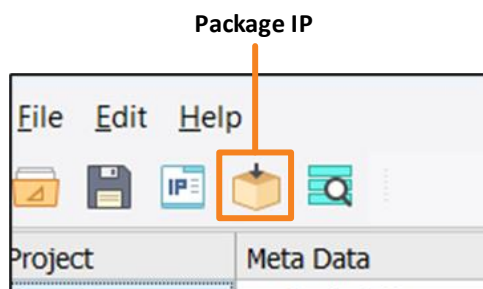


Figure 3.27. Package IP Icon

A message displays on the TCL console indicating where the resultant .IPK file is located.

```
% ipk_package
Package files to IP(C:\2025PROJECTS\IP Packager AN\driver_ip\latticesemi.com_led_driver_1.0.1.ipk) successfully.
%
```

Figure 3.28. Location of the Package Files

2. Locate IP Catalog in either the Radiant software or Propel software and select the **Install Custom IP** icon as shown in the following figure.

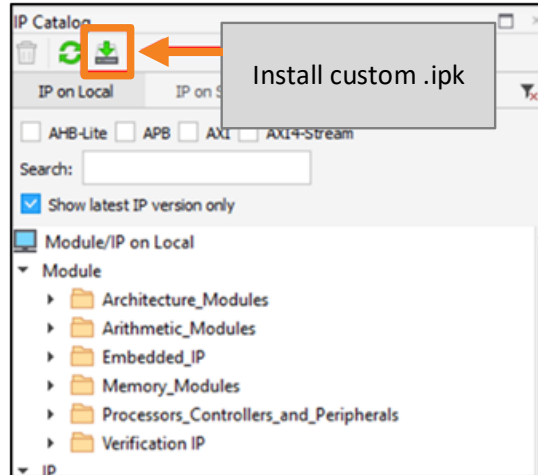


Figure 3.29. Install Custom IP Icon

The custom IP package is installed and can be found in the **IP on Local** tab under **IP → custom** alongside other user installed IP.

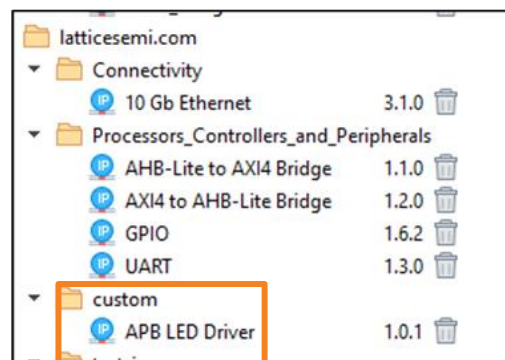


Figure 3.30. Location of the Custom IP Package

References

- [Lattice Radiant Software User Guide](#)
- [Lattice Radiant](#) FPGA design software
- [Lattice Solutions IP Cores](#) web page
- [Lattice Propel Design Environment](#) web page
- [Lattice Insights](#) for Lattice Semiconductor training courses and learning plans

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, April 2025

Section	Change Summary
All	Initial release.



www.latticesemi.com