

# **Vivado Design Suite User Guide**

## ***Creating and Packaging Custom IP***

UG1118 (v2016.3) October 21, 2016

# Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/21/2016	2016.3	<p>Added text to <a href="#">Introduction in Chapter 1</a>, and text about what to do when a GUI banner is not displayed in <a href="#">Importing Parameters from Top-Level HDL Files in Chapter 4</a>.</p> <p>Added an important note on page <a href="#">page 22</a>.</p> <p>Fixed links throughout document.</p> <p>Modified text in <a href="#">Top-Level HDL Requirements in Chapter 2</a>.</p> <p>Modified text in <a href="#">Processing Order of Constraints in Chapter 2</a>.</p> <p>Modified text in <a href="#">Packaging a Design with Global Include Files in Chapter 2</a>.</p> <p>Modified <a href="#">Editing a Packaged Block Design in Chapter 2</a>.</p> <p>Added a recommendation to <a href="#">Editing Your IP in an Existing Project</a>.</p> <p>Add a recommendation to <a href="#">Editing Your IP in a New Editing IP Project</a></p> <p>Added a Recommendation to <a href="#">page 29</a>.</p> <p>Added a section, <a href="#">Using XPMs in Chapter 3</a>.</p> <p>Added text to <a href="#">Using File Groups in Chapter 4</a>.</p> <p>Added text to <a href="#">Updating Ports from a Top-level HDL in Chapter 4</a>.</p> <p>Added text to <a href="#">Creating a New Interface Definition in Chapter 5</a>.</p> <p>Added <a href="#">Chapter 6, Encrypting IP in Vivado</a>.</p> <p>Moved the description of xilinx_enable_netlist_export third column, "Synthesis" to have the same content as its right side cell in column "Implementation in <a href="#">Table 6-5</a>.</p> <p>Removed an example from <a href="#">Syntax, page 91</a>.</p> <p>Added link to the BISTREAM. ENCRYPTION. ENCRYPT property in two locations in <a href="#">Chapter 6, Encrypting IP in Vivado</a>.</p> <p>Added link to the encrypt command in <a href="#">Encrypting IP with Vivado in Chapter 6</a>.</p>
06/08/2016	2016.2	<p>Modified links from UG949 to UG892 for "Revision Control Systems" information.</p>
04/06/2016	2016.1	<p>Updated Chapter 1, Using the Packager Settings. Added information about unsupported ports in Verilog, and a recommendation on how to circumvent in Top-Level HDL Requirements in Chapter 2. Added information about inferring clock and reset interfaces in Inferring Clock and Reset Interfaces in Chapter 2. Added link to UG1198 in Versioning and Revision Control in Chapter 2. Added an important note to Editing an Existing Custom IP. New Graphics in Figure 3-1, Figure 3-2, Figure 3-3, Figure 3-4, and Figure 3-10. Added added a note to UG994 to highlight the Module Reference feature, on page 24. New content on Merge changes in Adding or Removing a Parameter in Chapter 4. Added Editing a Packaged Block Design. Added Inferring Clock and Reset Interfaces. Added information about Adding or Removing Files to File Groups. Added Editing a Packaged Block Design. Added Packaging a Design with Global Include Files.</p>

# Table of Contents

## Chapter 1: Creating and Packaging Custom IP

Introduction . . . . .	5
Supported IP Packager Inputs . . . . .	7
Outputs from IP Packager . . . . .	7
Using the Packager Settings . . . . .	8

## Chapter 2: IP Packaging Basics

Introduction . . . . .	10
Top-Level HDL Requirements . . . . .	10
Inferring Signals . . . . .	12
Packaging a Design with Global Include Files. . . . .	14
Constraints Requirements. . . . .	15
Upgrading Custom IP . . . . .	17
Editing an Existing Custom IP . . . . .	19
Setting an Enablement Expression . . . . .	22
Versioning and Revision Control. . . . .	23

## Chapter 3: Using the Creating and Package IP Wizard

Introduction . . . . .	25
Using the Create and Package IP Wizard . . . . .	26
Packaging Your Current Project . . . . .	28
Packaging a Block Design. . . . .	29
Packaging a Specified Directory . . . . .	31
Creating a New AXI4 Peripheral . . . . .	33
Using XPMs. . . . .	36

## Chapter 4: Packaging IP

Introduction . . . . .	37
Identification . . . . .	37
Using the Compatibility Page . . . . .	41
Using File Groups . . . . .	44
Customization Parameters . . . . .	48
Ports and Interfaces. . . . .	55

Addressing and Memory . . . . .	65
Customization Parameters . . . . .	68
Review and Package . . . . .	71

## Chapter 5: Creating New Interface Definitions

Introduction . . . . .	73
Creating a New Interface Definition . . . . .	74
Using the Interface Definition Editor . . . . .	75
Re-Editing Interface Definitions . . . . .	79
Using a New Interface Definition . . . . .	79

## Chapter 6: Encrypting IP in Vivado

Introduction . . . . .	80
Access Rights Management. . . . .	80
Understanding IEEE 1735 Structural Elements. . . . .	81
Understanding How Rights Affect Vivado Tools . . . . .	88
RTL Encryption Examples. . . . .	89
Encrypting IP with Vivado . . . . .	91
Best Practices . . . . .	93
Known Limitations . . . . .	93

## Appendix A: Standard and Advanced File Groups

Introduction . . . . .	94
Standard File Groups . . . . .	94
Advanced File Groups . . . . .	95

## Appendix B: Additional Resources and Legal Notices and Legal Notices

Xilinx Resources . . . . .	97
Solution Centers. . . . .	97
Documentation Navigator and Design Hubs . . . . .	97
References . . . . .	98
Please Read: Important Legal Notices . . . . .	100

# Creating and Packaging Custom IP

---

## Introduction

Using the Vivado® IP packager flow gives you a consistent experience whether using Xilinx® IP, third-party IP, or customer-developed IP.

[Figure 1-1, page 6](#) shows the flow in the IP packager and its usage model. With the Vivado IP packager, you, as an IP Developer, can:

- Create and package files and associated data in an IP-XACT standard format.
- Add IP to the Vivado IP Catalog.
- Deliver packaged IP to an end-user in a repository directory or in an archive (.zip) file.

After you distribute IP, an end-user can create a customization of that IP in their designs.

Before packaging your RTL as an IP, it is recommended you do the following:

- Verify the design sources by running synthesis (see *Vivado Design Suite User Guide: Synthesis* (UG901) [\[Ref 22\]](#)) and implementation (see *Vivado Design Suite User Guide: Implementation* (UG904) [\[Ref 26\]](#)).
- Verify the design simulates as expected (see *Vivado Design Suite User Guide: Logic Simulation* (UG900) [\[Ref 6\]](#)).
- If using Xilinx parameterized macros (XPMs), see [Using XPMs in Chapter 3](#).

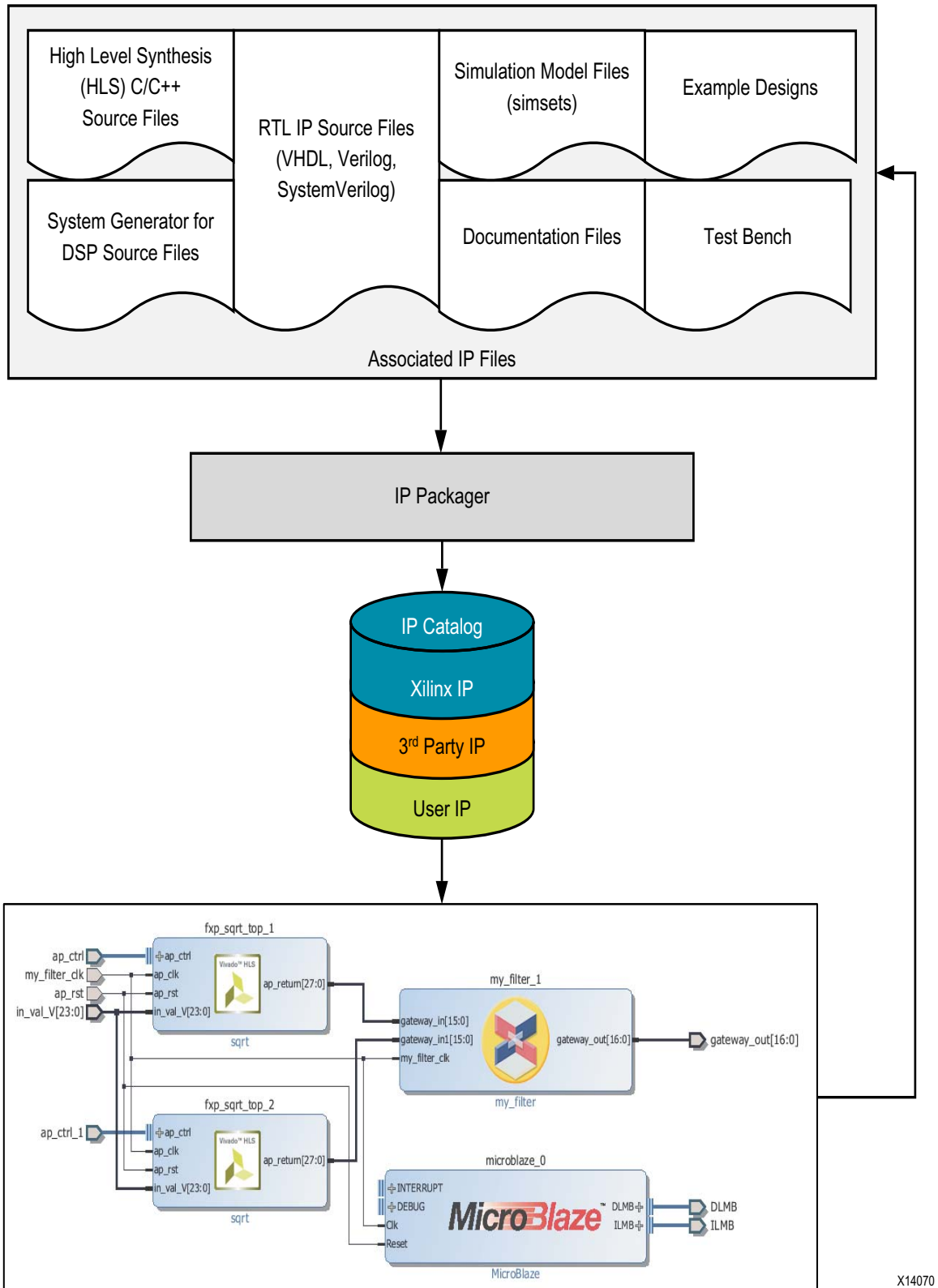


Figure 1-1: IP Packaging and Usage Flow

X14070

## Supported IP Packager Inputs

The Vivado IP packager supports the following input file groups:

- HDL synthesis
- HDL simulation
- Documentation
- HDL test bench
- Example design
- Implementation files (including constraint and structural netlist files)
- Drivers
- GUI customization



**TIP:** Verilog, SystemVerilog, and VHDL source files, or modules and architectures defined within the source files can be encrypted to protect the IP. See [Chapter 6, Encrypting IP in Vivado](#) for more information.

IP packager can designate as many or as few file groups as is appropriate to the IP. There is no requirement for a minimum set of file groups; however, the IP packager **IP File Groups** page presents a *typical* set of file groups, based upon the packaged project sources. When any of these file groups are empty, the final Review and Package page issues a warning about missing file.



**IMPORTANT:** The Vivado IP packager does not support IP in the Core Container format. Disable the Core Container feature for all IP prior to packaging. For more information on Core Container, see this [link](#) in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

## Outputs from IP Packager

The IP packager generates an XML file based on the IP-XACT standard, `component.xml`, and a XGUI customization Tcl file. These two files are generated at the location of the IP root directory. The IP-XACT component XML file identifies the IP definition information. The XGUI customization Tcl file, located in the `/XGUI` folder of the IP root directory location, displays the customization GUI of the custom IP from the IP Catalog.

The associated files of the custom IP are relative to the IP-XACT XML file. If you package the project remotely, the IP packager copies the associated IP files to the selected IP location. The files are categorized in directories based on usage (for example: `/src`, `/sim`, `/doc`).

## Using the Packager Settings

The following steps are to set Packager default behavior in a project. For more information about IP settings, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

To set the Packager options:

1. From **Tools > Project Settings > IP**, click the Packager tab, shown in the following figure:

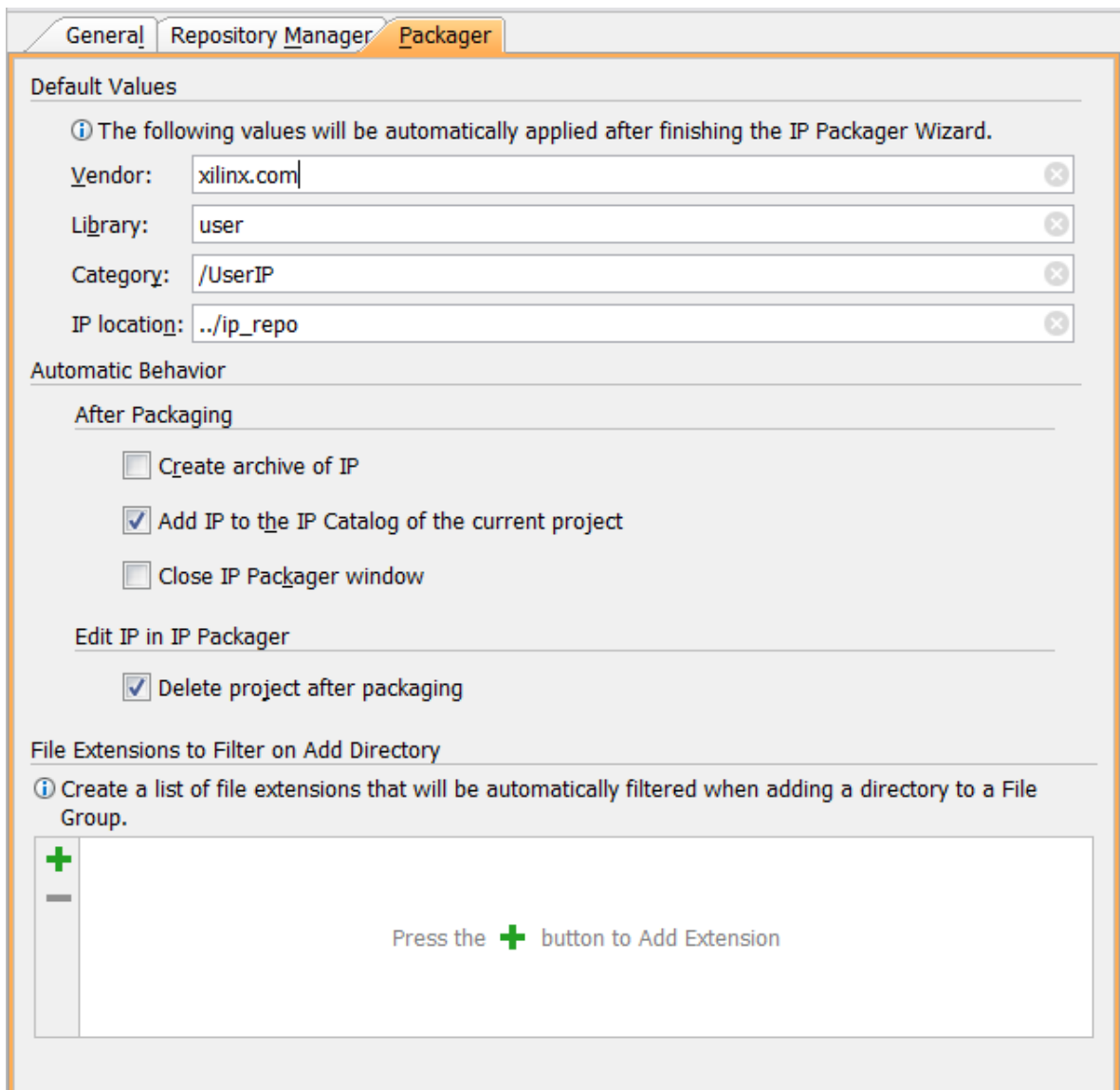


Figure 1-2: IP Project Settings—Packager Tab

2. Set the following options:
  - In **Default Values**:
    - **Vendor**: Sets the vendor name when packaging a new IP. This is, for example, the top domain name of a company.
    - **Library**: Sets the associated library for the IP. This category, along with the Vendor are used in conjunction with the IP name to create a unique identifier.
    - **Category**: Specifies the categories in the IP Catalog in which to place the IP. For example, /UserIP.  
*Note*: If necessary, you can change the default values during the IP packaging process.
    - **IP Location**: Specifies the location to use for your packaged IP.
3. In **Automatic Behavior**, check or uncheck the options you want:
  - **After Packaging**:
    - **Create archive of IP**: After packaging an IP, automatically creates an archive (ZIP format) of the IP. For information on how to set the location of a ZIP file, see [Creating an Archive of the IP](#).
    - **Add IP to the IP Catalog of the current project**: Adds the current IP to the IP Catalog.
    - **Close IP Packager window**: Closes the Package IP window automatically when IP packaging is complete.
4. In the **Edit IP in IP Packager**:
  - **Delete project after packaging**: Removes the iterative editing project after the IP is re-packaged.
5. In **File Extensions to Filter on Add Directory**: Add extensions (for example, TXT) to automatically filter when selecting a directory to include in a **File Group** when packaging an IP.

# IP Packaging Basics

---

## Introduction

This chapter describes some of the basic use cases of creating and packaging custom IP. Some of the topics include HDL requirements, versioning, editing an existing custom IP, and using expressions

It is recommended that certain actions are taken to ensure a smooth and flexible experience when creating your custom IP. These recommendations are for working within the framework of the Vivado IDE.

---

## Top-Level HDL Requirements

The IP packager supports HDL synthesis language constructs for the top-level HDL file of the IP. Following these requirements ensures proper functionality of your custom IP.

- The IP packager currently supports Verilog and VHDL as a top-level; if you have a SystemVerilog top-level design file, create a Verilog wrapper file prior to packaging.
- All IP used within the Vivado IP Catalog support multi-language usage, which allows the end user to generate an HDL wrapper for a language different than your IP.
- To avoid conflicts, avoid using HDL language keywords within the design.
- To ensure that the custom IP simulates properly when using VHDL, set the top-level ports to be `std_logic` or `std_logic_vector`.

Regardless of the top-level port type, when you synthesize the IP out-of-context (OOC), the resulting IP netlist ports are converted to `std_logic` or `std_logic_vector`. The converted netlist ports could cause type mismatch issues with RTL simulation.

For Verilog, module declarations with complex or split ports are not supported.



---

**RECOMMENDED:** *Modify the module declaration to remove these port types, or create a wrapper file around the module to contain only the supported port types for packaging your design.*

---

As an example, the following module declaration contains both complex and split port types:

```
module top({in1, in2}, out[0], out[1]);
  input      in1, in2;
  output [1:0] out;
endmodule
```

## Custom Functions in HDL

The Vivado IP packager requires that the IP ports are self-contained; therefore, the IP packager does not support custom functions defined in your HDL.



**RECOMMENDED:** *The recommendation for ports that require a custom function is to change your algorithm into a supported mathematical expression in either the HDL or the IP packager.*

The Vivado IP packager supports standard arithmetic and logical operators to create the desired mathematical expression. To support complex mathematical expressions, the IP packager supports XPATH functions. The following table lists the supported XPATH functions.

Table 2-1: Supported XPATH Functions

XPATH Function
number <b>max</b> (node-set)
number <b>min</b> (node-set)
number <b>sum</b> (node-set)
number <b>log</b> (base number, number)
number <b>pow</b> (number, exp number)
number <b>floor</b> (number)
number <b>ceiling</b> (number)
number <b>round</b> (number)
number <b>abs</b> (number)
boolean <b>not</b> (boolean)
boolean <b>true</b> ()
boolean <b>false</b> ()

For more information on XPATH and the supported functions, see the [W3C](http://www.w3c.org) website.

For example, the following Verilog code declares an output port whose width is defined from a `ceil_log2` function call on the `max_count` parameter. The function calculates the log base 2 of the input and returns the smallest integer that is *not less* than the log result.

```
output [ceil_log2(max_count)-1:0] count;
```

To convert this function into an expression that the IP packager can use, replace the custom function with XPATH functions. This change does not occur in the HDL because the custom function is still used, but the IP packager uses a different mechanism for calculating the correct value. The following XPATH expression produces the same result as the custom function as described.

```
ceiling(log(2, $max_count))-1
```

The expression contains the XPATH `log()` function which is passed the base 2, and the value of the `max_count` parameter. The output of the `log()` function is passed to the `ceiling()` function to return the smallest integer not less than the log result. Finally, one is subtracted from the final ceiling result.

For more information on setting the XPATH functions on ports in your custom IP, see the [Ports and Interfaces](#) section.

## Inferring Signals

This section describes inferring clock and reset interfaces, and gives a brief description of AXI signals.

### Inferring Clock and Reset Interfaces

The Vivado IP packager can automatically infer interfaces for clock and reset signals for your IP. This helps the use of custom IP in the IP integrator for validating clock and reset signals within the block diagram.

There is a required nomenclature to properly infer clock and reset interfaces.



**IMPORTANT:** *The IP Packager checks for the `ASSOCIATED_BUSIF` parameter for all clock interfaces. The reason for the warning is that IP Integrator works best with interfaces, and it was expected that user would typically be using AXI interfaces. If you do not have any bus interfaces in your design, you can safely ignore this warning. For more information on parameters related to clock interfaces, see this [link](#) in the Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994) [Ref 17].*

The following table describes how the reset is inferred based on the naming of the signal.

If the reset signal does not contain the required nomenclature, the interface can be manually created and the properties set accordingly.

**Note:** The `[*_]` and `[_*]` are optional and the `*` matches any text.

**Table 2-2: Reset Signal Naming**

[*_]reset[_*]
[*_]resetin
[*_]resetn
[*_]rst
[*_]rstin
[*_]aresetn

For reset signals that end with *n* such as *resetn* and *aresetn* which implies an active-Low signal, the interface automatically sets the `POLARITY` parameter to `active_Low`. This parameter is used in the Vivado IP integrator to determine if the reset is properly connected when the block diagram is generated. For all other reset interfaces, the `POLARITY` parameter is not set, and is determined through the parameter propagation feature of IP integrator.

For more information on parameter propagation and IP integrator, see this [link](#) in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)[Ref 17].

## Inferring Clock Interfaces

To properly infer a clock interface, the clock signal needs to have a required nomenclature as well. The following table describes how the clock signals are named.

If the clock signal does not contain the required nomenclature, the interface can be manually created and the properties set accordingly.

**Table 2-3: Clock Signal Naming**

[*_]clk
[*_]clkkin
[*_]clock[_*]
[*_]aclk
[*_]aclkkin

## Inferring Differential Clock Interfaces

To properly infer a differential clock interface, each pin of the differential pair requires the same nomenclature, as shown in the following table.

**Table 2-4: Differential Clock Interface Naming**

[*_]clk_p
[*_]clk_n

## Inferring AXI Signals

Xilinx has adopted the Advanced eXtensible Interface (AXI) protocol for its Intellectual Property (IP) cores, the use of the protocol in your custom IP gives the flexibility to connect with other IP in the Vivado IP Catalog. If the port signals of your custom IP adhere to the AXI naming conventions, the Vivado IP packager automatically infers the AXI interface.

The proper nomenclature to infer the AXI interface is to ensure that the port name consists of an interface name followed by the AXI signal name. Any name can be used for the interface as long as the name is consistent for each port. The following table is an example naming convention for inferring an AXI4-Stream interface with interface name `s0_axis`.

Table 2-5: Port Naming for AXI4-Stream Interface Nomenclature

<code>s0_axis_tdata</code>
<code>s0_axis_tvalid</code>
<code>s0_axis_tready</code>
<code>s0_axis_tstrb</code>
<code>s0_axis_tkeep</code>
<code>s0_axis_tlast</code>
<code>s0_axis_tid</code>
<code>s0_axis_tdest</code>
<code>s0_axis_tuser</code>

For more information regarding the AXI interface, see the *Vivado AXI Reference Guide (UG1037)* [Ref 18].

---

## Packaging a Design with Global Include Files

The Vivado IDE supports designating Verilog or Verilog Header files as global ``include` files to process before any other sources.

**Note:** This feature is not supported when packaging a custom IP.

After packaging, the Vivado tool treats global ``include` files as standard Verilog or Verilog Header files.

To package a design which uses global ``include` files, you must modify the HDL to place the ``include` statement at the top of any Verilog source file that references content from another Verilog or Verilog header file.

## Constraints Requirements

As with Xilinx IP, the IP packager supports the inclusion of constraints files (XDC) along with RTL files.

The IP packager supports the following types of XDC files, as follows:

- XDC to be used during synthesis and implementation (default)
- XDC to be used only during implementation
- XDC for use only during out-of-context synthesis

This section describes the following concepts:

- [Managing Out-of-Context Constraints](#)
- [Processing Order of Constraints](#)

For more details on IP constraints, see *Vivado Design Suite User Guide: Using Constraints* (UG903) [\[Ref 5\]](#), and for constraints in general see *Vivado Design Suite User Guide: Synthesis* (UG901) [\[Ref 22\]](#).

### Managing Out-of-Context Constraints

By default, IP are synthesized out-of-context (OOC) of the top-level design. When you use the out-of-context flow, a special OOC flow-only XDC is required for sequential logic timing during Vivado synthesis. This XDC file provides clock definitions for clocks from the top-level, either from the user or from another IP.

When creating your custom IP, it is recommended that you include an OOC XDC file to provide these clock definitions for when synthesizing the IP standalone. For more information on the Out-of-Context flow, see the following documents:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 8\]](#)
- *Vivado Design Suite User Guide: Synthesis* (UG901) [\[Ref 22\]](#)

**Note:** For Xilinx delivered IP, the out-of-context XDC file has `_ooc` appended to the filename. This is not a requirement, because the `USED_IN` file property determines if it is an OOC XDC file, not the filename.

In a typical design, an IP receives some of its required constraints from the top-level design. For example, if your custom IP does not directly interface with the device boundary, it typically expects the parent design to supply the input clock definitions.

The OOC XDC file generally only contains these input clock definitions.

This is required so that clock definitions will exist when the IP is synthesized out-of-context of the top-level constraints. While this is not a requirement, it is not recommended to add any additional constraint types. During implementation of the entire design, the IP netlists link with the top-level netlist and the OOC XDC is not required.

There are two instances in which an IP should have its input clock definition in the IP XDC file rather than the OOC XDC file, which are:

- IP contains a clock definition connected to an input buffer
- IP contains a clock definition internal to the IP

In the out-of-context mode, Vivado synthesis does not insert I/O buffers. If your custom IP port has an instantiated input buffer, leave the input clock definition in your IP XDC file. If the input clock definition is defined on an internal element of the IP, such as a Flip-Flop or a GT, leave the input clock definition in the IP XDC file. If your use case does not fall into one of the two categories, move the clock definition to the OOC XDC file.

After the OOC XDC file is created, set the `USED_IN` property to `out_of_context`. This marks the XDC file to be processed in the out-of-context flow only.




---

**IMPORTANT:** *The `USED_IN` property for an OOC XDC file should be `{synthesis implementation out_of_context}`. If it is only set to `out_of_context`, it is not used during synthesis or implementation.*

---

## Processing Order of Constraints

The Vivado IP that deliver constraints are processed either before or after the user design constraints. For more information on using constraints, see this [link](#) in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 5].




---

**IMPORTANT:** *An XDC marked as `OUT_OF_CONTEXT` is processed before all other XDC files used in the IP, including those marked to have a `PROCESSING_ORDER` value of `EARLY`. The default value of `PROCESSING_ORDER` is `NORMAL`. There is also a `LATE` value.*

---

When creating your custom IP, you must determine the order in which the constraints are processed in the context of a user design. The IP packager inherits the processing order set on the constraint files in the project. Set these properties properly in your project prior to packaging the IP.

The constraints delivered for an IP only support two processing orders: `EARLY` and `LATE`. All constraints marked as the default of `NORMAL` are converted to `EARLY`.

By default, top-level user constraints have the processing order set to `NORMAL`, which comes between `EARLY` and `LATE`. This way an IP can provide output clocks that the top-level reference in an `EARLY` XDC as well as allow for the top-level constraints to override an IP constraint if needed.

If an IP constraint has a dependency to a top-level constraint, such as a top-level clock object, place the constraint in an XDC marked as `LATE`. This ensures that the required object is present when the IP constraint is processed. In the case of the IP being synthesized out-of-context, the OOC XDC provides the top-level clock object.

**Note:** Xilinx IP XDC files that have their `PROCESSING_ORDER` property set to `LATE` are named `<IP_NAME>_clocks.xdc`. This naming convention is not required for a custom IP.

Most constraints for an IP belong in the `EARLY` processing order. The constraints marked for `LATE` require user constraints prior to processing. This generally refers to clock dependencies.

## Upgrading Custom IP

A custom IP upgrades as every other IP in the Vivado IP Catalog. The difference is how the custom IP is edited to create a new version or revision to upgrade to. For more information on the process of upgrading IP, see this [link](#) in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

For custom IP, you do not have to upgrade when a newer version is available in the IP Catalog. All versions, if accessible in the IP Catalog, are available for customization and use; however, when using Xilinx IP in your custom IP, moving to a new Vivado release could cause the custom IP to become locked.

This IP lock is because the IP used in the custom IP must be upgraded. If you do not want to update any information for the custom IP, you must ensure all the output products are fully generated from the previously supported release.



**IMPORTANT:** *For Xilinx IP, only one version of an IP is delivered in each release of the Vivado Design Suite.*

In the report IP status example, shown in the following figure, the status reports the IP definition has missing subcores.

In this case, the subcore is missing because the Xilinx IP used by the custom IP has a new version. Because Vivado only supports the latest version of a Xilinx IP, the previous version originally used when creating the custom IP no longer exists.

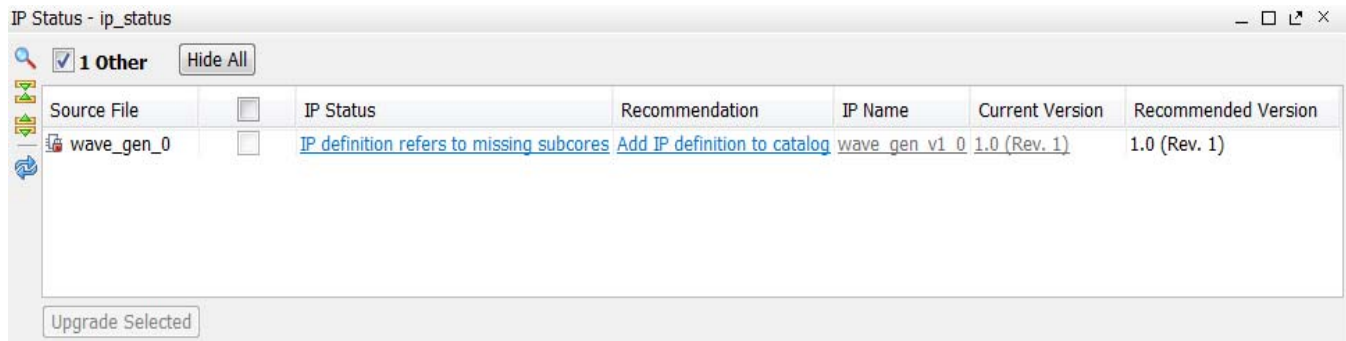


Figure 2-1: Locked Custom IP in Report IP Status View

To create a new version of the custom IP, you must edit the IP in an Editing IP project or in the original project from which the IP was packaged. For more information on creating and using an Editing IP project, see the [Editing an Existing Custom IP](#) section.



**RECOMMENDED:** Review the *Versioning your Custom IP* section to understand how to properly maintain versions of your custom IP.

In the Vivado project used to edit the custom IP, upgrade the Xilinx IP. After the upgrade of the IP is complete, ensure that the custom IP operates as expected and make any additional modifications to keep functionality.

After the custom IP is updated with the necessary changes, ensure the Package IP view has fully merged the changes and repackage the IP. For more information on the Package IP view, see the [Packaging a Specified Directory in Chapter 3](#) section.

Referencing the previous report IP status example, in the following figure, with the updated custom IP, the report IP status view now reflects the new available version.

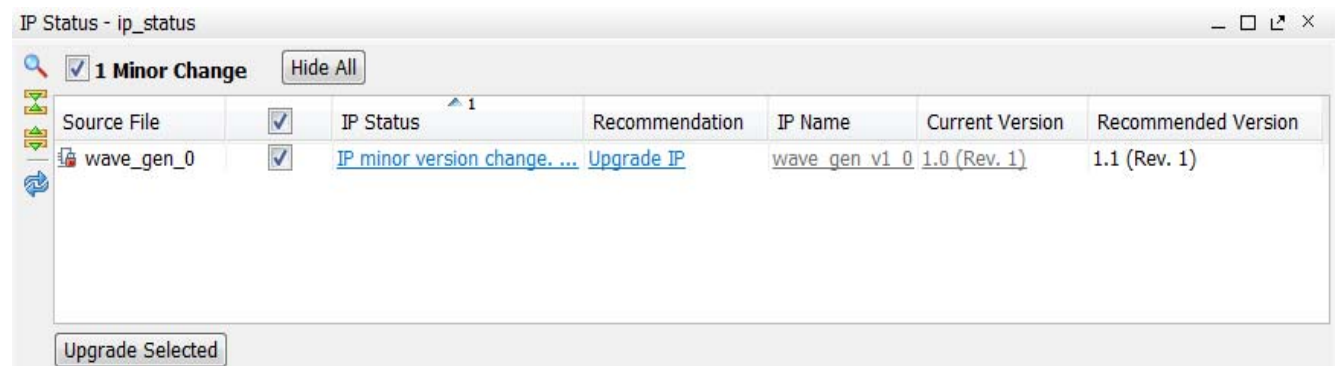


Figure 2-2: Custom IP with Minor Version Change

The custom IP can now be upgraded to the latest version which is supported with the current Vivado release.

## Editing an Existing Custom IP

After you create a custom IP, you can modify that IP. There are a few possible options for editing the existing custom IP:

- Edit the IP in an existing project
- Edit the IP in a new editing IP project

For more information on creating a version for your custom IP, see [Versioning your Custom IP](#).




---

**IMPORTANT:** When you package a custom IP, the IP definition is placed at a location relative to the project source files. Add the files to your custom IP below the IP definition file; this ensures that the newly merged files are added as relative paths instead of absolute paths.

---

### Editing Your IP in an Existing Project

To edit your IP in an existing project:

1. From the **Flow Navigator**, select **Package IP**.  
**Note:** This option only exists for projects with an associated IP-XACT component .xml file.
2. With the Package IP view open, add, remove, or modify the source files in the project or adjust settings in the **Packaging Steps**. For more information on each of the packaging steps, see [Chapter 4, Packaging IP](#).
3. After you complete the modifications to your custom IP, select the Review and Package step in the Package IP view and select **Re-Package IP**.

---

**IMPORTANT:** When you package a custom IP, the IP definition file (xml) is placed at a location relative to the project source files. Add the files to your custom IP in a directory below the IP definition file (xml); this ensures that the newly merged files are added as relative paths instead of absolute paths.

---

### Editing Your IP in a New Editing IP Project

If you do not have the original packaging project available, or would like to edit the IP in a new project, you can open a new editing IP project from the Vivado IDE as follows:

1. Open any Vivado project with the repository for your custom IP.
2. From the **Flow Navigator**, select the **IP Catalog**.
3. Right-click the custom IP you would like to edit, and select **Edit in IP Packager**.

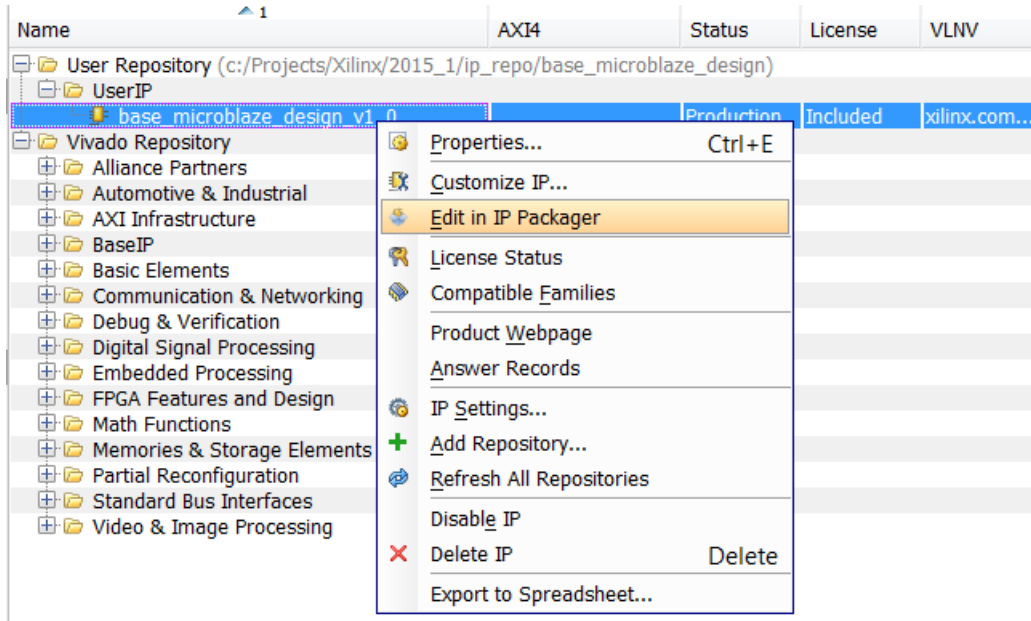


Figure 2-3: Edit in IP Packager Option

4. The Edit in IP Packager dialog box opens for you to select the name and location of the edit IP project, as shown in the following figure.

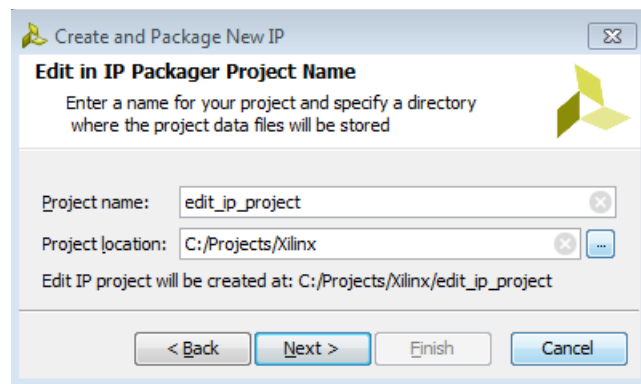


Figure 2-4: Edit in IP Packager Dialog Box

5. Click **OK**.

A new Vivado project opens that contains the contents of the custom IP. The component .xml from the custom IP is associated with the new edit IP project and the Package IP view becomes available.

From this project, you can add, remove, or modify the source files in the project or adjust settings in the packaging steps.

For more information on each of the packaging steps, see [Chapter 4, Packaging IP](#).




---

**IMPORTANT:** *When adding files to your custom IP through the edit IP project, ensure that you have selected the **Copy sources into IP Directory** in the **Add Sources** dialog box.*

---

6. After you complete the modifications to your custom IP, select the **Review and Package** step in the Package IP view and select **Re-Package IP**.

You are prompted if you would like to close the edit IP project.

7. Select **Yes** and, if you have previously selected the **Delete project after packaging option** in the IP Packager project settings, the edit IP project is deleted.

You can always open a new edit IP project if more edits are required.

## Editing a Packaged Block Design

When packaging a block design, the IP definition contains the files associated with the block design (BD); however, the diagram for the BD is not maintained.

Only the HDL wrapper files which represent the diagram are included in the IP definition along with the associated IPs that were used within the BD.




---

**IMPORTANT:** *Because the diagram of the block design is not maintained, it is not recommended to modify the packaged Block Design through the Edit in IP Packager flow.*

---

When opening a packaged block design in the **Edit in IP Packager** flow, the project is not recognizable compared to the original source block design. The modifications required in this state are, as follows:

- Add the newly required IP or edit one of the existing IPs similar to a standard RTL project.
- Manually edit the top-level wrapper file to connect and instantiate any additional or edited IP.

To modify a packaged block design (BD), follow the recommended method:

1. Add the original BD source in a new Vivado project or use the original Vivado project that contained the BD source.
2. Open the BD and make any necessary modifications.
3. Package the BD following the [Using the Create and Package IP Wizard](#).
4. Return to the Vivado project that uses the custom IP and upgrade the IP to the latest version.




---

**IMPORTANT:** To avoid errors when elaborating a BD that was packaged as an IP, name the BD something other than the default name. If the packaged BD uses the default name, and if you were to use this in a project and add it to a BD with which also uses the default name (or any name that matches), you fail synthesis.

---

When packaging the BD after the modifications, it is important to use the same vendor, library, and name as for the previous IP VLNV. If the same information is not used, the upgrade process is not available for the modified custom IP. For more information about versioning, see [Versioning your Custom IP](#).

---

## Setting an Enablement Expression

In the Vivado IP packager, you can use an expression for the following methods:

- Enabling or disabling ports or interfaces
- Enabling or disabling customization parameters
- Calculating the value of a customization parameter

The syntax that evaluates the expression can reference parameters defined through the Customization Parameters page in the Package IP view.

The variable name of the parameter in the expression view is the parameter name. The variable name is case-sensitive and requires a (\$) sigil prefix before the variable name. For example, a parameter named BAUD\_RATE has an expression variable reference of \$BAUD\_RATE.

To create a functioning enablement or an editable expression, use the variable names with numeric and comparison operators to form a Boolean expression. For example:

```
$BAUD_RATE == 115200 || $BAUD_RATE == 57600
```

- To create a functioning dependency expression, use the variable names to form a mathematical expression to generate a value. For example:

```
$NUM_OF_BYTES*8
```

**Note:** See <http://wiki.tcl.tk/583> to ensure conformity with expression rules.

# Versioning and Revision Control

## Versioning your Custom IP

Custom IP created by the IP packager contains a field for the version. The version information is set in the Identification page of the Package IP view, see [Chapter 4, Packaging IP](#).

A <major#.minor#> numbering scheme unifies the IP version numbers, and a revision number field distinguishes slight changes in the same version. For more information, see the Xilinx IP Versioning page available from the Xilinx website: *Vivado IP Versioning* [Ref 1].

Each time the IP is re-packaged, if the version is not changed, the IP packager increments the revision number automatically.

The Vivado IP Catalog supports only a single version of a Xilinx-created IP per release; however, custom IP is not required to adhere to this behavior.

A custom IP can have multiple versions available in the IP Catalog, and you can select any available version. You can also upgrade the currently used version of a custom IP to the latest available version.




---

**IMPORTANT:** *The IP packager does not automatically archive and save previous versions of custom IP. Each time the IP is re-packaged, the IP location overwrites the data with the newer version of the IP.*

---

To save a specific version of your IP prior to re-packaging, the recommended flow is to manually copy the packaged IP directory contents to a new location.

The IP Catalog can point to this new location to ensure the catalog can show both the original and updated version of the custom IP.

For more information on setting the custom IP location, see [Using the Create and Package IP Wizard in Chapter 3](#).

## Using Revision Control

When using revision control with your custom IP definition, the recommendation is to use an external repository location for your custom IP. By default, the Create and Package IP Wizard chooses a location relative to the project source files. By selecting a location outside of the project, the custom IP is separated from the project structure. Placing the project structure into revision control is *not recommended*. For more information on setting the custom IP location, see the [Using the Create and Package IP Wizard](#).



---

**RECOMMENDED:** *When you create an external repository, place the entire custom IP directory into the revision control system to preserve all the necessary outputs from the IP packager.*

---

For more information about revision or source control in the Vivado Design Suite, see:

- This [link](#) in the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 3].
- *Vivado Design Suite Tutorial: Interfacing with Revision Control Systems* (UG1198) [Ref 25].

You might also want to review the following video:



---

**VIDEO:** See [Vivado Design Suite QuickTake Video: Vivado Design Suite Revision Control](#).

---

# Using the Creating and Package IP Wizard

---

## Introduction

The Vivado® Integrated Design Environment (IDE) Create and Package IP wizard lets you create and package the following:

- IP using source files and information from a Vivado Design Suite project
- Block Diagrams
- IP from a specified directory
- A template AXI4 peripheral that includes:
  - HDL files
  - Drivers
  - A test application
  - A bus functional model (BFM) (which requires special licensing)
  - An example template

The Create and Package IP wizard can generate Xilinx-supported AXI interfaces. These are:

- **AXI4:** For memory-mapped interfaces, which allows burst of up to 256 data transfer cycles with a single address phase.
- **AXI4-Lite:** A light-weight, single transaction memory-mapped interface.
- **AXI4-Stream:** For high-speed streaming data.

For more information on the Xilinx adoption of AXI, see the Vivado *AXI Reference Guide* (UG1037) [Ref 18].

**Note:** For the simple purpose of adding custom RTL for use in IP integrator, the Module Reference feature is available and described in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 17].

## Using the Create and Package IP Wizard

From an open Vivado Project, the Create and Package New IP wizard takes you step-by-step through the IP creation and packaging steps.

To run the Create and Package New IP wizard:

1. From the **Tools** menu, select **Create and Package IP**, as shown in the following figure.

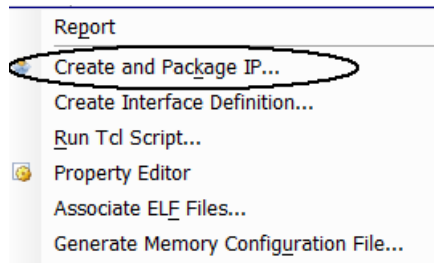


Figure 3-1: Create and Package IP Menu

The first page of the Create And Package IP wizard opens, as shown in the following figure.



Figure 3-2: Create and Package IP Wizard

Use the wizard to accomplish one of these tasks:

- **Package a new IP for the Vivado IP Catalog:** Guides you through the process of creating a new Vivado IP using source files and information from your current project or specified directory.
- **Create a new AXI4 Peripheral:** Create a new AXI4 peripheral that includes HDL, drivers, a test application, and a BFM example template.



**IMPORTANT:** You must acquire a BFM licensing file if you intend to use BFM models.

2. Click **Next**.

The Create Peripheral, Package IP, or Package a Block Design page opens, as shown in the following figure.

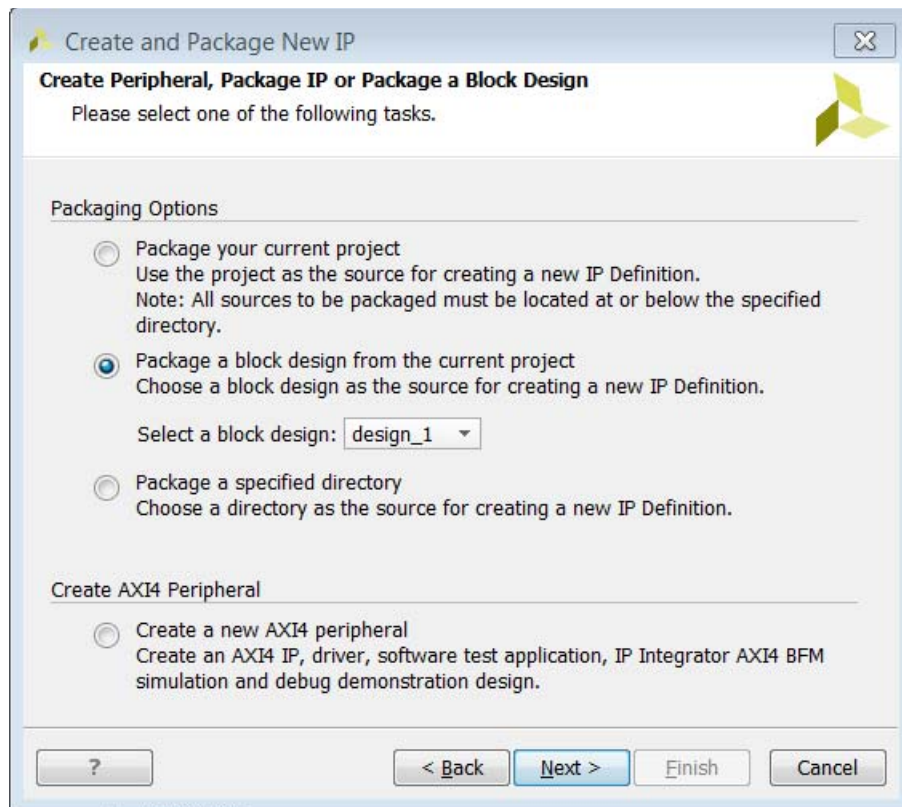


Figure 3-3: Create and Package IP: Create Peripheral, Package IP, or Package a Block Design

3. Select from the options:

- **Package your current project:** See [Packaging Your Current Project](#).
- **Package a block design from the current project:** See [Packaging a Block Design](#).
- **Package a specified directory:** See [Packaging a Specified Directory](#).
- **Create a new AXI4 peripheral:** See [Creating a New AXI4 Peripheral](#).

4. Make your selection, and click **Next**.

The next dialog box option differs, based upon the **Choose Create or Package** option you selected.

## Packaging Your Current Project

The **Package Your Current Project** option lets you package the files associated with your current Vivado project. The IP packager attempts to gather the necessary information about your IP and create a basic IP package in a specified location. When you select this option, the wizard page updates with the available options for packaging the current project as shown in the following figure.

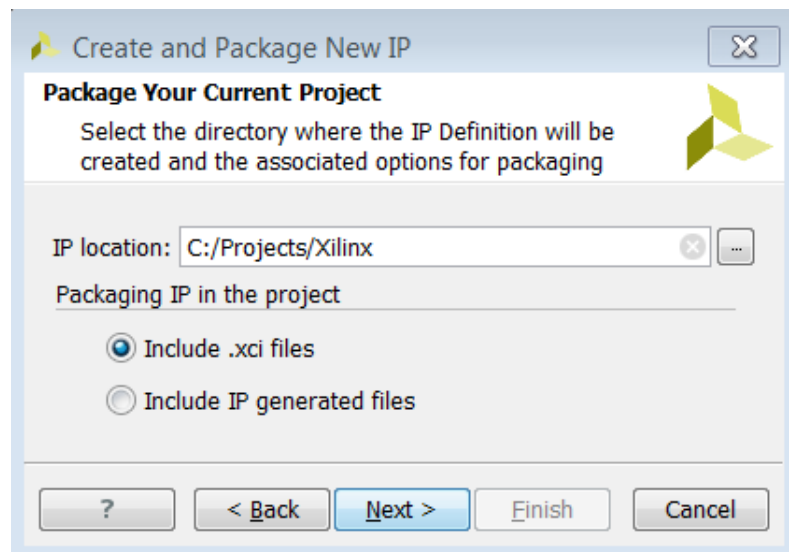


Figure 3-4: Create and Package New IP: Package Your Current Project

1. In the Package Your Current Project dialog box, (Figure 3-4), make your selections from the following options:
  - **IP Location:** The directory in which the IP Packager creates the IP Definition. The default is the project sources directory. The IP packager gives the user options on how they would like to package files in the current project. Generally, the IP location is the `/sources` directory of the project. This location, if all the files are copied into the project, is relative to all the project files. If files are stored remotely from the project source directory, the IP location is determined based upon where the majority of project files are relative.

If a location is selected outside of the hierarchical file paths of the project files, the Vivado IDE prompts you to copy the project source files into the indicated IP location directory. This process copies all the remote files to the IP location into a directory based on their category (for example, `src/`, `sim/`).

The IP packager heuristic always copies remote files based on the file category regardless of IP location name.

The Vivado IDE creates a new temporary editing project in the IP location for editing and modifying the newly created packaged project.

- **Packaging IP in the project:** If the project you are packaging includes IP, the following options determine how the IP is included in the newly packaged IP.
  - **Include .xci files:** Packages only the IP customization file. The Vivado IDE generates the IP output products with the newly created parent IP.
  - **Include IP generated files:** Packages the generated HDL and XDC sources from the IP customization.

When including only the XCI files in the packaged IP, this creates an association between the parent IP and enables the packaged XCI files to be managed by the Vivado IDE.

The advantage of the Vivado IDE managing the packaged XCI is that the IP can be upgraded to the latest release by using the IP upgrade instructions as described in this [link](#) to *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].




---

**RECOMMENDED:** *As only one version of Xilinx IP is delivered in each release, the parent IP can become locked if the associated XCI file has a new release. Vivado will attempt to automatically upgrade the XCI files as they are delivered; however, we strongly recommend that you repackage the parent IP with an upgraded XCI from the latest Vivado Design Suite to maintain functionality.*

---

In the case of including the IP generated files, all the generated HDL and XDC output products of the IP customization are packaged. This removes any reference of the original IP customization and treats the IP as project source files.

2. Click **Next**.

The New IP Creation page summarizes the information that the Create and Package IP wizard heuristically gathers about the design.

3. Click **Finish** to complete packaging and open the IP packager.
4. When this step completes, review the packaging steps shown in the Package IP view in [Chapter 4, Packaging IP](#).

---

## Packaging a Block Design

The **Package a block design from the current project** option lets you package the files associated with a Block Design in your current project.

The IP packager attempts to gather the necessary information about your Block Design and create a basic IP package in the staging area, as shown in the following figure.

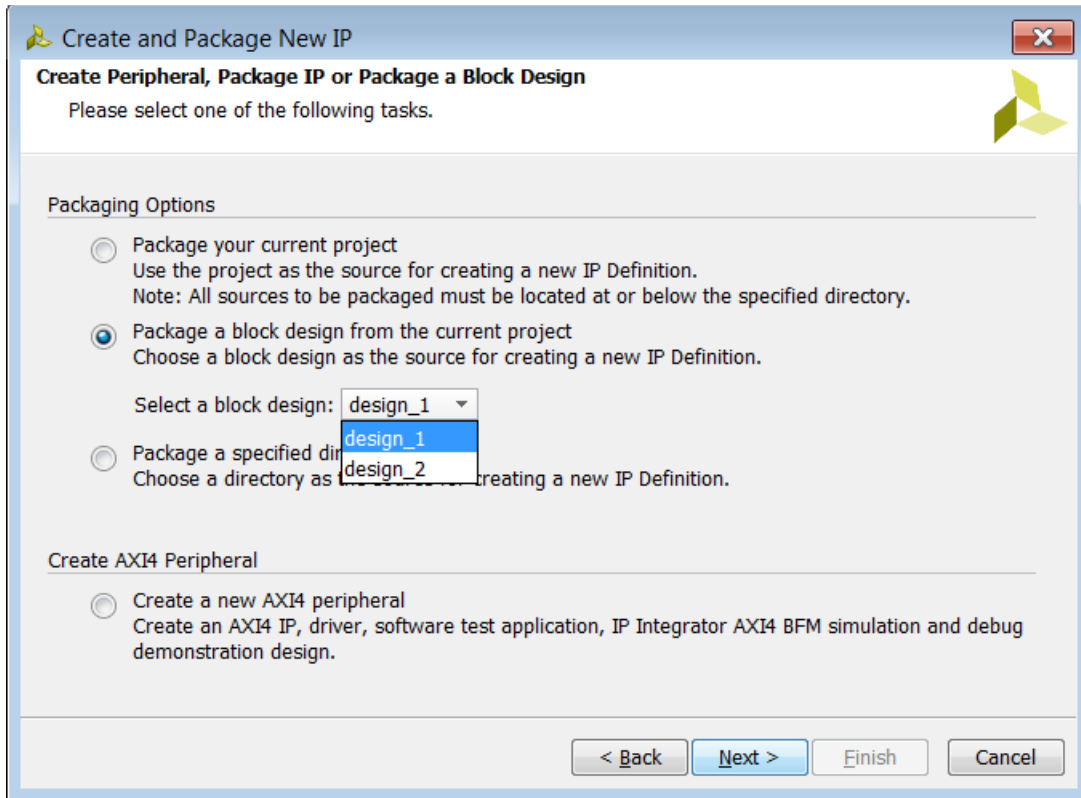


Figure 3-5: Package a Block Design from the Current Project Selection

1. Select the **Package a block design from the current project option**.

The drop-down menu allows the selection of the desired block design to package, as shown in Figure 3-5.



**IMPORTANT:** *Ensure that your block design is open prior to packaging.*

The drop-down list shows the block designs that are available in the current project:

2. In the Packaging Your IP dialog box, shown in the following figure, make your selections from the following options:
  - **IP Location:** The directory in which the IP packager creates the IP Definition. The default is the project sources directory.
  - **Packaging IP in the project:** Because the Block Design includes IP, the following options determine how the IP is included in the newly packaged IP.
    - **Include .xci files:** Packages only the IP customization file. The Vivado IDE generates the IP output products with the newly created parent IP.

- **Include IP generated files:** Packages the generated HDL and XDC sources from the IP customization.

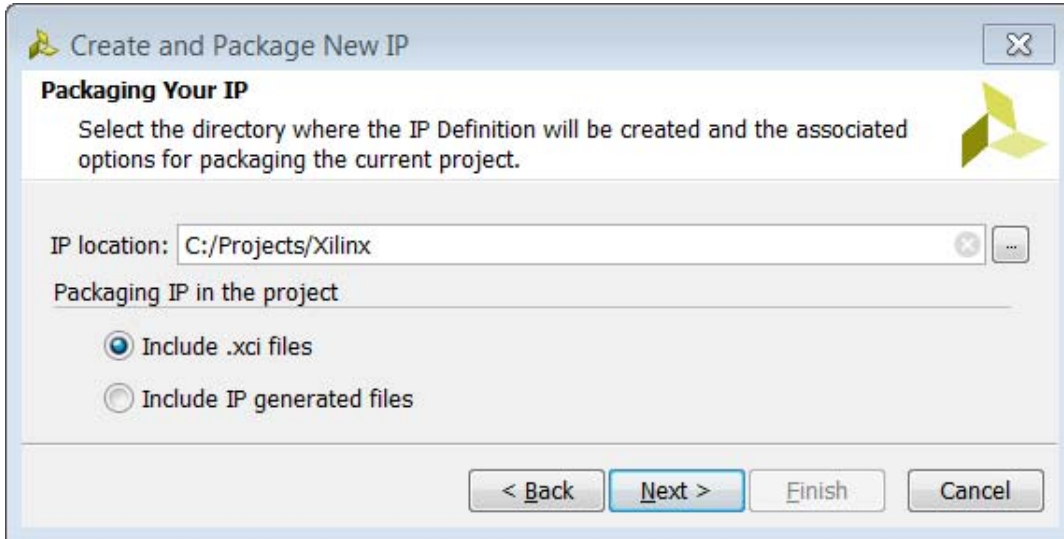


Figure 3-6: Packaging Your IP

3. Click **Next**.

The New IP Creation page summarizes the information that the Create and Package IP wizard heuristically gathers about the design.

4. Click **Finish** to complete the packaging, and open the Package IP view.



**VIDEO:** For more information, see the [Vivado Design Suite: QuickTake Video: Packaging Custom IP for sing in IP Integrator](#).

When this step completes, review the packaging steps in [Chapter 4, Packaging IP](#).

## Packaging a Specified Directory

When you package a specified directory, you can package the files in a specific directory within the file system. There are inference rules which assist in packaging the IP correctly. The following table describes the directory structure recommended for inferring an IP:

Table 3-1: Directory Inference Recommendation

Source Type	Directory Inference
Synthesizable Sources	src/, hdl/
Simulation Sources	sim/, simulation/
Example Sources	example/, ex/, examples

Table 3-1: Directory Inference Recommendation (Cont'd)

Source Type	Directory Inference
Testbenches	testbench/, tb/, test/
C Sim Models	cmodel/, c/
Documents	docs/, doc/, documents/

Using the directory structure shown in Table 3-1, the IP packager attempts to populate the contents into each corresponding file group. In the synthesizable sources directory, the files are filtered by the .sv, .v\*, and .xdc extensions. For all other directories, files are populated.

- If the simulation directories do not exist, the IP packager populates the synthesizable sources into the simulation file sets.
  - If the directory structure of the specified directory cannot be recognized, the IP packager recursively searches for synthesizable source files, and adds files to the synthesis and simulation file groups.
1. Select the **Package a Specified Directory** option and the Package a Specified Directory wizard page opens, as shown in the following figure.

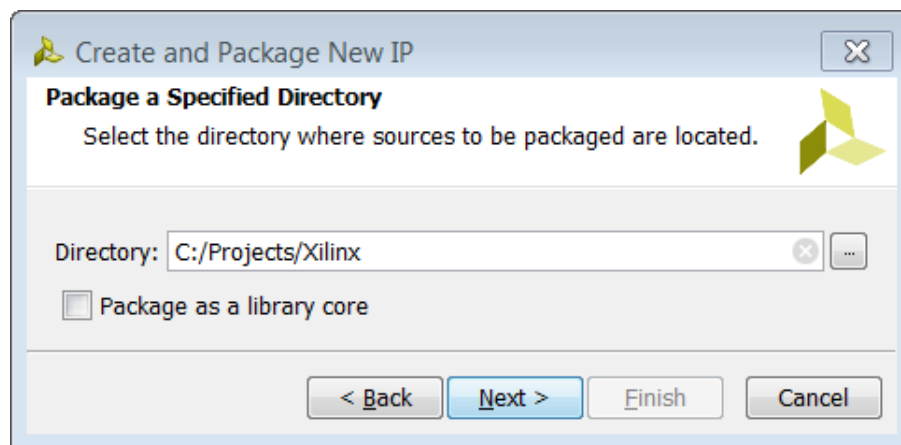


Figure 3-7: Create and Package New IP: Package a Specified Directory

2. Make your selections from the following options:
  - **Directory:** The location of the IP.
  - **Package as a library core:** The checkbox defines the IP as a library core, which is IP that is available in the IP repository, but is not visible in the IP Catalog.



**IMPORTANT:** Use the Packaging as a library core for IP that is not to be used as a standalone IP. This option lets you reference the IP from the IP Repository, but does not make the IP visible in the IP Catalog.

3. Click **Next**.

4. In the **Edit in IP Packager Project Name** dialog box, as shown in [Figure 3-8](#), set the following information:
  - **Project name:** Name of the project created with the generated IP definition.
  - **Project location:** Directory where the design sources exist and the Edit IP Packager project is located.




---

**RECOMMENDED:** *Keep file path lengths under 80 characters.*

---

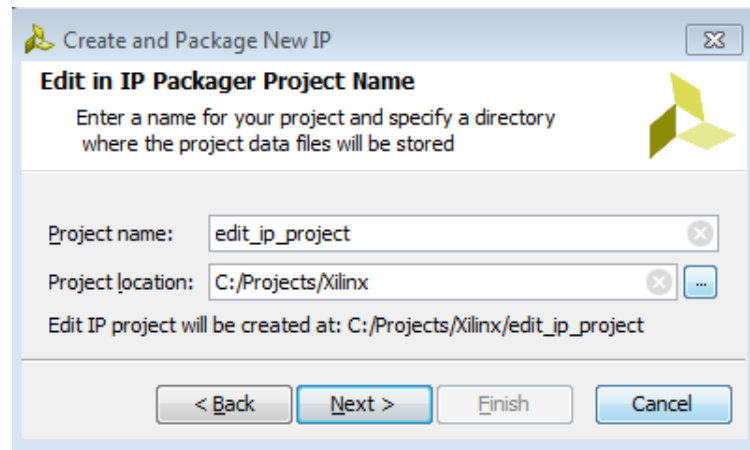


Figure 3-8: Create and Package New IP: Edit in IP Packager Project Name

5. Click **Next**.

The New IP Creation page summarizes the information that the Create and Package IP wizard heuristically gathers about the design.

6. Click **Finish** to complete packaging, and open the Edit IP Packager project. See [Chapter 4, Packaging IP](#), for more information.

---

## Creating a New AXI4 Peripheral

To create a new AXI4 peripheral:

1. From the Choose Create Peripheral or Package IP dialog box, select **Create a new AXI4 peripheral**, and click **Next**.
2. Enter the IP peripheral details:
  - **Name:** The name of the IP.
  - **Version:** IP version that reflects the <major#.minor#.Rev#> version scheme.
  - **Display Name:** The name of the IP that shows in the IP Catalog.

You can have different names in the **Name** and **Display Name** fields; however, the **Display Name** must be intuitive enough that any change in **Name** can reflect automatically in the **Display Name**.

- **Description:** The IP description to share with an end-user of the IP.
  - **IP Location:** IP packager automatically adds the IP repository location.
3. Click **Next**.
  4. Add interfaces to your IP, based on the functionality and the required AXI type, shown in the following figure.

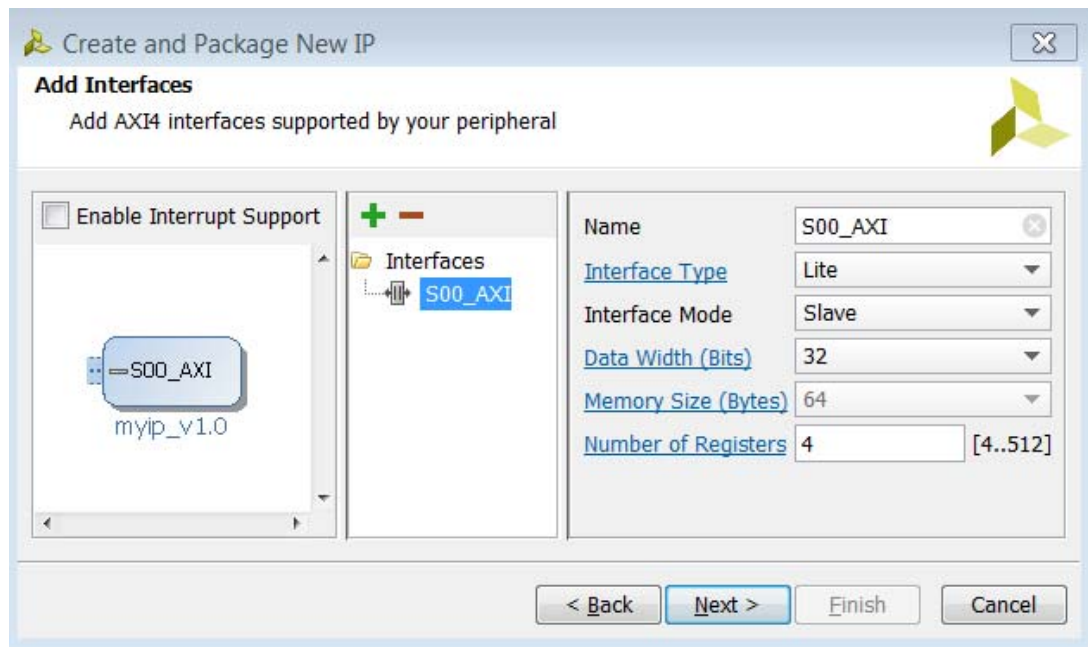




Figure 3-9: Create and Package New IP: Add Interfaces Dialog Box

5. To include interrupts to be available in your IP, check the **Enable Interrupt Support** option.

Figure 3-9 shows that generated IP supports edge or level interrupt (generated locally on the counter) and those interrupts can be extended to input ports by user and IRQ output.

- Add an interface using the **Add** button .
- Delete an interface using the **Remove** button .

The data width and the number of registers vary, based upon the AXI4 selection type.

6. Click **Next**.
7. Review your selections.

The details of your IP are listed in the final wizard page, as shown in the following figure.



Figure 3-10: Create and Package New IP: Peripheral Summary Dialog Box

The following options are available after you generate the IP:

- **Add IP to the repository:** Lets you add IP to the IP repository.
- **Edit IP:** Lets you edit the IP.
- **Verify peripheral IP using AXI4 BFM Simulation Interface:** Lets you use an AXI4 BFM simulation interface (licensing is required to use AXI4 BFM simulation).
- **Verify peripheral IP using JTAG interface:** Creates a block design with which you can debug your IP module in hardware for a system with JTAG-to-AXI IP. See the *Vivado Design Suite User Guide: Programming and Debugging (UG908)* [Ref 20] for more information about the Vivado debug tools.

You can generate a bitstream and then validate the register writes and reads (from the sample Tcl script generated by the tool for your design) in the debug mode after the targeted device is programmed. You can do so by connecting to the board server from hardware manager, programming the board, and then sourcing the Tcl script. See the *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)* [Ref 15] for more information.

After you create the peripheral, you have the option to add custom logic and make the peripheral a custom IP. See the *Vivado Design Suite User Guide: Creating and Packaging Custom IP (UG1118)* [Ref 21] for a demonstration. When this step completes, review the packaging steps shown in the Package IP view in [Chapter 4, Packaging IP](#).

---

## Using XPMs

Xilinx Parameterized Modules (XPMs) are simple, lightweight, inline customizable solutions for common HDL flow use cases such as RAM/ROM, clock domain crossings, and FIFOs. XPMs are currently enabled on a project by project basis using a property on the project. If the RTL you are packaging includes references to XPMs you must set the `XPM_LIBRARIES` property appropriately.

In this example, all XPM types are enabled:

```
set_property XPM_LIBRARIES {XPM_CDC XPM_MEMORY XPM_FIFO} [current_project]
```

For details on how to add XPMs to your RTL, see *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [\[Ref 9\]](#).

For details on the various XPMs and their parameterization options, see the *UltraScale Architecture Libraries Guide* (UG974) [\[Ref 24\]](#).

# Packaging IP

---

## Introduction

This chapter describes the features for adjusting and packaging a custom IP. The Vivado® IP packager is the interface between the Vivado integrated design environment (IDE) and the IP-XACT™ component file.

In the Vivado IP packager, the following packaging steps are available to customize the custom IP definition:

- [Identification](#): The identification information for custom IP.
- [Using the Compatibility Page](#): The device support for custom IP.
- [Using File Groups](#): The location and behavior of the files for custom IP.
- [Ports and Interfaces](#): The list of top-level ports or interfaces of custom IP.
- [Addressing and Memory](#): The address space and memory maps required for custom IP.
- [Customization Parameters](#): The GUI customization layout of the custom IP.
- [Review and Package](#): A summary and packaging of the custom IP.

After packaging the custom IP definition, you can use the IP within your current project or point the custom IP definition repository path to use in another Vivado project.

---

## Identification

The Identification page, ([Figure 4-1](#)), is the first section of the IP packager. The information is initially populated based on the information entered in the **Project Settings > IP**, described in [Using the Packager Settings](#). The Vivado IP packager heuristically determines the remainder of the information during packaging.

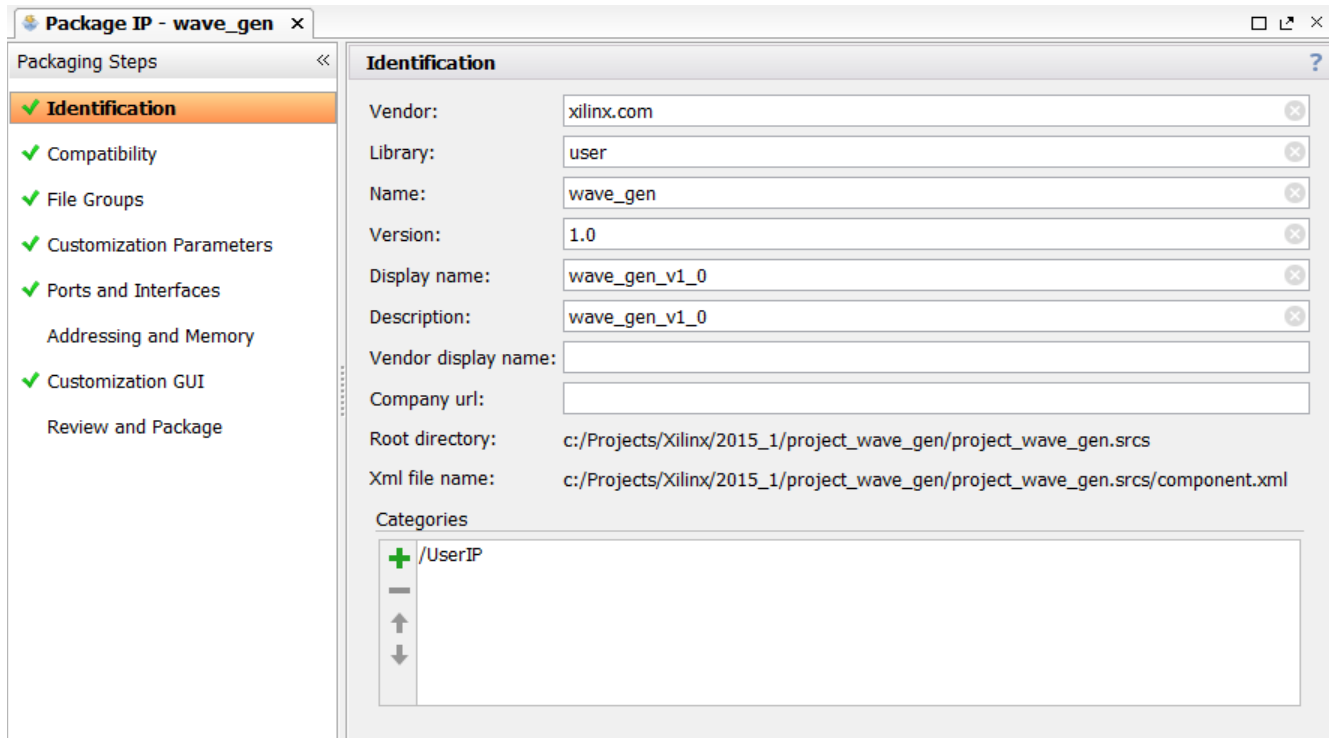


Figure 4-1: Package IP: Identification Page

The **Vendor**, **Library**, **Name**, and **Version (VLNV)** of the IP definition uniquely identifies the IP in the Vivado IP Catalog. The following fields are available to describe the identification of package IP:

- **Vendor:** The vendor of the IP. This is also the identifier for the vendor that displays in the **VLNV** of the IP definition.
- **Library:** The library in which the IP belongs. This is also the identifier for the library that displays in the **VLNV** of the IP definition. See [Versioning your Custom IP](#).
- **Name:** The name of the IP. This is also the identifier for the name that displays in the **VLNV** of the IP definition.
- **Version:** The version of the IP. This is also the identifier for the version that displays in the **VLNV** of the IP definition. See [Versioning your Custom IP](#).
- **Display name:** The Vivado IP Catalog display name.
- **Description:** The Vivado IP Catalog description.
- **Vendor display name:** The Vivado IP Catalog Vendor display name.
- **Company url:** The Vivado IP Catalog display of the company URL.
- **Root Directory:** The working directory of the packaged IP. The directory controls both the location of the input and the output files.
- **XML File Name:** The name and location of the IP-XACT standard XML file.

- **Categories:** The list of category names in which the IP belongs.



**IMPORTANT:** Only one VLVN can exist within the IP Repository. The IP names need to be concise with words separated by underscores.

Each IP within the IP Catalog have taxonomy for organization purposes, as described by the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]. These classifications are controlled by the categories set during IP packaging.

In the Categories list, each category is separated by the forward slash (/) character. Initially, Vivado defaults the custom IP to the **UserIP** category.

## Adding Categories to Your IP

To add or remove categories for your IP:

1. Click the **Add** button  in the **Categories** line of the Identification section to open the IP Categories dialog box, as shown in Figure 4-2.

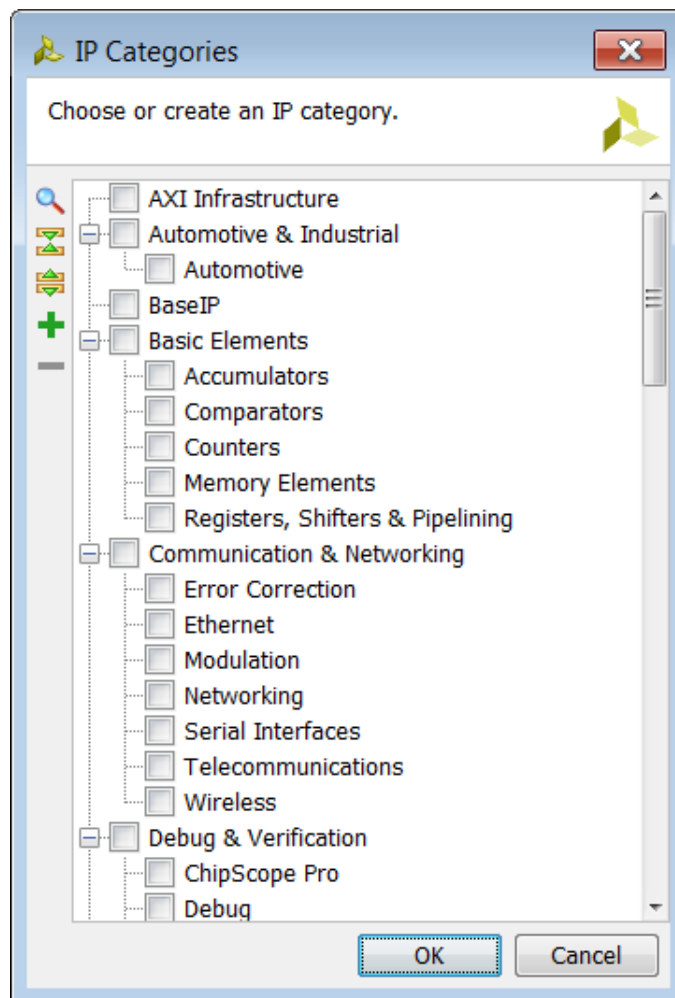



Figure 4-2: IP Categories Dialog Box

2. From the list, select the categories in which to display your custom IP.

## Creating Custom IP Categories

In addition to the predefined categories listed by Xilinx, you can add your own custom IP categories to organize and differentiate your custom IP.


To add a custom IP category:

1. Click the **Add** button  to open the Add IP Category dialog box.
2. Enter the **Name** of the Category you wish to add.  
Any subsequent levels described in the name should be separated by a '/'.  
3. Click **OK**.

The new custom IP category is selected automatically for your IP in the IP Categories dialog box.

## Removing Categories from your IP

If an IP has multiple categories selected, you can remove the previously associated categories. However, an IP is required to have at least one category associated at all times.

To remove categories from your IP, you can select the category in the Categories section of the Identification page and click the **Remove** button  .

You can additionally remove categories from the Add IP Category dialog box by de-selecting the categories in the list.

**Note:** The IP packager removes any custom IP categories that are empty (not containing any IP).

## Using the Compatibility Page

The IP Compatibility page, (shown in the following figure), configures the specific Xilinx parts or device families compatible with your custom IP.

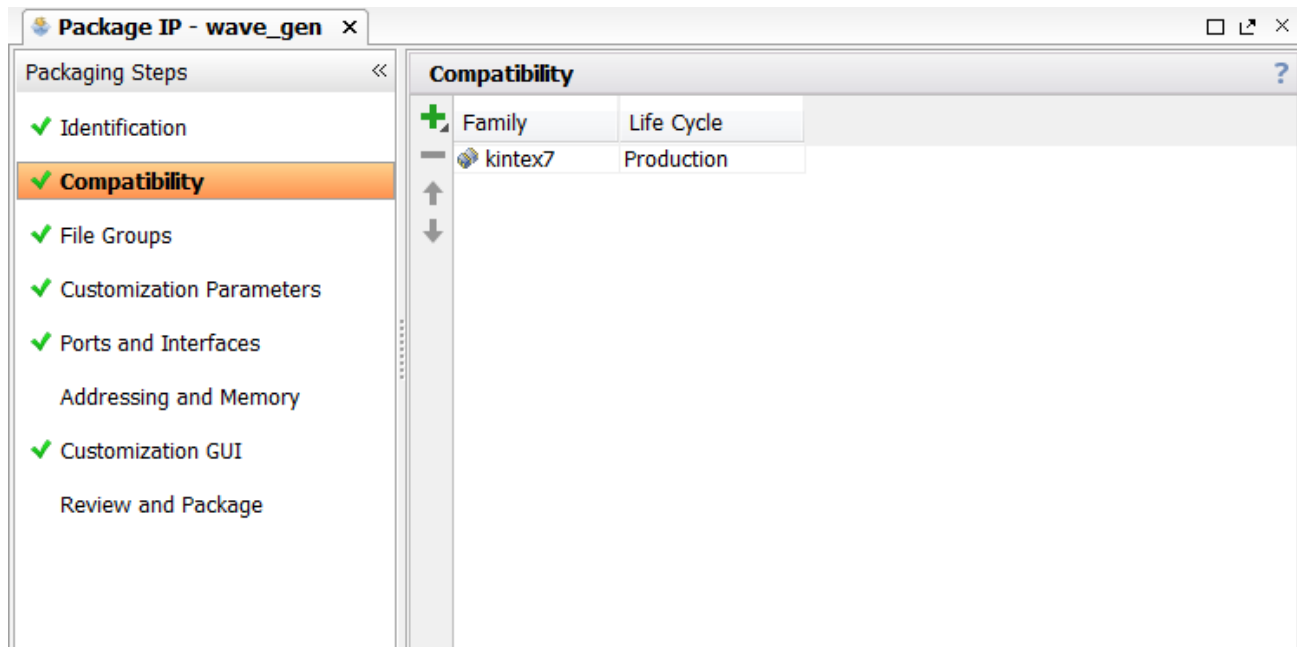


Figure 4-3: Package IP: Compatibility

The list is initially populated with the 7 series Kintex® device family with the with the Life Cycle set to **Production**. Any Xilinx device family and/or devices supported by the Vivado IDE can be included within the family support list of the custom IP. Any device family or part that is not listed in the Compatibility list is incompatible for that IP.

The **Life Cycle** property informs the user of the IP compatibility with the selected IP use case. Within the Compatibility list, each device family or part added can have their own Life Cycle property.

The following choices are available to describe the Life Cycle for a given part or family.

- Beta
- Discontinued
- Hidden
- Pre-Production
- Production
- Removed
- Superseded

Setting the property to **Discontinued**, **Hidden**, **Removed**, or **Superseded** ensures that the IP does not appear in the IP Catalog for the associated device family or part.

## Adding a Xilinx Family or Part

When adding a Xilinx device family or part to set the compatibility of your custom IP, you can add the devices by:

- Adding a Family explicitly
- Adding a Family using a regular expression

Adding a family or part explicitly requires you to select the individual desired devices or parts from a tree. By adding a family using regular expressions, you can select the multiple devices through a specific syntax.


The syntax for the regular expression search is `familyName{regex}`. For example:

```
virtex7{xc7v[hx].*}
```

This regular expression example returns all Virtex®-7 XT and Virtex®-7 HT devices.

### Adding a Family or Part Explicitly

To explicitly add a family or part:

1. Click the **Add** button  in the toolbar of the Compatibility page, and select **Add Family Explicitly** as shown in the following figure.

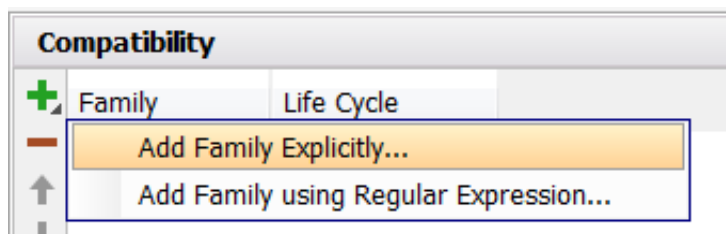


Figure 4-4: Add Family Explicitly

2. In the Add Family dialog box, select the desired device family or parts.
3. Click the **Life Cycle** property from the drop-down menu. The default value is **Beta**.  
**Note:** You can adjust the Life Cycle property later, if necessary.
4. Click **OK**.

## Adding a Family or Part Using Regular Expressions


1. Click the **Add** button  in the toolbar of the Compatibility page, and click **Add Family using Regular Expression**.
2. In the Add Family dialog, shown in the following figure, set the regular expression syntax to add the desired device family and parts.



Figure 4-5: Add Family Dialog Box for Regular Expressions

3. Click the **Life Cycle** property from the drop-down menu. The default value is **Beta**.  
**Note:** You can change the **Life Cycle** property later, if necessary.
4. Click **OK**.

## Editing the Life Cycle Property

1. Click the Life Cycle property for the family, part, or regular expression to select a new Life Cycle property from the drop-down menu as shown in the following figure.

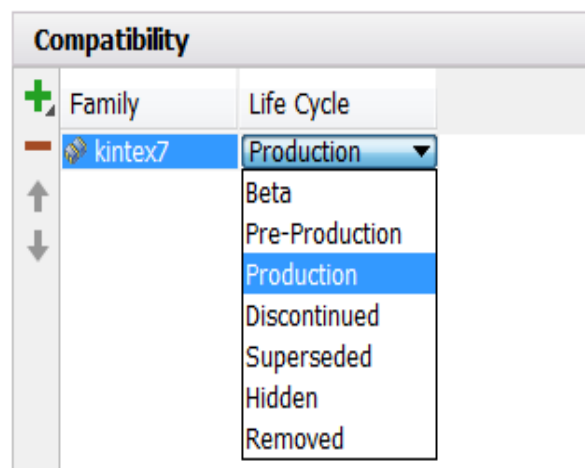


Figure 4-6: Editing the Life Cycle property

## Using File Groups

The File Groups page, shown in the following figure, provides a listing of files of your custom IP.

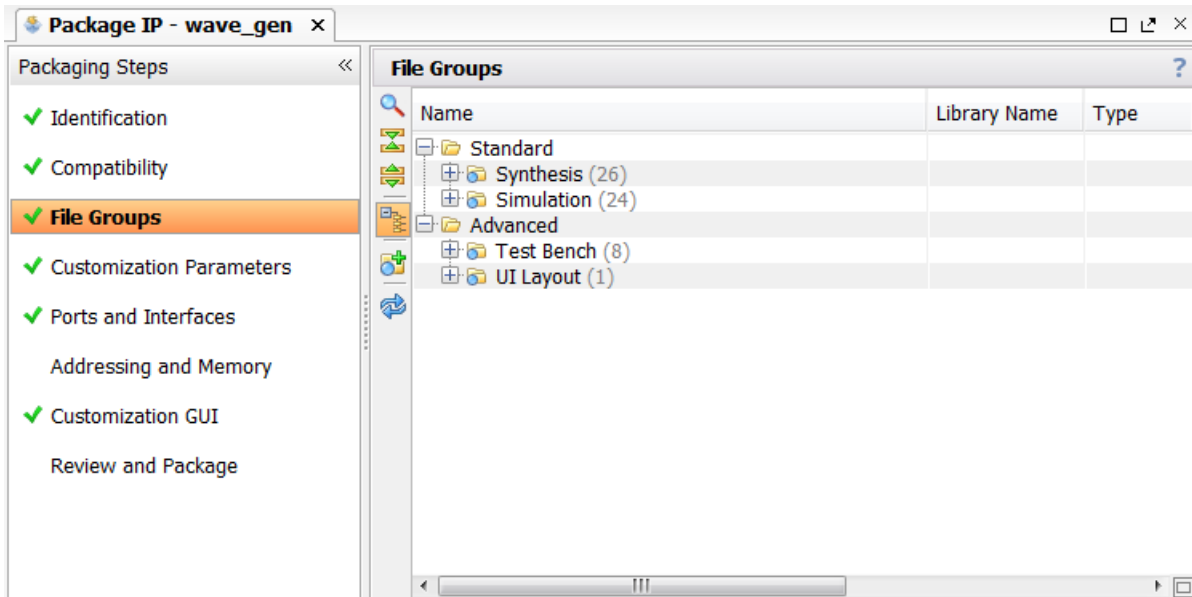


Figure 4-7: Package IP – File Groups

Each file is grouped into specific groups that define the behavior of the file and its use. For example, synthesis uses files located in the Synthesis file group, but simulation uses files in the Simulation file group. If the files in Synthesis and Simulation file groups differ, you could have different behavior.

When you run the Create and Package IP wizard, the files locations in the file groups are initially determined by the wizard either by:

- Using the file sets of the project
- Heuristically determining the information from the directory structure of the IP source files

After you place the files in file groups, you can adjust each file or group based on the custom IP requirements.

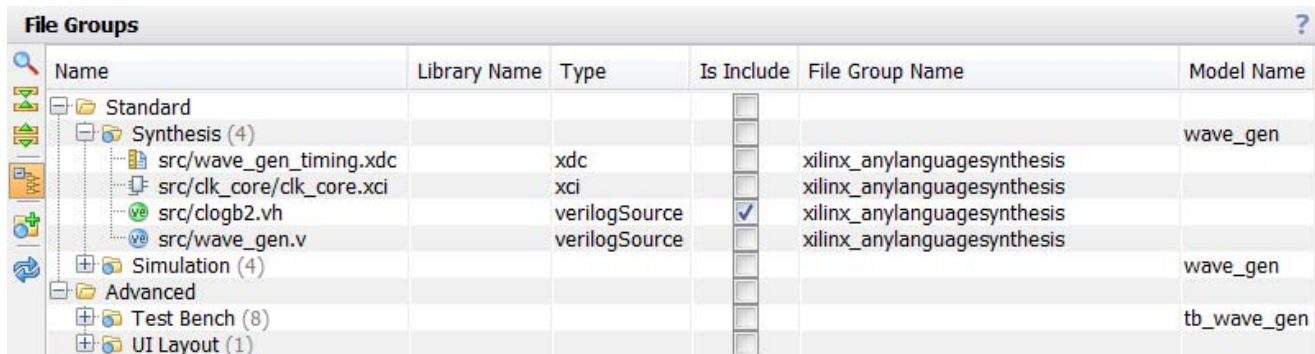
Two sections exist for separating files into different file groups:

- **Standard:** [Standard File Groups](#) describes the standard file groups. For many custom IP, the Standard file group list contains the necessary groups required for packaging an IP for reuse.

- **Advanced:** [Advanced File Groups](#) describes the advanced file groups. The Advanced file group list contains additional advanced features available for your custom IP.

Different functions categorize the file groups. The groups are collapsed at the name of the file group followed by a number in parenthesis, which corresponds to the total number of files in the group. Expand the file group to expose the list of associated files within the group.

The expanded File Groups view shows an example expanded list of files for the synthesis file group as illustrated in the following figure.



Name	Library Name	Type	Is Include	File Group Name	Model Name
Standard			<input type="checkbox"/>		
Synthesis (4)			<input type="checkbox"/>		wave_gen
src/wave_gen_timing.xdc		xdc	<input type="checkbox"/>	xilinx_anylanguagesynthesis	
src/clk_core/clk_core.xci		xci	<input type="checkbox"/>	xilinx_anylanguagesynthesis	
src/clogb2.vh		verilogSource	<input checked="" type="checkbox"/>	xilinx_anylanguagesynthesis	
src/wave_gen.v		verilogSource	<input type="checkbox"/>	xilinx_anylanguagesynthesis	
Simulation (4)			<input type="checkbox"/>		wave_gen
Advanced			<input type="checkbox"/>		
Test Bench (8)			<input type="checkbox"/>		tb_wave_gen
UI Layout (1)			<input type="checkbox"/>		

Figure 4-8: File Group List Expansion

The following properties in the column list are associated with the files of the file group:

- **Name:** File name within the hierarchy tree.
- **Library Name:** Determines the library name used by Vivado synthesis or simulation.
- **Type:** Type of the file in the file group (for example; VerilogSource, XDC, or TclSource).
- **Is Include:** Mark the file as an include file (for example, a Verilog Header file).
- **File Group Name:** The file property to determine to which file group the file is associated. This property is read-only.
- **Model Name:** Top-level design name.



**IMPORTANT:** *The Model Name is a required property and applies directly to the file group. This value is set for any synthesis, simulation, or implementation files.*

## Adding or Removing Files to File Groups

When adding or removing files to and from your custom IP, the recommended flow is to modify the files to the Vivado project and merge the changes using the Package IP view.

For more information on how to add and remove files to your Vivado project, see Vivado *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 9].

With the Package IP view open, a merge changes banner opens that prompts you to update the contents of the file groups with the updated files in the Vivado project. Similar to the Create and Package IP wizard, the merged files is associated with the necessary file groups.

The IP packager merges files from the active sources fileset into the **Synthesis** and **Simulation** file groups. Files that are in the sources fileset, but not used for synthesis are not added to either the Synthesis or Simulation file groups. Files associated with the active simulation fileset are merged only into the **Simulation** file group. A warning is issued for any un-referenced file in the active fileset.



**IMPORTANT:** Files added to the Vivado project should be copied into the project. Files associated with the IP should be contained in a path within the IP directory to ensure proper reuse.

1. Go to the **Flow Navigator > Package IP** to **open the Package IP view**.
2. Add or remove files from your Vivado project associated with the custom IP.
3. In the File Groups page, click the **Merge changes from File Groups Wizard** hyperlink in the banner, as shown in the following figure.

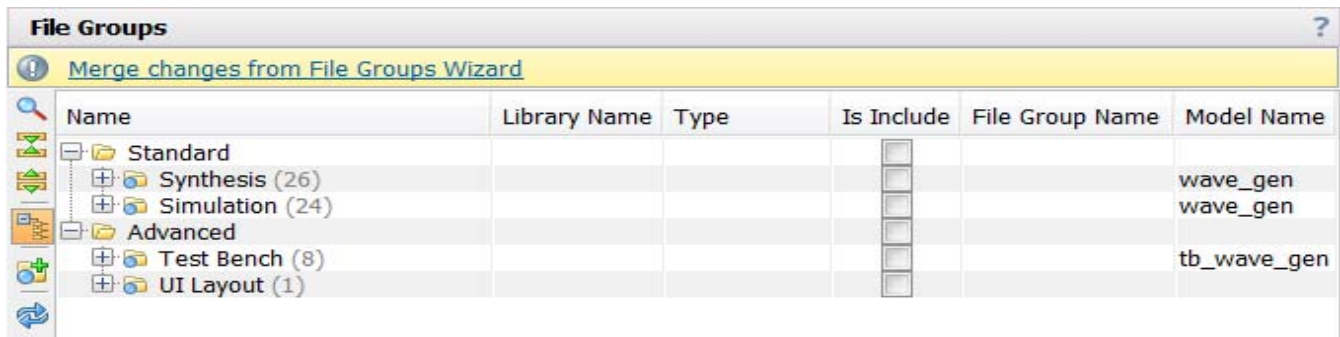


Figure 4-9: Merge Changes from File Groups Wizard Hyperlink

### ***Manually Adding Files to File Groups***

You can manually add files in the Package IP view, but this is not recommended as the wizard ensures that the files are associated to the correct file groups.

To manually add a file to a file group:

1. In the File Group page, right-click on the respective file group and click **Add Files**.
2. In the Add IP Files dialog box, add or create the files for the file group.
3. Click **Finish**.

**Note:** The Add IP Files dialog box only allows files to be added to one file group at a time.

### ***Manually Removing Files from a File Group***

You can manually remove files in the Package IP view, but this is not recommended as Package IP view becomes out-of-sync with the associated Vivado project.

To manually remove a file from a file group, in the File Groups page, right-click the respective file and click **Remove File**.

### ***Copying Files from a File Group***

For files that already exist in the Package IP view, you can copy a file or file group using the **Copy To** option, which extends the list of available file groups. The list contains the file groups listed within the File Groups page as well as additional commonly used groups.

If selecting a file group, the **Copy To** option copies the child files under the file group to the specified destination group.

1. In the File Groups page, select a file or file group, right-click and click **Copy To**.
2. Click the destination file group from the list in the extension menu.

### ***Adding a File Group***

1. In the File Groups page, right-click and select **Add File Group**.

**Note:** Alternatively, you can click the **Add File Group** button  on the toolbar.

2. In the Add File Group view dialog box shown in the following figure, select the file group to add from the list. Selecting the file group shows the description. See [Figure 4-10](#).

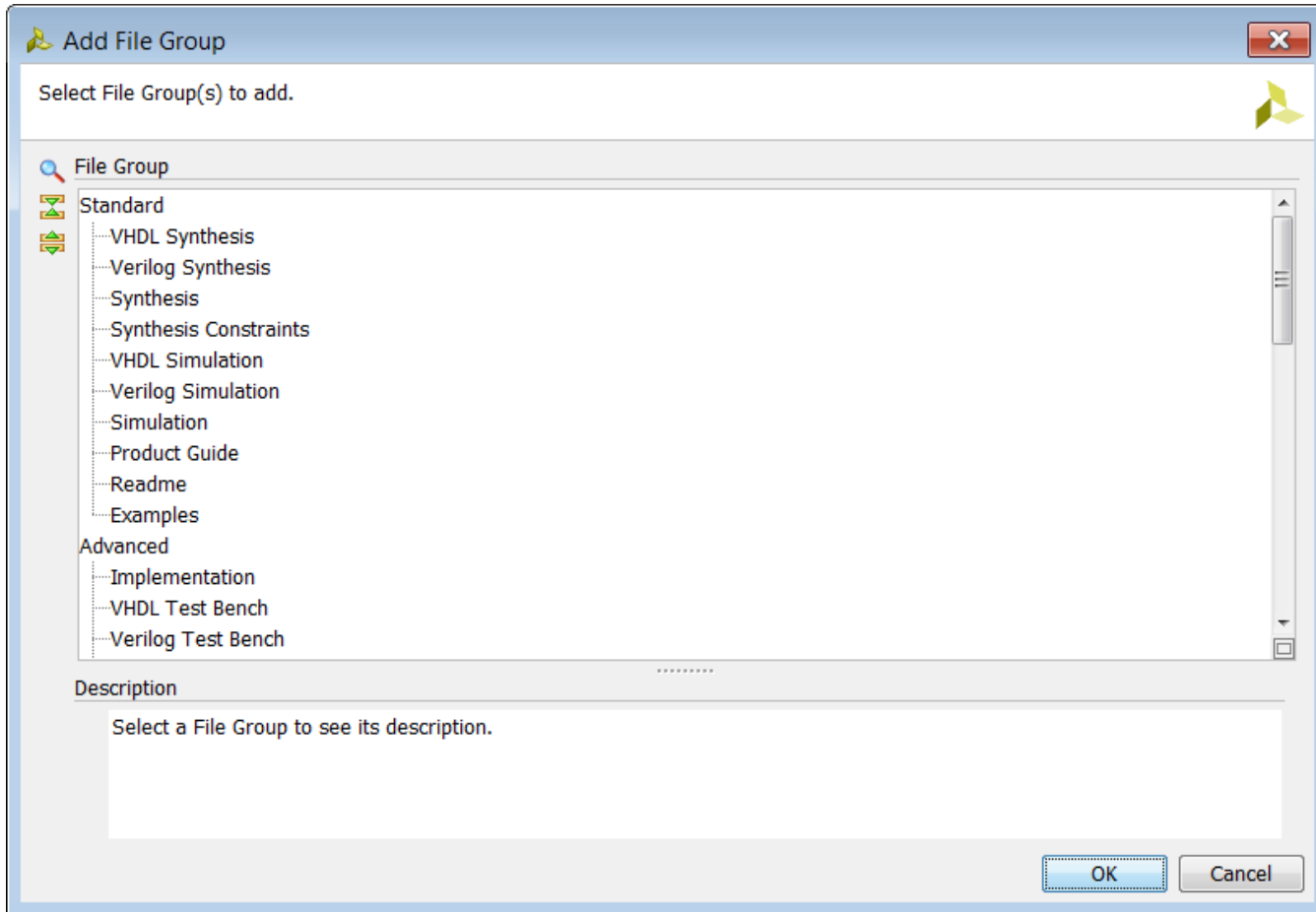


Figure 4-10: Add File Group

3. Click **OK**.

After completing the Create and Package IP wizard, the customization parameters list populates, based on the parsing of the top-level HDL source file.

---

## Customization Parameters

Two folders for parameters display in the Customization Parameters list:

- **Customization Parameters:** Customization Parameters shown in the IP Customization GUI.
- **Hidden Parameters:** Customization Parameters not shown in the IP Customization GUI.

By default, parameters parsed from the wizard are listed in the `/Customization Parameters` folder. These parameters display in the GUI of the custom IP during customization.

Hidden parameters are not intended for direct editing by the user. During customization of the custom IP, these parameters are not available for the user to edit. These parameters are generally dependent upon the parameters shown in the Customization GUI to determine their values.

## Importing Parameters from Top-Level HDL Files

The parameters are initially determined during the Create and Package IP wizard by parsing the top-level HDL file. When you want to update parameters from the top-level HDL source file due to a change to the HDL source after packaging, you can re-import the parameters.

When importing parameters from the HDL, the values are determined at the time of import or merge. If a parameter is calculated based on an expression, the expression is evaluated at the time of import. To keep the expression, you must edit the parameter to create the `XPATH` expression. For more information on editing a parameter, see [Editing a Parameter](#).

To import IP parameters:

1. From the Customization Parameters page, right-click and select **Import IP Parameters**.
2. Click the following options from the Import IP Parameters dialog box.
  - **Top-Level source file:** The top-level source HDL file that contains the top-level entity or module of the custom IP.
  - **Top entity name:** The name of the top-level entity or module that contains the parameters of your custom IP.
  - **Make all imported HDL parameters visible:** Sets all the imported parameters to be visible in the customization GUI.
3. Click **Finish**.

## Adding or Removing a Parameter

You can create or remove parameters for use in your custom IP. When adding parameters, you can only add parameters for use in the Customization GUI. To add a parameter to reference the top-level HDL, the source files must be modified and the parameters must be re-imported.

You can remove any parameter from the list regardless if it is referenced from the top-level HDL. If an HDL parameter is removed from the Customization Parameters list, a default value is required in your top-level HDL source.

### ***Adding a Parameter***

1. In the Customization Parameters page, right-click and select **Add Parameter**.

**Note:** Alternatively, you can click the **Add Parameter** button  on the toolbar.

2. In the Add Parameter dialog box, set the name of the new parameter.

**Note:** This parameter name is not the display name of the parameter in the Customization GUI.

3. Click **OK**.

After creating a new parameter, the Edit IP Parameter dialog box opens for parameter customization.

### ***Removing a Parameter***

Select one or more parameters in the Customization Parameters page, right-click and select **Remove Parameter**.

**Note:** Any parameter dependent on the remove parameter flags as an error.

### ***Editing a Parameter***

You can customize parameters in the Customization Parameters page for how it displays in the IP Customization GUI. You can edit the display name, data format, data range, default value, and visibility.

In the Customization Parameters page, right-click the respective parameter and select **Edit Parameter**.

The Edit IP Parameter dialog box opens with the detailed information set for the selected parameter as shown in [Figure 4-11](#).

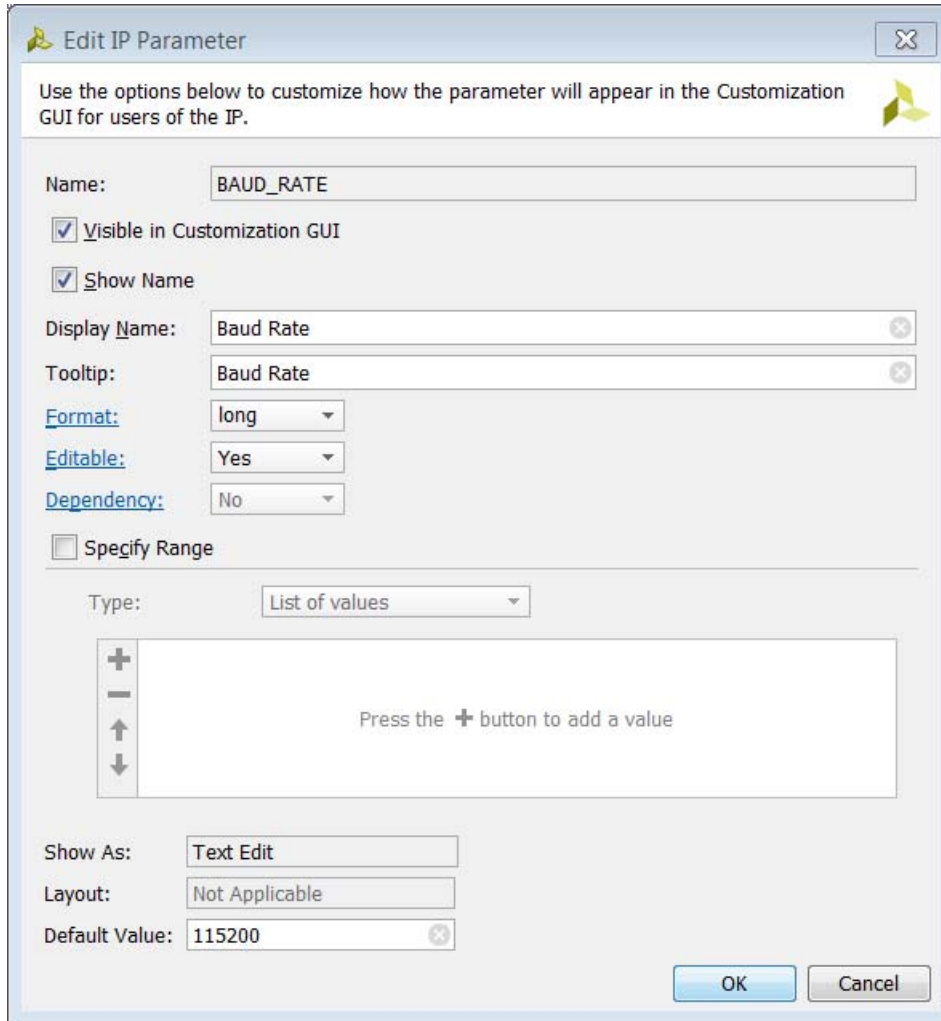


Figure 4-11: Edit IP Parameter

Many options are available to define the use of the parameter for the custom IP. Depending upon the selections within the dialog box, certain options become unavailable.

The following options in the dialog box are associated with the display properties of the parameter.

- **Name:** This is the name of the parameter. This is a read-only property.
- **Visible in Customization GUI:** If checked, the parameter becomes visible in the Customization GUI.
- **Show Name:** If checked, the display name of the parameter lists next to the display object in the Customization GUI. If not, the display object displays without a preceding label.
- **Display Name:** This name displays in the Customization GUI for the parameter.

When the IP is customized, this is the text that displays next to the value that a user can set. This can be a simple name or a small description.

- **Tooltip:** This is the text that displays when you hover over the parameter in the Customization GUI. This can be a simple name or a small description of the parameter.

The following options are associated with the data formatting and restrictions for the parameter.

- **Format:** This defines the data format of the parameter. The data format selection determines the supported values the user can use to customize the IP. Depending upon the selection, the remaining options adjust, based on the requirements for the format.
  - **Long:** Integer input
  - **Float:** Decimal input
  - **Bool:** Boolean input
  - **BitString:** Hex input
  - **String:** String input
- **Editable:** This defines if the parameter is editable by the user, or only under certain conditions.
  - **Yes:** This selection indicates the user can edit the parameter in the Customization GUI. This is the default setting.
  - **No:** This selection indicates the user cannot edit the parameter in the Customization GUI. If displayed, the parameter is read-only during customization. The value of the parameter is determined by the default value or expression.
  - **Dependent:** This selection indicates the user can only edit the parameter in the Customization GUI under certain conditions.

If selected, a new option displays to set the expression as shown in the following figure.

The evaluated editability expression, if **true**, allows the parameter to be editable. For more information on setting an editability expression, see [Setting an Enablement Expression](#).

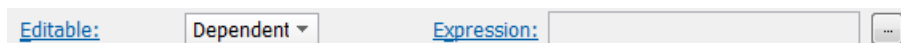


Figure 4-12: Expression Field for Editable Option

- **Dependency:** This defines how the parameter value is determined during customization.
  - **Yes:** This selection indicates the value of the parameter is dependent on another parameter.

If selected, a new option displays to set the expression to evaluate the parameter value, as shown in the following figure. The evaluated dependency expression is used as the parameter value.

For more information on setting a dependency expression, see [Setting an Enablement Expression](#).

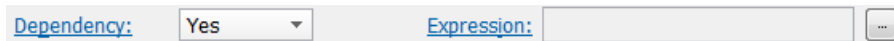


Figure 4-13: Expression Field for Dependency Option

- **No:** This selection indicates the value is not dependent on another parameter, and the value is set directly by the user or by the default value.
- **Specify Range:** If selected, this defines the bounds by which the parameter value is set. If not selected, the value can be any value in the selected data format, and the IP Customization GUI does not ensure that the value is within the expected bounds for usage.
  - **List of values:** This selection limits the value choices for the parameter in a predefined list. The list box defines the valid list of values for the parameter, as shown in the following figure.

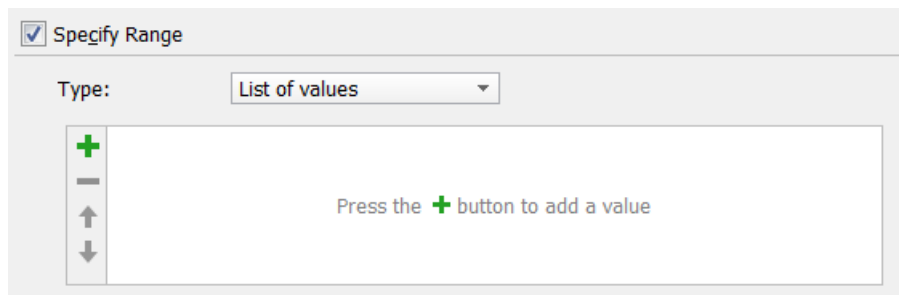






Figure 4-14: List of Values Restriction

- To add a value to the list, click the **Add** button  on the side bar. All created elements default the text to **value**.
- To change the element text, double-click the value in the list to enable modification. The elements in the list display in the IP Customization GUI in the same order they appear in the list.

- To change the order of the elements:
  - Click the element and then select the **Move Up** button  to move the element up by 1.
  - Click the **Move Down** button  to move the element down by 1.
- To remove an element, select one or more elements in the list, and click the **Remove** button .
- **Range of integers:** This selection limits the value of the parameter to a specific range of numeric integer values. The **Minimum** and **Maximum** range restricts the integer values to the user for the parameter, as shown in the following figure.

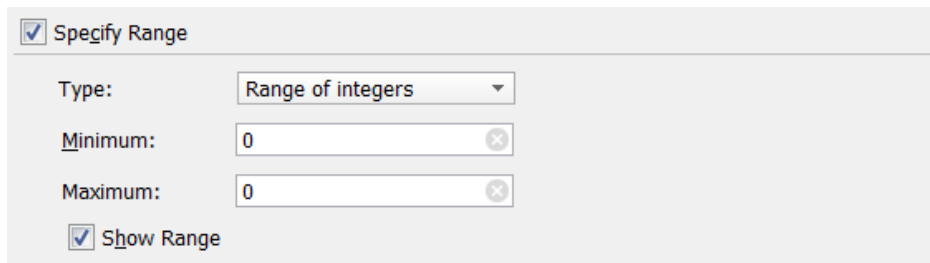


Figure 4-15: Range of Integers Restriction

A value set outside the specified range within the IP Customization GUI reports as an error.

If checked, the **Show Range** option displays the minimum and maximum range of the integer values in the Customization GUI.

- **Pairs:** This selection limits the value choices for the parameter from a predefined list similar to the list of values restriction; however, you can show the selections in the IP Customization GUI in a different format. A Key/Value pair list controls the value of the parameter.

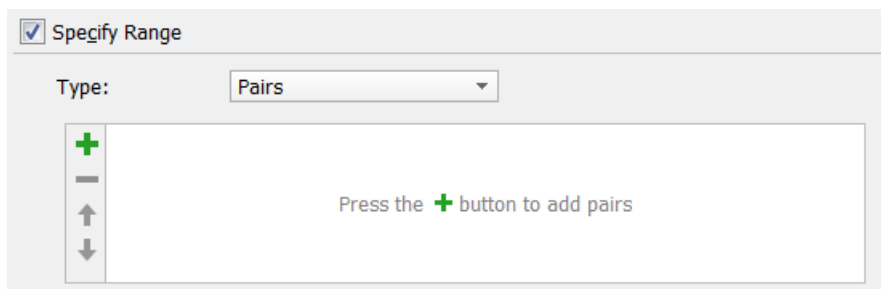






Figure 4-16: Pairs Restriction

- To add a key/value pair to the list, Click the **Add** button  on the side bar. All created elements default the key to **key** and the value to **value**.

- To change the element text, double-click the element in the list to enable modification. The elements in the list display in the IP Customization GUI in the same order they appear in the list.
- To change the order of the elements:
  - Click the element and then select the **Move Up** button  to move the element up by 1.
  - Click the **Move Down** button  to move the element down by 1.
- To remove an element, select one or more elements in the list and click the **Remove** button .

The last set of options describe the display object and the default value of the parameter.

- **Show As:** This describes the display object used for the parameter in the Customization GUI. This is a read-only property that is set depending the options defined within the dialog box.
- **Layout:** This describes the display layout of a Radio Group. The values are **Vertical** and **Horizontal**. In all other fields, **Not Applicable** is displayed.
- **Default Value:** This defines the default value for the parameter. If an IP is customized without any modification to the parameters, the parameter is set by the value defined in this option.

## Ports and Interfaces

The Ports and Interfaces page, shown in the following figure, provides a listing of ports and interfaces of the custom IP.

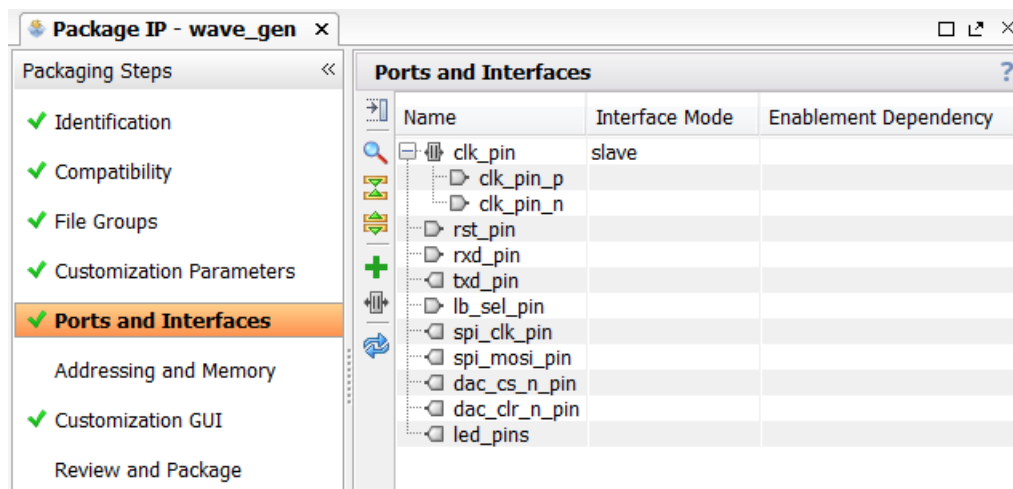


Figure 4-17: Package IP – Ports and Interfaces

After completing the Create and Package IP wizard, the ports and interfaces table populates based on the parsing of the top-level HDL source file. During the parsing of the top-level ports, if an interface can be heuristically determined, the interface is inferred automatically.

The following properties column list are associated with a port or interface:

- **Name:** The port or interface name. The direction of the port is described in the preceding icon.
- **Interface Mode:** The mode of the interface (master or slave).
- **Enablement Dependency:** The expression to determine whether the port or interface is enabled. For more information, see [Setting an Enablement Expression](#).
- **Is Declaration:** Indicates if this interface is a pre-declaration of a dynamically created interface.
- **Direction:** The direction of the port.
- **Driver Value:** The driver value of the port if disabled.
- **Size Left:** The value of the most significant bit (MSB).
- **Size Right:** The value of the least significant bit (LSB).
- **Size Left Dependency:** The dependency expression to determine the value of the most significant bit.
- **Size Right Dependency:** The dependency expression to determine the value of the least significant bit.
- **Type Name:** The port type (`std_logic` or `std_logic_vector`).

## Updating Ports from a Top-level HDL

The recommended flow for adding or removing ports to the Custom IP is to modify the top-level HDL file in the Vivado project and merge the changes using the Package IP view. With the Package IP view open, a merge changes banner appears that prompts you to update the contents of the ports and interfaces list from the updated file in the Vivado project.

1. Go to the **Flow Navigator > Package IP to open the Package IP view**.
2. In the Ports and Interfaces page of the Package IP view, Click the **Merge changes from Ports and Interfaces wizard** hyperlink in the banner.

The IP packager only monitors the top-level HDL file for file changes. It does not monitor secondary or ancillary files that, if changed, could affect the top-level ports.

For cases where ancillary files are modified and the merge changes banner is not present, the same behavior can be accessed through the Tcl design environment:

- **Tcl Command:** Merge changes from Ports and Interfaces wizard:

```
ipx::merge_project_changes ports [ipx::current_core]
```

## Importing Ports from a Top-Level HDL

The ports and interfaces are determined during the Create and Package IP wizard by parsing the top-level HDL file. When you want to update ports due to a change to the HDL source after packaging, you can re-import the ports.




---

**IMPORTANT:** *When you re-import the top-level source file, all pre-existing HDL parameters and associated data are removed.*

---

1. From the Ports and Interfaces page, right-click and select **Import IP Ports**.
2. In the Import Ports from HDL dialog box, select the following options:
  - **Top-Level source file:** The top-level source HDL file that contains the top-level entity or module of the custom IP.
  - **Top entity name:** The name of the top-level entity or module that contains the ports of your custom IP.
3. Click **Finish**.

## Editing a Port

You can modify the behavior for ports on your custom IP by defining restrictions on its use, as follows:

1. Click a port in the Ports and Interfaces view, right-click and select **Edit Port**.
2. In the Edit Port dialog box, shown in [Figure 4-18](#), set the following options:
  - **Driver Value:** If the port is disabled during customization, this is the driver value to the port.
  - **Port Presence:** This option determines if the port can be disabled.
    - The **Mandatory** option defines that the port are always available for the custom IP.
    - The **Optional** option defines that the port is disabled when the expression in the text box is **false**. For more information on setting an enablement expression, see [Setting an Enablement Expression](#).

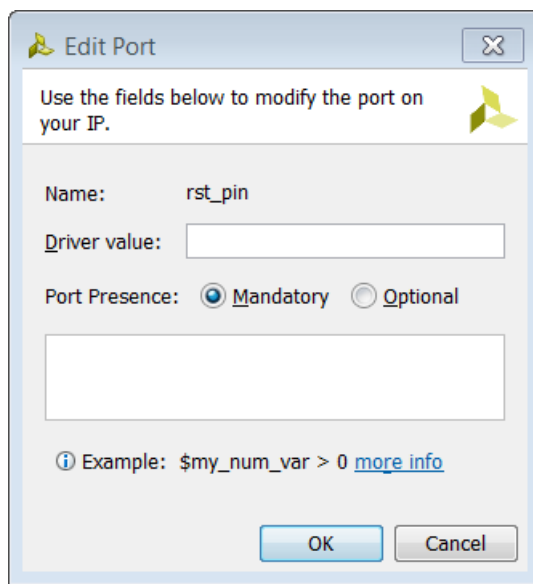


Figure 4-18: Edit Port Dialog Box

## Removing a Port

You can remove a port from the Ports and Interfaces page, but this is not recommended. Removing the port deletes the object reference to the port in the IP, but the port still exists in the HDL. The end result would be an undriven or unconnected port when the IP is customized. To remove a port, select one or more ports in the Ports and Interfaces page, right-click and click **Remove Port**.

## Adding or Removing Bus Interfaces

Interfaces provide the ability to group signals into a common grouping to use between IP in a Vivado IP integrator design. You can add an existing interface or create a new interface for a group of ports in your custom IP to simplify connectivity within IP integrator.

### *Adding an Existing Bus Interface*

In the Ports and Interfaces page, right-click and Click **Add Bus Interface**.

**Note:** Alternatively, you can click the **Add Bus Interface** button on the sidebar.

The Add Interface dialog box opens for you to modify the properties related to the interface. For more information on editing an interface, see [Editing an Existing Interface](#).

### *Creating a New Bus Interface*

1. Click the ports in the Ports and Interfaces page, right-click and select **Create Interface Definition**.

2. In the Create Interface Definition dialog box, shown in the following figure, select the following options:
  - **Vendor:** The vendor of the Interface Definition.
  - **Library:** The library in which the Interface Definition belongs.
  - **Name:** The name of the Interface Definition.
  - **Version:** The version of the Interface Definition.
  - **Mode:** The mode of the interface instance (Master or Slave).
  - **Location:** The repository location in which to store the Interface Definition.

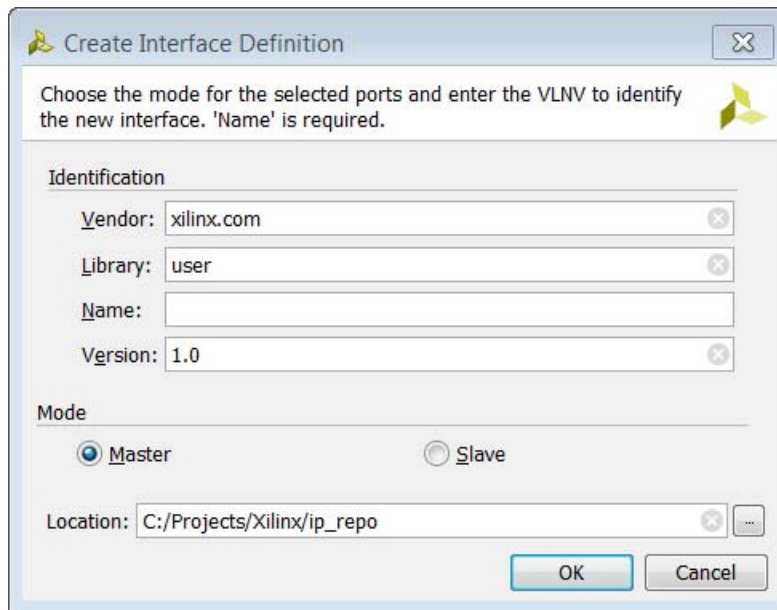


Figure 4-19: Create Interface Definition

3. Click **OK**.

For more information on creating an interface definition, see [Chapter 5, Creating New Interface Definitions](#).

### **Removing a Bus Interface**

To remove an interface or several interfaces, select one or more interfaces in the Ports and Interfaces page, right-click and click **Remove Interface**. After you remove the interface, the ports previously associated return to the Ports and Interface table as unassociated ports.

## Auto Inferring an Interface

There are two options for automatically inferring an interface: bus or single-bit.

- Infer the bus interface by selecting from a full interface list
- Infer a single-bit interface from a list of single-bit signal interfaces

The port names must match exactly as specified in the interface so the auto-inference heuristics can properly match the port list to the correct interface signals.

## Auto Inferring a Bus Interface

1. Click all the ports related to the interface, right-click and select **Auto Infer Interface**.  
*Note:* Alternatively, you can Click the **Auto Infer Interface** button on the sidebar.
2. In the Auto Infer Interface Chooser dialog box, shown in the following figure, select the interface to infer.

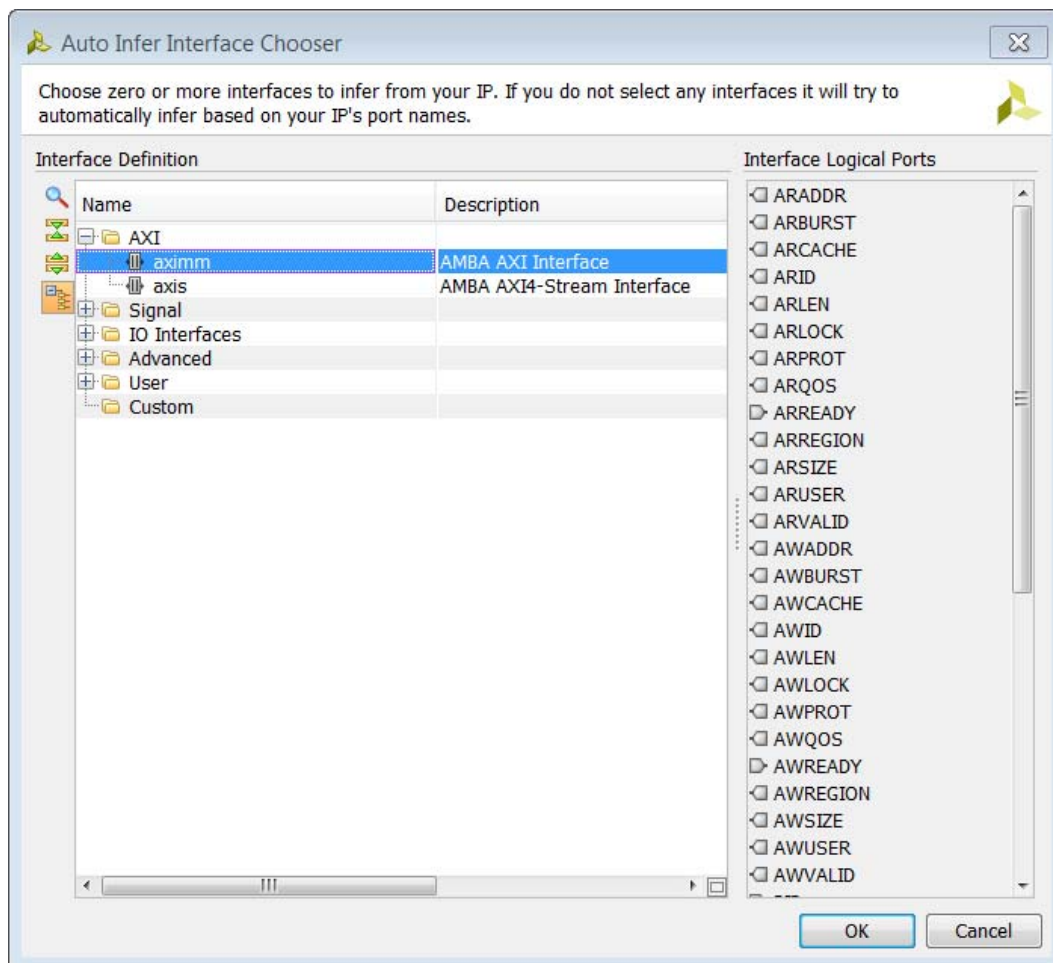


Figure 4-20: Auto Infer Interface Chooser

3. Click **OK**.



**TIP:** If the auto inference heuristics are unable to infer the interface, you must manually identify the bus interface ports through the **Add Bus Interface** option. See [Adding or Removing Bus Interfaces](#).

## Auto Inferring a Single Bit Interface

1. Click the port, right-click and select **Auto Infer Single Bit Interface**.
2. From the extended menu, select the interface you want to infer.

## Editing an Existing Interface

1. Click an interface in the Ports and Interfaces page, right-click and select **Edit Interface**.
2. In the Edit Interface dialog box, select the desired options.
3. Click **OK**.

## Editing the Interface Information

The General tab of the Edit Interface dialog box, shown in the following figure, displays the instance information for the interface. The tab defines the **Name**, **Mode**, **Display Name**, and **Description** information.

1. Click the General tab.

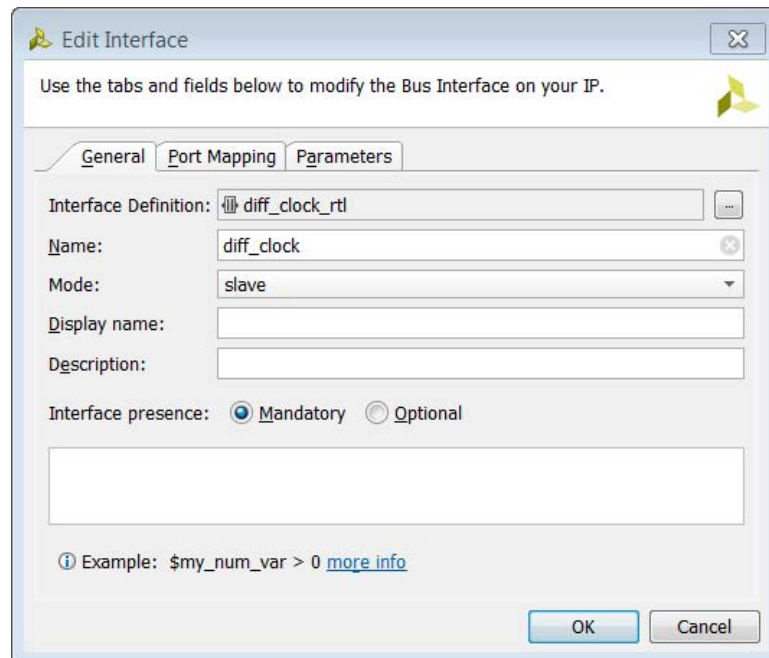


Figure 4-21: Edit Interface - General Tab

2. Define the following options:

- **Interface Definition:** The name of the interface definition.
- **Name:** The name of the interface instance. This is the name that displays in the Customization GUI and the IP integrator canvas.
- **Mode:** The mode of the interface instance (Master or Slave).
- **Display name:** The display name of the interface.
- **Description:** The description of the interface.
- **Interface presence:** This option determines if the interface can be disabled.
  - The **Mandatory** option defines that the interface is always available for the custom IP.
  - The **Optional** option defines that the interface is disabled when the expression in the text box is `false`.

For more information on setting an enablement expression, see [Setting an Enablement Expression](#).

3. Click an additional tab, or click **OK**.

## Mapping the Interface Ports

The Port Mapping tab, see [Figure 4-22](#), displays the mapping of the interface ports to the ports of the custom IP. The ports are listed in two columns representing the logical ports of the interface and the physical ports of the custom IP. Ports mapped to the interface display in the summary table at the bottom of the view.

1. Click the interface port in the left column, and select the associated physical port from the right column.
2. Click the **Map Ports** button.

For large interfaces, the **List Options** modifies the port displayed in the columns to simplify connectivity. The options are:

- **Filter Incompatible Physical Ports:** Hide physical ports that do not match the correct direction of the selected interface port.
- **Hide Mapped Ports:** Hide physical and logical ports that have been previously mapped.

The mapped ports are listed in the **Mapped Port Summary** section of the view. You can un-map all the previously mapped ports, or selectively un-map ports in the summary view.

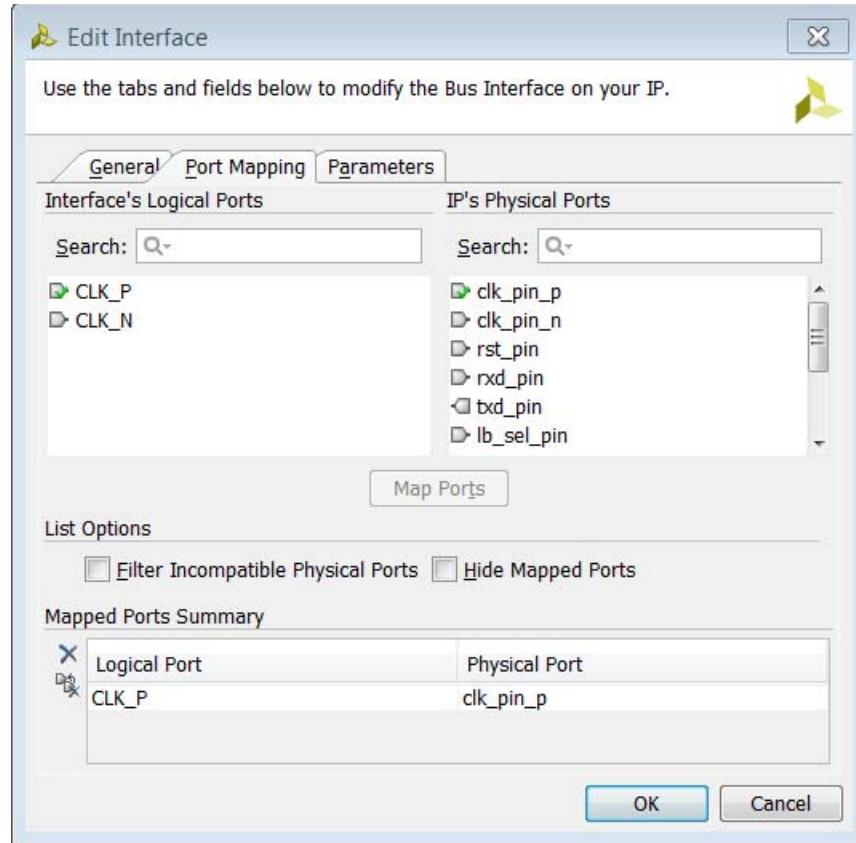


Figure 4-22: Edit Interface – Port Mapping

3. Click an additional tab or click **OK**.

## Adding and Removing Interface Parameters

The Parameters tab, shown in the following figure, displays the parameters defined for the interface. Some bus interfaces require associated parameters. The Vivado IDE identifies some parameters automatically, and recommends those parameters for the interface.

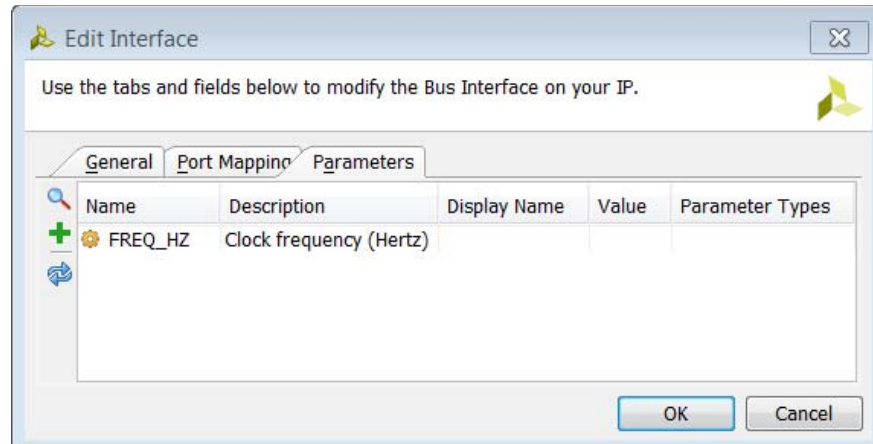


Figure 4-23: Edit Interface – Parameters

After adding parameters to the table, the following options exist for each parameter:

- **Name:** The name of the parameter.
- **Description:** The parameter description.
- **Display Name:** The parameter display name.
- **Value:** The parameter value.
- **Parameter Types:** The type of parameter.

### Adding Recommended Parameters

If the interface has predefined parameters, you can add the recommended parameters by doing the following:

1. In the Parameters tab, right-click and select **Add Recommended Parameter**.
2. In the Add IP Parameter dialog box, select the recommended parameter.
3. Click **OK**.



**TIP:** If the Add Recommended Parameter is not available, no predefined parameters are available for that interface definition.

### ***Adding Custom Parameters***

If you want to define your own parameters for the interface, you can add your own parameter using the following steps:

1. In the Parameters tab, right-click and select **Add Bus Parameter**.
2. In the Add Parameter dialog box, select the name of the parameter.
3. Click **OK**.

### ***Removing Parameters***

To remove parameters, select one or more parameters in the table, right-click **Remove Bus Parameter**.

## **Addressing and Memory**

The Addressing and Memory page, shown in the following figure, lets you add a memory-map or address space to the IP.

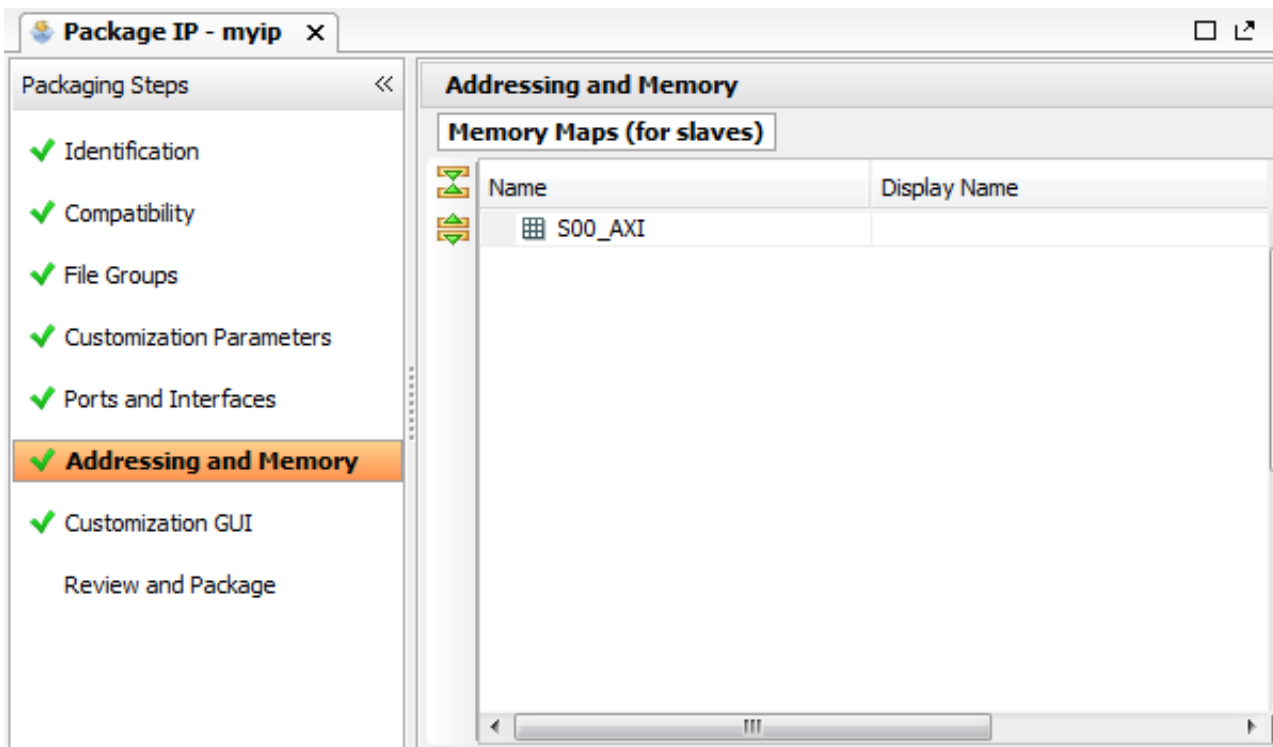


Figure 4-24: IP Addressing and Memory

If the Create and Package IP wizard was able to automatically infer an interface for your custom IP, the Addressing and Memory step is populated also if an address mapping requirement for the interface was identified.

## Identifying an Interface for Address Mapping

1. If an interface has not been previously mapped, click the Addressing and Memory Map Wizard link.

**Note:** Alternatively, right-click and select **IP Addressing and Memory Map Wizard** from the pop-up menu if an interface has already been mapped.

2. In the Addressing and Memory Wizard, click **Next**.
3. In the Choose IP Interface page, shown in [Figure 4-25](#), select the interface to add a memory map and click **Next**.

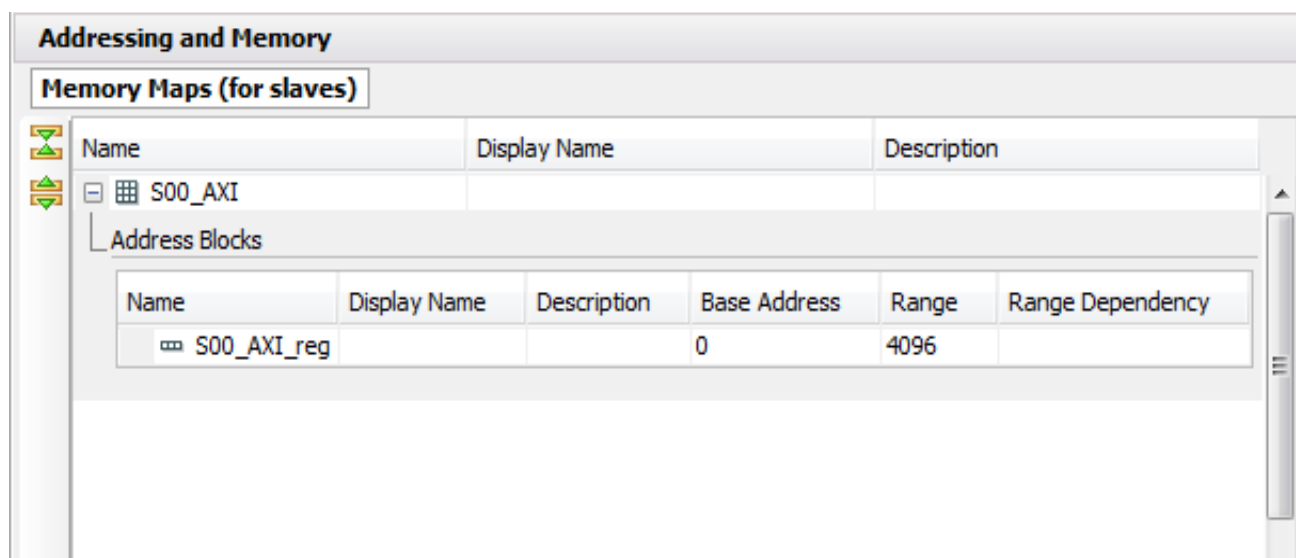


Figure 4-25: IP Addressing and Memory Configuration Wizard - Choose IP Interface

4. In the Choose IP Object Name page, select the name for the memory map, and click **Next**.
5. On the Summary page, click **Finish** to complete the wizard.

## Adding an Address Block

1. Select a Memory Map from the list, then right-click and click **Add Address Block**.
2. In the Add Address Block dialog box, select the name of the new Address Block, and click **OK**.

The selected memory map in the Addressing and Memory view now contains a child address block section with the newly-created address block. The address block list contains the following columns:

- **Name:** Address block name.
- **Display Name:** Address block display name.
- **Description:** Detailed description of the address block.
- **Base Address:** Base address of the address block.
- **Range:** Range of the address block.
- **Range Dependency:** The dependency expression for the address block range.

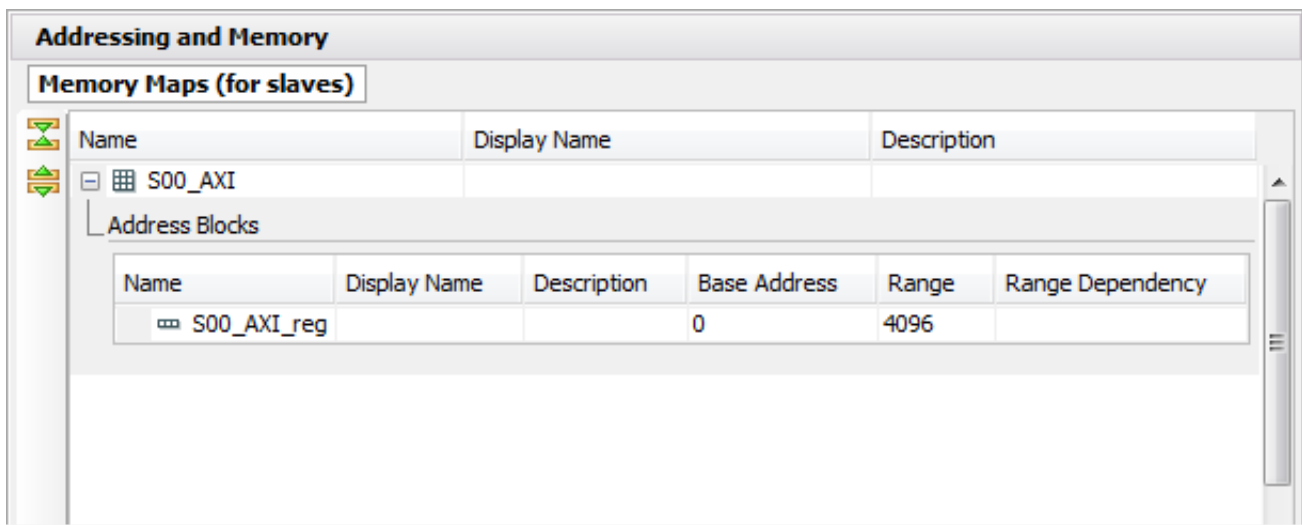


Figure 4-26: Memory Map with Address Block

3. Select the cell of the address block to enter the desired information.

## Add or Remove an Address Block Parameter

To add an address block parameter:

1. Click the address block in the list, right-click and select **Add Address Block Parameter**.
2. In the Add Address Block Parameter dialog box, select the name and click **OK**.
3. Select the cell of the address block parameter to enter the information.



**RECOMMENDED:** Give the interface in a meaningful name that reflects the functionality.

To remove an address block parameter from an address block, select the address block parameter in the list, and right-click and select **Remove Address Block Parameter**.

The IP Addressing and Memory Wizard Summary page opens, which describes the name of the new memory-map as well as to which interface the memory-map references.

4. Review the summary, and click **Finish**.

## Customization Parameters

The Customization page, shown in the following figure, provides an environment for the GUI customization of your custom IP. The Customization GUI section lets you customize the layout by adding display pages, parameter groupings, and text fields.

After you setup the parameters of your IP in the Customization Parameters section, you can customize the GUI to change how a user interacts with your custom IP.

Initially, the Customization GUI generates a layout with all the viewable parameters displayed on a single page of the GUI.

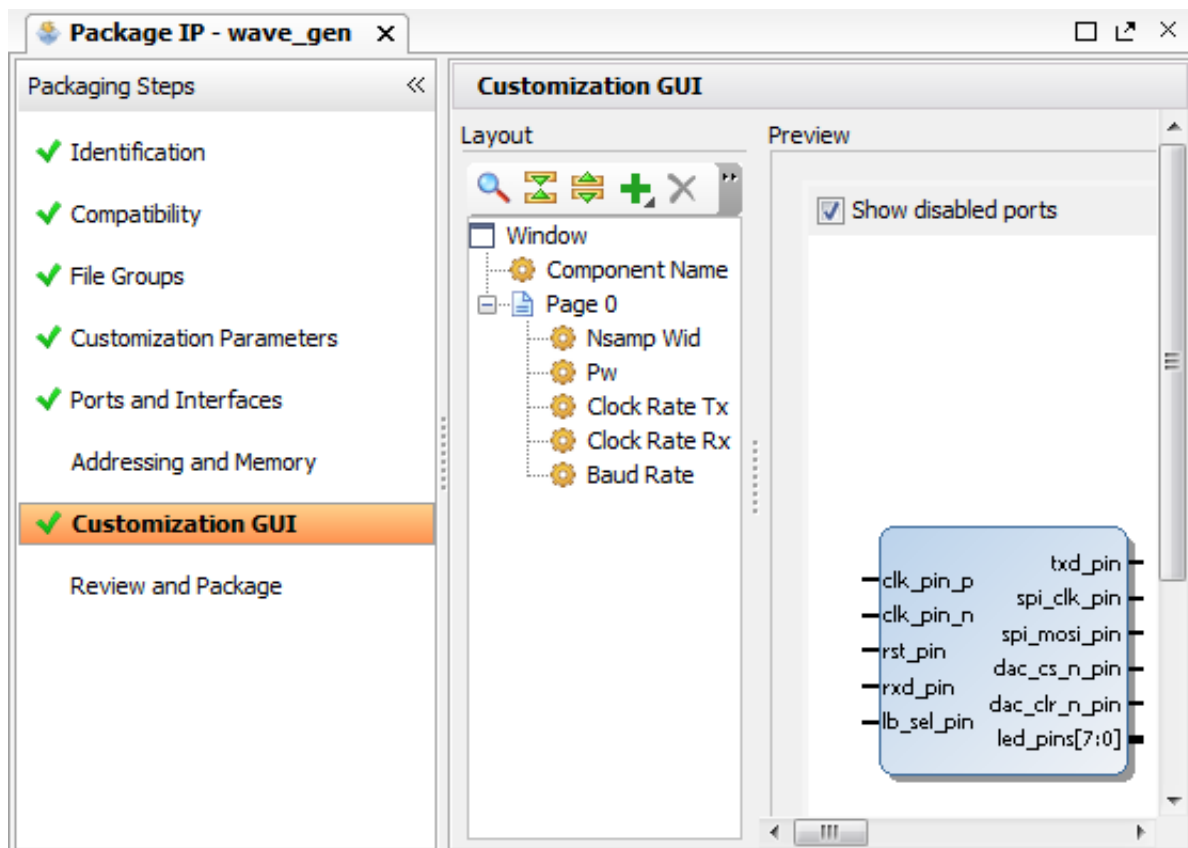


Figure 4-27: Package IP – Customization GUI

The two columns that display in the Customization GUI view are:

- **Layout:** A hierarchical display of the layout which allows for modification of the custom IP GUI.
- **Preview:** A preview display of the Customization GUI of the custom IP.

The **Layout** column displays a hierarchical view of the customization GUI components. Within a **Window** component, which is the top-level to which to associate the customization components, there are a total of four components that you can create to customize the display of your custom IP GUI:

- **Page:** An individual page to display the parameters of the custom IP.
- **Group:** A collection of parameters to display in a single group.
- **Parameter:** A parameter of the custom IP.
- **Text:** A text field to display any necessary information in the GUI.

Each component can be associated hierarchically to other components within the customization layout. The **Group**, **Parameter**, and **Text** components can be within a **Page** or **Group**. The **Page** component can only be a child of the **Window** component.

The preview view shows you a real-time feedback view of the customization GUI as it would appear if the IP was customized through the IP Catalog.


The components in the preview display in the same order in which they are arranged in the layout. You can change the order in which the components display by dragging the components in the layout column to the location you want.

The Customization GUI section retains its information, thereby allowing for a simple iterative process for updating the custom IP because it only affects the parameters that were added or removed.

## Adding Parameters to the Layout

To add parameters to the layout, do the following:

1. Click the hierarchical level (page or group) you want to add the parameter, right-click and select **Add Parameter**.

**Note:** Alternatively, you can click the **Add**  button from the toolbar.


2. In the Add Parameter dialog box, select the following options, and click **OK**.
  - **Available Parameters:** The available parameters of the custom IP. These are parameters that have not been previously added to the Customization GUI.
  - **Display Name:** The label text displayed for the parameter in the Customization GUI.

- **Tooltip:** The text displayed for the tooltip when hovering over the parameter in the Customization GUI.
- **Show Label:** The option to disable the label (Display Name) of the parameter.

## Adding Groups to the Layout

To add groups to the layout, do the following:

1. Click the hierarchical level (page or group) you want to add the group, right-click and select **Add Group** from the popup menu.


**Note:** Alternatively, you can click the **Add** button  from the toolbar.

2. In the Add Group dialog box, select the following options, and click **OK**.
  - **Display Name:** The label text displayed as a header for the group in the Customization GUI.
  - **Tooltip:** The text displayed for the tooltip when hovering over the group in the customization GUI.
  - **Layout:** The option to display the components of the group vertically or horizontally.

## Adding Pages to the Layout

To add pages to the layout, do the following:

1. Click the **Window** component, right-click, and select **Add Page**.


**Note:** Alternatively, you can click the **Add** button  from the toolbar.

2. In the Add Page dialog box, select the following options, and click **OK**.
  - **Display Name:** The label text displayed on the tab for the page in the Customization GUI.
  - **Tooltip:** The text displayed for the tooltip when hovering over the page contents in the Customization GUI.

## Adding Text to the Layout

To add text to the layout, do the following:

1. Click the hierarchical level (page or group) to which you want to add the text, then right-click and select **Add Text**.

**Note:** Alternatively, you can Click **Add** button  from the toolbar.

2. In the Add Text dialog box, select the following options, and click **OK**.

- **Display Name:** The label text displayed in the layout and the tooltip for the text in the customization GUI.
- **Text:** The text displayed in the Customization GUI.

## Review and Package

The Review and Package section, shown in the following figure, provides a summary of the IP and information about the settings you selected after packaging.

The IP is initially packaged at the end of the Create and Package IP wizard. If any changes occur to any of the packaging steps of the custom IP, the custom IP must be repackaged for the changes to go into effect.

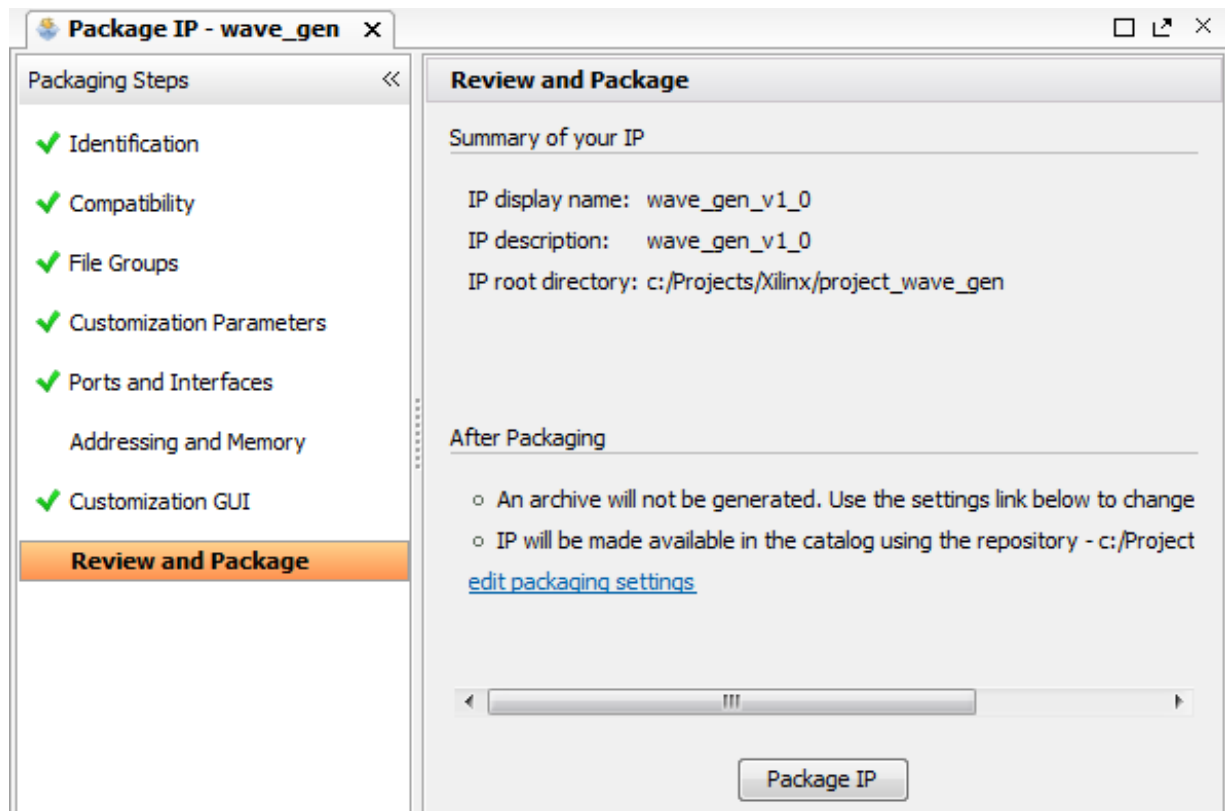


Figure 4-28: Package IP – Review and Package

The summary information is the identification data of the custom IP from the Identification section. To change the information, make the modifications in the Identification section, as described in [Identification](#).

The After Packaging section contains information about the Vivado IDE actions after packaging is complete.

The information that describes how the packager behaves is based on the IP Packager Settings, described in the [Using the Packager Settings](#), are, as follows:

- **Archive** Information: Select whether an archive of the IP is going to be created.
- **Repository Information:** The location of the IP for the Vivado repository manager.

In the IP Project Settings, you have an option of creating an archive of the custom IP definition. By default, the custom IP is generated at the IP root directory in which the source files are referenced relatively from the IP root directory location, if possible.

When an archive is created, the Vivado IDE compresses all the relatively referenced files into a single archive file. This archive file can be used in a Vivado repository by using the single archive file or extracting the archive file in a desired repository location.

After the IP is packaged, the custom IP is available in the IP Catalog of the current Vivado project. The Vivado IDE automatically adds the IP root directory of the custom IP to the Repository Manager in the IP Project Settings. The IP root directory is the location of custom IP definition and the location required to add to the Repository Manager of other Vivado projects that require the custom IP.

## Re-Packaging IP

To re-package IP, do the following:

1. Click the **Re-Package IP** button at the bottom of the Review and Package view.
2. If successful, click **OK** on the Package IP popup dialog box.




---

**IMPORTANT:** *Repackaging an IP and changing the default parameter values does not change the customization parameters in the IP instance. Each IP instance must be manually changed to the new default value if that change is required.*

---

## Creating an Archive of the IP

To create an archive of the IP:

1. Click the **Create archive of IP** option in the IP Packager Project settings, as described in [Using the Packager Settings](#). To open the IP Packager settings, click the **edit packaging setting** hyperlink.
2. In the **After Packaging** section, click the **edit** hyperlink to change the name and location of the archive.

**Note:** By default, the archive name is <Vendor>\_<Library>\_<Name>\_<Version#>.zip.

3. In the Package IP dialog box, select the following options, and click **OK**.
  - **Archive name:** The name of the archive file.
  - **Archive location:** The location where the archive file is created.

# Creating New Interface Definitions

---

## Introduction

Interface definitions provide the capability to group functional signals into a common interface grouping to use between IP in a Vivado® IP integrator diagram. The interface definition give you more a comprehensible diagram, and also enforces a standardized expectation that signals are designed to work between IP pairs.

Xilinx® provides many interface definitions, including standardized AXI protocols and other industry standard signaling; however, some legacy or custom implementations have unique IP signaling protocols.

You can define your own interface and capture the expected set of signals, and ensure that those signals exist between IP.

The Create Interface Definition option uses the IP-XACT industry standard specification. The interface definitions use two files that together correspond to a bus definition file (`myInterface.xml`) and an abstraction definition file (`myInterface_rtl.xml`).

The Vivado IDE uses the created interface definition to support mapping logical ports and inferring bus interfaces on the IP Definition in the IP packager.

The IP packager bus interface algorithms are almost completely data-driven, based on the bus definition and the algorithm works similarly with Xilinx-provided definitions as well as user created interfaces.

AXI4 memory-mapped and AXI4-Stream interfaces have additional DRC checks beyond the data-driven heuristics.

See [Inferring Signals](#) for more information about signal naming and inferring signal names.

## Creating a New Interface Definition

1. Select **Tools > Create Interface Definition**.

The Create Interface Definition dialog opens, as shown in the following figure.

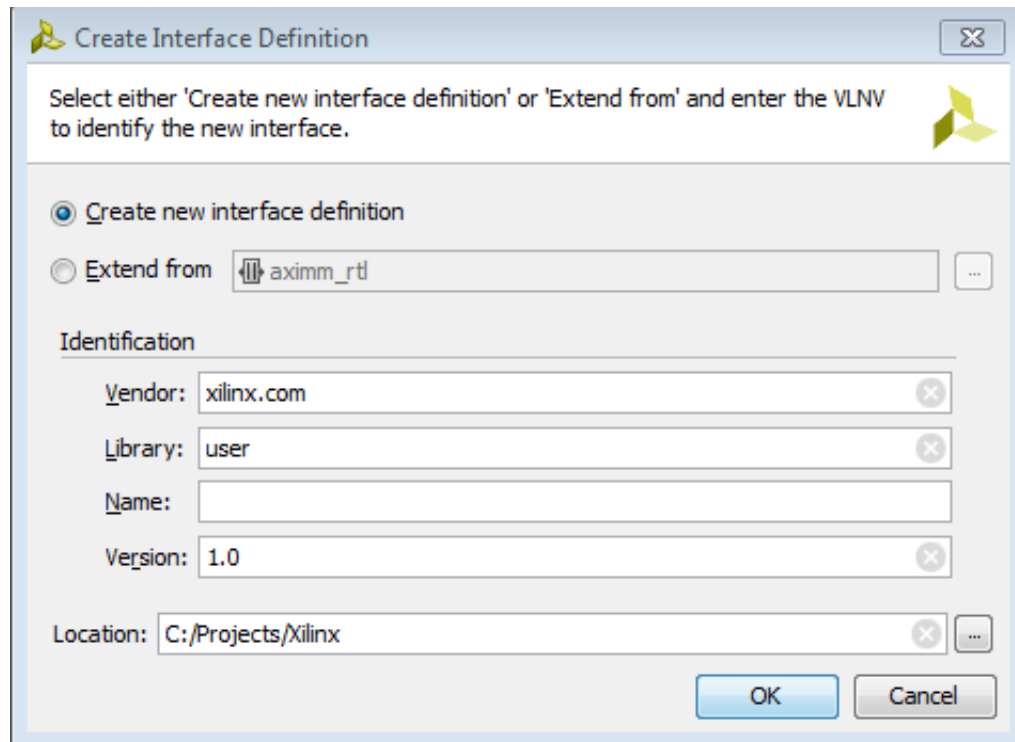


Figure 5-1: **Create Interface Definition Dialog Box**

2. Select one of the interface types from the following options:
  - **Create new interface definition:** Creates an empty interface, which is the typical use case.
  - **Extend from:** Creates a new interface using the existing interface definition port list as a template. This is an advanced feature.

If you create a new interface definition, the Interface tab opens without any pre-determined ports in the Ports list. You can add the ports as necessary.

If you extend from an existing interface, the Interface tab opens with the ports pre-populated in the Ports list. You can add or delete the ports as necessary.

3. In the Identification fields, enter the following information for the interface:
  - **Vendor:** The vendor of the interface. This is also the identifier for the vendor that displays in the interface definition **VLNV**. Provide the vendor information using the standard internet domain order.

- **Library:** The library in which the interface belongs. This is also the identifier for the library that displays in the interface definition VLNV.
  - **Name:** The name of the interface, which is also the identifier for the name that displays in the interface definition VLNV.
  - **Version:** The version of the interface, which is also the identifier for the version that displays in the interface definition VLNV.
  - **Location:** The directory where the pair of interface XML files are created.
4. Click **OK** to complete the interface creation.

## Using the Interface Definition Editor

After you finish creating the new interface definition, the workspace editor for the interface opens in the Interface Definition Editor for modifications, as shown in the following figure.

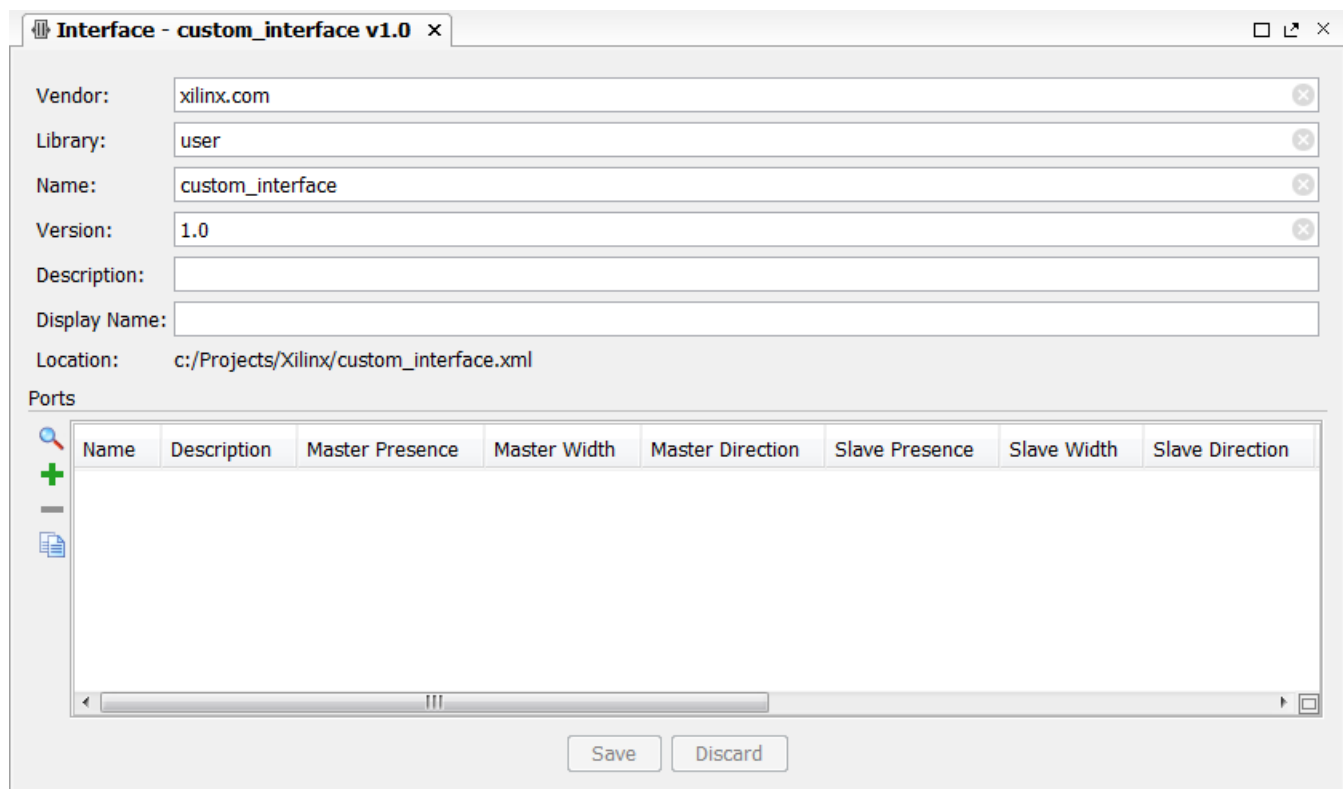


Figure 5-2: Interface Definition Editor

The following fields are available to describe the identification of the interface definition:

- **Vendor:** The vendor of the interface definition. This is also the identifier for the vendor that displays in the VLNV of the interface definition.

- **Library:** The library in which the interface belongs. This is also the identifier for the library that displays in the VLNV of the interface definition.
- **Name:** The name of the interface. This is also the identifier for the name that displays in the VLNV of the interface definition.
- Use short interface names because the IP block diagram has limited space.
- **Version:** The version of the interface. This is also the identifier for the version that displays in the VLNV of the interface definition.
- **Description:** The description of the interface definition.
- **Display Name:** The display name of the interface definition.
- **Location:** The name and location of the IP-XACT standard XML file.

## Adding Logical Ports to the Interface

1. On the ports list, right-click and select **Add Port**.

Alternatively, you can click the **Add** button  on the toolbar.

2. In the Add Port dialog box, shown in the following figure, set the following fields:

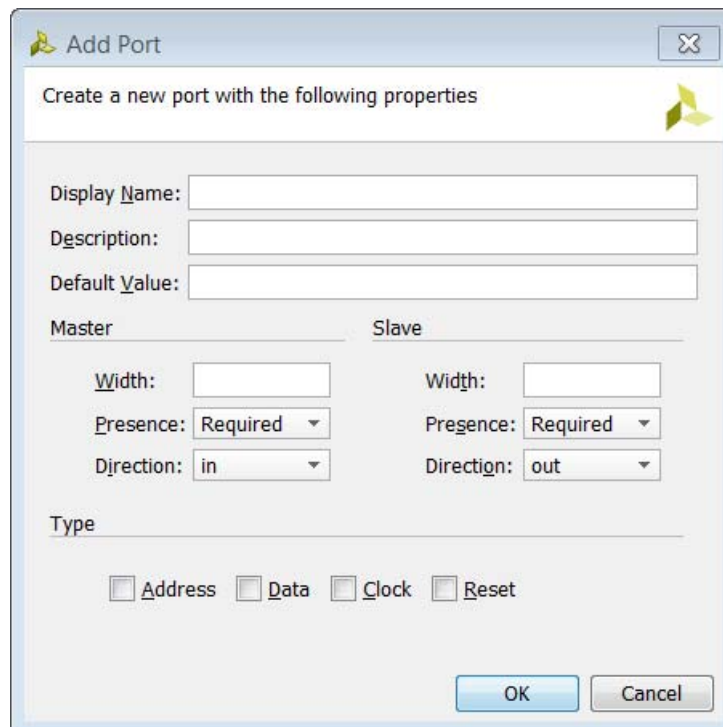


Figure 5-3: Add Port

- **Display Name:** The name of the logic port. In the Xilinx convention, these names are uppercase and are not prefixed with the name of the interface.

- **Description:** Enter a human-readable description for the logical port. Any transaction signaling must be described.
  - **Default Value:** When the port is optional, IP Integrator uses this default value to drive the **in** mode port when unconnected.
 

**Note:** If this port is part of the control logic, and is optional, it is important that the value is set to allow transactions to continue and not stall.-
3. In the Master section of the dialog box, set the following:
- **Width:** Required bit width of the mapped port for master mode bus interfaces exposed on an IP. Values are: -1 (undefined width), 1 (single bit signal), or a fixed value.
  - **Presence:** Set to either required or optional, indicating if the logical port is required to be mapped in master mode bus interfaces exposed on an IP. Allowed values are: optional, required, and illegal.
  - **Direction:** Required port mode when mapped for master mode bus interfaces exposed on an IP. Values are **in**, **out**, and **inout**.
4. In the Slave section of the dialog box, set the following:
- **Width:** Indicates the required bit width of this mapped port for slave mode bus interfaces exposed on an IP. Values are: -1 (undefined width), 1 (single bit signal), or a fixed value.
 

**Note:** Often the slave and master widths are the same.
  - **Presence:** Set to either required or optional, indicating if the logical port is required to be mapped in slave mode bus interfaces exposed on an IP. Values are optional, required, and illegal.
 

**Note:** Often the slave and master presence are the same.
  - **Direction:** Required port mode when mapped for slave mode bus interfaces exposed on an IP. Values are: **in**, **out**, and **inout**.
 

**Note:** Typically the slave and master directions are opposite. If the directions are the same, IP Integrator might not be able to properly directly connect the master and slave.
5. In the Type section of the dialog box, set the following:
- **Address:** Check this option when the logical port serves as an address line.
  - **Data:** Check this option when the logical port carries data and is not a control, addressing, or clock/reset.
  - **Clock:** Check this option when the logical port is a clock line.
  - **Reset:** Check this option when the logical port is a reset line.
6. Click **OK**.

## Adding Logical Ports from a Previously Defined Interface

As when creating the definition of the interface, you can additionally extend interfaces to the custom interface.

1. From the ports lists, right-click and select **Copy Ports From**.

Alternatively, click the **Copy Interface Ports** button  on the toolbar.

2. In the Copy Interface Ports dialog box, select the interface which logical ports from which to copy.
3. Click **OK**.

All the ports from the selected interface are populated in the Ports list.

**Note:** Any previously defined port with the same name as one of the imported ports is overwritten.

## Editing Logical Ports

After you create a logical port, a new row displays in the Create Interface Definition table. The logical port columns displays the information described in the Add Port dialog box when adding a logical ports.

To edit the port information, select the column cell for the port you want to edit. The cell will become editable to make the desired changes.

## Setting Tristate Signaling

The following cells provide advanced information for tristate signaling and can be specified to allow a cross-over like connection between masters and slaves with a triple of tristate signals.

Because a device does not have true tristate lines inside the fabric, a triple of signals (**in**, **out**, **tristate**) represents a tristate pair (**in**, **out**, **tristate**).

When you create an interface with the triple of signals, add the following information, which is required to allow connectivity between masters and slaves.

- **Tristate Role:** This indicates if the port is part of a tristate triple of signals, and which role. Non-tristate ports leave this cell blank. Values are **in**, **out**, and **tristate**.
- **Group:** When there are multiple tristate triples in the interface, it is important that the three signals be grouped together. This field must have the same string identifier entered for the three ports that form the same unique tristate.

## Completing the Interface Definition Creation

When you have completed providing the required information for the Create Interface Definition option, click the **Save** button centered at the bottom of the view to save the pair of XML files. To throw away edits, click the **Discard** button.

## Re-Editing Interface Definitions

To edit an interface definition, in **File > Open IP-XACT File**, browse to either of the bus definition file (`myInterface.xml`) or abstraction definition file (`myInterface_rtl.xml`). Both files must be in the same directory for the Vivado GUI to edit the definition correctly.

## Using a New Interface Definition

Similar to IP Definitions, the IP-XACT bus definition files must be in a repository path for the IP packager and IP integrator tools to use the files.

**RECOMMENDED:** Structure your /IP repository to have a single directory with an /interfaces subdirectory to contain the interface definition files, and a peer IP subdirectory to contain your IP definition files. A repository path must be added at, or preceding, the directory that contains the IP definition file. For example, `<repository_path>/IP/interface` or `<repository_path>/IP/IP`.

To see the interface definition files in the IP Catalog:

1. Ensure that the catalog is organized to display by repository. See [Using the Packager Settings](#) for the display settings.
2. Select the user repository with the new interface definition files. A tab displays the names and number of interfaces.

**Note:** After the interface definitions are recognized in the projects catalog, the IP packager lets the IP Definition feature create bus interfaces, based on the bus definition, including the ability to infer the bus interface port mapping.



**IMPORTANT:** The IP packager uses a name-matching heuristic which is based on the logical port names matching the physical ports in the IP. IP packager automatically detects AXI interfaces when the names of the HDL modules ports follow the `<interface_name>_<AXI signal name>` convention. For example, `s_axi_awvalid`, where `s_axi` is the interface name and `awvalid` is one of the AXI signal names used in the interface. Module ports with the same `<interface_name>` are mapped into the named interface.

Use short interface names because the IP block diagram has limited space.

# Encrypting IP in Vivado

---

## Introduction

The Vivado Design Suite® supports version 2 (V2) of the IEEE-1735-2014, *Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)* [Ref 9].



---

**TIP:** *In the event of any discrepancies or errors in this document, the IEEE 1735-2014 standard document supersedes this content.*

---

The IEEE-1735 version 2 encryption feature requires a license. Refer to this Vivado Answer Record: [68071](#).

The scope of IP encryption and protection in the Vivado Design Suite tool flow is limited up to the point of bitstream generation. Protection of the bitstream itself requires encryption of the bitstream on the programmed Xilinx® device, a step that usually resides in the hands of the IP user, not the IP creator. See this [link](#) to the *Vivado Design Suite User Guide: Programming and Debug* (UG908) [Ref 12] `BISTREAM. ENCRYPTION. ENCRYPT` property, for more information on bitstream encryption.

Although there are many different formats of design source files, IEEE-1735-2014 only applies to Verilog, SystemVerilog, and VHDL formats. These RTL standards are also governed by the IEEE, and it is by mutual agreement that those standards will allow IEEE-1735 to define behavior in IP security until such a time as the recommendations in IEEE 1735 can be retrofitted into the original language standards contained in the language reference manuals (LRMs). Popular netlist formats, like EDIF, are not covered by IEEE 1735.

---

## Access Rights Management

IEEE-1735-2014 V2 lets an IP author manage access rights to their IP, expressing how they want various vendor tools to interact with the IP. These access rights are placed into sections of the RTL code, inside the encryption definition area.

The following sections describe the types of rights, as follows:

- [Common Rights](#)
- [Vendor-Specific Rights](#)
- [Conditional Rights](#)

## Common Rights

Common rights specify general guidelines that apply to all EDA tools. These rights apply to all the tool-vendors specified by the IP author in the encryption definition area. If a tool vendor cannot comply with a common right, the tool vendor must error out and stop processing the encrypted file.

## Vendor-Specific Rights

The IP author can also grant each tool vendor specific rights that apply just to their tool flow. An example of this would be controlling the behavior of Vivado Logic Analyzer probes, which is unique to Xilinx. The value of an access right in the vendor-specific area overrides the value in the common rights block of the same name for the specified vendor.

## Conditional Rights

IEEE-1735-2014 V2 also introduces a language construct which allows IP authors to specify different access rights under different conditions. An example of a conditional right might include not allowing decryption of the IP for simulation.

---

# Understanding IEEE 1735 Structural Elements

The basic regions in a design protected with IEEE-1735-2014 V2 are, as follows:

- **Definition area:** The IP author determines which tool vendors to support, and what access rights to grant to all tools, or to a specific tool. Most of the definition area remains in plain-text, even after encryption, because there are basic formatting instructions that the design tools need in order to properly decrypt the encrypted data.
- **Encrypted Key Definition:** Contains the encryption keys for tool vendors supported by the IP.

Although technically part of the Definition area above, the key definition must, itself, be encrypted by the encryption tool in order to ensure that only authorized vendors can access the key and decrypt the encrypted payload.

- **Encrypted payload:** The encrypted IP Verilog, SystemVerilog, or VHDL source code.

- **Plain-text payload:** An unencrypted portion of the IP source-code. Within a single RTL source file, some content can be encrypted, and other content can be unencrypted. You do not need to encrypt the whole design file.




---

**IMPORTANT:** *In keeping with 1735 recommendations, Vivado handles encryption at the granularity of the module or entity and architecture pair.*

---

The following sections define different sections of the definition area, and related pragmas that must be defined in the IP source code to support encryption. The examples are written in VHDL form. The Verilog and System Verilog syntax is the same except the ``pragma protect` keywords in Verilog replaces the ``protect` keyword in VHDL.

## Version and Other Pragmas

IEEE-1735-2014 V2 compliance level is specified by the **Version** pragma, as follows:

```
`protect version = 2
```




---

**TIP:** *Version 1 of IEEE-1735 -2014 is supported by Xilinx under an early access program. Improper use of V1 can inadvertently expose your design. Please contact your local sales office or Applications Engineer for more information.*

---

The encryption tool might add some optional pragmas to identify itself, as follows:

```
`protect encrypt_agent = "XILINX"  
'protect encrypt_agent_info = "Xilinx Encryption Tool 2016"
```

## Common Block Definition

Common rights are placed into `commonblock` section. The beginning and end of this block are defined by the following pragmas, as follows:

```
`protect begin_commonblock  
...  
'protect end_commonblock
```

### Common Rights

The set of common rights is defined in the following table, with a description of the default value, valid values as defined by the IEEE specification, and valid values as accepted by the Vivado tools. The term *delegated* means that the IP author is delegating an access right to the tools to do what is needed to process the IP.

Rights must evaluate to the Xilinx *valid values* under all circumstances. Evaluation to non-supported values causes the tool to stop processing the encrypted data.

Table 6-1: Common Rights

Name	Purpose	Default Value	Valid Values <sup>a</sup>	Xilinx Valid Values
error_handling	What is the tool allowed to show in Error messages?	"delegated"	"delegated" "srcrefs" "plaintext"	"delegated" <sup>b</sup>
runtime_visibility	What is tool allowed to show during display, tcl, or output reports?	"delegated"	"delegated" "interface_names" "all_names"	"delegated"
child_visibility	If a protected module instantiates an unprotected child, how should error_handling and runtime_visibility be handled on that child? <b>Note:</b> Displayed messages could expose path-names through protected regions.	"delegated"	"delegated" "allowed" "denied"	"delegated" "allowed"
decryption	Is the tool even allowed to decrypt the module? This it will often be used in conditional rights.	"delegated"	"delegated" "true" "false"	"delegated" "true" "false" <b>Note:</b> "delegated" = "true"

a. Valid values are case sensitive, and must be specified as shown.

b. The behavior of Vivado tools can vary for delegated access rights. See [Table 6-5, page 88](#) for more information.

## Vendor-Specific Tool Block Definition

Tool vendor specific rights are placed into vendor specific `toolblock` sections. There has to be at least one tool block for each encrypted block. This block also contains the encryption key definition for the tool vendor.

The beginning and end of this block are defined by the following pragmas:

```

`protect begin_toolblock
...
`protect end_toolblock=""
    
```



**TIP:** Tool blocks are vendor-specific; consequently, each vendor has a separate `toolblock` defining their encryption key and access rights.

## Xilinx Tool Rights

Xilinx has created five specific tool rights that control Vivado tool behavior, as listed in the following table.

Table 6-2: Xilinx Specific Tool Rights

Xilinx Right Name	Meaning	Xilinx Valid Values	Default Values
xilinx_configuration_visible	Are the LUT values allowed to be visible in viewers, editors, and so forth, in Vivado?	"true", "false"	"false"
xilinx_enable_modification	Can netlist information within the protected region (hierarchy, connections, LUTs, and so forth) be modified using the Vivado tool?	"true", "false"	"false"
xilinx_enable_probing	Is the customer allowed to insert or instantiate Vivado debug probes in the protected region?	"true", "false"	"false"
xilinx_enable_netlist_export	Is Vivado allowed to export a netlist of protected region?	"true", "false"	"true"
xilinx_enable_bitstream	Is the Vivado tool allowed to write out a bitstream?	"true", "false"	"true"

## Key Definition and Rights Digest Method

The encryption key and rights digest method must be defined as part of a vendor specific `toolblock`. These are a set of mandatory pragmas which define the vendor public encryption key, key related attributes, and method for calculating the digest for rights in the `toolblock` of each vendor. The following example shows these pragmas and their valid values for Xilinx.

```

`protect key_keyowner = "Xilinx"
`protect key_method = "rsa"
`protect key_keyname = "xilinx_2016_05"
`protect rights_digest_method = "sha256"
`protect key_public_key
...
    
```

## Rights Definition

The basic syntax for expressing an access right uses the `control` keyword introduced for IEEE-1735-2014 V2.

Generally, the pragma looks as follows:

```
`protect control <right> = <rights_expression>
```

Where:

- `control` is the keyword identifying an access right.
- `<right>` is the access right being defined.
- `<rights_expression>` is a statement of the availability of the right.




---

**IMPORTANT:** *Valid values for the `rights_expression` are case sensitive.*

---

For example:

```
`protect control xilinx_configuration_visible = "false"
```

## Conditional Rights Definition

Conditional rights let IP authors specify different access rights under different evaluated conditions. The basic syntax for defining a conditional access right is:

```
`protect control <right> = <condition> ? <true_expression> : <false_expression>
```

Where:

- `control` is the keyword identifying an access right.
- `<right>` is the access right being defined.
- `<condition>` is a condition test for the specified access right.
- `<true_expression>` is the rights expression applied when the `<condition>` is true.
- `<false_expression>` is the rights expression applied when the `<condition>` is false.
- When `<condition>` evaluates as true, the right takes the value of `<true_expression>`, else right gets the value of `<false_expression>`.

For example:

```
`protect control decryption = (xilinx_activity==simulation) ? "false" : "true"
```

The results of the conditional expression defined above, when the activity is simulation, are detailed in the following table.

Table 6-3: Conditional Expression Evaluated

<b>xilinx_activity</b>	<b>Condition (xilinx_activity==simulation)</b>	<b>?: Expression Result</b>	<b>Comment</b>
simulation	true	"false"	Source data is <i>not</i> decrypted.
synthesis	false	"true"	Source data is decrypted.
implementation	false	"true"	Source data is decrypted.

### Conditional Rights Qualifiers

The IEEE 1735-2014 V2 standard defines `activity` as a conditional keyword. However, Xilinx does not support this keyword, and instead uses `xilinx_activity` as a conditional keyword.



**IMPORTANT:** Other conditional keywords defined by IEEE 1735-2014 standard are not supported by Xilinx.

The `xilinx_activity` keyword breaks down the tool flow into abstract activities that align with the specific activities supported by the Vivado Design Suite. Supported values for the `xilinx_activity` keyword are described in the following table.

Table 6-4: `xilinx_activity` Condition Keywords

<b>Value</b>	<b>Definition</b>
simulation	The abstract activity that applies to tools that provide execution semantics. For Vivado simulator, this would refer to simulation at any stage of the tool chain.
synthesis	The abstract activity that applies to tools that transform the IP into another symbolic form. In the Vivado tool flow, this equates to Vivado synthesis.
implementation	The abstract activity that applies to tools that does place and route of the netlist on FPGA device resources. In Vivado, this equals Vivado Implementation phase.



**TIP:** The condition can test for `xilinx_activity==simulation`, `synthesis`, or `implementation`.

### Valid Rights Specification for Xilinx

The following is a partial VHDL encryption sample that shows a valid specification of the `error_handling` right.

Although the `commonblock` specifies a value of `"screfs"`, which Xilinx *does not* support; that value is overridden in the Xilinx-specific `toolblock`. Here, the IP author has defined the `error_handling` right as `"delegated"`, which Xilinx does support.

```
\protect begin_commonblock
\protect control error_handling = "screfs"
\protect end_commonblock
\protect begin_toolblock
\protect key_keyowner="Xilinx",...
\protect control error_handling = "delegated"
```

### Invalid Rights Specification for Xilinx

In the following invalid example, `error_handling` specifies a value of `"screfs"` in the `commonblock`, which Xilinx *does not* support. The Xilinx `toolblock` contains a conditional statement to override the `error_handling` right; however, when the `xilinx_activity` condition evaluates to `TRUE`, the unsupported value `"screfs"` is assigned to the right. In this case, the Vivado tool issues an error and stops when attempting to decrypt this block.

```
\protect begin_commonblock
\protect control error_handling = "screfs"
\protect end_commonblock
\protect begin_toolblock
\protect key_keyowner="Xilinx", ...
\protect control error_handling = (xilinx_activity==synthesis)? "screfs" :
"delegated"
```

## Encryption Payload

This is the IP RTL source code which needs to be protected. The beginning and end of the source code to be encrypted is defined using the following pragmas:

```
\protect begin
...
\protect end
```




---

**TIP:** Remember that a module or entity/architecture pair definition is the smallest object within the RTL code that can be specified within the encryption payload.

---

## Understanding How Rights Affect Vivado Tools

Following table shows how different rights affect the various Vivado tools. It includes both common rights and Xilinx-specific rights.

Table 6-5: How Rights Affect Vivado Tool Activities

Right/ Type	Simulation	Synthesis Vivado Synthesis	Implementation Vivado Implementation/ Timing/Viewers
error_handling/ common	<ul style="list-style-type: none"> <li>delegated = Hide error message references inside encrypted areas.</li> </ul>	<ul style="list-style-type: none"> <li>delegated = Hide error message references inside encrypted areas.</li> </ul>	<ul style="list-style-type: none"> <li>Not Applicable (NA)<sup>a</sup></li> </ul>
runtime_visibility/ common	<ul style="list-style-type: none"> <li>delegated = Follow current Vivado simulator behavior-hiding visibility in hierarchy viewer, and so forth.</li> </ul>	<ul style="list-style-type: none"> <li>delegated = Follow current Vivado default behavior-hiding visibility.</li> </ul>	<ul style="list-style-type: none"> <li>delegated = All names/hierarchy visible</li> </ul>
child_visibility/ common	<ul style="list-style-type: none"> <li>delegated = No change to current Vivado simulator behavior (equivalent to Denied - hidden)</li> <li>allowed = Show hierarchy, error messages for child levels.</li> </ul>	<ul style="list-style-type: none"> <li>delegated = No change to current behavior (equivalent to Denied - hidden)</li> <li>allowed = Show names/message for child levels.</li> </ul>	<ul style="list-style-type: none"> <li>delegated = allowed = No change to current behavior of Vivado implementation.</li> </ul>
decryption/ common	<ul style="list-style-type: none"> <li>delegated/true = Read in decrypted file.</li> <li>false = Do not read in decrypted file - issue message and stop processing.</li> </ul>	<ul style="list-style-type: none"> <li>delegated/true = Read in decrypted file.</li> <li>false = Do not read in decrypted file - issue message and stop processing.</li> </ul>	<ul style="list-style-type: none"> <li>delegated/true = Read in decrypted file</li> <li>false = Do not read in decrypted file - issue message and stop processing</li> </ul>
xilinx_enable_netlist_export/ xilinx-specific	<ul style="list-style-type: none"> <li>NA</li> </ul>	<ul style="list-style-type: none"> <li>true = write_vhdl and write_verilog output netlists</li> <li>false = write_vhdl and write_verilog do not run</li> </ul>	<ul style="list-style-type: none"> <li>true = write_vhdl and write_verilog output netlists</li> <li>false = write_vhdl and write_verilog do not run</li> </ul>
xilinx_configuration_visible/ xilinx-specific	<ul style="list-style-type: none"> <li>NA</li> </ul>	<ul style="list-style-type: none"> <li>true = LUT contents visible.</li> <li>false = LUT contents NOT visible.</li> </ul>	<ul style="list-style-type: none"> <li>true = LUT contents visible.</li> <li>false = LUT contents NOT visible.</li> </ul>

Table 6-5: How Rights Affect Vivado Tool Activities (Cont'd)

Right/ Type	Simulation	Synthesis Vivado Synthesis	Implementation Vivado Implementation/ Timing/Viewers
xilinx_enable_modification/ xilinx-specific	<ul style="list-style-type: none"> <li>• NA</li> </ul>	<ul style="list-style-type: none"> <li>• true = Netlist modifications are allowed</li> <li>• false = Netlist modifications are allowed</li> </ul>	<ul style="list-style-type: none"> <li>• true = Netlist modifications are allowed</li> <li>• false = Netlist modifications are allowed</li> </ul>
xilinx_enable_probing/ xilinx-specific	<ul style="list-style-type: none"> <li>• NA</li> </ul>	<ul style="list-style-type: none"> <li>• true = Allow customer to insert Analyzer probes into encrypted module/entity.</li> <li>• false = Analyzer probes may not be inserted.</li> </ul>	<ul style="list-style-type: none"> <li>• true = Analyzer probes may be inserted into encrypted module/entity.</li> <li>• false = Analyzer probes may not be inserted.</li> </ul>
xilinx_enable_bitstream/ xilinx-specific	<ul style="list-style-type: none"> <li>• NA</li> </ul>	<ul style="list-style-type: none"> <li>• NA</li> </ul>	<ul style="list-style-type: none"> <li>• true = Grant right to generate bitstream.</li> <li>• false = Do not generate bitstream.</li> </ul>

a. NA = Has no affect on the Vivado tool.

## RTL Encryption Examples

### VHDL

The following VHDL example features a common block that delegates error handling, and prohibits decryption during simulation using the IEEE defined `activity` condition.

Then, a Xilinx tool-specific block hides the configuration, prevents modification, and prevents probing of the encrypted logic. These are the three most-recommended Xilinx-specific rights. The tool-specific block also overrides the common block, granting the decryption right during simulation with the `xilinx_activity` condition.



**TIP:** Remember `activity` is not a Xilinx supported condition.

```

`protect version = 2
`protect encrypt_agent = "XILINX"
`protect encrypt_agent_info = "Xilinx Encryption Tool 2015"
`protect begin_commonblock
`protect control error_handling = "delegated"
`protect control decryption = (activity==simulation)? "false" : "true"
`protect end_commonblock
`protect begin_toolblock
    
```

```

`protect rights_digest_method="sha256"
`protect key_keyowner = "Xilinx", key_method = "rsa", key_keyname =
"xilinx_2016_05", key_keyowner
...
`protect control xilinx_configuration_visible = "false"
`protect control xilinx_enable_modification = "false"
`protect control xilinx_enable_probing = "false"
`protect control decryption = (xilinx_activity==simulation)? "false" : "true"
`protect end_toolblock = ""
`protect begin
-- Secure Data Block
-- Protected IP source code is inserted here.
...
...
...
`protect end
    
```

## Verilog



**TIP:** The Verilog and System Verilog syntax is the same as VHDL except the ``pragma protect` keywords in Verilog replace the ``protect` keyword in VHDL.

The following example is a Verilog version of the preceding VHDL example:

```

`pragma protect version = 2
`pragma protect encrypt_agent = "XILINX"
`pragma protect encrypt_agent_info = "Xilinx Encryption Tool 2015"
`pragma protect begin_commonblock
`pragma protect control error_handling = "delegated"
`pragma protect control decryption = (activity==simulation)? "false" : "true"
`pragma protect end_commonblock
`pragma protect begin_toolblock
`pragma protect rights_digest_method="sha256"
`pragma protect key_keyowner = "Xilinx", key_method = "rsa", key_keyname =
"xilinx_2016_05", key_public_key
...
`pragma protect control xilinx_configuration_visible = "false"
`pragma protect control xilinx_enable_modification = "false"
`pragma protect control xilinx_enable_probing = "false"
`pragma protect control decryption = (xilinx_activity==simulation)? "false" : "true"
`pragma protect end_toolblock = ""
`pragma protect begin
// Secure Data Block
// Protected IP source code is inserted here.
...
...
...
`pragma protect end
    
```

## Encrypting IP with Vivado

The Vivado Design Suite provides a Tcl command, `encrypt`, that performs encryption on IEEE-1735-2014 V2 valid Verilog, SystemVerilog, and VHDL source files.

See this [link](#) to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 11] for details.

The `encrypt` command can also be used on the Verilog or VHDL output from any stage of the Vivado tool flow. The RTL source code can be encrypted, and then any files output by the `write_verilog` or `write_vhdl` commands later in the tool flow also have the protected IP automatically encrypted.

### Syntax:

```
encrypt [-key <arg>] -lang <arg> [-quiet] [-verbose] [-ext <arg>] <files>...
```

The `encrypt` Tcl command is available from the Tcl Console in the Vivado IDE, or in the standalone Vivado Tcl shell. By default, this command encrypts the existing files in-place, which means that the original files are overwritten and replaced with the encrypted versions unless you use the `-ext` option.



**IMPORTANT:** *Either use the `-ext` option to prevent `encrypt` from overwriting, or make copies of your source file prior to running the `encrypt` command.*

The IEEE-1735-2014 V2 definition area can be placed either in-line in the HDL source file, or in a separate key file. The `encrypt -key` option directs the Vivado tool where to look for this information.

- The `-key` option specifies an RSA key file that includes the IEEE-1735-2014 V2 supported pragmas that provide the encryption key, define access rights, and other optional information. The key file must use the same language and extension as the source files being encrypted (VHDL, Verilog, SystemVerilog).



**TIP:** *If `-key` is specified and the source file already has the encryption key and required pragmas in the definition area, the `-key` argument will be ignored.*

- If `-key` is not specified, the Vivado tool looks for encryption keys and pragmas embedded within the source files that is being encrypted.

The following example uses the `-key` option to do the following:

- Point to an encryption key file.
- Specify the target language as Verilog.

- Specify the design file to encrypt.
- Specify a new file extension to use when generating the encrypted files to prevent overwriting the original source files.

```
encrypt -lang verilog -ext .vp -key keyfile.txt myip.v
```

The final example searches for the encryption key inside the design source file, specifies the target language as Verilog, specifies the design file to encrypt, and specifies a new file extension to use when generating the encrypted files to prevent overwriting the original source files:

```
encrypt -lang verilog -ext .vp my_ip.v
```

## Xilinx IEEE-1735-2014 Public Key

Xilinx public keys (VHDL and Verilog) can be found inside the Vivado Design Suite installation hierarchy at the following location:

```
<Install_Dir>/Vivado/<version>/data/pubkey/
```

## Third-Party Public Keys

To encrypt IP such that it can be read by a IEEE-1735-2014 V2 compliant third-party tool, you must obtain the public encryption key directly from the third-party tool provider. The third-party key definition must be in the pre-defined format, and can be placed into the key file next to the Xilinx key. During encryption, the Vivado tools use both public keys.



---

**IMPORTANT:** Consult with third-party tool vendors to determine if they support V2 compliance level. Vendors who only support IEEE-1735-2014 version 1 are not compatible with V2 rights.

---

---

## Best Practices

- Keep encryption keys and pragmas in a key file and point to it using `encrypt -key` option.
- Use the `encrypt -ext` option to avoid inadvertently overwriting input files.
- Obtain public keys from tool vendors.



---

**CAUTION!** *Beware cutting and pasting from PDF and Word documents as these operations often corrupt the key; corruption is usually not apparent until the file is subsequently decrypted resulting in syntax errors.*

---

- Use the same key file for as much IP as possible, which allows greater optimization of the resulting netlist.
- Encrypt all the files of an IP in a single call to `encrypt`.
- Do not split a Verilog module between multiple encryption blocks and/or plain text sections.
- In VHDL, put the entire entity and architecture pair in a single encryption block.
- Verify that the encrypted code loads correctly into Vivado and a subsequent `write_verilog` re-encrypts all the secure design elements.
- Verify interoperability with third-party tools.

---

## Known Limitations

- The schematic view displays the structure of the encrypted blocks, and it is possible to browse the hierarchy of the secured netlist with the schematic viewer or using Tcl commands in the Tcl Console.
- Encryption does not prevent connectivity tracing within Vivado. If net-names/structure is an important part of the IP, then use net-name obfuscation as a way to make name tracing harder.

**Note:** Xilinx does not provide an obfuscater.

# Standard and Advanced File Groups

## Introduction

This appendix lists the Standard and Advanced file group categories and descriptions.



**IMPORTANT:** *NGC format files are not supported in the Vivado Design Suite for UltraScale® devices. It is recommended that you regenerate the IP using the Vivado Design Suite IP customization tools with native output products. Alternatively, you can use the NGC2EDIF command to migrate the NGC file to EDIF format for importing. However, Xilinx recommends using native Vivado IP rather than XST-generated NGC format files going forward*

## Standard File Groups

The following table lists the **Standard** file group types and a description.

Table A-1: **Standard File Group Types and Descriptions**

Standard File Groups	
Examples	Files that make up an example. Typically contains a constraint (XDC), HDL, and/or XIT files. Vivado uses these files to seed a new Vivado example project and shows this to the end-user for their exploration. The files are available for both synthesis and simulation.
Product Guide	The production guide for an IP.
Readme	Any required <code>readme.txt</code> file.
Simulation	Simulation files to deliver. Use when you have a mix of VHDL and Verilog to simulate together. Typically, exclusive of "VHDL Simulation" and "Verilog Simulation." The files could be the same as the files in the corresponding synthesis file group (when the synthesis files can also be used for simulation) or could be different (when a behavioral simulation model files are to be used).
Synthesis	Synthesis files to deliver. Use when you have a mix of VHDL and Verilog to synthesize to together. Typically exclusive of "VHDL Simulation" and "Verilog Simulation." Adding a constraint (XDC) file here causes the constraint file to be applied to the top-level of the IP during implementation.

Table A-1: Standard File Group Types and Descriptions (Cont'd)

Standard File Groups	
Verilog Simulation	Simulation files to deliver. Use when you have a Verilog only representation to simulate. You might see both this file group and "VHDL Simulation" to allow the ability to have a language-specific IP simulation. The files could be the same as the files in the corresponding synthesis file group (when the synthesis files can also be used for simulation) or might be completely different (when a behavioral simulation model files are to be used).
Verilog Synthesis	Synthesis files to deliver. Use when you have a Verilog-only representation to synthesize. You might see both this file group and "VHDL Synthesis" to allow the ability to have a language specific implementation of the IP. Adding a constraint (XDC) file here applies the constraint file to the top-level of the IP during implementation.  <b>Note:</b> Synthesis run files are not stored in the IP packager.
VHDL Simulation	Simulation files to deliver. Use when you have a VHDL only representation to simulate. You might see both this file group and "Verilog Simulation" to allow the ability to have a language specific IP simulation. The files could be the same as the files in the corresponding synthesis file group (when the synthesis files can also be used for simulation) or might be completely different (when a behavioral simulation model files are to be used).
VHDL Synthesis	Synthesis files to deliver. Use when you have a VHDL only representation to synthesize. You might see both this file group and "Verilog Synthesis" to allow the ability to have a language specific implementation of the IP. Adding a constraint (XDC) file here causes the constraint file to be applied to the top-level of the IP during implementation.

## Advanced File Groups

The following table contains a listing and a description of each **Advanced** file group type.

Table A-2: Advanced File Group Types and Descriptions

Advanced File Groups	
Catalog Disabled Icon	GUI icon to show in the IP Catalog when the IP is disabled. (For example, when the IP does not support the selected device family). Supported file types are GIF, JPEG, and PNG.
Catalog Icon	GUI icon to show in the IP Catalog. Supported file types are GIF, JPEG, and PNG.
C Simulation	Use to deliver files for c-model simulation. These files are copied out without any further processing (excepting Tcl and XIT which is always evaluated). If you are delivering pre-compiled libraries, it is suggested to deliver these in machine specific directories following the Vivado convention (for example; Inx32, Inx64, win32, and win64).
Custom UI Layout	Tcl file to control custom GUI layout and customization. Only a single file is allowed, additional files should be added to the Utility XIT file group.
Data sheet	IP Data sheet.
Encrypted Data sheet	Encrypted IP Data sheet.

Table A-2: Advanced File Group Types and Descriptions (Cont'd)

Advanced File Groups	
Example Implementation	Files to apply for implantation, post synthesis in an example design. Typically only XDC files should be added.
Examples Script	Script to create example project. Typically only Tcl files should be added.
Examples Script Extension	Script to extend our default example project script. Typically only Tcl files should be added.
Examples Simulation	Files that make up a simulation example design. Typically exclusive of the "Examples" and "Example Synthesis" file groups.
Getting Started Guide	Getting Started Guide.
Implementation	Files that make up an implementation design. Typically this file group contains only implementation (XDC) constraints.
MATLAB Simulation	MATLAB® software simulation file.
MIF Files	MIF file.
Miscellaneous	Vivado copies any files in the group to disk during generation. It is recommended that you use another, appropriate file group.
Reference Design	Files that make up a reference design.
Software Drivers	Any created software drivers.
System C Simulation	Use to deliver files for System-C Simulation. If you are delivering pre-compiled libraries, it is suggested to deliver these in machine-specific directories following the Vivado convention (such as lnx32, lnx64, win32, and win64).
System Generator Simulation	Any System Generator simulation.
System Verilog Simulation	Use to deliver files for System-Verilog simulation.
Test Bench	One or more test bench files written in both VHDL and Verilog. Add all mixed language test bench files for delivery to the end-user, even if there are multiple top modules.
UI DRCs	User Interface Design Rule Checks.
UI Icon	GUI icon to use in IP Customization GUI. Supported file types are GIF, JPEG, and PNG.
UI Layout	Tcl file to control GUI layout and customization. Only a single file is allowed, additional files should be added to the Utility XIT file group.
Upgrade Tcl Functions	Xilinx scripts supporting upgrades from previous versions of IP. These allow the later version of provide an equivalent instance to the earlier version of IP.
Utility XIT/TTCL	Add any utility files used during TTCL, XIT, or XSpice generation. These can either be data files, include files or extra evaluation Tcl files. For XSpice, only a single Tcl file is allowed in its file group, therefore any add any additional supporting Tcl files here.
Verilog Instantiation Template	Verilog instantiation template.
Verilog Test Bench	Test bench files written in Verilog.
Version Information	Version information.
VHDL Instantiation Template	VHDL instantiation template.
VHDL Test Bench	Test Bench files written in VHDL.

# Additional Resources and Legal Notices and Legal Notices

---

## Xilinx Resources

Support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

---

## Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

---

## Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

**Note:** For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

---

## References

### Xilinx Web Sites

1. [Vivado IP Versioning](#)
2. Vivado Answer Record: [68071](#)

### Vivado Design Suite Documentation

The following documents are either cited within this guide or are helpful resources:

3. *Vivado Design Suite User Guide: Design Flows Overview* ([UG892](#))
4. *Vivado Design Suite User Guide: Hierarchical Design* ([UG905](#))
5. *Vivado Design Suite User Guide: Using Constraints* ([UG903](#))
6. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
7. *Vivado Design Suite Tutorial: Designing with IP Tutorial* ([UG939](#))
8. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
9. *Vivado Design Suite User Guide: System-Level Design Entry* ([UG895](#))
10. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* ([UG973](#))
11. *Vivado Design Suite Tcl Command Reference Guide* ([UG835](#))
12. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
13. *Vivado Design Suite Tutorial: Logic Simulation* ([UG937](#))
14. *Vivado Design Suite Tutorial: Design Flows Overview* ([UG888](#))
15. *Vivado Design Suite User Guide: Using Tcl Scripting* ([UG894](#))
16. *Vivado Design Suite User Guide: Using the Vivado IDE* ([UG893](#))
17. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* ([UG994](#))
18. *Vivado AXI Reference Guide* ([UG1037](#))
19. *Vivado Design Suite Properties Reference Guide* ([UG912](#))
20. *Vivado Design Suite Tutorial: Programming and Debugging* ([UG936](#))
21. *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* ([UG1119](#))
22. *Vivado Design Suite User Guide: Synthesis* ([UG901](#))

23. *UltraFast Design Methodology Guide for the Vivado Design Suite* ([UG949](#))
24. *UltraScale Architecture Libraries Guide* ([UG974](#))
25. *Vivado Design Suite Tutorial: Revision Control Systems* ([UG1198](#))
26. *Vivado Design Suite User Guide: Implementation* ([UG904](#))

[Vivado Design Suite Documentation](#)

## **Xilinx QuickTake Videos**

1. [Vivado Design Suite QuickTake Video: Creating an AXI Peripheral in Vivado](#)
2. [Vivado Design Suite QuickTake Video: Packaging Custom IP Integrator](#)
3. [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#)

[Vivado Design Suite QuickTake Video Tutorials](#)

## **Xilinx IP Documentation**

4. *Integrated Logic Analyzer LogiCore Product Guide* ([PG172](#))
5. *IBERT for 7 Series GTX Transceivers LogicCORE Product Guide* ([PG132](#))
6. *IBERT for 7 Series GTP Transceivers LogiCore Product Guide* ([PG133](#))
7. *BERT for 7 Series GTH Transceivers LogiCore Product Guide* ([PG152](#))
8. *Virtual Input/Output LogiCore Product Guide* ([PG159](#))

## **Third-Party Documentation**

9. *Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)* ([IEEE-1735-2014](#))

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third-party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

AMBA, AMBA Designer, ARM, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, and MPCore are trademarks of ARM in the EU and other countries.

### **AUTOMOTIVE APPLICATIONS DISCLAIMER**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2012-2016, Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.