Vivado Design Suite User Guide

Designing with IP

UG896 (v2016.3) October 5, 2016

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revisions
10/05/2016	2016.3	Updated for Vivado® Design Suite 2016.3. Changes include: • Added note to managed IP section about using while editing. • Using Revision and Source Control • Changed Title of *LogiCORE IP UltraScale Architecture-Based FPGAs Memory Interface Solutions Product Guide* (PG150), • Changed titles of all LogiCORE documents. • Added content to Tcl Command Example for Copying IP. • Added Tcl command to determine output width and output source of a property. • Fixed all instances of QuickTake Video to match Translation requirements. • Replaced figure for Project Settings: IP General Tab in Chapter 2. • Added text under figure Using the Packager Settings regarding Simulation: • Replaced figure for Project Settings: Packager Tab and added extensive new text. • Clarified Setting the IP Cache. • Changed content in Using the Repository Manager. • Modified information in Tcl Commands for Simulation. • Added a description of the new cache and simulation options in Using IP Project Settings in Chapter 2. • Added text to Generating Output Products. • Added a note on upgrading IP prior to add COE file and where to locate the COE file in Using a COE File. • Added new DocNav boilerplate and automotive statement to Additional Resources and Legal Notices.
06/08/2016	2016.2	Updated for Vivado Design Suite 2016.2. Changes include: • Changed a link from UG949 to UG892.

Date	Version	Revisions
04/12/2016	2016.1	Updated for Vivado Design Suite 2016.1. Changes include: • Added a link to the Vivado Design Suite Tutorial and the UltraFast Methodology Guide (UG949) in Using Revision Control in Chapter 1. • Enhanced the Using Fee-Based Licensed IP: Added content to describe taxonomy in Filtering IP in Chapter 2. • Modified text in Understanding the IP User Files (ip_user_files). • Added references to document and videos in Upgrading IP. • Added a caution to Upgrading IP using a Tcl Command and in the IP Tcl Commands in Order of Design Use table. • Added Setting the Target Clock Period, and Setting a Target Clock Period using Tcl Commands in Chapter 2, and added create_clock. • Added Using a Core Container in Chapter 2. Added Understanding Hierarchical IP in Chapter 2. • Added -log Tcl option to Upgrading IP in Chapter 2. • Updated all figures to reflect the release. • Added more detail to the description in Setting the IP Cache. • Added QuickTake Video links to the following sections: ◦ Using Revision Control ◦ Using Fee-Based Licensed IP ◦ Creating an IP Customization ◦ Using a Core Container ◦ Constraint File Processing Order ◦ Determining Clocking Constraints and Interpreting Clocking Messages. • Added using IP Tcl Commands In Design Flows in Chapter 6. • Updated Editing Subsystem IP in Appendix D. • Added documentation links to Additional Resources and Legal Notices.

Table of Contents

Chapter 1: IP-Centric Design Flow

Introduction 6
IP Terminology 8
IP Packager 9
IP Integrator 9
Using Revision and Source Control 9

Chapter 2: IP Basics

Introduction 10
Using IP Project Settings 11
Using the IP Catalog 19
Creating an IP Customization 25
Generating Output Products 36
Using a Core Container 43
Understanding the IP User Files (ip_user_files) 49
Instantiating an IP 52
Understanding IP States Within a Project 56
Understanding IP Constraints 57
Simulating IP 63
Upgrading IP 68
Understanding Hierarchical IP 71
Working with Debug IP 74

Chapter 3: Using Manage IP Projects

Introduction 77
Managed IP Features 78
Using the Manage IP Flow 78

Chapter 4: Using IP Example Designs

Introduction 82
Opening an Example Design 82
Examining Standalone IP 84

Chapter 5: Using Xilinx IP with Third-Party Synthesis Tools

Introduction . . . 85
Third-Party Synthesis Flow . . . 85

Chapter 6: Tcl Commands for Common IP Operations

Introduction . . . 88
Using IP Tcl Commands In Design Flows . . . 88
Tcl Commands for Common IP Operations . . . 90
Example IP Flow Commands . . . 91

Appendix A: Determining Why IP is Locked

Introduction . . . 96

Appendix B: IP Files and Directory Structure

Introduction . . . 101
IP-Generated Directories and Files . . . 101
Files Associated with IP . . . 103
Using a COE File . . . 103

Appendix C: Using the Platform Board Flow for IP

Introduction . . . 108

Appendix D: Editing or Overriding IP Sources

Introduction . . . 111
Overriding IP Constraints . . . 111
Editing IP Sources . . . 114
Editing Subsystem IP . . . 115

Appendix E: Additional Resources and Legal Notices

Xilinx Resources . . . 116
Solution Centers . . . 116
Documentation Navigator and Design Hubs . . . 116
References . . . 117
Please Read: Important Legal Notices . . . 120

IP-Centric Design Flow

Introduction

The Xilinx® Vivado® Design Suite provides an IP-centric design flow that lets you add IP modules to your design from various design sources. Central to the environment is an extensible IP Catalog that contains Xilinx-delivered *Plug-and-Play* IP. The IP Catalog can be extended by adding the following:

- Modules from System Generator for DSP designs (MATLAB® from Simulink® algorithms)
- Vivado High-Level Synthesis (HLS) designs (C/C++ algorithms)
- Third-party IP
- Designs packaged as IP using the Vivado IP packager

The following figure illustrates the IP-centric design flow.

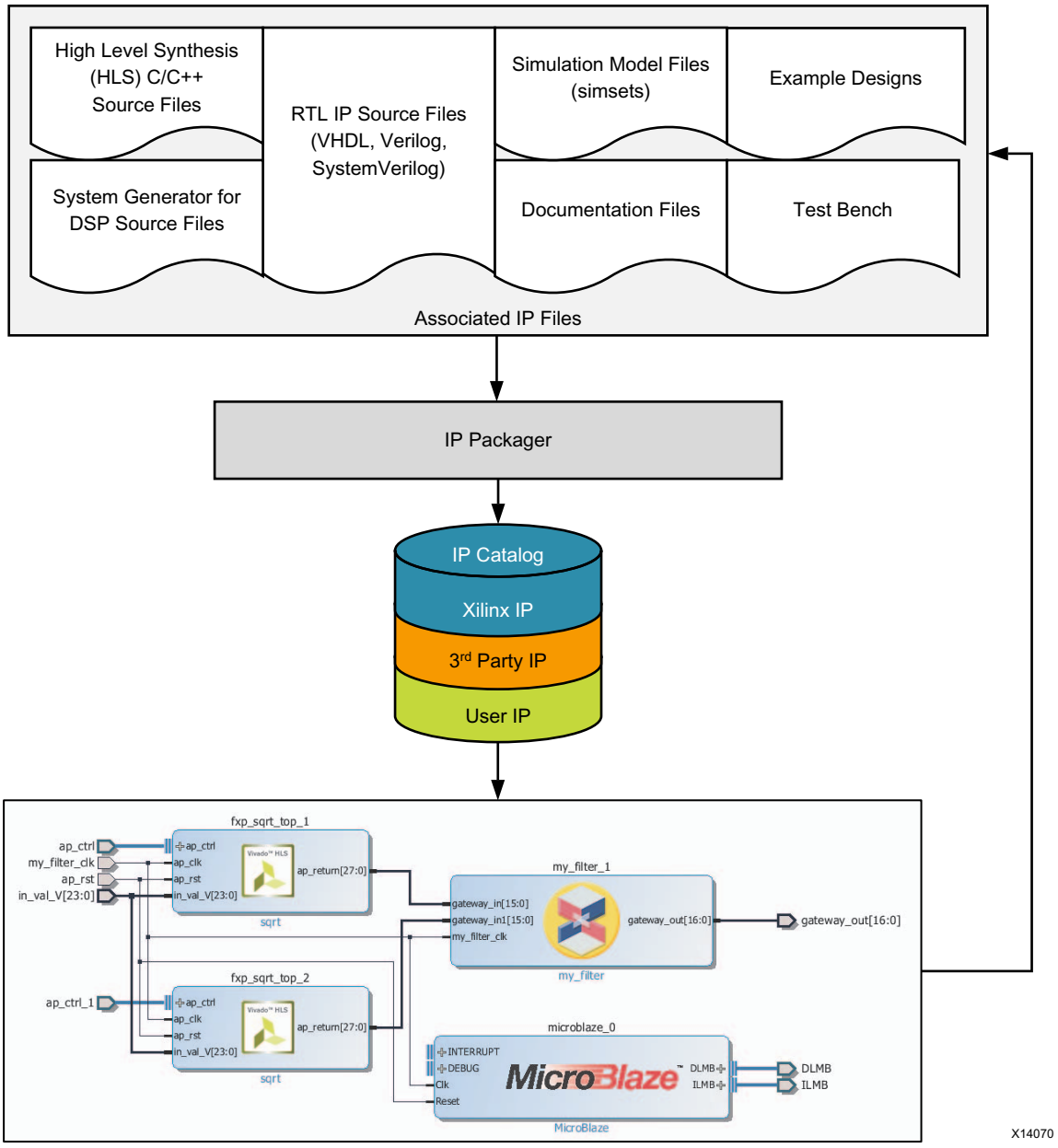


Figure 1-1: **IP-Centric Design Flow**

Note: In some cases, third-party providers offer IP as synthesized EDIF netlists. You can load these files into a Vivado design using the **Add Sources** command.

The available methods to work with IP in a design are as follows:

- Use the Managed IP Flow to customize IP and generate output products, including a synthesized design checkpoint (DCP) to preserve the customization for use in the current and future releases. See Chapter 3, Using Manage IP Projects for more information.

- Use IP in either Project or Non-Project modes by referencing the created Xilinx core instance (XCI) file, which is a recommended method for large projects with many team members.
- Access the IP Catalog from a project to customize and add IP to a design. Store the IP files either local to the project, or save it externally from the project, which is the recommended method for projects with small team sizes.
- Create and customize IP and generate output products in a Non-Project script flow, including generation of a DCP. See this link for more information about Non-Project mode in the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 15].

The *Vivado Design Suite Tutorial: Designing with IP* (UG939) [Ref 19] provides instruction on how to use Xilinx IP in Vivado.

TRAINING: *Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use these links to explore related courses:* Essentials of FPGA Design *and* Embedded Systems Software Design.

IP Terminology

The Vivado IDE uses the following terminology to describe IP, where it is stored, and how it is represented

- **IP Definition**: The description of the IP-XACT characteristics for IP.
- **IP Customization**: Customizing an IP from an IP definition, resulting in an XCI file. The XCI file stores the configuration and constraint options that are user-specified during customization.
- **IP Location**: A directory that contains one or more customized IP referenced in the current project.
- **IP Repository**: A unified view of a collection of IP definitions added to the Xilinx IP Catalog.
- **Output Products:** Generated files produced for an IP customization. They can include HDL, constraints, and simulation targets. During output product generation, the IP customizations stored in the XCI file used to produce the files that are used during synthesis and simulation.
- **IP Catalog:** The IP Catalog allows for the exploration of Xilinx Plug-and-Play IP, as well as other IP-XACT compliant Intellectual Property. This can include designs that you package as IP. See Chapter 2, IP Basics for more information.

IP Packager

The Vivado IP packager lets you create plug-and-play IP to add to the extensible Vivado IP Catalog. The IP packager wizard, is based on the IEEE Standard for IP-XACT (IEEE Std 1685), *Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows* [Ref 40].

After you have assembled a Vivado Design Suite user design, the IP packager lets you turn your design into a reusable IP module that you can then add to the Vivado IP Catalog, and that others can use for design work. You can use packaged IP within a Project or Non-Project-based design. See the following documents for more information:

- *Vivado Design Suite: Creating and Packaging Custom IP* (UG1118) [Ref 4] for more information about using the packaging feature.
- *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* (UG1119) [Ref 5] provides labs with design solutions that show you how to use the packaging feature.

TIP: *You can locate the IP within the project or use a remote location.*

IP Integrator

The Vivado Design Suite IP integrator tool lets you create complex subsystem designs by instantiating and interconnecting IP cores from the Vivado IP Catalog onto a design canvas. For more information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 6].

Using Revision and Source Control

The Vivado Design Suite is designed to work with any revision control system. For IP designs there are trade-offs to that you should consider when using revision control systems to manage design sources. These trade-offs affect run-time versus the number of files being managed. For information on how to use Vivado Design Suite with version and source control systems, see this link in the *Vivado Design Suite User Guide: Design Flows Overview* [Ref 15].

See the *Vivado Design Suite Tutorial: Revision Control* (UG1198) [Ref 27] for recommendations of revision control.

VIDEO: *See the Vivado Design Suite QuickTake Video: Vivado Design Suite: Revision Control for more information.*

IP Basics

Introduction

This chapter describes the following features of the designing with IP, as follows:

- Using IP Project Settings
- Using the IP Catalog
- Creating an IP Customization
- Generating Output Products
- Using a Core Container
- Understanding the IP User Files (ip_user_files)
- Instantiating an IP
- Understanding IP States Within a Project
- Understanding IP Constraints
- Simulating IP
- Upgrading IP
- Understanding Hierarchical IP
- Working with Debug IP

Working with Xilinx® IP consists of first customizing an IP for use in an RTL design. You can create an IP customization in various ways using the Vivado® Design Suite, as follows:

- Directly customizing an IP into a project from the IP Catalog.
- Using the Manage IP project flow to create a stand-alone customization of an IP for use in the current project as well as others. See Chapter 3, Using Manage IP Projects for more information.
- Using a Tcl script to create an IP customization in either Project or Non-Project mode.

After creating a customization, you can automatically generate output products, or defer generation until later.

- In Project mode, if the output products are not present, they are generated automatically prior to synthesis or simulation.
- In Non-Project mode, you must generate the output products manually prior to synthesis or simulation.

To use an IP customization in a design you must instantiate the IP in the HDL code or your top-level design. One of the IP output products is an instantiation template based upon the target language set in the project settings. See Using IP Project Settings for more information.

VIDEO: *See the Vivado Design Suite QuickTake Video: Getting Started with Vivado IDE for move information on using Vivado IDE.*

Using IP Project Settings

When working with IP in a Manage IP project or in an RTL project, you can configure IP-specific project settings using the IP category in the Project Settings dialog box. The following tabs are available:

- **General**: Lets you specify the use of Core Containers, automatic generation of simulation scripts, upgrade log creation, setting the default location for IP output products, and turning on IP caching.
- **Repository Manager**: Adds IP repositories and specifies the IP to include in the IP catalog.
- **Packager**: Sets the default behavior used by the IP packager when packaging IP. See *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 4] for more information.

Note: The IP Project Settings and the Vivado IP Catalog are available when working with an RTL project or when using Manage IP from the Getting Started page. When using Manage IP, only a subset of the IP settings is available unless a project is created.

General IP Settings

The General tab for IP settings let you specify various project specific options for IP, as shown in the following figure. For each project you create, if you wish to change the defaults you must the project settings again.

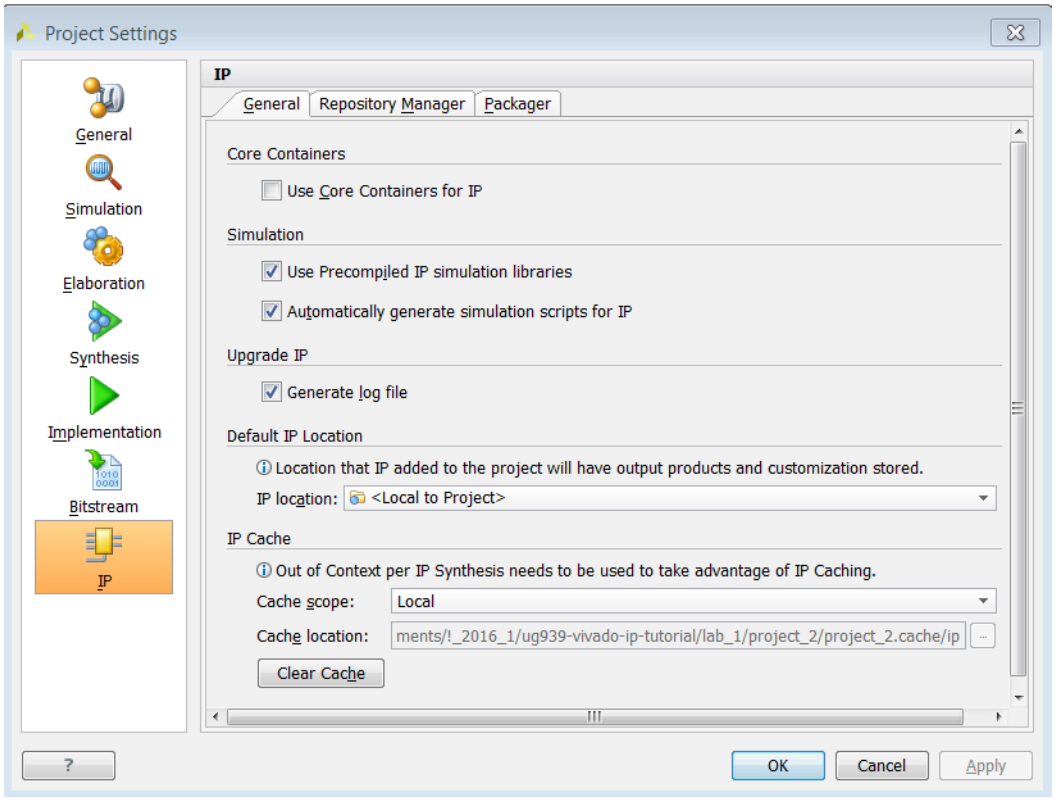


Figure 2-1: **Project Settings: IP General Tab**

The IP General tab contains the following:

- **Core Containers:** Check the **Use Core Containers for IP** to use the core container feature, which optionally lets you have the IP and all generated output files contained in one compressed binary file with an extension of XCIX. See Using a Core Container for more information.
- **Simulation:**
  - By default, the **Use Precompiled IP simulation libraries** option is checked. Vivado delivers precompiled libraries for all the static files for Xilinx IP for use with the `Xsim` simulator.

  When simulation scripts are created, they reference these precompiled libraries. If you are using a third-party simulator you must create these libraries as explained in *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 17].

- By default, the **Automatically Generate Simulation Scripts for IP** is checked. This results in having the simulation scripts generated automatically for each IP. The scripts are placed in the `<project name>.ip_user_files` directory. See Simulating IP for more information. Uncheck to disable simulation scripts from being created.

- **Upgrade IP:** By default, the **Generate log file** under Upgrade IP is checked. This results in an `ip_upgrade.log` file to be created when you upgrade IP. This file contains all the upgrade logs for all IP.

  As you upgrade additional IP they are added to the top of the log file. The log file is stored in the project directory at the root location (where the project XPR file is placed). See Upgrading IP for more information, including how to specify the name and location of the log file using Tcl commands. To disable the log file being created, uncheck the **Upgrade IP** checkbox.

- **Default IP Location**: You can use this to set the location where all IP created and their output products will get stored. By default, IP in an RTL project are stored within the project directory structure in the `.src/sources_1/ip` directory. When working with revision control systems it is recommend that IP be stored outside of the project as with other source files. When customizing an IP you can use the **IP Location** to set the location where the IP and its output products are stored.
- **IP Cache**: Options include:
  - **Cache scope**: Options are **disabled**, **local**, or **remote**.
  - **Cache location:** Browse to, and select the location for cache.
  - **Clear Cache**: Button to delete the cache files from the disk by issuing the following command on the Tcl Console:

    ```
    config_ip_cache -clear_output_repo.
    ```

For an expanded description of the IP Cache, see Setting the IP Cache.

Using the Repository Manager

The following figure shows the Repository Manager.

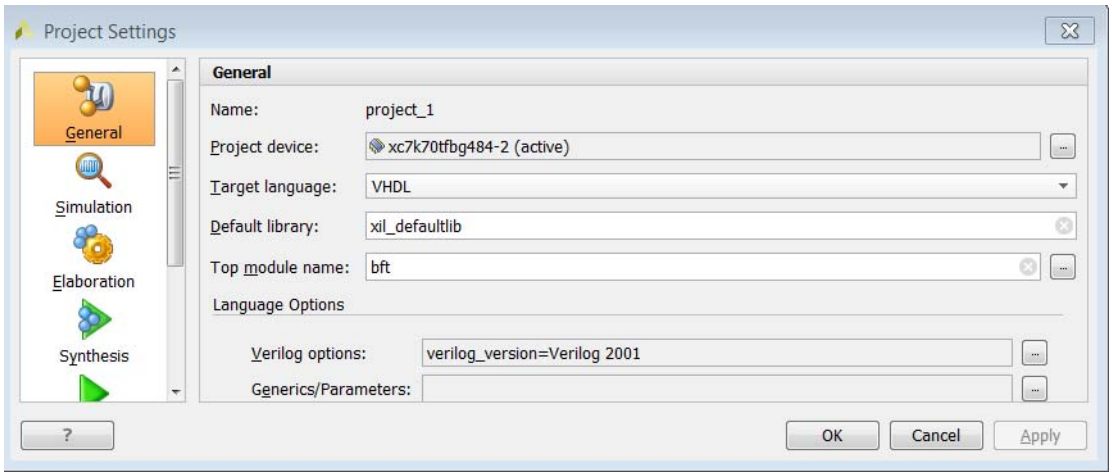


Figure 2-2: **IP Project Settings: Repository Manager**

To change settings in the Repository Manager, do the following:

1. If the Repository Manager is not already open, select **Tools > Project Settings**, and then click **IP**.
2. Click the **Repository Manager** tab, and do the following to manage locations for IP (either custom packaged or third-party):
   - Click the **Add** button to specify the location that contains IP definitions to add. The Repository Manager hierarchically searches within the specified path for IP definitions. The IP is listed after the directory is specified, as shown in the following figure.

     IP that are added that support the current project show as Yellow. Unsupported IP are grey.

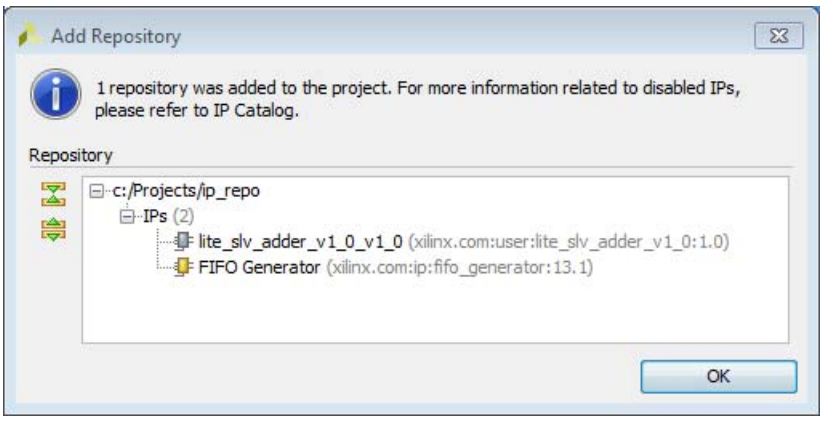


Figure 2-3: **Color Coding of Supported and Unsupported IP**

   - Use the **Remove** button to remove a repository listing.

     The IP catalog shows that the repository is no longer present.
3. To update the contents of the IP Catalog with the IP within each repository, click **Apply** or **OK**.

TIP: *As an alternative to using the Repository Manager of the IP Project Settings dialog box, you can also use the right-click menu in the Vivado IP Catalog, and add the repository directly to the IP Catalog.*

Resolving Duplicate IP

IMPORTANT: *It is possible to create duplicate IP.*

If you enter an IP that has the same name as an existing IP, the Repository Manager issues a warning banner. The actions you can take are, as follows:

- Review the duplicated IP in the IP Catalog.
- Review the directories in the description.
- If necessary, remove the IP or remove the repository that contains the IP.

Using the Packager Settings

The following figure shows the Packager tab, in which you set the Packager settings.

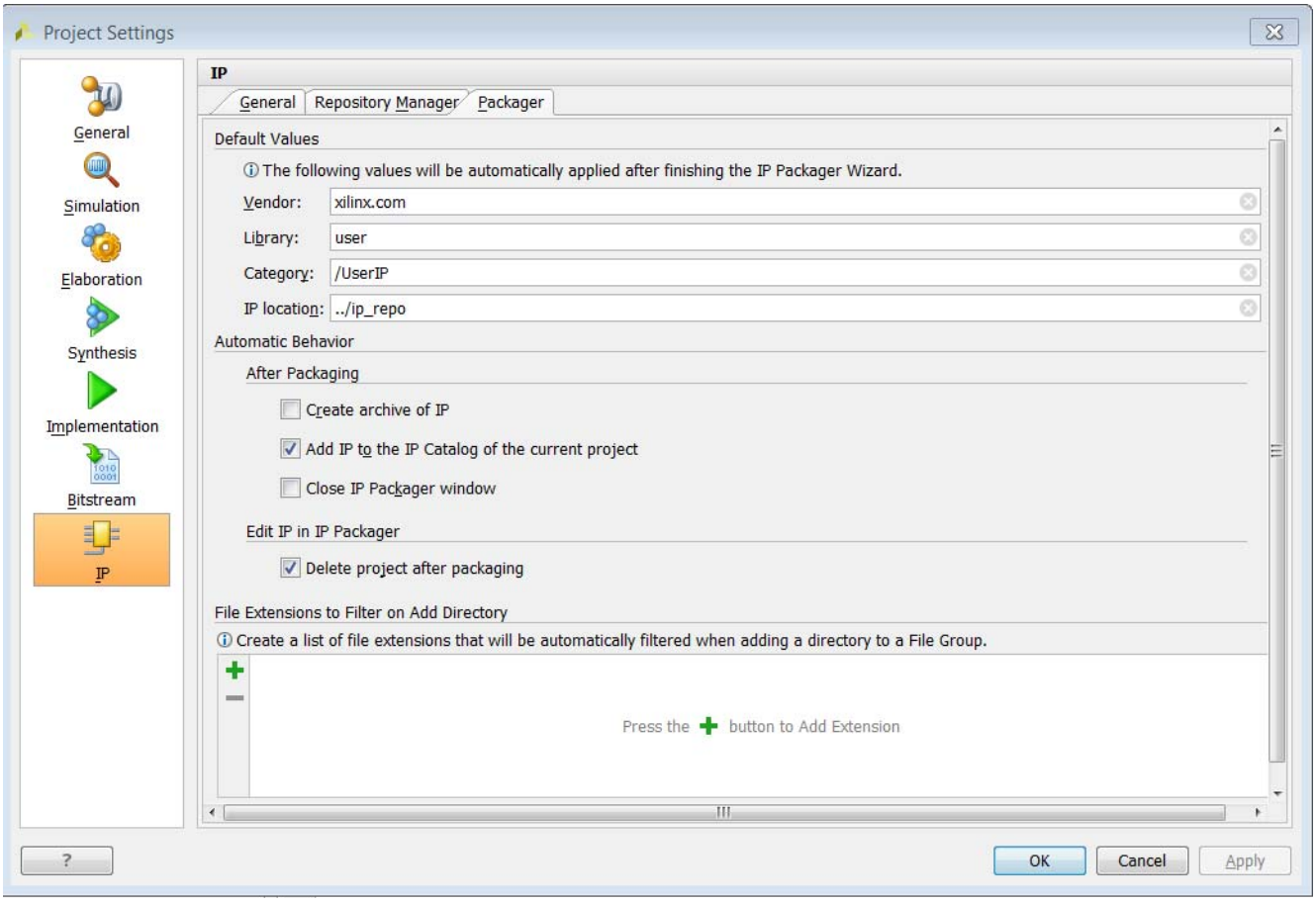


Figure 2-4: **Project Settings: Packager Tab**

See *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 4] for more information about packaging IP.

Setting the IP Cache

To speed up generation of the synthesis output products for an IP using the default out-of-context (OOC) flow, the IP Cache option is enabled by default.

With the cache enabled, whenever you generate an IP using out-of-context for synthesis, the Vivado tool creates synthesis output products (such as DC and stub files), the tool also creates a cache entry.

A cache entry consists of two directories on disk:

- <cache ID>: Contains the XCI, DCP, sim_netlist, and stub files.
- <cache ID>.logs: Contains the synthesis log (`runme.log`).

After the cache is populated, when a new customization of the IP is created which has the exact same properties, the IP are not synthesized again during generation. Instead, the cache is referenced and the synthesis output is copied. Because the IP is not synthesized, generation of the output products is completed more quickly.

The options for the Cache Scope are:

- **Disabled**: IP caching is not done.
- **Local**: (default) The cached IP is stored and referenced from within the project directory structure in `<project_name>.cache/ip`. This option lets you create a synthesis cache for the IP in the current project, separate from the IP in other projects.
- **Remote**: The cached IP is stored and referenced from the specified location. This lets you create and reference a synthesis cache that is shared across multiple projects.

You can also select the location of the cache with the Cache Location option to browse to, and select a directory.

When you generate an IP with the cache enabled, the Vivado tool creates a design run as normal if there is no cache hit. However, if there is a cache hit, then the synthesis results (DCP and stub files) are copied from the cache directory to the IP directory and renamed. No design run is created, though an entry is added which reports the cache hit (see the following figure).

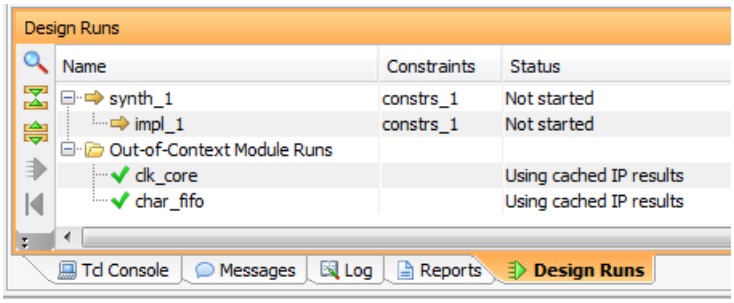


Figure 2-5: **Design Runs**

After generation, a dialog informs you if a cache hit has occurred, as shown in the following figure.

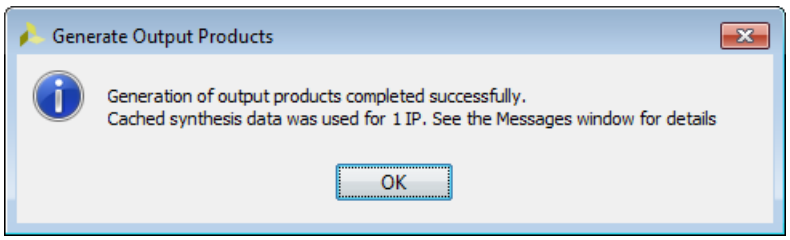


Figure 2-6: **Generate Output Products Dialog Box**

The following is an example of the `INFO` message produced in the Tcl Console and stored in the Vivado.log:

```
INFO: [IP_Flow 19-4993] Using cached IP synthesis design for IP fifo_0, cache-ID =
aa71c47ae9ccd380; cache size = 0.383 MB.
```

This shows you the cache ID used as well as the current size of the IP Cache.

For a cache hit to occur, the IP must be customized identically *and* have the same part and language settings. After you generate the IP, that IP does not reference the cache location. All output products for an IP are stored local to the IP XCI file. The cache is only referenced during the generation of the IP output products.

CAUTION! *IP Cache can grow large, depending on the number of IP present.*

To manage the IP Cache, use the `config_ip_cache` Tcl command. Using this command, you can list the contents of the cache, and the size of the cache in KB.

Configuring IP Cache for all New Projects

By default, all new projects have the IP Cache enabled and configured to be local to the project. It can be advantageous to have multiple people/groups point to the same cache location. This can reduce the use of disk space because each user does not have an individual cache. It also increases the likelihood of a cache hit.

You can use the parameter `project.defaultIPCacheSetting<option>` in the `init.tcl` file to configure all new created projects for the following purposes:

- Disable the use of the IP Cache.
- Specify a remote location for the IP Cache.
- Enable the local use of IP Cache (default)

To use the `project.defaultIPCacheSetting<option>` parameter, place this line in the `init.tcl` file:

```
set_param project.defaultIPCacheSetting <option>
```

Where `<option>` is:

- `none`: Disable the IP Cache.
- `<remote location>`: Path to the remote location.
- `local`: default. Place IP Cache in the project directory (`<project name>.cache/ip`).

The location of the `init.tcl` file is in one of the following locations:

- Linux: The users home directory in `.Xilinx/Vivado/`
- Windows: `C:/users/<username>/AppData/Roaming/Xilinx/Vivado`

Setting Cache in init.tcl Examples

The following is an example on how to disable the IP Cache for all newly created project:

```
set_param project.defaultIPCacheSetting none
```

The following is an example on setting the IP Cache to be in a remote location for all newly created projects:

- Linux: `set_param project.defaultIPCacheSetting /wrk/staff/smith/ip_cache/`
- Windows: `set_param project.defaultIPCacheSetting c:/project/ip_cache/`

RECOMMENDED: A parent directory is not created. IP Cache directories are placed at the location specified. If you set the remote location somewhere where there are a many other sub-directories it slows the project creation as the directories are scanned for cache entries.
It is recommended you create a directory specifically for the IP Cache and point to that. The IP Cache entries will be created under that directory.

Using the IP Catalog

The key features of the Vivado IP Catalog include:

- Consistent, easy access to Xilinx IP, including building blocks, wizards, connectivity, DSP, embedded, AXI infrastructure, and video IP from a single common repository regardless of the end application being developed.
- Support for multiple physical locations in an IP repository, including shared network drives, allowing users or organizations to leverage a consistent IP deployment environment for third-party or internally-developed IP.
- Access to the Xilinx-delivered IP, which is rigorously tested prior to inclusion in the IP Catalog.
- Access to IP customization and generation using the Vivado IDE or an automated script-based flow using Tcl.
- On-demand delivery of IP output products such as instantiation templates and simulation models (HDL, C, or MATLAB® software).
- IP example designs that provide capability to evaluate IP directly as an instantiated source in a Vivado Design Suite project.
- Access to version history details as recorded in the Change Log.
- A `<major#.minor#.Rev#>` numbering scheme unifies the IP version numbers. For information, see the Xilinx IP Versioning page available from the Xilinx website for *Vivado IP Versioning* [Ref 29].
    - The recommended response to the disposition of an IP version is as follows:
        - `Major#`: You need to make a change.
        - `Minor#`: You might need to make a change.
        - `Rev#`: No need to make a change.
- Catalog filter options that let you filter by Supported Output Products, Supported Interfaces, Licensing, Provider, and Status.
- Group IP by Taxonomy or Repository.

For information on IP that supports the Vivado Design Suite, see the Xilinx website for *IP Documentation* [Ref 30].

For information on specific IP, see the *IP Center* [Ref 31] or look at the IP Catalog.

The following figure shows the default Vivado IP Catalog view that lists the available categories (Cores) of IP.

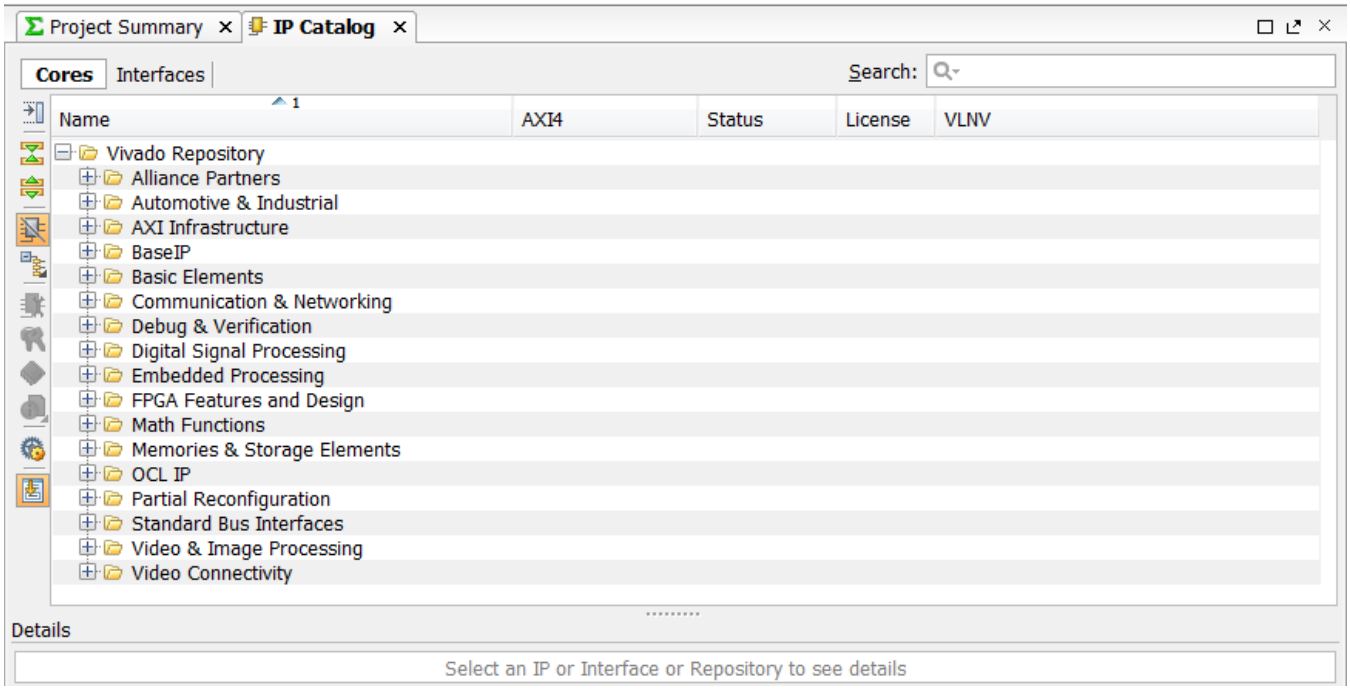


Figure 2-7: **Xilinx IP Catalog - Cores**

There are two views in the IP Catalog. They contain the following:

- **Cores**: IP provided by Xilinx, Alliance Partners, and Customer repositories as shown in the previous figure.
- **Interfaces**: A list of available interfaces, shown in the following figure.

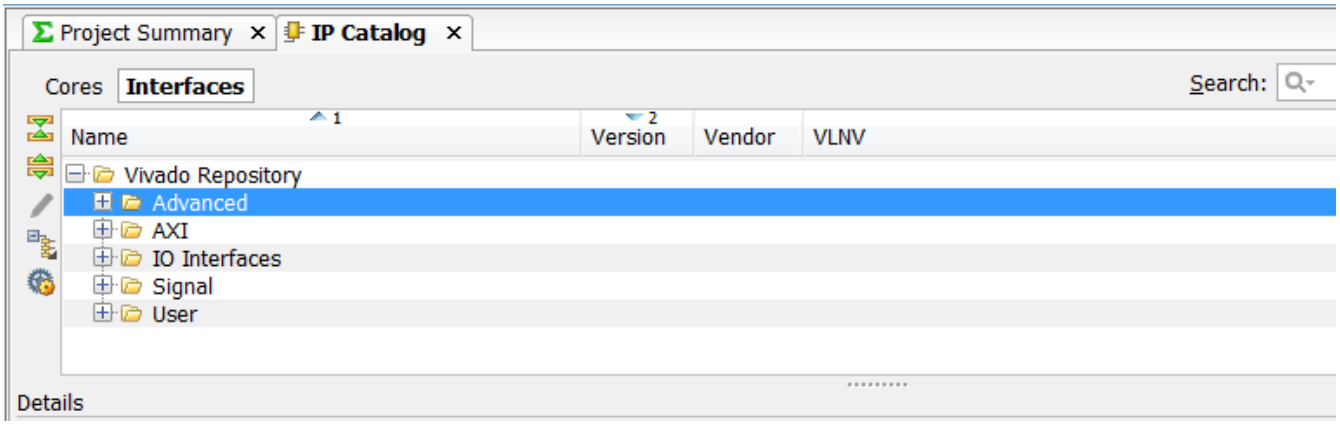


Figure 2-8: **Xilinx IP Catalog - Interfaces**

The Interfaces are categorized by how one would store contents in the Vivado Repository when packaging an IP.

Major interfaces in the Vivado Repository are as follows:

- **Advanced**
- **AXI**
- **IO Interfaces**
- **Signal**
- **User**

Filtering IP

The IP Catalog contains categories of IP that you can filter and search.

The following figure shows the IP Catalog filter options that let you filter the IP Catalog by categories. Select the **Options** button in the top-left corner to open the filter options.

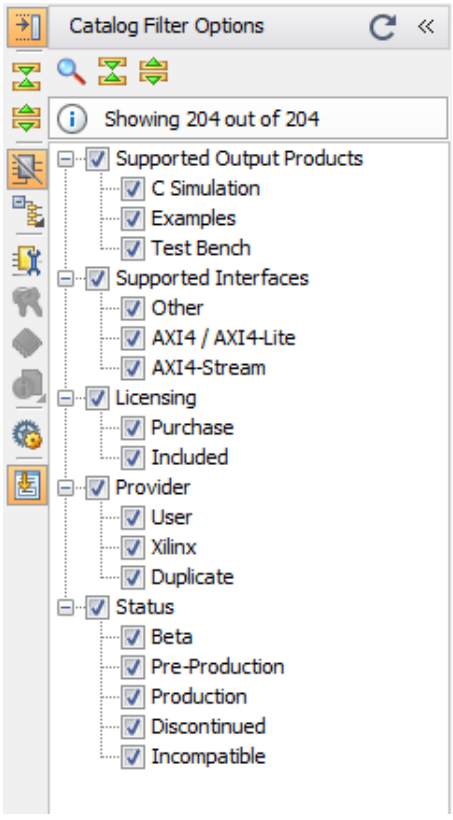


Figure 2-9: **IP Catalog Filter**

Uncheck the filter options to filter out IP that meets unwanted criteria.

When searching in the IP Catalog, an IP is shown in the taxonomy structure, even it is not a complete match for your search pattern, if they are in a directory or folder that matches the search pattern. The following figure shows the results of a search on "filter". All IP which have "filter" in their name are shown, as well as IP whose folder contains the word "filter".

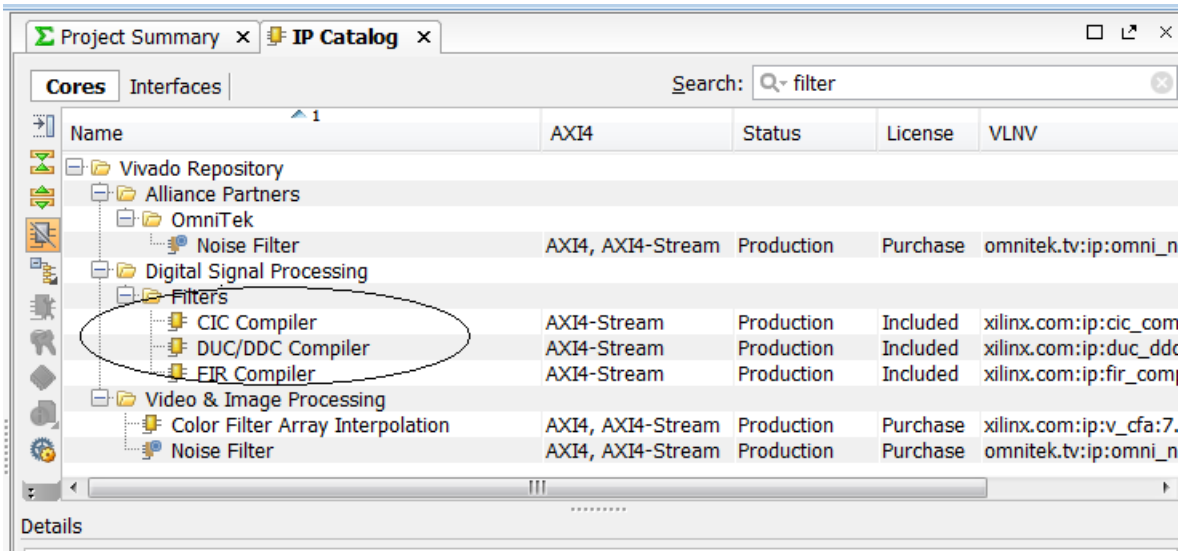
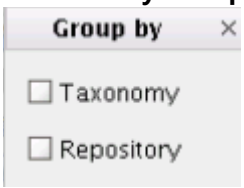


Figure 2-10: **Filtered Search on Word Filter**

IP Catalog options give you the ability to expand, collapse, and search the IP Catalog content. These icons are consistent with other windows in the Vivado IDE.

Other IP Catalog options to filter or view IP information are:

Option	Description
	Hide Incompatible IP
	Group by Category. Click this button and the **Group by** selection option opens, where you can choose **Taxonomy** or **Repository** or both.  Group by repository lets you group IP according to Xilinx IP or user-IP; for instance, and grouping by Taxonomy lets you group IP by function or category.
	Customize IP. Open the Customize IP dialog box for the most recently-selected IP. See Creating an IP Customization.

Option	Description
	License Status. See Using Fee-Based Licensed IP for more information.
	Show Compatible Families. Displays the device families that are compatible with the currently-selected IP.
	View Product Guide, Change Log, Product Webpage, and Answer Records.
	Settings for IP Catalog, IP generation, and Repository Manager and IP Packager. This command opens the Project Settings dialog box for the current project and displays the IP settings. See Using IP Project Settings for more information.
	Auto-scroll to selected items.

Partner Alliance IP

The Vivado IP Catalog includes IP available for purchase from select Alliance Partners. These IP are signified by the blue color disk on the icon, as shown in the following figure.

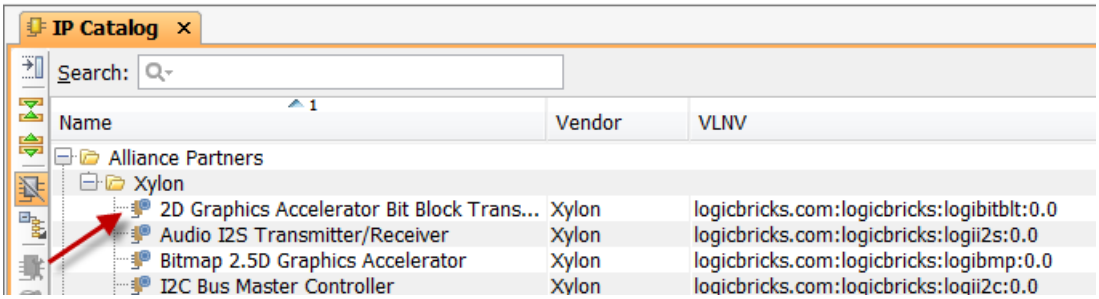


Figure 2-11: **Alliance Partner IP**

When you select an Alliance Partner IP, a dialog box opens that provides you with a link to where you can purchase the IP, as shown in the following figure. The option to Customize IP is grayed-out until you purchase and install the IP.

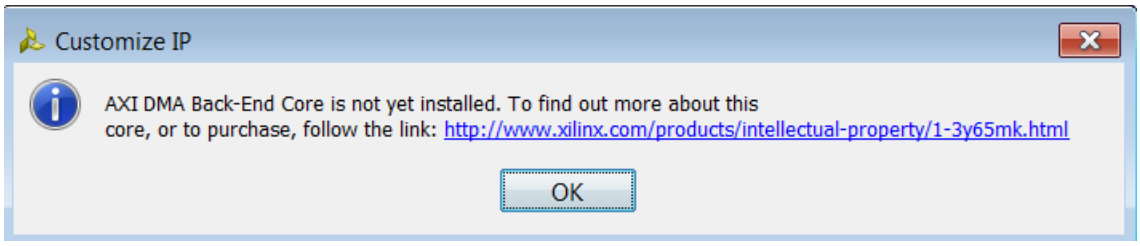


Figure 2-12: **Partner Alliance IP Dialog Box**

Using Fee-Based Licensed IP

The Vivado IP catalog displays either **Included** or **Purchase** under the License column in the IP catalog, and also provides a field on the status of the license. The following definitions apply to IP offered by Xilinx:

- **License Status**: IP licenses can be **Full** (also know as Purchased), **Simulation**, or **Eval**.
  - **Included**: The Xilinx *End User License Agreement* [Ref 1] applies to Xilinx LogiCORE™ IP cores that are licensed within the Xilinx Vivado Design Suite software tools at no additional charge.
  - **Purchase**: The *Core License Agreement* [Ref 2] applies to fee-based Xilinx LogiCORE IP, and the *Core Evaluation License Agreement* [Ref 3] applies to the evaluation of fee-based Xilinx LogiCORE IP.
- **License Type**: License types can be **Floating** or **Node-Locked**.
  - Certificate-based Network *Floating* Licenses and activation-based Server Licenses are locked to a license server host running the FLEX license server daemon. A license is checked out per unique user.
  - *Node-Locked* or Client license is a license that is locked to a specific machine or, for certificate based-licenses, a dongle. As long as you do not replace your hard drive, the Disk Serial Number (Volume ID) is reliable to identify the Node.
- Other license levels include the following: **Design_Linking**, **Hardware_Evaluation**, and **Full** (Purchased).

For more information on how to obtain IP licenses, see the Xilinx *IP Licensing* site [Ref 1].

See this link for more information on licensing that is provided in the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) [Ref 14].

VIDEO: *See the following:* Vivado QuickTake Video: Vivado Activation and Floating License Generation *and* Vivado QuickTake Video: Vivado Licensing and Activation Overview

For fee-based IP, the **OK** button on the Customize IP dialog box is disabled until an evaluation or a paid license is found, as shown in the following figure.

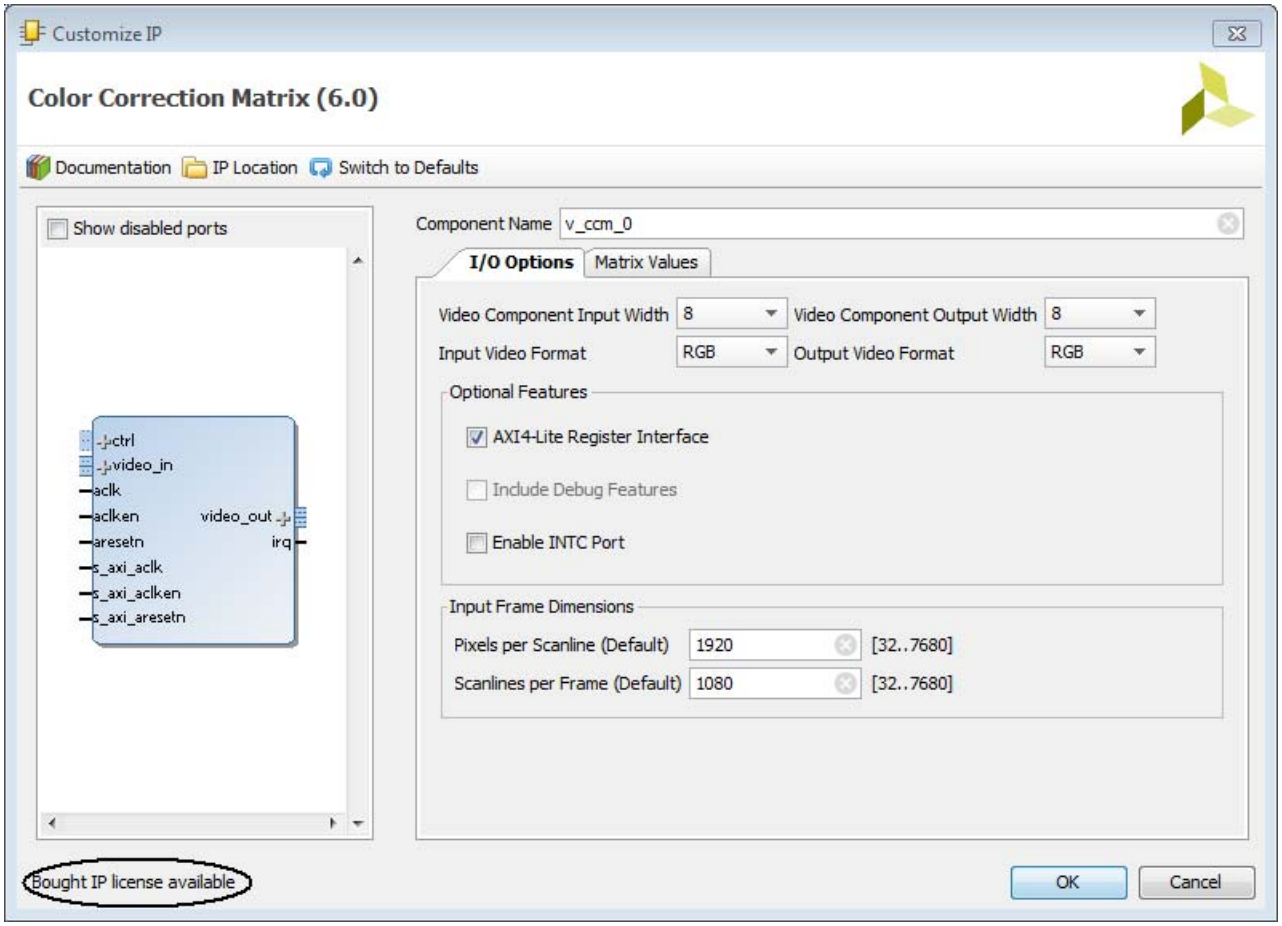


Figure 2-13: **Fee-Based IP "OK" Button Enabled after License Found**

IMPORTANT: *During implementation, the Vivado tools compare both the current License status (level) and the license level of the generated IP core netlist in a design, then passes the lowest level forward; consequently, you must regenerate the LogiCORE IP core for the core netlist to receive the current license status.*

Creating an IP Customization

TIP: *When entering or modifying data in a text box, if a value is used and editable, the text is black and the background is white. If a value is used but not editable, the text is black and the background is grey. If a value is unused or not applicable, the text is gray, including the label that precedes or follows it.*

VIDEO: *See the following for more information:* Vivado QuickTake Video: Customizing and Instantiating IP.

To create an IP customization using the IP Catalog:

1. From the IP Catalog, select the IP using one of the following methods:
   - Double-click the IP.
   - From the toolbar, select the **Customize IP** command or right-click and select the command.

   The Customize IP dialog box shows the various parameters available for you to customize the IP. This interface varies, depending on the IP you select, and can include one or more tabs in which to enter values.

   The Customize IP dialog box includes the following:

   - An IP symbol
   - Tabs for setting configuration options for the specific IP. The Vivado IDE writes these configuration options to the `<ip_name>.xci` file, and stores them as properties on the IP object.

CAUTION! *The Windows operating system has a total 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs. Keep this in mind when storing IP outside of a project.*

The IP symbol supports zoom, re-size, and auto-fit options that are consistent with the schematic viewer canvas in Vivado IDE.

The following figure shows the Customize IP interface for the FIFO Generator IP.

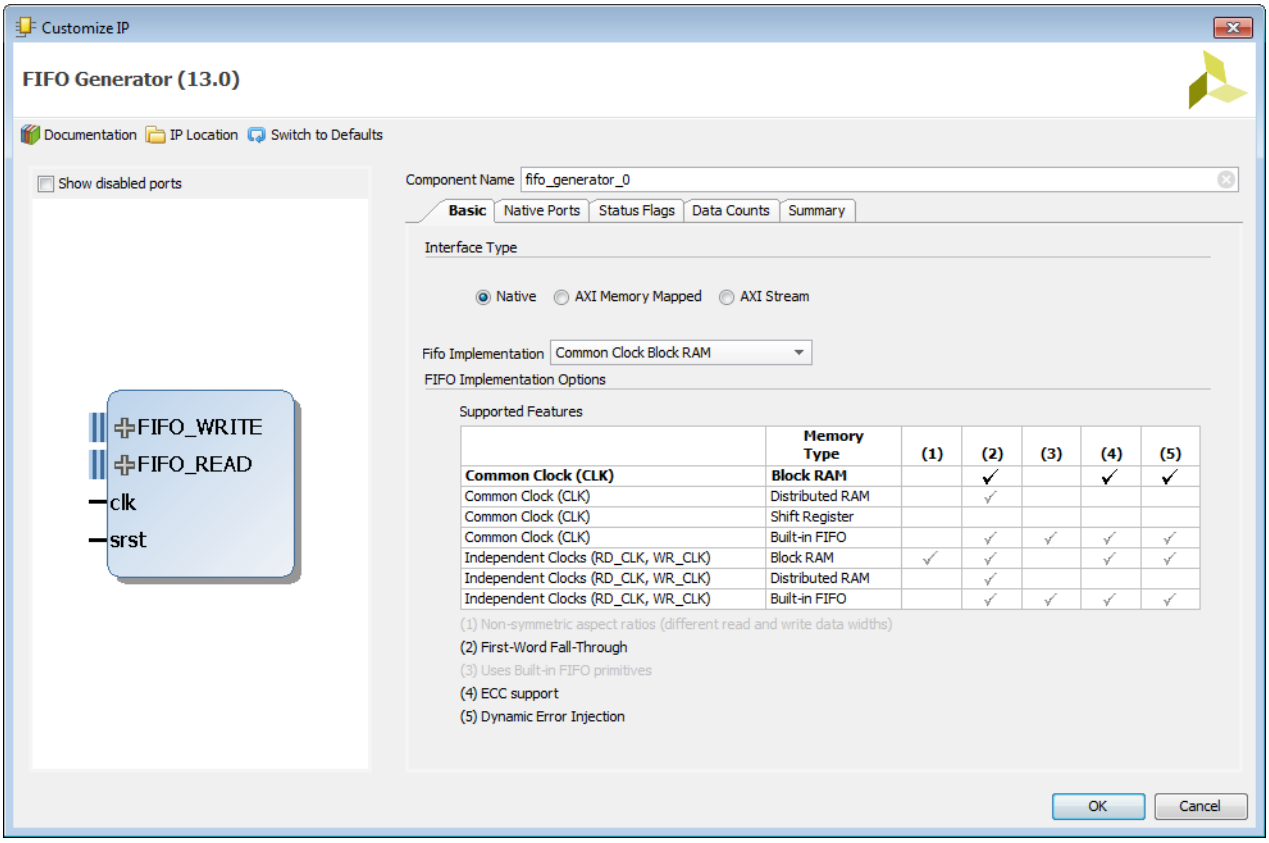


Figure 2-14: **FIFO Generator Dialog Box**

2. In the IP dialog box, set the following options:
   - **Documentation > Product Guide** to open the product guide for the selected IP.
   - **IP Location** to specify the location on disk to store the IP. You can only change this location when using the IP Catalog in an RTL project. This is *not an option* in a Manage IP project.
   - **Switch to Defaults** to display a window asking if you want to reset all configuration options back to their default starting point.
3. Customize the IP as needed for your design, and click **OK**.

RECOMMENDED: *Xilinx recommends that when specifying a numerical value, use hexadecimal to speed processing.*

VIDEO: *See the Vivado Design Suite QuickTake Video: Customizing and Instantiating IP for a demonstration of the IP Customization and Instantiation process.*

Tcl Command Example for Creating an IP Customization

Note: Where there is a blue link, you can go directly to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 7] for more information about the command. You can also use `<command_name> -help` in the Vivado IDE.

You can also create IP customizations using the `create_ip` Tcl command. For example:

```
create_ip -name fifo_generator -version 12.0 -vendor xilinx.com -library ip
-module_name fifo_gen
```

You must specify either `-vlnv` or all of `-vendor`, `-library`, `-name`, and `-version`.

Note: Executing the `create_ip` Tcl command creates the IP customization file (XCI), which is the configuration for the IP, as well as the instantiation template and BOM (XML) file, but does not create any other output products. The default configuration for the IP is set with the `create_ip` command.

Tcl Command Example for Setting IP Properties

To define the different configuration settings of the IP, use the `set_property` command. For example:

```
set_property CONFIG.Input_Data_Width 12 [get_ips fifo_gen]
```

Tcl Command Example for Reporting IP Properties

To get a list of properties available for an IP use the `report_property` command. For example:

```
report_property CONFIG.* [get_ips <ip_name>]
```

Configuration properties start with `CONFIG`.

Tcl Commands for Querying an IP Customization Property

To determine if an IP customization property is set to the default or set by the user, see the following example:

```
# Find the read data count width.
get_property CONFIG.Read_Data_Count_Width [get_ips char_fifo]
10
# Determine the source of CONFIG.Read_Data_Count_Width property.
# We see that this is the default value
get_property CONFIG.Read_Data_Count_Width.value_src [get_ips char_fifo]
default

# Get the output data width.
get_property CONFIG.Output_Data_Width [get_ips char_fifo]
8
# Determine the source of CONFIG.Output_Data_Width property.
# We see that this is set by the user.
get_property CONFIG.Output_Data_Width.value_src [get_ips char_fifo]
user
```

Setting the Target Clock Period

By default, IP are synthesized standalone, and out-of-context from the rest of a design.

- During implementation, the IP netlist is linked with other IP netlists and the user netlist.
- During synthesis of the user logic, the IP is seen as a black box.

Because the IP is synthesized separate from the top-level, the Vivado Tools create the `_ooc.xdc` to provide clock definitions for the IP, as explained in Synthesis Options for IP.

If you do not specify a target period, the IP uses a default clock period. This might result in a warning when the period used for the IP standalone differs from the period seen when synthesizing the top-level, as follows:

```
[Timing 38-316] Clock period '10.000' specified during out-of-context synthesis of instance 'char_fifo_i0' at clock pin 'rd_clk' is different from the actual clock period '6.000', this can result in different synthesis results.
```

The warning informs you that if global synthesis had been used, the IP would be synthesized using a different clock period than was used when synthesizing the IP out-of-context.

Some IP have options in the customization GUI to set the target frequency/period to be used during out-of-context synthesis. Some of the IP have a tab labeled **Clocks**, which allows the setting of the target frequency for the IP. Other IP have the setting of the target mixed in with other settings.

Typically, there is an information popup to explain the setting as shown in the following figure.

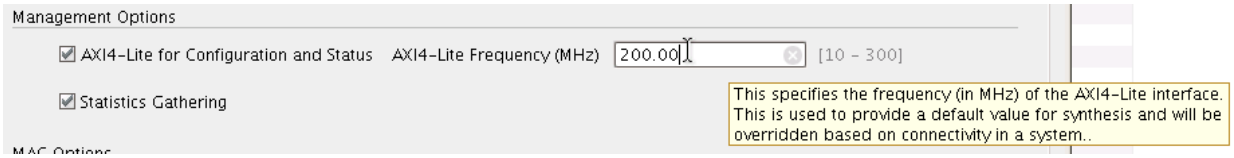


Figure 2-15: **Setting Target Frequency used During Out-of-Context Synthesis of an IP**

Some IP do not provide a GUI option to customize the target clock frequency/period. When the GUI option is not available, use the Tcl Console to query and set the configuration option for the IP.

A clock port of an IP has a property associated with it ending with `FREQ_HZ`. Changing these properties will result in the `_ooc.xdc` file for the IP to use these values when output products are generated for the IP.

Setting a Target Clock Period using Tcl Commands

All IP customizations are performed by setting properties on the IP object.

- When using the GUI to configure an IP, the Vivado IDE issues the Tcl commands automatically
- If an IP does not provide a GUI method for setting the target period for an IP in out-of-context synthesis, you can set it manually.

Changing an IP customization is permanent unless changed again using Tcl (or the IP customization GUI if applicable), so if you reset and regenerate the IP your changes are persistent.

The following is an example of how to set the target clock for the FIFO generator IP called `char_fifo`. The IP was customized to use a common clock for the read and write ports and the native interface.

1. Report the properties available for the IP using the `report_property` command. Type the following command at the Tcl Console:

   ```
   report_property [get_ips char_fifo]
   ```

2. From the output, you see there are five properties that end in `FREQ_HZ` for this IP:
   - `CONFIG.core_clk.FREQ_HZ`: Applicable when using a common clock (our example)
   - `CONFIG.read_clk.FREQ_HZ`: Applicable when using independent clocks
   - `CONFIG.write_clk.FREQ_HZ`: Applicable when using independent clocks
   - `CONFIG.slave_aclk.FREQ_HZ`: Applicable when using AXI
   - `CONFIG.master_aclk.FREQ_HZ`: Applicable when using AXI

   Only the first one is applicable for the native interface with a common clock. The `CONFIG.core_clk.FREQ_HZ` is by default set to 100000000 or 100MHz.

   If the IP was generated already, you could look at the `char_fifo_ooc.xdc` file, and see the following line:

   ```
   create_clock -period 10 -name clk [get_ports clk]
   ```

   The period of 10 corresponds to the 100MHz value of the `CONFIG.core_clk.FREQ_HZ`.

For this example, with a desired clock frequency of 250MHZ, set as follows:

3. Type the following Tcl command in the Tcl Console:

```
set_property CONFIG.core_clk.FREQ_HZ 250000000 [get_ips char_fifo]
```

4. After you set the property, generate the IP.

   This causes the out-of-context run (if present) to be reset and rerun.

   After the run finishes, look at the `char_fifo_ooc.xdc` file. You see:

```
create_clock -period 4 -name clk [get_ports clk]
```

Now the desired clock period/frequency is used when synthesizing the IP.

Adding Existing IP to a Project

You can also add IP into your project that was previously created in the CORE Generator™ tool (`<ip_name>.xco` files) or Vivado IP (`<ip_name>.xci` files) by using the **Add Existing IP** option in the Add Sources dialog box, as shown in the following figure.

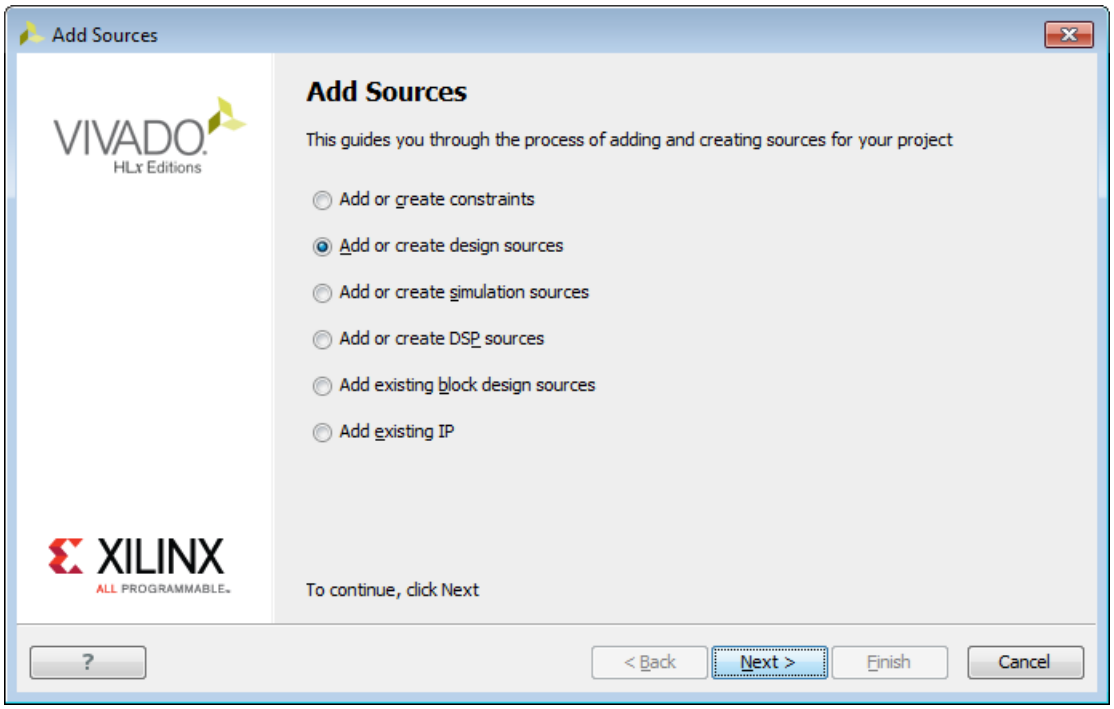


Figure 2-16: **Add Sources Dialog Box**

You can either reference the IP and any generated output products from its current location, or copy the IP and any generated output products into your project.

Existing IP can be IP customized for use in another design, or customized for use in many designs using a Managed IP project. A Managed IP project can create the XCI file for the IP customization, as well as generate any needed output products. See Chapter 3, Using Manage IP Projects for details on creating IP using the Managed IP flow.

After you click **Next**, the Add Existing IP dialog box opens, as shown in the following figure.

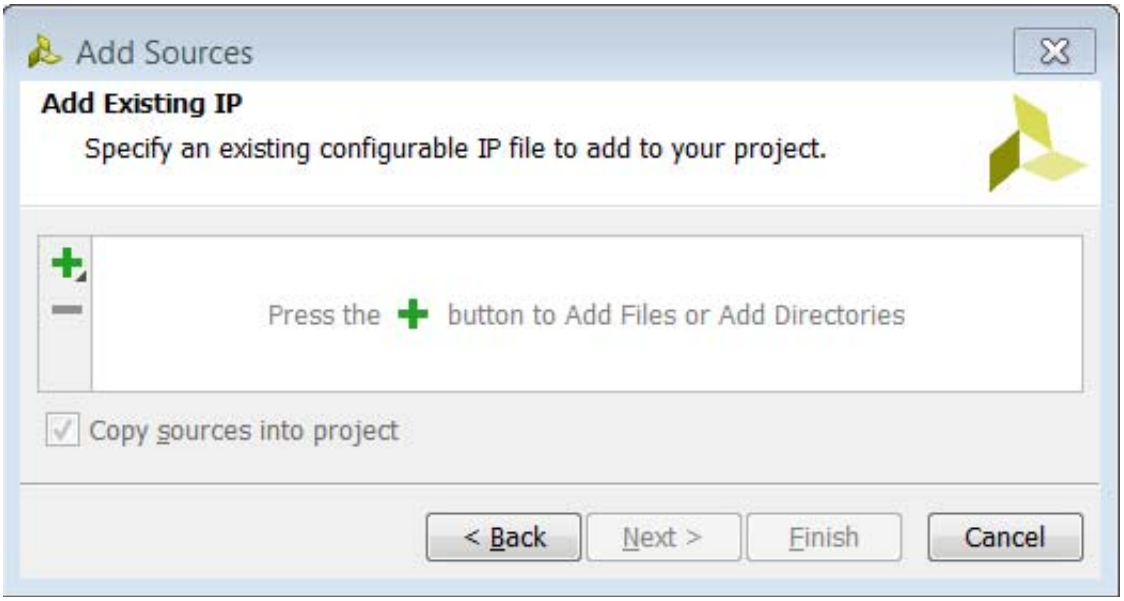


Figure 2-17: **Add Existing IP Dialog Box**

The added IP and any output products are shown in the IP Sources tab of the Sources view, as well as with other source files in the Hierarchy, Libraries, and Compile Order views.

You can select the IP in the IP Sources and view the properties for it in the Source File Property window.

IMPORTANT: *NGC format files are not supported in the Vivado Design Suite for UltraScale devices. Xilinx recommends 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.* See this link in the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 26]for more information about migrating files for Vivado.

When adding or importing an existing IP into a project, the existing output products for the IP are referenced or copied into the project; however, the design runs are not.

To create the design runs for the IP you have two options:

1. Regenerate the output products for the IP, ensuring that the **Synthesis Options** are set to **Out-of-Context per IP**.
2. Enter the create_ip_run command into the Tcl Console:

   ```
   create_ip_run -force [get_ips <ip_name>]
   ```

   The run reports that it has not been started yet, because, in this project, it has not been started.

   The added IP also needs to be instantiated into the top-level design as described in Instantiating an IP.

Tcl Commands

Tcl commands to add existing IP and its generated output products to a project are as follows:

Use the import_files to add existing IP. For example:

```
import_files
```

Use the read_ip to remotely access an IP. For example:

```
read_ip <ip_name>
```

Use the add_files command does not copy IP into the project.

To add files:

```
add_files <file_name>
```

Creating a Memory IP Customization

The Memory IP create memory controllers for Xilinx devices and IP. Memory IP creates complete customized Verilog or VHDL RTL source code, pinout, and design constraints for the selected FPGA, and script files for implementation and simulation.

In 7 series devices, memory IP is referred to as Memory Interface Generator (MIG). This terminology is being deprecated with the UltraScale devices.

See the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)[Ref 39], and *Zynq-7000 SoC and 7 Series FPGAs Memory Interface Solutions* (UG586) [Ref 37] for more information.

For memory IP on the Zynq UltraScale+™ MPSoC processor, the Vivado tools launch a pin planning project, and lets you set the appropriate pins for that device. See this link in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 25].

VIDEO: *See the Vivado Design Suite QuickTake Video: Designing with UltraScale Memory IP for instructions on how to use memory IP.*

Also, visit the Solution Centers for more information about Memory IP. The Designing with IP Design Hub in the Xilinx Document Navigator provides videos and links to Memory IP documentation.

Re-Customizing Existing IP

You can re-customize existing IP in either an RTL project or in a Manage IP project.

1. To open the IP customization dialog box for an IP, you can either:
   - In the sources window, double-click the IP.
   - In the **IP Sources** view, right-click the IP, and select **Re-customize IP** from the menu as shown in the following figure.

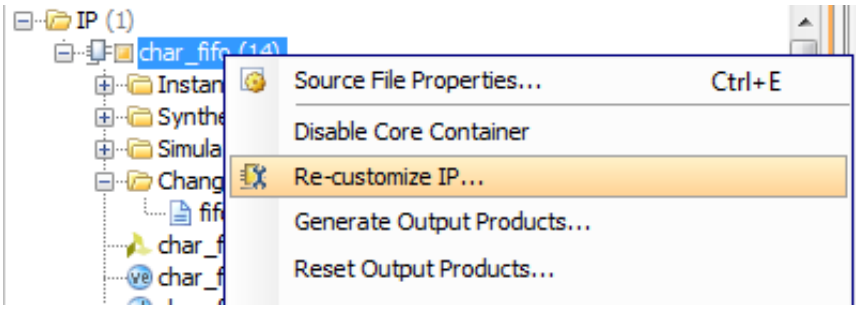


Figure 2-18: **Re-customize IP Option**

2. Change IP configuration settings, and click **OK**.

   The Generate Output Products dialog box opens.

Alternatively, you can review the current settings, and click **Cancel** to keep the settings intact.

When you do make changes to the IP configuration and generate the output products, the existing products are reset and, if enabled, the design run resets and launches again.

RECOMMENDED: *Xilinx recommends that you always generate the output products for IP, including the synthesized DCP;* however, if you do not generate output products after customizing, or re-customizing the IP, the Vivado Design Suite generates the output products automatically, as needed; for example, when synthesizing the top-level design for instance.

Copying an IP

You can copy an existing IP customization to use as a starting point for a new IP. This is useful when you have already customized an IP, need to only make small or simple customization changes for the new IP.

To copy an IP:

1. In the IP Sources tab, select the IP, right-click and select **Copy IP,** as shown in the following figure.

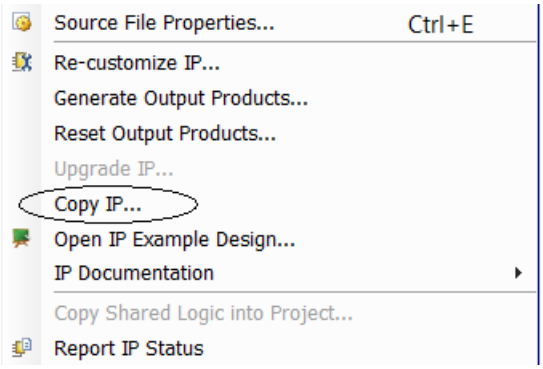


Figure 2-19: **Select IP to Copy**

2. Provide a destination name and location for the copy, as shown in the following figure.

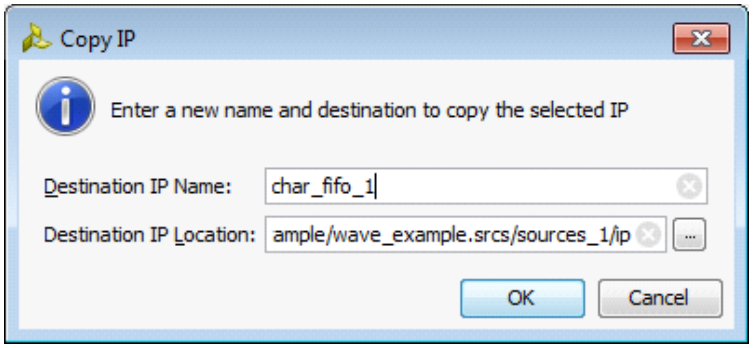


Figure 2-20: **Copy IP Dialog Box**

You can save the copied IP into the project directory structure, which by default is located at `<project_name>.src/sources_1/ip/`, or specify a different location to store the copied IP outside of the current project.

When working with a Manage IP project the default location is the same location as the `/mange_ip_project` directory.

Then, you can re-customize the copied IP customization by either double-clicking the IP, or right-clicking and selecting **Re-customize IP** from the IP Sources tab.

The copied IP customization window opens with the customization settings from the original IP. You can now make edits.

CAUTION! *It is possible to have an IP that references different sub-IP versions; an older version that is locked and another, newer version. In such a case, the synthesis tool could produce errors or logic bugs because files exist with the same module names but with different contents. Upgrade the sub-IP or synthesize the IP out-of-context.*

Tcl Command Example for Copying IP

You can use the `copy_ip` command to create a copy of an IP customization:

```
copy_ip -name newFIFo [get_ips char_fifo]
```

This example creates a copy of the `char_fifo` IP, names it `newFIFO`, and adds it to the project. Because no directory was specified using the `-dir` option, the IP is created inside the project directory structure.

Generating Output Products

After IP customization is complete, the Generate Output Products dialog box opens, as shown in 2-21.

Output products delivered by the IP are listed in the Preview area, and you can do any of the following:

- To generate the listed output products, click **Generate**. By default, the Vivado IDE creates an XCI and a DCP for the IP when you generate the output products, as well as a change log, a behavioral simulation model, and an instantiation template.
- To delay generation of output products, click **Skip**. This lets you select multiple IP customizations and generate all output products at one time, including launching parallel synthesis runs for IP DCP files.

TIP: *When working in Project mode, output products are automatically generated as needed prior to synthesis of the top-level design. This includes any specified DCP files. In addition, XCI files and Instantiation Templates are always generated for IP cores, even when other output products are not generated.*

By default, the Vivado Design Suite generates the synthesized IP DCPs out-of-context.

In the Generate Output Products dialog box, do one of the following:

- Click **Out-of-Context Settings**, shown in the following figure
- Check the **Global Synthesis** option. This instructs the tool to perform top-down synthesis on the current design. All OOC run files are removed when you check this option.

  You can also specify the number of out-of-context synthesis runs to launch at one time. By default, one job is specified, and the design runs launch sequentially.

  A higher number specifies the maximum number of design runs that can be running in parallel; see the following figure.

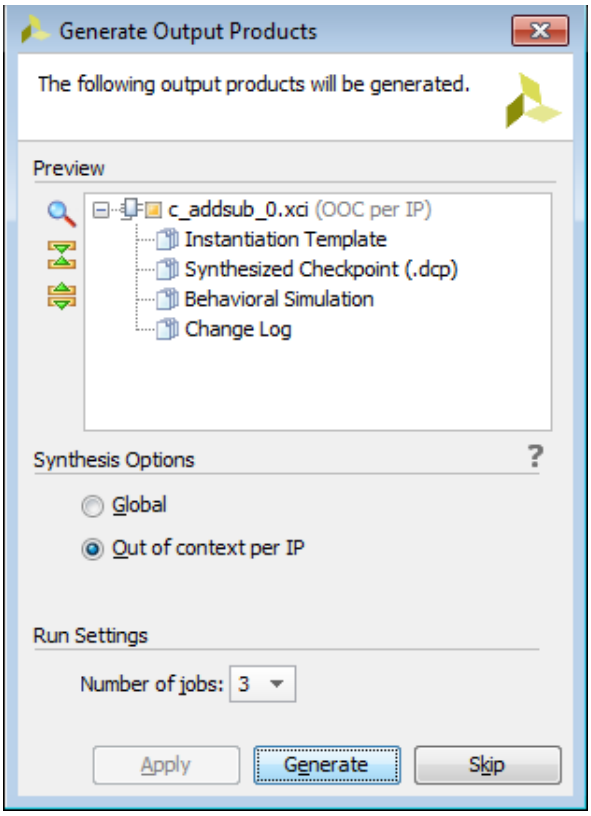


Figure 2-21: **Generate Output Products Dialog Box**

After changing these settings, you can either generate the output products or delay output product generation. DCP generation preferences are preserved whether or not you generate output products now.

Tcl commands: Reset and Generate Target

The Tcl command required to reset the and regenerate the output products are as follows:

- reset_target:

```
reset_target all
[get_files /project_1/project_1.srcs/sources_1/ip/<core_name>.xci]
```

- generate_target:

```
generate_target all
[get_files project_1/project_1.srcs/sources_1/ip/<core_name>.xci]
```

Synthesis Options for IP

When generating the output products for an IP, the default behavior is to produce an out-of-context (OOC) synthesized design checkpoint (DCP). Alternatively, you can choose to synthesize the IP along with the top-level user logic, which is called *global synthesis*.

The following figure shows the two possible flows for synthesizing IP:

- The default OOC flow
- Synthesizing the IP HDL along with the top-level user HDL (global synthesis)

In either flow, Vivado IDE generates HDL and XDC files for the IP, and uses those files during synthesis and during implementation.

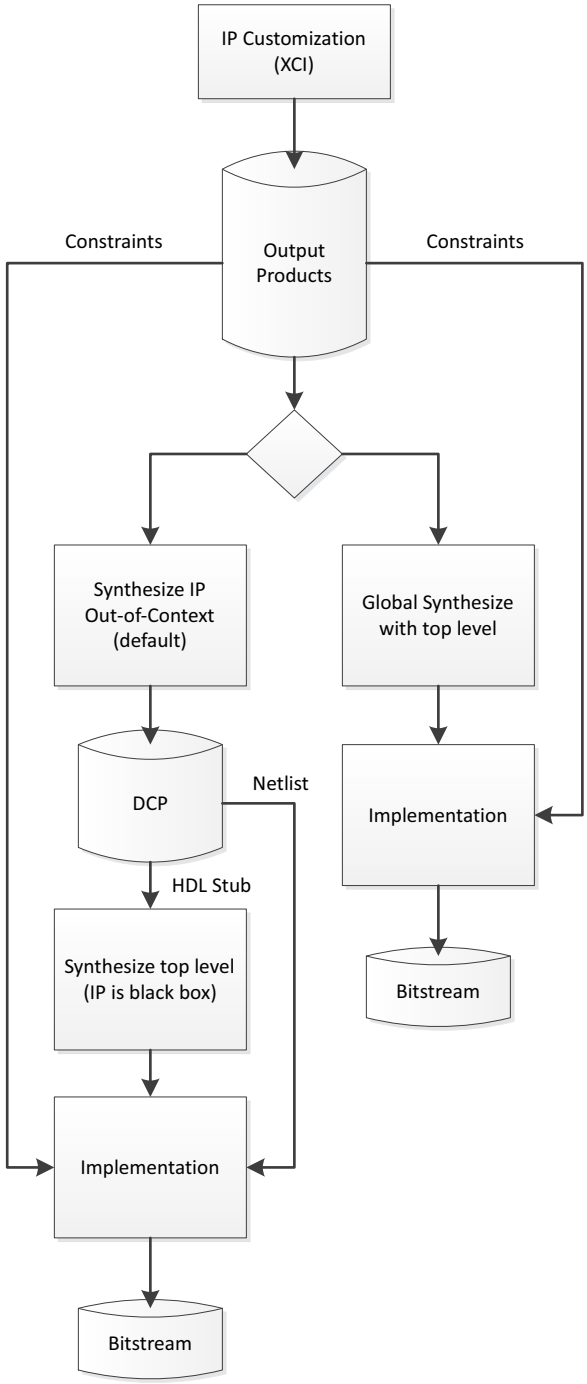


Figure 2-22: **Out-Of-Context (OOC) and Global Synthesis Flow**

Global Synthesis Flow

In the global synthesis flow, the IP is synthesized along with user HDL. Any changes made to user HDL result in the IP being re-synthesized as well.

During implementation, the IP XDC output products that were generated for use during implementation are applied along with any user constraints.

RECOMMENDED: *Always reference the IP using the XCI file. It is not recommended to read just the IP DCP file, either in a Project Mode or Non-Project Mode flow. While the DCP does contain constraints, it does not provide other output products that an IP could deliver and that could be needed, such as ELF, COE, and Tcl scripts.*

Out-of-Context Flow

When working with Vivado Design Suite IP, you can disable the generation of an out-of-context (OOC) DCP and instead synthesize the IP RTL with the top-level design using the **Global Synthesis** option.

RECOMMENDED: *Using the OOC flow when generating IP is recommended and is also the default behavior in the Vivado Design Suite. The OOC flow speeds up run time for the complete project, and lets you avoid re-synthesizing IP when doing project runs.*

In the OOC flow, the IP is synthesized as a standalone module and an out-of-context (OOC) DCP is produced. A special OOC flow-only XDC output product file, the `_ooc.xdc`, is used when synthesizing the IP, which provides default input clock definitions. The produced DCP is a container file, and includes a netlist as well as constraints. The `_ooc.xdc` file is part of the IP definition, and is stored in the IP Catalog.

When synthesizing the top-level design, an HDL stub module is provided with the DCP file, and causes a black box to be inferred for the IP.

Also, during synthesis of the entire design, the DCP provides an XDC file, the `_in_context.xdc` file, which defines any clocks an IP might output, for use by the top-level design. See Understanding IP Constraints for more information.

During implementation, the netlists from the IP DCPs are linked with the netlist produced when synthesizing the top-level design files, and the Vivado Design Suite resolves the IP black boxes. The IP XDC output products that were generated for use during implementation are applied along with any user constraints.

The OOC flow is the default flow because of two main benefits:

- It improves synthesis run times because you only synthesize the IP when changes to the IP customization or version require it, rather than re-synthesizing it as part of the top-level design.
- It produces an `<ip_name>_sim_netlist.v` or an `<ip_name>_sim_netlist.vhdl` structural simulation netlist. You can use these files during simulation if you use a single language simulator and the IP does not deliver behavioral HDL in that language. See Simulating IP for more information.

Note: In versions of the Vivado Design Suite that are older than 2015.3, the simulation files are named `*_funcsim.v` and `*_funcsim.vhdl`.

Examining Generated Output Products

The IP Sources window shows the generated output products for all IP in the project. By default, the output products for an IP are written to the local project directory, at `<project_name>.srcs/sources_1/ip/<ip_name>`; however, when you customize the IP from the IP Catalog, the IP location can be specified as outside the local project directory.

After generating the synthesis output products, the Vivado IDE creates and launches a design run to produce the OOC DCP.

By default, the Vivado IDE creates a synthesized design checkpoint (DCP) file automatically during the customization process for most Vivado Design Suite IP.

When performing synthesis of the top-level design, IP is marked for the out-of-context flow, with an associated DCP file, and treated as a *black box* because it is being synthesized OOC.

While the synthesis run is processing, the OOC related files are shown as missing, as shown in the following figure.

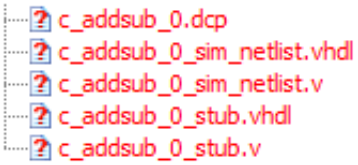


Figure 2-23: **OOC Flow Output Products Pending Creation**

If you elected to use **Global Synthesis**, and to not generate the DCP, the Vivado IDE does not create the `_sim_netlist`, and stub files.

If you do not generate output products, an instantiation template is the only generated product (besides the XCI and BOM files, which are not displayed) as shown in the following figure.

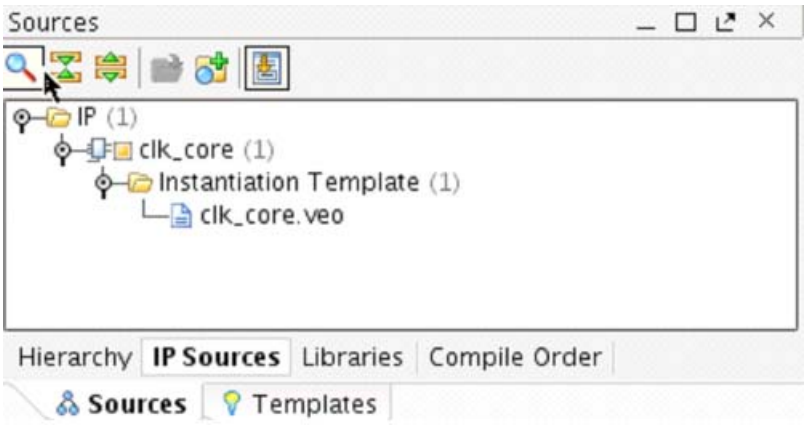


Figure 2-24: **IP Customization with Generation of Output Products Skipped**

As shown in the following figure, when the output products are generated, the Sources window lists unencrypted files.

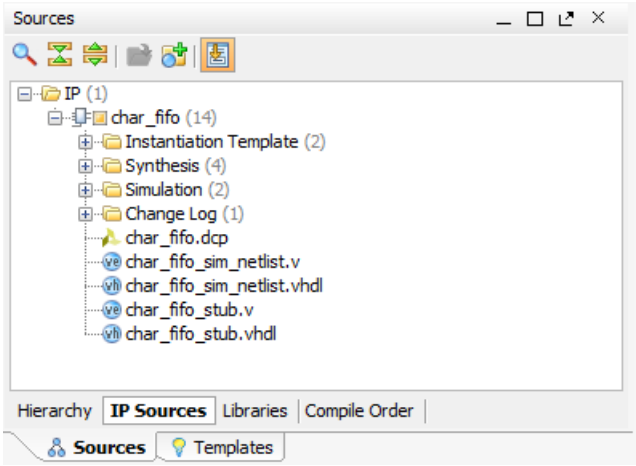


Figure 2-25: **IP Customization with Output Products Generated, Including OOC DCP**

These files include: an instantiation template, synthesis and simulation targets, XDC constraints, a change log, and other products.

By default, the Vivado IDE creates an OOC DCP along with structural simulation netlists (`<ip_name>_sim_netlist.v` or `<ip_name>_sim_netlist.vhdl`) and creates stub files (`*_stub.v`/`*_.vhdl`) for use with third-party synthesis tools to infer a black box for the IP.

Note: In versions of the Vivado Design Suite that are older than 2015.3, the simulation files are named `*_funcsim.v` and `*_funcsim.vhdl`.

Note: Not all output products for an IP are shown in the IP Sources after generation. Encrypted files are not shown, nor are XDC files that are not placed in the synthesis file group. To see these files, look in the Design Sources or Compiler Order views. You can also use the report_compile_order command.

Manually Generating Output Products

At any point you can manually generate output products as follows:

1. In the IP Sources or the Hierarchy view, select the IP.
2. Right-click and select **Generate Output Products**.

The default flow when generating output products is to create and launch an out-of-context (OOC) synthesis design run for the IP. This results in the inference of a black box for the IP when synthesizing the top-level of the design.

IMPORTANT: *The implementation stage resolves black boxes by extracting the netlists from the DCP of the IP.*

Note: If you do not want to generate an out-of-context DCP file for an IP, or if you want to use a scripted flow, you can set the `GENERATE_SYNTH_CHECKPOINT` property to `FALSE`, and the checkpoint is not created when the output products are generated.

Tcl Command to Disable OOC Options on IP

```
set_property GENERATE_SYNTH_CHECKPOINT FALSE [get_ips <ip_name>]
```

RECOMMENDED: *Xilinx recommends that you use the default OOC to reduce the run time on synthesizing your design. When using the OOC flow, you do not need to synthesize the IP every time you run synthesis during development.*

Using a Core Container

The Core Container feature helps simplify working with revision control systems by providing a single file representation of an IP.

VIDEO: *For more information, see the following:* Vivado Design Suite: QuickTake Video: Using Core Containers for IP.

This optional feature lets you elect to have IP and all generated output files contained in one compressed binary file with an extension of XCIX. This extension is similar to the XCI file used for the IP customization file and works in a similar way.

When adding or reading an IP, you specify the XCI file, and in the case where you have enabled the core container, you add or read the XCIX file.

When enabling the core container feature for an existing IP, the XCIX file replaces the IP directory and the output products. When disabling the core container feature for an IP, the XCIX file is converted to the IP directory with all the output products including the XCI file.

Enabling the core container for an IP changes the on-disk representation of that IP instance; the internal representation for the IP remains the same within Vivado. The following figure shows two IP (`char_fifo` and `clk_core`) in the IP Sources view in Vivado as well as their location in the project directory in a file explorer.

- The `char_fifo` IP is set to use core container
- The `clk_core` does not use the core container.

Within the Vivado Sources view, shown in the following figure, the two IP appear the same, both listing the output products, all of which can be opened for viewing from within the Vivado IDE.

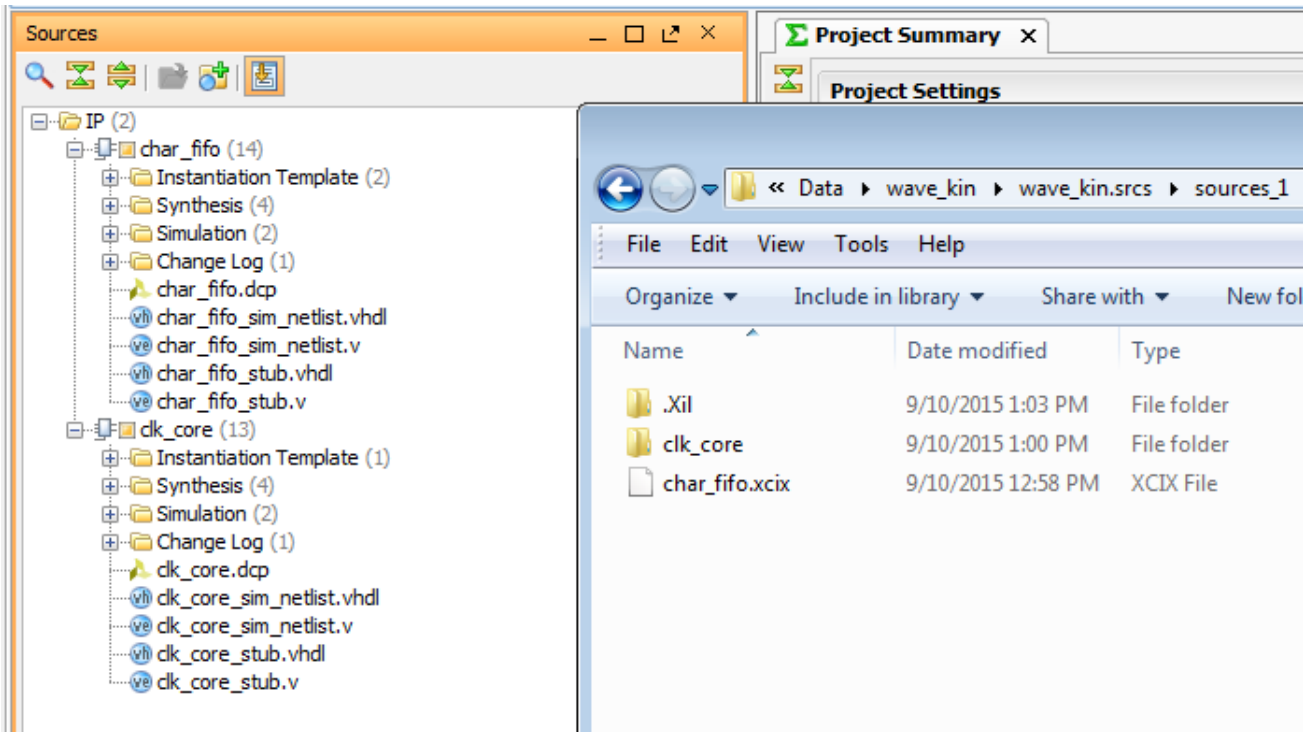


Figure 2-26: **IP Using Core Container**

On disk, there is a folder for `clk_core` and the single XCIX file for the `char_fifo` IP. When an IP uses the Core Container, Vivado reads the IP source files needed during synthesis and implementation from this single XCIX file. The files are not extracted to a temporary directory, they are read directly from the binary.

IMPORTANT: *Again, having a single file representation for the IP simplifies revision control.*

Using just the XCIX file allows you the same abilities as with the XCI file, such as:

- Reporting status and details of the IP (Report IP Status)
- Upgrading the IP when a new version is available; as well as the ability to use an older version of the IP if desired
- Ability to re-customize the IP (if the latest version)
- Ability to reset and regenerate the IP (if the latest version)

Visually, the IP looks the same in a project whether using the core container feature or not (2-26). The IP Sources tab shows all the files that have been generated for the IP and you retain the ability to open then for viewing as before.

Also, Tcl commands related to IP remain the same whether or not you use the core container.

As with all IP, certain files are stored in the `ip_user_files` directory for ease of use. See Understanding the IP User Files (ip_user_files) for more details.

Enabling and Disabling the Core Container

The Core Container feature is disabled by default. To have all newly-created IP use the Core Container feature, go to **Project Settings > IP > General**, and check the **Use Core Containers for IPs**. See the following figure for an example of the General IP Project Settings.

VIDEO: *See the* Vivado Design Suite QuickTake Video: Using Core Containers for IP *for more information.*

There are two methods to enable Core Container:

- To use the Core Container format for all IP, select the **Project Settings > IP> General** tab, right-click, and check the **Use Core Container** option.

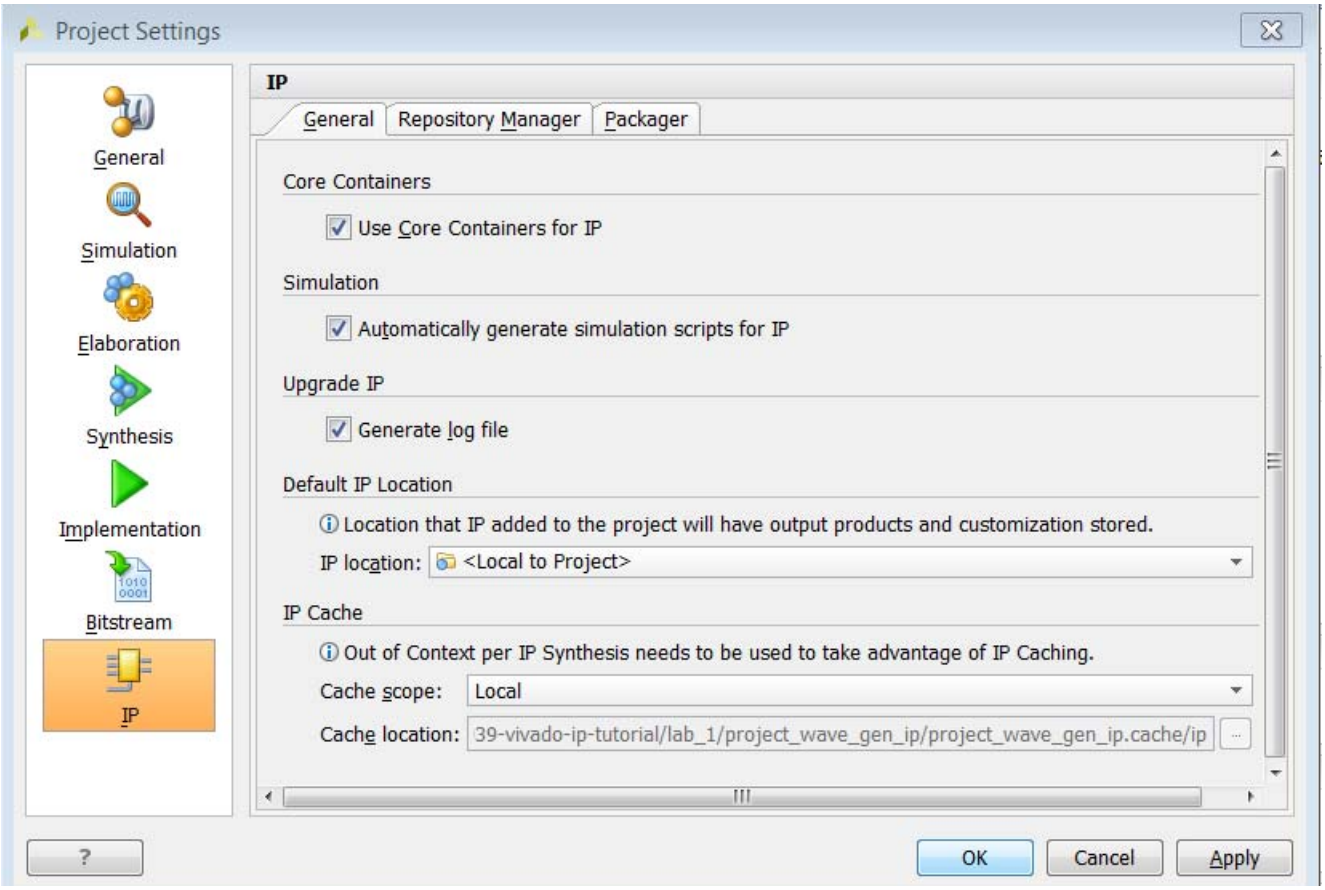


Figure 2-27: **Project Setting to Enable Core Container**

- If you have an existing IP that you want to use the Core Container format, select the IP from the IP Sources, right-click, and select **Enable Core Container** as shown in the following figure.

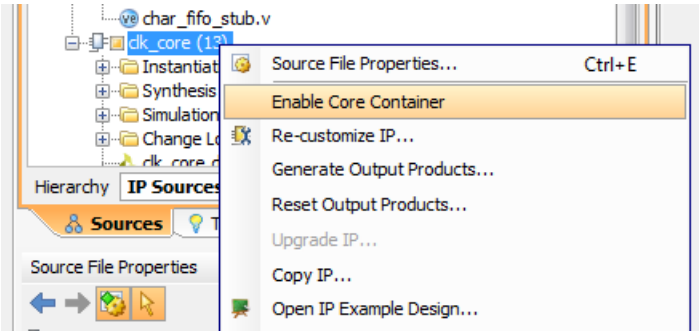


Figure 2-28: **Enabling Core Container for an IP**

To disable the Core Container for an IP using it, select the IP from the IP Sources tab, right-click, and select **Disable Core Container.**

When upgrading a Vivado project from an old release to the current release, if IP is detected, the Enable Core Container dialog box opens and prompts you with an option to **Convert IP to Core Container and Set as Default in Project use the Core Container** feature as shown in the following figure.

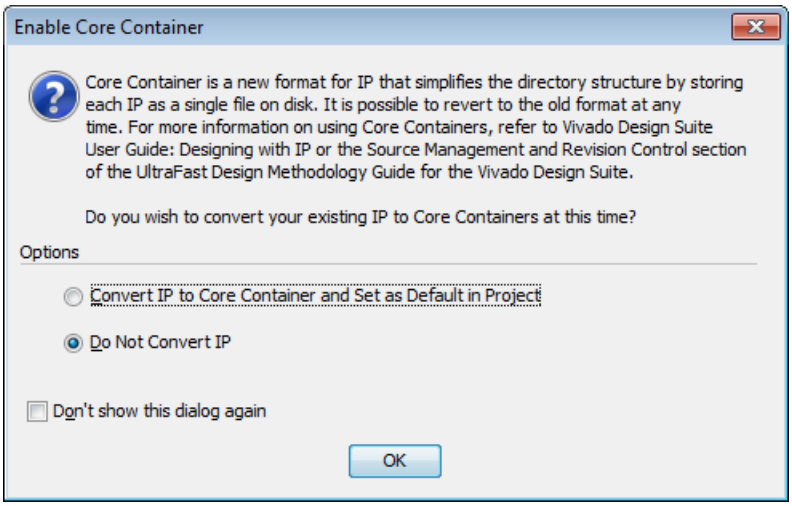


Figure 2-29: **Covert to Core Container Dialog Box**

Note: The 7 series Memory Interface IP does not support the Core Container feature. Additionally, IP that are inside of an IP integrator block design cannot use the Core Container feature.

Simulating with Core Container

When you enable Core Container, the Vivado tools store simulation-related files for an IP outside of the XCIX file during the generation of the IP for user convenience. If only the XCIX is available, these files can be extracted using the `export_ip_user_files` command. For more information, see this link to the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 17].

Behavioral simulation files for an IP using Core Container are stored in a `sim` directory, which can be in one of two locations:

- When using a project, the simulation files are located in: `<project_directory>\<project name>.ip_user_files\ip\<ip_name>\`
- When using a Managed IP Project to create IP, the simulation files are located in: `<managed_ip_project directory>\ip_user_files\ip\<ip_name>\`

For third-party simulators, in the case where an IP only delivers behavioral simulation in a single language which is not supported, functional simulation files are provided in the location listed above:

- `<ip_name>_sim_netlist.v`
- `<ip_name>_sim_netlist.vhdl`

Note: In versions of the Vivado Design Suite that are older than 2015.3, the simulation files are named `*_funcsim.v` and `*_funcsim.vhdl`.

For more details on the IP simulation-related files produced and their locations see Simulating IP, page 63.

Support Files for Core Container

As with the simulation files, additional support files for an IP using Core Container are extracted for convenience during generation of the IP. These files consist of:

- Simulation files as described in the previous section, Simulating with Core Container.
- Instantiation template files for Verilog and VHDL (`.veo` and `.vho`)
- Stub files for use in a third-party synthesis tool to infer a black box for the IP (`*_stub.v` and `*_stub.vhdl`)

These support files are located in one of two places:

- When using a project, the files are in: `<project_directory>\<project name>.ip_user_files\ip\<ip_name>\`
- When using a Managed IP project, the files are in: `<managed_IP_project_directory>\ip_user_files\ip\<ip_name>`

Tcl Command to Export Support Files

During generation of an IP regardless of the use of the Core Container feature, the support files are automatically placed in the locations described in the previous sections Simulating with Core Container and Support Files for Core Container.

In the case you only have the XCIX file and want the support files exported for you, type the following at the Tcl Console:

```
export_ip_user_files -of_objects [get_ips <ip_name>]
```

If you omit the option, the Vivado tools export all IP files in the design again; regardless of whether you are using the Core Container use or not. All files for the Core Container are taken from the XCIX file. For XCI-based IP, the files are copied from the IP directory.

Understanding the IP User Files (ip_user_files)

During generation of the IP output products, some files are automatically copied into a special directory called `ip_user_files` for convenience. This is especially useful when using the Core Container feature (see Using a Core Container). IP support files are stored in a convenient location under the `ip_user_files` directory. This directory structure allows you access to instantiation templates and simulation files for an IP using the Core Container feature without having to manually extract the files from the binary container. IP support files are stored in the `ip_user_files` directories regardless of whether you use Core Container feature.

When you create an IP customization (XCI), the Vivado IDE creates a directory whose name is the same as the IP that contains the IP definition and output products. These files are described in Appendix B, IP Files and Directory Structure.

When you elect to use the Core Container feature, the Vivado IDE creates an XCIX binary file that contains all the files of the IP (see Using a Core Container for more details).

During IP generation certain user files are stored in a special directory outside of the IP directory for convenience. Depending on whether the IP was created in an RTL project or in a Managed IP project this directory is either:

- RTL project: `<directory to project>/<project name>/ip_user_files/`
- Managed IP project: `<managed_ip_project_directory>/ip_user_files/`

Inside of `ip_user_files` there are a number of folders (seen in the following figure). Which folders will be present depends on what is in your project (IP, Block Designs, and so forth).

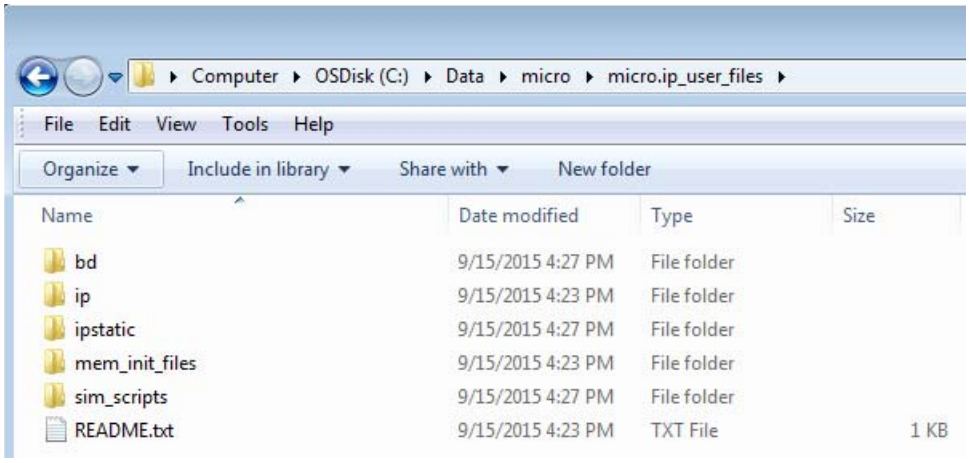


Figure 2-30: **ip_user_files Directories**

The following is a brief description of each of the directories that could be present. Each directory is covered in more detail in this section, and also described in Appendix B, IP Files and Directory Structure.

- `bd`: Contains a sub-folder for each IP integrator block design (BD) in the project. These sub-folders have support files for the IP used.
- `ip`: Contains files specific to each IP customization (XCI/XCIX) that is present in the project or that was created in the Managed IP project.
- `ipstatic`: Contains common IP static files from all IP/BDs in the project.
- `mem_init_files`: This directory is present if any IP deliver data files.
- `sim_scripts`: By default, scripts for all supported simulators for the selected OS are created for each IP and for each Block Design present.

Regardless of whether you use the Core Container feature, the Vivado Design Suite creates these files and directories. In both cases, the files exist; either in the XCIX binary or in the IP directory.

To manually export IP/BD files to the `ip_user_files` directory you can use the `export_ip_user_files` command in the Tcl Console. When you reset and generate an IP or BD, this command runs automatically.

Contents of the bd Directory

The `bd` directory is present if your project has one or more IP integrator block designs. Each BD has a unique sub-folder that contains support files for the used IP. 2-31 shows an example of a project with one BD named `base_microblaze_design`.

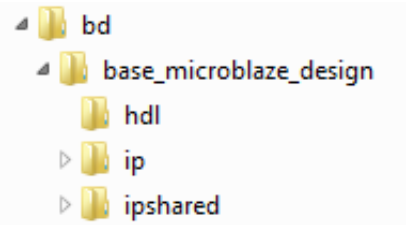


Figure 2-31: **Contents of bd Directory**

The three directories present are:

- `hdl`: Simulation top-level file for the block design.
- `ip`: each IP in the BD will have a directory present containing simulation files.
- `ipshared`: The simulation files which are common between IP present in the BD.

If you selected an out-of-context (OOC) per BD during generation, then stub files are present in the `bd` directory in the respective Block Design sub-folder.

Contents of the ip Directory

The ip directory contains support files for the IP present in the project. These files are placed in a sub-directory named after the IP, as shown in the following figure.

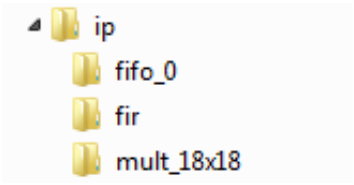


Figure 2-32: **Contents of IP Directory**

The support files include:

- Simulation files in a sub-folder called sim (Core Container only – see Simulating with Core Container.)
- Instantiation template files for Verilog and VHDL (.veo and .vho)
- Stub files for use in a third-party synthesis tool to infer a black box for the IP (*_stub.v and *_stub.vhdl)

The support files are also located in the IP directory. For convenience and consistency with IP using the Core Container feature, copies of files that a user might need are placed in the ip_user_files directory as well.

Contents of the ipstatic Directory

There are many IP that share files used for simulation that do not change for each customization. The ipstatic directory contains these files for all IP and BD in the project. The scripts created for simulation reference the files in this directory as needed.

The dynamic simulation files that an IP deliver are in the IP customization directory. When using the Core Container feature, the dynamic simulation files are located in the ip directory. See Contents of the ip Directory.

Contents of the mem_init_files Directory

Some IP deliver data files. These files are marked with a DATA property. All these files are stored in the mem_init_files directory. The files that can be present are tagged as data, and include memory initialization files (MIF) and text files (TXT).

Contents of the `sim_scripts` Directory

Scripts are created for each IP and BD simulation. By default, the Vivado Design Suite generates scripts for all simulators that are supported by the OS on which the IP was generated.

For Microsoft Windows, this includes:

- Vivado simulator
- Mentor Graphics ModelSim
- Mentor Graphics Questa
- Riviera
- Active HDL (Windows only)

For Linux, this includes the following simulators also:

- Synopsys Verilog compiler simulator (VCS)
- Cadence incisive enterprise simulator (IES)

To control scripts generation, see the General IP Settings.

The generated scripts reference the simulation files from the IP customization directory. For IP that use the Core Container feature, the scripts reference the simulation files in the Contents of the `sim_scripts` Directory directory. For IP in a block design, the scripts reference the simulation files in the Contents of the `bd` Directory.

Tcl Command to Export Support Files for IP

During generation of an IP, regardless of the use of the Core Container feature, support files are placed in the described locations automatically. See Tcl Command to Export Support Files.

Instantiating an IP

After you create an IP Customization, you can instantiate that IP into a design. An instantiation template is created after IP customization, regardless of whether you generated the output products.

The instantiation template is available in the IP Sources tab of the Sources window in the `Instantiation Template` directory as shown in the following figure.

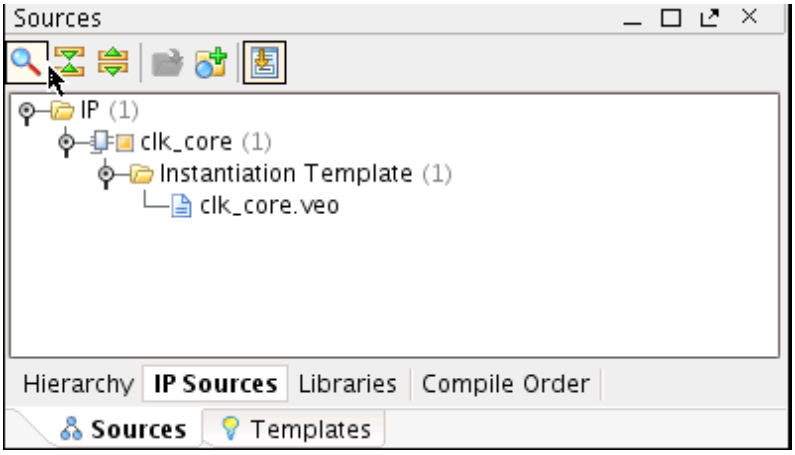


Figure 2-33: **Location of Instantiation Template**

Depending on the **TARGET_LANGUAGE** property setting on the current project, a Verilog (VEO) or VHDL (VHO) instantiation template file is generated that you can copy and paste into your RTL design, shown in the following figure, illustrates the instantiation template for a Clocking Wizard IP customization.

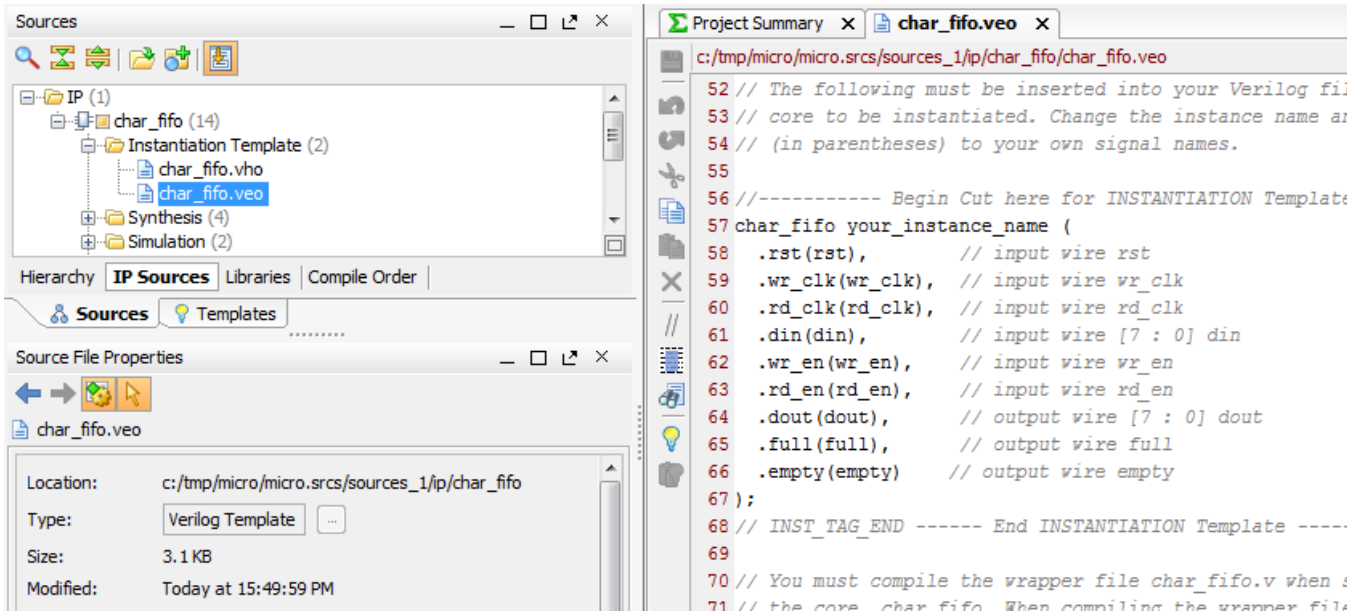


Figure 2-34: **Instantiation Template**

To use the instantiation template in your design, use the following steps:

1. Open the instantiation template file for the IP customization by double-clicking the file in the Sources view, or by selecting the file using the **Open Files** command.
2. Highlight the Instantiation Template between the comments as indicated in the text of the instantiation template, and copy the section.

3. Open the design HDL file in which you want to instantiate the IP either at the top-level or in the hierarchy of the design.
4. Paste the copied template to the location of your choice.
5. Edit the HDL to integrate the template into your design as needed; for example, change the port connections.

   After the IP is instantiated into a design, the IP is listed in the Hierarchy tab of the Sources window at the location in the design where it is instantiated, as shown in the following figure, which shows the `clk_gen` and the `char_fifo` IP instantiated in a design.

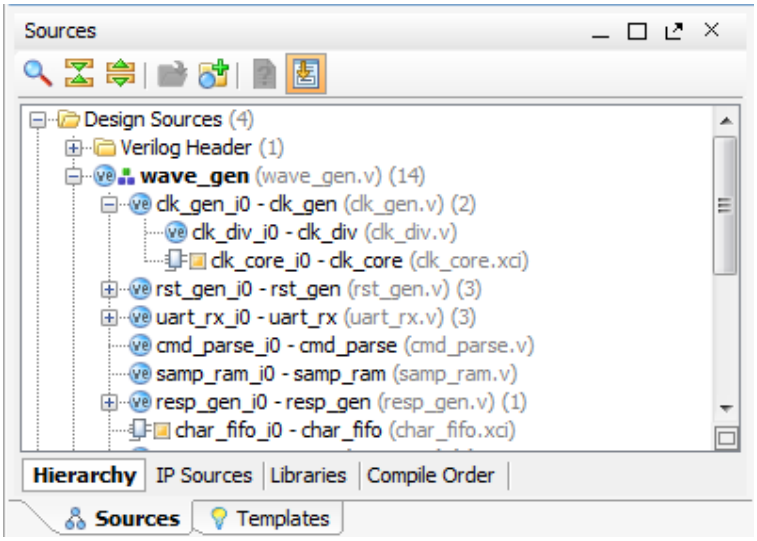


Figure 2-35: **IP Instantiated in a Design**

With the IP customization properly instantiated into your design, you are ready to synthesize the IP along with the rest of your design, either as a black box if the OOC flow is used, or with the top-level of the design, if you are using global synthesis. See Synthesis Options for IP for more details.

TIP: *When you expand the IP hierarchy by right-clicking the IP, you could see the Encrypted IP source icon* *. The content of this source cannot be viewed.*

Reporting IP Status

You can view the report of all IP in a project using the **Tools > Report > Report IP Status** drop-down menu. A new tab titled **IP Status** displays the report results, as shown in the following figure.

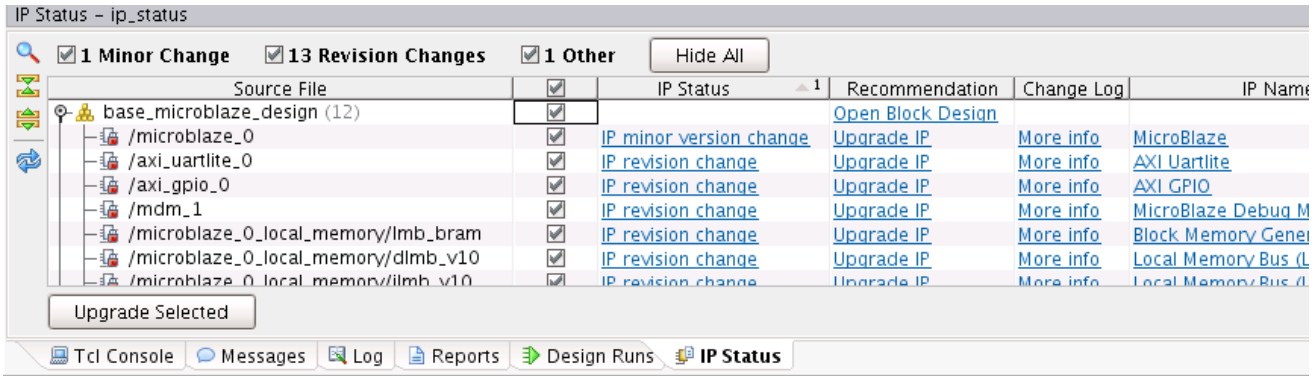


Figure 2-36: **Report IP Status View in GUI**

Filtering Status

You can filter what is displayed in the IP Status report by using the check boxes at the top of the Report window. Based on the IP in the design any of the following check boxes can be present.

- Major change
- Minor change
- Revision change
- Part change
- Up-to-date
- Other: This category consists of IP in miscellaneous states. The following are examples:
  - IP definition not found
  - Read-only XCI, XML, or XPR file
  - User-managed IP
  - Disabled component
  - Incompatible license
  - Incompatible XCI or XML file
  - Deprecated flow
  - Locked due to child IP being locked

Viewing the Change Log

Each IP delivers a change log. The change log provides information about changes to the IP for each release.

You can access the Change Log with the following options:

- In the IP Sources section of the Sources view, select the IP, then right-click and select IP **Documentation > View Change Log**.
- In the Report IP Status view, select the IP, then right-click and select **View Change Log**.
- In the Report IP Status view, in the Change Log column, select **More info** to produce a pop up with the information for the latest release. You can select the view full log link to open the full change log as shown in the following figure.

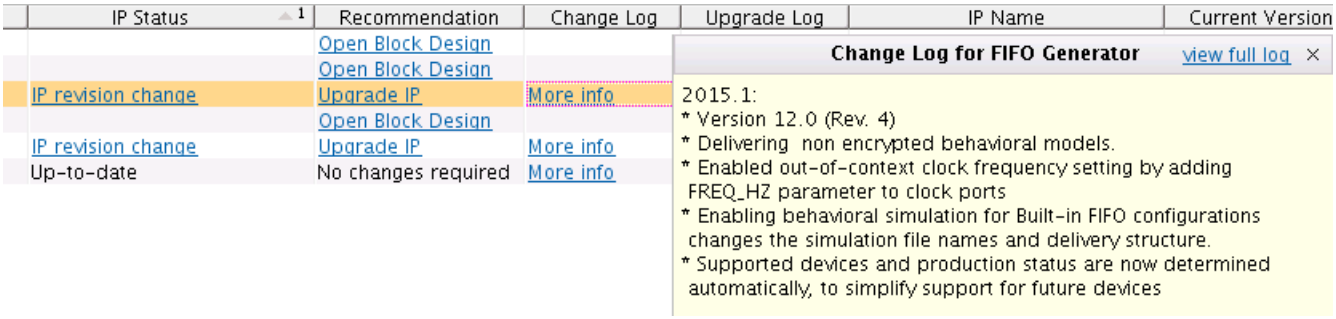


Figure 2-37: **Change Log for More info link**

Reporting IP Status using a Tcl Command

You can also generate this information using the report_ip_status Tcl command:

```
report_ip_status
```

Understanding IP States Within a Project

There are several states that an IP can display within in a project, depending if it is the current version in the catalog and if you have generated output products.

IMPORTANT: *For imported IP cores with versions that are not accessible from the Vivado IP Catalog, re-customizing, re-setting, and re-generating the IP is not enabled.*

When you add existing IP (either in the XCI or XCO form), if present, the output products (such as HDL files) are also added.

The following table shows the buttons that represent the states of the Vivado IDE IP:

Table 2-1: **IP States in a Project**

Button	Description
	IP in an RTL project to be synthesized out-of-context (OOC). See Out-of-Context Flow.
	Customized IP which is in the IP Catalog that is to be synthesized with the project (global synthesis). See Global Synthesis Flow.
	Unmanaged IP. The user has changed the `IS_MANAGED` property of the IP to be `false` and has taken responsibility for the management of the IP. The IP becomes locked also. The purpose would be for the user to make modifications to unencrypted HDL sources or constraints. See Appendix D, Editing or Overriding IP Sources.
	Locked IP that have output products can be used in the flow, but cannot be recustomized or regenerated. Locked IP with no output products are not usable in the flow. To use this locked IP with no output products, either provide the original output products or upgrade to the latest version. See Appendix A, Determining Why IP is Locked for more information.

Understanding IP Constraints

The Vivado IDE manages both user-defined XDC timing and physical constraints for the entire design, as well as for Xilinx IP. It handles the association and the unification of constraints for Xilinx IP instantiated multiple times within a project.

Most IP in the IP Catalog deliver IP-specific XDC constraints based on user customization. The constraints delivered by the IP are optimized using the default synthesis settings.

RECOMMENDED: *Do not change these settings for any of the IP design runs because you could encounter issues with applying constraints. To take ownership of constraining an IP, disable the XDC file(s) that are delivered with an IP. If you must change the synthesis settings for an IP, select the IP run from the* ***Out-of-Context Module Runs*** *in the Design Runs tab, then make changes in the Options tab of the* ***Synthesis Run Properties****. You can also script this action using the Tcl command:* `set_property <synthesis_option> <value [get_runs <IP_Name>_synth_1]`.

Tcl Command Example for Changing Synthesis Run Properties

```
set_property STEPS.SYNTH.DESIGN.ARGS.FSM_EXTRACTION sequential /
[get_runs my_IP_synth_1]
```

During design synthesis and implementation, the Vivado Design Suite processes the IP-delivered XDC constraints before processing the user-defined constraints, or after, depending on the constraint file.

Constraint File Processing Order

By default, IP XDC constraints have the PROCESSING_ORDER value of EARLY, and user constraints are marked NORMAL. In this way, the constraints processed later can override constraints on the same object that are processed earlier.

The order in which IP XDC files could be processed are, as follows:

- User XDC set to EARLY
- IP XDC set to EARLY
- USER XDC set to NORMAL (default)
- IP XDC set to LATE
- User XDC set to LATE

Using this method, you can have an XDC file(s) processed before or after IP XDC(s). See the following documents for more detail:

- *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 4]
- This link to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 8]

Vivado IP can generate multiple XDC constraints files. By default, IP constraints are processed before user constraints because of the following possibilities:

  - The IP might produce a clock that must be available to the end-user constraints.
  - If the IP delivers physical constraints, the end-user can override them if necessary.

Tcl Command Example to Report Constraint Compile Order

- report_compile_order

```
report_compile_order -constraints
```

This command provides the processing order of constraints used for synthesis and implementation of user logic, and provides a breakdown of the constraints used for each IP synthesis run used for the generation of the IP DCP.

Some constraints that an IP delivers could have a dependency on a clock object that comes from either the end-user or another IP. These constraints are provided in a separate XDC file and are processed after the end-user constraints.

Typically, an IP delivers a core XDC file that can contain clock creation commands as well as commands without external clock dependencies. The customization file name is <ip_name>.xdc, and is referred to as the *core* XDC file.

Some IP can also include another XDC file that contains clock-dependent commands. Because the top-level clock can come from other constraints, or from other IP with a dependency, any constraints that need those clocks to be defined first should be placed in the `<ip_name>_clocks.xdc`. By default, the Vivado IDE processes the `<ip_name>_clocks.xdc` file after user constraints and other IP core XDC files.

Most IP deliver an OOC XDC file as well, (`<ip_name>_OOC.xdc`). This file contains default top-level definitions for input clocks to the IP. This file is only used in the DCP creation when using the recommended default flow (IP synthesized OOC to the top-level design). When the Vivado Design Suite synthesizes the IP OOC of the top-level design, clocks that are created by the end-user or other IP are not available; consequently, this file is necessary to provide the clock definitions for synthesizing the IP.

The `<ip_name>_OOC.xdc` is not needed during implementation of the user logic with the IP, because all the netlists are linked together before constraints are applied. At that point a user-created clock or an IP-created clock is available to any IP that requires a clock.

Some IP can deliver additional XDC files. This might be because they deliver constraints that are to be used only during synthesis or only during implementation. For a list of possible XDC files that an IP can deliver, see Appendix B, IP Files and Directory Structure.

Some IP support a the Vivado Board Flow as defined at this link to the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 18].

When you create a project that targets a platform board instead of a target part, that board is available during the IP customization letting you specify which connections on the board to use in connecting to the IP. This produces an `<ip_name_board>.xdc` file which contains `PACKAGE_PIN`, `IOSTANDARD`, and other physical constraints.

For more detailed information on XDC constraints, see this link to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 8].

After some constraints are processed for a project, those constraints can become project *Properties*. For more information regarding properties, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

The following subsections briefly describe some of the constraint files that the Vivado Design Suite creates when processing IP.

VIDEO: *See the Vivado Design Suite QuickTake Video: IP Constraints Overview, for a demonstration of how the following constraints are used during IP flow.*

dont_touch.xdc

The Vivado Design Suite uses the `dont_touch.xdc` to set `DONT_TOUCH` properties on the IP top-level during synthesis of the IP. This prevents interface ports from being removed.

You can see this constraint file being processed in the synthesis log file, either for the IP when it is synthesized using OOC (the default flow), or in the global synthesis log file when synthesizing the IP with the end-user RTL.

in_context.xdc

By default, IP is treated as a black box during top-level synthesis because it is synthesized out-of-context. During the creation of the IP DCP, an `<ip_name>_in_context.xdc` file is created and stored in the IP DCP file, under the following conditions:

- The IP produces a clock which can be referenced on the IP boundary
- The IP has an instance of any I/O buffers

If present, the `<ip_name>_in_context.xdc` file is processed before the end-user constraints when synthesizing the user logic. This file is not necessary during implementation because the IP are no longer a black box.

If clocks are created, they are placed on the boundary pins of the IP black box cell. The clock can be of the following types:

- A primary clock on an input port of the IP (such as a Clocking Wizard IP)
- A primary clock on an output port of the IP
- A generated clock on an output port of the IP with the master being an input clock (such as the Clocking Wizard IP)

A clock would be created on an input port of the IP only in the case that the IP contained an input buffer. The clocking wizard is configured so by default. This clock propagates to a top-level port during synthesis of the top level user logic.

If a user constraint must reference a clock produced by an IP, it should be done indirectly by referencing the pin of the IP where the clock is produced, such as in the get_clocks command:

```
get_clocks -of_objects [get_pins <IP_clock_pin>]
```

If I/O buffers are present, the `IO_BUFFER_TYPE` property is set to `NONE` for the interface pin with an I/O buffer. Setting this property prevents an additional I/O buffer from being inserted during top-level synthesis.

Determining Clocking Constraints and Interpreting Clocking Messages

Vivado can contain hierarchical constraints, top-level user constraints, and constraints that are delivered by an IP. These constraints can have dependencies which must be met to work correctly. One such constraint is clock creation.

- Some IP create clocks that might be needed by other IP or the top-level design.
- Some IP require a clock to exist to function correctly and not produce critical warnings.

If the necessary clock constraint is not being provided, then the IP at the top-level of the design issues a CRITICAL WARNING as described in Examples of Critical Warnings and Warnings on Clocking.

For more information about working with the designs with clocking requirements, see this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 9].

VIDEO: *See the following for more information* Vivado Design Suite QuickTake Video: Creating Basis Clock Constraints *and* Vivado Design Suite QuickTake Video: Creating Generated Clock Constraints

Tcl Command Example of an IP Clock Dependency

The following is an example of a constraint that an IP could deliver that has a dependency on a top-level clock which enters the IP on the port `ref_clk` using the set_max_delay command:

```
set_max_delay -from [get_cells data_reg] -to [get_cells synchro_stage0_reg]\ -
datapath_only [get_property PERIOD [get_clocks -of_objects [get_ports ref_clk]]]
```

The *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 9], at this link describes how the `get_ports` command is converted into a `get_pins` command on the IP cell instance. Depending upon how the IP is connected in a design, the clock could come either from a user-supplied clock or from another IP.

- In the case of another IP supplying the clock, the clock is provided and no critical warnings are produced.
- If the clock is provided in the end-user logic, a critical warning is produced if no clock object is created (using the `create_clock` or `create_generated_clock` command).

Tcl Command Examples for Clocking

One way to find clocks in your design that are not being properly generated is to use report_clock_networks Tcl command:

```
report_clock_networks
```

This command produces a clock report for the design, including constrained and unconstrained clocks. You can use the report to determine if the clock module connected to your IP is missing a clock definition.

Other useful commands are:

- report_clocks

  This command returns a table showing all the clocks in a design, including propagated clocks, generated and auto-generated clocks, virtual clocks, and inverted clocks in the current synthesized or implemented design.

- report_compile_order `-constraints`

  This command shows which XDC files the design is using for synthesis and implementation and in what order they are processed. If an IP XDC that is creating a clock comes after an IP XDC that needs the clock, this clarifies the relationship.

Often you can resolve issues of a missing clock coming from an IP by adding a constraint to your top-level XDC timing constraints file. This could be the case when working with an XPS design where there are no XDC files present for some IP that could be creating a clock, such as a serial transceiver.

Examples of Critical Warnings and Warnings on Clocking

The following are examples of the warnings returned for a design that fails to find the clock constraint needed by an IP core.

```
CRITICAL WARNING: [Vivado 12-259] No clocks specified, please specify clocks using
-clock, -fall_clock, -rise_clock options [C:/Design/v_tc.xdc:1]INFO: [Vivado
12-1399]
There are no top level ports directly connected to pins of cell 'system/v_tc',
returning the pins matched for query '[get_ports s_axi_aclk]' of cell 'system/v_tc'.
[C:/Design/v_tc.xdc:1]Resolution: The get_ports call is being converted to a
get_pins call as there is no direct connection to a top level port. This could be due
to the insertion of IO Buffers between the top level terminal and cell pin. If the
goal is to apply constraints that will migrate to top level ports it is required that
IO Buffers manually be instanced.

CRITICAL WARNING: [Vivado 12-1387] No valid object(s) found for set_max_delay
constraint with option 'from'. [C:/Design/v_tc.xdc:1]Resolution:
Check if the specified object(s) exists in the current design. If it does, ensure
that the correct design hierarchy was specified for the object.
```

Simulating IP

Using simulation is an important and necessary step in the design flow to verify the functionality and performance of the design. To enable this, IP in the Vivado IP Catalog delivers a simulation model that you can include in the simulation of the overall design.

The simulation model delivered for the IP can be one of the following:

- Custom behavioral simulation model
- Plain text or encrypted synthesizable RTL sources used for simulation
- Structural simulation model

Note: Some IP (for example, the FIR Compiler IP) deliver IP-level test benches that you can directly use to simulate the IP. See Using a Test Bench for IP for more information.

TIP: *Third-party simulators that are typically used for simulating Xilinx devices are integrated as options in the Vivado® Integrated Design Environment (IDE). See this link in the Vivado Design Suite User Guide: Logic Simulation (UG900) [Ref 17] for more information on working with third-party simulators.*

Delivering IP Simulation Models

Most Xilinx IP deliver RTL sources for a single language only, either Verilog or VHDL, effectively disabling simulation for *language locked* simulators if you do not have licensing for the language supported by the IP.

To simulate your design and include IP, the Vivado tools ensure the availability of an appropriate simulation model for the IP using the project property **Simulator language** setting. The `SIMULATOR_LANGUAGE` property of the current project lets you tell the Vivado tool which language your simulator supports. The values are **Verilog**, **VHDL**, and **Mixed**. Set this property in Manage IP**,** Project-based, and Non-project based flows.

Some IP deliver simulation files for VHDL and some for Verilog. When the simulator language is set to **Mixed**, the same module for both languages can be sent to the simulator by different IP.

The Vivado simulator is a mixed language simulator and can handle simulation models in both VHDL and Verilog. If you are using a third-party simulator and have license for a single language only, change the **Simulator language** to match your license.

If the IP does not deliver a behavioral model or does not match the chosen and licensed simulator language, the Vivado tools automatically generate a structural simulation netlist (`<ip_name>_sim_netlist.v` or `<ip_name>_sim_netlist.vhdl`) to support simulation.

Note: In versions of the Vivado Design Suite that are older than 2015.3, the simulation files are named `*_funcsim.v` and `*_funcsim.vhdl`.

RECOMMENDED: *When you generate IP output products, enable the synthesized design checkpoint (DCP) option to ensure that the Vivado IDE can deliver a structural simulation netlist for the IP. For more information, see Generating Output Products.*

Note: Some Xilinx IP use the Vivado High Level Synthesis (HLS) tool to produce RTL. These IP require synthesis to be run for these RTL files to be generated. When launching simulation from within Vivado, either the out-of-context synthesis runs for IP which use HLS, or a global synthesis run launch automatically, if needed.

RECOMMENDED: *To have all files required for simulation available in the IP directory for placement in a revision control system it is recommended you run synthesis first.*

If your simulator language is not set to **Mixed**, then you might be required to generate the IP using the default out-of-context synthesis. If the IP you are using does not deliver RTL in the simulation language specified you must create an `_sim_netlist.v` or an `_sim_netlist.vhdl` file to simulate. These files are only created as part of the out-of-context synthesis flow.

The following figure displays if you have this situation.

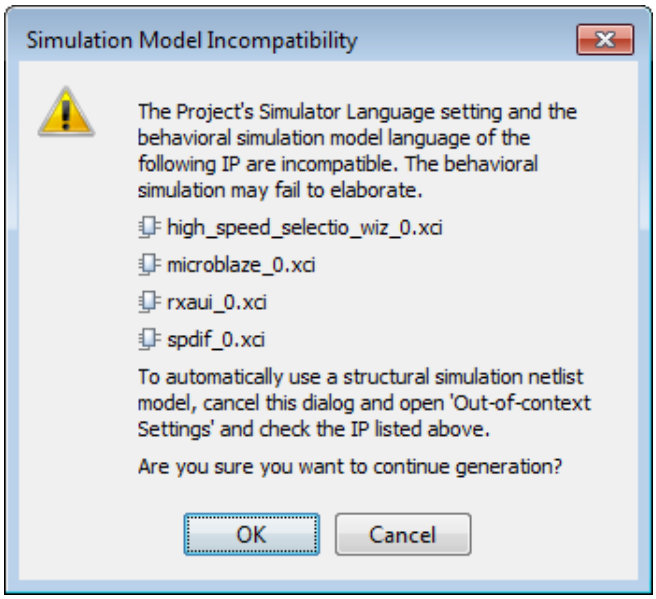


Figure 2-38: **Simulation Model Incompatibility Dialog Box**

Tcl Commands for Simulation

To specify the simulator language:

```
set_property SIMULATOR_LANGUAGE <language_option>
[current_project]
```

Table 2-2: **Simulator Language Property**

Simulation Model	Language	Simulation Model
IP delivers VHDL and Verilog behavioral models	Mixed	Behavioral simulation model provided in the specified `SIMULATOR_LANGUAGE`.
	Verilog	Verilog behavioral model.
	VHDL	VHDL behavioral model.
IP delivers Verilog behavioral model only	Mixed	Verilog behavioral model.
	Verilog	Verilog behavioral model.
	VHDL	VHDL simulation netlist generated from the IP DCP.
IP delivers VHDL behavioral model only	Mixed	VHDL behavioral model.
	Verilog	Verilog simulation netlist generated from the IP DCP.
	VHDL	VHDL behavioral model.
IP deliver no behavioral models	Mixed/Verilog/VHDL	Structural simulation netlist generated from the DCP in the specified `SIMULATOR_LANGUAGE`.

Note: Where available, a *Behavioral Simulation* model always takes precedence over a *Structural Simulation netlist.* Vivado does not offer a choice of simulation model.

Note: The setting for the project property `SIMULATOR_LANGUAGE` is used to determine the simulation models delivered when the IP supports both Verilog and VHDL.

Using a Test Bench for IP

Many IP in the IP Catalog also deliver a test bench for simulating the IP standalone. If an IP delivers a test bench, you see it listed as an output product in the Generate Output Product dialog box, as shown in the following figure.

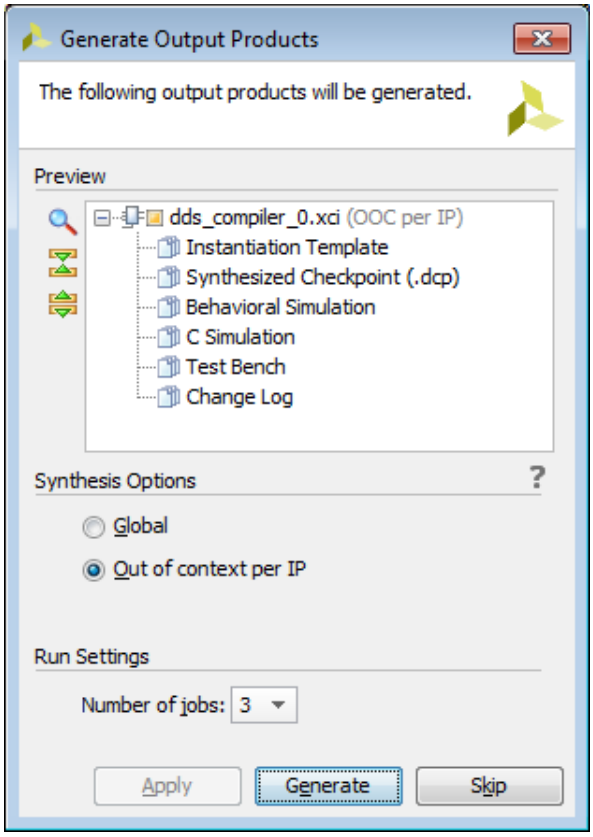


Figure 2-39: **Test Bench Output Product**

To use the test bench provided by the IP, do the following:

1. In the Sources window, find the IP in the hierarchy in the Simulation Sources section and expand the IP hierarchy. You do this by pressing one of the following:
   - Circle (Linux)
   - **Add** button (Windows) next to the IP in the `sim_1` set.

   The Show IP Hierarchy dialog box displays as shown in the following figure.

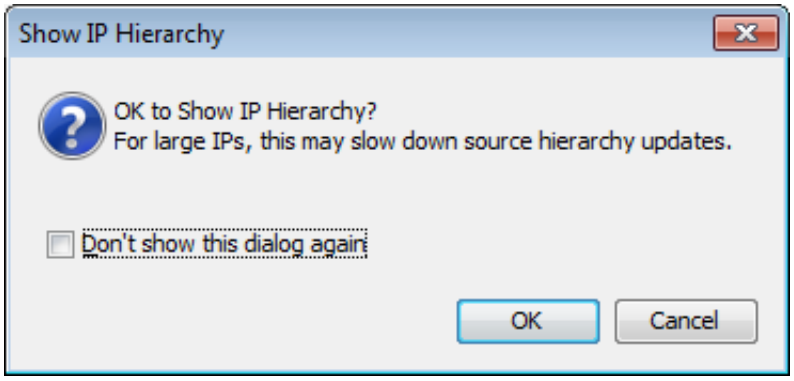


Figure 2-40: **Expanding the IP hierarchy**

2. Click **OK** to expand the hierarchy.
3. Find and select the IP test bench, named `tb_<ip_name>`.
4. Right-click and select **Set as Top**.

   You could get an Invalid Top Module dialog box. If so, select the **Ignore and continue with invalid top module** and click **OK**, as shown in the following figure.

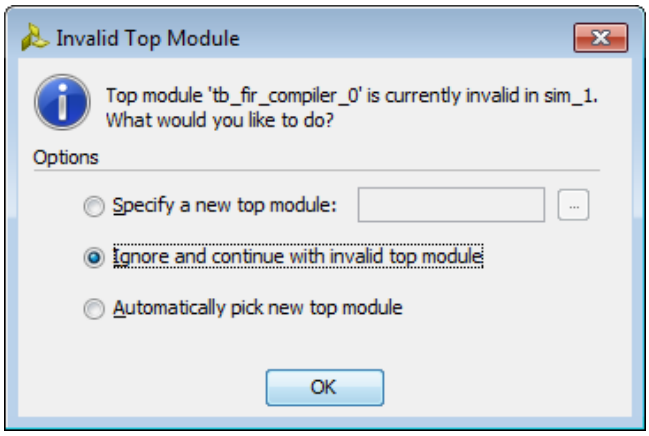


Figure 2-41: **Invalid Top Module**

TIP: *Setting the IP Test Bench as the top module can result in the IP synthesis run being set out-of-date because a project level property has changed. You can force the run up-to-date by right-clicking and selecting **Force Up-to-Date**, as shown in the following figure. If you do not do this, the next time you launch top-level synthesis the IP re-synthesizes.*

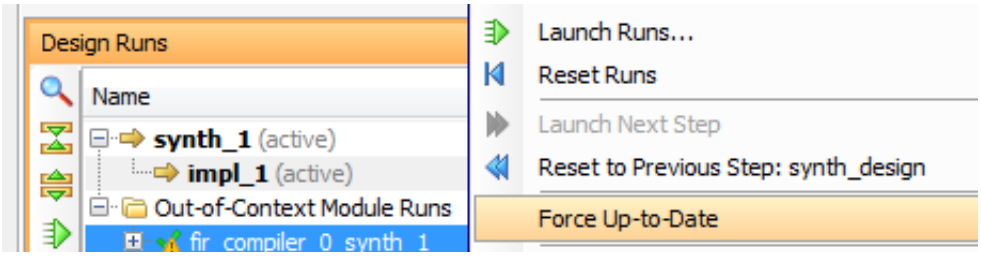


Figure 2-42: **Force IP Synthesis Run Up-to-Date**

In the **Flow Navigator** select **Run Simulation** or use the `launch_simulation` command in the Tcl Console to launch the simulator on the new top-level of the design, which is the simulation test bench for the IP.

Note: You can look at the current simulation settings in the **Project Settings** in the Simulation section. Here you see the simulation top-module name, which should match the IP test bench that you set. You can also change the simulator setting here, which affects the behavior of the **Run Simulation** button.

Upgrading IP

Each release of the Vivado Design Suite delivers only one version of an IP. New releases and patches of the Vivado Design Suite might include newer versions of IP in the Vivado IP Catalog that are used in your existing projects. In this case, the IP in your current projects become *locked*, and must be upgraded to let you use the latest version of the IP.

VIDEO: *See the Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades for a demonstration of upgrading IP.*

Prior to moving to a new Vivado Design Suite release, do one or more of the following:

- Generate all the output products for the IP in your project, including the DCPs. This lets you use the old version of the IP in the new release of the Vivado Design Suite, if needed.

  Note: You cannot generate output products, including DCPs, for IP that is not the *current version* for the release.

- If you are using Manage IP projects, copy the entire Manage IP project location as a backup.
- Archive design projects that contain IP.

Before upgrading an IP, view the change log for information on the changes.

IMPORTANT: *It is especially important for IP that have a major revision change between Vivado Design Suite releases because these IP typically require RTL changes.*

When you upgrade a project from a previous version of Vivado, and an upgrade is available for an IP, used in that project, the Project Upgraded dialog box opens informing you that an upgrade is available for the IP, as shown in the following figure.

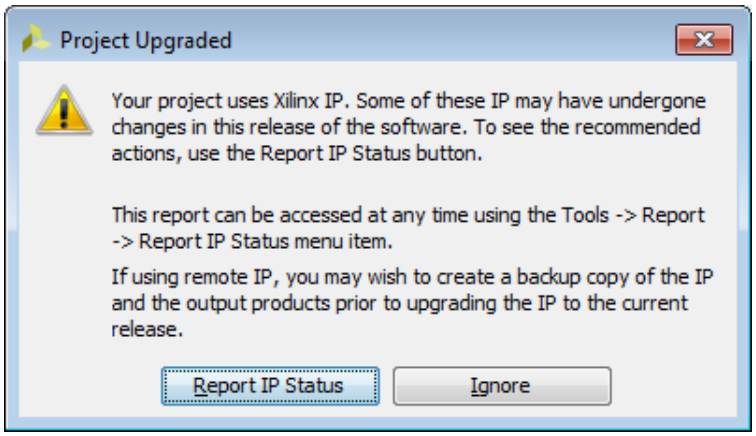


Figure 2-43: **Project Upgraded Dialog Box**

It is recommended that you select **Report IP Status** to see which IP have an upgrade and to view the change logs. The following figure shows the IP Status window that opens when you run the **Report IP Status** command.

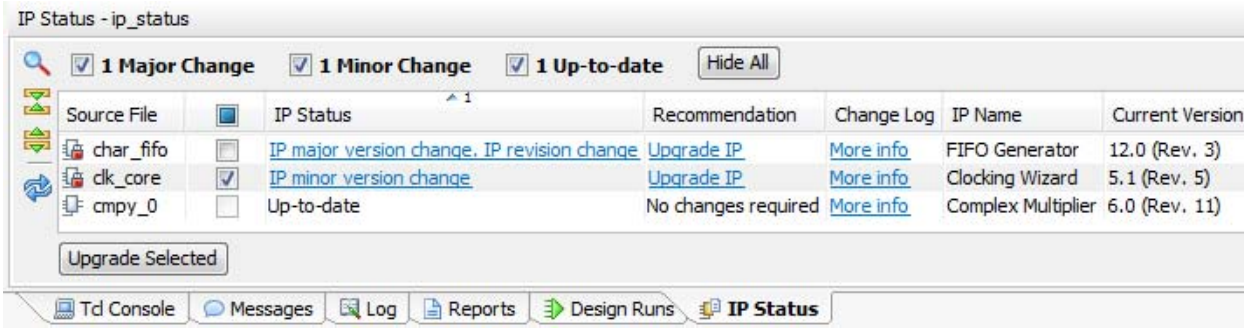


Figure 2-44: **Report IP Status with Upgrade IP Option**

Click the **Upgrade IP** option next to the IP.

he report provides information about all the IP in the project. All the blue, underlined text can be clicked to provide more details. The information includes:

- **IP Status**: This shows if the IP is up-to-date, or if there is a minor or major version change. Other possibilities are a part change.
- **Recommendation**: Lists the recommendation action
- **Change Log**: By selecting the **More info** link you can view the change log for the IP. The change log provides information on the latest release of the IP. It is recommended you review the change log before upgrading the IP. Major version changes could require modification to the RTL that connects to the IP. The change log contains the following:
  - **IP Name**: Name of the IP as shown in the IP Catalog.
  - **Current Version**
  - **Recommended Version**
  - **License**: Shows the status of the IP license.
  - **Current Part**

You can check the box next to the **Source File** to selective upgrade IP. Checking the box in the column will select all IP that have an upgrade available. Click **Upgrade Selected** to upgrade the selected IP.

By default, upgrading IP results in the upgrade information stored in the `ip_upgrade.log` file which is in the project directory with the XPR file.

See the following for more information on upgrading IP:

- This link to the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 6] for upgrading IP in a block design (BD).
- This link to the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 4] for information regarding editing a packaged BD.

Upgrading an IP in a design creates an *update log*. These logs are available from the `IP/Change Log` folder in the Sources window, as shown in the following figure.

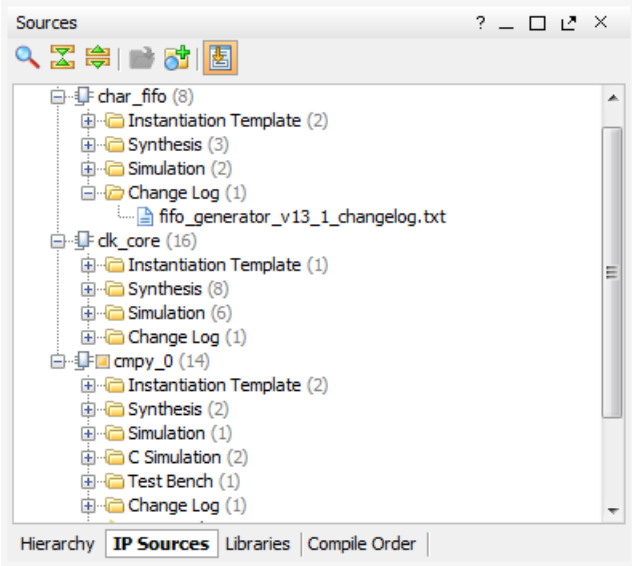


Figure 2-45: **Change Log for an IP**

This log contains information about the IP upgrade, such as:

- If the upgrade is successful
- If user intervention is required

  Note: If the upgraded IP requires user intervention, the log lists the issues encountered, such as parameters that no longer exist.
- The original version and revision of the IP
- The upgraded version and revision of the IP
- Additional information, as appropriate:
  - A description of changes made by the upgrade script.

    Examples of such information are: "`Renamed parameter DATA_W to C_DATA_WIDTH`", and "`Set parameter BUFFER_LENGTH to value 32`").
  - Warnings issued during customization of the IP, such as ports added, changed, and removed during an upgrade.

- Warnings associated with the upgrade; either general issues relating to the upgraded version, or specific issues related to the current parameterization.
- Any warnings issued during customization of the IP.

CAUTION! *When upgrading an IP, all previously generated output products are removed, including the DCPs and any associated design runs. As a precaution, archive the project prior to upgrading the IP.*

Upgrading IP using a Tcl Command

You can use the `upgrade_ip` command to upgrade all specified IP. For example, to upgrade all IP in the design, type upgrade_ip in the Tcl Console, as follows:

```
upgrade_ip
```

The IP will be upgraded, though no upgrade log is created. Add the `-log` option to specify an upgrade log.

CAUTION! *Do not use* `upgrade_ip [get_ips -all]`*; this can cause issues with Vivado. The -all option returns sub-core IP. These IP might get removed during an upgrade of the parent and can lead to unreferenced Tcl objects.*

To upgrade an IP, and create a log file for that IP, type the following in the Tcl Console:

```
upgrade_ip [get_ips cfifo] -log c:/prj/IP/cfifo_upgrade.log
```

The upgrade log is not overwritten for each IP upgrade. The latest IP that was upgraded has the upgrade information added to the top of the file.

Understanding Hierarchical IP

Some IP are designed to use other IP as design sources. Depending on how the parent IP was created there can be out-of-context synthesis runs for the children IP.

The following are types of IP with a parent-child relationship:

- IP which reference another IP as a library of files (subcore reference)
- IP which are packaged with XCI files for child IP (static IP)
- IP which dynamically create child IP and HDL (dynamic IP)
- IP which use IP Integrator technologies to dynamically create and interconnect IP (subsystem IP)

The first type of hierarchical IP, those that use sub-core references as covered in *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 4], have one out-of-context synthesis run. In the IP Sources there will be no sub IP shown, there is only one XCI.

The second type of hierarchical IP, static IP are those that were packaged with XCI files for other IP, and have one out-of-context synthesis run because all the IP are synthesized together. Looking at the IP Sources, you see the XCI for the child IP with output products in the `Synthesis` folder for the parent IP, as shown in the following figure.

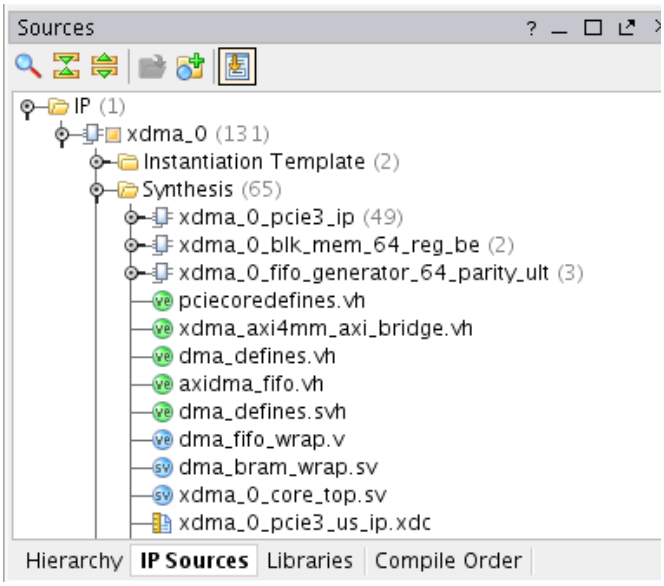


Figure 2-46: **Output Products for Hierarchical IP Using XCI files**

The third type of hierarchical IP, those which dynamically create children IP (dynamic IP), have one out-of-context synthesis run as all the IP are synthesized together. Similar to the static IP, you see multiple XCI in the IP sources.

The fourth type of hierarchical IP (subsystem IP), are the IP that the IP integrator technology creates. These IP have out-of-context synthesis runs for the children IP as shown in the following figure. For example, the 10G Ethernet Subsystem IP is a block design which consists of multiple IP, including another Hierarchical IP, 10G Ethernet PCS/PMA, which in turn has additional out-of-context runs under it for its children IP. In the IP Sources, you see the block design and children XCI also.

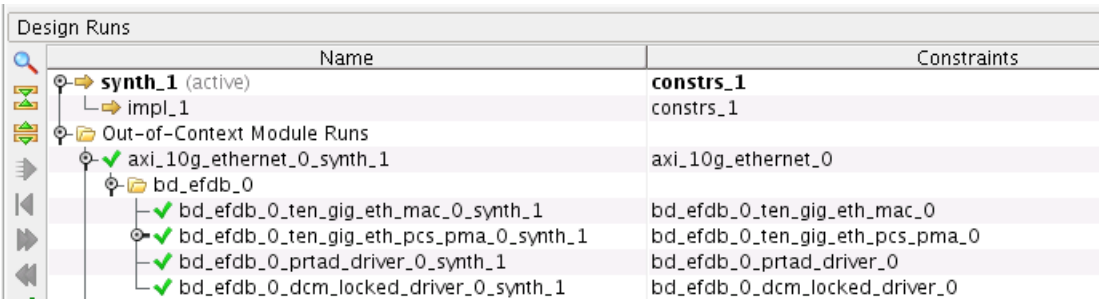


Figure 2-47: **Hierarchical IP with Out-of-Context Synthesis Runs**

When viewing the synthesis log for the 10G Ethernet Subsystem IP you see that black boxes are inferred for the child IP. This is similar to the default flow of IP in a user design during synthesis. During out-of-context synthesis per IP, after the synthesis of the children IP is completed, they are linked together to create the combined DCP for the IP, as shown in the following figure. In this way hierarchical look just like other IP.

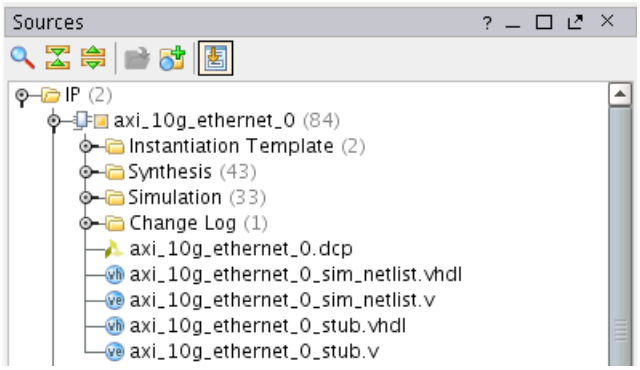


Figure 2-48: **Output Products for Hierarchical IP**

The primary benefit of hierarchical IP based upon block design is that when generating the output products the children IP out-of-context runs are launched in-parallel.

The number of parallel runs is specified in the Run Settings in the Generate Output Products dialog box, as shown in the following figure.

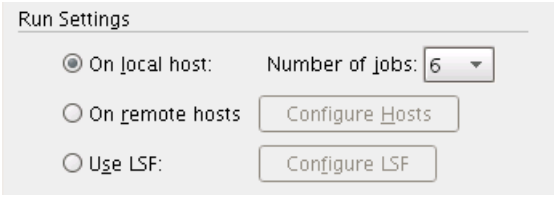


Figure 2-49: **Run Settings for Hierarchical IP**

Additionally, if IP Caching is enabled, the hierarchical IP can have cache hits for the children IP. These can greatly speed-up generation.

Another benefit of hierarchical IP, based upon block designs, is that when re-customizing the IP changes might only affect one or two of the children IP. When regenerating the output products, only the children IP that require re-synthesis is re-launched, thus generation is faster.

Working with Debug IP

The Vivado Design Suite includes features to let you perform in-system programming and debugging of the post-implemented design in a device. The benefits of debugging your design in-system include debugging your timing-accurate, post-implemented design in the actual system environment running at system speeds.

You can use the Vivado Lab Edition tools to test and verify the IP capabilities when attached to a Xilinx board with a JTAG connection. The available debug IP cores include:

- **Vivado Integrated Logic Analyzer**: The integrated logic analyzer (ILA) also called *Vivado logic analyzer*, lets you perform in-system debugging of post-implemented designs on an FPGA.

  Use this feature when you need to monitor signals in a design. You can also use this feature to trigger on hardware events and capture data at system speeds. You can instantiate the ILA core in your RTL code or insert the core, post-synthesis, in the Vivado design flow.

- **Vivado Virtual I/O Analyzer**: The virtual input/output (VIO) debug feature, also called the *Vivado serial I/O analyzer* can both monitor and drive internal FPGA signals in real time. In the absence of physical access to the target hardware, you can use this debug feature to drive and monitor signals hat are present on the real hardware.

  This debug core must be instantiated in the RTL code; consequently, you need to know what nets to drive.

- **IBERT Serial Analyzer**: The integrated bit error ratio tester (IBERT) serial analyzer enables in-system serial I/O validation and debug. This allows you to measure and optimize your high-speed serial I/O links in your FPGA-based system.

  Use the LogiCORE IBERT Serial Analyzer when you are interested in addressing a range of in-system debug and validation problems from simple clocking and connectivity issues to complex margin analysis and channel optimization issues.

  Using this core you can measure the quality of a signal after a receiver equalization is applied to the received signal. This ensures that you are measuring at the optimal point in the TX-to-RX channel and thereby real and accurate data. You can access this design by selecting configuring, and generating the IBERT core from the IP catalog and selecting the **Open Example Design** feature of this core.

- **JTAG to AXI**: The JTAG-to-AXI debug feature generates AXI transactions that interact with various AXI4 and AXI4-Lite slave cores in a system that is running in hardware.

  Use this core to generate AXI transactions and debug and to drive AXI signals internal to an FPGA at run time. You can use this core in IP designs without processors as well. The IP Catalog lists the core under the **Debug** category.

  See the following documents for more information:

  - *JTag to AXI LogiCore IP Product Guide* (UG174) [Ref 38]
  - *Integrated Logic Analyzer LogiCORE IP Product Guide* (PG172) [Ref 32]
  - *IBERT 7 Series GTX Transceivers LogiCORE IP Product Guide* (PG132) [Ref 33]
  - *IBERT 7 Series GTP Transceivers LogiCORE IP Product Guide* (PG133) [Ref 34]
  - *IBERT 7 Series GTH Transceivers LogiCORE IP Product Guide* (PG152) [Ref 36]
  - *Vivado Design Suite Tutorial: Programming and Debugging* (UG936) [Ref 10]
  - *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 11]
  - *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) [Ref 12]

Debugging Flows

The Vivado tools provide several methods to add debug probes into your design. You need to determine which flow suits the requirements of your design. The available debug flows are:

HDL instantiation debug probing flow: This flow involves explicitly adding debug IP cores into your HDL design, and attaching signals in the HDL source to an ILA debug probe. See this link to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 11] for more information on the HDL insertion flow.

There are advantages and disadvantages to this flow, as follows:

- Advantage: Provides the ability to probe at the HDL design level.
- Disadvantages:
  - You must manually add and remove debug nets and IP in your design, by modifying your HDL source.
  - Easy to make mistakes when generating, instantiating, and connecting debug cores.

Netlist insertion debug probing flow (Recommended): this flow involves explicitly attaching signals in the synthesized netlist to an ILA debug core instance:

  - Use the `MARK_DEBUG` attribute to mark signals for debug in the source RTL code.
  - Use the `MARK_DEBUG` right-click menu option to select nets for debugging in the synthesized design netlist.

    The netlist insertion flow uses the Set up Debug wizard guides you through the process of adding debug cores and probing signals of your design.

- Advantages:
  - Most flexible with good predictability.
  - Allows probing at different design levels (HDL, synthesized design, system design).
  - Does not require HDL source modification.
- Disadvantages: Cannot be used for IBERT or JTAG-to-AXI Master cores.

Tcl-based netlist insertion flow: Use the `set_property` Tcl command to set the `MARK_DEBUG` property on debug nets then use the following Tcl commands: `create_debug_core`, `create_debug_port`, and `connect_debug_port` to add debug cores and probes to your synthesized design.

Using Manage IP Projects

Introduction

The Vivado® Integrated Design Environment (IDE) provides mechanisms for:

- Exploring IP in the IP Catalog
- Customizing IP
- Managing centralized location of customized IP.

The Vivado IDE can create a special project for managing customizations and output products of specified IP, referred to as a *Manage IP Project*. From the Manage IP Project, you can view the IP catalog, customize IP, and generate output products. The IP customization (XCI) and generated output products are stored in separate directories located outside of the Manage IP project. The Manage IP project manages the IP design runs for the generation of the synthesized design checkpoint (DCP) files and other output products. Customized IP, with all of the output products generated, can be used as configured in multiple designs. See Adding Existing IP to a Project.

When working in teams, or if the design uses many Xilinx® IP, create and maintain your customized IP in a location outside of the Vivado project structure. This method makes revision control more straightforward and allows for ease of sharing customized IP with others. This is also the recommended methodology for working with IP in a non-project, script-based flow.

Managed IP Features

When you use the Manage IP flow in the Vivado IDE, the following features are available:

- Simple IP project interface
- Direct access to the Xilinx IP Catalog
- Ability to customize multiple IP
- Separate, unique directories for each IP customization with all related IP files
- Option to generate or skip generation of the design checkpoint (DCP) file. A DCP file consists of both a netlist and constraints for the IP, and creating this file is the default flow.

VIDEO: *Also, see the following:* Vivado Design Suite QuickTake Video: Configuring and Managing Reuseable IP in Vivado *and the* Vivado Design Suite QuickTake Video: Working with Design Checkpoints.

Using the Manage IP Flow

1. Invoke the Vivado IDE, and from the **Getting Started** page, select **Manage IP**, as shown in the following figure.

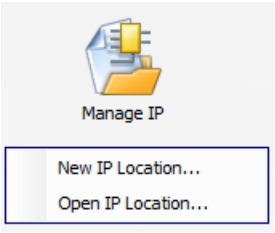


Figure 3-1: **Invoking the Manage IP Flow**

2. Open a new or an existing IP location.
   - **New IP Location**: Opens a new IP project at the location specified for exploring the IP catalog and customizing IP, including generation of output products.
   - **Open IP Location**: Lets you navigate to an existing location from which to open an IP.
   - **Recent IP Locations:** The right side of the Getting Started page lists recently open locations for Manage IP Projects

When you select the **New IP Location** option, the **Create a New Customized IP Location** menu informs you that a wizard will guide you through creating and managing a new customized IP location, as shown in the following figure.

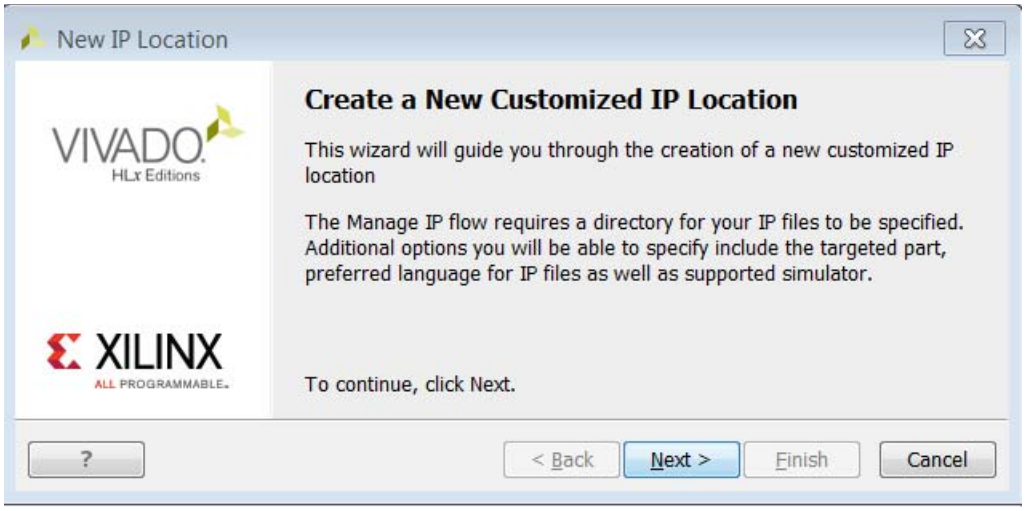


Figure 3-2: **Create a New Customized IP Location**

If the location specified to create a new manage IP project already contains a project, a dialog box opens (see the following figure).

You can either choose to open the existing project or cancel.

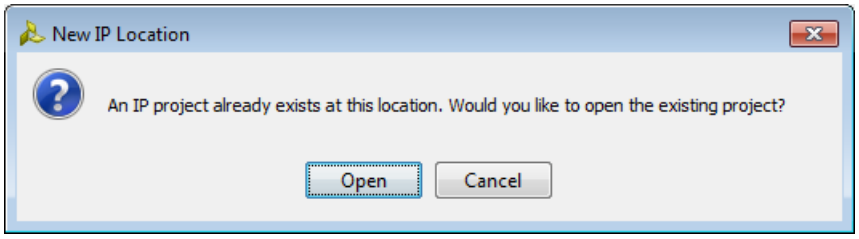


Figure 3-3: **New IP Location Dialog Box**

3. Select **Next**.

   The Manage IP Settings dialog box opens, as shown in the following figure.

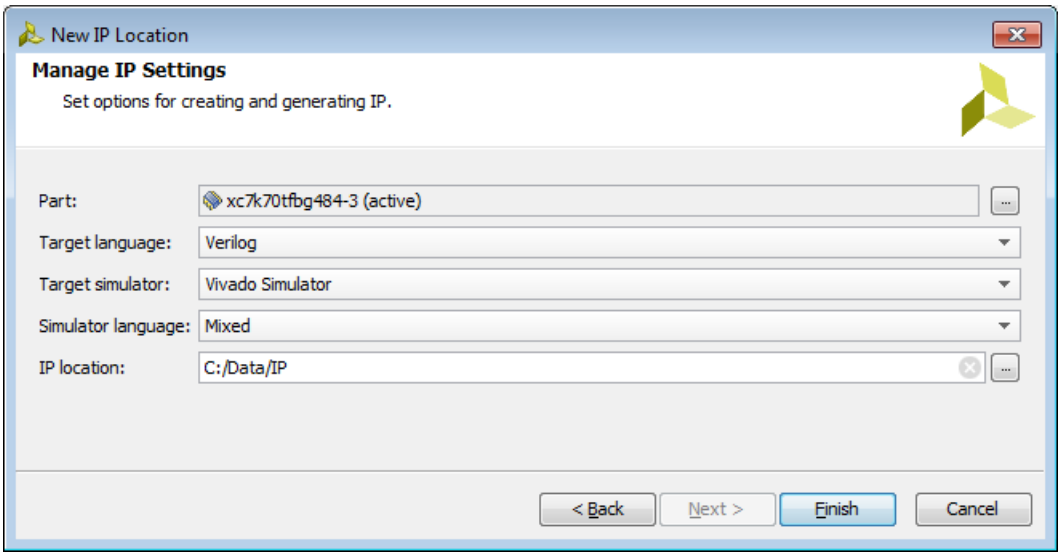


Figure 3-4: **Manage IP Settings Dialog Box**

Managing IP Settings

To manage the IP settings:

1. Enter the following:
   - **Part**: Select the active part. All output products generated for the IP will be based on the specified part.
   - **Target language:** Set the target language to VHDL or Verilog depending on your design top.
   - **Target Simulator**: Specify the simulator to use as either the Vivado simulator, or one of a number of third-party simulators.
   - **Simulator language:** Options are **VHDL**, **Verilog**, **SystemVerilog**, or **Mixed** depending on the license that you have available for the simulator. For the Vivado simulator, the default is **Mixed**.
   - **IP location**: The location where the Vivado IDE creates the `managed_ip_project` directory.
2. Click **Finish**.

   Vivado IDE opens the Managed IP project, and you can now select and customize IP. You have access to the full IP Catalog, including IP Product Guides, Change Logs, Product web pages, and Answer Records.

After you customize an IP, the Sources and Properties windows display, providing information about the IP created in the project.

Each IP customization has a directory created under the specified manage IP location. This directory contains the XCI file and any generated output products. The following figure shows the Manage IP Project window where you customize and manage multiple IP.

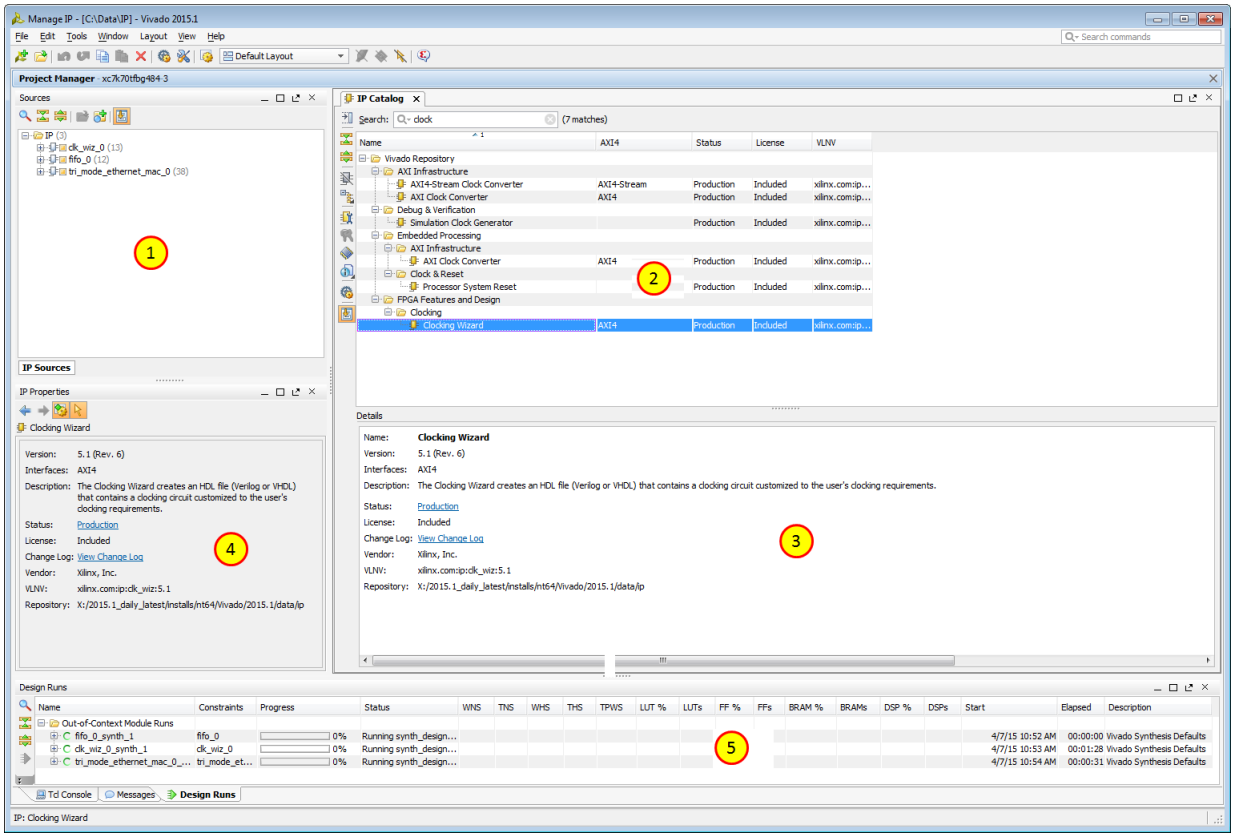


Figure 3-5: **Manage IP Project Window Containing Three IP**

Using IP Example Designs

Introduction

Many Xilinx® IP deliver an example design project. The example design project consists of top-level logic and constraints that interact with the created IP customization. These example designs typically come with an example test bench that helps simulate the design.

TIP: *Rather than upgrading an existing example design with the latest IP in a new release, create a new example design from the IP in the new release. This ensures that the example design is tuned to support the latest version of the IP.*

Opening an Example Design

To open an example design project for an IP in either a standard project or a Manage IP project, select the IP customization in the IP Sources tab, then right-click and select **Open IP Example Design** from the context menu.

This opens the **Open IP Example Design** dialog box for you to specify the location, as shown in Figure 4-1. The project is called `<ip_name>_ex`.

IMPORTANT: *Do not store example designs in the IP directory in either a standard project or a Manage IP project. Xilinx recommends putting the entire IP directories into revision control, and also recommends that you do not put Projects into revision control. This can also cause issues when enabling and disabling core containers.*

This can also be accomplished for IP in an IP integrator block design, either by selecting the IP in IP Sources or directly from a block design.

The following figure shows the Open IP Example Design dialog box.

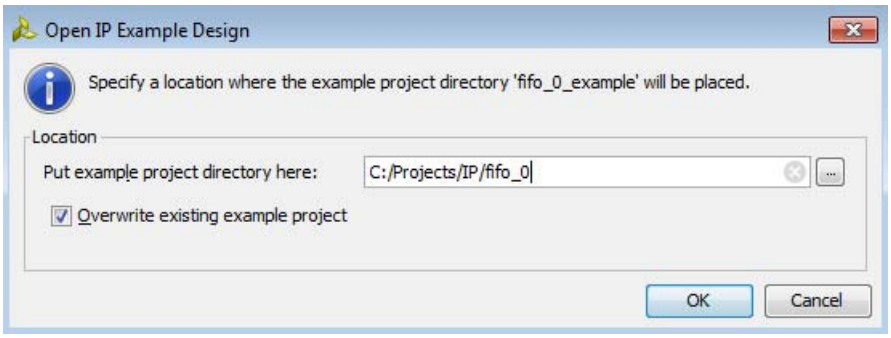


Figure 4-1: **Open IP Example Design Dialog Box**

A new session of the Vivado IDE opens with the example design, shown in the following figure.

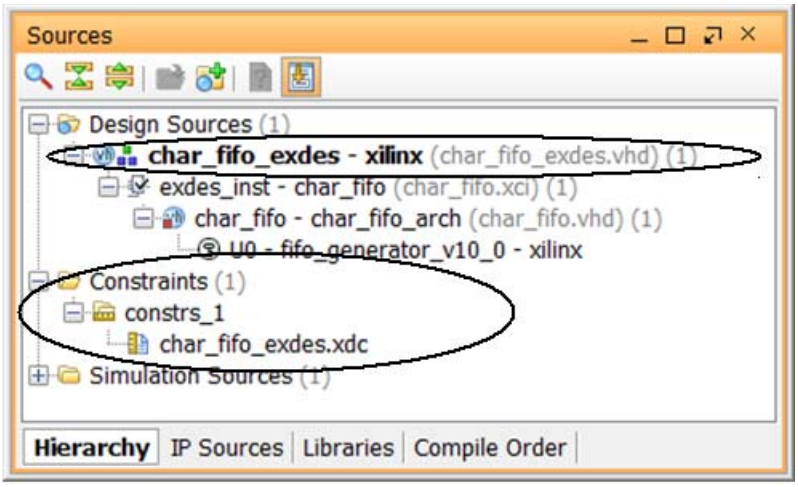


Figure 4-2: **IP Example Design Instance with Constraint File**

The IP is instantiated in the example design with an example XDC constraint file to enable further evaluation of the IP.

Tcl Command to Open a Project

Alternatively, you can use the open_example_project Tcl command to open a project:

```
open_example_project [get_ips <ip_name>]
```

Examining Standalone IP

After implementation completes, right-click the IP design run, and select **Open Implemented Design** to open the design for analysis, as shown in the following figure.

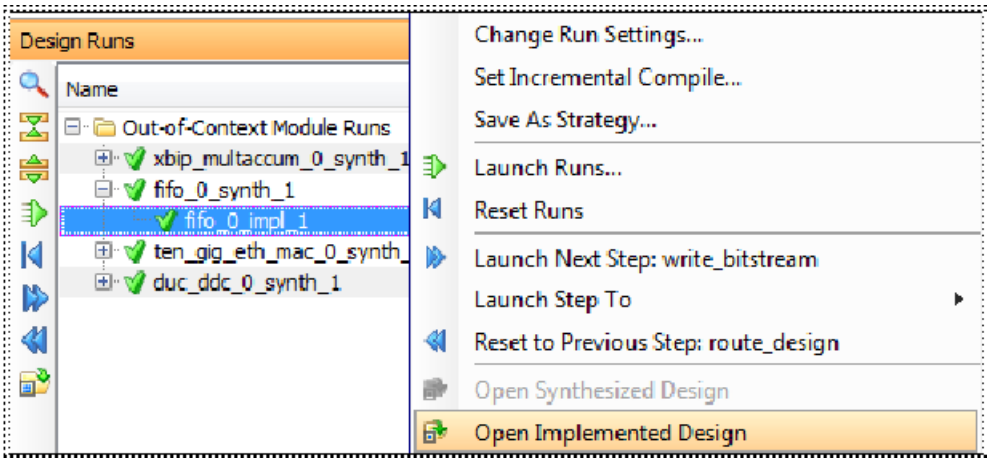


Figure 4-3: **Opening an Implemented IP**

When performing timing analysis, the results are not accurate because the clocks are not yet routed and ideal clocks are used. This is most obvious when performing hold analysis, because the router cannot fix hold violations.

Some IP include the `HD.CLK_SRC` property in the `<ip_name>_ooc.xdc` file, which provides a location to a clock buffer and the SLEW timer models to improve the accuracy of post-implementation timing analysis.

IMPORTANT: *The implemented IP is for analysis only, and the results are not used or preserved during implementation of the top-level design. For information on module reuse and using an implemented version of the IP, see the Vivado Design Suite User Guide: Hierarchical Design* (UG905) [Ref 16].

Chapter 5

Using Xilinx IP with Third-Party Synthesis Tools

Introduction

Xilinx® supports netlists created by third-party synthesis tools for user logic. When using Xilinx IP the only supported synthesis tool is the Vivado® synthesis tool. The Vivado Design Suite infers a black box during logic synthesis for Xilinx IP. A file is provided to infer the black box as described in the following sections. The Vivado Design Suite resolves the black boxes during implementation. Synthesis of Xilinx IP is supported with the Vivado synthesis tool only, including the IP core and any example design files an IP might deliver.

VIDEO: *See the* Vivado Design Suite QuickTake Video: Using IP with Third-Party Synthesis Tools *for more information.*

IMPORTANT: *Xilinx encrypts IP HDL files with the IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP) (IEEE Std P1735). Consequently, IP HDL files are readable only when using the Vivado Design Suite for synthesis. You can use a third-party synthesis tool for the end-user logic and generate a netlist that Vivado implementation can use. Support is enabled for third-party simulation tools to perform behavioral simulations using the encrypted RTL.*

Third-Party Synthesis Flow

When using a Synopsys® Synplify Pro or Mentor® Graphics Precision netlist for synthesis of a design that has Xilinx IP, the recommended flow is:

1. Use the **Manage IP** flow in to create and customize IP.
2. Generate output products for the IP, including the synthesis design checkpoint (DCP) for each IP.

   When you generate the DCP file, stub files are created to infer a black box when used with the third-party synthesis tool: `<ip_name>_stub.v` and `<ip_name>_stub.vhdl`.

3. Add the Verilog or the VHDL stub file to the project for use by the third-party synthesis tool. The Verilog or VHDL stub file infers a black box during synthesis and also prevents the synthesis tool from adding I/O buffers.

   The `<ip_name>_stub.v` and the `<ip_name>_stub.vhdl` contain synthesis directives that prevent the third-party synthesis tool from inferring I/O buffers for the IP if the IP connects to top-level ports. You can change these directives as required for use with third-party synthesis tools.

4. Generate a netlist for your top-level design with the third-party synthesis tool.

   Note: See the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 18] for more information about netlist projects.

5. Create a Vivado netlist project to place and route the top-level design, and generate the bitstream for the device.

   You can also create an RTL project for the design, and encapsulate the EDIF netlist from the third-party synthesis tool in a wrapper.

   a. Create an HDL wrapper around the EDIF netlist produced by the third-party synthesis tool.

   b. Select the hierarchy sources view (HSV).

   c. Right-click and select **Hierarchy Update**, then check the **No Update, Manual Compile Order** option.

6. Add the following into the Vivado netlist project:

   - The netlist from the third-party synthesis tool.
   - User-level top-level design constraints.
   - The XCI files for the IP (one XCI file per IP).

   The netlist in the IP DCP as well as the XDC output products are used automatically during implementation when using the XCI file for the IP.

7. Implement the design.

   Vivado implementation adds any required I/O buffers if they are not already present in the DCP of the IP.

RECOMMENDED: *Use the IP XCI file when referencing Xilinx IP in either Project Mode or Non-Project Mode and not the DCP file directly. While the DCP does contain constraints, they are resolved Out-Of-Context of the end-user constraints. Using the XCI results in the XDC output product for the IP being applied after all the netlists are combined (end-user and IP). Additionally, any Tcl script in the IP XDC is then evaluated in context of the end-user constraints and netlist.*

Example Tcl Script for Third-Party Synthesis in Project Mode

```
# Create a project on disk
create_project <name> -part <part>

# configure as a netlist project
set_property design_mode "GateLvl" [current_fileset]

# Add in the netlist from third-party synthesis tool
add_files top.edif

# Add in XCI files for the IP
add_files {ip1.xci ip2.xci ip3.xci}

# Add in top level constraints: this might include XDC files from the third-party
# synthesis tool
add_files top.xdc
# Launch implementation
launch_run impl_1 -to write_bitstream
```

Example Tcl Script for Third-Party Synthesis in Non Project Mode

```
# Set target part
set_part <part>

# Read the netlist from third-party synthesis tool
read_edif top.edif

# Read in the IP XCIs
read_ip ip1.xci
read_ip ip2.xci

# read in top level constraints
read_xdc top.xdc

# Implement the design
link_design -top <top>
opt_design
place_design
phys_opt_design
route_design
write_bitstream -file <name>
```

Note: Ensure that, when reading in the IP, you are reading the XCI file from the location where the output products of the IP were previously generated or alternatively, read in the XCI file and then generate the IP using the `synth_ip` command.

Tcl Commands for Common IP Operations

Introduction

This chapter covers the Tcl commands to use for common IP operations.

For more information about using Tcl and Tcl scripting, see the following:

- *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 22]
- *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 14]

For a step-by-step tutorial that shows how to use Tcl in the Vivado tool, see the *Vivado Design Suite Tutorial: Design Flows Overview* (UG888) [Ref 21].

IMPORTANT: *When associating a file using a Tcl command, ensure that the path to the file is an absolute path and not relative.*

Using IP Tcl Commands In Design Flows

Generally, the IP Tcl commands used for working are consistent between the Project Mode flow and the Non-Project Mode flow with a few exceptions related to setting the part to be used for IP creation and synthesis. The following table lists the Tcl command in the order that you would use them in a design.

Table 6-1: **IP Tcl Commands in Order of Design Use**

Action	Project Mode Command	Non Project Mode Command

Set part for IP creation	N/A. Part is a project setting.	`# Set target part`
		`set_part <part>`
		Note: Creates project in memory, not on disk. Commands that have a part option, for example, `synth_design`, then use the specified part.

Create an IP Customization	create_ip `<ip_name>`	`create_ip <ip_name>`

Table 6-1: **IP Tcl Commands in Order of Design Use *(Cont'd)***

Action	Project Mode Command	Non Project Mode Command
Upgrade an IP	upgrade_ip <ip_name>	`upgrade_ip <ip_name>` **CAUTION!** *Do not use `upgrade_ip [get_ips -all]`; this can cause issues with Vivado. The `-all` option returns sub-core IP. These IP might get removed during an upgrade of the parent and can lead to unreferenced Tcl objects.*

Configure IP Customization	`set_property \`
	`CONFIG.Input_Data_Width 8 \`
	`[get_ips <ip_name>]`
		`set_property \`
		`CONFIG.Input_Data_Width 8 \`
		`[get_ips <ip_name>]`

Create a target Clock Period	If supported, use IP Customization GUI. If not supported use the Tcl command as shown is Non Project mode.	`set_property \`
		`CONFIG.<clock_name>.FREQ_HZ \`
		`<#>[get_ips char_fifo]`
		See Setting a Target Clock Period using Tcl Commands.

Generate output products	`generate_target \`
	`[get_ips <ip_name>]`
	Note: Optionally, you can specify the target(s) you want to generate.
		`generate_target \`
		`[get_ips <ip_name>]`
		Note: Optionally, you can specify the target(s) you want to generate.

Synthesize IP to create OOC DCP	create_ip_run \
	`[get_ips <ip_name>]`
	`launch_runs`
	`<ip_name>_synth_1`
		`synth_ip [get_ips <ip_name>]`

Read an IP	Copy an IP into a project along with any output products:
	`import_files <ip_name>.xci`
	Add the IP to a project along with any output products and reference from the specified location. Use either of the following:
	`add_files <ip_name>.xci`
	`read_ip <ip_name>.xci`
		Read the IP as well as any generated output products. Use either of the following:
		`add_files <ip_name>.xci`
		`read_ip <ip_name>.xci`
		Note: Unlike in the Project Flow, the output products are not generated automatically. You must generate them using the `generate_target` command.
		If you use the `synth_ip` command to produce a DCP for the IP, it is not necessary to generate the output targets first; those targets are generated automatically.

File queries	`get_files -of_objects \` `[get_ips <ip_name>]`	`get_files -of_objects \` `[get_ips <ip_name>]`
Simulation	See Simulating IP.	See Delivering IP Simulation Models.
Debug	See Debugging Flows.	

Tcl Commands for Common IP Operations

Within the Vivado IDE, the Vivado IP Catalog can be accessed from the Vivado IDE and the Tcl design environment.

To accommodate end-users that prefer batch scripting mode, every IP Catalog action such as IP creation, re-customization, output product generation, which is performed in the Vivado IDE, echoes an equivalent Tcl command into the `vivado.log` file; consequently, anything that you can do in the Vivado IDE you can script also.

The Vivado IP catalog provides direct access to IP parameter customization from the integrated Vivado IDE Tcl Console so you can set individual IP parameters directly from the Tcl Console.

The following are examples of common IP operations using Tcl commands:

Create a customization of the accumulator IP:

- ```
  create_ip -name c_accum -vendor xilinx.com -library ip  \
  -module_name c_accum_0
  ```
- Change customization parameter such as input and output widths:

  ```
  set_property -dict [list CONFIG.Input_Width {10}
  CONFIG.Output_Width {10}] [get_ips c_accum_0]
  ```
- Generate selective output products:

  ```
  generate_target {synthesis instantiation_template simulation} \
  [get_ips c_accum_0]
  ```
- Reset any output products generated:

  ```
  reset_target all [get_ips c_accum_0]
  ```

You can use a Tcl script to list the user configuration parameters that are available for an IP. by using either the `list_property` or `report_property` command and referencing the created IP.

The difference between these commands is:

- Return a list of objects which can be processed with a Tcl script as a list.

  ```
  list_property
  ```
- Return a text report giving the current value for each parameter, its type, and other parameters.

  report_property
  ```

- Get a list of all properties which apply to an IP:

```
list_property [get_ips fifo_generator_0]
```

- Get an alphabetized list of just the customization parameters you can augment this further:

```
lsearch -all -inline [ list_property [ get_ips fifo_generator_0 ] ] CONFIG.*
```

- Create a report listing all the properties for an IP, including the configuration parameters:

```
report_property [get_ips fifo_generator_0]
```

- Remove a design run:

```
delete_ip_run
```

Use this command to remove the design run for an IP. If you reset the output products for an IP in the Vivado IDE, two Tcl commands are issued: `reset_target` and `delete_ip_run`.

- Export a simulation:

```
export_simulation
```

For more information on the supported IP Tcl commands type, `help -category IPFlow` in the Tcl Console as shown in the following figure.

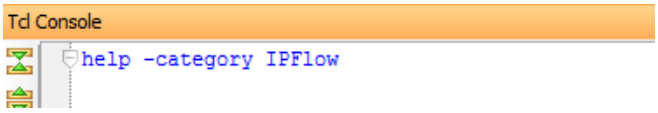


Figure 6-1: **Getting Help on IP Tcl Commands**

Note: The *Vivado Design Suite Tutorial: Designing with IP* (UG939) [Ref 18] contains labs that cover scripting of both Project Mode and Non-Project Mode flows with IP. They include examples of generating output products as well as selectively upgrading IP.

Example IP Flow Commands

This section provides Tcl script examples for some common operations.

Commands to Create IP

The `create_ip` command is used to create IP customizations.

RECOMMENDED: *Perform this operation as described in Using the Manage IP Flow, page 78. When you create IP with the Manage IP flow, you can subsequently use that IP in Project and Non-Project mode.*

The following script shows how to created a manage IP project, create and customize an IP, and generate a DCP:

```
# Create a Manage IP project
create_project <managed_ip_project> ./managed_ip_project -part <part> -ip

# Set the simulator language (Mixed, VHDL, Verilog)
set_property simulator_language Mixed [current_project]

# Target language for instantiation template and wrapper (Verilog, VHDL)
set_property target_language Verilog [current_project]

# Create an IP customization
create_ip -name c_accum -vendor xilinx.com -library ip -module_name c_accum_0

# configure the parameters for the IP customization
set_property -dict {CONFIG.Input_Width 10 CONFIG.Output_Width 10} [get_ips
c_accum_0]

# Create a synthesis design run for the IP
create_ip_run [get_ips c_accum_0]

# Launch the synthesis run for the IP
# Because this is a project, the output products are generated automatically
launch_run c_accum_0_synth_1
```

IMPORTANT: *Xilinx recommends project-based flows. Project-based flows can run in either the Vivado IDE or using the Tcl commands.*

Querying IP Customization Files

Tcl Script for Getting Files for Source Control

This example script shows how to get all files for a given IP customization. You can use this script to generate a list of files for use with a source control system.

```
# Create a project in memory, no project directory
# created on disk
create_project -in_memory -part <part>

# read an IP customization
read_ip <ip_name>.xci

# Generate all the output products
generate_target all [get_ips <ip_name>]

# Create a DCP for the IP
synth_ip [get_ips <ip_name>]

# Query all the files for this IP
get_files -all -of_objects [get_files <ip_name>.xci]
```

Note: See Table 6-1 for a more detailed explanation on these Tcl commands and when to use them.

Querying an Ordered Source List

When creating custom scripts, you can use one of the following Tcl commands:

- For IP Only: Synthesis

```
get_files -compile_order sources -used_in synthesis \
-of_objects [get_files <ip_name>.xci]
```

- For IP Only: Simulation

```
get_files -compile_order sources -used_in simulation \
-of_objects [get_files <ip_name>.xci]
```

- For Top-Level Design: Including IP For Synthesis

```
get_files -compile_order sources -used_in synthesis
```

- For Top-Level Design: Including IP For Simulation

```
get_files -compile_order sources -used_in simulation
```

Scripting Examples

Implementing an IP Example Design

Create a project to run implementation on an IP example design.

```
# Create a project
create_project <name> <dir> -part <part>

# Create an IP customization and a DCP
# This will also generate all the output products
create_ip ...
create_ip_run [get_ips <ip>.xci]
launch_runs <ip>_synth_1
wait_on_run <ip>_synth_1

# Open the example design for the IP

# This will use the IP DCP generated
open_example_project -force -dir "." -in_process [get_ips <ip>]
launch_runs synth_1
wait_on_run synth_1
launch_runs impl_1
wait_on_run impl_1 -to write_bitstream
open_run impl_1

# produce some reports
report_timing_summary ...
report_utilization ...
```

Non-Project Synthesis

Synthesize and implement design in a non-project flow with one IP which has an OOC DCP generated and one IP being synthesized along with user logic.

When reading an IP XCI file, all output products present, including an OOC DCP, are used, there is no need to generate them.

If the output products have not been generated for the IP, you must generate the output products (or create a DCP using the `synth_ip` command which generates the output products also).

If you elect to use global synthesis for an IP (see the Synthesis Options for IP in Chapter 2) then you must disable checkpoint support and generate the output products. This Tcl script provides a template for this.

```
#create an in memory project to provide the part to use for IP creation and for
#running synthesis

set_part <part>


# read in sources
read_verilog top.v

# Read in an existing IP customization
# or create an IP from scratch
# create_ip ... or read_ip ip1.xci

# Generate a DCP for the IP
# will generate output products if needed
synth_ip [get_ips ip1]

# Read in an existing IP customization
# or create an IP from scratch
# create_ip ... or read_ip ip2.xci

# Set IP to use global synthesis (no DCP generated)
set_property generate_synth_checkpoint false [get_files ip2.xci]

# Need to generate output products for IP
generate_target all [get_ips ip2]

# synthesis the complete design
synth_design -top top

# run implementation
opt_design
place_design
route_design

# write the bitstream
write_bitstream -file top
```

Simulating an IP Example Design

Create a project to run simulation on an IP example design.

```
#create the project
create_project <name> <dir> -part <part>

# create IP and a synthesis run
create_ip ...
create_ip_run [get_ips <ip_name>]

#launch runs
launch_runs <ip>_synth_1
wait_on_run <ip>_synth_1

#open the example project
open_example_project -force -dir "." -in_process [get_ips <ip>]

#launch simulation
<launch_simulation> | <target_simulator>
```

Synthesizing and Simulating an IP

If an IP does not deliver an example design, but does deliver a test bench, you can perform simulation of just the IP.

```
#create the project
create_project <name> <dir> -part <part>

# create_ip ... or add_files ip.xci

# create an IP design run
create_ip_run [get_ips <ip_name>]

#launch IP synthesis run
launch_run <ip>_synth_1
wait_on_run <ip>_synth_1

# Setting up simulation test bench
set_property top <tb> [current_fileset -simset]

# Launch simulation
<launch_simulation> | <target_simulator>
```

VIDEO: *See the* Vivado Design Suite QuickTake Video: Tcl Scripts and Constraint Files in Vivado *for more information.*

Appendix A

Determining Why IP is Locked

Introduction

IP cores become locked for several reasons. The Vivado IDE provides an IP status report that provides the reason and a recommendation.

The following table lists the locked IP messages and recommendations. See Reporting IP Status, page 55 for information on the IP Status Report.

IMPORTANT: *When working with Xilinx-delivered patches, you might notice IP locking due to changes in the IP definitions from the patch.*

Table A-1: **IP Locked Reasons and Recommendations**

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
IP file read-only	<ip_name> has a read-only file <IPXMLFILE>. IP is write-protected.	Check file and project permissions	Review your project and file system permissions causing the IP to be read-only. See Editing IP Sources for more information.
	<ip_name> has a read-only file <IPXCIFILE> with restricted functionality. Commands to change the configuration of this IP are disallowed.		
Shared output directory	<ip_name> shares a common output directory with other IP. Xilinx recommends that you place each IP in its own directory.	Move IP	• Manually remove the IP from the project with the remove_files command (see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 7]. • Import it back into the project with a unique directory import_files command.
IP definition not found	IP definition <CURRENT_IPDEF> for <ip_name> (customized with software release <SW_VERSION>) was not found in the IP Catalog.	Add IP definition to catalog	Consult the IP Catalog for the replacement IP. See Using the IP Catalog for more information.
IP major version change	IP definition <CURRENT_IPDEF> for <ip_name> (customized with software release <SW_VERSION>) has a newer major version in the IP Catalog.	Upgrade IP	Target IP definition <TARGET_IPDEF> requires a major version change. Review the impact on the design before upgrading the IP. See Upgrading IP for more information.
IP minor version change	IP definition <CURRENT_IPDEF> for <ip_name> (customized with software release <SWVERSION>) has a newer minor version in the IP Catalog.	Upgrade IP	Target IP definition <TARGET_IPDEF> requires a minor version change. Review the change log before upgrading the IP. See Upgrading IP for more information.
IP revision change	IP definition <CURRENT_IPDEF> for <ip_name> (customized with software release <SW_VERSION>) has a different revision in the IP Catalog.	Upgrade IP	Target IP definition <TARGET_IPDEF> requires a revision change. Review the change log before upgrading the IP. See Upgrading IP for more information.

Table A-1: **IP Locked Reasons and Recommendations *(Cont'd)***

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
Incompatible IP data detected	The IP Data in the repository is not compatible with the current instance (despite having identical Version and Revision). This typically occurs if you are using IP that is currently under development. You are not able to view the customization or generate outputs until it is updated.	Upgrade IP	Upgrade the IP. See Upgrading IP for more information.
IP unsupported part	IP <ip_name> does not support the current project part <CURRENT_PART>. However part differences can result in undefined behavior.	Unsupported Upgrade IP	Target IP definition <TARGET_IPDEF> does not support the current project part <CURRENT_PART>. Select a supported project part before upgrading the IP. See Appendix C, Using the Platform Board Flow for IP for more information.
IP license not found	IP <ip_name> requires one or more mandatory licenses but no valid licenses were found. However license checkpoints could prevent the use of this IP in some tool flows.	Unlicensed Upgrade IP	Target IP definition <TARGET_IPDEF> requires a valid license. Obtain a valid license before upgrading the IP. See Using Fee-Based Licensed IP for more information.
		Check IP license	IP <IP_Name> requires a valid license. Obtain a valid license or review your licensing environment. Using Fee-Based Licensed IP for more information.
IP board change (1)	This IP has board specific outputs. Current project board <CURRENT_BOARD> and the board <ORIGINAL_BOARD> used to customize the IP <ip_name> do not match.	Retarget IP	Change the project part or re-target this IP using the upgrade flow to the current project part or board. See Upgrading IP for more information.
IP part change	Current project part <CURRENT_PART> and the part <ORIGINAL_PART> used to customize the IP <ip_name> do not match.		Change the project part or re-target this IP using the upgrade flow to the current project part or board. See Upgrading IP for more information.
IP contains locked subcore	IP <ip_name> contains one or more locked subcores.	Upgrade parent IP	Upgrade the parent IP <PARENTNAME>. See Editing IP Sources for more information.

Table A-1: **IP Locked Reasons and Recommendations *(Cont'd)***

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
Other	IPDef not found	The IP definition was not found in the IP Catalog	Add the IP definition to the catalog or consult the IP Catalog for the replacement IP. See Using the IP Catalog for more information.
	Read-only XCI/BOM/Project	The XCI, XML, or XPR file is read-only, so the IP is write-protected.	Review your project and file system permissions causing the IP to be read-only. See Editing IP Sources for more information.
	User-managed IP	The IP is configured as a user-managed IP. In this mode it is the users responsibility to manage all IP files.	Reconfigure to be a system managed IP (see `is_managed` property described in Editing IP Sources) if this is unexpected.
	Disabled component	Unsupported Upgrade IP.	Target IP definition `<TARGET_IPDEF>` does not support the current project part `<CURRENT_PART>`. Select a supported project part before upgrading the IP. See Appendix C, Using the Platform Board Flow for IP for more information.
	Incompatible license	The IP instance requires one or more mandatory licenses but no valid licenses were found.	Obtain a valid license before upgrading the IP. See Using Fee-Based Licensed IP.

Table A-1: **IP Locked Reasons and Recommendations *(Cont'd)***

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
Other	Incompatible XCI/BOM	Upgrade the IP	The IP Data in the catalog is incompatible with the current IP instance (despite having identical Version and Revision). You need to update the IP before viewing the customization and generating outputs. See Upgrading IP for more information.
	Deprecated flow	Upgrade the IP	The IP instance supports Vivado generation but is currently generated using the CORE™ Generator tool. See Upgrading IP for more information.
	Locked due to child IP being locked	The IP instance contains one or more locked subcores.	Either run upgrade on the IP or repackage the component using a newer version of the child IP that is currently locked. See Editing or Overriding IP Sources for more information.

1. When an IP core is locked due to a part or board change and is upgraded, you need to review the ports. Some IP have port differences based upon the part selected. For example, debug ports names and functions change when you update from 7 series FPGAs to an UltraScale® platform for the QSGMII IP. You must make RTL changes to avoid encountering errors during synthesis and or implementation. See the appropriate IP product guide for more details.

Appendix B

IP Files and Directory Structure

Introduction

When customizing an IP using the IP Catalog, either directly in a project or using the Managed IP Flow, the Vivado® Integrated Development Environment (IDE) creates a unique directory for each IP.

After creating an IP customization, a unique directory is made which contains the Xilinx® Core Instance (XCI) file, instantiation template, BOM file, and any generated output products. In this IP directory, there are several additional directories. There is no common structure for the organization of the files that each IP delivers, but there are some common files that are created for each IP.

IP-Generated Directories and Files

The following table lists the IP-generated target directories and files, which are also known as output products.

RECOMMENDED: *Xilinx recommends that you use Tcl commands to access the list of related files rather than view the file and directory structure. For example, you can use the* `get_files` *Tcl commands, which are shown in Querying IP Customization Files in Chapter 6. For more information, see the Vivado Design Suite Tcl Command Reference Guide (UG835)* [Ref 7].

Table B-1: **IP Output Products**

Directory Name, File Name, or File Type	Description
/doc	Contains the `<Core_Name>_changelog.txt` file that provides information about changes to the IP for each release.
/sim	Contains the simulation sources files for IP. This directory is not present for all IP.
/synth	Contains synthesizeable source files for IP. This directory is not present for IP that does not support synthesis, such as simulation-only bus functional model (BFM) IP.

Table B-1: **IP Output Products *(Cont'd)***

Directory Name, File Name, or File Type	Description
<ip_name>.xci	Contains the IP customization information. You can generate the output products from this file. If an upgrade path exists for the IP in the Catalog, you can upgrade from this file to the latest version.
<ip_name>_.xcix	Core Container file, which lists all the common elements between IP in a design.
<ip_name>.xml	IP Bill of Material (BOM) file that keeps track of the current state of the IP, including generated files, computed parameters, and interface information.
<ip_name>.veo|vho	Verilog (VEO) or VHDL (VHO) instantiation template. You would use one of these files to instantiate the IP inside your design.
<ip_name>.dcp*	Synthesized Design Checkpoint file contains a post-synthesis netlist and processed XDC constraints. Xilinx recommends that you do not directly reference the IP DCP file; instead use the XCI file, which brings in the DCP when needed.
<ip_name>_stub.[v|vhdl]*	Module (Verilog) and component (VHDL) for use with third-party synthesis tools to infer a black box for the IP.
<ip_name>_funcsim.[v|vhdl]*	Post-synthesis structural simulation netlist files prior to Vivado release 2015.3.
<ip_name>_sim_netlist	Post-synthesis structural simulation netlist files in Vivado release 2015.3.
<ip_name>.xdc	Timing and/or physical constraints. These files are not present for all IP, and their location varies by IP.
`<ip_name>_in_context.xdc`	See Determining Clocking Constraints and Interpreting Clocking Messages for more information.
`dont_buffer.xdc`	Deprecated file. Functionality is included in `<ip_name>_in_context.xdc`.
<ip_name>_clocks.xdc	Constraints with a clock dependency. These files are not present for all IP, and their location varies by IP.
<ip_name>_board.xdc	Constraints used in a platform board flow. These files are not present for all IP, and their location varies by IP.
<ip_name>_ooc.xdc	Default clock definitions used when synthesizing the IP out-of-context.
Encrypted HDL for the IP	Files used for synthesizing and simulating the IP. These files are not present for all IP, and their location varies by IP.

* The DCP, `_stub`, and `*_funcsim` or `*_sim_netlist` files are created only when using the Out-of-Context flow for synthesis (default). See Synthesis Options for IP in Chapter 2 for more details.

Note: Although example design are not output products, they are commonly generated for IP. The example design files are only available when the example design is opened with one of the following:

- In the Tcl Console, using the `open_example_project` command.
- Vivado IDE with the **Open IP Example Design** menu command.

For more information, see Chapter 4, Using IP Example Designs.

Files Associated with IP

The following table lists other types of files that can be associated with IP.

Table B-2: **Files Associated with IP**

File Type	Description
`Name.coe`	Coefficient file (COE) file. An ASCII text file with a single radix header followed by several vectors. The radix can be 2, 10, or 16. Each vector must be terminated by a semi-colon.
`Name.mif`	Memory information file (MIF). An ASCII text file into which the Vivado IDE translates a COE file.
`Name.bmm`	Block memory manager file.
Name.csv	Comma-separated version - a spreadsheet file.
Name.elf	Executable and linkable format file. Used by the MicroBlaze™ processor.

Note: Only some IP use these files. If using a project, the user should add them as a source. The files are typically set using a configuration property that is available in the customization GUI of the IP. For details see the product guides for the respective IP.

Using a COE File

In certain cases, some parameter values are passed to the Vivado IP Catalog using a COE (COEfficient) file; an ASCII text file with a single radix header followed by several vectors. The radix can be 2, 10, or 16. Each vector must be terminated by a semi-colon.

The Vivado tool reads the COE file and writes out one or more MIF files when the core is generated. The VHDL and Verilog behavioral simulation models for the core rely on these MIF files.

IMPORTANT: *You must upgrade all IP prior to adding a COE file. Additionally, locate the COE file in the same directory as the XCI file.*

Note: If a COE file is no longer used by an IP, remove the file. Failure to remove an old COE file can result in both the newly associated COE and the old COE being passed to synthesis. Additionally, if the old COE is removed from disk, but not from the project, an error occurs during synthesis.

COE File Syntax

The following syntax displays the general form for a COE file:

```
Keyword =Value ; Optional Comment
Keyword =Value ; Optional Comment
<Radix_Keyword> =Value ; Optional Comment
<Data_Keyword> =Data_Value1, Data_Value2, Data_Value3;
```

The following table describes COE file keywords for specifying radix values for data. Keywords are not case-sensitive. For information on the specific keywords required for a IP, see the Product Guide for that IP.

Table B-3: **COE File Keywords for Radix Values**

Keyword	Description
RADIX	Used for non-memory cores to indicate the radix being used to specify the coefficients of the filter.
MEMORY_INITIALIZATION_RADIX	Used for memory initialization values to specify the radix used.

The following table describes COE file keywords for data values. Keywords are not case sensitive.

Table B-4: **COE File Keywords for Data Values**

Keyword	Description
COEFDATA	Used for filters to indicate that the data that follows comprises the coefficients of the filter.
MEMORY_INITIALIZATION_VECTOR	Used for block and distributed memories.
PATTERN	Used for Bit Correlator COE files.
BRANCH_LENGTH_VECTOR	Used in Interleaver COE files.

Note: Any text after a semicolon is treated as a comment and ignored.

One of the following keywords must be the last keyword specified in the COE file:

- COEFDATA
- MEMORY_INITIALIZATION_VECTOR

Any other keywords that follow are ignored.

COE File Examples

The following are examples of COE files for various Xilinx IP.

Virtex Bit Correlator COE File Example

```
************************************************************************
*************  Example of Virtex Bit Correlator.COE  *************
************************************************************************
; Sample .COE coefficient file for v2.0 and later
; versions of the Bit Correlator core.
;
; In this core, a COE file is used to specify the value
; of the bit mask when the Pattern Mask option is selected.
;
; Specifications:
;
; - 19 taps, hexadecimal coefficients
; - Serial input data
;
; Please refer to the datasheet for this core for more
; details on using the Mask option.
radix = 16;
pattern = 3 0 3 1 0 1 1 3 0 2 2 2 3 0 1 1 3 0 3;
```

Dual Port Block Memory COE File Example

```
************************************************************************
*********  Example of Dual Port Block Memory .COE file  **********
************************************************************************
; Sample memory initialization file for Dual Port Block Memory,
; v3.0 or later.
;
; This .COE file specifies the contents for a block memory
; of depth=16, and width=4. In this case, values are specified
; in hexadecimal format.
memory_initialization_radix=2;
memory_initialization_vector=
1111,
1111,
1111,
1111,
1111,
0000,
0101,
0011,
0000,
1111,
1111,
1111,
1111,
1111,
1111,
1111;
```

Single Port Block Memory .COE file Example

```
*****************************************************************
********  Example of Single Port Block Memory .COE file  *********
*****************************************************************
; Sample memory initialization file for Single Port Block Memory,
; v3.0 or later.
;
; This .COE file specifies initialization values for a block
; memory of depth=16, and width=8. In this case, values are
; specified in hexadecimal format.
memory_initialization_radix=16;
memory_initialization_vector=
ff,
ab,
f0,
11,
11,
00,
01,
aa,
bb,
cc,
dd,
ef,
ee,
ff,
00,
ff;
```

Distributed Memory .COE File Example

```
*****************************************************************
************  Example of Distributed Memory .COE file  ***********
*****************************************************************
; Sample memory initialization file for Distributed Memory v2.0 and
; later.
;
; This .COE file is NOT compatible with v1.0 of Distributed Memory Core.
;
; The example specifies initialization values for a memory of depth= 32,
; and width=16. In this case, values are specified in hexadecimal
; format.
memory_initialization_radix = 16;
memory_initialization_vector = 23f4 0721 11ff ABe1 0001 1 0A 0
 23f4 0721 11ff ABe1 0001 1 0A 0
 23f4 721 11ff ABe1 0001 1 A 0
 23f4 721 11ff ABe1 0001 1 A 0;
*****************************************************************
******  Example of Distributed Arithmetic FIR Filter .COE file ***
*****************************************************************
; Example of a Distributed Arithmetic (DA) FIR Filter .COE file
; with hex coefficients, 8 symmetrical taps, and 12-bit
; coefficients.
;
```

```
; Compatible with all versions of the Distributed Arithmetic
; FIR Filter which supports Virtex and Spartan
Radix = 16;
CoefData= 346, EDA, 0D6, F91, F91, 0D6, EDA, 346;
```

MIF File Description

The COE file provides a high-level method for specifying initial memory contents. When the core is generated the Vivado tools convert the COE file into a MIF file, which holds the actual binary data used to initialize the memory in the core and simulation models.

The MIF file consists of one line of text per memory location. The first line in the file corresponds to address 0, and the second line corresponds to address 1, and so forth. The text on each line must be the initialization value (MSB first) for the corresponding memory address in binary format, with exactly one binary digit per bit of memory width.

Note: For HDL simulations, the MIF file must reside in the simulation directory.

Using the Platform Board Flow for IP

Introduction

The Vivado® Design Suite Platform Board Flow feature is supported by some IP and gives you the ability to select board interfaces while customizing an IP. When you use this feature, the creation of physical constraints for the IP is automated by delivering additional XDC constraints in to define pin assignments and `IOSTANDARDS` for interface signals implemented on the target board.

As shown in the following figure, when creating a new project, you can select a board as the default part.

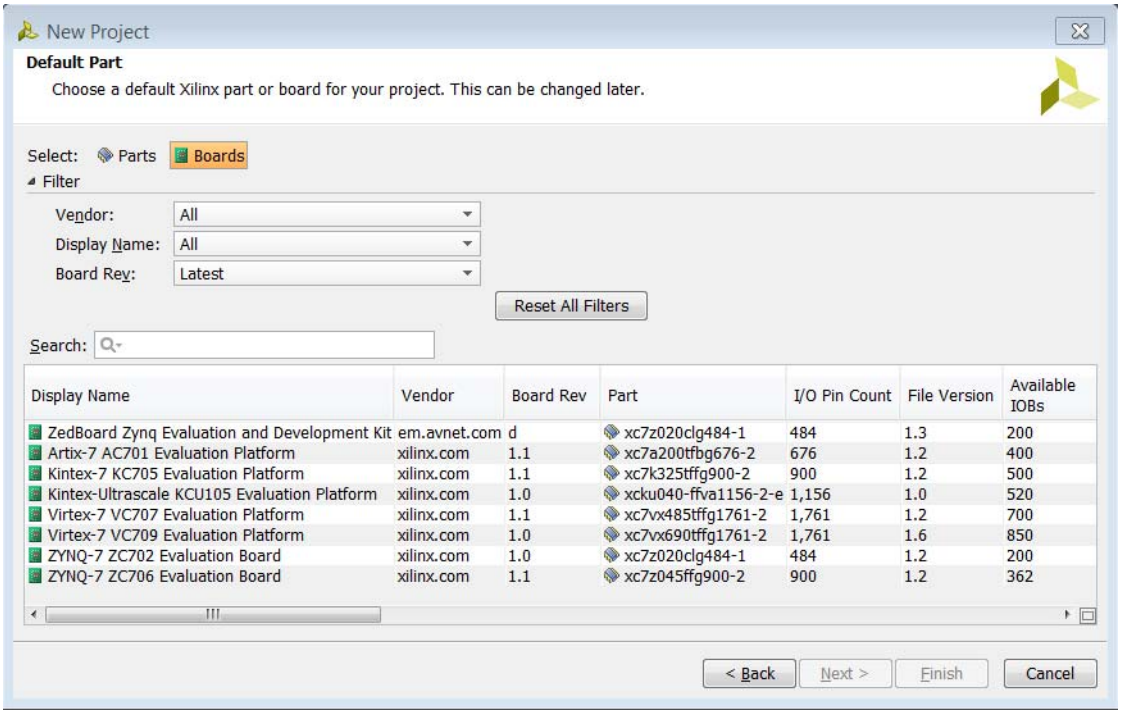


Figure C-1: **Selecting a Board as the Default Part**

Selecting one of the listed boards enables a Board tab within the IP customization dialog box for IP cores that support the platform board flow.

Figure C-2: **Board Summary Information**

Selecting one of the listed boards results in IP that supports the platform board flow by providing a new tab visible during customization, as shown in the following figure.

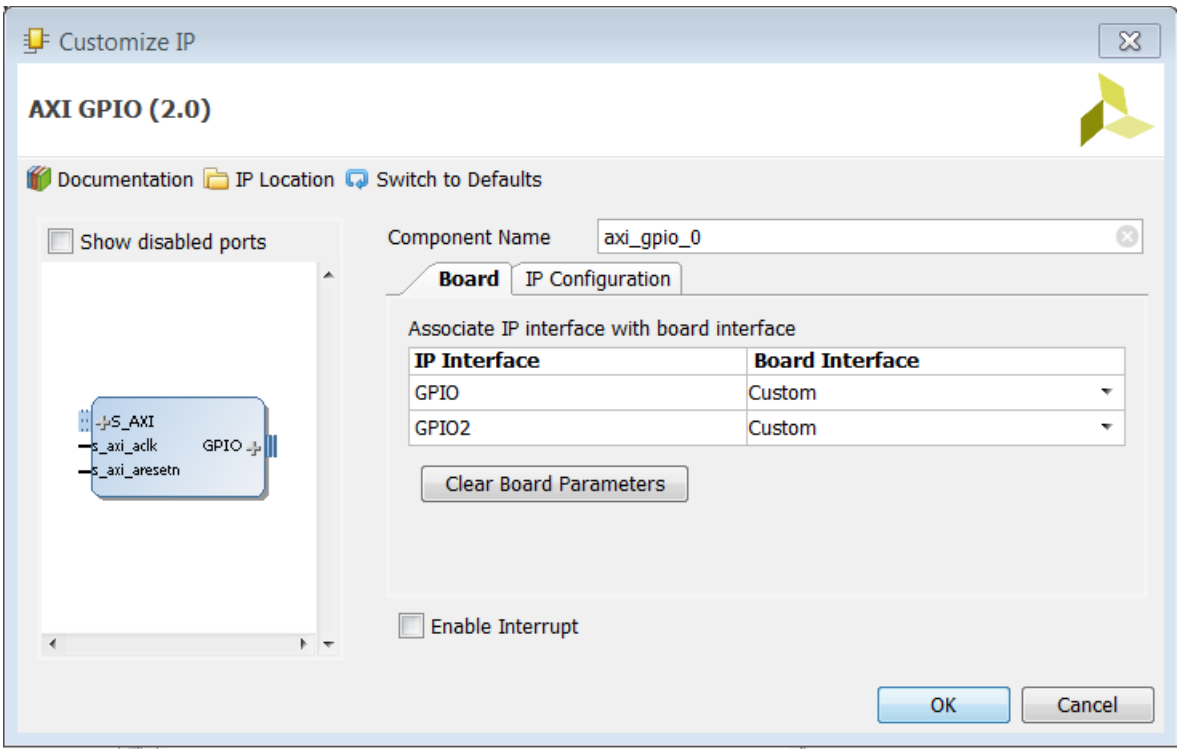


Figure C-4: **Board Type Visible in Supported IP Customization**

The Board tab lets you associate the interfaces defined on the IP core with interfaces implemented on the target board.

The following figure shows that you can associate the IP interface to the one of the associated board interfaces.

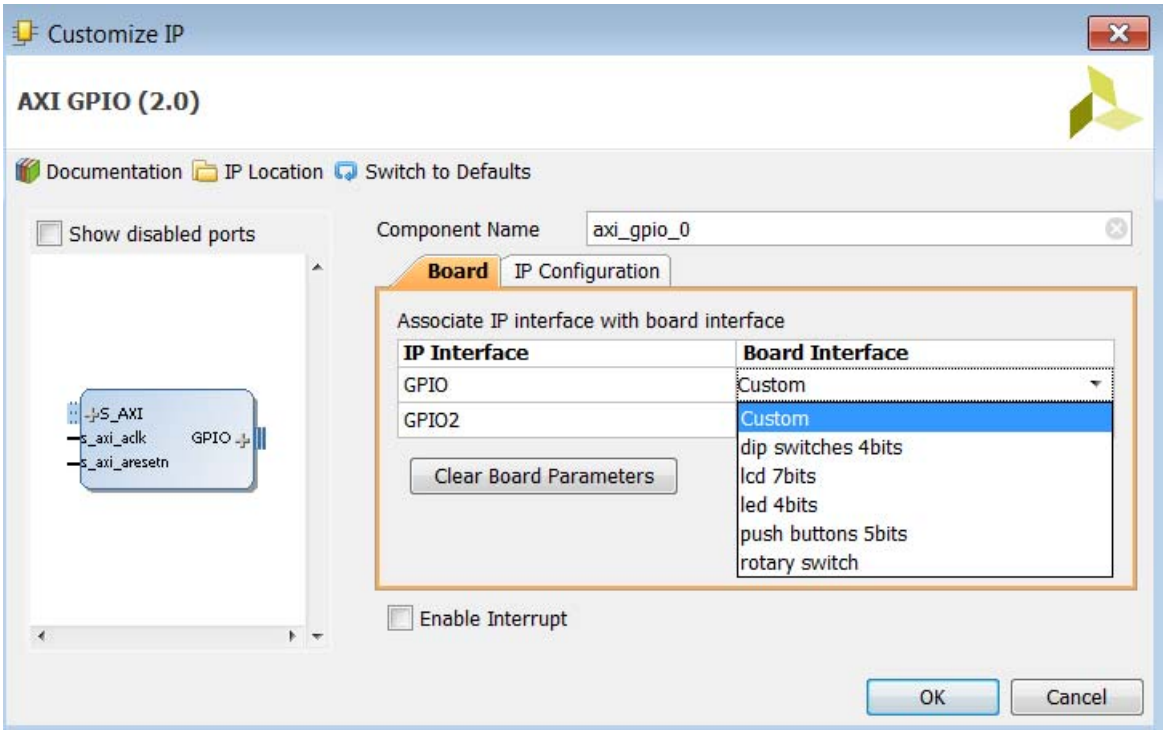


Figure C-4: **Associating the IP Interface with the Board Interface**

When the Vivado IDE generates the IP output products in the IP Sources view, you can see the `<IP_Name>_board.xdc` file listed.

This file contains physical constraints assigning ports of the IP to the package pins that connect to the related board connector or device such as a USB port, LED, button, or switch.

The following figure shows the XDC constraints created for the GPIO IP when you connect the **GPI0** interface to the board **lcd7bits** interface and the connect the **GPIO2** interface to the board **push button 5bits** interface.

The use of the Vivado Design Suite platform board flow can let you quickly connect IP interface signals onto the target board to speed implementation of the design onto the board.

If you have selected a target board for your project, any IP that supports the Vivado platform board flow has a Board tab in the IP customization dialog box, as shown in Figure C-4.

- See this link to the *Vivado Design Suite User Guide: Designing IP Subsystems with IP Integrator* (UG994) [Ref 6] for more information on using the platform board flow.
- See this link to the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 18] for information on the Board Interface file and creating your own board files.

Editing or Overriding IP Sources

Introduction

At times, you might need to modify or override unencrypted source files that an IP delivers, including XDC files and HDL files. This should only be done if absolutely necessary. Modifying IP sources could result in the IP not functioning correctly.

IMPORTANT: *If you determine you must modify any of the IP sources, do not directly modify the sources on disk unless you follow the guidelines provided in this appendix. Directly making modifications can result in your changes being removed because the IP could become reset or regenerated during the flow.*

In the case where there is a need to modify an IP RTL source see Editing IP Sources.

To modify XDC commands delivered by the IP there are two options:

1. Override the XDC in a top-level XDC or Tcl file.

2. Edit the IP source.

Overriding IP Constraints

IP are validated with the constraints that are delivered with them. In some cases though an IP delivered constraint might need to be changed, such as a physical constraint like a `LOC` or `PACKAGE_PIN` property, to meet design goals.

You can edit the IP XDC using the method described in Editing IP Sources. Alternatively, you can override the IP XDC command by providing a top-level user XDC or a Tcl file with the desired commands.

VIDEO: *The Vivado Design Suite QuickTake Video: Working with Constraint Sets can provide more information also.*

Depending on what kind of constraint you want override, you can use either a XDC file or a Tcl file (see this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 8]).

RECOMMENDED: *You are strongly recommended to not modify any IP timing constraints with the possible exception of the* `_ooc.xdc` *to set a target frequency for synthesizing the IP out-of-context.*

Because the IP is synthesized out-of-context by default, overriding a physical constraint should be done during the implementation stage only. Physical constraints are ignored during synthesis of the IP standalone; consequently make the Tcl or XDC file be for implementation use only.

Follow the procedure outlined in the Editing IP Sources when it is required to override an IP timing constraint. This ensures that the changes are used during synthesis of the IP out-of-context as well as being used during implementation at the top-level.

XDC commands are processed in order, where the last command takes precedence. With timing constraints, this is not always successful; If an IP sets a path to have a false path exception and you later apply a `max_delay` constraint on the same path, the false path remains because it has higher precedence (see this link to the *Vivado Design Suite: Using Constraints* (UG903) [Ref 8], for more details). To make these levels of changes you must modify the XDC delivered by the IP.

Some actions and commands are not allowed in an XDC, necessitating use of a Tcl file. An example of this is the changing of a `LOC` property on a `BUFG_GT` cell. The placer is not able to place an instance on a site which is already occupied. You must first clear the current setting and then set the new `LOC`. This should be done with the `reset_property` command, which is not an XDC command, and must be placed in a Tcl file. After resetting the `LOC` property you would then set the new value.

Scoping Constraints

The XDC files that an IP delivers are *scoped* to the IP instance(s) using two properties on the XDC file(s):

- `SCOPED_TO_REF`: Specifies the module to which to apply the XDC file.
- `SCOPED_TO_CELLS`: Specifies the cell within the module to which to apply the XDC file.

For more information on these properties see *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 8].

When overriding IP constraints at the top-level you have two choices:

1. Specify the hierarchy to specific cell of the IP. If there are multiple instances of the IP, do either of the following:
   - Use wild cards
   - Duplicate the constraint for each IP
2. Use the SCOPED_TO_REF and SCOPED_TO_CELLS properties that the IP uses and write your constraints as if the IP cell were the top-level of the hierarchy (*recommended*).

   To find the SCOPED_TO_REF and SCOPED_TO_CELLS values you can use the report_compile_order -constraints command. Look at the synthesis or implementation section for the IP fileset.

```
Constraint evaluation order for 'synthesis' with fileset 'pcie3_X8_X0Y1':
Index  File Name                    Used_In       Scoped_To_Ref     Scoped_To_Cells
-----  ---------------------------  ------------  ----------------  ---------------
1      pcie3_X8_X0Y1_ooc.xdc        Synth & Impl  pcie3_X8_X0Y1     inst
2      pcie3_X8_X0Y1_gt.xdc         Synth & Impl  pcie3_X8_X0Y1_gt  inst
3      pcie3_X8_X0Y1-PCIE_X0Y1.xdc  Synth & Impl  pcie3_X8_X0Y1     inst
```

Figure D-1: **Synthesis Fileset**

The SCOPED_TO_REF is typically the IP customization name. The SCOPED_TO_CELLS is typically either inst in Verilog or U0 in VHDL.

RECOMMENDED: *Xilinx recommends you create a new XDC or Tcl file and place all the XDC/Tcl commands to override the IP XDC and set the* SCOPED_TO_REF *and* SCOPED_TO_CELLS *properties to match what* 1 *lists.*

The complete procedure is:

1. Create a new XDC or Tcl file and add it to your active constraint set.
2. Place any XDC or Tcl commands required to override the IP XDC in the new file.
3. Use the set_property command to set the SCOPED_TO_REF and SCOPED_TO_CELLS properties:

   ```
   set_property SCOPED_TO_REF <REF> [get_files <new XDC/Tcl file>]
   set_property SCOPED_TO_CELLS <CELL> [get_files <new XDC/Tcl file>]
   ```

4. Mark the XDC/Tcl file to be used in implementation only:

   ```
   set_property USED_IN IMPLEMENTATION [get_files <net XDC/Tcl file]
   ```

Editing IP Sources

To prepare an IP for editing:

1. If you have not customized the IP, do so, and generate all output products, including the DCP.

   If you do not want to use the default OOC flow for the IP, disable the DCP creation by going to the out-of-context settings.

RECOMMENDED: *Xilinx highly recommends that you use the default flow.*

2. After you generate the output products (including the DCP, if applicable) are generated, set the `IS_MANAGED` property to `false` on the XCI file for the IP using the following Tcl command:

   ```
   set_property IS_MANAGED false [get_files <IP_NAME>.xci]
   ```

   If it is a complex subsystem IP, the following error message displays:

   ```
   ERROR: [IP_Flow 19-3666] The is_managed property cannot be directly modified for
   hierarchical IP.
   ```

3. Upon receipt of this error, read the Editing Subsystem IP, and follow those steps.

   Setting the `IS_MANAGED` property to `false` causes the property `IS_LOCKED` to become `TRUE`. The IP icon in the IP Sources window changes to , showing the IP is not managed by Vivado and is instead user-managed.

   In the output window of the **Report IP Status** command you see that the IP is under user management, and you can modify non-encrypted HDL files and XDC files.

4. Complete the required edits.
5. Recreate the DCP using the modified files as follows:
   a. In the Design Runs tab, right-click the IP and select **Launch Runs**.

      A dialog box opens and asks you if you want to launch the selected run.

   b. Click **OK**.

      Another dialog box opens and informs you that the run must be reset and all the files will be deleted for the run.

      Note: This is not referring to the IP source files, only the run-related files and output.

After the run is complete, you can use the IP as before.

RECOMMENDED: *By referencing the XCI file (which is recommended) you have access to the IP source files for simulation, the DCP for synthesis of the top-level file, as well as for implementation.*

Editing Subsystem IP

Some complex subsystem IP do not allow changes to the `IS_MANAGED` property. This condition is applicable for IP supporting 7 series® and UltraScale™ device families.

Whether or not a subsystem IP allows the `IS_MANAGED` property to be changed depends on particular customization options of the specific IP.

CAUTION! *Editing the RTL files of such IP has risks. It is possible to make a change that invalidates the connectivity to the sub-cores. Making changes to these IP HDL sources should be carefully considered.*

1. Make sure the IP has been fully generated using Out-of-Context (OOC) per IP for the synthesis option. You will need to have an existing design run for the IP present.
2. Find the IP RTL file that requires the edit and make changes as needed.

   You must either:

   - Change to another editor using **Tools > Options > General** in the text editor section.
   - Edit the files directly on disk using your text editor of choice.
3. Reset and re-launch the IP run following these steps:
   a. In the Design Runs tab, right-click the IP and select **Launch Runs**.

      A dialog box opens and asks you if you want to launch the selected run.

   b. Check the **Delete the generated files in the working directory box**. This will remove only the run-related files and output, not the file(s) you edited.

   c. In the Design Runs tab, right-click the IP run and select Launch Runs. A dialog box opens and asks you if you want to launch the selected run.

After the run is complete, you can use the IP as before.

Appendix E

Additional Resources and Legal Notices

Xilinx Resources

For 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.

See the Xilinx Memory Interface Solution Center for information regarding the Memory IP.

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. End User License Agreement
2. Core License Agreement
3. Core Evaluation License Agreement

Vivado Design Suite Documentation

The following documents are cited within this guide:

4. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
5. *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* (UG1119)
6. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
7. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
8. *Vivado Design Suite User Guide: Using Constraints* (UG903)
9. *Vivado Design Suite Properties Reference Guide* (UG912)
10. *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
11. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
12. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
13. *Vivado Design Suite User Guide: Getting Started* (UG910)
14. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
15. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
16. *Vivado Design Suite User Guide: Hierarchical Design* (UG905)
17. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
18. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)
19. *Vivado Design Suite Tutorial: Designing with IP Tutorial* (UG939)
20. *Vivado Design Suite Tutorial: Logic Simulation* (UG937)
21. *Vivado Design Suite Tutorial: Design Flows Overview* (UG888)
22. *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894)
23. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
24. *Vivado AXI Reference Guide* (UG1037)

25. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)
26. *ISE to Vivado Design Suite Migration Guide* (UG911)
27. *Vivado Design Suite Tutorial: Revision Control* (UG1198)
28. Vivado Design Suite Documentation

Xilinx IP Documentation

29. Vivado IP Versioning
30. IP Documentation
31. IP Center
32. *Integrated Logic Analyzer LogiCORE IP Product Guide* (PG172)
33. *IBERT 7 Series GTX Transceivers LogiCORE IP Product Guide* (PG132)
34. *IBERT 7 Series GTP Transceivers LogiCORE IP Product Guide* (PG133)
35. *IBERT 7 Series GTH Transceivers LogiCORE IP Product Guide* (PG152)
36. *Virtual Input/Output* LogiCORE IP Product Guide (PG159)
37. *Zynq-7000 SoC and 7 Series FPGAs Memory Interface Solutions* (UG586)
38. *JTag to AXI LogicCORE IP Product Guide* (PG174)
39. *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)

Standards and Third-Party Documentation

40. IP-XACT (IEEE Std 1685), *Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows*

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Essentials of FPGA Design
2. Embedded Systems Software Design

Vivado QuickTake Videos

1. Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades
2. Vivado Design Suite QuickTake Video: Vivado Licensing and Activation Overview
3. Vivado Design Suite QuickTake Video: Customizing and Instantiating IP
4. Vivado Design Suite QuickTake Video: Creating an AXI peripheral in Vivado
5. Vivado Design Suite QuickTake Video: Configuring and Managing Reusable IP in Vivado
6. Vivado Design Suite QuickTake Video: Design Constraints Overview
7. Vivado Design Suite QuickTake Video: Global Timing Constraints
8. Vivado Design Suite QuickTake Video: Managing Sources with Projects
9. Vivado Design Suite QuickTake Video: Migrating UCF Constraints to XDC
10. Vivado Design Suite QuickTake Video: Tcl Scripts and Constraint Files in Vivado
11. Vivado Design Suite QuickTake Video: Using Core Containers with IP
12. Vivado Design Suite QuickTake Video: Using IP with 3rd-Party Synthesis Tools
13. Vivado Design Suite QuickTake Video: Vivado Activation and Floating License Generation
14. Vivado Design Suite QuickTake Video: Working with Constraint Sets
15. Vivado Design Suite QuickTake Video: Working with Design Checkpoints
16. Vivado Design Suite QuickTake Video: Designing with UltraScale Memory IP
17. Vivado Design Suite QuickTake Video: Getting Started with the Vivado IDE
18. Vivado Design Suite QuickTake Video Tutorials

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, and PrimeCell are trademarks of ARM in the EU and other countries. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. PCI, PCI Express, PCIe, and PCI-X are trademarks of PCI-SIG.

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.