See all versions
of this document

Vivado Design Suite User Guide

Designing with IP

UG896 (v2025.2) December 17, 2025

Table of Contents

Chapter 1

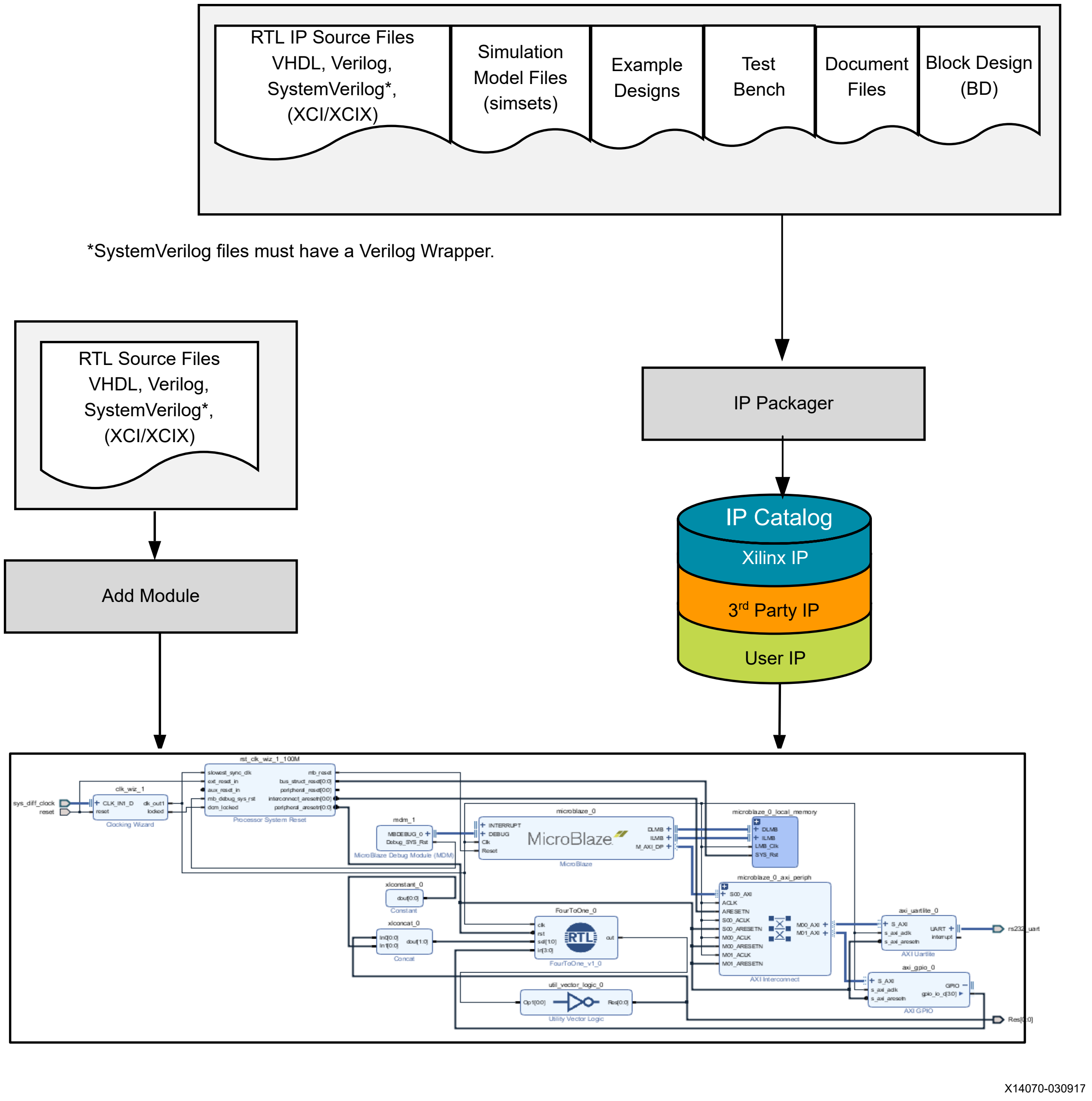
IP-Centric Design Flow

The AMD Vivado™ Design Suite provides an intellectual property (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 AMD-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: **IP-Centric Design Flow**


RTL IP Source Files VHDL, Verilog, SystemVerilog*, (XCI/XCIX)
Simulation Model Files (simsets)
Example Designs
Test Bench
Document Files
Block Design (BD)
*SystemVerilog files must have a Verilog Wrapper.
RTL Source Files VHDL, Verilog, SystemVerilog*, (XCI/XCIX)
Add Module
IP Packager
IP Catalog
Xilinx IP
3rd Party IP
User IP
MicroBlaze
X14070-030917

***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:

- 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 AMD core instance (XCI) file, which is a recommended method for working with large projects with contributing 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 for projects with small team sizes, it is recommended that you save it externally from the project.
- Add sources by right-clicking in IP integrator canvas and add an RTL module to a design diagram, which provides an *RTL on Canvas*. See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) for more information on module references.
- Create and customize IP and generate output products in a Non-Project script flow, including generation of a DCP. See the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information about Non-Project mode.

  Always reference the IP using the XCI file. It is not recommended to read only the IP DCP file, either in a Project Mode or Non-Project Mode flow. While the DCP did contain constraints prior to 2017.1, in Vivado releases going forward, it does not contain constraints or provide other output products that an IP could deliver and that could be needed, such as ELF or COE files, and Tcl scripts.

The *Vivado Design Suite Tutorial: Designing with IP* (UG939) provides instruction on how to use AMD IP in Vivado.

Navigating Content by Design Process

AMD documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

Hardware, IP, and Platform Development

Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:

- Chapter 2: IP Basics
- Chapter 3: Using Manage IP Projects

System Integration and Validation

Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:

- Working with Debug IP

IP Terminology

The AMD 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 user-specified configuration.
- IP Location: A directory that contains one or more customized IP in the current project.
- IP Repository: A unified view of a collection of IP definitions added to the AMD IP catalog.
- IP Catalog: The IP catalog allows for the exploration of AMD plug-and-play intellectual property (IP), and other IP-XACT-compliant IP provided by third-party vendors. This can include designs that you package as IP. See Chapter 2: IP Basics, for more information.
- Output Products: Generated files produced for an IP customization. They can include HDL, constraints, and simulation targets. During output product generation, the Vivado tools store IP customizations in the XCI file and uses the XCI file to produce the files used during synthesis and simulation.
- Global Synthesis: To synthesize the IP along with the top-level user logic.
- Out-Of-Context (OOC) Design Flow: The OOC design flow creates a standalone synthesis design run for generated output products. This default flow creates a design checkpoint file (DCP) and an AMD design constraints file (`_ooc.xdc`). See Out-of-Context Flow for more information.
- Hierarchical IP and Subsystem IP: These terms are used interchangeably to describe an IP which is a sub-system built with multiple IP in a hierarchical topology as a part of a block design or RTL flow.
- Sub-core IP: The term *sub-core* IP refers to an IP used within another IP that is not Hierarchical (Subsystem) IP. This could be IP from the Vivado IP catalog, user-defined IP, third-party IP, or IP core libraries.

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.

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

IP Integrator

The AMD Vivado™ Design Suite IP integrator tool lets you create complex subsystem designs by instantiating and interconnecting IP cores and module references 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).

Using Revision and Source Control

The AMD Vivado™ Design Suite is designed to work with any revision control system. For IP designs there are trade-offs 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 the *Vivado Design Suite User Guide: Design Flows Overview* (UG892).

Using Encryption

AMD encrypts IP HDL files with the IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP) (IEEE Std P1735).

VIDEO: *See the Vivado Design Suite QuickTake Video: Using IP Encryption Tool in Vivado Design Suite for more information about encryption.*

Also, see the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118).

IP Basics

This chapter describes the features of Designing with IP.

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

Working with AMD IP consists of first customizing an IP for use in an RTL design. You can create an IP customization in various ways using the AMD 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 and 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.
- Adding or creating block designs (BDs).

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

- In Project mode, if output products are not present, the Vivado tools generate the required output products automatically prior to synthesis or simulation. By default, output products are generated out-of-context (OOC) for synthesis. See Out-of-Context Flow for more information.
- 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 of your top-level design. The IP output products have instantiation templates in both VHDL and Verilog that are generated automatically. See Using IP Project Settings for more information.

VIDEO: *See the Vivado Design Suite QuickTake Video: Getting Started with Vivado IDE for more 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 settings using the IP category in the Settings dialog box. The following options are available:

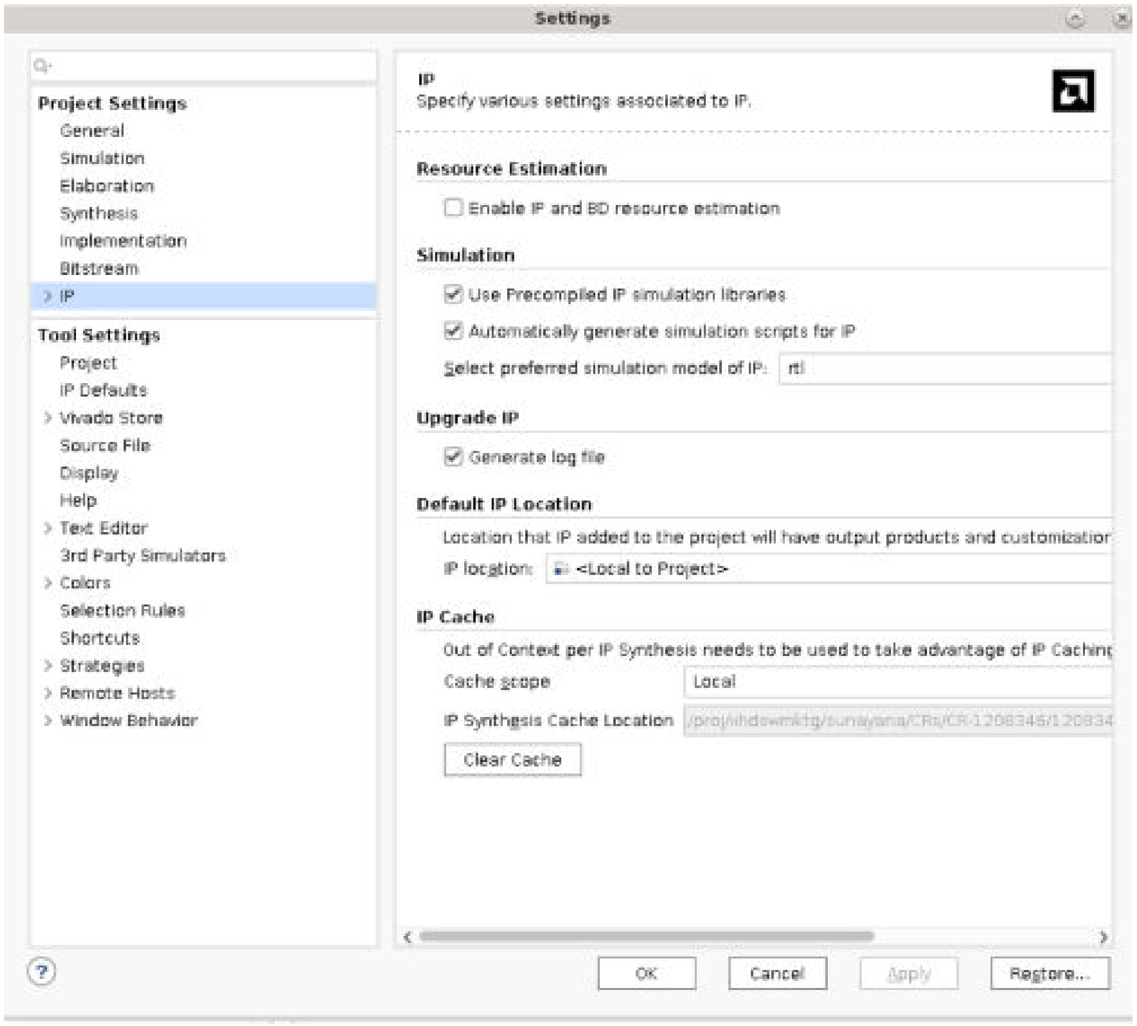
- IP: Lets you specify the use of Core Containers, automatic generation of simulation scripts, upgrade log creation, setting the default location for IP output products, and turning on IP caching.
    - **Repository**: Adds IP repositories and specifies the IP to be included in the IP catalog.
    -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

    The IP Settings for a project, and the Vivado IP catalog are available when working with an RTL project or when using **Manage IP** option from the Getting Started page. When using a Manage IP project, you see a subset of the IP settings that are available.

IP Settings

The IP settings let you specify various project-specific options for IP, as shown in the following figure. For each project you create, you must access the settings for a project again to change the settings.

Figure 2: **General IP Settings Dialog Box**



Settings

Project Settings
- General
- Simulation
- Elaboration
- Synthesis
- Implementation
- Bitstream
- IP

Tool Settings
- Project
- IP Defaults
- Vivado Store
- Source File
- Display
- Help
- Text Editor
- 3rd Party Simulators
- Colors
- Selection Rules
- Shortcuts
- Strategies
- Remote Hosts
- Window Behavior

IP
Specify various settings associated to IP.

Resource Estimation

☐ Enable IP and BD resource estimation

Simulation

☑ Use Precompiled IP simulation libraries

☑ Automatically generate simulation scripts for IP

Select preferred simulation model of IP: rtl

Upgrade IP

☑ Generate log file

Default IP Location

Location that IP added to the project will have output products and customization

IP location: <Local to Project>

IP Cache

Out of Context per IP Synthesis needs to be used to take advantage of IP Caching

Cache scope: Local

IP Synthesis Cache Location: /proj/xhdswmktg/sunayana/CRs/CR-1208346/120834

The IP settings contain the following options:

- 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 two check boxes, Use Precompiled IP simulation libraries and Automatically generate simulation scripts for IP options are checked.

  Vivado delivers precompiled libraries for all the AMD IP static files to use with the Vivado 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).

The Automatically Generate Simulation Scripts for IP option generates simulation scripts for each IP automatically. The Vivado tool places the scripts in the `<project name>.ip_user_files` directory. See Simulating IP for more information. To disable simulation scripts from being created, uncheck the option.

- Upgrade IP: By default, the Generate log file is checked. This creates an `ip_upgrade.log` file when you upgrade 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 in which to create and store your IP sources. By default, the Vivado tools store IP in an RTL project within the project directory structure in the `.srcs/sources_1/ip` directory.
  - When working with revision control systems, it is recommend that you store you IP outside of the project as with other source files.
  - Vivado generates all IP output products in a separate `<project_name>.gen` directory. This separates generated output products from the current sources that reside in `<project_name>.srcs` directory. This provides a cleaner directory structure to differentiate between IP sources and output products. It also minimizes the number of checked-in files required to re-create the project IPs for most revision control use cases because for many revision control scenarios it is not required to check in the output products.
  - When customizing an IP, use the IP Location to set the location where the IP and its output products are stored. Setting the default IP location persists across multiple Vivado sessions.
- IP Cache: Lets you define how Vivado uses IP caching for the project. OOC synthesis runs will check for a cache entry before launching synth_design. If IP runs are queued and later after finding cache entries, if their OOC synthesis is using cached results. This is because the synthesis run process did start (sourcing the synthesis run tcl file), and the cache check was done as part of that run handling. The cache entry was found and copied over to the IP's gen area and so the synth_design tool was never invoked, and the run process ends. For the cases that show the "Using cached results..." msg: in these cases, the cache check was done BEFORE the synthesis run was launched. After a cache entry was found and copied over, no synthesis run was needed. Caching options include:
  - Cache scope: Options are disabled, local (default setting), or remote. For local, the caching directory is local to the project and the location cannot be changed. The remote option is for a directory that you specify.
  - Cache location: Browse to, and select the location for cache. For local, the caching directory is local to the project (project_name.ip_cache) and the location cannot be changed. For remote, it is a directory you specify.
  - Clear Cache: Deletes the cache files from the disk by issuing the following command, `config_ip_cache`, on the Tcl Console:

```
config_ip_cache -clear_output_repo
```

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

IMPORTANT! *The Non-Project flow does not support IP Caching.*

Setting the IP Cache

To speed up generation of the synthesis output products for an IP using the default OOC flow in Project Mode, the IP Cache option is enabled by default. With the cache enabled, when you generate an IP using the default OOC flow for synthesis, the Vivado tool creates synthesis output products (such as DCP and stub files) and a cache entry.

Cache ID: Unique series of random character and number for reference generated.

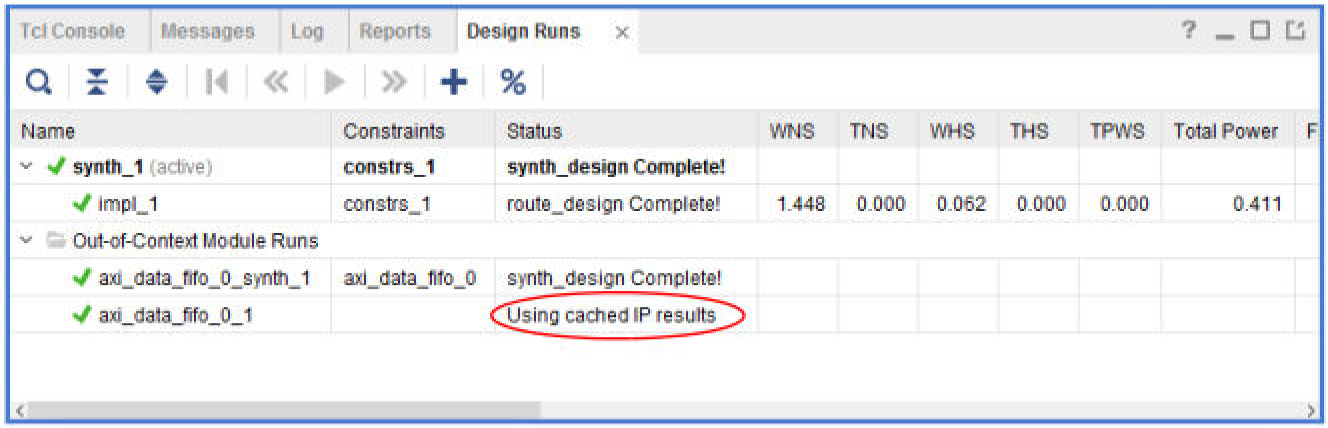
A cache entry consists of two directories on disk:

- `<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, and when you create a new customization of the IP with the same properties, the IP are not synthesized again during generation. Instead, the Vivado references the cache and copies the synthesis output. The IP refers to the already synthesized IP, eliminating the need to run OOC synthesis for other IP of the same customization.

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, 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, as shown in the following figure.

Figure 3: **Design Runs Status**



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	F
✓ synth_1 (active)	constrs_1	synth_design Complete!							
✓ impl_1	constrs_1	route_design Complete!	1.448	0.000	0.062	0.000	0.000	0.411	
Out-of-Context Module Runs									
✓ axi_data_fifo_0_synth_1	axi_data_fifo_0	synth_design Complete!							
✓ axi_data_fifo_0_1		Using cached IP results							

After generation, a dialog box informs you if a cache hit has occurred.

The following is an example of the `INFO` message that the Vivado tools produce in the Tcl Console and stores 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 and 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 project. The cache is only referenced during the generation of 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 cache contents and the size of the cache in KB. Additionally, you can run the `config_ip_cache -zip_cache` command to zip up the cache entries used in the current project. This zipped cache can be used as a read-only user repository cache without having to unzip the file. See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for more information.

Configuring IP Cache for New Projects

By default, all new projects have the IP Cache enabled and configured to be local to the project; however, 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 need to generate the same IP used by others.

To configure newly-created projects to disable the cache or specify a remote location as a default, you can use the `Vivado_init.tcl` and add a parameter to control the `project.defaultIPCacheSetting`. See the*Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) for more information about the `Vivado_init.tcl` file and initialization scripts.

When using a shared cache, create a directory specifically for the IP cache and point to that directory. IP cache entries are created under that directory, and a parent directory is not created. If you set the remote location somewhere where there are many other sub-directories, it slows the project creation because the Vivado tool scans the directories for cache entries.

Setting Cache in the Vivado_init.tcl Example

The following is an example of how to disable the IP Cache using the `Vivado_init.tcl` file:

```
set_param project.defaultIPCacheSetting none
```

The following is an example on setting the IP Cache to be in a remote location:

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

Configuring IP Cache Archive for Projects

IP cache entries created for the project can be zipped up by the tool as a single package.

To create the zipped IP cache package, user needs to run `config_ip_cache -zip_cache` command to zip up all cache entries used by the current project into a zip file. Cache entries can be found in multiple places (project or global cache, user repositories).

The zip file can be used as your repository without unzipping the file and you can add it as a repository to the IP catalog identified in `ip_repo_paths` project property.

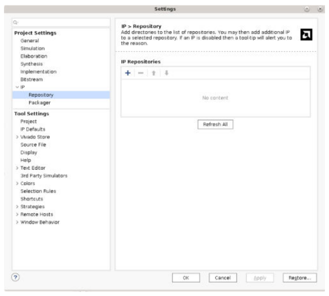
IP instances whose IP cache entries are found in the zipped repository do not need to be re-generated.

This would allow user to re-build the project with a much faster turn-around time especially when project design sources are distributed remotely or shared among team members.

Managing IP Repositories

The following figure shows the Repository option.

Figure 4: **IP Repository Option**



Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

To change settings for the Repository, from **Flow Navigator > Settings**, click **IP > Repository**. Manage locations for IP (either custom-packaged IP or third-party IP) with the following actions:

- Click the **Add** button ➕ to specify the location of the IP definitions. The Repository Manager hierarchically searches within the specified path for IP definitions. The IP is listed after the directory is specified. When you add an IP repository, a dialog box displays with a list of IP that are in the repository. The IP that contain a gray icon are disabled, and those with a yellow icon are enabled.

- Use the **Remove** button to remove a repository listing. The IP catalog shows that the repository is no longer present.
- Change the search order of the Repositories using the up and down arrows.

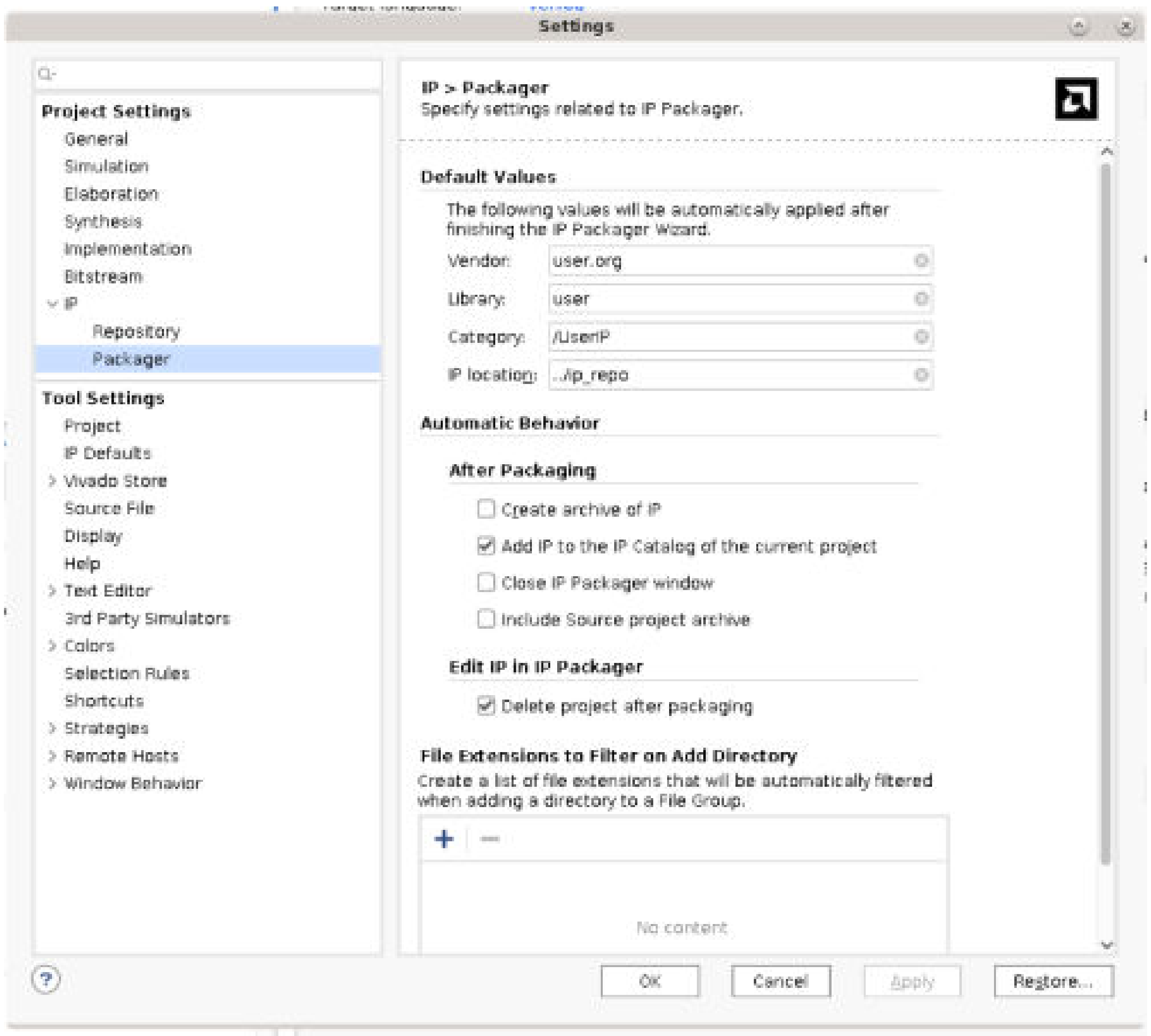
Update the contents of the IP catalog with the IP within each repository, click **Apply**, next click **OK**.

TIP: *As an alternative to using the Repository Manager of the IP 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.*

Using the Packager Settings

The following figure shows the Packager settings.

Figure 5: **IP Settings: Packager**



IP > Packager
Specify settings related to IP Packager.

Default Values

The following values will be automatically applied after finishing the IP Packager Wizard.

Vendor: user.org
Library: user
Category: /UserIP
IP location: ../ip_repo

Automatic Behavior

After Packaging

- [ ] Create archive of IP
- [x] Add IP to the IP Catalog of the current project
- [ ] Close IP Packager window
- [ ] Include Source project archive

Edit IP in IP Packager

- [x] Delete project after packaging

File Extensions to Filter on Add Directory

Create a list of file extensions that will be automatically filtered when adding a directory to a File Group.

No content

See the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) for more information about packaging IP and the Packager settings.

Using the IP Catalog

The key features of the Vivado IP catalog include:

- Consistent, easy access to AMD 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 AMD-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.
- *Table 1:* **IP Versioning**

Change Level	User Action	Examples of Changes
Revision	**No** need to react	Add new device support Cosmetic GUI changes Move device support from Pre-Production to Production Extend parameter range Bug fix for unusable configurations (no working configuration changed)
Minor	**May** need to react	Reduction in parameter range Remove an optional port Add a memory-mapped register whose use is optional Increased resource usage

Table 1: **IP Versioning** *(cont'd)*

Change Level	User Action	Examples of Changes
Major	**Will** need to react	Add a non-optional, non-static input port Rename a non-optional port (including case change if Verilog) Change a non-optional port's size Remove a non=optional port Change the interface standard Change or remove a memory-mapped register Behavior changes for all configurations

- Access to version history details as recorded in the Change Log. A `<Major#.Minor#.Rev#>` numbering scheme unifies the IP version numbers.

  For more information, see the AMD IP Versioning page available from the AMD website for Vivado IP Versioning.

  - 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.

Use the `report_property` Tcl command to list the properties available for an IP Definition, as follows:

```
report_property -all [get_ipdefs <IP VLNV>]
```

This information is also shown in the IP catalog. An example of the command is, as follows:

```
report_property -all [get_ipdefs amd.com:ip:fifo_generator:13.1]
```

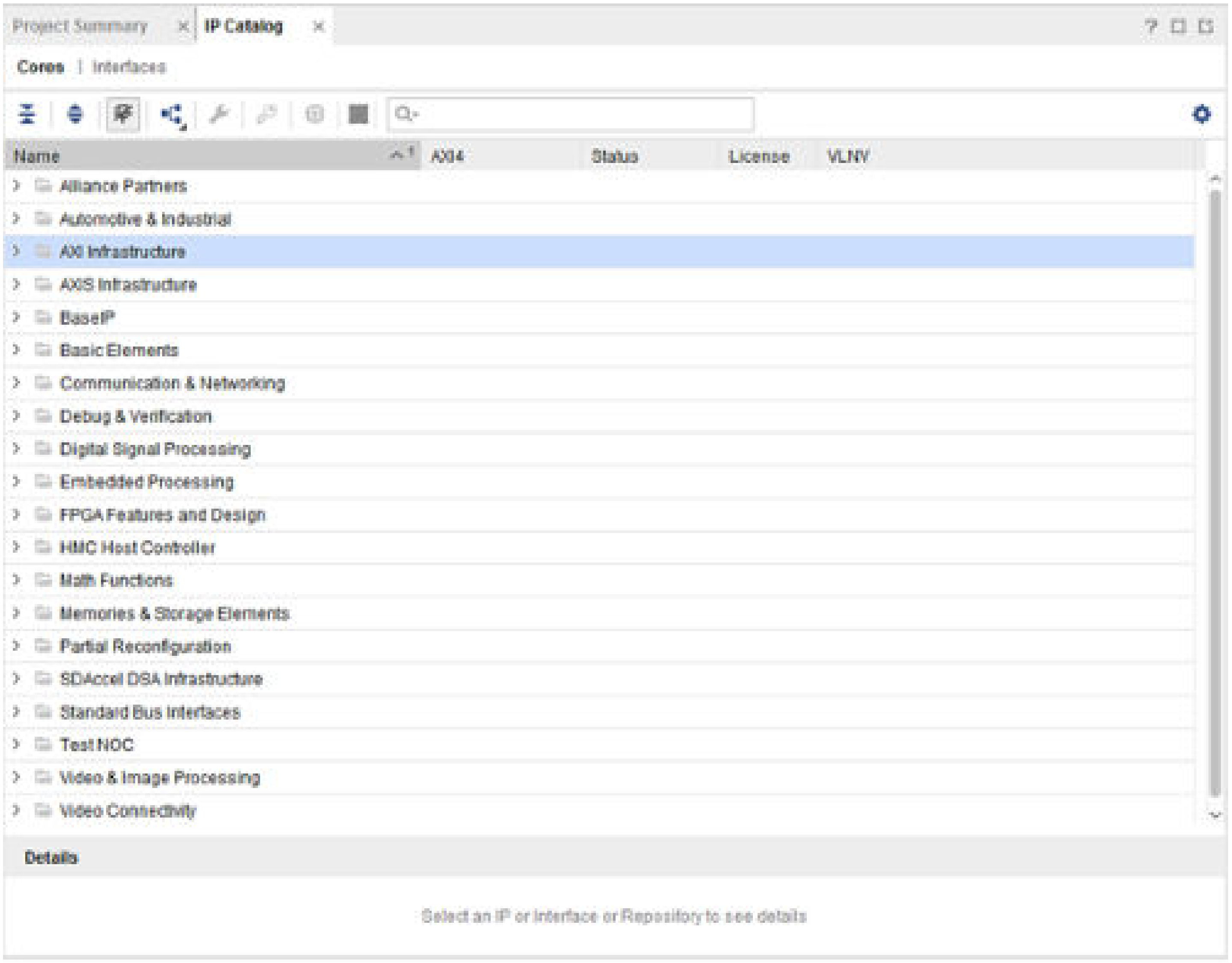
You can also query for a specific property, as follows:

```
get_property supported_families [get_ipdefs
amd.com:ip:fifo_generator:13.1]
```

For information on IP that supports the Vivado Design Suite, see the AMD website for IP Documentation. For information on specific IP, see the IP Center 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 6: **AMD IP Catalog - Cores**



Alliance Partners
Automotive & Industrial
AXI Infrastructure
AXIS Infrastructure
BaseIP
Basic Elements
Communication & Networking
Debug & Verification
Digital Signal Processing
Embedded Processing
FPGA Features and Design
HMC Host Controller
Math Functions
Memories & Storage Elements
Partial Reconfiguration
SDAccel DSA Infrastructure
Standard Bus Interfaces
Test NOC
Video & Image Processing
Video Connectivity

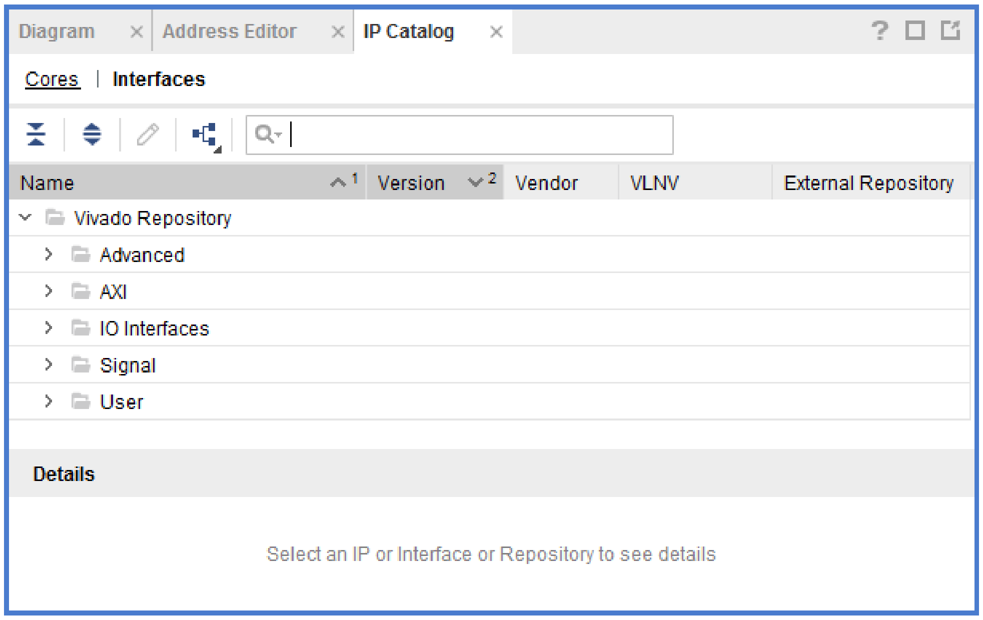
Details

Select an IP or Interface or Repository to see details

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

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

Figure 7: **AMD IP Catalog - Interfaces**

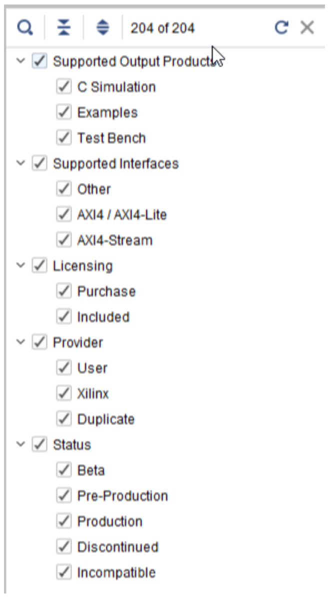


The Interfaces are categorized by how one would store contents in the Vivado Repository when packaging an IP.

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 **Settings** button to open the filter options.

Figure 8: **IP Catalog Filters**



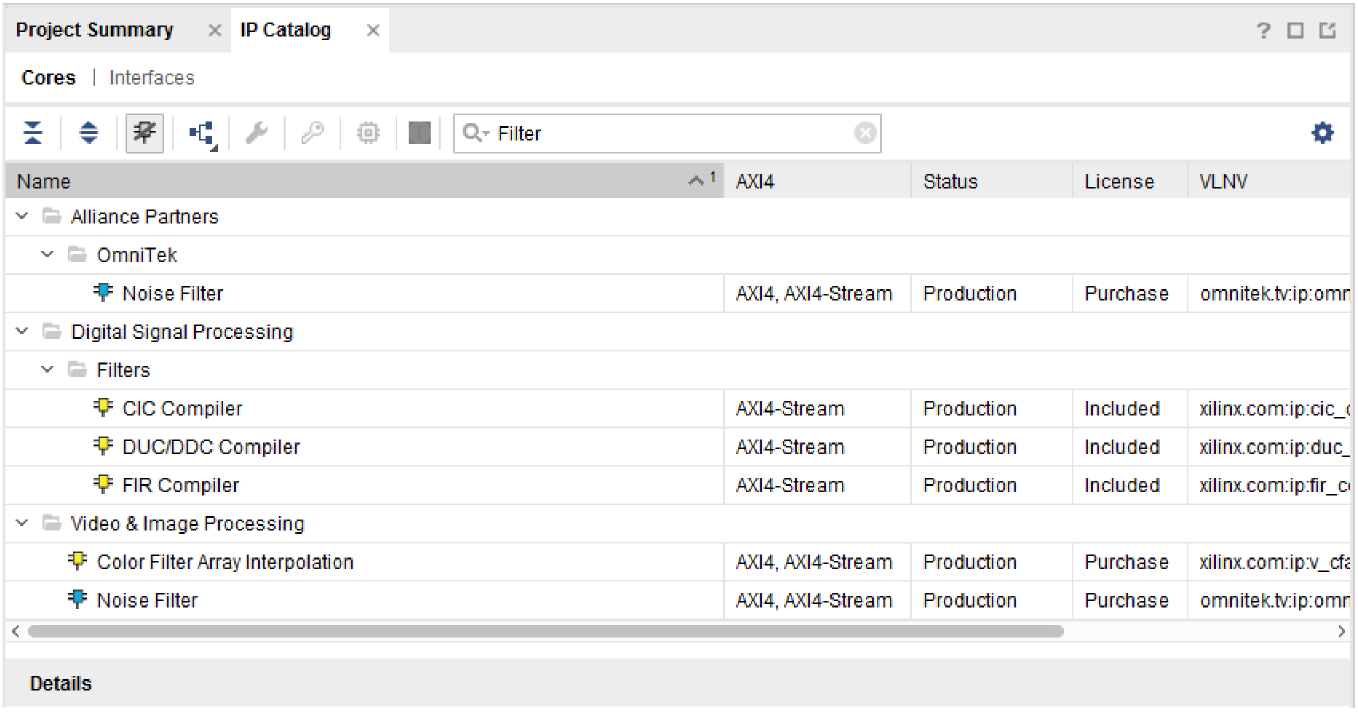
204 of 204

- ☑ Supported Output Products
  - ☑ C Simulation
  - ☑ Examples
  - ☑ Test Bench
- ☑ Supported Interfaces
  - ☑ Other
  - ☑ AXI4 / AXI4-Lite
  - ☑ AXI4-Stream
- ☑ Licensing
  - ☑ Purchase
  - ☑ Included
- ☑ Provider
  - ☑ User
  - ☑ Xilinx
  - ☑ Duplicate
- ☑ Status
  - ☑ Beta
  - ☑ Pre-Production
  - ☑ Production
  - ☑ Discontinued
  - ☑ Incompatible

Uncheck the filter options to filter out the IP that is unnecessary for your project.

The following figure shows the results of a search on IP that have the word "filter" in their name, and IP whose folder contains the word "filter".

Figure 9: **Filtered Search on the Word "Filter"**



Name	AXI4	Status	License	VLNV
Alliance Partners				
OmniTek				
Noise Filter	AXI4, AXI4-Stream	Production	Purchase	omnitek.tv:ip:omr
Digital Signal Processing				
Filters				
CIC Compiler	AXI4-Stream	Production	Included	xilinx.com:ip:cic_
DUC/DDC Compiler	AXI4-Stream	Production	Included	xilinx.com:ip:duc_
FIR Compiler	AXI4-Stream	Production	Included	xilinx.com:ip:fir_c
Video & Image Processing				
Color Filter Array Interpolation	AXI4, AXI4-Stream	Production	Purchase	xilinx.com:ip:v_cfa
Noise Filter	AXI4, AXI4-Stream	Production	Purchase	omnitek.tv:ip:omr

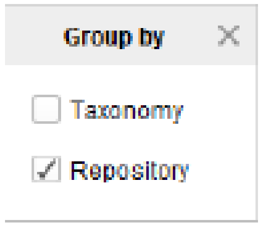
IP Catalog Options

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

When searching in the IP catalog, an IP is shown in the taxonomy structure, even if 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 table shows the symbols that represent IP catalog information.

Table 2: **IP Catalog IP Information**

Option	Description
▨	Hide Incompatible IP
⊞	Group by Usage. Click this button and the **Group by** selection option opens, where you can choose **Taxonomy** or **Repository** or both.  Group by ☐ Taxonomy ☑ Repository. Group by Repository lets you group IP according to AMD IP or user-IP. Group 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.
🔑	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.
ℹ	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 Settings dialog box for the current project and displays the IP settings. See Using IP Project Settings for more information.
🟦	IP available for purchase from select Alliance Partners. See more information in Partner Alliance IP.

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. 🟦

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

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 AMD:

- **License Status**: IP licenses can be **Full** (also know as Purchased), **Simulation**, or **Eval**.
    - **Included**: The AMD End User License Agreement applies to AMD LogiCORE™ IP cores that are licensed within the AMD Vivado Design Suite software tools at no additional charge.
    - **Purchase**: The Core License Agreement applies to fee-based AMD LogiCORE IP, and the Core Evaluation License Agreement applies to the evaluation of fee-based AMD 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).

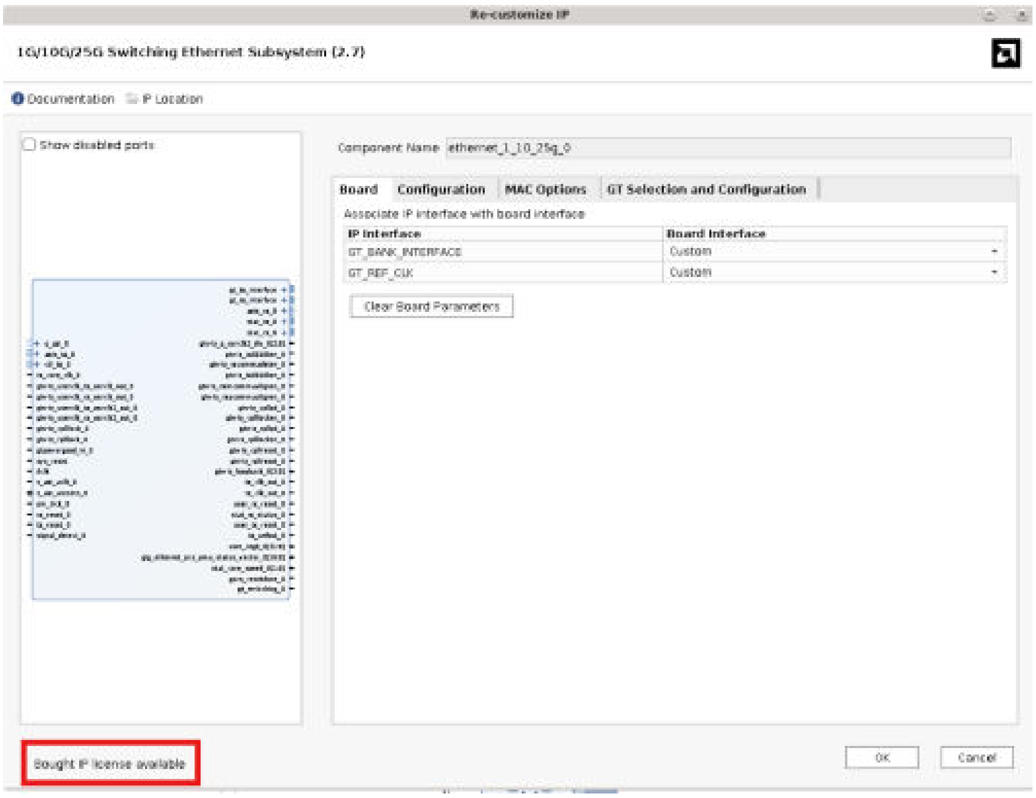
For more information on how to obtain IP licenses, see the AMD *IP Licensing* site.

For more information on licensing see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

VIDEO: *See the following: Vivado QuickTake Video: Vivado Activation and Floating License Generation and Vivado QuickTake Video: Vivado Licensing and Activation Overview for more licensing information.*

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 10: **Fee-Based IP OK Button Enabled after License Found**



Re-customize IP

10G/10G/25G Switching Ethernet Subsystem (2.7)

Documentation   IP Location

Show disabled ports

Component Name: ethernet_1_10_25g_0

Associate IP interface with board interface

IP Interface	Board Interface
GT_BANK_INTERFACE	Custom
GT_REF_CLK	Custom

Bought IP license available

IMPORTANT! *During implementation, the Vivado tools confirms that a license exists for the IP; 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 grey, including the label that precedes or follows it.*

To create an IP customization using the IP catalog, select the IP by double-clicking 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.

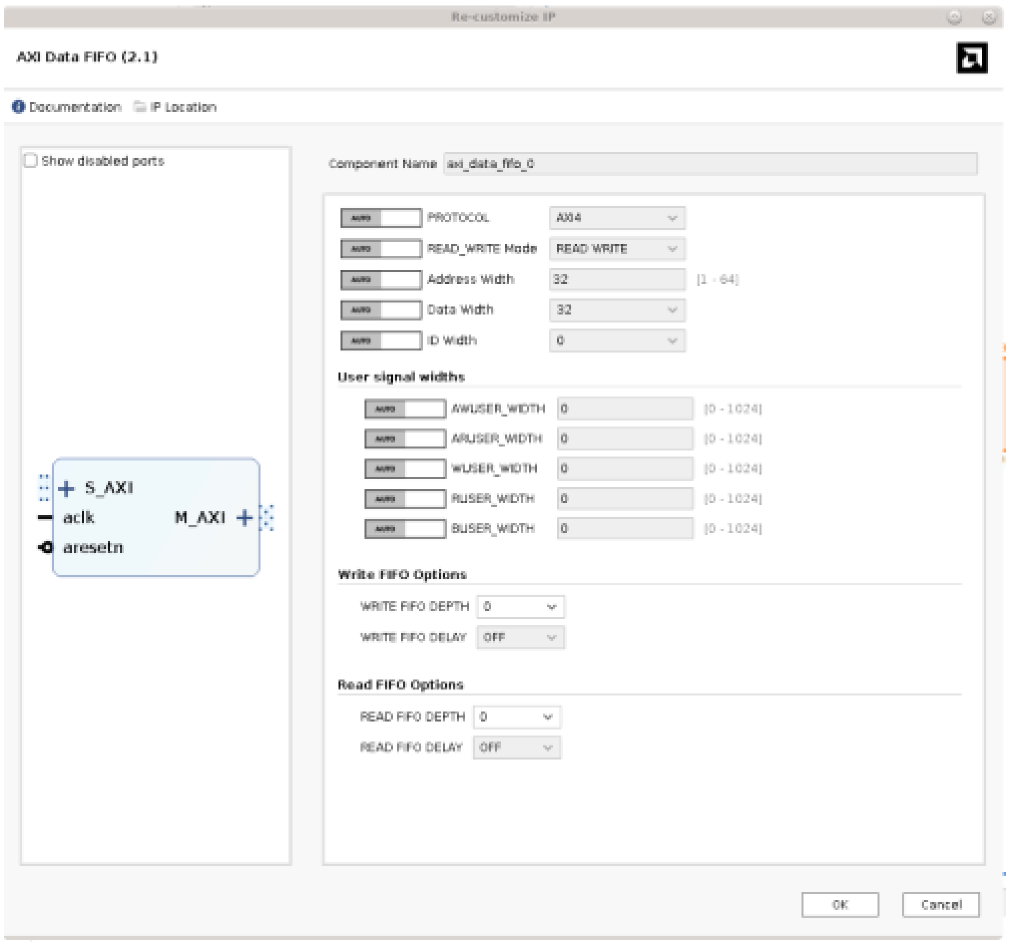
Also, the Customize IP dialog box includes an IP symbol and tabs for setting configuration options for the specific IP. TheVivado IDE writes these configuration options to the `<ip_name>.xci file`, and stores them as properties on the IP object.

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 11: **FIFO Generator Dialog Box**



Component Name axi_data_fifo_0

PROTOCOL	AXI4	
READ_WRITE Mode	READ WRITE	
Address Width	32	[1 - 64]
Data Width	32	
ID Width	0	

User signal widths

AWUSER_WIDTH	0	[0 - 1024]
ARUSER_WIDTH	0	[0 - 1024]
WUSER_WIDTH	0	[0 - 1024]
RUSER_WIDTH	0	[0 - 1024]
BUSER_WIDTH	0	[0 - 1024]

Write FIFO Options

WRITE FIFO DEPTH	0
WRITE FIFO DELAY	OFF

Read FIFO Options

READ FIFO DEPTH	0
READ FIFO DELAY	OFF

In the IP dialog box, set the following options:

- **Documentation > Product Guide**: Open the product guide for the selected IP.
- **IP Location**: Specify the location on disk to store the IP. This location can only be adjusted per IP in an RTL-based project. This functionality is not provided in a Managed IP project because Managed IP builds a repository.
- **Switch to Defaults**: Displays a window asking if you want to reset all configuration options back to their default starting point.

Customize the IP as needed for your design, and click **OK**.

AMD recommends that when specifying a numerical value, use hexadecimal to speed processing.

Tcl Commands for IP

The following are example Tcl commands for use with IP.

Example for Creating an IP Customization

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

create_ip -name fifo_generator -version 12.0 -vendor amd.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, and 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.

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]

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`.

Example of a Query of 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.
# 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.
# See that this is set by the user.
get_property CONFIG.Output_Data_Width.value_src [get_ips char_fifo]
user
```

Generating Output Products

After IP customization is complete, the Generate Output Products dialog box opens. Output products delivered by the IP are listed in the Preview area. Click **Generate**, which creates an XCI and a DCP for the IP, along with a change log, a behavioral simulation model, and an instantiation template; otherwise, 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 OOC runs for the synthesized IP DCPs. In the Generate Output Products dialog box, select one of the following:

- **Global Synthesis**: Instructs the tool to perform top-down synthesis on the current design. All OOC run files are removed when you check this option.
- **Out-of-Context Settings**: Lets you add OOC settings description.

You can also specify the number of OOC synthesis runs to launch at one time. By default, one job is specified, and the design runs launch sequentially. A higher number in the **Number of Jobs** option specifies the maximum number of design runs that can be running in parallel.

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.

Using Tcl Commands to Reset and Generate Target IP

The Tcl command required to reset 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]
```

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>.gen/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 generation process for most Vivado Design Suite IP.

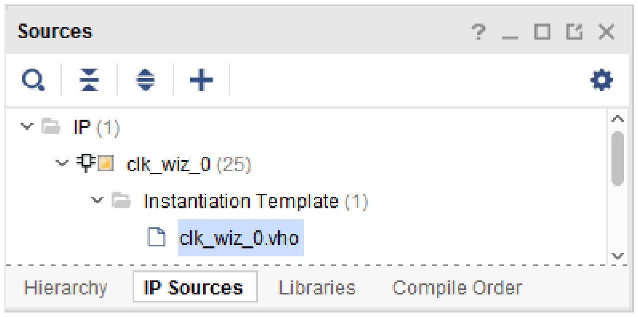
When performing synthesis of the top-level design, IP is marked for the out-of-context flow with an associated DCP file, and treated as a *black box* because it is being synthesized OOC.

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

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

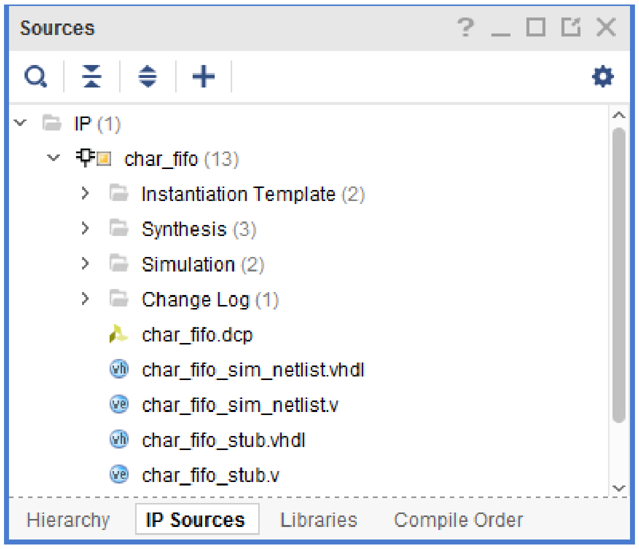
If you do not generate output products, the instantiation templates are the only generated product (besides the XCI and BOM files, which are not displayed) shown in the following figure.

Figure 12: **IP Customization with Generation of Output Products Skipped**



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

Figure 13: **IP Customization with Output Products Generated, Including OOC DCP**


Sources

IP (1)
- char_fifo (13)
  - Instantiation Template (2)
  - Synthesis (3)
  - Simulation (2)
  - Change Log (1)
  - char_fifo.dcp
  - char_fifo_sim_netlist.vhdl
  - char_fifo_sim_netlist.v
  - char_fifo_stub.vhdl
  - char_fifo_stub.v

These files include: the instantiation templates, 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`/`*_stub.vhdl`) for use with third-party synthesis tools to infer a black box for the IP.

Note: In versions of 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 by selecting the IP from IP Sources, right-clicking and selecting **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 OOC 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>]
```

AMD 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.

Adding Existing IP to a Project

You can add IP that was previously created in the CORE Generator tool (`<ip_name>.xco` files) or Vivado IP (`<ip_name>.xci` or `<ip_name>.xcix` files) by using the **Add Sources** option. 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, and generate any needed output products. See Chapter 3: Using Manage IP Projects for details on creating IP using the Managed IP flow.

The added IP and any output products show in the IP Sources tab of the Sources window, and 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 AMD UltraScale™ devices. AMD 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 the ISE to Vivado Design Suite Migration Guide (UG911) 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: regenerate the output products for the IP, or 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. The added IP also needs to be instantiated into the top-level design as described in Instantiating an IP.

Adding Existing IP using 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: `import_files <ip_filename>`

- Use the `read_ip` to remotely access an IP: `read_ip <ip_filename>`

Resolving 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.

Creating a Memory IP Customization

The Memory IP creates memory controllers for AMD devices and IP. Memory IP creates complete customized 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 deprecated with the AMD UltraScale™ and UltraScale+ devices. See the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150), and *Zynq 7000 SoC and 7 series Devices Memory Interface Solutions* (UG586) for more information.

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

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

The Designing with IP Design Hub in the AMD 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. To open the IP customization dialog box for an IP, either double-click the IP, or in the **IP Sources** view, right-click the IP, and select **Re-customize IP** from the context menu.

Change IP configuration settings, and click **OK**. The Generate Output Products dialog box opens for you to re-generate output products. 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.

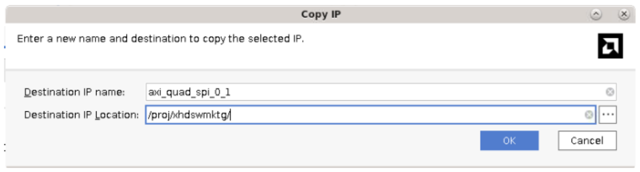
AMD recommends that you always generate the output products for IP, including the synthesized DCP; however, if you do not generate output products after customizing, or re-customizing the IP, the Vivado Design Suite generates the output products automatically, as needed; for example, when synthesizing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

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, in IP Sources, select the IP, right-click and select **Copy IP**. Next provide a destination name and location for the copy, as shown in the following figure.

Figure 14: **Copy IP Dialog Box**



Enter a new name and destination to copy the selected IP.

Destination IP name: axi_quad_spi_0_1

Destination IP Location: /proj/xhdswmktg/

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 `/manage_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.

Instantiating an IP

The Vivado tools create instantiation templates after IP customization, regardless of whether you generated the output products. The instantiation templates are displayed under the `Instantiation Template` directory on the IP Sources view of the Sources window.

After you create an IP customization, open the IP instantiation template and copy the relevant code from the template into your code. The Vivado tool generates both a VHDL and a Verilog instantiation template that you select from and copy and paste into your RTL design.

To use the instantiation template in your design, do 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, and give the instantiation a unique name.

After the IP is instantiated into a design, the IP is listed correctly in the design hierarchy. 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 can see the Encrypted IP source icon . The content of this source cannot be viewed. See the Vivado Design Suite User Guide: Creating and Packaging Custom IP (UG1118) for information on Encypting IP in Vivado.*

IMPORTANT! *It is possible to create duplicate IP. See Resolving Duplicate IP if you have duplicates.*

Reporting IP Status

You can view the report of all IP in a project using the **Reports→Report IP Status**. A new tab titled **IP Status** displays the report results.

Filtering Status

You can filter the information in the IP Status report by right-clicking the Source file to display the options list. The options are, as follows:

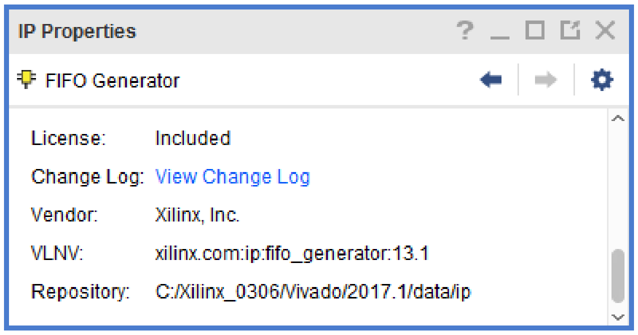
- Major change
- Minor change
- Revision change
- Part change
- Up-to-date
- Other: This category consists of IP in miscellaneous states. The following are examples:
    - IP definition not found
    - Read-only XCI, XML, or XPR file
    - User-managed IP
    - Disabled component
    - Incompatible license
    - 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, and right-click and select **IP Documentation → View Change Log**.
- In the Source Files Properties view, select the IP, and scroll down and select **View Change Log**. You can select the view full log link to open the full change log, as shown in the following figure.

Figure 15: **View Change Log Link**



License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.
VLNV:	xilinx.com:ip:fifo_generator:13.1
Repository:	C:/Xilinx_0306/Vivado/2017.1/data/ip

Reporting IP Status using a Tcl Command

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

```
report_ip_status
```

Designing with System Verilog

The support for System Verilog wrappers has been added from 2025.2 release onwards. System Verilog wrapper support for BD and .xci files involves enabling tools and features that allow for the creation and customization of HDL wrappers in System Verilog for Block Designs (BD) and Intellectual Property (IP) .xci files. Here are the key details:

GUI Updates

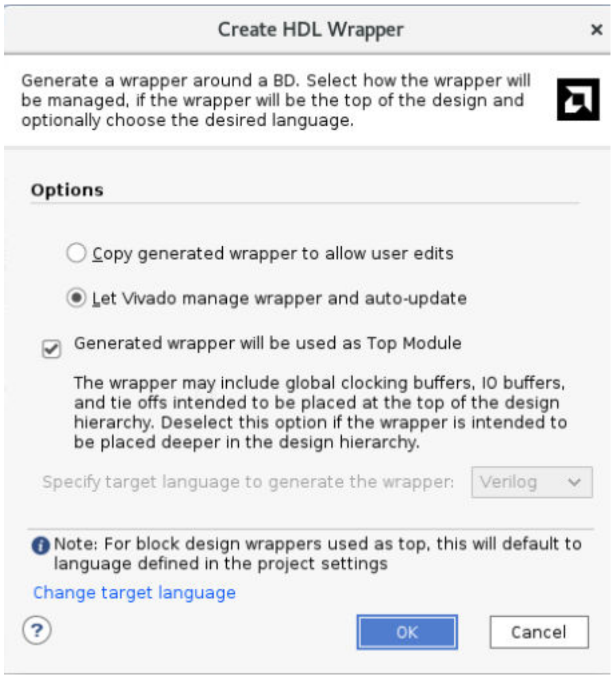
- The GUI for generating System Verilog wrappers has been enhanced in Vivado to include options for wrapper generation across different types, including System Verilog, Verilog, and VHDL.
- The Create HDL Wrapper dialog has been updated to support System Verilog as one of the languages to generate wrapper. Target language options are dynamically customizable based on user selections.

In previous builds, the Create HDL Wrapper dialog in Vivado used to support Verilog and VHDL but used to not allow the user to generate System Verilog wrappers for BDs. This lack of support limited flexibility and prevented streamlined integration with modern workflows that involve System Verilog interfaces.

Dialog Box Updates

BD Updates: The Create HDL Wrapper dialog will be revamped to support System Verilog wrappers for BDs.

Figure 16: **Create HDL Wrapper**



Generate a wrapper around a BD. Select how the wrapper will be managed, if the wrapper will be the top of the design and optionally choose the desired language.

Options

( ) Copy generated wrapper to allow user edits

(●) Let Vivado manage wrapper and auto-update

[✓] Generated wrapper will be used as Top Module

The wrapper may include global clocking buffers, IO buffers, and tie offs intended to be placed at the top of the design hierarchy. Deselect this option if the wrapper is intended to be placed deeper in the design hierarchy.

Specify target language to generate the wrapper: Verilog

Note: For block design wrappers used as top, this will default to language defined in the project settings

Below is the functionality:

Checkbox Feature: Generated wrapper will be used as Top Module.

When Checked: The wrapper's language will default to the target language defined in the **Project Settings** > **General** > **Target Language**.

When Unchecked: Users can manually select the language for wrapper generation from three options:

- System Verilog
- Verilog
- VHDL

Dropdown Language Selection: Dynamically enabled based on checkbox state:

- Default Behavior: Displays the current target language.
- Custom Selection: Allows manual choice of preferred language.

Workflow Integration

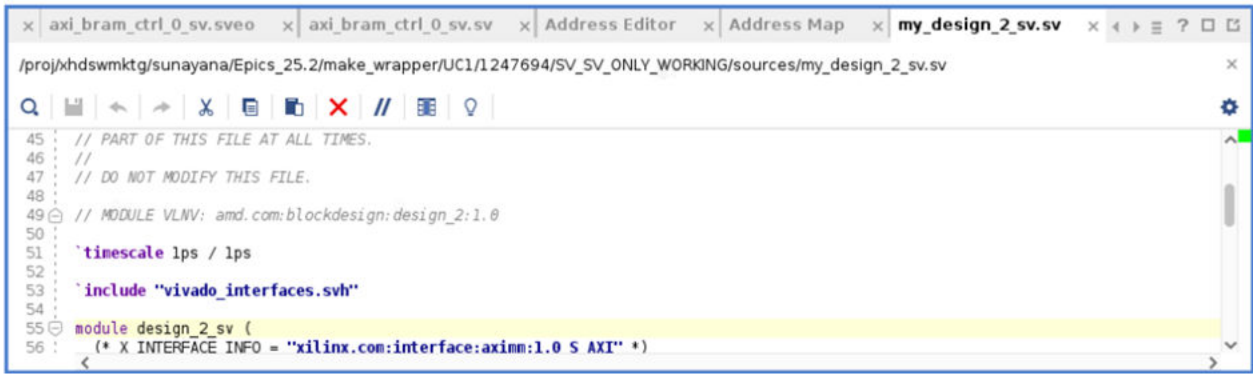
- Allows easy navigation between Project Settings to adjust target language and the Create HDL Wrapper dialog.

- Support dynamic generation of wrapper files using Tcl commands executed in the backend:

  ```
  make_wrapper -language system_verilog -inst_template [get_files my_bd.bd]
  ```

System Verilog Wrapper will have an extension (*_sv.sv).

Figure 17: **Example System Verilog Generated Wrapper**


axi_bram_ctrl_0_sv.sveo | axi_bram_ctrl_0_sv.sv | Address Editor | Address Map | my_design_2_sv.sv

/proj/xhdswmktg/sunayana/Epics_25.2/make_wrapper/UC1/1247694/SV_SV_ONLY_WORKING/sources/my_design_2_sv.sv

```
// PART OF THIS FILE AT ALL TIMES.
//
// DO NOT MODIFY THIS FILE.

// MODULE VLNV: amd.com:blockdesign:design_2:1.0

`timescale 1ps / 1ps

`include "vivado_interfaces.svh"

module design_2_sv (
  (* X_INTERFACE_INFO = "xilinx.com:interface:aximm:1.0 S_AXI" *)
```

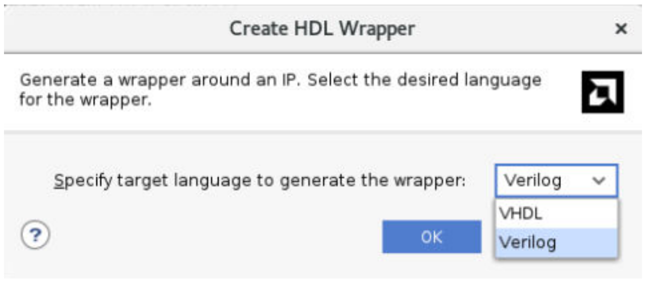
System Verilog Wrapper (*_sv.sv) content:

- File registered using 'add_files' command (for example, 'add_files c:/path/to/project_1.gen/sources_1/ip/my_ip/my_ip_sv.sv')
- File sourced directly from IP/BD generation directory (for example, c:/path/to/project_1.gen/sources_1/ip/my_ip/my_ip_sv.sv)
- Wrapper will automatically update when IP or BD is regenerated, when the file is rewritten in place.

***Note*:** System Verilog cannot be used as Top level wrapper.

Updates for .xci

Figure 18: **Create HDL Wrapper for .xci**


Create HDL Wrapper

Generate a wrapper around an IP. Select the desired language for the wrapper.

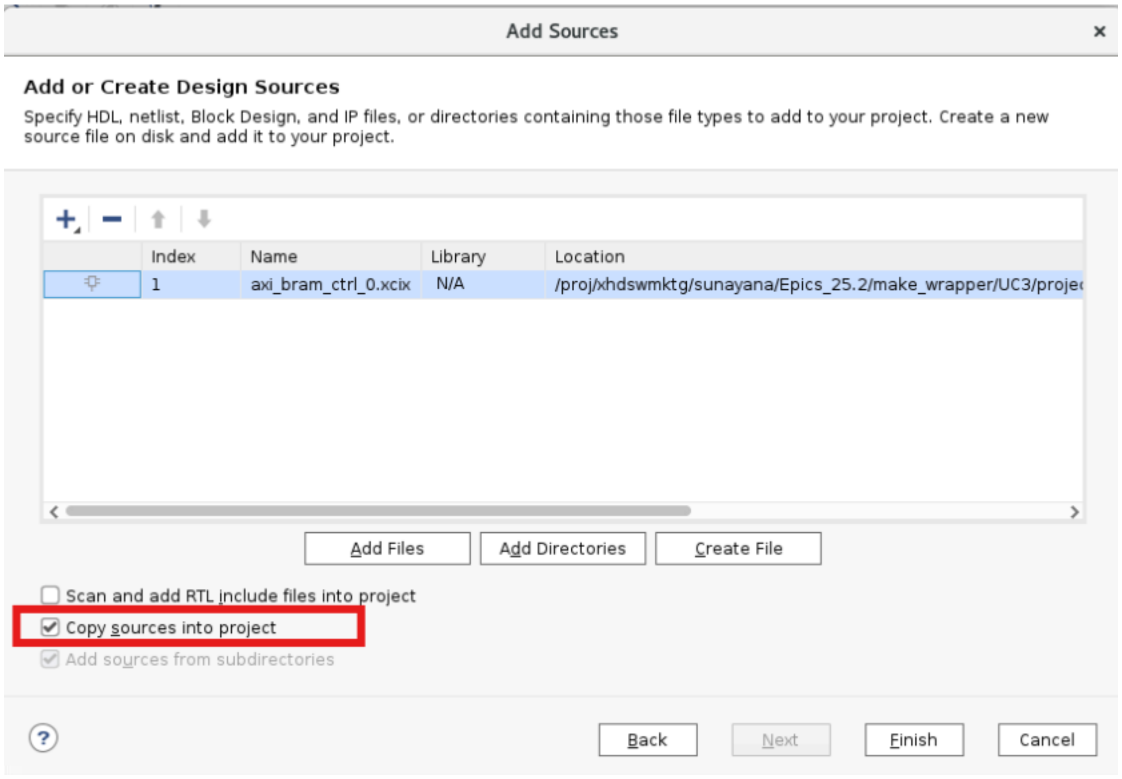
Specify target language to generate the wrapper: Verilog

Tcl commands to generate the wrapper:

The make_wrapper command in Vivado plays a critical role in generating HDL wrappers for different components like Block Design (BD), .xci, and .xcix files. Here are the key details about the command and how it is used.

Command Overview (TCL)

The make_wrapper command is used to generate wrapper files in different HDL languages. With the addition of System Verilog support, make_wrapper now includes options for generating System Verilog wrappers and templates for both BD and .xci/.xcix files.



Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
	1	axi_bram_ctrl_0.xcix	N/A	/proj/xhdswmktg/sunayana/Epics_25.2/make_wrapper/UC3/proje

☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

```
copy .srcs checked ->import
make_wrapper [get_files
my_ip.xci] -language systemverilog -import

Uncheck copy .srcs
-> add
make_wrapper [get_files
my_ip.xci] -language systemverilog -add

import_files
/path/to/sources/top.sv

set_property top
"top" [current_fileset]
```

Note: The make_wrapper command is integrated with the Vivado GUI. When right-clicking on a BD, .xci, or .xcix file, the "Create HDL Wrapper" option invokes the command.

***Note*:** Core container should be enabled for the IP to create .xcix version and then the make_wrapper command should be used for the .xci core.

Instantiation templates .sveo for BD and XCI

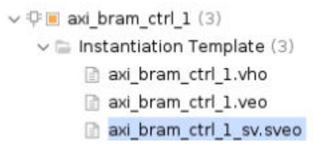
The .sveo file is a System Verilog instantiation template file generated by Vivado. The .sveo file includes pre-configured instantiation examples for the module defined in the wrapper, making it easier for users to integrate the System Verilog wrapper into their top-level design.

Purpose of .sveo Files

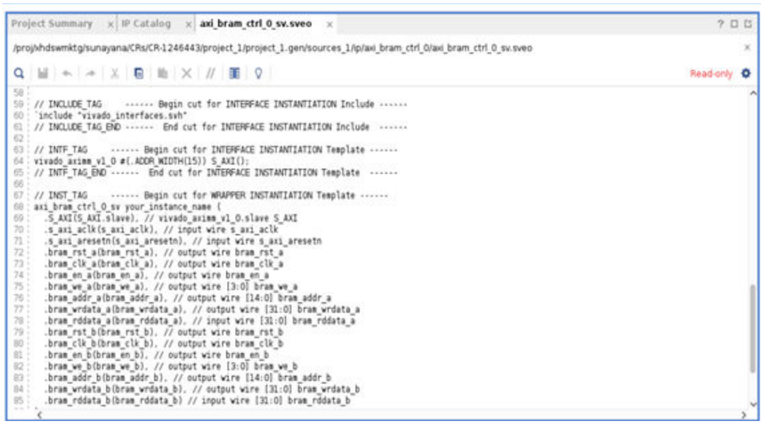
1. Quick Instantiation:
   - .sveo files provide pre-generated instantiation templates that can be directly used in your top-level System Verilog design.
   - They define the module ports, parameter values, and connection patterns for interfaces such as AXI Memory Mapped or AXI Streaming.
2. Improved Workflow Integration:
   - Eliminates manual effort in creating instantiation code for wrappers.
   - Ensures consistency across instances generated from block designs (BDs) or IP cores.
3. Usage for System Verilog Wrappers:
   - When creating wrappers for BDs or .xci/.xcix files, .sveo files are automatically generated alongside the wrapper files.
4. Benefits:
   - Time-Saving: Reduces the effort required to instantiate complex modules manually.
   - Consistency: Ensures that instances are accurately parameterized and connected, avoiding errors in integration.
   - Ease of Debugging: Provides standard instantiations that simplify the debugging process during simulation and synthesis.

Tcl command: make_wrapper -language system_verilog -inst_template [get_files my_ip.xci].

Generated Files:



```
axi_bram_ctrl_1 (3)
  Instantiation Template (3)
    axi_bram_ctrl_1.vho
    axi_bram_ctrl_1.veo
    axi_bram_ctrl_1_sv.sveo
```

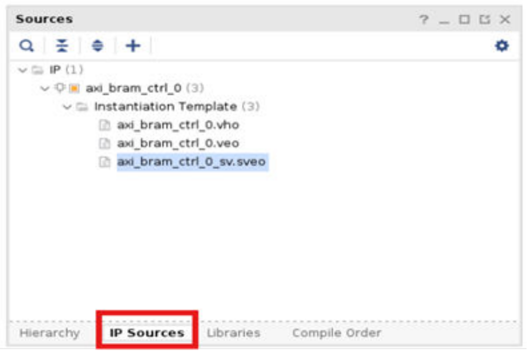


/proj/xhdswmktg/sunayana/CRs/CR-1246443/project_1/project_1.gen/sources_1/ip/axi_bram_ctrl_0/axi_bram_ctrl_0_sv.sveo

```
// INCLUDE_TAG     ------ Begin cut for INTERFACE INSTANTIATION Include ------
`include "vivado_interfaces.svh"
// INCLUDE_TAG_END ------  End cut for INTERFACE INSTANTIATION Include  ------

// INTF_TAG     ------ Begin cut for INTERFACE INSTANTIATION Template ------
vivado_aximm_v1_0 #(.ADDR_WIDTH(15)) S_AXI();
// INTF_TAG_END ------  End cut for INTERFACE INSTANTIATION Template  ------

// INST_TAG     ------ Begin cut for WRAPPER INSTANTIATION Template ------
axi_bram_ctrl_0_sv your_instance_name (
  .S_AXI(S_AXI.slave), // vivado_aximm_v1_0.slave S_AXI
  .s_axi_aclk(s_axi_aclk), // input wire s_axi_aclk
  .s_axi_aresetn(s_axi_aresetn), // input wire s_axi_aresetn
  .bram_rst_a(bram_rst_a), // output wire bram_rst_a
  .bram_clk_a(bram_clk_a), // output wire bram_clk_a
  .bram_en_a(bram_en_a), // output wire bram_en_a
  .bram_we_a(bram_we_a), // output wire [3:0] bram_we_a
  .bram_addr_a(bram_addr_a), // output wire [14:0] bram_addr_a
  .bram_wrdata_a(bram_wrdata_a), // output wire [31:0] bram_wrdata_a
  .bram_rddata_a(bram_rddata_a), // input wire [31:0] bram_rddata_a
  .bram_rst_b(bram_rst_b), // output wire bram_rst_b
  .bram_clk_b(bram_clk_b), // output wire bram_clk_b
  .bram_en_b(bram_en_b), // output wire bram_en_b
  .bram_we_b(bram_we_b), // output wire [3:0] bram_we_b
  .bram_addr_b(bram_addr_b), // output wire [14:0] bram_addr_b
  .bram_wrdata_b(bram_wrdata_b), // output wire [31:0] bram_wrdata_b
  .bram_rddata_b(bram_rddata_b) // input wire [31:0] bram_rddata_b
```

Location: `project_1/project_1.gen/sources_1/ip/axi_bram_ctrl_1/axi_bram_ctrl_1_sv.sveo`.

Common Scenarios

1. Block Design (BD) Integration:
   - When generating wrappers for BDs, .sveo templates simplify the use of System Verilog instances by defining how the module fits into the top-level design.
2. In IP Flows:
   - .sveo templates standardize instantiations for Xilinx IP such as AXI block RAM Controller or AXI DMA.
3. 

Path: `./project_1.gen/sources_1/ip/my_ip/my_ip_sv.sveo`

System Verilog XPM

From this release, SV XPMs are included in Language Template section of Vivado.

In Vivado 2025.2, System Verilog (SV) templates are integrated into the "Language Template" section as part of efforts to streamline the design and instantiation workflow for Xilinx Parameterized Macros (XPMs).

Currently, System Verilog templates for XPM modules like xpm_noc, xpm_fifo, or xpm_memory are included under Tools > Language Templates > System Verilog.

Templates will include:

- Module Definitions: For quick instantiation into RTL designs.
- Interface Definitionss: Pre-configured ports aligned with Vivado's AXI interfaces (AXI MM, AXI Lite, AXI Stream).
- Automation in Vivado: The tool will auto-detect the use of XPM instantiated in the RTL design and provide related templates under the System Verilog hierarchy.
- Examples for Common Use Cases: Example SV templates such as:

  1. xpm_fifo
     - xpm_fifo_axil_sv
     - xpm_fifo_axif_sv
     - xpm_fifo_axis_sv
  2. xpm_noc
     - xpm nmu_strm_sv
     - xpm_nsu_strm_sv
     - xpm_nmu_mm_sv
     - xpm_nsu_mm_sv

Access via Language Templates:

Navigate to **Tools** > **Language Templates** > **System Verilog** > **XPMs** in Vivado. Select the desired XPM templates for integration into your top-level design.

NOC Simulation

Simulating a Network-on-Chip (NOC) design with Vivado involves the use of System Verilog wrappers and instantiation templates. With recent enhancements, System Verilog support has been added for components such as Xilinx Parameterized Macros (XPM) and block designs (BD), simplifying NOC integration with System Verilog interfaces.

Use Case Benefits

1. Instance Management: Easily manage user simulation wrapper and xlnoc simulation wrapper.
2. Simplified Design Entry: Use generated templates (.sveo) for instantiation, reducing manual effort.
3. Flexibility Across Standards: Mix System Verilog and Verilog components seamlessly in parallel top-level designs.

Understanding IP States Within a Project

There are several states that an IP can display within in a project, depending on 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 3: **IP States in a Project**

Button	Description
	IP in an RTL project to be synthesized 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 is 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.

Managing IP Constraints

The Vivado IDE manages both user-defined XDC timing and physical constraints for the entire design, and for AMD IP. It handles the association and the unification of constraints for AMD 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.

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 OOC run, you can use the following `set_property` command in the Tcl console:

```
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 <ip_name>_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.

CAUTION! *If any IP is synthesized in OOC mode, the top level synthesis run infers a black box for these IP. Users might not be able to reference objects such as pins, nets, and cells that are internal to the IP as part of the top level synthesis constraints. 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.*

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:

- The *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
- The *Vivado Design Suite User Guide: Using Constraints* (UG903)

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.

The following is an example of the `report_compile_order` Tcl command to report constraint 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 with commands without external clock dependencies. The constraint file name is `<ip_name>.xdc`, and is referred to as the *core* XDC file.

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 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 in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895).

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 the *Vivado Design Suite User Guide: Using Constraints* (UG903).

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).

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

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

dont_touch.xdc Constraint

The Vivado tools use 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 Constraint

By default, IP is treated as a black box during top-level synthesis because it is synthesized using OOC. 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 your 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 the 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 your 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.

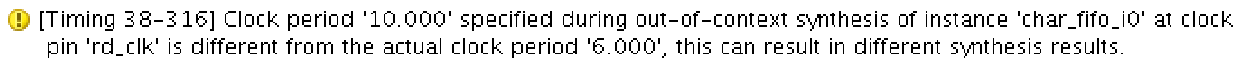
Setting the Target Clock Period

By default, IP are synthesized standalone, and OOC from the rest of a design.

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

Because the IP is synthesized separate from the top-level, the Vivado tools create the `get_runs <ip_name>_ooc.xdc` after OOC is complete 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 OOC IP.

IP have a variety of clock options:

- IP might have options in the customization GUI to set the target frequency/period to be used during OOC synthesis. You can use the Tcl Console to query and set the configuration option for the IP.
- IP might have a tab labeled Clocks, which lets you set the target frequency for the IP.
- IP might have the setting of the target mixed in with other settings. Typically, there is a tool tip to explain the setting.

- IP that does not provide a GUI option to customize the target clock frequency or period must rely upon other clock sources for frequency.

A clock port of an IP has a property associated with it ending with FREQ_HZ. Changing these properties results 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

To customize an IP, set unique properties on the IP object. When you use the Vivado IDE to configure an IP, the Vivado IDE issues the corresponding Tcl commands automatically. If an IP does not provide a GUI method for setting the target period for an IP in OOC synthesis, you can set it manually.

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

The following is an example of the steps to set the target clock for the FIFO generator IP called `char_fifo`, which is used in the Wave Generator example design. 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 in 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 (this 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 output 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 100 MHz value of the `CONFIG.core_clk.FREQ_HZ`.

   For this example, with a desired clock frequency of 250 MHz, 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 OOC 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.

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.

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

If the necessary clock constraint is not being provided, the IP at the top-level of the design issues a CRITICAL WARNING as described in Setting the Target Clock Period.

For more information about working with the designs with clocking requirements, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

VIDEO: *See the following for more information:*

- Vivado Design Suite QuickTake Video: Creating Basis Clock Constraints
- Vivado Design Suite QuickTake Video: Creating Generated Clock Constraints

Tcl Command Example of an IP Clock Dependency

The following is an example of an IP constraint that is using the `set_max_delay` command, which has a dependency on a top-level clock, which is provided by the IP on the `ref_clk` port.

```
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), 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_clocks_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

  ```
  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'.
```

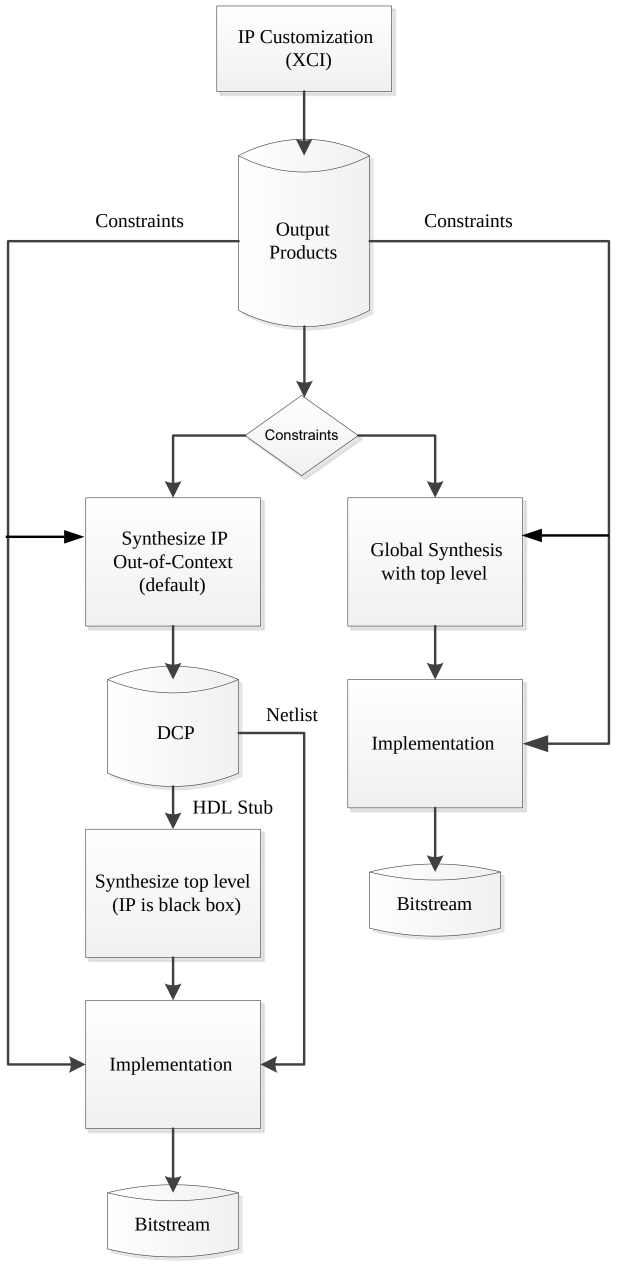
```
[C:/Design/v_tc.xdc:1]Resolution: The get_ports call is being converted to
a get_pins call
as there is no direct connection to a top level port. This could be due to
the insertion of
IO Buffers between the top level terminal and cell pin. If the goal is to
apply constraints
that will migrate to top 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.
```

Synthesis Options for IP

When generating the output products for an IP, the default behavior is to produce an 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*.

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

Figure 19: **Out-Of-Context (OOC) and Global Synthesis Flow**



Global Synthesis 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.

When you select the global synthesis flow, the Vivado tools synthesize the IP along with user HDL. Any changes made to user HDL result in the IP being re-synthesized as well.

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

Always reference the IP using the XCI file. It is not recommended to only read 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 or COE files, and Tcl scripts.

Out-of-Context Flow

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 runtime for the complete project, and lets you avoid re-synthesizing IP when doing project runs.

In the OOC flow, the Vivado tools synthesize the IP as a standalone module and produces an OOC design checkpoint (DCP). The Vivado tools also generate and use a special OOC flow-only, the Xilinx design constraint (XDC) output product file, the `_ooc.xdc`, when synthesizing the IP, which provides default input clock definitions. The produced DCP is a container file, and includes a netlist and constraints. The `_ooc.xdc` file is part of the IP definition.

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

CAUTION! *If any IP is synthesized in OOC mode, the top level synthesis run infers a black box for these IP. Hence, users are not able to reference objects such as pins, nets, cells, etc., that are internal to the IP as part of the top level synthesis constraints.*

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 runtimes 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`.

Simulating IP

Using simulation is an important and necessary step in the design flow to verify the functionality and performance of the design. When IP output products are generated, several simulation models are created that you can include in the simulation of the overall design.

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

- Custom behavioral simulation model.
- Plain text or encrypted synthesizable RTL sources used for simulation.
- Structural simulation model.
- C 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 AMD devices are integrated as options in the AMD Vivado™ Integrated Design Environment (IDE). See the Vivado Design Suite User Guide: Logic Simulation (UG900) for more information on working with third-party simulators. The files are located in the ip_user_files directory.*

Delivering IP Simulation Models

Most AMD 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`.

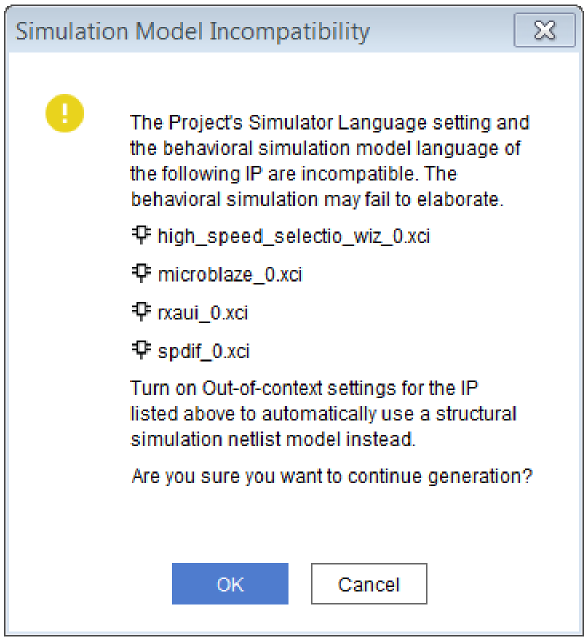
When you generate IP output products, enable the synthesized design checkpoint (DCP) option to ensure that the Vivado IDE can deliver a structural simulation netlist for the IP. For more information, see Generating Output Products.

Note: Some AMD 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 launches automatically, if needed.

To have all files required for simulation available in the IP `<project_name>.gen` directory for placement in a revision control system it is recommended you run synthesis first.

If your simulator language is not set to **Mixed**, you might be required to generate the IP using the default OOC 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 created as part of the OOC synthesis flow only. The following message displays when you have a mismatch between available simulation files and the Simulation Language setting.

Figure 20: **Simulation Model Incompatibility Dialog Box**


Simulation Model Incompatibility

The Project's Simulator Language setting and the behavioral simulation model language of the following IP are incompatible. The behavioral simulation may fail to elaborate.

high_speed_selectio_wiz_0.xci

microblaze_0.xci

rxaui_0.xci

spdif_0.xci

Turn on Out-of-context settings for the IP listed above to automatically use a structural simulation netlist model instead.

Are you sure you want to continue generation?

Verification IP

Verification IP can be helpful when performing simulation on designs that are using AXI IP. See the following documents for more information:

- *AXI Verification IP LogiCORE IP Product Guide* (PG267)
- *AXI4-Stream Verification IP LogiCORE IP Product Guide* (PG277)
- *Zynq UltraScale+ MPSoC Verification IP Data Sheet* (DS940)
- *Zynq 7000 SoC Verification IP Data Sheet* (DS941)

IMPORTANT! *The AXI Verification IP is written in SystemVerilog and uses randomization. Not all third-party simulators support SystemVerilog and randomization. Check Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973) for information about third-party compatibility to the AXI VIP.*

VIDEO: *Also, the following Video is available to help you understand how to use the Zynq 7000 VIP for simulation: Vivado Design Suite QuickTake Video: How to Use the Zynq 7000 Verification IP to Verify and Debug using Simulation.*

You can instantiate the IP. If additional Verification IP support for interfaces is required, use one of the following Tcl commands:

```
set_property CONFIG.INSERT_VIP 1 [get_bd_intf_pin <path_to_interface>]
set_property CONFIG.<interface_name>.INSERT_VIP 1 [get_ips <ipname>]
```

Tcl Commands for Simulation

To specify the simulator language, type the following command in the Tcl Console:

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

The following table shows the simulation language properties, language, and simulation model where the property is applied.

Table 4: **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`.

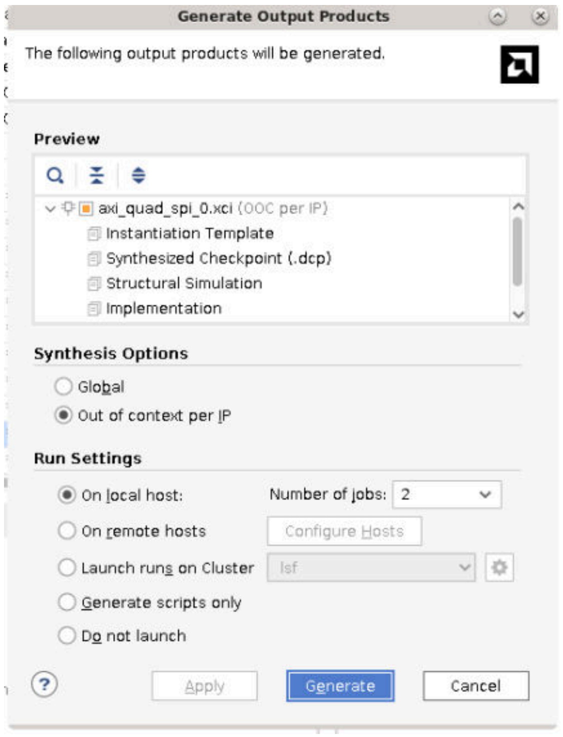
Note: Where available, a *Behavioral Simulation* model always takes precedence over a *Structural Simulation netlist*. Vivado does not offer a choice of simulation model.

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 21: **Test Bench Output Product**



Generate Output Products

The following output products will be generated.

Preview

axi_quad_spi_0.xci (OOC per IP)
- Instantiation Template
- Synthesized Checkpoint (.dcp)
- Structural Simulation
- Implementation

Synthesis Options

- Global
- Out of context per IP [✓]

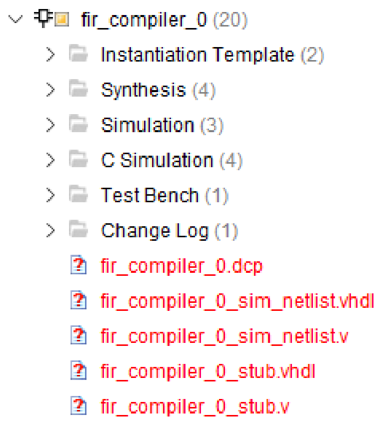
Run Settings

- On local host: [✓] Number of jobs: 2
- On remote hosts (Configure Hosts)
- Launch runs on Cluster (lsf)
- Generate scripts only
- Do not launch

To use the test bench provided by the IP, in the Sources window, find the IP in the hierarchy in the Simulation Sources section and expand the IP hierarchy.

The files in the Hierarchy tab of the sources window display as shown in the following figure.

Figure 22: **Expanding the IP Hierarchy**



fir_compiler_0 (20)
- Instantiation Template (2)
- Synthesis (4)
- Simulation (3)
- C Simulation (4)
- Test Bench (1)
- Change Log (1)
- fir_compiler_0.dcp
- fir_compiler_0_sim_netlist.vhdl
- fir_compiler_0_sim_netlist.v
- fir_compiler_0_stub.vhdl
- fir_compiler_0_stub.v

***Note*:** Files are red are because an OOC Synthesis has yet to be run.

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

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 by clicking on **Settings** and navigating to 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

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.*

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.

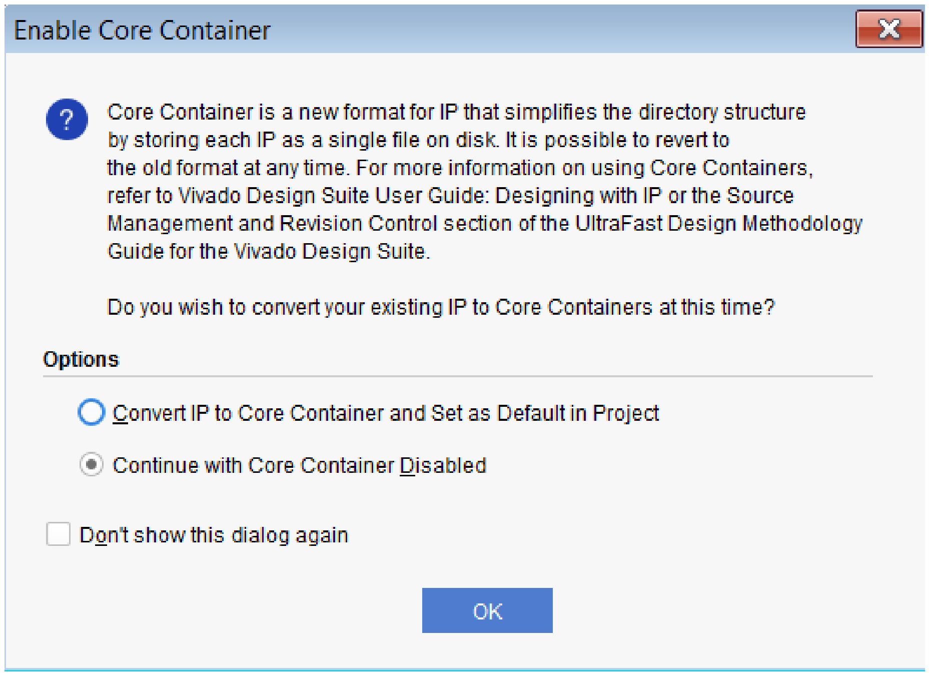
- If you are using Manage IP projects, copy the entire Manage IP project location as a backup.
- Archive design projects that contain IP.
- Before upgrading an IP, view the **report_ip_status** window for information on the changes.

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 and you can upgrade the IP. The following figure shows the dialog box that opens and asks if you want to continue with the Core Container feature disabled.

***Note*:** The **Core Container** option is no longer available starting in the 2023.1 release.

Figure 23: **Enable Core Container Dialog Box**



Enable Core Container

Core Container is a new format for IP that simplifies the directory structure by storing each IP as a single file on disk. It is possible to revert to the old format at any time. For more information on using Core Containers, refer to Vivado Design Suite User Guide: Designing with IP or the Source Management and Revision Control section of the UltraFast Design Methodology Guide for the Vivado Design Suite.

Do you wish to convert your existing IP to Core Containers at this time?

Options

[ ] Convert IP to Core Container and Set as Default in Project

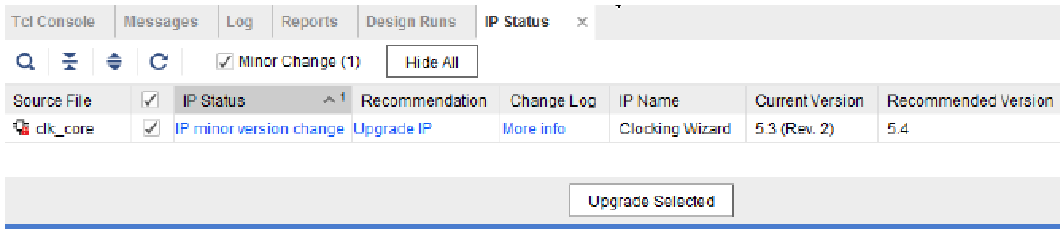
[✓ Continue with Core Container Disabled]

[ ] Don't show this dialog again

Select the **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.

The following figure shows the IP Status tab with a message that an upgrade is recommended for the IP.

Figure 24: **Report IP Status with Upgrade IP Option**



Source File	✓	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version
clk_core	✓	IP minor version change	Upgrade IP	More info	Clocking Wizard	5.3 (Rev. 2)	5.4

Click the **Upgrade IP** option next to the IP. The report provides information about all the IP in the project. Click any blue, underlined text 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.
- **IP Name**: Name of the IP.
- **Current Version**: Provides the current version of the IP.
- **Recommended Version**: Gives the recommended version of the IP to which to upgrade.

Major version changes could require modification to the RTL that connects to the IP. The IP Status window contains the following:

- **IP Name**: Name of the IP as shown in the IP catalog.
- **Current Version**: Provides the current version of the IP.
- **Recommended Version**: Provides the recommended version of the IP to which to upgrade.
- **License**: Shows the status of the IP license.
- **Current Part**: Part used in the design.

You can check the box next to the **Source File** to selectively upgrade IP. Checking the box in the column selects all IP that have an available upgrade. 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.

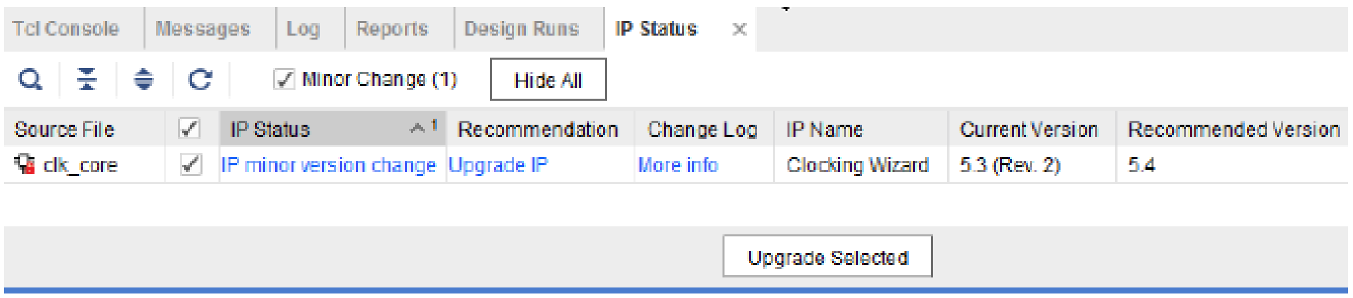
Selectively Upgrading IP

Selective upgrade of IP is available for IP that has already been generated in a previous Vivado version. The DCP of the non-upgraded IP is brought in as `LOCKED` and un-modifiable. This feature lets you interact with a block design even if all IP are not upgraded, and lets you generate bitstreams even with some or all locked IPs (assuming the OOC DCPs for those locked IPs are available).

***Note*:** The IP Parameter propagation is limited, see the information on selectively upgrading IP in Block Designs in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994).

When you selectively upgrade IP, the Report IP Status window looks like in the following figure:

Figure 25: **Report IP Status with Selective IP Upgraded**



Source File	✓	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version
clk_core	✓	IP minor version change	Upgrade IP	More info	Clocking Wizard	5.3 (Rev. 2)	5.4

For more information on updating designs for a new release, see the Updating Designs for a New Release section in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994). For information about upgrading IP in a block design (BD), see the Editing a Packaged Block Design section in the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) for information on editing a packaged BD.

Upgrading an IP in a design creates an *Update log*. These logs are available from the `IP/Upgrade Log` folder in the Sources window.

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

- If the upgrade is successful
- If user intervention is required: 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 an IP customization.

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 is 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 Multi-Level IP

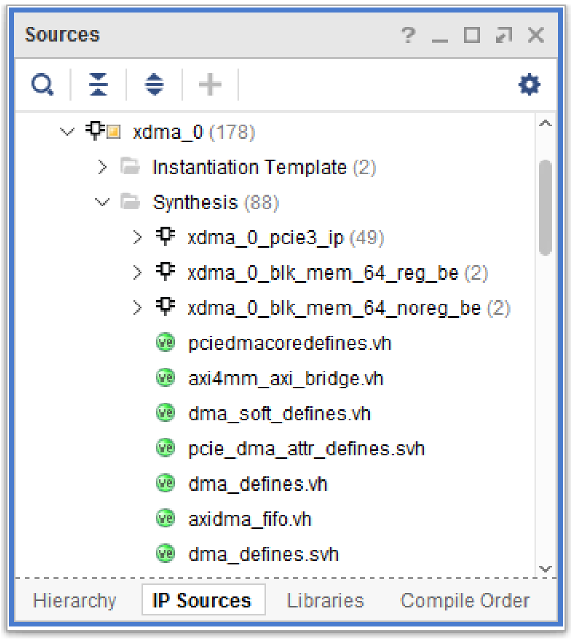
Some IP are designed to use other IP as design sources. Depending on how the parent IP was created, there can be OOC synthesis runs for the children IP. The following are types of subsystem IP with a parent-child relationship:

- IP that reference another IP as a library of files or IP that are beneath the top-level (*sub-core reference*): Sub-core references are described in *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118), have one OOC synthesis run. In the IP Sources there are no sub-core IP shown, there is only one XCI.
- IP that are packaged with XCI files for child IP (*static IP*): Static IP are those that were packaged with XCI files for other IP.
- IP that dynamically create child IP and HDL (*dynamic IP*): Dynamic IP have one OOC synthesis run because all the IP are synthesized together. Similar to the static IP, you see multiple XCI in the IP sources.

- IP that use the IP integrator technologies to dynamically create and interconnect IP (*subsystem IP*): Subsystem IP are the IP that the IP integrator technology creates. For example, 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.

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.

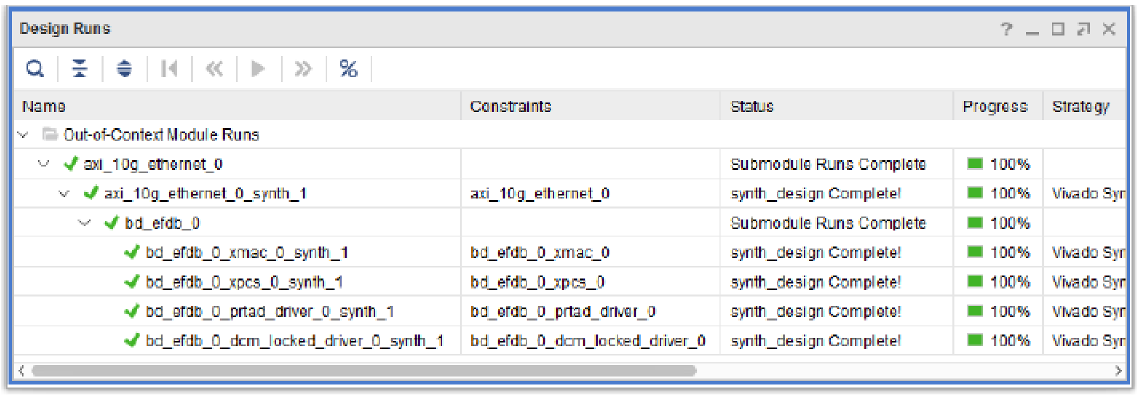
Figure 26: **Output Products for Subsystem IP Using XCI Files**



```
xdma_0 (178)
    Instantiation Template (2)
    Synthesis (88)
        xdma_0_pcie3_ip (49)
        xdma_0_blk_mem_64_reg_be (2)
        xdma_0_blk_mem_64_noreg_be (2)
        pciedmacoredefines.vh
        axi4mm_axi_bridge.vh
        dma_soft_defines.vh
        pcie_dma_attr_defines.svh
        dma_defines.vh
        axidma_fifo.vh
        dma_defines.svh
```

During OOC 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 subsystem IP look just like other IP.

Figure 27: **Output Products for Subsystem IP**



Name	Constraints	Status	Progress	Strategy
Out-of-Context Module Runs				
✓ axi_10g_ethernet_0		Submodule Runs Complete	🟩 100%	
✓ axi_10g_ethernet_0_synth_1	axi_10g_ethernet_0	synth_design Complete!	🟩 100%	Vivado Syn
✓ bd_efdb_0		Submodule Runs Complete	🟩 100%	
✓ bd_efdb_0_xmac_0_synth_1	bd_efdb_0_xmac_0	synth_design Complete!	🟩 100%	Vivado Syn
✓ bd_efdb_0_xpcs_0_synth_1	bd_efdb_0_xpcs_0	synth_design Complete!	🟩 100%	Vivado Syn
✓ bd_efdb_0_prtad_driver_0_synth_1	bd_efdb_0_prtad_driver_0	synth_design Complete!	🟩 100%	Vivado Syn
✓ bd_efdb_0_dcm_locked_driver_0_synth_1	bd_efdb_0_dcm_locked_driver_0	synth_design Complete!	🟩 100%	Vivado Syn

TIP: *The primary benefit of subsystem IP, based upon block design, is that when generating the output products the children IP OOC runs are launched in-parallel.*

The Generate Output Products dialog box lets you specify the number of parallel runs in the **Run Settings** field.

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

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 an AMD 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 that 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.

An example design can be generated for any customization of the IBERT core. After you have customized and generated a core instance, right-click the generated core and select **Open IP Example Design** feature for 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:

  - *IBERT 7 Series GTX Transceivers LogiCORE IP Product Guide* (PG132)
  - *IBERT 7 Series GTP Transceivers LogiCORE IP Product Guide* (PG133)
  - *IBERT 7 Series GTH Transceivers LogiCORE IP Product Guide* (PG152)
  - *Integrated Logic Analyzer LogiCORE IP Product Guide* (PG172)
  - *JTAG to AXI LogicCORE IP Product Guide* (PG174)
  - *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
  - *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
  - *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)

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 the HDL Instantiated Debug Probing Flow Overview section in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) for more information on this 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.
    - It is very 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 that 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, next use the following Tcl commands to add debug cores and probes to your synthesized design:
  - `create_debug_core`
  - `create_debug_port`
  - `connect_debug_port`

Using a Core Container

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

Note: Binary files are not preferred for revision control. It is recommended to bring the XCI outside of core container which allows you to diff IP in a revision control environment to see if anything has changed.

set_property coreContainer.alwaysCreateXCI 1 [current_project]

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.

Note: IS_MANAGED cannot be set to true in this flow for Upgrade IP. To upgrade IP, use the following command:

```
[get_ips {<ip_name>}]
```

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.

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

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, the Vivado tools read 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 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; and 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. The IP Sources tab shows all the files that are generated for the IP and you retain the ability to open them 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 IP User Files (ip_user_files) for Core Container for more details.

Enabling and Disabling the Core Container

The Core Container feature is disabled by default. To have newly-created IP use the Core Container feature, go to **Settings → IP**and check the **Use Core Containers for IPs**.

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

***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.

Where applicable, the methods to enable Core Container are, as follows:

- To use the Core Container format for all IP, select the **Settings → IP** and check the **Use Core Container** option.
- 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.

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.

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` Tcl command. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

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.

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, SystemVerilog, 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 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.

IP User Files (ip_user_files) for Core Container

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. Appendix B: IP Files and Directory Structure describes these files.

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

Depending on whether the IP was created in a 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 the `ip_user_files` directory there are a number of folders The folders that are present depend on what is in your project (IP, Block Designs, and so forth).

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.

The three directories present are:

- `hdl`: Simulation top-level file for the block design.
- `ip`: each IP in the BD has 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, 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. 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 you 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 IP User Files (ip_user_files) for Core Container.

Contents of the `mem_init_files` Directory

Some IP deliver data files. These files are marked with a `DATA` property. 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 Simulator
- Mentor Graphics Questa Advanced Simulator
- Riviera-PRO Simulator
- Active-HDL Simulator (Windows only)

For Linux, this includes these additional simulators:

- Synopsys Verilog Compiler Simulator (VCS)
- Cadence Incisive Enterprise Simulator (IES)

To control scripts generation, see the 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 IP User Files (ip_user_files) for Core Container directory. For IP in a block design, the scripts reference the simulation files in the IP User Files (ip_user_files) for Core Container.

Tcl Command to Export Support Files

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

Using Manage IP Projects

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

- 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.

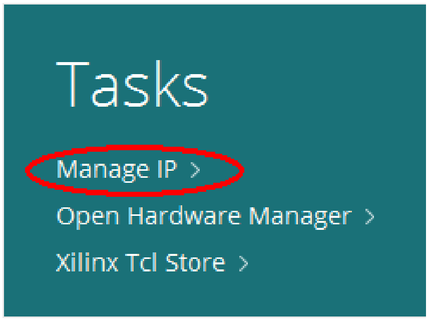
When working in teams, or if the design uses many AMD IP, create, and maintain your customized IP in a location outside of the Vivado project structure. This method makes revision control more straightforward and allows for ease of sharing customized IP with others. This is also the recommended methodology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

Using the Manage IP Flow

Note: This flow does not support subsystem IP.

To use the Manage IP flow, invoke the Vivado IDE, and from the **Getting Started** page, select **Manage IP**.

Figure 28: **Invoking the Manage IP Flow**

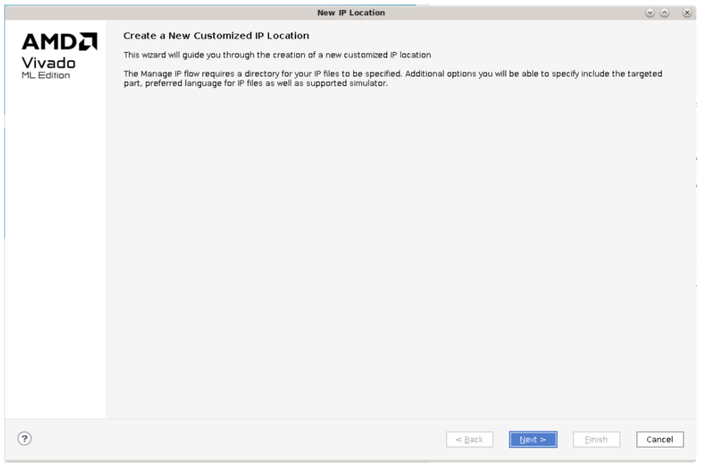


Open a new or an existing IP location.

- **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 Projects**: 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 guides you through creating and managing a new customized IP location, as shown in the following figure.

Figure 29: **Create a New Customized IP Location**



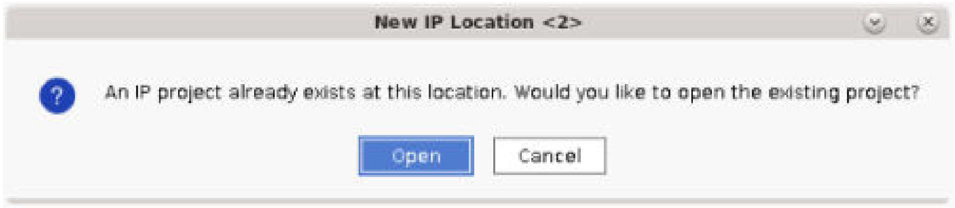
Create a New Customized IP Location

This wizard will guide you through the creation of a new customized IP location

The Manage IP flow requires a directory for your IP files to be specified. Additional options you will be able to specify include the targeted part, preferred language for IP files as well as supported simulator.

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

Figure 30: **New IP Location Dialog Box**

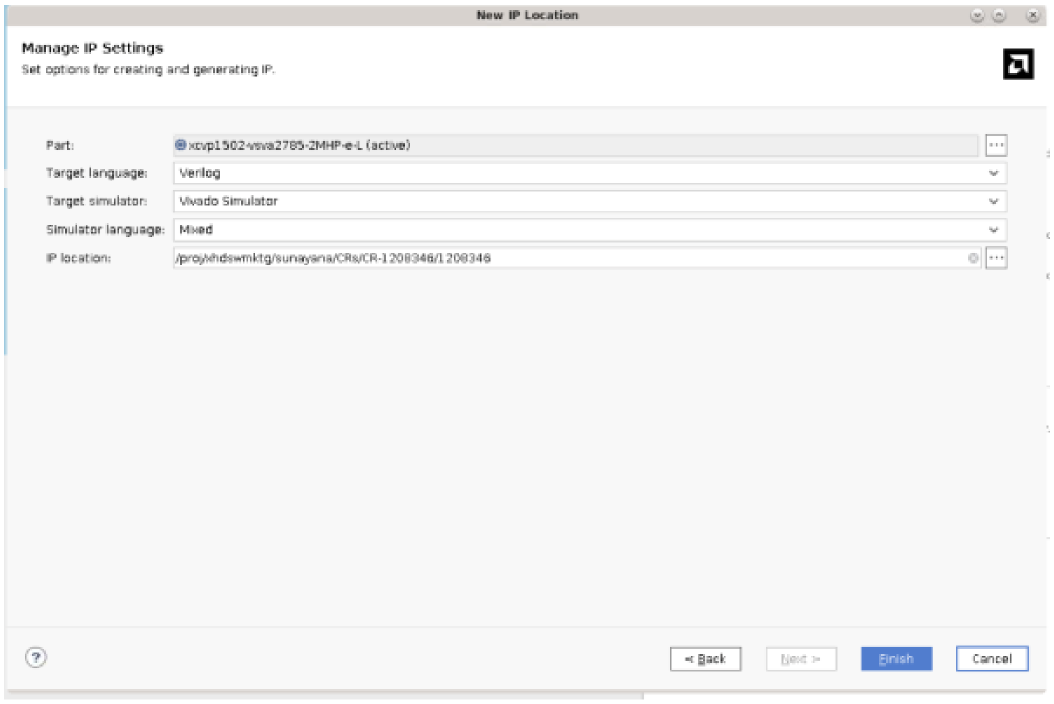


An IP project already exists at this location. Would you like to open the existing project?

You can either choose to open the existing project or cancel, select **Open**.

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

Figure 31: **Manage IP Settings Dialog Box**



Manage IP Settings
Set options for creating and generating IP.

Part:	xcvp1502-vsva2785-2MHP-e-L (active)
Target language:	Verilog
Target simulator:	Vivado Simulator
Simulator language:	Mixed
IP location:	/proj/xhdswmktg/sunayana/CRs/CR-1208346/1208346

Managing IP Settings

To manage the IP settings, enter the following information, and click **Finish**:

- **Part**: Select the active part. All output products generated for the IP are based on the specified part.
- **Target language**: Set the target language to the language of the top-level module of your design.
- **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.

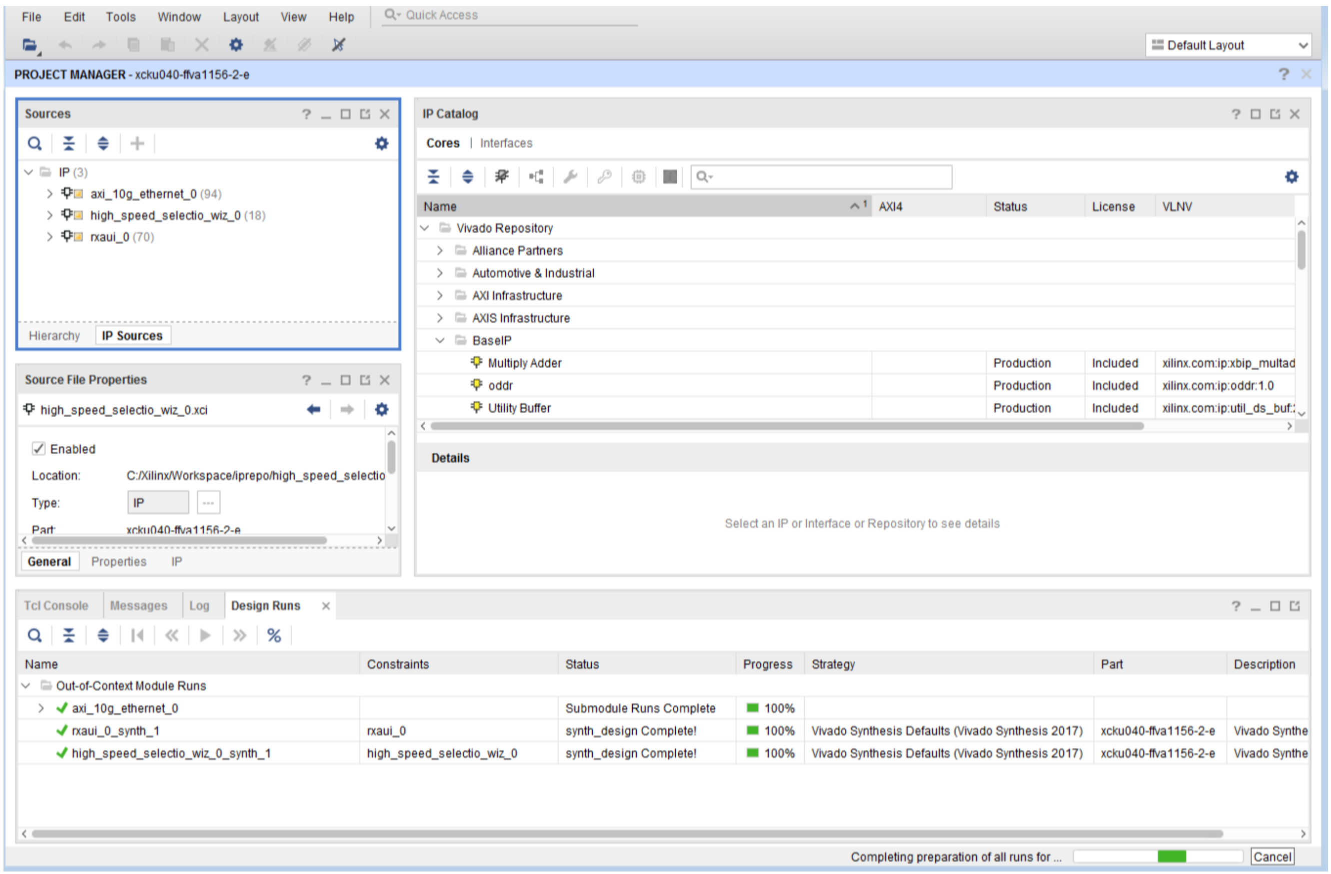
***Note*:** IP location might be referred to as an IP Repository.

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 Xilinx custom interface (XCI) file and any generated output products. The following figure shows the Manage IP Project window where you customize and manage multiple IP.

Figure 32: **Manage IP Project Window Containing Three IP**



PROJECT MANAGER - xcku040-ffva1156-2-e

Sources:

- IP (3)
  - axi_10g_ethernet_0 (94)
  - high_speed_selectio_wiz_0 (18)
  - rxaui_0 (70)

Source File Properties: high_speed_selectio_wiz_0.xci

- ✓ Enabled
- Location: C:/Xilinx/Workspace/iprepo/high_speed_selectio
- Type: IP
- Part: xcku040-ffva1156-2-e

IP Catalog:

Name	AXI4	Status	License	VLNV
Vivado Repository				
Alliance Partners				
Automotive & Industrial				
AXI Infrastructure				
AXIS Infrastructure				
BaseIP				
Multiply Adder		Production	Included	xilinx.com:ip:xbip_multad
oddr		Production	Included	xilinx.com:ip:oddr:1.0
Utility Buffer		Production	Included	xilinx.com:ip:util_ds_buf:

Select an IP or Interface or Repository to see details

Design Runs:

Name	Constraints	Status	Progress	Strategy	Part	Description
Out-of-Context Module Runs						
✓ axi_10g_ethernet_0		Submodule Runs Complete	🟩 100%			
✓ rxaui_0_synth_1	rxaui_0	synth_design Complete!	🟩 100%	Vivado Synthesis Defaults (Vivado Synthesis 2017)	xcku040-ffva1156-2-e	Vivado Synthe
✓ high_speed_selectio_wiz_0_synth_1	high_speed_selectio_wiz_0	synth_design Complete!	🟩 100%	Vivado Synthesis Defaults (Vivado Synthesis 2017)	xcku040-ffva1156-2-e	Vivado Synthe

Completing preparation of all runs for ...

CAUTION! *When creating a new AXI4 peripheral, verification through the AXI4 VIP and JTAG interface are not available. To use this peripheral verification, you must create a Vivado project with that peripheral.*

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 AMD 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.

***Note*:** 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.

IMPORTANT! *AXI Peripheral IP is not suited for use in a Managed IP Project. The Vivado Integrated Design Environment (IDE) issues an error when you attempt to use such IP. If you need AXI Peripheral IP, use a regular Vivado project.*

Chapter 4

Using IP Example Designs

Introduction

Many AMD 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, right-click and select **Open IP Example Design** from the context menu.

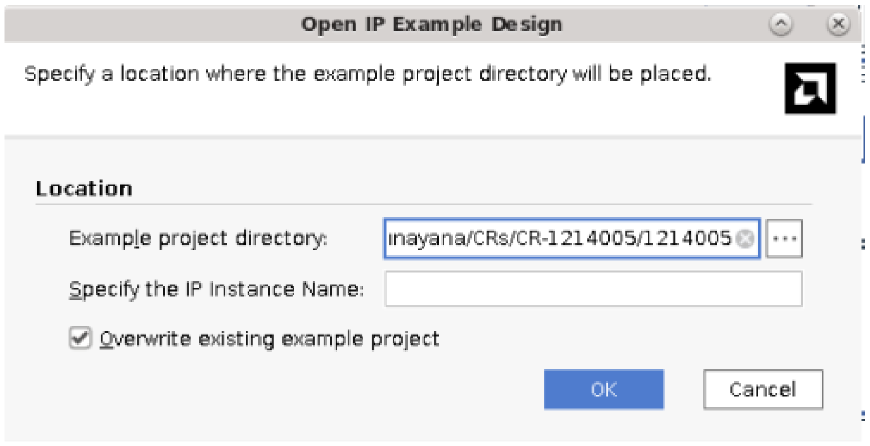
The **Open IP Example Project** dialog box opens for you to specify the location, as shown in the following figure. The project is called <ip_name_ex>.

IMPORTANT! *Do not store example designs in the IP directory in either a standard project or a Manage IP project. AMD 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.*

In an IP integrator block design, either select the IP in IP Sources or access the IP directly from a block design.

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

Figure 33: **Open IP Example Design Dialog Box**


Open IP Example Design

Specify a location where the example project directory will be placed.

Location

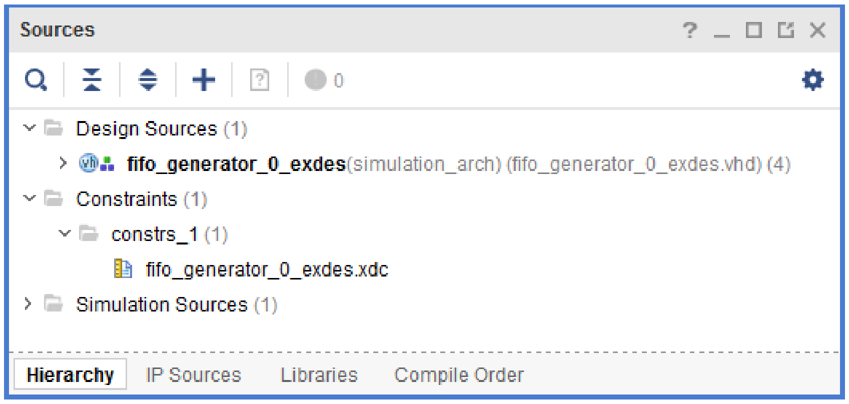
Example project directory: inayana/CRs/CR-1214005/1214005

Specify the IP Instance Name:

✓ Overwrite existing example project

The New Project Summary dialog box provides a review of your selections, and a new session of the AMD Vivado™ IDE opens with the example design, shown in the following figure.

Figure 34: **IP Example Design Instance with Constraint File**



```
Design Sources (1)
    fifo_generator_0_exdes(simulation_arch) (fifo_generator_0_exdes.vhd) (4)
Constraints (1)
    constrs_1 (1)
        fifo_generator_0_exdes.xdc
Simulation Sources (1)
```

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

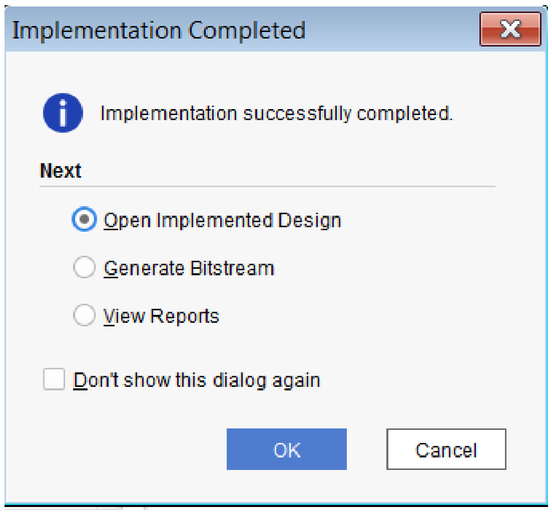
Alternatively, you can use the `open_example_project` Tcl command to open a project:

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

Examining Standalone IP

When using an IP that has already been synthesized, after implementation completes, the Implementation Completed dialog box opens to give you the option to open the implemented design, generate a bitstream, or view reports, as shown in the following figure.

Figure 35: **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 includes 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.*

Chapter 5

Using AMD IP with Third-Party Synthesis Tools

Third-Party Synthesis Flow

When using a Synopsys Synplify Pro or Mentor Graphics Precision netlist for synthesis of a design that has AMD IP, the recommended flow is to use the **Manage IP** flow to create and customize IP (including AMD XPMs), and 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`.

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.

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

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

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, and implement the design with the following steps:

1. Create an HDL wrapper around the EDIF netlist produced by the third-party synthesis tool.
2. Select the hierarchy tab of the sources window.
3. Right-click and select **Hierarchy Update**, and check the **No Update, Manual Compile Order** option.

4. Add the following into the AMD 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 and the XDC output products are used automatically during implementation when using the XCI file for the IP.
5. Implement the design.

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

   Use the IP XCI file when referencing AMD 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 evaluated in context of the end-user constraints and netlist.

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 generate the IP using the `synth_ip` command.

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]
set_property top <top> [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
```

Introduction

AMD supports netlists created by third-party synthesis tools for user logic. When using AMD IP the only supported synthesis tool is the Vivado synthesis tool. The Vivado Design Suite infers a black box during logic synthesis for AMD 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 AMD 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! *AMD 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 Vivado 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.*

Chapter 6

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)
- *Vivado Design Suite Tcl Command Reference Guide* (UG835)

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

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

IMPORTANT! *When using HLS IP in non-project mode, be sure to run the `compile_c` command prior to running synthesis.*

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 5: **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>`
		Creates project in memory, not on disk. Commands that have a part option, for example, `synth_design`, and use the specified part.

Create an IP Customization	`create_ip <ip_name>`	`create_ip <ip_name>`
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 in Non Project mode.	`set_property \`
		`CONFIG.<clock_name>.FREQ_HZ \`
		`<#>[get_ips char_fifo]`
		See Setting the Target Clock Period.

Generate output products	`generate_target all \`
	`[get_ips <ip_name>]`
	Optionally, you can specify the target(s) you want to generate.
		`generate_target all \`
		`[get_ips <ip_name>]`
		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 and any generated output products. Use either of the following:
		`add_files <ip_name>.xci`
		`read_ip <ip_name>.xci`
		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 .	N/A
IP Definition	N/A	`report_property -all [get_ipdefs <IPVLNV>`

Table 5: **IP Tcl Commands in Order of Design Use** *(cont'd)*

Action	Project Mode Command	Non-Project Mode Command
IP Creation	\ Writes out IP from a Tcl script.	`write_ip_tcl <ip_name>` Writes out IP from a Tcl script.

Tcl Commands for Common IP Operations

Within the AMD 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 amd.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 following are useful Tcl commands for IP.

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]
```

Return an alphabetized list of the customization parameters:

```
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
```

Export a simulation:

```
export_simulation
```

Extract files from a core container to disk:

```
extract_files
```

When you remove a design run for an IP and reset the output products for an IP in the Vivado IDE, two Tcl commands are issued: `reset_target` and `delete_ip_run`.

Write constraints to an XDC file:

```
write_xdc <file>
```

The command writes the constraints into the file in the same order in which the constraints are executed, based on the constraint settings. See Managing IP Constraints for more information on setting IP constraints. Also, see Overriding IP Constraints. Go to the *Vivado Design Suite User Guide: Using Constraints* (UG903) for more information.

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

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

Create a Tcl script of either all RTL IP or individual RTL IP with customizations:

```
write_ip_tcl
```

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

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

Perform this operation as described in Using the Manage IP Flow. 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 amd.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! *AMD 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 the following table for a more detailed explanation on these Tcl commands and when to use them.

***Note*:** Run the `generate_target all [get_ips]` and `compile_c [get_ips]commands` before `synth_ip` for HLS IPs.

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 <project_location> -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

When you synthesize and implement a design in a non-project flow, you could have 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 that are present, including an OOC DCP, are used, and there is no need to generate these files.

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) you must disable checkpoint support and generate the output products. The following Tcl script provides a template for this action:

```
#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 <project_location> -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 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

AMD Vivado™ IP cores become locked for several reasons. The Vivado integrated design environment (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 for information on the IP Status Report.

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

Table 6: **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. AMD 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)). Import it back into the project with a unique directory using the `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.

Table 6: **IP Locked Reasons and Recommendations** *(cont'd)*

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
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.
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. See 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.	Re-target 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.

Table 6: **IP Locked Reasons and Recommendations** *(cont'd)*

Brief Reason	Verbose Reason	Brief Description	Verbose Recommendation
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.
Other	IP Def 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 your 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.
	USER_LOCKED property	The IP is locked by the user. In this mode, it is the responsibility of the user to manage all IP files.	Remove the USER_LOCKED property on the IP. See Editing IP Sources for more information.
	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 6: **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 Appendix D: Editing or Overriding IP Sources for more information.

Notes:

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 AMD Vivado™ Integrated Design Environment (IDE) creates a unique directory for each IP depending on the flow. For an IP created directly within Vivado RTL projects, IP sources are placed in <project_name>.srcs directory and IP output products are placed in <project_name>.gen directory. For Managed IP flow, all IP source or output product files are created in a directory parallel to the location of the top-level Managed IP project.

After creating an IP customization, IP-generated files include the Xilinx core instance (XCI) file, instantiation template, BOM file, and any generated output products. In the IP directory or the <project_name>.gen 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.

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

Table 7: **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.

Table 7: **IP Output Products** *(cont'd)*

Directory Name, File Name, or File Type	Description
`/synth`	Contains synthesizeable source files for IP. This directory is not present for IP that does not support synthesis, such as simulation-only Verification IP.
`<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. AMD 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 Setting the Target Clock Period 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 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.
- The 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 8: **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 initialization 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, you 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 9: **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 10: **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

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.

Appendix C

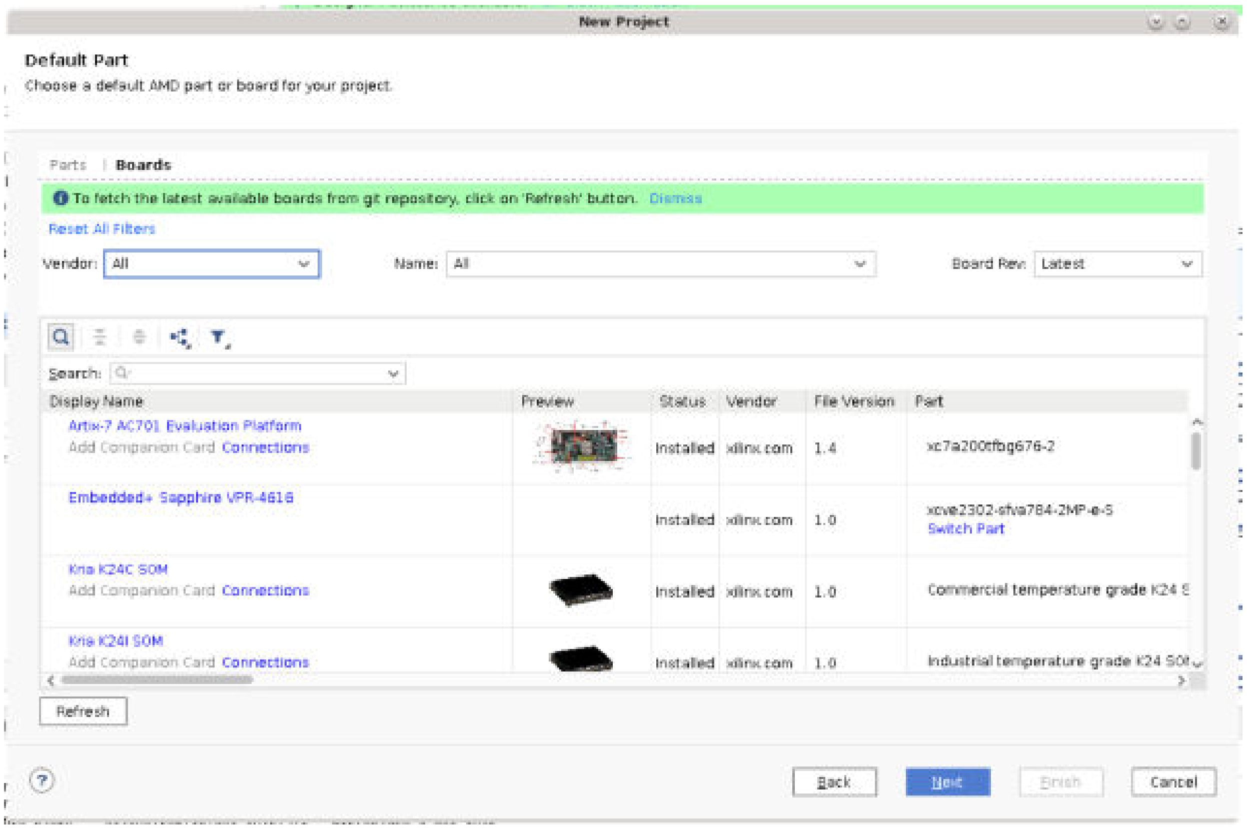
Using the Platform Board Flow for IP

Introduction

The AMD 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 36: **Selecting a Board as the Default Part**



New Project

Default Part
Choose a default AMD part or board for your project.

Parts | **Boards**

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Reset All Filters

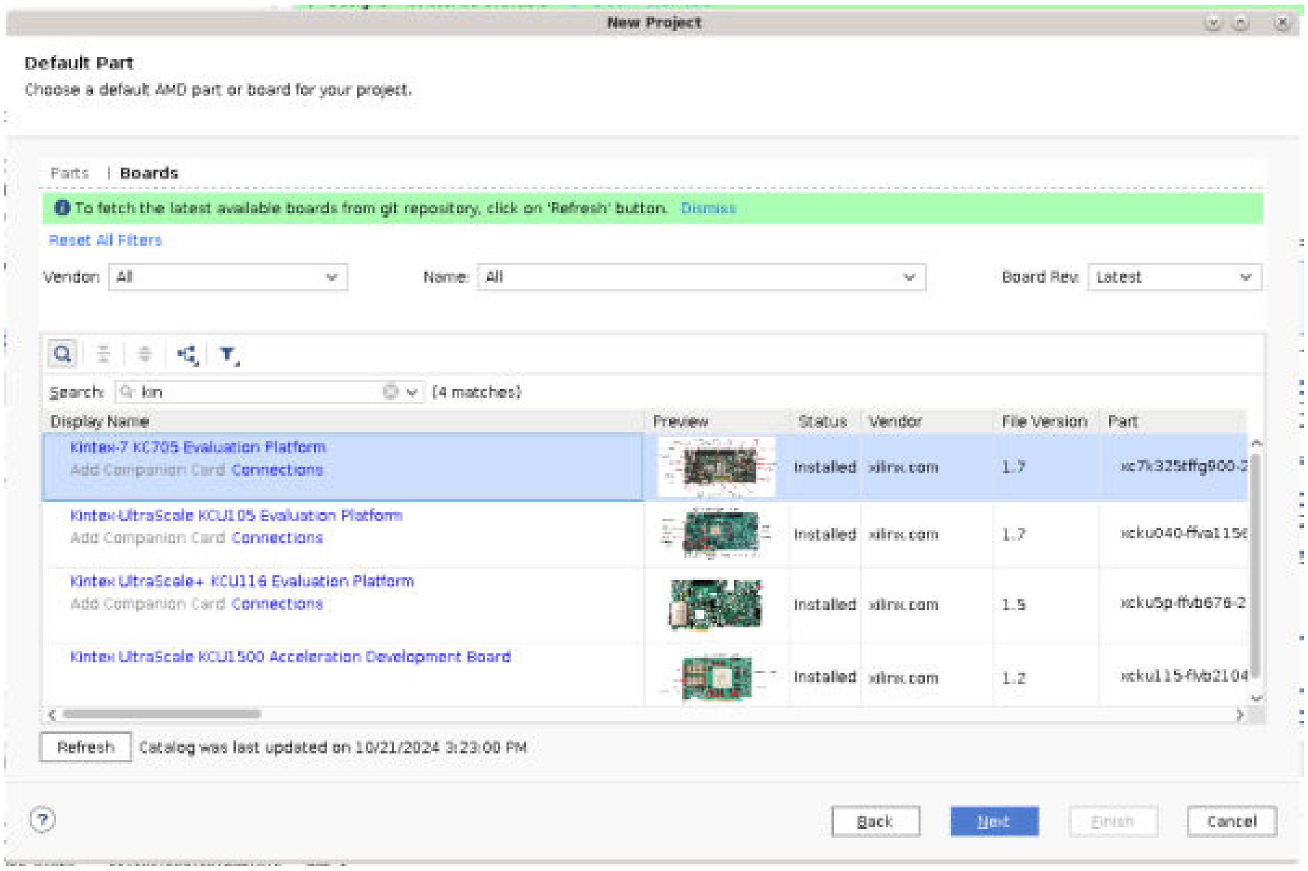
Vendor: All Name: All Board Rev: Latest

Search:

Display Name	Preview	Status	Vendor	File Version	Part
Artix-7 AC701 Evaluation Platform Add Companion Card Connections		Installed	xilinx.com	1.4	xc7a200tfbg676-2
Embedded+ Sapphire VPR-4616		Installed	xilinx.com	1.0	xcve2302-sfva784-2MP-e-S Switch Part
Kria K24C SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Commercial temperature grade K24 S
Kria K24I SOM Add Companion Card Connections		Installed	xilinx.com	1.0	Industrial temperature grade K24 SOI

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, as shown in the following figure.

Figure 37: **Board Summary Information**



Default Part

Choose a default AMD part or board for your project.

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Vendor: All   Name: All   Board Rev: Latest

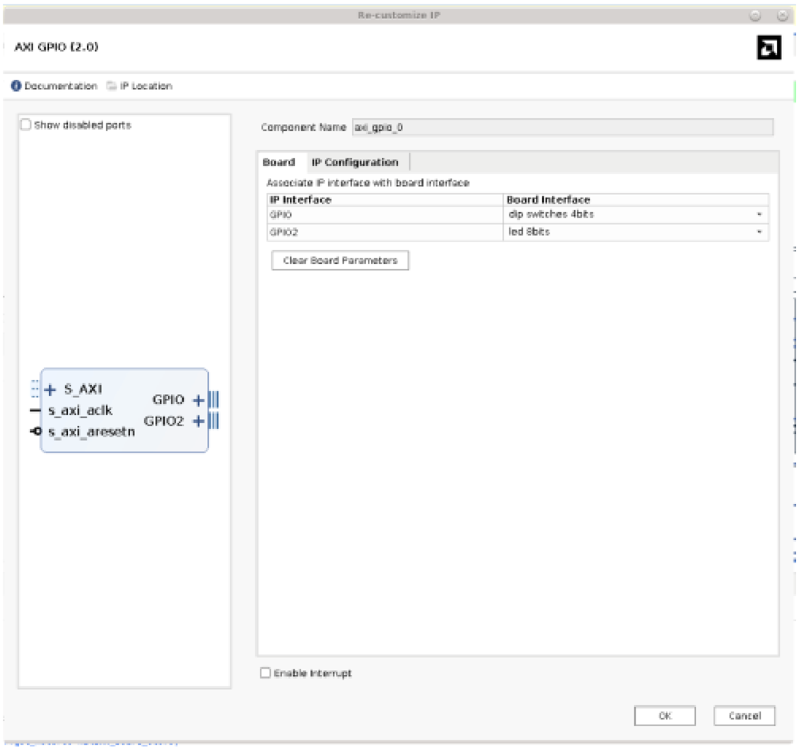
Search: kin (4 matches)

Display Name	Preview	Status	Vendor	File Version	Part
Kintex-7 KC705 Evaluation Platform Add Companion Card Connections		Installed	xilinx.com	1.7	xc7k325tffg900-2
Kintex-UltraScale KCU105 Evaluation Platform Add Companion Card Connections		Installed	xilinx.com	1.7	xcku040-ffva156
Kintex UltraScale+ KCU116 Evaluation Platform Add Companion Card Connections		Installed	xilinx.com	1.5	xcku5p-ffvb676-2
Kintex UltraScale KCU1500 Acceleration Development Board		Installed	xilinx.com	1.2	xcku115-flvb2104

Catalog was last updated on 10/21/2024 3:23:00 PM

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 38: **Board Type Visible in Supported IP Customization**



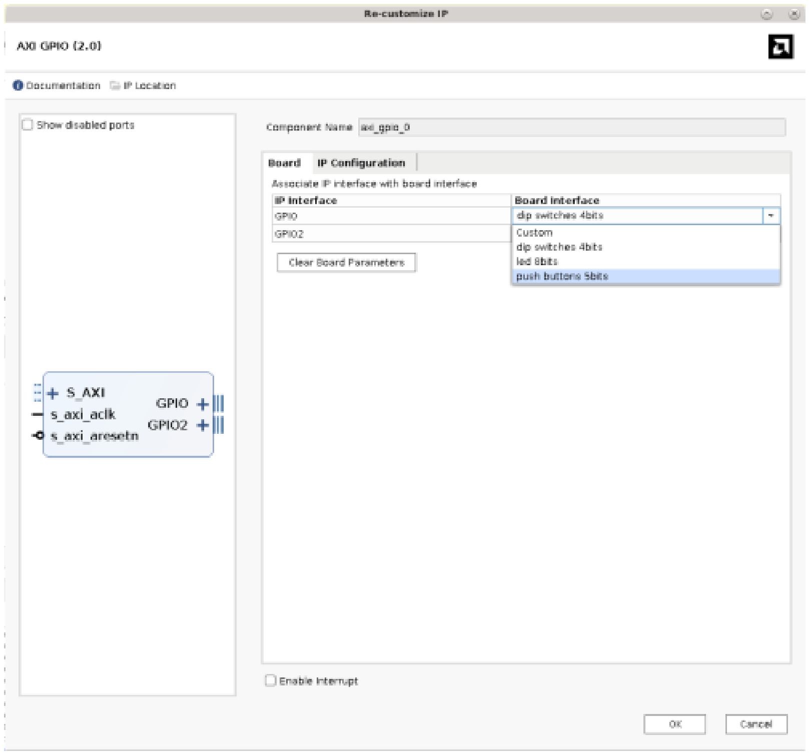
Component Name: axi_gpio_0

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	dip switches 4bits
GPIO2	led 8bits

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 39: **Associating the IP Interface with the Board Interface**



Re-customize IP

AXI GPIO (2.0)

Documentation   IP Location

Show disabled ports

Component Name axi_gpio_0

Board   **IP Configuration**

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	dip switches 4bits
GPIO2	

Custom
dip switches 4bits
led 8bits
push buttons 5bits

Clear Board Parameters

Enable Interrupt

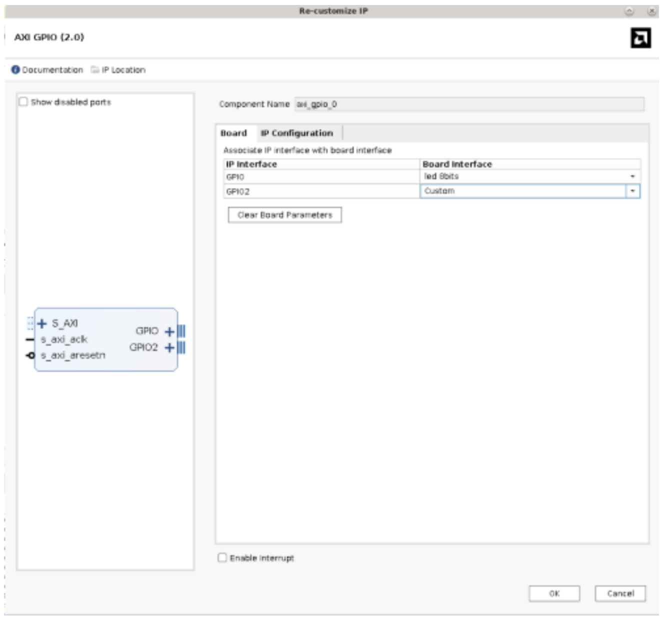
OK   Cancel

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 **led8bits** interface and the connect the **GPIO2** interface to the board **Custom** interface.

Figure 40: **Board Interface Selected**



Component Name axi_gpio_0

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	led 8bits
GPIO2	Custom

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.

- See the Using the Platform Board Flow in IP Integrator section in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) for more information on using the platform board flow.
- See the Board File Linter section in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) for information on the Board Interface file and creating your own board files.

Appendix D

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, you can either override the XDC in a top-level XDC or Tcl file, or edit the IP source.

***Note*:** These options work for most IP in an RTL project. You cannot modify IP that contain Hierarchical IP.

IMPORTANT! *Be sure that the Core Container feature is disabled before editing IP sources.*

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 regarding constraints.*

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

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 and 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 the XDC Precedence section in the *Vivado Design Suite User Guide: Using Constraints* (UG903), 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`. Do this with the `reset_property` command, which is not an XDC command, and must be placed in a Tcl file. After resetting the `LOC` property, 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).

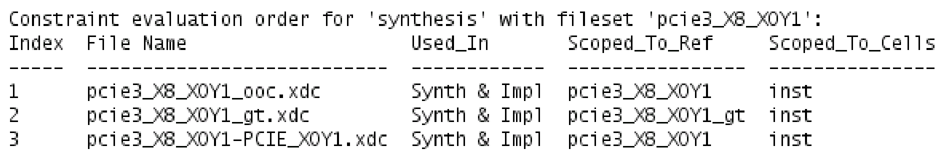
When overriding IP constraints at the top-level you have two choices:

- 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

- 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.

Figure 41: **Synthesis 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
```

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

AMD 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, as follows:

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.

   AMD 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.
   ```

   CAUTION! *Once you set IS_MANAGED property to false, the IP is user-managed. You cannot switch the property back to true as thereâs a reasonably high risk of the userâs edited sources being overwritten.*

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. Re-create the IP output products, including the DCP, as follows:

   a. Reset the IP OOC run. This has to be performed using the Tcl Console. Look at the Design Runs tab in the Out-of-Context Module Runs folder, and find the IP where you set the `IS_MANAGED` property to FALSE, with the name `<IP_Name>_synth_1`. Execute the following command in the Tcl Console to reset the run:

      ```
      reset_run <ip_name>_synth_1
      ```

   b. Re-launch the run using the following command:

      ```
      launch_run <ip_name>_synth_1
      ```

      This uses any of the HDL or constraints of the IP that you modified.

      After the run completes, you can use the IP as before.

      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 and 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 OOC per IP for the synthesis option. You need to have an existing design run for the IP present.
2. Set the IS_MANAGED property to false, which turns the IS_LOCKED property to true. For complex subsystem IP that do not allow changes to the IS_MANAGED property, set the IS_LOCKED property to true.
3. This puts the IP under user management.

   CAUTION! *Once you set IS_MANAGED property to false, the IP is user-managed. You cannot switch the property back to true as there is a reasonably high risk of the user's edited sources being overwritten.*

4. 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.
5. Recreate the IP output products, including the DCP, as follows:
   a. Reset the IP OOC run using the Tcl Console.

   b. Look at the Design Runs tab in the Out-of-Context Module Runs folder, and find the IP you want to re-synthesize; it as called `<IP_Name>_synth_1`.

   c. Execute the following in the Tcl Console to reset the run:

   ```
   reset_run <ip_name>_synth_1
   ```

   d. Re-launch the run using the following command in the Tcl Console:

   ```
   launch_run <ip_name>_synth_1
   ```

   This uses any of the HDL or constraints of the IP that you modified.

After the run completes, you can use the IP as before.

***Note*:** Because subsystem IP do not allow the changing of the `IS_MANAGED` property there is not any visual indication to show that you have made changes. It is up to the user to keep track of modified IP.

Appendix E

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD 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, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

AMD 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:

1. *Zynq 7000 SoC Verification IP Data Sheet* (DS940)
2. *Zynq 7000 SoC Verification IP Data Sheet* (DS941)
3. *IBERT 7 Series GTX Transceivers LogiCORE IP Product Guide* (PG132)
4. *IBERT 7 Series GTP Transceivers LogiCORE IP Product Guide* (PG133)
5. *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)
6. *IBERT 7 Series GTH Transceivers LogiCORE IP Product Guide* (PG152)
7. *Virtual Input/Output LogiCORE IP Product Guide* (PG159)
8. *Integrated Logic Analyzer LogiCORE IP Product Guide* (PG172)
9. *JTAG to AXI LogicCORE IP Product Guide* (PG174)
10. *AXI Verification LogiCORE IP Product Guide* (PG267)
11. *AXI4-Stream Verification IP LogiCORE IP Product Guide* (PG277)
12. *Zynq 7000 SoC and 7 Series Devices Memory Interface Solutions User Guide* (UG586)
13. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
14. *Vivado Design Suite Tutorial: Design Flows Overview* (UG888)
15. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
16. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
17. *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894)

18. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)
19. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)
20. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
21. *Vivado Design Suite User Guide: Using Constraints* (UG903)
22. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
23. *Vivado Design Suite User Guide: Getting Started* (UG910)
24. *ISE to Vivado Design Suite Migration Guide* (UG911)
25. *Vivado Design Suite Properties Reference Guide* (UG912)
26. *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
27. *Vivado Design Suite Tutorial: Logic Simulation* (UG937)
28. *Vivado Design Suite Tutorial: Designing with IP* (UG939)
29. *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)
30. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
31. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
32. *Vivado Design Suite: AXI Reference Guide* (UG1037)
33. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
34. *Vivado Design Suite Tutorial: Creating, Packaging Custom IP* (UG1119)
35. Vivado Design Suite Documentation
36. Vivado IP Versioning
37. IP Documentation
38. IP Center

Standards and Third-Party Documentation

1. IP-XACT Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows (IEEE Std 1685)
2. IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP) (IEEE Std P1735)

Training Resources

AMD 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: Creating an AXI Peripheral in Vivado
4. Vivado Design Suite QuickTake Video: Configuring and Managing Reusable IP in Vivado
5. Vivado Design Suite QuickTake Video: Design Constraints Overview
6. Vivado Design Suite QuickTake Video: Global Timing Constraints
7. Vivado Design Suite QuickTake Video: Managing Sources with Projects
8. Vivado Design Suite QuickTake Video: Migrating UCF Constraints to XDC
9. Vivado Design Suite QuickTake Video: Tcl Scripts and Constraint Files in Vivado
10. Vivado Design Suite QuickTake Video: Using Core Containers with IP
11. Vivado Design Suite QuickTake Video: Using IP with 3rd-Party Synthesis Tools
12. Vivado Design Suite QuickTake Video: Vivado Activation and Floating License Generation
13. Vivado Design Suite QuickTake Video: Working with Constraint Sets
14. Vivado Design Suite QuickTake Video: Working with Design Checkpoints
15. Vivado Design Suite QuickTake Video: Designing with UltraScale Memory IP
16. Vivado Design Suite QuickTake Video: Getting Started with the Vivado IDE
17. Vivado Design Suite QuickTake Video: How to Use the Zynq 7000 Verification IP to Verify and Debug using Simulation
18. Vivado Design Suite QuickTake Video Tutorials

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/17/2025 Version 2025.2	
General updates	Editorial updates.
06/11/2025 Version 2025.1	
General updates	Editorial updates.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright