**Vivado Design Suite User**

**Guide**

***Designing with IP***

**UG896 (v2024.1) June 20, 2024**

*Chapter 1* IP-Centric Design Flow

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 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.

**1.1、Navigating Content by Design Process**

Hardware, IP, and Platform Development

System Integration and Validation

**1.2、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.

**1.3、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，已下载), Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.

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.

**1.4、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).

**1.5、Using Revision and Source Control**

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).

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

*Chapter 2 IP Basics*

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.

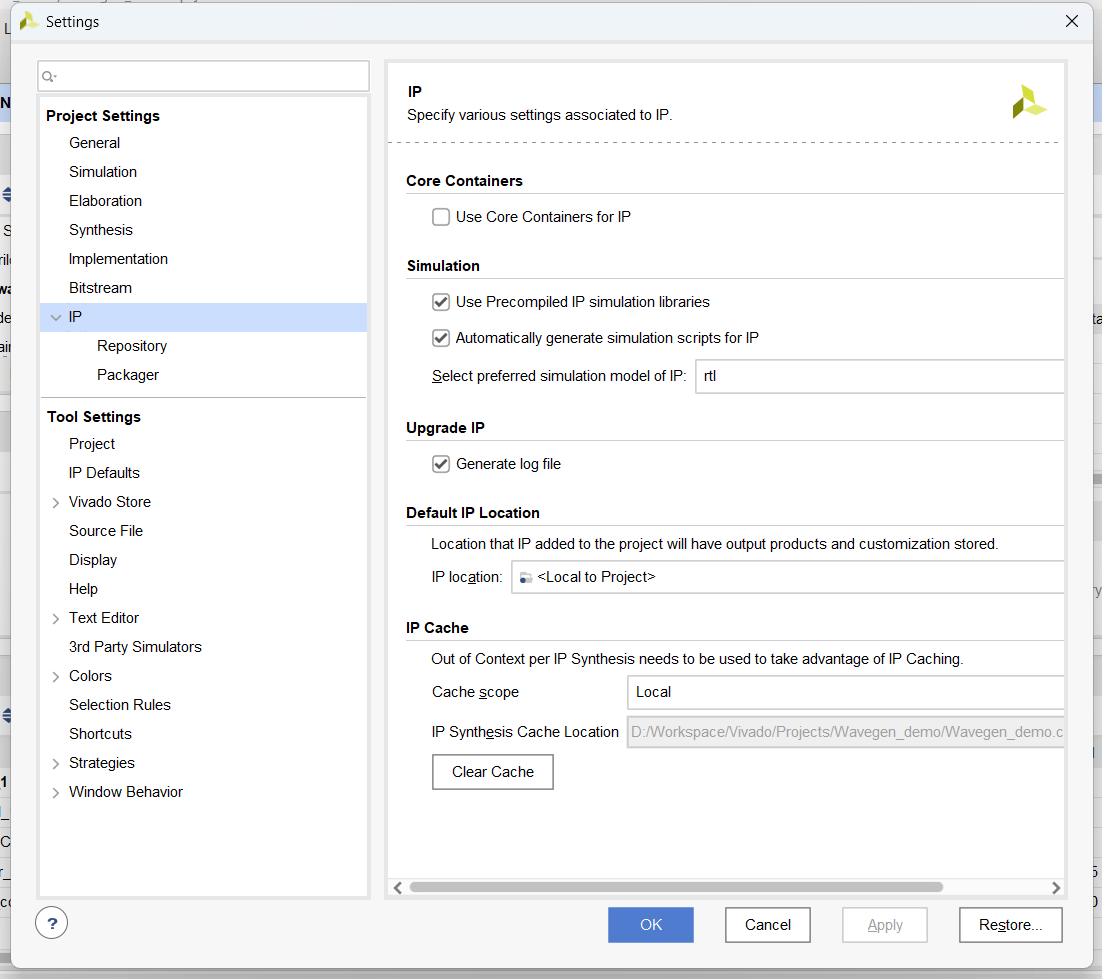
• In Project mode, if output products are not present, the Vivado tools generate the required output products automatically prior to synthesis or simulation. By default, output products are generated out-of-context (OOC) for synthesis.

• 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 toplevel 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.

**2.1、Using IP Project Settings**

When working with IP in a Manage IP project(独立的IP管理项目) or in an RTL project, you can configure IP-specific settings using the IP category in the Settings dialog box.



• 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 UG1118)

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.

**2.1.1、IP Settings**

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.

• Simulation: By default, the two check boxes 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.

• Upgrade IP: By default, the Generate log file is checked. This creates an ip\_upgrade.log file when you upgrade IP.

• 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<project\_name>.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.

○ 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.

○ 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

**2.1.2、Setting the IP Cache**

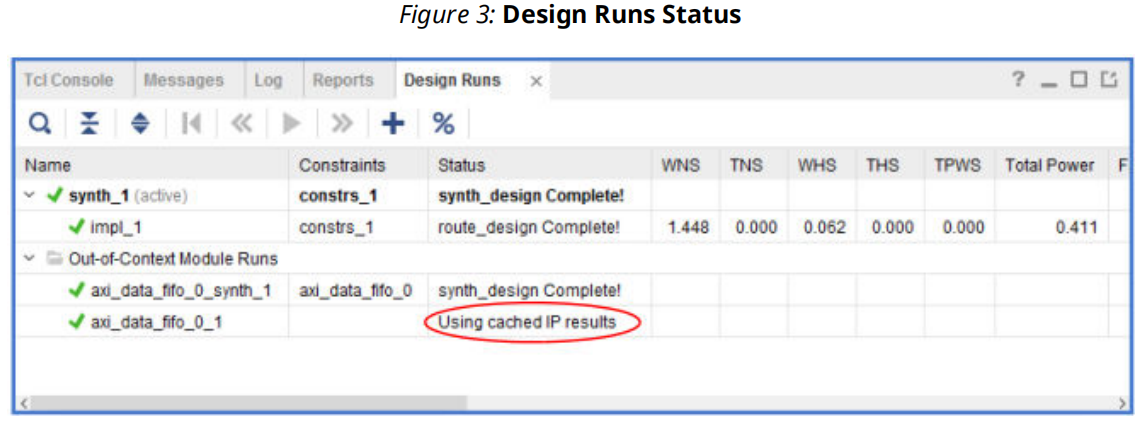
To speed up generation of the synthesis output products for an IP using the default OOC flow in Project Mode, the IP Cache option is enabled by default.

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).



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 theVivado 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.

**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/

**2.1.3、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.

**2.1.4、Managing IP Repositories**

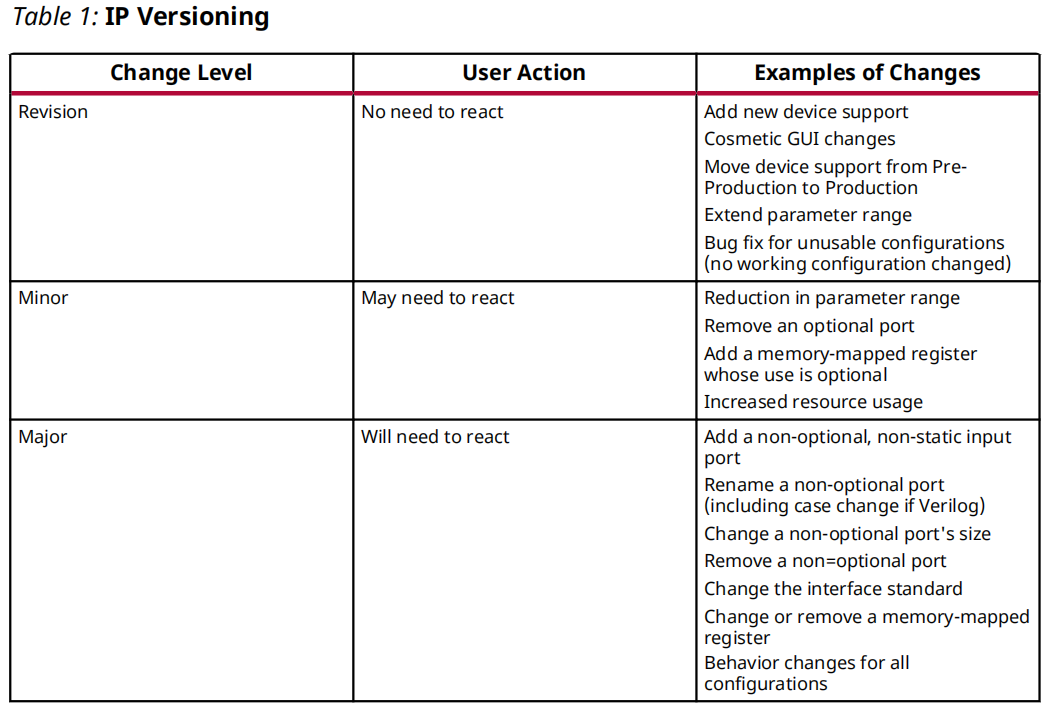
The Repository Manager hierarchically searches within the specified path for IP definitions. The IP is listed after the directory is specified. When you add an IP repository, a dialog box displays with a list of IP that are in the repository. The IP that contain a gray icon are disabled, and those with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

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.

**2.1.5、Using the Packager Settings**

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

**2.2、Using the IP Catalog**



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:

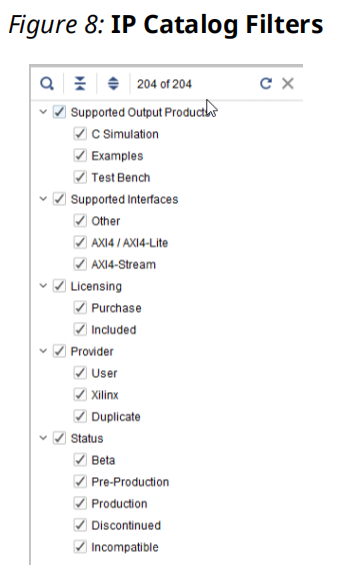
report\_property -all [get\_ipdefs \ amd.com:ip:fifo\_generator:13.1]

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

get\_property supported\_families [get\_ipdefs \ amd.com:ip:fifo\_generator:13.1]

**2.2.1、Filtering IP**

Select the Settings button to open the filter options.



**2.2.2、IP Catalog Options**

Group by Repository lets you group IP according to AMD IP or user-IP. Group by Taxonomy（分类学） lets you group IP by function or category.

**2.2.3、Partner Alliance IP**

**2.2.4、Using Fee-Based Licensed IP**

• License Status: IP licenses can be Full (also know as Purchased), Simulation, or Eval.

• 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.

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

**2.3、Creating an IP Customization**

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.

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

**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\_sr\ [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

**2.3.1、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.

**2.3.2、Using Tcl Commands to Reset and Generate Target IP**

• 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]

**2.3.3、Examining Generated Output Products**

By default, the output products for an IP are written to the local project directory, at <project\_name>.gen/sources\_1/ip/<ip\_name>. The Vivado tools store IP (.XCI) in an RTL project within the project directory structure in the <project\_name>.srcs/sources\_1/ip/<ip\_name>directory.

After generating output products needed by synthesis, 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.

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 except the instantiation templates.

**2.3.4、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**.

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.

AMD recommends that you use the default OOC to reduce the run time on synthesizing your design.

**2.3.5、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.

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.

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>]

Adding Existing IP using Tcl Commands

• 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>

**2.3.6、Resolving Duplicate IP**

**2.3.7、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).

**2.3.8、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, rightclick the IP, and select **Re-customize IP** from the context menu.

**2.3.9、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**.

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.

Tcl Command Example for Copying IP

copy\_ip -name newFIFo -dir newdir [get\_ips char\_fifo]

**2.4、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.

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.

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.

**2.4.1、Reporting IP Status**

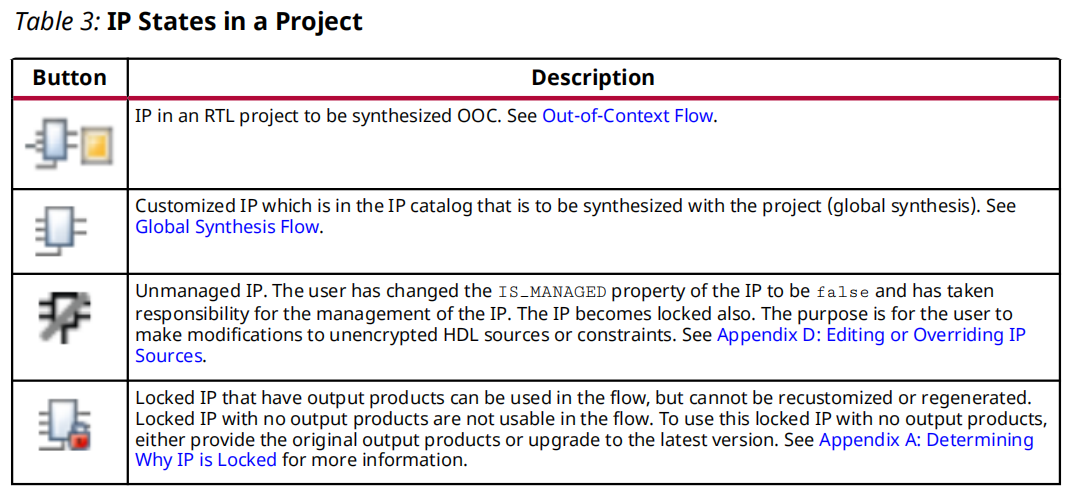
You can view the report of all IP in a project using the **Reports → Report IP Status**.

You can also generate this information using the report\_ip\_status Tcl command.

**2.5、Understanding IP States Within a Project**

**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 by Vivado IP Catalog form or XCO by Vivado Core Generator form), if present, the output products (such as HDL files) are also added.



**2.6、Managing IP Constraints**

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. 如果不使用默认综合设置产生的XDC文件，则应该禁止生成这些文件。

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]

不建议修改IP默认的综合设置。

**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.

**2.6.1、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).

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 report\_compile\_order -constrains Tcl command to report constraint compile order.

Typically, an IP delivers a **core XDC** file that can contain clock creation commands and be without external clock dependencies. The constraint file name is <ip\_name>.xdc(无关联外部时钟核心的约束文件), and is referred to as the core XDC file.

IP can also include another XDC file that contains clock-dependent commands. Any constraints that need dependent clocks to be defined first should be placed in the <ip\_name>\_clocks.xdc（关联外部时钟的约束文件）. 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 toplevel 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 enduser 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.

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).

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.

**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.

**2.7、Setting the Target Clock Period**

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.

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.

**2.7.1、Setting a Target Clock Period using Tcl Commands**

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.

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.

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 250MHZ, set as follows:

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

set\_property CONFIG.core\_clk.FREQ\_HZ 250000000 [get\_ips\ char\_fifo]

4） After you set the property, generate the IP. This causes the 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]

**2.7.2、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.

**2.7.3、Tcl Command Example of an IP Clock Dependency**

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

**2.7.4、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.

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

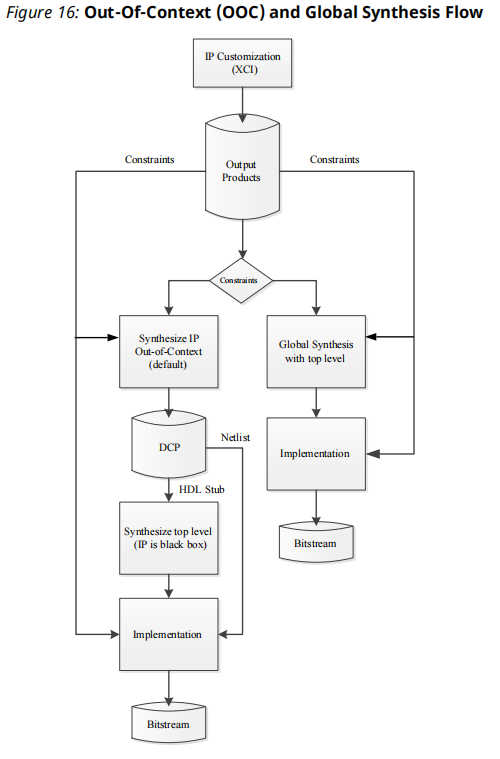
report\_compile\_order -constraints

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.

**2.7.5、Examples of Critical Warnings and Warnings on Clocking**

**2.8、Synthesis Options for IP**



**2.8.1、Global Synthesis Flow**

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.

**2.8.2、Out-of-Context Flow**

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 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.

**2.9、Simulating IP**

When IP output products are generated, several simulation models are created that you can include in the simulation of the overall design.

• 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.

**2.9.1、Delivering IP Simulation Models**

Most AMD IP deliver RTL sources for a single language only, either Verilog or VHDL.

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.

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 the IP does not deliver a behavioral model

the Vivado tools automatically generate a structural simulation netlist

(<ip\_name>\_sim\_netlist.v or <ip\_name>\_sim\_netlist.vhdl) to support simulation. If your simulator language is not set to Mixed, you might be required to generate the IP using the default OOC synthesis. an \_sim\_netlist.v or an \_sim\_netlist.vhdl files are created as part of the OOC synthesis flow only.

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.

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.

**2.9.2、Verification IP**

Verification IP can be helpful when performing simulation on designs that are using AXI IP.

• AXI Verification LogiCORE IP Product Guide (PG267)

• AXI4-Stream Verification LogiCORE IP Product Guide (PG277)

• Zynq 7000 SoC Verification IPData Sheet (DS940)

• Zynq MPSoC UltraScale Verification IP Data Sheet (DS941)

**IMPORTANT!** The AXI Verification IP is written in **SystemVerilog** and uses **randomization**. Not all thirdparty 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.

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>]

**2.9.3、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]

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

**2.9.4、Using a Test Bench for IP**

Many IP in the IP catalog also deliver a test bench for simulating the IP standalone. 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.

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.

**2.10、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.

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.

• 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.

**2.10.1、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). 老版本IP可在新版IDE中继续使用，同时也可以选择升级IP。

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) .

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

**2.10.2、Upgrading IP using a Tcl Command**

upgrade\_ip

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

upgrade\_ip [get\_ips cfifo] -log c:/prj/IP/cfifo\_upgrade.log

**2.11、Understanding Multi-Level IP**

Some IP are designed to use other IP as design sources. 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.

**2.12、Working with Debug IP**

The benefits of debugging your design insystem include debugging your timing-accurate, post-implemented design in the actual system environment running at system speeds.

The available debug IP cores include: 四种

• **Vivado Integrated Logic Analyzer**: 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 insystem serial I/O validation and debug. This allows you to measure and optimize your highspeed serial I/O links in your FPGA-based system.

• **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 LogiCORE IP Product Guide (UG174)

○ 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)

**2.12.1、Debugging Flows**

• 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.

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

**2.13、Using a Core Container**

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

set\_property coreContainer.alwaysCreateXCI 1 [current\_project]

This optional feature lets you elect to have IP and all generated output files contained in one

compressed binary file with an extension of XCIX. This extension is similar to the XCI file used for the IP customization file and works in a similar way.

When 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.

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

**2.13.1、Enabling and Disabling the Core Container**

• 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**.

**2.13.2、Simulating with Core Container**

If only the XCIX is available, these files can be extracted using the export\_ip\_user\_files Tcl command.

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>\

**2.13.3、Support Files for Core Container**

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>\

**2.13.4、Tcl Command to Export Support Files**

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.

**2.13.5、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.此文件目录主要是为了在使用XCIX文件管理IP时，方便储存过程文件。

The following is a brief description of each of the directories that could be present. Appendix B: IP Files and Directory Structure.

• bd: Contains a sub-folder for each IP integrator block design (BD) in the project.

• ip: Contains files specific to each IP customization (XCI/XCIX).

•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.

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.

Simulator brands:

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)

*Chapter3 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**. 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.

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.

将使用的IP独立于项目管理，方便版本控制。这些在设计项目中使用的多个IP，可以独立建立一个manage ip project，这个ip project与设计项目相关联，但使用一个独立于设计项目的文件夹。当然，不同的设计项目可以关联不同的manage ip project，因而存在多个ip project文件夹，此时要注意这些文件夹的命名和分类。

**3.1、Using the Manage IP Flow**

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

**Note**: This flow does not support subsystem IP.

**3.1.1、Managing IP Settings**

Vivado IDE opens the Managed IP project, and you can now select and customize IP. 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.

**3.1.2、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.

**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.

**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.

*Chapter4 Using IP Example Designs*

The example design project consists of toplevel 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.

**4.1、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.（在资源窗口中右键打开）

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建议将IP纳入版本控制，而设计项目不要纳入版本控制？）

**4.1.1、Tcl Command to Open a example Project**

open\_example\_project [get\_ips <ip\_name>]

**4.1.2、Examining Standalone IP**

When performing timing analysis to an IP that has already been synthesized, after implementation completes, the results are not accurate because the clocks are not yet routed and ideal clocks are used. Some IP include the HD.CLK\_SRC property in the <ip\_name>\_ooc.xdc file, which provides a location to a clock buffer and the **SLEW** timer models to improve the accuracy of post implementation timing analysis.

*Chapter 5 Using AMD IP with Third-Party Synthesis Tools*

AMD supports netlists created by third-party synthesis tools for user logic. 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.

**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.

**5.1、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，Xilinx Parameterized Macros), 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.