Notice of Disclaimer

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

©Copyright 2012-2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, Vivado, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners...

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/26/2013</td>
<td>2013.2</td>
<td>Labs #1 and #2 merged. Labs #7 and #8 eliminated. Remaining labs renumbered, updated and validated with the release.</td>
</tr>
<tr>
<td>05/07/2013</td>
<td>2013.1</td>
<td>Added Steps 4 and 5 to Lab #3.</td>
</tr>
<tr>
<td>04/25/2013</td>
<td>2013.1</td>
<td>Minor editorial updates.</td>
</tr>
<tr>
<td>04/10/2013</td>
<td>2013.1</td>
<td>Labs updated and new labs added for the 2013.1 release.</td>
</tr>
</tbody>
</table>
# Table of Contents

Revision History ........................................................................................................................................... 2
Table of Contents ........................................................................................................................................... 3

Designing with IP .......................................................................................................................................... 6

Overview ....................................................................................................................................................... 6
Tutorial Description ....................................................................................................................................... 7
Software Requirements ................................................................................................................................. 7
Hardware Requirements ............................................................................................................................... 7
Locating and Preparing the Tutorial Design Files ....................................................................................... 8

Lab 1: Designing with the IP Catalog ......................................................................................................... 9

Step 1: Opening the Project ....................................................................................................................... 9
  Launch Vivado ........................................................................................................................................... 9
  Open the Project ..................................................................................................................................... 10
Step 2: Customizing the FIFO Generator ................................................................................................ 11
Step 3: Generate Output Products ........................................................................................................... 15
Step 4: Instantiating IP into the Design .................................................................................................... 17
Step 5: Synthesizing the Design .............................................................................................................. 20
Conclusion .................................................................................................................................................. 22

Lab 2: Creating and Managing Reusable IP ............................................................................................ 23

Step 1: Starting a Manage IP session ........................................................................................................ 23
Step 2: Customizing the FIFO Generator ................................................................................................ 25
Step 3: Using Third Party Simulators ..................................................................................................... 29
  Using Export Simulation ....................................................................................................................... 30
  Getting Required Simulation Files ..................................................................................................... 30
  Structural Netlists for Simulation ....................................................................................................... 31
Step 4: Additional IP ................................................................................................................................ 32
Step 5: Using Third Party Synthesis Tools ............................................................................................... 33
Conclusion .................................................................................................................................................. 35
Lab 3: Packaging an IP for Reuse ........................................................................................................ 36
  Step 1: Create and Package IP Wizard ............................................................................................ 36
    Launch Vivado ............................................................................................................................. 36
  Step 2: Preparing Design Constraints ........................................................................................... 37
    Analyzing the Current Constraints Files .................................................................................... 38
    Creating an Out-Of-Context (OOC) XDC file ............................................................................ 39
  Step 3: Packaging IP ..................................................................................................................... 43
    Identifying the IP Definition ........................................................................................................ 45
    Adding Documentation to the IP Definition .............................................................................. 47
    Review and Package the IP .......................................................................................................... 50
  Step 4: Validating the New IP ........................................................................................................ 52
  Conclusion ...................................................................................................................................... 56
Lab 4: Scripting the Project Mode .................................................................................................... 57
  Step 1: Creating a Project ................................................................................................................ 57
  Step 2: Adding RTL Source Files ................................................................................................ 58
  Step 3: Adding XDC Constraints .................................................................................................. 59
  Step 4: Adding Existing IP ............................................................................................................ 59
  Step 5: Disabling an IP’s XDC ....................................................................................................... 61
  Step 6: Upgrading an IP ................................................................................................................ 62
  Step 7: Setting up Design Runs for IP ........................................................................................... 63
  Step 8: Launching Synthesis and Implementation ......................................................................... 64
  Step 9: Running the Script ........................................................................................................... 66
  Conclusion ...................................................................................................................................... 68
Lab 5: Scripting the Non-Project Mode ............................................................................................ 69
  Step 1: Reading Design Source Files ............................................................................................ 69
  Step 2: Adding Existing IP ............................................................................................................ 71
  Step 3: Disabling XDC Files .......................................................................................................... 72
  Step 4: Upgrading IP ...................................................................................................................... 73
  Step 5: Creating DCP for IP .......................................................................................................... 75
  Step 6: Running Synthesis ............................................................................................................ 75
Step 7: Running Implementation.................................................................................................................................................. 76
Step 8: Running the Script ................................................................................................................................................................ 78
    Launch Vivado............................................................................................................................................................................. 79
Conclusion..................................................................................................................................................................................... 80
Designing with IP

Overview

The Vivado® Design Suite provides multiple ways to use IP in a design. IP can be customized and added from the IP Catalog into a project. A repository of customized IP can be created and referenced in either a project or non-project based flow, with full scripting capabilities as well.

The Vivado Design Suite provides an IP-centric design flow that helps you quickly turn designs and algorithms into reusable IP. As shown in Figure 1, the Vivado IP catalog is a unified IP repository that provides the framework for the IP-centric design flow. This catalog consolidates IP from all sources including Xilinx® IP, IP obtained from third parties, and end-user designs targeted for reuse as IP into a single environment.

![Figure 1: Vivado Design Suite IP Design Flow](image)

The Vivado IP packager tool is a unique design reuse feature based on the IP-XACT standard. The IP packager tool provides any Vivado user the ability to package a design at any stage of the design flow and deploy the core as system-level IP.

Tutorial Description

This tutorial contains several labs as described below:

- **Lab 1:** Open a modified version of the Xilinx `wave_gen` example design that is missing a FIFO; locate and customize the IP in the catalog; and instantiate the IP into the design.

- **Lab 2:** You will learn how to create and customize IP using the Manage IP flow. Create a project, include an IP from the IP catalog as the top-level source; customize and verify the IP. Optionally, use the customized IP as a black box in a 3rd party synthesis flow.

- **Lab 3:** Package a design as an IP module and add it to the Vivado IP catalog. Open a completed Vivado Design Suite project; package the design as an IP core and add it to the IP catalog using IP packager; then verify the new IP through synthesis and implementation.

- **Lab 4:** Write and run a Tcl script using the Vivado Design Suite to create a project, add IP, upgrade IP, disable IP sources and generate output products including Synthesized Design Checkpoints (.dcp).

- **Lab 5:** Write and run a Non-Project Tcl script using the Vivado Design Suite to read in IP sources, upgrade IP, disable IP sources and generate output products including Synthesized Design Checkpoint (.dcp).

Software Requirements

This tutorial requires that the 2013.3 Vivado Design Suite software release or later is installed. For installation instructions and information, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing*(UG973).

Hardware Requirements

The supported Operating Systems include Redhat 5.6 Linux 64 and 32 bit, and Windows 7, 64 and 32 bit.

Xilinx recommends a minimum of 2 GB of RAM when using the Vivado tool.
Locating and Preparing the Tutorial Design Files

There are separate project files and sources for each of the labs in this tutorial. You can find the design files for this tutorial under *Vivado Design Suite -2013.3 Tutorials* on the Xilinx.com website.

You will find the tutorial files in the **ug939-design-files.zip** that you will need to download and extract into a write accessible location on your hard drive, or network location.

The extracted **ug939-design-files** directory is referred to as <Extract_Dir> in this Tutorial.

---

**RECOMMENDED:** You will modify the tutorial design data while working through this tutorial. You should use a new copy of the **ug939-design-files** directory each time you start this tutorial.
Lab 1: Designing with the IP Catalog

In this exercise, you will learn how to use the IP catalog in the Vivado™ Integrated Design Environment (IDE). You will be using a version of the Xilinx wave generator example project. The design is lacking a FIFO, which you will locate in the IP catalog, customize, instantiate into the design, and generate output products.

**Step 1: Opening the Project**

Figure 2: Vivado IDE - Getting Started Page

**Launch Vivado**

On Linux,

1. Change to the directory where the lab materials are stored:
   
   ```
   cd <Extract_Dir>/lab_1
   ```

2. Launch the Vivado IDE: **vivado**
On Windows,

1. Launch the Vivado Design Suite IDE:
   
   **Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x**

2. As an alternative, click the **Vivado 2013.x** Desktop icon to start the Vivado IDE.

The Vivado IDE Getting Started page, shown in **Figure 2**, contains links to open or create projects and to view documentation.

### Open the Project

1. Select **Open Project** from the Getting Started page and browse to:
   
   `<Extract_Dir>/lab_1/project_wave_gen_ip`

2. Select **project_wave_gen_ip.xpr**, and click **OK**.

![Image of Open Project dialog box](Figure 3: Open Project)

The design will load and you will see the Vivado IDE in the default layout view, with the Project Summary information as shown in **Figure 4**.

---

1 Your Vivado Design Suite installation may be called something other than **Xilinx Design Tools** on the **Start** menu.
Step 2: Customizing the FIFO Generator

Since this is an RTL project, you can run behavioral simulation, elaborate the design, launch synthesis and implementation, and generate a bitstream for the device. The Vivado IDE also offers a one-button flow to generate a bitstream, which will automatically launch synthesis and implementation. For more information, refer to the Vivado Design Suite User Guide: Using the Vivado IDE (UG893).

Step 2: Customizing the FIFO Generator

1. Select **IP Catalog** from the Flow Navigator in the Project Manager area. The Xilinx IP Catalog displays in a new tab.

   You can work with the IP Catalog in a variety of ways. You can search using keywords in the search box or browse through the catalog in the various categories.

2. Type **fifo** in the search box.

   The search results narrow the list of IP definitions displayed in the catalog.
3. From the **Memories & Storage Elements > FIFOs** group select **FIFO Generator**, as shown in **Figure 5**.

![Xilinx IP Catalog – FIFO Core](image_url)

**Figure 5: Xilinx IP Catalog – FIFO Core**

4. Right-click to **open the popup menu**, and select **Customize IP**, or double-click on the selected IP.

The FIFO Generator customize window opens, as shown in Figure 6. There is a schematic symbol for the selected core displayed on the left with only the enabled ports generated. The schematic symbol changes as you customize the IP.

5. **Check** the **Show Disabled Ports** checkbox to view all of the ports available on the FIFO Generator IP.

6. **Zoom** into the schematic symbol **using mouse strokes** with the left mouse button, just like in the Device window.

7. **Uncheck** the **Show Disabled Ports** checkbox to hide unused ports on the symbol.

8. Above the symbol, **open** the **Documentation** menu to examine the options for viewing available information.
The **Documentation** menu lets you open the PDF file datasheet for the IP, open the change log to review the revision history of the core, or open an Internet browser to navigate to the IP webpage, or view any Answer Records related to the IP.

The **IP Location** is used to specify the location to save the IP customization (.xci) and any generated output products. By default, these are saved inside the project structure.

The **Switch to Defaults** resets the configuration options back to the default settings.

9. At the top of the Customize IP dialog box, **change the Component Name to char_fifo** from the default name.

   The Basic tab defines the interface type, memory type, and other implementation options of the IP.

10. Select the **Interface Type** of **Native**, which is the default.

11. From the **Fifo Implementation** drop down menu, set **Independent Clocks Block RAM**.

12. Select the **Native Ports** tab.

   Here you can set the Read Mode, Data Port Parameters, ECC and Output Register Options, and configure Initialization.

---

**Figure 6: Customize IP**
13. Set the **Read Mode** to **First Word Fall Through**.

14. Set the **Write Width** to be **8** bits.
   
   Setting the Write Width will automatically change the Read Width to match, if you click on the Read Width field.

15. **Click** on the **Read Width** field, it will automatically change to 8 bits as well.
   
   Leave everything else with the default settings on this tab.

16. **Examine** the fields of the **Status Flags** and **Data Counts** tabs.
   
   These fields configure other options for the FIFO Generator. For this design, leave everything with the default settings.

17. **Select** the **Summary** tab.
   
   This displays a summary of all the options selected as well as listing resources used for this configuration. The summary for the FIFO Generator customization should look like [Figure 7](#).

   For this configuration you are using one **18K BRAM**.

   ![Figure 7: Summary of FIFO Generator Core](image)

18. **Verify** that the information is correct as shown, and click **OK** to add the IP customization to your design.
Step 3: Generate Output Products

After the char fifo IP customization completes, the Generate Output Products dialog box opens as seen in Figure 8. The output products for an IP are the various files required to support the use of the IP in synthesis, simulation, and implementation.

Click Generate to generate the required output products.

By default, the Generate Synthesized Checkpoint (.dcp) is selected and an Out-of-Context Module Run for the char_fifo IP is added to the Design Runs window, and the run is launched to synthesize the IP as shown in Figure 9.
The Out-of-Context (OOC) run creates the Design Checkpoint File (DCP) for the specified IP customization. This allows the IP to be complete with regard to synthesis, helps to insure that the integrity of the core is preserved in the current design, and reduces synthesis time for the top-level design in future iterations of the design flow. For more information on OOC runs, and the use of DCP files, refer to the Vivado Design Suite User Guide: Designing with IP (UG896).

**IMPORTANT:** Xilinx recommends that you generate the DCP file to reduce synthesis time for the top-level design. It is also recommended that you do not change the default synthesis setting for the IP Design Runs.

The FIFO now appears in the Sources view. The IP Sources tab is shown in Figure 10. The FIFO Generator customization includes an instantiation template, `char_fifo.veo`, synthesis constraints and VHDL entity and architecture definition, Verilog simulation files, and the synthesized design checkpoint.

Since the FIFO IP was originally defined in VHDL, the entity and architecture are added as VHDL source files. However, because Verilog is the target language for the current project, the instantiation template is a VEO file for instantiating the VHDL FIFO entity into the Verilog design.

You can also customize and add IP to your design, using TCL commands.

In the IP Sources tab of the Sources window, examine the output products produced for the FIFO Generator customization.
3. Examine the Tcl Console in the Vivado IDE, as shown in Figure 11, and **review** the **Tcl commands** used to add the FIFO Generator core to your project.

![Figure 11: Tcl Console Commands for Adding IP](image)

- The `create_ip` command adds the IP into the current project.
- The `set_property` command defines the various configuration options of the Customize IP dialog box.
- The `generate_target` command creates the specified output products for the IP customized.
- The `create_ip_run` command defines the Out-of-Context synthesis run for the IP customized.
- The `launch_run` command runs synthesis on the IP customization.

Refer to the **Vivado Design Suite Tcl Command Reference Guide** *(UG835)* for specific information on the different Tcl commands used in this tutorial.

### Step 4: Instantiating IP into the Design

You will now instantiate the IP customization into the design by copying and pasting the Verilog Instantiation Template into the appropriate Verilog source file in your project and modifying the signals.

1. In the IP Sources tab of the Sources window, expand the **Instantiation Template** and double click on the `char_fifo.veo` file to open the template in the Vivado Text Editor.
2. Scroll down to line 56 or 57 of the template file, and select and copy the module instantiation text, as seen in Figure 12.
Step 4: Instantiating IP into the Design

Next, you will copy the instantiation template into the appropriate RTL source file. In this case, you will copy the module into the top-level of the design, in the `wave_gen` source file.

3. From the **Hierarchy** tab of the **Sources** view, double click on `wave_gen.v` to open this file for editing.

4. Go down to line **337, which** contains a comment stating the Character FIFO should be instanced at this point. Paste the template code into the file as shown in Figure 14.

   However, since it is only a template for the module, you will need to do some local editing to make the module work in your design.
5. **Change** the module name from `your_instance_name` to `char_fifo_i0`.

6. **Change** the **wire names** as follows, to connect the ports of the module into the design:

   ```
   char_fifo char_fifo_i0 (
      .rst(rst_i), // input rst
      .wr_clk(clk_rx), // input wr_clk
      .rd_clk(clk_tx), // input rd_clk
      .din(char_fifo_din), // input [7 : 0] din
      .wr_en(char_fifo_wr_en), // input wr_en
      .rd_en(char_fifo_rd_en), // input rd_en
      .dout(char_fifo_dout), // output [7 : 0] dout
      .full(char_fifo_full), // output full
      .empty(char_fifo_empty) // output empty
   );
   ```

7. In the Text Editor side-bar menu, click on the **Save File** button () to save the changes to the `wave_gen.v` file.

   Notice that the Hierarchy, Libraries, and Compile Order tabs are updated to indicate that the IP has been instanced into the design, as seen in Figure 15.
Step 5: Synthesizing the Design

With the customized IP integrated into your design, you are now ready to proceed to synthesis. All Xilinx IP in the Vivado Design Suite IP Catalog are delivered as RTL sources. This provides the benefit of being able to perform behavioral simulation, which is much faster than netlist-based simulation. However, having to synthesize each IP, along with the overall design, with every design iteration, can add significant synthesis time to the project.

The design checkpoint file delivered with the IP output products, and generated by the Out-of-Context synthesis run, can eliminate the need to re-synthesize the core over multiple iterations.

The default behavior of the Vivado Design Suite, when you add an IP to the project, is to generate the necessary output products including the DCP file. However, you can disable generation of the DCP file, or skip generating output products altogether. If you skip generation during customization, the Vivado Design Suite will automatically generate the required output products at the point in the design flow they become required, such as during synthesis or simulation. By default, a DCP will also generated during synthesis unless you disable and generate the output products prior to synthesis.

1. Look at the clk_core IP customization instantiated under wave_gen as seen in Figure 15.

   Notice the clk_core icon lacks a check mark as seen on the char_fifo IP. This indicates that this IP was customized, and added to the project, but no output product were generated for the IP. These output products will be generated automatically at this time.

2. In the Flow Navigator, click the Run Synthesis button.
The Vivado Design Suite automatically generates the required output products for the clk_core IP as shown in Figure 16.

![Sources tab showing IP cores and synthesis files](image)

**Figure 16: Clk Core Output Products**

The Vivado tool creates a new Out-of-Context Module synthesis run for the clk_core, and launches that synthesis run, as seen in Figure 17. This synthesis run creates the synthesized checklist (.dcp) file for the IP customization. Now when the top-level of the design is synthesized, a black box will be inferred for the FIFO generator IP (char_fifo) and the Clock generator IP (clk_core) in the design.

![Design Runs tab showing synthesis runs](image)

**Figure 17: Out-of-Context Synthesis Runs**

With the required output products for the clk_core created, the Vivado synthesis tool is run on the top-level of the design, as seen in Figure 18. Notice that the OOC synthesis runs for both char_fifo and clk_core are complete.
3. When the Synthesis Complete dialog box opens, select the **View Reports** option and click **OK**.

   This opens the Reports window at the bottom of the Vivado IDE.

4. Switch to the Log window and **open the Synthesis log**.

5. Use the **Show Find** command, ![Find](image), to search the Log window for “blackbox”.

6. Use the **Find Next** command until you come to the following section, which summarizes the blackboxes found in the current design:

   ```plaintext
   Report BlackBoxes:
   +-----------------+-----------------+-----------------+
   |                | BlackBox name   |     Instances   |
   |-----------------+-----------------+-----------------|
   | 1               | char_fifo       |     1           |
   | 2               | clk_core        |     1           |
   +-----------------+-----------------+-----------------+
   ```

7. **Look** in the project **IP runs folder** for the results of the out-of-context synthesis runs:

   ```plaintext
   <Extract_Dir>/lab_1/project_wave_gen_ip/project_wave_gen_ip.runs
   ```

   The checkpoint files can also be used in other projects. It encapsulates the netlist and the constraints for the out-of-context IP module.

## Conclusion

You have successfully created a FIFO Generator IP customization, and instanced it a design. Close the project and exit the Vivado tool, or continue and implement the design if you wish to explore further. This concludes Lab1.

In this Lab, you have learned how to select and customize an IP from the IP catalog, how to instantiate the customized IP into an HDL design, and some details of the output products required to support the IP in the design flow. You can add and manage IP in a design interactively within the Vivado IDE, or via Tcl scripting.
Lab 2: Creating and Managing Reusable IP

To simplify revision control, and to support the use of custom IPs across multiple projects and designs, you can manage and store the customized IP in a repository, separate from any design projects they are used in. The IP customization file (.xci), and the output products for synthesis, simulation, examples, etc., should be contained together in a unique directory. You can reference these IP customizations in new projects and designs, to simulate, synthesize, and implement the IP as part of the design. Having all the generated output products available also preserves that customized version of the IP for use in a future release of the Vivado Design Suite, even if the IP is updated in the Xilinx IP Catalog. See the Vivado Design Suite User Guide: Designing with IP (UG896) for more information on managing IP.

In this exercise, you will create and verify an IP customization in an IP repository, using the Manage IP flow in the Vivado Design Suite. This flow allows you to browse the IP catalog to create and manage IP customizations for use in either a Project or a Non-Project design flow. You can create a repository of the customized IP for use in your design(s), managed and maintained outside of a Vivado Project. A special IP project is created at the location specified for the Manage IP flow, to facilitate the creation of a Synthesis Design Checkpoint (DCP) and structural simulation models. When using a customized IP in a project on non-project flow reading/adding the IP customization file (.xci) all output products, including a DCP if present, are used. Using the synthesis DCP files speed synthesis of the top level. A stub file is provided for use in third party synthesis tools to infer a black box for IP.

---

Step 1: Starting a Manage IP session

On Linux,

1. Change to the directory where the lab materials are stored:
   
   `cd <Extract_Dir>/lab_2`

2. Launch the Vivado IDE: vivado

On Windows,

1. Launch the Vivado Design Suite IDE:
   
   `Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x`

2. As an alternative, click the Vivado 2013.x Desktop icon to start the Vivado IDE.

   The Vivado IDE Getting Started page displays with links to open or create projects, and to view documentation.

---

2 Your Vivado Design Suite installation may be called something different than Xilinx Design Tools on the Start menu.
3. On the Getting Started page, click the **Manage IP** link.

![Manage IP](image)

**Figure 19: Manage IP from the Getting Started Page**

A drop down menu displays, as seen in Figure 19, letting you:

- Specify a New IP location for a new Manage IP project and generated output products.
- Open an existing IP location for prior Manage IP projects.

4. Select **New IP Location**.

5. Press **Next** to move to the Manage IP Settings dialog box, as seen in Figure 20.

6. Change the dialog box settings as follows:

- IP location: `<Extract_DIR>/lab_2/my_ip`
- Part: `xc7k70tfbg484-2`

  **Note**: Use the browse button to select the specified target part.

![Manage IP Settings](image)

**Figure 20: Manage IP Settings**

7. Press **Finish** to proceed.

The IP Catalog displays in an IP Project, which is a simple interface for the creation and management of IP customizations.
Step 2: Customizing the FIFO Generator

You can work with the IP catalog in two ways, either searching with a keyword, or browsing through the categories.

1. Type `fifo` in the search bar.
2. Double-click the **FIFO Generator** from the Memories & Storage Elements group. The Customize IP dialog box opens.

![Figure 21: Customize IP Window](image)

**TIP:** For a complete description of the Customize IP dialog box, and its use, refer to [Lab 1: Designing with the IP Catalog](#). This Lab assumes that you have previously completed Lab 1, and are familiar with the concepts covered in it.

3. At the top of the Customize IP dialog box, change the **Component Name** to `char_fifo` from the default name of `fifo_generator_0`, as shown in Figure 21.

4. In the Basic tab:
   - Select the **Interface Type** of **Native**, which is the default.
   - From the **Fifo Implementation** drop down menu, set **Independent Clocks Block RAM**.
5. In the **Native Ports** tab.
   - Set the **Read Mode** to **First Word Fall Through**.
   - Set the **Write Width** to be **8 bits**.
   - Click on the **Read Width** field to adjust it automatically to 8 bits as well.

6. Select the **Summary** tab.

   The Summary page displays a summary of all the options selected as well as listing resources used for this configuration. The summary for the FIFO Generator core should look like Figure 22. For this configuration you will see you are using one 18K BRAM.

![Figure 22: Summary of FIFO Generator Core](image)

7. **Verify** that the information is correct as shown, and click **OK** to generate the customized IP core for use in your design.

   The Generate Output Products popup appears, as seen in Figure 23.

8. Leave the **Generate Synthesized Design Checkpoint (.dcp)** box **checked**, as this is the default flow.

   When creating a repository of customized IPs using the Managed IP flow, Xilinx recommends that you generate all output products for each IP, including the design checkpoint file (DCP). A specific release of the Vivado Design Suite only supports a single version of an IP core. You cannot re-customize or generate output products for a previous version of IP in the Vivado Design Suite. Instead, you would need to update the IP to the latest version if you have not preserved the needed output products.
IMPORTANT: Only one version of an IP is supported in a given release of the Vivado tools. To preserve older versions of an IP, the output products must be available in your custom IP repository.

9. Click the Generate button to create the output products.

![Figure 23: Generate Output Products](image)

To reduce synthesis runtime for a design using one or more customized IP cores, the IP can be pre-synthesized as a standalone module, with the netlist saved in a design checkpoint file (DCP). During synthesis of the overall design, a black box is inferred for the IP. During implementation the netlists from the DCP files are linked into the design. This method can be scripted in Tcl for both Project and Non-Project Mode.

When the synthesis DCP is specified for an IP core, a corresponding Out-of-Context Module Run is also created and launched, as seen in Figure 24.

![Figure 24: Out-of-Context Module Run to Create the DCP](image)
This run is automatically launched, and when synthesis is completed the **char_fifo** directory contains a few additional files:

- **char_fifo.dcp**: The Synthesis Design Checkpoint which consists of both a netlist and constraints for the IP.
- **char_fifo_stub.v**: A Verilog port module for use with Synplify Pro, to infer a black box for the IP with no IO buffers inserted.
- **char_fifo_funcsim.v/char_fifo_funcsim.vhd**: A Verilog and VHDL netlist for functional simulation of the IP core.

If you are using a third party synthesis tool for the design, a Verilog stub file with the port declarations, or a VHDL component declaration, is required for the black box to be inferred. The Vivado Design Suite automatically creates this file along with the synthesis design checkpoint (DCP) when the output products are generated.

Customized IPs can be referenced from both Project and Non-Project Mode. In a project-based design, Xilinx recommends that you do not copy sources into the local project structure, but rather reference them from your custom IP repository. For more information on Project Mode and Non-Project Mode, refer to the *Vivado Design Suite User Guide: Design Flows Overview (UG892)*.

10. **Examine** the `<Extract_Dir>/lab_2/my_ip` location.

You will notice that two directories have been created, as shown in Figure 25.

- One is the IP customization (**char_fifo**) containing the XCI file, which has all the customization information for the IP, and all the output products generated.
- The second is the IP Project (**managed_ip_project**) which you created in this Lab to customize the char_fifo IP and manage the output products, including the synthesized DCP.

11. **Examine** the **Sources window** to see the generated output products in the Manage IP project.
The generated output products for the char_fifo core include the Instantiation Template, Synthesis files, Simulation files, and the synthesis design checkpoint.

**Step 3: Using Third Party Simulators**

The purpose of the Managed IP project is to create and manage IP customizations; there is no support for directly simulating IP in a Managed IP project. Customized IP can be instanced into a standard design project for simulation, in either Project or non-Project Mode.

The Vivado Design Suite includes the Vivado simulator, for mixed language simulation, as well as directly supporting Mentor Graphics® ModelSim/QuestaSim. Refer to the Vivado Design Suite User Guide: Designing with IP ([UG896](#)) and the Vivado Design Suite User Guide: Logic Simulation ([UG900](#)) for more information on simulating with these simulators, including creating scripts for running simulations outside of the Vivado Design Suite.

Xilinx IP delivered in the Vivado Design Suite are encrypted using industry standard IEEE P1735 encryption. Simulators such as Cadence® Incisive Enterprise Simulator (ies) and Synopsys® VCS MX simulator (vcs_mx) support this encryption standard and can be used to run behavioral simulation. A list of simulation files and the libraries they belong to is needed.

There are two options available:

- Use the `export_simulation` command to export a script and associated data files (if needed) for driving simulation using Cadence IES or Synopsys VCS.
- Use query commands to get the simulation files for the IP and the libraries they are associated with.
Using Export Simulation

To create scripts for Cadence/Synopsys simulators, use the `export_simulation` command.

1. In the Tcl Console, type the following command:
   ```tcl
   export_simulation -simulator ies -of_objects [get_ips char_fifo] \
   -directory "c:/Data/ug939-design-files/my_ip" -script_name test_ies.sh
   ```
   This will create a file called “test_ies.sh” for simulating with Cadence IES. A default ".do" file that is used in the compiler commands in the simulation script for the target simulator, will be written to the output directory, in this case it is called “char_fifo.do”.

2. For more information on the various options available for this command:
   ```tcl
   export_simulation -help
   ```

Getting Required Simulation Files

As an alternative to export_simulation, you can manually gather the required files to support third-party simulation. To get all files that an IP delivers for simulation, use the `get_files` Tcl command.

1. In the Tcl Console, type the following command:
   ```tcl
   get_files -compile_order sources -used_in_simulation -of_objects \
   [get_files char_fifo.xci]
   ```
   The `-used_in` option lets you specify files that are marked for use in simulation, or marked for use in synthesis. The `-of_objects` option lets you extract files that are associated with the specified IP customization file.

   This will produce a list of file names, including the full path, required to simulate the IP. In this case, the list includes:
   ```
   .../my_ip/char_fifo/fifo_generator_v11_0/simulation/fifo_generator_v11_0.v
   .../my_ip/char_fifo/sim/char_fifo.v
   ```
   Each simulation file has a LIBRARY property that can be queried. For VHDL files, the library associated with each file is required for simulation.

2. To extract the LIBRARY property, type the following Tcl command:
   ```tcl
   get_property LIBRARY [get_files char_fifo.v]
   ```
   This should return the work Library.

3. Use the following Tcl script to print out each file used for simulation, including the path, and its associated library:
   ```tcl
   # Get the list of files required for simulation
   set ip_files [get_files -compile_order sources -used_in_simulation \
   -of_objects [get_files <IP name>.xci]]
   # For each of these files, get the library information
   foreach file $ip_files { 
     puts "[get_property LIBRARY $file] $file"
   }
   ```
In the preceding script, replace <IP_Name> with the name of the customized IP to extract files from. In this case, you would use char_fifo.

**Structural Netlists for Simulation**

Depending on which **Simulator Language** you select during the Manage IP project creation (see Figure 20) the results of the queries above will differ. If you did not specify support for a “Mixed” simulator and instead selected “Verilog” or “VHDL” then depending on the IP you might not be able to do behavioral simulation. You will be required to do simulations using a structural netlist, which will be produced automatically for you.

When generating the output products, if the IP can deliver behavioral simulation files based upon your simulator language selection you will see “Behavioral Simulation” listed as an output product as shown in Figure 23. If however, the IP does not deliver simulation files for the simulator language supported you will instead see “Structural Simulation” as shown in Figure 27.

![Figure 27 Structural Simulation Output]

If this IP were queried for simulation files using:

```
get_files -compile_order sources -used_in simulation -of_objects \
[get_files fir_0.xci]
```

The only file returned would be:

```
../my_ip/fir_0/fir_0_funcsim.v
```

**Note:** Structural simulation files are always created when a Synthesized Design Checkpoint is generated as shown in Figure 26. It is just the querying results of the simulation files that will differ depending on whether behavioral simulation was possible with the simulator language setting used.
Step 4: Additional IP

At this point in the tutorial, you can repeat some of the earlier steps to add some additional IP cores to the Managed IP project, and customize them as well. In Figure 28, you can see that two additional cores, Tri Mode Ethernet MAC and Complex Multiplier, have been added to the Manage IP project.

1. Explore the IP Catalog, and create customizations for a few additional IP cores.
2. Generate output products for the additional cores.

   In the IP Sources tab of the Sources window, you will see the various output products that have been generated for the additional cores.

   ![Figure 28: Output products of an IP](image)

When an IP customization completes, an Instantiation Template is generated, even if you choose to skip generating output products. The Instantiation Template is the minimum output.

If you included the DCP file in the Generate Output Products dialog box, Out-of-Context Module Runs are added and launched for each of the IP cores. The progress of these synthesis runs can be viewed in the Design Run tab, as shown in Figure 29.
Step 5: Using Third Party Synthesis Tools

Xilinx IP delivered with the Vivado Design Suite only supports synthesis using the built in Vivado synthesis tool. User logic can be synthesized using supported third party synthesis tools, such as Synopsys® Synplify Pro.

A Verilog stub for the IP is required so the third party synthesis tool can infer black boxes for Xilinx IP used. It is important that the synthesis tool does not insert IO buffers for any Xilinx IP. The stub file (<ip_name>_stub.v) that is produced when the synthesis design checkpoint (DCP) is created contains synthesis directives to do both these. The stub file is only created if a DCP has been generated for the IP.

The following sample Tcl script, <Extract_Dir>/lab_2/run.tcl, reads a top-level design netlist including the char_fifo, reads the char_fifo synthesized DCP, then reads the design constraints and runs implementation, in the Vivado tool Non-Project Mode. The synthesis design checkpoint file contains constraints for the IP, as well as the netlist.

```tcl
# Change directory to <Extract_Dir>/lab_2
cd <Extract_Dir>/lab_2

# Read top-level EDIF produced by 3rd party synthesis tool
read_edif ./sources/wave_gen.edif

# Read IP DCP
add_files ./my_ip/char_fifo/char_fifo.dcp

# Read top level constraints
read_xdc ./sources/wave_gen_timing.xdc

# Link the netlists
link_design -top wave_gen -part xc7k70tfbg484-2

# Logic optimization
opt_design

# Placement
```
Step 5: Using Third Party Synthesis Tools

```groovy
place_design
# routing
route_design

# write out a checklist
write_checkpoint wave_gen_post_route.dcp
```

Although there could be many instances of an IP customization in a design, the DCP file only needs to be read once. All black boxes for that IP customization will be replaced with the netlist from the DCP during implementation, and the design constraints will be applied to each instance.

1. **Close** the Manage IP project which is now open in the Vivado IDE.
2. In a Text Editor like Notepad or Emacs, **open** the file `<Extract_Dir>/lab_2/run.tcl`.
3. **Edit** the file to change `<Extract_Dir>` to the actual location of the extracted files.
4. Using the **Tools > Run Tcl Script** command from the main menu, **source** the `run.tcl` script.
   The netlist from the synthesis DCP of the IP core is linked to the top-level EDIF netlist, and implementation is run.
5. **Examine** the Netlist window to see that the char_fifo IP was correctly linked to the top-level design.

![Netlist Window](image)

**Figure 30: Netlist Window**

This concludes Lab #2. You can continue examining the design, and create additional reports, or exit the Vivado Design Suite.
Conclusion

In this lab, you learned how to use the Manage IP flow to browse the IP Catalog and create IP customizations to store in an IP repository for later reuse. It is a convenient method for creating custom IPs to manage under revision control for use in future designs. From the Manage IP Flow you can easily create a synthesis design checkpoint (DCP) to use the custom IP in Vivado Project Mode or Non-Project Mode, or for inferring a black box for use with a third-party synthesis tool. You can also generate a structural netlist for simulation if needed.

To perform behavioral simulation with a custom IP, you would use the IP in a project, or generate scripts for Vivado simulator or ModelSim/ QuestaSim for standalone simulation. To use third party simulators, you will need to either query the HDL files required for simulation, and the libraries with which the files are associated or use the `export_simulation` Tcl command.
Lab 3: Packaging an IP for Reuse

In Lab #3, you will define a new IP from an existing design, using the Create and Package IP wizard. You will start with an existing design project in the Vivado IDE, define identification information for the new IP, add documentation to support its use, and add the IP to the IP Catalog. After packaging, you will verify the new IP through synthesis and implementation in a separate design project.

The Lab #3 project contains Verilog source files for a simple UART interface. The project consists of seven source files as well as an XDC file with timing information.

Step 1: Create and Package IP Wizard

Launch Vivado

On Linux,

1. Change to the directory where the lab materials are stored:
   ```
   cd <Extract_Dir>/lab_3
   ```
2. Launch the Vivado IDE: `vivado`

On Windows,

1. Launch the Vivado Design Suite IDE:
   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x
   ```
2. As an alternative, click the `Vivado 2013.x` Desktop icon to start the Vivado IDE.
   
   The Vivado IDE Getting Started page displays with links to open or create projects, and to view documentation.
   
   For either Windows or Linux, continue the lab from this point.
3. Click Open Project, and browse to: `<Extract_Dir>/lab_3/my_simple_uart`
4. Select the `my_simple_uart.xpr` project and click OK.

   The design will load and you will see the Vivado IDE in the default layout view, with the Project Summary information as shown in Figure 31.

---

Your Vivado Design Suite installation may be called something different than Xilinx Design Tools on the Start menu.
Step 2: Preparing Design Constraints

The existing design includes timing constraints defined in an XDC file (uart_top.xdc). These constraints were defined for the UART design as a standalone design. However, when it is packaged as an IP, the design will inherit some of the needed constraints from the parent design. In this case the XDC file must be modified to separate constraints the IP requires when used in the context of a parent design, and the constraints the IP requires when used out-of-context (OOC) in a standalone capacity. This requires splitting the current XDC file.

You should prepare the design constraints prior to packaging the design for inclusion in the IP catalog; however, these steps can also be done after packaging the IP.

**IMPORTANT:** Starting with the Vivado 2013.3 release, a synthesized design checkpoint (DCP) is created as part of the default Out-of-Context (OOC) design flow for IP packaging and use.

To ensure that the packaged IP will function properly in the default Out-of-Context (OOC) design flow, the IP packaging should include a standalone XDC file to define all external clocking.
Step 2: Preparing Design Constraints

information for the IP. This standalone or OOC XDC file is used by the Out-of-Context synthesis run to properly constrain the IP to the recommended clock frequency. After OOC synthesis, the standalone XDC file is not needed by any top-level design that the IP is used in.

For more information on the Out-Of-Context (OOC) design flow, and the use of the DCP file, please refer to the Vivado Design Suite User Guide: Designing with IP (UG896).

TIP: Depending on the function and use of the packaged IP, the design constraints may also have to be adjusted to ensure proper scoping. For more information refer to Constraints Scoping in the Vivado Design Suite User Guide: Using Constraints (UG903).

Analyzing the Current Constraints Files

1. Open the target XDC file (*uart_top.xdc*) listed under the Constraints folder in the Hierarchy pane of the Sources window.

![Image](image.png)

**Figure 32: File Contents of uart_top.xdc**

There are two items to take note of in the XDC file, as seen in Figure 32:

- create_clock constraints (Lines 1 and 2)
- set_max_delay constraint relying on the clock object period value (Lines 16-18)

All create_clock constraints should be analyzed prior to packaging the new IP Definition. If the created clock is internal to the IP (GT), or if the IP contains an input buffer (IBUF), the create_clock constraint should stay in the IP XDC file because it will be needed to define
local clocks. Otherwise, the clock constraints should be moved from the IP XDC file to an OOC XDC file.

For this example, you will move the create_clock constraints on line 1 and 2 from the design XDC file to an OOC XDC file. When the user instantiates the IP from the catalog into a design, the IP will inherit the clock definitions from the parent design.

The set_max_delay constraint is also noteworthy in that it has a dependency on defined clocks, (get_clocks -of_objects). This dependency is affected by the order of processing of the constraints of the IP and top-level design.

By default, when IP customizations are instantiated into a design, the Vivado Design Suite processes the XDC files of an IP before the XDC files of the top-level design. This is known as EARLY processing, and is defined by the PROCESSING_ORDER property on the XDC file.

The XDC files of the top-level design are marked for NORMAL processing by default. This means that the processing of XDC files for IP constraints will happen before the top-level design constraints created by the user. However, in this case, the dependency on the clock definition will cause errors in processing the IP constraints early. To resolve this issue, you will mark the XDC files of the UART IP for LATE processing.

---

**TIP:** Xilinx delivered IP with ".clock" appended to the XDC filename are all marked for LATE processing.

---

**Creating an Out-Of-Context (OOC) XDC file**

1. Select the Add Sources button in the Flow Navigator panel or from the File menu, select Add Sources.
   - The Add Sources dialog box opens.
2. Select Add or Create Constraints, click Next.
3. In the Add or Create Constraints pane, click the Create File button.
4. In the Create Constraints File dialog box, fill in the constraints file information with the following, as shown in Figure 33:
   - File type: XDC
   - File name: uart_top_ooc.xdc
   - File location: <Local to Project>
5. Click OK.
Step 2: Preparing Design Constraints

**Figure 33: Create Constraints File dialog box**

---

**TIP:** For Xilinx delivered IPs, the Out-of-Context XDC file has “_ooc” appended to the filename. However, it is the USED_IN property of the file, not the filename, that determines if it is an OOC XDC file.

6. Click **Finish** to complete the Add Sources dialog box.

   The new XDC file will be created in the project and display under the Constraints section in the Hierarchy pane of the Sources window.

   You will now move the `create_clock` constraints from the XDC file of the original design (`uart_top.xdc`) into the OOC XDC file (`uart_top_ooc.xdc`).

7. Open the new OOC XDC file (`uart_top_ooc.xdc`) by double-clicking the file in the Sources window.

   The file is empty.

8. Cut and paste the `create_clock` constraints, from lines 1 and 2 of the IP XDC file (`uart_top.xdc`) into the empty OOC XDC file.

   The OOC XDC file should now contain only those two `create_clock` constraints as shown in Figure 34.

---

**Figure 34: OOC XDC**
9. Select the **Save File** button, to save the updated contents of the OOC XDC file.

10. **Check** to be sure that the `create_clock` commands are removed from the IP XDC file (`uart_top.xdc`), and **save** the file.

As previously mentioned, the `create_clock` constraints are no longer needed because the clocks will be defined by the parent design. The IP XDC file should now only contain the constraints as shown in Figure 35. The OOC XDC file defines the clocks needed for standalone processing.

```
1 set_multicycle_path -from {get_cells "uart_rx_i0/uart_rx_ctl_i0/*" -filter [IS_SEQUENTIAL]} \
  -to {get_cells "uart_rx_i0/uart_rx_ctl_i0/*" -filter [IS_SEQUENTIAL]} 108
2 3
4 set_multicycle_path -from {get_cells "uart_rx_i0/uart_rx_ctl_i0/*" -filter [IS_SEQUENTIAL]} \
  -to {get_cells "uart_rx_i0/uart_rx_ctl_i0/*" -filter [IS_SEQUENTIAL]} 107
5 6
7 set_multicycle_path -from {get_cells "uart_tx_i0/uart_tx_ctl_i0/*" -filter [IS_SEQUENTIAL]} \
  -to {get_cells "uart_tx_i0/uart_tx_ctl_i0/*" -filter [IS_SEQUENTIAL]} 50
8 9
10 set_multicycle_path -from {get_cells "uart_tx_i0/uart_tx_ctl_i0/*" -filter [IS_SEQUENTIAL]} \
  -to {get_cells "uart_tx_i0/uart_tx_ctl_i0/*" -filter [IS_SEQUENTIAL]} 89
11 12
13 set_max_delay -from {get_cells "uart_rx_i0/meta_hardened_rx_i0/image_meta_reg"} \
  -to {get_cells "uart_rx_i0/meta_hardened_rx_i0/image_meta_reg"} [get_property PERIOD {get_clocks -of_objects {get_ports rx_clk}}]
14 15
```

**Figure 35: Updated uart_top.xdc**

With the OOC and IP XDC files defined, you must set the `USED_IN` and `PROCESSING_ORDER` properties on the XDC files so that the Vivado Design Suite correctly processes the constraint files for the IP.

11. Select the IP XDC file (`uart_top.xdc`) listed under the Constraints section in the **Hierarchy** pane of the **Sources** window.

12. Right-click on the file, and select **Source File Properties** from the popup menu.

13. From the Source File Properties window, scroll down and **change** the `PROCESSING_ORDER` property value to **LATE**, as shown in Figure 36.
The property value can also be changed in the Tcl Console with the following Tcl command:

```
set_property PROCESSING_ORDER LATE [get_files uart_top.xdc]
```

14. In the Tcl console, set the USED_IN property of the OOC XDC file to include the “out_of_context” using the following Tcl command:

```
set_property USED_IN {synthesis implementation out_of_context} \[get_files uart_top_ooc.xdc]
```

When the USED_IN property includes the `out_of_context` setting, the XDC file is only used for synthesis or implementation in Out-of-Context runs (mode out-of-context).

**IMPORTANT:** The **USED_IN property for an OOC XDC file should be (synthesis implementation out_of_context). If it is just out_of_context then it will not be used during synthesis or implementation.**

After completing the above steps, the XDC files are correctly prepared for packaging and the Out-Of-Context (OOC) design flow.
Step 3: Packaging IP

After setting up the design and supporting constraint files, the next step will be to create and package the new IP Definition, and add it to the IP Catalog.

1. From the Tools menu, select the **Create and Package IP** command to open the Create and Package IP Wizard.
   The Welcome window opens for the **Create And Package New IP** dialog box.
2. Click **Next**.
   The **Choose Create Peripheral or Package IP** dialog box appears, as shown in Figure 37.

![Figure 37: Choose Create Peripheral or Package IP Window](image)

3. Choose the **Package your project** option as you will be using the open project as the source for creating the new IP Definition.
4. Click **Next**.
   The Begin IP Creation dialog box, as shown in Figure 38, opens to summarize the information the wizard will gather about the project automatically once the Create and Package New IP Wizard is finished. This will create a basic IP package in a staging area.
5. Click **Finish**.

After completion of the Wizard, the IP Packager Summary dialog box, as shown in Figure 39, opens with the information summarized and added to the new IP Definition.
Step 3: Packaging IP

The following information was found and added to your IP:

**IP Identification**
- xilinx.com:users:uart_top:1.0

**IP Compatibility**
- Added compatibility for families: kintex2

**IP File Groups**
- Found 9 files for Verilog Synthesis.
  - Top Module found for Verilog Synthesis: uart_top
- Found 7 files for Verilog Simulation.
  - Top Module found for Verilog Simulation: uart_top
- Found 1 file for UI Layout.

**IP Customization Parameters**
- Found 2 IP Parameters for customization.

**IP Ports**
- Found 13 ports on your IP.

**IP Interfaces**
- No bus interfaces found for your IP.

**IP GUI Customization Layout**
- Default IP GUI created.

**IP Definition Location**
- C:\Data\ug939-design-files\ab_3\my_simple_uart\component.xml

Next Steps: Walk through each of the items on the left that do not have a check mark or have an exclamation mark and verify the information populated.

---

Figure 39: IP Packager Summary

6. Click **OK**.

**Identifying the IP Definition**

The Package IP window shows the current IP identification information, including Vendor, Library, Name, and Version (VLNV) attributes of the newly packaged IP.

7. In the **Package IP** window, select the **IP Identification pane** in the left side panel, and fill in the right side with the following information, as seen in Figure 40:

   - **Vendor**: my_company
   - **Name**: my_simple_uart
   - **Display name**: My Simple UART
   - **Description**: My simple example UART interface
   - **Vendor display name**: My Company
   - **Company url**: http://www.my_company_name.com
Figure 40: IP Identification

8. For the **Categories** option, select the browse button, ![button](image.png), to open the **Choose IP Categories** dialog box, as shown in Figure 41.

   The Choose IP Categories dialog box allows the IP designer to select various appropriate categories to help classify the new IP Definition. When the IP Definition is added to the IP Catalog, the IP will be listed under the specified categories.

9. Select the **Serial Interfaces** box under **Communications & Networking** as the IP is a UART interface.

10. Click **OK**.
Adding Documentation to the IP Definition

11. On the left side of the Package IP window, select the **IP File Groups** item to display the IP File Groups panel on the right side, as shown in Figure 42.
The IP File Groups panel provides a listing of the files that will be packaged as part of the IP.

12. Click on the 4 info messages link at the top of the IP File Groups panel to open the Messages window, and review the IP Packager messages as seen in Figure 43.

You can see from the messages that the IP definition does not include any documentation for the IP. This is a check of the IP contents, but will not prevent you from moving forward. However, in this case, you will add the Product Guide to the IP definition, by first defining the Product Guide file group.

13. In the Package IP window, right click in the IP File Groups panel and select Add File Group from the popup menu.
14. In the Add IP File Group dialog box, select **Product Guide** from the Standard File Groups section, as shown in Figure 44.

15. Click **OK** to add the Product Guide file group.

The IP File Groups pane will now be updated with the Product Guide group in the list. There will be a 0 next to the Product Guide name as there are 0 files added to the newly created group.

**Note:** A critical warning will appear when the Product Guide file group is added, noting that the file group is empty.

16. In the IP File Groups panel, right click on the **Product Guide** file group and select **Add Files** from the popup menu.

17. Click **Add Files** in the opened **Add IP Files (Product Guide)** dialog box.

18. Browse to `<Extract_Dir>/lab_3/my_simple_uart/docs`, and select **All Files** in the **Files of type**: entry line.

19. Select **my_simple_uart_product_guide.pdf** and click **OK**.

20. In the **Add IP Files (Product Guide)** dialog box, ensure that **Copy sources into project** is selected, as shown in Figure 45.

![Figure 45: Add Product Guide](image)

21. Click **OK**.

The PDF file of the Product Guide is added to the files defined as part of the IP. The Critical Warning has been resolved, and there is only one remaining message indicating that the IP does not have a README file defined.
Review and Package the IP

22. On the left side of the Package IP window, select the **Review and Package** panel.

   The Review and Package panel provides a Summary of the IP being packaged, as shown in Figure 46.

   ![Figure 46: Review and Package IP](image)

   Based on the settings of the current project, Vivado will not generate an archive for this IP after packaging. This is reflected in the **After Packaging** section of the Review and Package panel of the Package IP window.

23. Click the **settings** link next to After Packaging heading, as seen in Figure 46, to change the setting.

   The settings link will open the **Tools > Project Settings > IP** dialog box.

24. Enable the **Create archive of IP** checkbox under the **Automatic Behavior** section of the Packager tab, as seen in Figure 47.

25. Click **OK** to close the Project Settings dialog box.
After closing the Project Settings dialog box, the Review and Package pane will refresh with the updated settings. The default location of the IP archive is the current project directory. You can click the edit link next to the Archive name and location to change the default.

26. In the Package IP window, click Package IP, to package the current project, add it to the IP catalog, and create a Zip file archive of the IP.

27. After the packaging process completes, close the Vivado project.
Step 4: Validating the New IP

With the new IP definition packaged and added to the IP Catalog, you can validate that the IP works as expected when added to designs. To validate the IP, you will add a new customization of the UART IP to a project, and synthesize the design.

1. From the Vivado IDE Getting Started page, select Manage IP > New IP Location to create a new project.

   ![New Manage IP Project](image)

   **Figure 48: New Manage IP Project**

   **TIP:** You can use either an RTL project or a Manage IP project to validate IP.

2. Click Next at the New IP Location dialog box that opens.

   ![Manage IP Settings](image)

   **Figure 49: Manage IP Settings**

3. In the Manage IP Settings dialog box, set the following options as they appear in Figure 49.
   - Part: `xc7k325tffg900-2`
   - Target language: **Verilog**
   - IP Location: `<Extract_Dir>/lab_3`
4. Click **Finish** to create the Manage IP project.
   
The new Manage IP project opens in the Vivado IDE. The IP Catalog will open automatically in a Manage IP project; however, the IP catalog will not contain the repository used to package the UART IP. You will add the IP repository to the catalog at this time.

5. In the IP Catalog window, right-click and select **IP Settings** from the popup menu.
   
The **Tools > Project Settings > IP** dialog box opens.

6. In the Repository Manager tab, click the **Add Repository** button to open the IP Repositories dialog box.

7. In the IP Repositories dialog box, browse to and select the following location:
   
   `<Extract_Dir>/lab_3/my_simple_uart`

8. Click **Select** to add the selected repository.

   As seen in Figure 50, the added location displays in the **IP Repositories** section, and any packaged IP found in the repositories is listed under the **IP in Selected Repository**. The **My Simple UART** IP definition which you packaged in Step #3 is listed.

![Figure 50: Manage IP Repository](image)

9. Press **OK** to add the IP repository to the IP Catalog and close the dialog box.
TIP: To define a custom IP repository for use across multiple design projects, or for use by multiple designers, you can add the repository to your init.tcl script that is sourced when the Vivado Design Suite is launched. Refer to Vivado Design Tcl Command Reference Guide (UG835) for more information on the init.tcl script.

10. In the IP Catalog, type UART in the Search at the top of the window.

The My Simple UART that is reported under the BaseIP and Serial Interfaces categories that it was previously assigned to during packaging.

![Image of IP Catalog window with UART search results]

Figure 51: Search IP Catalog for UART

11. Select the My Simple UART by clicking on it under either the BaseIP or Serial Interface category.

Examine the Details pane of the IP Catalog window, as shown in Figure 52. Notice the details match the information provided when you packaged the IP.

![Image of My Simple UART details]

Figure 52: My Simple UART - Details

12. Double click on My Simple UART in the IP Catalog to open the Customize IP dialog box, as seen in Figure 53.
13. In the **Customize IP** dialog box, click on **Documentation** and open the **Product Guide**.

14. Click **OK**, accepting the default Component Name and other options.

   The customized IP is added to the current project, and is shown in the IP Sources window. In addition, the Generate Output Products dialog box opens, as shown in Figure 54.

---

**Figure 53: Customize IP – My Simple UART**

**Figure 54: Generate Output Products**
15. Ensure that the **Generate Synthesized Checkpoint (.dcp)** is checked, and click **Generate**.

This will generate the various files required for this IP in the current Manage IP project, and launch an Out-of-Context synthesis run for the IP to create a DCP. Recall this OOC synthesis run will use the OOC XDC file that defines the needed clocks for the standalone IP.

Examine the IP Sources window and the various design and simulation source files that are added to the project. In the Design Runs window, verify that the Out-Of-Context synthesis run was successful.

---

**Conclusion**

In this Lab, you used the **Create and Package IP Wizard** to create an IP definition for the tutorial project, my_simple_uart. You setup the XDC files to support the processing order requirements as well as Out-Of-Context synthesis. You created an archive of the newly packaged IP definition which contains all the source files and documentation for the IP.

You then validated the packaged IP by creating a Managed IP project, and then adding the new IP repository to the IP Catalog. Finally, you created a customization of the IP, and generated a DCP of the IP to validate that the IP definition was complete and included all the necessary files to support using the IP in other designs.
Lab 4: Scripting the Project Mode

In this exercise, you will write a Project Mode Tcl script, creating a new project and adding source RTL and IP definitions. When working in Project Mode, a directory structure is created on disk in order to manage design source files, run results, and track project status. A runs infrastructure is used to manage the automated synthesis and implementation process and to track run status.

In Lab 5: Scripting the Non-Project Mode, you will explore creating Tcl scripts to run the Vivado tools in Non-Project Mode, which does not rely on project files and managed source files. For more information on Project Mode and Non-Project Mode, refer to the Vivado Design Suite User Guide: Design Flows Overview (UG892).

The best way to become familiar with scripting the Project Mode design flow is by first running the design flow interactively in the Vivado IDE, and then referencing the journal file, `vivado.jou`, that the Vivado Design Suite creates. By editing the journal file, you can create a Tcl script that encompasses the entire Project Mode design flow. In this lab, you will build the script by hand. For more information on writing and using Tcl scripts, see the Vivado Design Suite User Guide: Using Tcl Scripting (UG894).

---

**CAUTION!** When copying Tcl commands and script examples from this Tutorial document and pasting them into the Vivado Design Suite Tcl shell, or Tcl Console, the dash “-” character can sometimes be converted to an em-dash “–” which will result in errors when the commands are run.

---

**Step 1: Creating a Project**

1. Invoke a text editor of your choice, such as Emacs, vi, or Notepad; or launch the Text Editor from within the Vivado IDE.

2. Save a new file called `project_run.tcl` in `<Extract_Dir>/lab_4`.

   You will start your script by creating a new project using the `create_project` command. This results in a new project directory being created on disk. However, you want to make sure that the project is created in the right location to find source files referenced by the script.

3. Add the following line to your Tcl script to change to the appropriate directory for this lab:

   ```tcl
   cd <Extract_Dir>/lab_4
   ```

   Now you are ready to create your project.

---

4 Replace the `<Extract_Dir>` variable with the actual path to your tutorial data. For example:

   ```bash
   C:/Data/ug939-design-files/lab_4
   ```
4. Add the following Tcl command to your `project_run.tcl` script:

   ```tcl
   create_project -force -part xc7k70t-fbg484-3 my_project my_project
   ```

   A directory called `my_project` is created, and a project named `my_project` is added to it. The directory and project are created at the location where the script is run. You can specify a different directory name with the `-dir` option of the `create_project` command.

   All the reports and log files, design runs, project state, etc. are stored in the project directory, as well as any source files that you import into the project.

   The target Xilinx part for this project is defined by the `-part xc7k70t` option. If `-part` is not specified, the default part for the Vivado release is used.

   **TIP:** You can use the `set_property` command to change the part at a later time, for example: `set_property part xc7k325tfbg900-2 [current_project]`

   The `-force` option is technically not required, since the project directory should not exist prior to running this script. However, if the project directory does exist, the script will error out unless the `-force` option is specified.

   Refer to the *Vivado Design Suite Tcl Command Reference Guide (UG835)*, or at the Tcl prompt type `help <command_name>`, for more information on the `create_project` command, or any other Tcl command used in this tutorial.

---

**Step 2: Adding RTL Source Files**

For this script, you will be copying all the RTL source files into the local project directory.

Since all the HDL files that you need are located in `<Extract_Dir>/lab_4/sources`, you can add the entire directory directly.

1. Add the following two lines to your script:

   ```tcl
   add_files -scan_for_includes ./sources/HDL
   import_files
   ```

   The `-scan_for_includes` option scans the Verilog source files for any `include` statements, and also adds these referenced files as source files to the project. By default, `include` files are not added to the fileset.

   The specification of `./sources` provides a relative path for locating the source files from where the Tcl script is being run. You will recall that the `project_run.tcl` script is being created in the `<Extract_Dir>/lab_4` directory, so the `./sources` folder is found within that directory.

   The `import_files` command copies the files into the local project directory. When no files are specified, as is the case here, the Vivado Design Suite imports files referenced in the source fileset for the current project.
Step 3: Adding XDC Constraints

For this design, there are two XDC files that are required: top_physical.xdc and top_timing.xdc.

1. Add the following lines to your script to import the XDC files into your project:

   ```tcl
   import_files -fileset constrs_1 \
   ./sources/Constraints/top_timing.xdc \
   ./sources/Constraints/top_physical.xdc
   ```

By default, all XDC files are used in both synthesis and implementation. However, in this case, you will assign the XDC files for use as follows:

- **top_timing.xdc** is used in both synthesis and implementation.
- **top_physical.xdc** is used only in implementation.

2. To disable the use of **top_physical.xdc** during synthesis, add the following line to your script:

   ```tcl
   set_property used_in_synthesis false [get_files top_physical.xdc]
   ```

   This disables the **used_in_synthesis** property on the specified XDC file.

   The property for implementation is **used_in_implementation**, though you will leave that enabled (true).

Step 4: Adding Existing IP

You will also import IP cores into the project. There are four IP cores used in this design:

- **Accumulator**: A legacy CORE Generator IP, with the associated NGC.
- **Block memory generator**: A 2012.4 version of a native Vivado Design Suite IP with no output products generated.
- **FIFO Generator**: Vivado 2013.3 version with all output products, including DCP.
- **Clock Wizard**: Vivado 2013.3 version with no output products.

All of these IPs, with the exception of the Accumulator IP, are native Vivado cores. They have already been customized, and have a Xilinx Core Instance (XCI) file.
In the case of the Accumulator IP, the imported file is a CORE Generator log file (.xco). This is a legacy IP.

1. To import these IP cores into the project, add the following lines to your script:

```bash
import_ip -files \
./sources/IP.Accumulator.Accumulator.xco \
./sources/IP.blk_mem_gen_v7_3_0.xci \
./sources/IP.clk_wiz.clk_wiz_0.xci \
./sources/IP.char_fifo.char_fifo.xci
```

When this line is processed, a warning message(s) for each of the IPs is returned by the Vivado Design Suite:

**WARNING:** [IP_Flow 19-2162] IP 'Accumulator' is locked. Locked reason: IP definition 'Accumulator' for IP 'Accumulator' has a newer major version in the IP Catalog. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.

**WARNING:** [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition 'Block Memory Generator' for IP 'blk_mem_gen_v7_3_0' has a newer major version in the IP Catalog. No useable outputs are available for this IP. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.

**WARNING:** [IP_Flow 19-3197] Could not import any simulation or synthesis outputs for IP 'blk_mem_gen_v7_3_0'

**WARNING:** [IP_Flow 19-3197] Could not import any simulation or synthesis outputs for IP 'clk_wiz_0'

The problem with the Accumulator is that the version in the design does not match the latest version in the IP catalog. However, there is a netlist output product (.ngc) so you can work with the version in the design, or upgrade it to the latest version from the IP catalog.

The blk_mem_gen_v7_3_0 core is also not the latest version in the IP catalog, however, there are no output products to drive synthesis or simulation, so it will have to be upgraded to the latest version. You will upgrade this IP in a subsequent step. If no upgrade path is available, you will have to recreate the IP.

For the clk_wiz_0, no output products were found with the customization (.xci), but the IP is the current version in the IP catalog. You will manually generate the output products for this IP in the next step.

The char_fifo version is current and all output products are present so no warning messages are produced when importing.

**RECOMMENDED:** When there is a major version change to an IP core in the catalog, changes to the IP such as parameter or port name changes, may make upgrading the IP to the latest version require changes to the design.
Step 5: Disabling an IP’s XDC

For this design, you will disable the XDC files that are included with the Clock Wizard IP, so that you can apply the top-level XDC constraints to the clk_wiz_0 module. This IP has not had the output products generated, so you will first generate the synthesis targets, which includes the XDC files.

In a project mode you are not required to generate output products manually. The output products from IP are generated automatically as needed in the design flow, including the generation of a Synthesis design checkpoint. However, since you will be changing a property on XDC files delivered with the clk_wiz IP, you must manually generate the synthesis output products to create the constraints file or files. Also, since we want to disable the use of an IP XDC file and provide them during synthesis of the entire design you need to disable to generation of the synthesis DCP as well.

1. To **Disable** the automatic generation of a Synthesis Design Checkpoint file **Add** the following to your script:
   ```tcl
   set_property generate_synth_checkpoint false [get_files clk_wiz_0.xci]
   
   Now when synthesis is done for the design, a design run for the clk_wiz_0 IP will not automatically be created, and launched. The DCP will not be used for the Clock Wizard IP during synthesis.
   
   2. **Add** the `generate_target` command to your Tcl script:
   ```tcl
   generate_target synthesis [get_files clk_wiz_0.xci]
   
   Multiple output products can be generated by passing a list to the command, such as `synthesis instantiation_template`, or you can generate all the available output products by specifying `all`.
   ```

   **TIP:** To find out what output products an IP supports, use either the `report_property` command on the IP, or `get_property` to get the KNOWN_TARGETS property from the IP. For example (do not add these to your script):
   ```tcl
   report_property [get_ips clk_wiz_0]
   get_property KNOWN_TARGETS [get_ips clk_wiz_0]
   ```

   To disable the XDC constraints delivered with the Clock Wizard, you need the names of the files. You can query the XDC files(s) that are delivered with an IP, by using the `get_files` command with the `-of_objects` and `-filter` options.

   3. To **capture** the **XDC file names** of the Clock Wizard IP in a Tcl variable, add the following lines to your script:
   ```tcl
   set clk_wiz_xdcs [get_files -of_objects [get_files \ clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
   
   This will return the names of the XDC files that are delivered with the Clock Wizard.
   ```

   4. To **disable** the **XDC files**, add this line to your script as well:
   ```tcl
   set_property generate_synth_checkpoint false [get_files clk_wiz_0.xci]
   ```

Designing with IP

UG939 (v 2013.3) November 12, 2013
Step 6: Upgrading an IP

As mentioned earlier, the block memory generator IP in the design has a newer version available in the IP catalog. The IP is locked as a result, because it cannot be re-customized from the IP catalog unless you upgrade it to the latest version. When adding the XCI to a project this warning appears:

WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition 'Block Memory Generator' for IP 'blk_mem_gen_v7_3_0' has a newer major version in the IP Catalog. No useable outputs are available for this IP. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.

In an interactive session this message can be helpful, but in a batch mode script this would not be seen until after synthesis or implementation fails. To anticipate and prevent this, you can use your script to:

- Determine if an IP is locked.
- Check for a newer version of the IP in the catalog.
- Upgrade an IP if it is locked, and a new version is available.

The following sequence shows you how.

1. First, you will check to see if the IP is locked, and store the state in a Tcl variable. Add the following line to your Tcl script:
   ```tcl
   set locked [get_property IS_LOCKED [get_ips blk_mem_gen_v7_3_0]]
   ```

2. Next, you will check to see if there is an upgrade available in the IP catalog, and store that information in a Tcl variable as well. Add the following line to your Tcl script:
   ```tcl
   set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
   ```
   This will return the VLNV (Vendor:Library:Name:Version) identifier of the upgrade, if there is one available. If there is no upgrade, the variable will contain an empty string (""). In the case of the blk_mem_gen_v7_3_0 IP, there is an upgrade available.

3. Now you can check the stored Tcl variables, $locked and $upgrade, to see if the IP is locked AND if there is an upgrade version for the IP. If so, you can upgrade the IP. Add the following lines to your Tcl script:
   ```tcl
   if {$locked && $upgrade != ""} {
     upgrade_ip [get_ips blk_mem_gen_v7_3_0]
   }
   ```
   This will result in upgrading the block memory generator IP from the current version in the design, to the latest version in the IP catalog.
The Accumulator core is legacy IP that was created with CORE Generator, rather than native IP created in the Vivado Design Suite. The IP has all the necessary output products, for instantiating the HDL module into a design, for synthesis, and for simulation. So it can be used in its current form though it will not have any timing constraints.

However, you should upgrade legacy IP to native Vivado IP whenever possible. This will ensure you have the latest updates and fixes for an IP, and any XDC constraints delivered with it.

4. Following the steps in 1-3 above, add a sequence of commands to your Tcl script to check if the Accumulator IP is locked, has an available upgrade, and upgrade the IP if so.

TIP: You could create a foreach loop in your Tcl script that will perform these checks for all IPs in a design:

```
foreach design_IP [get_ips] {
  add code here...
}
```

Refer to the Vivado Design Suite User Guide: Using Tcl Scripting (UG894) for more information.

---

**Step 7: Setting up Design Runs for IP**

Now that all the IP have been upgraded to the latest version you can optionally setup and launch out-of-context synthesis runs for the IP. You already configured the Clocking Wizard IP to not use the out-of-context flow in Step 5. The FIFO Generator IP already had all the output products, including the DCP, generated when it was imported. The Block Memory Generator and Accumulator IP however have not had any output products generated yet. In a project flow if the IP is current then all the output products, including the DCP, will be generated automatically. However, if you want you can script this. To demonstrate this we will create a design run for the Accumulator IP and launch it.

One reason to manually create and launch design runs for IP in a project is to have the out-of-context synthesis processes run in parallel. If you let Vivado create the runs for IP which do not have a DCP they will be launched serially.

1. For a Synthesis Design Checkpoint to be generated an IP Design Run must be first created. Add the following to your Tcl script:

   ```
   create_ip_run [get_ips Accumulator]
   ```

   A new run is now created for the IP named `<IP_name>_synth_1`

2. The IP design run is launched using the `launch_runs` command. Add the following to the script:

   ```
   launch_runs [get_runs Accumulator_synth_1]
   ```

   The run is now launched and when completed a Synthesis Design Checkpoint will be added to the project. If you do not launch the IP run it will automatically be launched when
Step 8: Launching Synthesis and Implementation

The project is now ready for synthesis and implementation. The Vivado Design Suite automatically generates the necessary output products for the various IP in your project, as needed. You do not need to manually generate the output products unless you want to make changes to the generated output products prior to using them in synthesis, simulation, or implementation.

In the Project Mode, the Vivado Design Suite manages the details of synthesis and implementation runs, using run strategies and maintaining the state of the design. Therefore, you will use the launch_runs command to run synthesis and implementation in project-based designs.

1. **Add** the following line to your Tcl script:

   ```tcl
   launch_runs synth_1
   ```

   By default, a synthesis run called `synth_1` is created for every project. You can also manually create new runs using the `create_run` command, and configure run properties using the `set_property` command. See the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) for more information on creating and configuring runs.

   After the synthesis run has completed, you can launch an implementation run. However, since the implementation run is dependent on the completion of the synthesis run, you must use the `wait_on_run` command to hold your Tcl script until synthesis is complete.

2. **Add** these two lines to your script:

   ```tcl
   wait_on_run synth_1
   launch_runs impl_1 -to_step write_bitstream
   ```

   When the synthesis run, `synth_1`, completes, the implementation run, `impl_1`, begins.

   Implementation is a multi-step process that begins with netlist optimization, runs through placement and routing, and can even include generating the bitstream for the Xilinx FPGA. The `-to_step` option that you added to your Tcl script, indicates that implementation should include generating the bitstream for the device. By default, implementation does not...
include that step. Refer to the *Vivado Design Suite User Guide: Implementation* (UG904) for more information.

---

**TIP:** Alternatively, you can use the `write_bitstream` command; this requires that you open the implementation run first using the `open_run` command.

---

Just as implementation needed to wait on synthesis to complete, you will want your Tcl script to wait on implementation to complete before generating any reports, or exiting.

3. **Add** the `wait_on_run` command to your Tcl script, to wait for the implementation run to complete:

   ```tcl
   wait_on_run impl_1
   ```

   The script will wait until the implementation run completes before continuing.
Step 9: Running the Script

You are now ready to run the script. Your script should be similar to the following:

```
#Step 1: Create Project
cd C:/Data/ug939-design-files/lab_4
create_project -force -part xc7k70t-fbg484-3 my_project my_project

#Step 2: Adding RTL Files
add_files -scan_for_includes ./sources/HDL
import_files

#Step 3: Adding XDC Files
import_files -fileset constrs_1 \ 
   ./sources/Constraints/top_timing.xdc \ 
   ./sources/Constraints/top_physical.xdc
set_property used_in_synthesis false [get_files top_physical.xdc]

#Step 4: Importing IP
import_ip {./sources/IP/Accumulator/Accumulator.xco 
   ./sources/IP/blk_mem/blk_mem_gen_v7_3_0.xci 
   ./sources/IP/clk_wiz/clk_wiz_0.xci \ 
   ./sources/IP/char_fifo/char_fifo.xci}

#Step 5: Disable XDC
set_property generate_synth_checkpoints false [get_files clk_wiz_0.xci]
generate_target synthesis [get_files clk_wiz_0.xci]
set clk_wiz_xdc [get_files -of_objects [get_files \ 
   clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
set_property is_enabled false [get_files $clk_wiz_xdc]

#Step 6: Upgrade IP
set locked [get_property IS_LOCKED [get_ips blk_mem_gen_v7_3_0]]
set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
if {$upgrade != "" && $locked} {
   upgrade_ip [get_ips blk_mem_gen_v7_3_0]

   set locked [get_property IS_LOCKED [get_ips Accumulator]]
   set upgrade [get_property UPGRADE_VERSIONS [get_ips Accumulator]]
   if {$upgrade != "" && $locked} {
      upgrade_ip [get_ips Accumulator]

   #Step 7: Launch IP run
   create_ip_run [get_ips Accumulator]
   launch_runs [get_runs Accumulator_synth_1]

   #Step 8: Launching Synthesis and Implementation
   launch_runs synth_1
   wait_on_run synth_1
   launch_runs -to_step write_bitstream impl_1
   wait_on_run impl_1
```
You can run the `project_run.tcl` script in Vivado Design Suite batch mode or Tcl mode.

- Batch mode will run the sourced script, and then automatically exit the tool after the script has finished processing.
- Tcl mode will run the sourced script, and return to the Tcl command prompt when finished.

On Linux,
1. Change to the directory where the lab materials are stored:
   ```
   cd <Extract_Dir>/lab_4
   ```
2. Launch the Vivado Design Suite Tcl shell, and source a Tcl script to create the tutorial design:
   ```
   vivado -mode tcl -source project_run.tcl
   ```

On Windows,
1. Launch the Vivado Design Suite Tcl shell:
   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x Tcl Shell
   ```
2. In the Tcl shell, change to the directory where the lab materials are stored:
   ```
   Vivado% cd <Extract_Dir>/lab_4
   ```
3. Source the Tcl script to create the design:
   ```
   Vivado% source project_run.tcl
   ```

After the sourced script has completed, the Tcl shell displays the `Vivado%` prompt.

![Figure 55: Vivado Tcl Mode and Sourced Tcl Script](image)

---

Your Vivado Design Suite installation may called something different than Xilinx Design Tools on the Start menu.
IMPORTANT: If your Tcl script has an error in it, the script will halt execution at the point of the error. You will need to fix the error, and re-source the Tcl script as needed. If you are running in Tcl mode, you may need to close the current project with `close_project`, or exit the Vivado tool with `exit` to source the Tcl script again.

After the script executes, a project directory structure is created, default log and reports are generated, a synthesized netlist is produced, design is fully implemented, and a bitstream is created. You can use the Vivado IDE to examine the design, the various reports, and to perform analysis on the design.

To open the GUI from the Tcl prompt type `start_gui`.

**Conclusion**

Creating a design project does not require the use of the Vivado IDE. The benefits of the Project Mode, such as automatic report generation, design runs infrastructure, and managed source files, are available from a Tcl script. The result of the script is a design project, which you can open in the Vivado IDE for interactive timing analysis and design exploration.

Specific areas covered in this lab are:

- Creating a project and adding HDL sources.
- Adding IP sources to a project, both native XCI files and legacy XCO files.
- Generating IP Output Products.
- Disabling IP Output Products, such as an XDC file.
- Disabling the creation of a Synthesis Design Checkpoint (DCP) file.
- Creating and launching IP design runs.
- Querying an IP’s properties.
- Updating an IP to the latest version.
- Launching synthesis and implementation runs, and generating a bitstream.
Lab 5: Scripting the Non-Project Mode

In Lab 4: Scripting the Project Mode, you created a Tcl script to run the Vivado Design Suite in Project Mode. In this lab, you will create a Tcl script to run the Vivado tools in Non-Project Mode. Many of the commands used in this lab are the same commands used in Lab 4. The main difference between Project Mode and Non-Project Mode is that there is no project file or directory structure created on disk. Instead, Vivado Design Suite manages the design directly in an in-memory database.

In Non-Project Mode, you do not have a project file to add source file references to, or a project directory to manage source files. In Non-Project Mode, you read source files into the Vivado Design Suite to create the in-memory design. In addition, there is no design runs infrastructure to store run strategies and results. Instead, you directly call the various commands to run the different stages of synthesis and implementation. Unlike Project Mode, you must manually write out design checkpoints, netlists, and reports. These items are not created automatically for you in Non-Project Mode. For more information on Project Mode and Non-Project Mode, refer to the Vivado Design Suite User Guide: Design Flows Overview (UG892).

When working with IP in Non-Project Mode you must manually generate output products, including synthesis Design Checkpoints if desired.

---

**CAUTION!** When copying Tcl commands and script examples from this Tutorial document and pasting them into the Vivado Design Suite Tcl shell, or Tcl Console, the dash “-“ character can sometimes be converted to an em-dash “—” which will result in errors when the commands are run.

---

**Step 1: Reading Design Source Files**

1. **Invoke** a text editor of your choice, such as Emacs, vi, or Notepad; or launch the Text Editor from within the Vivado IDE.

2. **Save** a new file called `nonproject_run.tcl` in `<Extract_Dir>/lab_5`.

   In Lab 4, you started your project by creating a project; here you will begin by creating an in-memory design, and reading source files. However, you want to first make sure that the Tcl script is in the right location to find source files referenced by the script.

3. **Add** the following line to your Tcl script to change to the appropriate directory for this lab:

   ```tcl
   cd <Extract_Dir>6/lab_5
   ```

---

6 Replace the `<Extract_Dir>` variable with the actual path to your tutorial data. For example:

   C:/Data/ug939-design-files/lab_5
Even though this is a non-project flow script, it is advised that an in-memory project be created to set the target part for the design. This target part will be used for all IP that are added to the design, otherwise the default part for the Vivado Design Suite is used. Additionally, setting the target part removes the need to specify a part when synthesizing the top-level design, or any IP for out-of-context synthesis.

4. **Add** the following to your script to create the in-memory project:

   ```
create_project -in_memory -part xc7k70t-fbg484-3
   ```

   Creating an in-memory project does not result in a project being created on disk.

---

**RECOMMENDED:** *In Non-Project Mode, there is no project part unless you create an in-memory project. If you do not create an in-memory project, the IP output products are generated using the default part of the Vivado Design Suite release. This default part might not be the intended target part specified by the `synth_design` command, and can result in mismatched synthesis results between the IP and the top-level design in Non-Project Mode designs. See Lab 2: Creating and Managing Reusable IP for more information on managing IP customizations.*

---

Now you are ready to read the source files for the design. In Project Mode, you use commands such as `add_files` and `import_files` to add source files to the project. In Non-Project Mode, you can use `add_files`, which will call the appropriate lower-level command, but it is more typical to directly read the file. This is similar to an ASIC tool flow. For this lab, you are working with Verilog source files and will use `read_verilog` to read them.

5. **Add** the following line to your script to read all the Verilog source for this project:

   ```
read_verilog [glob ./sources/HDL/*.v]
   ```

---

**TIP:** The `glob` command is a built-in Tcl command that creates a list out of the specified objects. Alternatively, you can make a Tcl list to pass to `read_verilog`, or use a separate `read_verilog` command for each file.
**Step 2: Adding Existing IP**

You will also read the following IP cores into the design:

- **Accumulator**: A legacy CORE Generator IP, with no output products or NGC.
- **FIFO Generator**: 2013.3 version native IP with all output products, including a DCP.
- **Clock Wizard**: 2013.3 version native IP with no output products.
- **Block memory generator**: A 2012.4 version of a native Vivado Design Suite IP with no output products generated.

All of these IPs, with the exception of the Accumulator IP, are native Vivado cores. They have already been customized, and have Xilinx Core Instance (XCI) files. The Accumulator IP is a legacy CORE Generator log file (XCO).

The FIFO Generator IP already has all required output products available, and can be read and used directly from its current location. Whenever you create an IP customization, you should always generate all available output products.

1. **Add** the following to your Tcl script to read the FIFO Generator IP, including all the output products that are present:
   ```
   read_ip ./sources/IP/char_fifo/char_fifo.xci
   ```

2. **Add** the following to your Tcl script to create a local directory, with sub-directories for the block memory, accumulator and the clock wizard IP:
   ```
   file mkdir IP/blk_mem
   file mkdir IP/clk_wiz
   file mkdir IP/accum
   ```

3. **Add** the following to your Tcl script to copy the needed XCI files from the current IP repository into the local directory:
   ```
   file copy -force ./sources/IP/blk_mem/blk_mem_gen_v7_3_0.xci ./IP/blk_mem
   file copy -force ./sources/IP/clk_wiz/clk_wiz_0.xci ./IP/clk_wiz
   file copy -force ./sources/IP/Accumulator/Accumulator.xco ./IP/accum
   ```

   The -force option causes the file to be overwritten if it already exists. Without -force an error could be returned and the script will quit.

For the clocking wizard IP, you will need to generate the output products before you can synthesize the design. In Non-Project Mode, the Vivado Design Suite will not automatically generate output products, and errors will be encountered if you do not do this prior to launching synthesis.

Generating output products results in files and directories being written to the location the IP XCI files are read from. In a managed IP repository, these locations may be read-only or under revision control. In this case, you would copy the needed XCI files to a local directory before reading them into the in-memory design.
Step 3: Disabling XDC Files

**IMPORTANT:** It is important to have each XCI file stored in its own directory. When output products are generated, they are written to the same directory as the XCI file. If IP files are written to the same directory, it is possible that output products from different IPs could overwrite each other.

4. **Add** these lines to your Tcl script to read in the needed XCI files:

   ```tcl
   read_ip ./IP/blk_mem/blk_mem_gen_v7_3_0.xci
   read_ip ./IP/clk_wiz/clk_wiz_0.xci
   read_ip ./IP/accum/Acculator.xco
   ```

   The specified XCI files are read into the in-memory design.

   Unlike in Lab 4, the warnings related to locked IP do not display when the IP are processed into the design using the `read_ip` command. In Project Mode, the Vivado Design Suite performs checks as an IP is added to a project, resulting in the warning messages seen in Lab 4, Step 4: Adding Existing IP. In Non-Project Mode, the checks are only performed when the IP are processed during synthesis.

---

**Step 3: Disabling XDC Files**

As in Project Mode, if an IP delivers XDC constraints, they are automatically processed and added to the in-memory design. For this design, you will disable the XDC files that are included with the Clock Wizard IP as you have constraints in the top-level XDC file that you will apply instead. However, the Clock Wizard IP has not had the output products generated, so you will first need to generate the output products, which include the XDC files.

When generating the output products for an included XCI file IP, you must decide whether to use an out-of-context flow, including the creation of a synthesis Design Checkpoint (DCP), or to let the IP be synthesized as part of the top-level design.

Since for the Clocking Wizard we want to disable the use of an XDC we need to also configure the IP not to use a DCP.

1. **Add** the following to your script to configure the IP to not use a Synthesis Design Checkpoint:

   ```tcl
   set_property generate_synth_checkpoint false [get_files clk_wiz_0.xci]
   ```

   Now when synthesis is done of the top level the RTL output products will be used and a DCP will not be expected. Unlike in a Project Flow, output products are not automatically generated as needed, and need to be manually created.

2. **Add** the `generate_target` command to your Tcl script:

   ```tcl
   generate_target all [get_ips clk_wiz_0]
   ```

   Since you copied the XCI file from the source IP repository into a local directory, the output products will be written to the local directory.
**TIP:** To find out what output products an IP supports, use either the `report_property` command on the IP, or `get_property` to get the `KNOWN_TARGETS` property from the IP. For example (do not add these to your script):

```
report_property [get_ips clk_wiz_0]
get_property KNOWN_TARGETS [get_ips clk_wiz_0]
```

Multiple output products can be generated by passing a list to the command, such as `{synthesis instantiation_template}`.

To disable the XDC files delivered with the Clock Wizard, you need the names of the files. You can query the XDC file(s) that are delivered with an IP, by using the `get_files` command with the `-of_objects` and `-filter` options.

3. To **capture** the XDC file names of the Clock Wizard IP in a Tcl variable, add the following lines to your script:

```
set clk_wiz_xdc [get_files -of_objects \[get_files clk_wiz_0.xci\] -filter {FILE_TYPE == XDC}]
```

This will return the names of the XDC files delivered with the Clock Wizard.

4. To **disable** the XDC files, add this line to your script as well:

```
set_property is_enabled false [get_files $clk_wiz_xdc]
```

The XDC files delivered with `clk_wiz` IP will be disabled when you run your script.

To check what XDC files are evaluated, and in what order, you can use the `report_compile_order` command with the `-constraints` option.

---

**Step 4: Upgrading IP**

If you attempt to run synthesis at this time, you will see this warning for the Block Memory Generator IP:

```
WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition 'Block Memory Generator' for IP 'blk_mem_gen_v7_3_0' has a newer major version in the IP Catalog. No useable outputs are available for this IP. Please select 'Report IP Status' from the 'Tools/Report' menu.
```

The Block Memory Generator is locked because it is not the most recent version of the IP. It also does not have any output products and so must be upgraded before output products can be generated.

Similar messages would be seen for the Accumulator. Both the Block Memory Generator and Accumulator have updated versions in the Xilinx IP Catalog. In an interactive design session, these messages can be helpful; but in a batch mode Tcl script these messages would not be seen until after synthesis or implementation fails.
To anticipate and prevent this, you can use your script to:

- Determine if an IP is locked.
- Check for a newer version of the IP in the catalog.
- Upgrade an IP if it is locked, and a new version is available.
- Generate the output products for the IP.

You will do this for the `blk_mem_gen_v7_3_0` IP using following sequence:

1. First, you will **check** to see if the **IP is locked**, and store the state in a Tcl variable. Add the following line to your Tcl script:
   ```tcl
   set locked [get_property IS_LOCKED [get_ips blk_mem_gen_v7_3_0]]
   ```

2. Next, you will **check** to see if there is an **upgrade available** in the IP catalog, and store that information in a Tcl variable as well. Add the following line to your Tcl script:
   ```tcl
   set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
   ```
   This will return the VLNV (Vendor:Library:Name:Version) identifier of the upgrade, if there is one available. If there is no upgrade, the variable will contain an empty string ("""). In the case of the `blk_mem_gen_v7_3_0` IP, there is an upgrade available.

3. Now you can **check** the stored Tcl variables, `$locked` and `$upgrade`, to see if the IP is locked AND if there is an upgrade version for the IP. If so, you can upgrade the IP. Add the following lines to your Tcl script:
   ```tcl
   if {$locked && $upgrade != ""} {
      upgrade_ip [get_ips blk_mem_gen_v7_3_0]
   }
   ```
   This will result in upgrading the block memory generator IP from the current version in the design, to the latest version in the IP catalog.

4. Now that the IP is current, add the following to your script:
   ```tcl
   generate_target all [get_ips blk_mem_gen_v7_3_0]
   ```
   The Accumulator core is legacy IP created with CORE Generator, rather than native IP created in the Vivado Design Suite. If the IP had the necessary output products, for instantiating the HDL module into a design, for implementation, and for simulation it could be used in its current form.

   However, you should upgrade legacy IP to native Vivado IP whenever possible. This will ensure you have the latest updates and fixes for an IP, and any XDC constraints delivered with it.

5. Following the steps in 1-4 above, add a sequence of commands to your Tcl script to check if the Accumulator IP is locked, has an available upgrade, upgrade the IP if so, and generate output products for it.
TIP: You could create a foreach loop in your Tcl script that will perform these checks for all IPs in a design:

```tcl
foreach design_IP [get_ips] {
    add code here...
}
```

Refer to the Vivado Design Suite User Guide: Using Tcl Scripting (UG894) for more information.

---

**Step 5: Creating DCP for IP**

The default flow for Vivado is to use a Synthesis Design Checkpoint for IP. Typically when referencing IP in a Non-Project Flow you would have created the IP customizations using a Manage IP project. At that point, you would have created the output products and decided to disable DCP use or generated the DCP. This is the case with the FIFO Generator IP (char_fifo); it has all the output products generated including a DCP. The other three IP all consist of just an XCI or XCO file and thus you need to configure the synthesis option. Either create a DCP for the IP or configure it to be synthesized with the top-level logic.

In Step 3, you configured the Clocking Wizard to not use a DCP by setting a property on the IP XCI file. At this point, you will generate a DCP for the Accumulator and Block Memory Generator IP.

1. **Add** the following two lines to your script to create DCP for the Accumulator and Block Memory Generator IP:
   ```tcl
   synth_ip [get_ips Accumulator]
   synth_ip [get_ips blk_mem_gen_v7_3_0]
   ```
   This results in a DCP file being created and stored in the directory containing the IP XCI file.

---

**Step 6: Running Synthesis**

For this design, there are two XDC files that are required, `top_timing.xdc` and `top_physical.xdc`. One of the XDC files is used in both synthesis and implementation (`top_timing.xdc`) while the other is used only during implementation (`top_physical.xdc`).

At this point in your Tcl script, you want to read the XDC file, using `read_xdc`, which is used in both synthesis and implementation.

1. **Add** the following to your Tcl script:
   ```tcl
   read_xdc ./sources/Constraints/top_timing.xdc
   ```
   The design is now ready for synthesis.
In Non-Project Mode, unlike Project Mode, there are no design runs to launch, and no runs infrastructure managing the strategies used and the state of the design. You will manually launch the various stages of synthesis and implementation.

2. For synthesis, you use the `synth_design` command. Add the following to your Tcl script:

   ```tcl
   synth_design -top sys_integration_top
   ```

   Since we created an in-memory project and set the target part that is not needed. However, you must provide the top-level module name with the `synth_design` command.

   The various Verilog files read into the in-memory design in Step 1: Reading Design Source Files do not reference other files via an `include` statement. If they did, you would need to define the `include` search directories with the `-include_dirs` option.

   After synthesis, you should generate a design checkpoint to save the results. This way you can restore the synthesized design at any time without running synthesis again.

   3. Add the following `write_checkpoint` command to your Tcl script:

   ```tcl
   write_checkpoint -force post_synth.dcp
   ```

   The `-force` option is used to overwrite the checkpoint file if it already exists.

   4. You can also generate any needed reports at this point, such as a post-synthesis timing summary report. Add the following line to your Tcl script:

   ```tcl
   report_timing_summary -file timing_syn.rpt
   ```

   This command creates a comprehensive timing report for the design and writes the report to a file.

---

**Step 7: Running Implementation**

With synthesis completed, you are now ready to script implementation. There are many steps to implementation, in both Project Mode and Non-Project Mode. However, in Project Mode, you select a design run strategy that controls all of the various steps, and launch that run. In Non-Project Mode, without a design run, you must determine your implementation strategy by manually running each step of implementation, and selecting the Tcl command options to use at each step. You can also choose to skip some steps, such as logic optimization, power optimization, and physical synthesis.

For this lab, you will run the following steps:

- Logic optimization: `opt_design`
- Placement: `place_design`
- Physical synthesis: `phys_opt_design`
- Routing: `route_design`
- Bitstream generation: `write_bitstream`
For a complete description of each of these steps, see the Vivado Design Suite User Guide: Implementation (UG904). Between each of these steps, you can generate reports, and write checkpoints to save the design in different stages of implementation.

Before launching implementation, you must read the design constraints that are only used in implementation. The XDC file, top_physical.xdc, contains physical constraints that are used in implementation, but do not apply to synthesis. In this case, these constraints could have been read into the in-memory design prior to synthesis, because synthesis will simply ignore them. However, this file may also contain different timing constraints, not to be used in synthesis, that must be read in after synthesis and just prior to implementation.

1. **Add** the following line to your Tcl script:

   read_xdc ./sources/Constraints/top_physical.xdc

2. **Add** optimization and placement commands to your Tcl script:

   opt_design
   place_design
   write_checkpoint -force post_place.dcp
   report_timing -file timing_place.rpt

   After placement completes, your script writes a post-placement checkpoint and create a custom timing report, which provides a detailed timing report for the single worst timing path in the design.

3. **Add** physical synthesis and routing commands to your Tcl script:

   phys_opt_design
   route_design
   write_checkpoint -force post_route.dcp
   report_timing_summary -file timing_summary

   After routing completes, your script writes a post-routing design checkpoint and creates a timing summary report.

4. Finally, write out a bitstream by **adding**:

   write_bitstream -force sys_integration_top.bit

   This is the complete Non-Project Mode design flow for implementing a design from RTL source files, including designing with IP, through bitstream generation.
Step 8: Running the Script

You are now ready to run the Tcl script. Your script should be similar to the following:

```
#Step 1: Reading RTL
cd C:/Data/ug939-design-files/lab_5
create_project -in_memory -part xc7k70t-fbg484-3
read_verilog [glob ./sources/HDL/*.v]

#Step 2: Adding Existing IP
read_ip ./sources/IP/char_fifo/char_fifo.xci
file mkdir IP/blk_mem
file mkdir IP/clk_wiz
file mkdir IP/accum
file copy -force ./sources/IP/blk_mem/blk_mem_gen_v7_3_0.xci ./IP/blk_mem
file copy -force ./sources/IP/clk_wiz/clk_wiz_0.xci ./IP/clk_wiz
file copy -force ./sources/IP/Accumulator/Accumulator.xco ./IP/accum
read_ip ./IP/blk_mem/blk_mem_gen_v7_3_0.xci
read_ip ./IP/clk_wiz/clk_wiz_0.xci
read_ip ./IP/accum/Accumulator.xco

#Step 3: Disable DCP and XDC
set_property generate_synth_checkpoint false [get_files clk_wiz_0.xci]
generate_target all [get_ips clk_wiz_0]
set clk_wiz_xdc [get_files -of_objects [get_files clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
set_property is_enabled false [get_files $clk_wiz_xdc]

#Step 4: Upgrade IP
set locked [get_property IS_LOCKED [get_ips blk_mem_gen_v7_3_0]]
set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
if {$upgrade != "" && $locked} {
    upgrade_ip [get_ips blk_mem_gen_v7_3_0]
    generate_target all [get_ips blk_mem_gen_v7_3_0]
}
set locked [get_property IS_LOCKED [get_ips Accumulator]]
set upgrade [get_property UPGRADE_VERSIONS [get_ips Accumulator]]
if {$upgrade != "" && $locked} {
    upgrade_ip [get_ips Accumulator]
    generate_target all [get_ips Accumulator]
}

#Step 5: Creating DCP for IP
synth_ip [get_ips Accumulator]
synth_ip [get_ips blk_mem_gen_v7_3_0]

#Step 6: Running Synthesis
read_xdc ./sources/Constraints/top_timing.xdc
synth_design -top sys_integration_top
write_checkpoint -force post_synth.dcp
report_timing_summary -file timing_syn.rpt

#Step 7: Running Implementation
read_xdc ./sources/Constraints/top_physical.xdc
opt_design
place_design
write_checkpoint -force post_place.dcp
```
Step 8: Running the Script

Launch Vivado

You can run the nonproject_run.tcl script in Vivado Design Suite batch mode or Tcl mode.

- Batch mode will run the sourced script, and then automatically exit the tool after the script has finished processing.

- Tcl mode will run the sourced script, and return to the Tcl command prompt when finished.

On Linux,

1. Change to the directory where the lab materials are stored:
   ```
   cd <Extract_Dir>/lab_5
   ```
2. Launch the Vivado Design Suite Tcl shell, and source a Tcl script to create the tutorial design:
   ```
   vivado -mode tcl -source nonproject_run.tcl
   ```

On Windows,

1. Launch the Vivado Design Suite Tcl shell:
   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.x > Vivado 2013.x Tcl Shell
   ```
2. In the Tcl shell, change to the directory where the lab materials are stored:
   ```
   Vivado% cd <Extract_Dir>/lab_5
   ```
3. Source the Tcl script to create the design:
   ```
   Vivado% source nonproject_run.tcl
   ```

After the sourced script has completed, the Tcl shell displays the **Vivado%** prompt.

---

**IMPORTANT:** If your Tcl script has an error in it, the script will halt execution at the point of the error. You will need to fix the error, and re-source the Tcl script as needed. If you are running in Tcl mode, you may need to close the current project with `close_project`, or exit the Vivado tool with `exit` to source the Tcl script again.

---

7 Your Vivado Design Suite installation may called something different than Xilinx Design Tools on the **Start** menu.
Figure 56: Vivado Tcl Mode and Sourced Tcl Script

Running the script results in the creation of a directory called “IP”. Output products for the various IPs used in the design are written to this directory. Reports, design checkpoints, and a bitstream for the design are also written to disk.

You can open the design can in the Vivado IDE to perform further analysis. To open the Vivado IDE from the Tcl prompt type start_gui.

Conclusion

Using the Non-Project Mode gives you the greatest control over the Vivado Design Suite, and gives you access to advanced features that may not be available in Project Mode. However, Non-Project Mode also requires manually managing source files, updating the design when source files have changed, and manually planning and running synthesis and implementation strategies. Specific areas covered in this lab are:

- Reading in Verilog source files and reading IP sources.
- Generating required IP output products for synthesis and implementation, and disabling them as needed.
- Querying an IP’s upgradability, and updating to a newer version when appropriate.
- Creating Synthesis Design Checkpoints for IP.
- Manually running synthesis and individual steps of implementation.
- Generating custom reports