Vivado Design Suite Tutorial

Designing with IP
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>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 ......................................................................................................................................... 6  
  Software Requirements ................................................................................................................................. 7  
  Hardware Requirements ............................................................................................................................... 7  
  Locating and Preparing the Tutorial Design Files ...................................................................................... 7  
Lab 1: Designing with the IP Catalog .................................................................................................................. 8  
  Step 1: Opening the Project ......................................................................................................................... 8  
    Launch Vivado ............................................................................................................................................ 8  
    Open the Project ....................................................................................................................................... 9  
  Step 2: Customizing the FIFO Generator ................................................................................................. 10  
  Step 3: Generating IP Output Products ................................................................................................... 15  
  Step 4: Instantiating the IP Template into a Design ................................................................................. 17  
  Conclusion ...................................................................................................................................................... 20  
Lab 2: Generating an Out-of-Context Netlist ................................................................................................. 21  
  Step 1: Opening the Project ....................................................................................................................... 21  
  Step 2: Setting IP as Out-of-Context ........................................................................................................ 23  
  Step 3: Synthesizing the Out-Of-Context IP ............................................................................................ 25  
  Conclusion ...................................................................................................................................................... 27  
Lab 3: Creating IP with the Manage IP Flow .................................................................................................. 28  
  Step 1: Starting a Manage IP session ....................................................................................................... 28  
  Step 2: Customizing the FIFO Generator .................................................................................................. 30  
  Step 3: Generating Output Products ....................................................................................................... 31  
  Step 4: Generating a Netlist .................................................................................................................... 34
Step 5: Using the Netlist in Implementation ............................................................... 37
Conclusion .................................................................................................................. 38

Lab 4: Packaging an IP for Reuse ............................................................................. 39

Step 1: Packaging an IP Project ............................................................................... 39
  Launch Vivado .......................................................................................................... 39
  Packaging IP ............................................................................................................. 39
Step 2: Identifying and Documenting the IP ............................................................ 41
Step 3: Archiving IP .................................................................................................. 45
Step 4: Adding IP to the Catalog ............................................................................. 45
Step 5: Validate the New IP ..................................................................................... 47
Conclusion .................................................................................................................. 51

Lab 5: Scripting the Project Mode ............................................................................. 52

Step 1: Creating a Project ........................................................................................ 52
Step 2: Adding RTL Source Files .......................................................................... 53
Step 3: Adding XDC Constraints ............................................................................ 54
Step 4: Adding Existing IP ...................................................................................... 54
Step 5: Disabling an IP’s XDC ................................................................................ 56
Step 6: Upgrading an IP .......................................................................................... 57
Step 7: Launching Synthesis and Implementation .................................................. 58
Step 8: Running the Script ........................................................................................ 60
Conclusion .................................................................................................................. 62

Lab 6: Scripting the Non-Project Mode ................................................................... 63

Step 1: Reading RTL Source Files .......................................................................... 63
Step 2: Adding Existing IP ....................................................................................... 64
Step 3: Disabling XDC Files .................................................................................... 65
Step 4: Upgrading IP ................................................................................................. 66
Step 5: Running Synthesis ....................................................................................... 68
Step 6: Running Implementation ............................................................................. 69
Step 7: Running the Script ........................................................................................ 71
Conclusion .................................................................................................................. 73
Lab 7: Creating an IP Integrator Subsystem Design

Step 1: Opening the Project
Step 2: Opening the IP Integrator Design Canvas
Step 3: Customizing IP
Step 4: Completing the Subsystem Design
Step 5: Generating IP Output Products
Step 6: Instantiating IP into the Design
Conclusion

Lab 8: Converting Legacy EDK IP

Step 1: Managing IP
Step 2: Running the Package IP Wizard
Conclusion
Designing with IP

Overview

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:** Continuing from Lab 1, you will mark a FIFO core for out-of-context (OOC) synthesis and implementation to save time in subsequent design iterations.
- **Lab 3:** You will learn how to verify IP as a standalone design. Create a project, include an IP from the IP catalog as the top-level source; customize and verify the IP; then netlist the IP for inclusion in another design.
- Lab 4: 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 5: Write and run a Tcl script using the Vivado Design Suite Project Mode.

- Lab 6: Write and run a Tcl script using the Vivado Design Suite Non-Project Mode.

- Lab 7: You will complete an IP subsystem design using the IP integrator feature of the Vivado Design Suite to instantiate, configure, and connect multiple IP cores.

**IMPORTANT:** The Vivado IP integrator environment is a licensed early access feature in the 2013.1 release. Please contact your Field Applications Engineer to obtain a license.

- Lab 8: Walk through the conversion of a Pcore into Vivado Design Suite native IP for use in IP Integrator.

---

### Software Requirements

This tutorial requires that the 2013.1 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

You can find the design files for this tutorial under *Vivado Design Suite -2013.1 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.

There are separate project files and sources for each of the labs in this tutorial.

**RECOMMENDED:** You will modify the tutorial design data while working through this tutorial. You should use a new copy of the `ug939-2013-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 2013.1 > Vivado 2013.1

2. As an alternative, click the Vivado 2013.1 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**.

![Figure 3: Open Project](image)

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 different than Xilinx Design Tools on the Start menu.
Step 2: Customizing the FIFO Generator

Figure 4: Default Layout

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 cores displayed in the catalog.
3. From the **Memories & Storage Elements > FIFOs** group select **FIFO Generator**, as shown in Figure 5.

   ![Figure 5: Xilinx IP Catalog – FIFO Core](image)

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

   The FIFO Generator core opens in the Customize IP window, 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 core.

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. **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 core, open the change log to review the revision history of the core, or open an Internet browser to navigate to the IP core webpage, or view any Answer Records related to the core.
Step 2: Customizing the FIFO Generator

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 core.

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.

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.

   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 core should look like **Figure 8**. For this configuration you are using one 18K BRAM.

![Summary of FIFO Generator Core](image)

**Figure 8: Summary of FIFO Generator Core**

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

   The FIFO now appears in the Sources view. Because the IP core is not instantiated into your design, the core displays at the same level as the top-level wave_gen module in the Hierarchy tab. In the Libraries and Compile Order tabs, it appears in the Unreferenced folder.

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

19. **Examine** the Tcl Console in the Vivado IDE, and review the **Tcl commands** used to add the FIFO Generator core to your project.
Step 3: Generating IP Output Products

The customiz
ed FIFO Generator core includes an instantiation template, char_fifo.veo, and synthesis constraints and VHDL entity and architecture definition. Since the target language for the project is set to Verilog, the instantiation template is a VEO file defining the Verilog module.

1. In the **IP Sources** tab of the Sources window, **examine the default output products** produced for the customized FIFO Generator core, as shown in Figure 10.

---

Figure 9: Tcl Console

- 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 core.

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

---

Figure 10: FIFO Generator Sources
Step 3: Generating IP Output Products

You can change what output products are generated by default by going to Tools > Project Settings > IP, selecting the Generation tab, and checking which output products are generated when customizing IP.

2. In the IP Sources tab select char_fifo, right click and select Generate Output Products. The Manage Output Products dialog box displays the output products available for the selected IP core, and the current state of those outputs, as shown in Figure 11.

You can click on the ‘+‘ to expand the specific output product categories, and get a description and a list of the selected cores that support that output product. Not all cores are available with Examples for instance.

Notice that the Instantiation Template and Synthesis output products are already generated. The Vivado Design Suite generated these output products when the core was added to the current project, due to the IP Project Settings, as previously discussed.

3. Under the Examples output, set the Action to Do Not Generate.

4. Under the Simulation output, set the Action to Generate, if not already set.

5. Click OK.

The simulation output products are generated and added to the Sources window, under the IP Sources tab.
Step 4: Instantiating the IP Template into a Design

You will now instantiate the IP into the design by copying, or cutting and pasting the Verilog module definition from the Instantiation Template, into the appropriate Verilog source file in your project.

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.

2. Scroll down to line 56 or 57 of the template file, and select and copy the module instantiation text, as shown in Figure 12.

![Figure 12: FIFO Core Generator - Instantiation Template](image)

Next, you will copy the template module instantiation 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 an instantiation template, 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:

   ```vhdl
   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 (保存) to save the changes to the `wave_gen.v` file.

   Notice that the Hierarchy, Libraries, and Compile Order tabs are all updated to indicate that the IP has been instanced into the design. You can elaborate the RTL design to verify that the FIFO module has been properly instantiated, and your connections are correct.

8. In the RTL Analysis section of the **Flow Navigator** select **Open Elaborated Design**.

   An RTL Netlist window opens, and a schematic of the RTL design is displayed.

9. In the RTL Netlist window, select the `char_fifo` cell.

   The cell is cross-selected in the schematic.
Conclusion

Congratulations! You have successfully customized the FIFO Generator IP core, and added it to the design. Either close the project and exit Vivado tool, or proceed to the next lab, which will continue and demonstrate the out-of-context flow.

In this exercise, you learned how to select and customize an IP from the IP catalog, and how to instantiate the customized IP into an HDL design. You can do this interactively within the Vivado IDE, or via Tcl scripting.
Lab 2: Generating an Out-of-Context Netlist

In this exercise, you will learn how to mark an IP in a design to be synthesized out of context of the rest of the project. This way the IP is synthesized only once, saving time while developing the rest of the design.

You can either continue from where you left off in Lab 1, or start from a completed design. To continue from Lab 1: Designing with the IP Catalog, start this lab at Step 2: Setting IP as Out-of-Context.

---

**Step 1: Opening the Project**

On Linux,

1. Change to the directory where the lab materials are stored:
   ```bash
   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 2013.1 > Vivado 2013.1
   ```

2. As an alternative, click the **Vivado 2013.1** 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.

3. To start from a completed design, select **Open Example Project** from the Getting Started page and select **Wave (HDL) Small IP Project** from the drop down list seen in Figure 16.

![Open Example Project](image)

**Figure 16: Open Example Project**

The example Wave design will open.

---

2 Your Vivado Design Suite installation may be called something different than **Xilinx Design Tools** on the **Start** menu.
Step 1: Opening the Project

4. If you are prompted to save the read-only project as shown in Figure 17, click the **Save Project ...** button.

   ![Figure 17: Save Read-Only Project](image)

5. If you are not prompted to save the read-only project, use **File > Save Project** As to save the project to the `<Extract_Dir>` folder.

   The **Save Project As** dialog box will appear.

   ![Figure 18: Save Project As...](image)

6. Enter the following values:
   - **Project name**: `project_wave_gen_ip`
   - **Project location**: browse to the `<Extract_Dir>/lab2` directory
   - Select the **Create project subdirectory** box

7. Press **OK**.

   The Vivado IDE saves the `project_wave_gen_ip` project to the specified location, and opens the project.
Step 2: Setting IP as Out-of-Context

1. In the Hierarchy tab of the Sources window, expand the top level module, `wave_gen`. Notice the `char_fifo_i0` instance of the FIFO Generator IP that has been instanced inside the project.

2. Right click on the `char_fifo_i0` to open the popup menu, and select `Set As Out-Of-Context Module`.

The Set As Out-Of-Context Module dialog box opens.

3. **Name** the block file set which is created for the IP, `char_fifo`.

4. **Select** the **Use auto-generated blackbox stub for top level synthesis** checkbox.

   This will cause the insertion of an HDL stub for the character FIFO. During synthesis, a black box will be inferred for the IP. After synthesis of the top level, the IP netlist will be linked back into the design.
Step 2: Setting IP as Out-of-Context

5. Press **OK** to set the character FIFO as an out-of-context module.

A number of things change in the project as a result of setting the IP as out-of-context.

![Set As Out-Of-Context Module](image)

**Figure 20: Out-of-Context Synthesis Runs**

- An **Out-of-Context Module Runs** folder is created in the Designs Runs window, with a new run called `char_fifo_synth_1`.
- In the Hierarchy tab of the Sources window, the `char_fifo_i0` instance has a new symbol denoting it has been marked as out-of-context.
- In the Libraries and Compile Order tabs of the Sources window, new Block Sources folders contain source files for the out-of-context modules, which the Vivado Design Suite now treats as a separate design inside the current project.
Step 3: Synthesizing the Out-Of-Context IP

Synthesis runs for any out-of-context modules must be launched manually, separate from the synthesis run for the overall design. The active synthesis run in the current project is synth_1, and selecting Run Synthesis from the Flow Navigator will launch this top-level synthesis run.

RECOMMENDED: You should synthesize any out-of-context modules first, so the results are available when you synthesize the top level and open the synthesized design, otherwise errors will be generated.

1. In the Design Runs window right click on the out-of-context synthesis run, char_fifo_synth_1, and select Launch Runs.

Figure 21: Block Sources for Out-of-Context IP

Figure 22: Launch Out-of-Context Module Synthesis
2. The **Launch Selected Runs** window will appear.

3. Press **OK** to launch the run.

   After the synthesis of the Character FIFO is completed, you can launch implementation of the top-level design. The `char_fifo` module will not be included in the top-level synthesis, since it is marked as out-of-context.

   Vivado synthesis reports a black box for the `char_fifo` module during top-level synthesis.

4. From the Flow Navigator, press **Run Synthesis** to launch the top-level synthesis.

   Alternatively, from the Design Runs window, you can right click on `synth_1` and select **Launch Runs**.

5. From the Synthesis Completed dialog box, select **Open Synthesized Design**, and click **OK**.

6. Right click in the **Log Window** and select **Show Find**, or press **Ctrl-F**.

7. In the **Find:** field enter “black box”.

   You will see the following message:

   ```
   INFO: [Designutils 20-1022] Could not resolve non-primitive black box cell 'char_fifo' instantiated in module 'char_fifo_i0'.
   Resolution: File names need to match cell names: an EDIF definition will be found in char_fifo.edf; a HDL definition may be placed in any Verilog/VHDL file.
   ```

   The IP is identified as a black box because only an HDL stub is present. When the synthesized design is opened the black box is resolved.

   Scrolling up in the Tcl Console you will see the following message:

   ```
   INFO: [Project 1-454] Reading design checkpoint 'C:/Data/ug939-design-files/lab_2/project_wave_gen_ip/project_wave_gen_ip.runs/char_fifo_synth_1/char_fifo.dcp' for cell 'char_fifo_i0'
   ```

   The out-of-context synthesis run for the `char_fifo` module results in a design checkpoint (.dcp) being created in the runs directory of the IP.

8. Look in the following folder for the results of the out-of-context synthesis run:

   `<Extract_Dir>/lab_2/project_wave_gen_ip/project_wave_gen_ip.runs/char_fifo_synth_1`

   The file is called `char_fifo.dcp`. The checkpoint file can be used in other projects. It encapsulates the netlist and the constraints for the IP module.
Conclusion

In this exercise, you learned how to select an IP in a design and mark it for out-of-context synthesis. Doing this will mean that the out-of-context module must be synthesized separately, and synthesis of the top-level design will not include the out-of-context module. The HDL version of the IP module is replaced with the synthesized netlist, which is linked in by the Vivado Design Suite when the top-level synthesized design is opened. The out-of-context module is synthesized once, and used many times, even in other designs.

You can continue to the next lab, or exit the Vivado IDE.

To continue to Lab 3:

1. **File > Close Project.**
2. Say **No** if prompted to Save Design.
   
   You may resume this tutorial at Lab 3 when you are ready.
3. To Exit the Vivado IDE and resume Lab 3 another time, **File > Exit.**
Lab 3: Creating IP with the Manage IP Flow

To simplify revision control, and to support the use of custom IPs across multiple projects and designs, you should manage and store the 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. Having all the generated output products available also preserves that customized version of the IP core 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 core as a standalone source using the Manage IP flow in the Vivado Design Suite. This flow allows you to browse the IP catalog and customize IP for use in either a Project or a Non-Project design flow. You can create a repository of the customized IPs for use in your design(s), managed and maintained outside of a Vivado Project. You can reference these custom IPs in new projects and designs, to simulate, synthesize, and implement the IP core as part of the design.

---

**Step 1: Starting a Manage IP session**

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 2013.1 > Vivado 2013.1
   ```
2. As an alternative, click the *Vivado 2013.1* 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.
3. On the Getting Started page, click the *Manage IP* link.

---

3 Your Vivado Design Suite installation may be called something different than *Xilinx Design Tools* on the *Start* menu.
Step 1: Starting a Manage IP session

A drop down menu displays, as seen in Figure 23, allowing you to:

- Open the IP Catalog
- Open a previously customized IP
- Open a recent customization location

4. Select **Open IP Catalog**.

The Manage IP Initial Settings dialog box displays.

5. Change the **Default IP location** to be `<Extract_Dir>/lab_3` as shown in Figure 24.

6. Press **OK** to proceed. The IP Catalog is displayed.
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.
3. The Customize IP dialog box opens as shown in Figure 25.

![Figure 25: 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.

4. At the top of the Customize IP dialog box, change the Component Name to `char_fifo` from the default name of `fifo_generator_0`.

5. 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.
6. 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.

7. 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 26. For this configuration you will see you are using one 18K BRAM.

   ![Summary of FIFO Generator Core](image)

   **Figure 26: Summary of FIFO Generator Core**

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

---

**Step 3: Generating Output Products**

![IP Sources](image)

**Figure 27: IP Sources**
Step 3: Generating Output Products

The **Sources** window appears next to the IP Catalog. The FIFO Generator core, `char_fifo`, appears in the Sources window, as seen in Figure 27.

1. In the **IP Sources** tab, notice the default **output products** delivered for the core.
2. **Right-click** in the IP catalog to open the popup menu, and select the **IP Settings** command.
   - The IP Project Settings dialog box opens, as seen in Figure 28. The Default IP Output Products lists the output products that are generated when the core is customized. The FIFO Generator core delivers an instantiation template and synthesis output products by default. Because the target language is set to Verilog, the instantiation template is a VEO file.

   ![Figure 28: IP Project Settings](image)

3. **Click Cancel** to close the **Project Settings** dialog box without making any changes.
4. In the **IP Sources** tab select `char_fifo`, right click and select **Generate Output Products**.
The Manage Output Products dialog box displays the output products available for the selected IP core, and the current state of those outputs, as shown in Figure 29. You can click on the ‘+’ to expand specific output product categories to see a description of the output product.

If multiple IP cores are selected prior to using Generate Output Products, the Manage Output Products dialog box also lists which of the selected cores support a specific output product.

**TIP:** Not all IP cores support all available output products.

Notice in Figure 29 that the Instantiation Template and Synthesis output products have already been generated. The Vivado Design Suite generated those output products by default when the IP was customized.

![Figure 29: Manage Output Products](image)

5. Click **OK** to generate the Examples and Simulation output products.

6. Look in the `<Extract_Dir>/lab_3` directory to examine the `char_fifo` folder.

   This directory contains the XCI file, which has all the customization information for the IP, as well as all the output products generated.

   When creating a repository of customized IPs using the Managed IP flow, Xilinx recommends that you generate all the available output products for each IP. A specific release of the Vivado Design Suite only supports one 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 output products.

**RECOMMENDED:** Only one version of an IP is supported in a given release of the Vivado tools. To preserve older versions of an IP, all output products must be available in your custom IP repository.

You can repeat these steps to customize additional IPs. Each IP customization is displayed in the IP Sources window and a directory will be created with the name of the component in the default location specified when starting a Managed IP session.

These customized IPs can be referenced from either a project or non-project flow. In a project flow it is recommend you do not copy sources into the project, instead reference them from your IP repository.

### Step 4: Generating a Netlist

Up to this point in the lab, you have not created a Vivado project. The IP customization (XCI) and output products that you generated are the only files stored to disk. However, for flows where a 3rd party Synthesis tool is being used, a black box module or a synthesized netlist will be required to use the Vivado IP in an HDL design.

Files from the Vivado IP that you need, include:

- An instantiation template to instance the IP in the HDL design.
- A Verilog stub module providing port directions. For VHDL, use the COMPONENT declaration from the .vho instantiation template.
- Any XDC files that the IP delivers.

Once the IP has been synthesized standalone, a few files will need to be generated:

- An EDIF or Verilog Netlist,
- Structural Simulation Netlist, which is optional - if no multi-language simulator used.

Create a design project to contain the IP and to launch synthesis.

1. Use File > Save Project As to create the design project.

   Using Project Mode simplifies the creation of netlists for multiple IPs as opposed to working 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).

---

4 Do not use an Out-of-Context constraints file, `<component_name>_ooc.xdc`, that may be delivered with the IP. These constraints are only used during bottom-up-synthesis, when the IP is defined as the top-level module. The OOC constraints define top-level clocks that are not needed when the IP is used in a design that already defines the top-level clocks.
2. In the **Save Project As** window use the following settings:
   - Project Name: `project_char_fifo`
   - Project Location: `<Extract_Dir>/lab_3`
   - Enable **Create project subdirectory**

   ![Figure 30: Save Project As](image)

   The `project_char_fifo` Vivado project opens in the default view layout.

3. In the Hierarchy tab of the Sources window, **right click** on the `char_fifo` IP and select **Set As Out-Of-Content Module** from the popup menu.

   The Set As Out-of-Context Module dialog box opens, as shown in Figure 31, letting you define the top-level name of the IP module, and generate a black box stub for the IP.

   ![Figure 31: Set as Out-of-Context](image)

4. Click **OK**. 
Marking an IP as out-of-context does a number of things. It creates out-of-context synthesis and implementation runs, and configures the IP to be synthesized with no IO buffer insertion. If there were multiple IPs in the Manage IP session, you could select all the IPs at once and set them all to be out of context. This would result in a run being created for each, and you could launch the synthesis jobs in parallel.

5. **Notice** in the Hierarchy tab of the Sources window, the **IP icon** has changed to indicate it is now an out-of-context block.

6. Looking in the **Design Runs** window, you will notice a new folder named **Out-Of-Context Module Runs** under which is `char_fifo_synth_1` for the IP.

![Figure 32: Out-of-Context Design Runs](image)

7. In the Design Runs view, **right click** on the `char_fifo_synth_1` run and select **Launch Runs** to start synthesis.

---

**TIP:** You could also use the **Make Active** command in the Design Runs window popup menu to make the selected run active, and launch synthesis from the Flow Navigator.

---

After synthesis completes, you will need to open the synthesized design, and write a Verilog or EDIF netlist to support the use of the IP in third party synthesis tools.

8. In the Design Runs window, **right click** on the `char_fifo_synth_1` run and select **Open Synthesized Design** from the popup menu.

9. Write out either an EDIF or Verilog netlist by typing in the Tcl Console:

   ```tcl
   write_edif <Extract_Dir>/lab_3/char_fifo/char_fifo.edif
   write_verilog <Extract_Dir>/lab_3/char_fifo/char_fifo.v
   ```

   This is the netlist that you will use when implementing the entire design.

   Notice that you are writing the netlist to the `char_fifo` folder, which is the repository of the output products generated from this IP core in **Step 3: Generating Output Products**. This keeps all the required outputs for the IP core in the same location to simplify source management and version control.
Step 5: Using the Netlist in Implementation

For a black box to be inferred correctly during synthesis when using Verilog, a module “stub” file is needed which gives the directions of the ports. Include this file when running synthesis, either with Vivado or a 3rd party synthesis tool.

10. To create the stub type the following command in the Tcl Console:

```
write_verilog -mode port <Extract_Dir>/lab_3/char_fifo/char_fifo_stub.v
```

For netlist simulations, you also need to create a structural simulation for the IP, which will include the encrypted simulation sources.

11. In the Tcl Console type one of the following commands:

```
write_verilog -mode funcsim <Extract_Dir>/lab_3/char_fifo/char_fifo_sim.v
write_vhdl -mode funcsim <Extract_Dir>/lab_3/char_fifo/char_fifo_sim.vhd
```

Again, all of the outputs for the IP are directed to the same IP folder.

---

Step 5: Using the Netlist in Implementation

After the netlists have been created for the IP, the module can be linked with other netlists and HDL modules for implementation as a single design. However, to correctly constrain the IP for implementation in Vivado, any constraints (XDC) that the IP delivers must also be read into the hierarchy of the design.

Note that the IP constraints were written in the context of the IP as the top-level of the design. The constraints will need to be properly scoped to work correctly when applying them to the IP within the hierarchy of a design.

In the default, top-down synthesis design flow, the Vivado Design Suite automatically applies constraints to the IP in the design context. In the Out-of-Context design flow, you must manually read the IP’s XDC in the proper context.

Most Vivado IPs come with a single XDC file. This XDC may have dependencies on an outside clock. Before reading the IP XDC, ensure that all necessary top-level clocks have been properly created.

Some IPs come with an XDC named `<component_name>_clocks.xdc` which creates clock objects. Since this XDC creates a clock that could be used by other IPs or by the user, it should be read in before other constraint files.
The following is a sample Tcl script that reads a top-level design netlist, and the char_fifo netlist, then reads the various design and IP constraints into the proper design context:

```tcl
# Read top-level EDIF and IP netlist
read_edif ./sources/top.edf
read_edif ./sources/char_fifo.edf

# read top level constraints
read_xdc ./constraints/top.xdc
# read the IP XDC and scope to the IP cell reference
read_xdc -cells inst -ref char_fifo ./constraints/char_fifo.xdc

# Link the netlists
link_design -top top -part xc7v2000tfhg1761-2L
# Continue with the other stages of implementation
```

For this example, there could be many instances of the FIFO Generator IP, char_fifo, in the design. The single `read_xdc` command will find all instances of the IP module in the design, and apply the constraints to the `inst` cell within it.

---

TIP: The `-cells` option is specified relative to the reference module, `-ref`, when both options are used. Refer to the `read_xdc` command in the Vivado Design Suite Tcl Command Reference Guide (UG835) for more information.

---

When IP is instanced in a design, Vivado automatically creates a wrapper around the IP instance. In this example `inst` is the IP instance name specified by `-cells`, inside the module name specified by `-ref`. Providing both the `-ref` and the `-cells` arguments allows the `read_xdc` command to properly scope the IP constraints.

Alternatively, you can just provide the absolute hierarchy and cell names using the `-cells` option. Refer to the Vivado Design Suite User Guide: Using Constraints (UG903) for more details on the scoping of constraints.

---

**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. The Manage IP flow does not require a design project. 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 create a netlist to use the custom IP in a Vivado Project Mode or Non-Project Mode design flow, or for use with a third-party synthesis tool.

To perform behavioral simulation with a custom IP, you would use the IP in a design and do structural simulation after completing synthesis. To use 3rd party simulators you will need to query the HDL files required for simulation and the library they are associated with. See the Vivado Design Suite User Guide: Designing with IP (UG896), which provides Tcl scripts to help with this.
Lab 4: Packaging an IP for Reuse

In Lab #4, you will create a new IP core using the 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 core to the IP Catalog. Finally, you will verify the new IP through synthesis and implementation in a separate design project.

Step 1: Packaging an IP Project

Launch Vivado

On Linux,
1. Change to the directory where the lab materials are stored:
   ```
   cd <Extract_Dir>/lab_4
   ```
2. Launch the Vivado IDE: `vivado`

On Windows,
1. Launch the Vivado Design Suite IDE:
   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.1 > Vivado 2013.1
   ```
2. As an alternative, click the Vivado 2013.1 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.
3. Click Open Project, and browse to: `<Extract_Dir>/lab_4/my_complex_mult`
4. Select the `my_complex_mult.xpr` project and click OK.
   The project opens in the Vivado IDE.

Packaging IP

1. Select the Tools > Package IP command from the pull-down menu.
   The Package New IP dialog box opens which informs you that you are starting the process of creating IP using the source files and information from your current project.

----------------------

5 Your Vivado Design Suite installation may be called something different than Xilinx Design Tools on the Start menu.
2. Click **Next**.

   The Choose IP Source Location dialog box opens to let you specify the source of the IP you would like to package. You can choose to package the current project, or a specific directory. In this case, you will package the current project.

3. **Set** the IP Definition Location as `<Extract_Dir>/lab_4/source_files`.

4. Click **Next**.

   The Begin IP Creation page displays, showing what work will be performed during the IP packaging flow.

5. Click **Finish**.

   In the Sources window a new Design Source folder is created called IP-XACT which contains a component.xml file. This is an IP-XACT formatted file representing the IP definition, including information such as the files, HDL ports, and configuration parameters.

   The Package IP wizard collects the available information in the my_complex_mult project, and publishes the results in the IP Packager Summary. The Package IP window also opens in the graphical window area, next to the Project Summary.

6. **Examine** the contents of the **IP Packager Summary**, as seen in Figure 34.

   The IP Packager Summary dialog box lists various information that was extracted from the current project and added to the IP.

7. Click **OK** to close the IP Packager Summary dialog box.
Step 2: Identifying and Documenting the IP

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

1. In the Package IP window, fill in the IP Identification pane with the following information:
   - Vendor: my_company
   - Display Name: My Complex Multiplier
   - Description: This is a great little complex multiplier
   - Vendor Display Name: My Company
   - Company Url: http://www.my_company.com

   Figure 35 shows how this should look.
Step 2: Identifying and Documenting the IP

2. For Categories, select the browse button (“…”) to open the Choose IP Categories dialog box, as seen in Figure 36.

   The Choose IP Categories dialog box lets you select various categories appropriate to help identify the new IP core.

   When your IP definition is added to the IP Catalog, it will be listed under the specified categories.

3. Press Cancel to close the Choose IP Categories dialog box.
Step 2: Identifying and Documenting the IP

Figure 36: Choose IP Categories

4. In the Package IP window, click **Review and Package**, as shown in **Figure 37**.

Figure 37: Package IP - Review and Package
Notice the Possible Missing Information section that identifies missing elements such as documentation and design examples. You will add a PDF Product Guide at this time.

5. In the Package IP window, select **IP File Groups** in the left frame.

6. Under the **Standard** section, right-click on the **Product Guide** to open the popup menu, and select **Add Files (Product Guide)**, as shown in Figure 38.

![Figure 38: Package IP - Add Product Guide](image)

The Add IP Files (Product Guide) dialog box opens, to let you add files to the IP package.

7. Click **Add Files**... to open the **Add IP Files** dialog box.

8. Navigate to `<Extract_Dir>/lab_4/source_files/doc`

9. In the **Files of type**: drop down menu field, select **All Files**.

10. **Select** the file `my_complex_mult_product_guide.pdf`.

11. Click **OK** to close the **Add IP Files** dialog Box.

12. Click **OK** again to add the selected file as a Product Guide to the IP definition.

   In the Package IP window, you can expand the Product Guide category, to see that the selected document is added to the package.
Step 3: Archiving IP

With the definition of the IP completed, including added documentation in the form of the product guide, you are now ready to create a zip file to archive the IP core, and to add it to the IP catalog.

1. In the Package IP window, on the left side pane, click Review and Package.
2. Click the Archive IP button to open the Package IP dialog box.
3. Verify it matches Figure 39.

![Figure 39: Package IP - Create Archive](image)

- **Archive name:** my_company_user_my_complex_mult_1.0
- **Archive location:** `<Extract_Dir>/lab_4/source_files`

4. Click OK to close the Package IP dialog box, and generate the archive file.
5. **Examine** the `<Extract_Dir>/lab_4/source_files` folder to make sure that the IP archive file was created.

   The archive will have the extension .zip.

   The IP archive file contains everything required for this IP definition, including source HDL and documentation, etc. Everything that was added to the definition that was created.

Step 4: Adding IP to the Catalog

A typical use case is to put all packaged IP definitions into a central repository, and add that repository to the IP Catalog.

1. In the Review and Package pane of the Package IP window, click the Add to Catalog button.

   The Deploy IP to Catalog window opens as shown in Figure 40. This dialog box displays a message that explains the process of adding IP repositories to the IP Catalog. This includes adding the path to the packaged IP to the IP repository, and updating the IP catalog with any IPs found in the new repository.
2. Click **OK** to add the IP to the catalog.

![Deploy IP to Catalog](image1)

**Figure 40: Deploy IP to Catalog**

3. From the **Flow Navigator**, select **IP Catalog** to open the updated catalog.

4. In the IP Catalog, browse the **BaseIP** category to verify that **My Complex Multiplier** was added to the IP catalog for this project.

![Updated IP Catalog](image2)

**Figure 41: Updated IP Catalog**

The packaged IP is now available for use in new projects.
Step 5: Validate the New IP

With the new IP core packaged and added to the IP catalog, you can now validate that the IP works as expected, and can be used in other designs. To validate the IP, you will customize the IP, instantiate it into a design, and synthesize and implement that design.

1. From the main menu, select the **File > Close Project** command to close the project you used to create the IP definition.

2. Create a new project by selecting **Create New Project** from the Vivado IDE Getting Started page.

3. Press **Next** at the New Project information window that appears.

4. In the Project Name dialog box, set the following options, as seen in Figure 43:
   - Project name: **test_IP**
   - Project location: `<Extract_Dir>/lab_4`
   - Select the **Create project subdirectory** box.

5. Press **Next** to open the Project Type dialog box.

6. In the Project Type dialog box, select **RTL Project**, and select the **Do not specify sources at this time** checkbox.
7. Press **Next** to open the Default Part dialog box.

8. In the Default Part dialog box, press **Next** to accept the default part.

9. In the New Project Summary dialog box, press **Finish** to create the *test_IP* project.

   The new project opens in the Vivado IDE. Now you must add the packaged IP definition into the IP catalog for this project.

10. From the **Flow Navigator**, select **IP Catalog** to open the catalog.

11. Right click in the IP Catalog and select **IP Settings**.

   The Project Settings dialog box opens. This dialog box can also be accessed from the main menu **Tools > Project Settings** command.

12. Click the **Add Repository button**.

13. In the IP Repositories dialog box, browse to: `<Extract_Dir>/lab_4/source_files`.

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

   In the **Repository Manager** the added path will be displayed in the IP Repositories, as seen in Figure 44, and the definition of any packaged IPs that are found in that repository are listed in the **IP in Selected Repository** pane.

---

*Figure 44: IP Repository Manager*
In this case, the Vivado tool finds and lists the **My Complex Multiplier** IP core, which you packaged earlier in this lab.

15. Press **OK** to add the IP repository to the IP Catalog.

**RECOMMENDED:** To use a custom IP repository across multiple design projects, or for use by multiple designers, you can add the repository when the Vivado tool launches by adding it to your `init.tcl` script. Refer to Vivado Design Tcl Command Reference Guide (**UG835**) for more information on the `init.tcl` script.

16. In the IP Catalog, **browse** the **BaseIP** category to verify that **My Complex Multiplier** is listed.

17. **Select** the **My Complex Multiplier** core and examine the Details pane, as seen in **Figure 45**.

   Notice the details match the information you provided when packaging the IP.

![Figure 45: Details of the New IP](image)

18. Double click on **My Complex Multiplier** in the IP catalog to start the customization process.

   The **Customize IP** dialog box opens to allow customization of the IP as shown in Figure 46.

19. In the Customize IP dialog box, click on **Documentation** and select **Product Guide**.

   The PDF Product Guide you added to the IP definition is displayed.
Step 5: Validate the New IP

Figure 46: Customize My Complex Multiplier

20. In the Customize IP dialog box, click **OK**, accepting the default Component name and customization options.

The customized IP is added to the current project, and is set as the top-level module in the Sources window.
21. From the Flow Navigator, click Run Synthesis.

22. From the Synthesis Completed dialog box, click Run Implementation.

23. In the Implementation Completed dialog box, click Open Implemented Design.

   The implemented IP is opened. Examine the results to verify that implementation completes successfully.

Conclusion

In this exercise, you invoked the Package IP wizard to create an IP definition for the tutorial project, my_complex_mult. You created a zip archive of the newly packaged IP definition, containing all the source files for the IP.

You then validated the packaged IP by creating a new project, and adding the packaged IP repository to the IP Catalog. Finally, you created a customization of the IP, added it to the new project, and ran synthesis and implementation to validate that the IP definition was complete, and included all the necessary files to support using the IP in other designs.
Lab 5: 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 6 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_5`.

   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_5
```

   Now you are ready to create your project.

---

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

`C:/Data/ug939-design-files/lab_5`
Step 2: Adding RTL Source Files

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:** Use the `set_property` command to change the part at a later time, for example:

```tcl
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_5/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_5` 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
   ```

   **TIP:** The "\" character in the preceding text is used to split Tcl commands across multiple lines. This is useful for making scripts that are more readable and easier to edit.

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** – 2012.4 version native IP with required output products for synthesis.
- **Clock Wizard** – 2013.1 version native IP 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:

```
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 for each of the IPs is produced:

WARNING: [IP_Flow 19-2162] IP 'Accumulator' is locked. Locked reason: IP definition for IP 'Accumulator' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP.

WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition for IP 'blk_mem_gen_v7_3_0' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP. This IP does not have the required outputs for use in synthesis.

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'

WARNING: [IP_Flow 19-2162] IP 'char_fifo' is locked. Locked reason: IP definition for IP 'char_fifo' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP. This IP does not have the required outputs for use in simulation.

WARNING: [IP_Flow 19-3198] Could not import any simulation outputs for IP 'char_fifo'

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 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 generate the output products for this IP in the next step.

The char_fifo version in the design does not match the latest version in the IP catalog. The IP also does not have an output product for simulation, but does have the required targets for synthesis. However, there is a major version change with this IP, from 9.3 to 10.0, so there may be reasons why you would not want to upgrade the IP, such as parameter or port name changes. For this tutorial, you will not upgrade the char_fifo IP.
**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 file.

Normally, you are not required to generate output products manually. The output products from IP are generated automatically as needed in the design flow. However, since you will be changing a property on an XDC file delivered with the clk_wiz IP, you must manually generate the synthesis output products to create the constraints file or files.

1. **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 `generate_target synthesis [get_files clk_wiz_0.xci instantiation_template]`.

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.

2. **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_xdc [get_files -of_objects [get_files clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
   ``
   This will return the names of the XDC file that are delivered with the Clock Wizard.

3. **To disable** the **XDC file**, add this line to your script as well:
   ```tcl
   set_property is_enabled false [get_files $clk_wiz_xdc]
   ``
   The XDC constraint(s) delivered with clk_wiz IP are disabled when you run your script.

   To check what XDC constraints are evaluated, and in what order, you can use the `report_compile_order` command with the `-constraints` option.
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 for IP 'blk_mem_gen_v7_3_0' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP. This IP does not have the required outputs for use in synthesis.`

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.
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:

```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 7: 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 \texttt{-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 \textit{Vivado Design Suite User Guide: Implementation (UG904)} for more information.

\begin{flushleft}
\textbf{TIP:} Alternatively, you can use the \texttt{write\_bitstream} command; this requires that you open the implementation run first using the \texttt{open\_run} command.
\end{flushleft}

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 \texttt{wait\_on\_run} command to your Tcl script, to wait for the implementation run to complete:

\begin{verbatim}
wait\_on\_run impl\_1
\end{verbatim}

The script will wait until the implementation run completes before continuing.
Step 8: 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_5
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 -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}

#Step 5: Disable XDC
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: Launching Synthesis and Implementation
launch_runs synth_1
wait_on_run synth_1
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
You can run the script in the Vivado Design Suite batch mode, or Tcl mode.

- Batch mode will automatically exit the tool after the script has finished processing.
- Tcl mode will return to the Vivado Design Suite Tcl prompt when the script has finished.

**TIP:** You can also source the Tcl script from within the Vivado IDE using the Tools > Run Tcl Script command.

1. Open a Linux shell, or a Windows Command Prompt window.

2. From the command line, go to: `<Extract_Dir>/ug939-design-files/lab_5`.
   
   ```
   > vivado -mode batch -source project_run.tcl
   
   -or-  
   > vivado -mode tcl -source project_run.tcl
   ```

**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.

3. The script results in a project directory structure being created, default log and report generation, a synthesized netlist, a fully implemented design, and a bitstream. The project can be opened in the Vivado IDE GUI at a later point:

   ```
   > vivado project_wave_gen/project_run.xpr
   ```

   To open the GUI after running in Tcl mode type `start_gui` at the Tcl Console.

---

7 This presumes that the Vivado Design Suite is properly installed, and can be found in your $PATH environment. Refer to the Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973) for more information.
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.
- Querying an IP’s properties.
- Updating an IP to the latest version.
- Launching synthesis and implementation runs, and generating a bitstream.
Lab 6: Scripting the Non-Project Mode

In Lab 5 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 #5. The main difference between Project Mode and Non-Project Mode is that there is no project file or directory created on disk, though there is one created in memory.

In Non-Project Mode, you do not have a project file to add source file references to, or a project directory to import source files into. 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. Also, 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.


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 RTL 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_6`.

   In Lab #5 you started your project by creating a project, here you will begin by reading source files to create an in-memory design. 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>/lab_6
   ```

   Now you are ready to read the source files.

---

8 Replace the `<Extract_Dir>` variable with the actual path to your tutorial data. For example: `C:/Data/ug939-design-files/lab_6`
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.

4. Add the following line to your script:
   ```
   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 the associated NGC.
- **Block memory generator**: A 2012.4 version of a native Vivado Design Suite IP with no output products generated.
- **FIFO Generator**: 2012.4 version native IP with required output products for synthesis.
- **Clock Wizard**: 2013.1 version native IP 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 Xilinx Core Instance (XCI) files. The Accumulator IP is a legacy CORE Generator log file (XCO).

The Accumulator and FIFO Generator IPs already have all required output products available, and can be read and used directly from their current location.

1. Place the following two lines into your script:
   ```
   read_ip ./sources/IP/Accumulator/Accumulator.xco
   read_ip ./sources/IP/char_fifo/char_fifo.xci
   ```

   For the block memory generator and clocking wizard IPs, you will need to generate the output products before you can synthesize the design. The tool will automatically do this if the IP used in the design is the current version from the IP catalog.

   Generating these 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.

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

3. Add the following to your Tcl script to copy the needed XCI files from the current IP repository into the local directory:
   ```tcl
   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
   ```
   The -force option causes the file to be overwritten if it exists, otherwise an error is returned and the script would quit.

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
   ```
   The XCI files are now read into the in memory design.

   Unlike in Lab #5, the warnings related to locked IPs will 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 the IP is added to the project, resulting in the warning messages seen in Step 4: Adding Existing IP of Lab #5. In Non-Project Mode the checks are done during synthesis, because that is when the IPs are processed.

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 file that is 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 generate the synthesis targets, which the XDC file is included with.

Normally, output products are generated as needed, and do not need to be manually created. However, since you are going to disable the XDC file delivered as an output product, you will need to manually generate the output product for the Clock Wizard.

1. Add the generate_target command to your Tcl script:
   ```tcl
   generate_target synthesis [get_files clk_wiz_0.xci]
   ```
   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.
Multiple output products can be generated by passing a list to the command, such as `{synthesis instantiation_template}`.

**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.

1. 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_xdc [get_files -of_objects \
                     [get_files clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
   
   This will return the names of the XDC file that are delivered with the Clock Wizard.
   
2. To **disable** the XDC file, add this line to your script as well:

   ```tcl
   set_property is_enabled false [get_files $clk_wiz_xdc]
   
   The XDC constraint(s) delivered with clk_wiz IP are disabled when you run your script.
   
   To check what XDC constraints 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, in your script, Vivado synthesis will return several warnings related to the IP in the design:

```
WARNING: [IP_Flow 19-2162] IP 'Accumulator' is locked. Locked reason: IP definition for IP 'Accumulator' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP.

WARNING: [IP_Flow 19-2162] IP 'char_fifo' is locked. Locked reason: IP definition for IP 'char_fifo' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP. This IP does not have the required outputs for use in simulation.

WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition for IP 'blk_mem_gen_v7_3_0' was not found in the IP catalog. You may continue to use existing outputs but may not modify this IP. This IP does not have the required outputs for use in synthesis.
```

All of these IPs have updated versions in the Xilinx IP Catalog. The versions you read into the in-memory design can be used in their current forms, because they all have the output products required for synthesis. However, you could also upgrade these IPs to the latest version.
In an interactive design session these messages can be helpful; but in a batch mode Tcl script these messages would not been 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.

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.

   The Accumulator core is legacy IP 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.

   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.
**Step 5: 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`, that 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 -part xc7k70t-fbg484-3 -top sys_integration_top
   ```
   Since there is no project file storing the target part, or identifying the top-level of the design, you must provide the part and top-level module name with the `synth_design` command.

   The various Verilog files read into the in-memory design in Step 1: Reading RTL 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.

   **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...
   }
   ```

Step 6: 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 design, 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:

```
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 7: Running the Script

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

```tcl
#Step 1: Reading RTL
cd C:/Data/ug939-design-files/lab_6
read_verilog [glob ./sources/HDL/*.v]

#Step 2: Adding Existing IP
read_ip ./sources/IP/accumulator/accumulator.xco
read_ip ./sources/IP/char_fifo/char_fifo.xci
file mkdir IP/blk_mem
file mkdir IP/clk_wiz
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
read_ip ./IP/blk_mem/blk_mem_gen_v7_3_0.xci
read_ip ./IP/clk_wiz/clk_wiz_0.xci

#Step 3: Disable XDC
generate_target synthesis [get_files clk_wiz_0.xci]
set clk_wiz_xdc [get_files -of_objects [get_files \nclk_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 != "" && $locked} {
  upgrade_ip [get_ips blk_mem_gen_v7_3_0]
}
set locked [get_property IS_LOCKED [get_ips Accumulator]]
set upgrade [get_property UPGRADE_VERSIONS [get_ips Accumulator]]
if {$upgrade != "" && $locked} {
  upgrade_ip [get_ips Accumulator]
}

#Step 5: Running Synthesis
read_xdc ./sources/Constraints/top_timing.xdc
synth_design -part xc7k70t-fbg484-3 -top sys_integration_top
write_checkpoint -force post_synth.dcp
report_timing_summary -file timing_syn.rpt

#Step 6: Running Implementation
read_xdc ./sources/Constraints/top_physical.xdc
opt_design
place_design
write_checkpoint -force post_place.dcp
report_timing -file timing_place.rpt
phys_opt_design
route_design
write_checkpoint -force post_route.dcp
report_timing_summary -file timing_summary
write_bitstream -force sys_integration_top.bit
```
Step 7: Running the Script

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

- Batch mode will automatically exit the tool after the script has finished processing.
- Tcl mode will return to the Vivado Design Suite Tcl prompt when the script has finished.

**TIP:** You can also source the Tcl script from within the Vivado IDE using the Tools > Run Tcl Script command.

1. Open a Linux shell, or a Windows Command Prompt window.

2. From the command line, go to: `<Extract_Dir>/ug939-design-files/lab_6.`
   
   ```
   > vivado -mode batch -source nonproject_run.tcl
   or
   > vivado -mode tcl -source nonproject_run.tcl
   ```

3. Running the script results in a directory called “IP” being created, output products for the various IPs used in the design, reports, design checkpoints, and a bitstream that is written to disk. The design checkpoints can be opened in the Vivado IDE to perform further analysis:
   
   ```
   > vivado post_synth.dcp
   ```

   To open the Vivado IDE while running in Tcl mode, simply type `start_gui` at the Tcl prompt.

---

9 This presumes that the Vivado Design Suite is properly installed, and can be found in your $PATH environment. Refer to the Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973) for more information.
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 tutorial 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.
- Manually running synthesis and individual steps of implementation.
- Generating custom reports and creating Design Checkpoints (DCP)
Lab 7: Creating an IP Integrator Subsystem Design

This lab provides a brief introduction to using the IP integrator feature of the Vivado Design Suite. The Vivado IP integrator feature lets you create complex system designs by instantiating and interconnecting IPs from the Vivado IP catalog on a design canvas.

In this exercise you will be using a version of the Xilinx Wave Generator example project, that is missing a FIFO, which you will quickly customize, generate, and instantiate using the IP integrator. You will then synthesize and implement the top-level design.

**IMPORTANT:** The Vivado IP integrator environment is a licensed early access feature in the 2013.1 release. Please contact your Field Applications Engineer to obtain a license.

---

**Step 1: Opening the Project**

Invoke the Vivado IDE.

On Linux,

1. Change to the directory where the lab materials are stored:
   ```
   cd <Extract_Dir>/lab_7
   ```
2. Launch the Vivado IDE: `vivado`

On Windows,

1. Launch the Vivado Design Suite IDE:
   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.1 > Vivado 2013.1
   ```
2. As an alternative, click the Vivado 2013.1 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.
3. Select Open Project from the Getting Started page, and navigate to and select:
   ```
   <Extract_Dir>/lab_7/project_wave_gen_ipi/project_wave_gen_ipi.xpr
   ```
4. Click OK.

The Vivado IDE opens the specified project as shown in Figure 48.

---

10 Your Vivado Design Suite installation may be called something different from Xilinx Design Tools on the Start menu.
Step 2: Opening the IP Integrator Design Canvas

You can create designs interactively through the IP integrator canvas GUI or programmatically through a Tcl programming interface. Designs are typically constructed at the interface level (for enhanced productivity) but may also be manipulated at the port level (for precision design manipulation).

1. In the Flow Navigator, under **IP Integrator**, click on **Create Block Design**. The Create Design dialog box opens as shown in Figure 49.
Step 2: Opening the IP Integrator Design Canvas

2. In the **Create Design** dialog box, for **Design Name**: type `char_fifo`.

3. Click **OK**.

   The IP integrator design canvas opens, as shown in Figure 50, with an empty IP subsystem design.

   **TIP**: All of the features of the IP integrator are available through the use of Tcl commands. The command for creating a new block diagram is:

   ```tcl
   create_bd_design char_fifo
   ```

---

**Figure 49: Create Block Design**

**Figure 50: IP Integrator Design Canvas**
In the Tcl Console type the following command:

```
set_param bd.skipSupportedIPCheck true
```

This will enable the FIFO Generator to show up in the IP integrator IP Catalog.

4. Right click in the IP integrator design canvas to open the popup menu, and click **Add IP**. The IP integrator IP catalog opens, displaying a list of IP available in the IP Integrator.

5. Type **FIFO** in the **search** box at the top of the IP integrator Catalog, as seen in Figure 51.

![Figure 51: IP Integrator Catalog](image)

6. **Double click** the **FIFO** Generator, or select it and hit Enter.

   The FIFO is added to the IP integrator design canvas.

![Figure 52: FIFO Instance](image)
Step 3: Customizing IP

To customize the FIFO Generator, double click on it, or select it and right click to select Customize Block from the popup menu. The FIFO Generator displays in the Re-customize IP dialog box, as shown in Figure 53.

The Basic tab defines the interface type, and other implementation characteristics.

1. For **Interface Type**, check that the default **Native** is still selected.
2. For FIFO Implementation, select **Independent Clocks Block RAM**.
3. **Deselect** the **Show disabled ports** in the symbol view, if it is currently selected.

![Figure 53: Re-customize FIFO](image)

4. **Select** the **Native Ports** tab.

   From the Native Ports tab you can configure the Read Mode, Built-in FIFO Options, Data Port Parameters, and Implementation Options.

5. Set the **Read Mode** to **First Word Fall Through**.
6. Set the **Write Width** to be **8 bits**.
7. **Click** in the **Read Width** field, to change it automatically to match the Write Width. Leave everything else with the default settings on this tab.

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

9. Select the **Summary** tab. This displays a summary of all the selected configuration options, as well as listing resources used for this configuration, as seen in Figure 54.

![Figure 54: Re-customize FIFO -Summary](image)

Verify that the information is correct. For this configuration you are using one 18K BRAM.

10. Make any changes required to match the configuration, and click **OK**.
Step 4: Completing the Subsystem Design

To integrate the subsystem design into the top-level of the design, you can add ports to create connections external to the subsystem.

1. Click to select the `wr_clk`, then press and hold the Ctrl key and click to select the `rd_clk` port of the FIFO.
   You can select multiple ports and add external connections to all of them at the same time.

2. With the ports highlighted, right-click to open the popup menu, and select **Make External**, as shown in Figure 55.

![Make External Connection](image)

**Figure 55: Make External Connection**

Two external connections are created for the selected FIFO ports. Notice the external connections have the same name as the IP module port they connect to. You can rename these connections, by selecting them and changing the name in the External Port Properties window.

3. **Select** the external connection port named `wr_clk`.
4. In the **External Port Properties** window, type the name `clk_rx`, and press **Enter**.
5. **Select** the external connection port named `rd_clk`.

---

Designing with IP  
UG939 (v 2013.1) May 7, 2013  
www.xilinx.com  
80
6. In the **External Port Properties** window, type the name `clk_tx`, and press **Enter**.

![Image](image_url)

_Figure 56: External Port Properties – Rename Connections_

7. Click on all of the remaining FIFO input and output ports, while pressing the Ctrl key, to select all remaining ports.

8. Right click and select the **Make External** command.

   This will add external connections to all the ports of the FIFO.

9. **Select** one **external connection** at a time, and **change** the **name** in the External Port Properties window to match the following:

   - `wr_clk = clk_rx`
   - `rd_clk = clk_tx`
   - `din = char_fifo_din`
   - `dout = char_fifo_dout`
   - `empty = char_fifo_empty`
   - `full = char_fifo_full`
   - `rd_en = char_fifo_rd_en`
   - `wr_en = char_fifo_wr_en`
   - `rst = rst_i`

   When you have finished, your subsystem design should look like Figure 57.

---

**TIP:** You can refresh the IP integrator design canvas at any time using the **Regenerate** command from the sidebar menu.
Step 5: Generating IP Output Products

With the char_fifo subsystem design added to the current project, you can generate the required output products to drive synthesis, simulation, and implementation.

1. In the IP Sources tab of the Sources window, select the char_fifo.
2. Right click to open the popup menu, and select Generate Output Products.

The Manage Output Products dialog box displays the various output products that are available for the char_fifo subsystem design, as shown in Figure 58.
3. Verify that the **Actions** are set to **Regenerate**, and click **OK**.

   You will see the various IP output products displayed in the IP Sources tab of the Sources window.

   ![IP Sources](image)

   **Figure 59: IP Sources – Generated Outputs**

---

**Step 6: Instantiating IP into the Design**

With the `char_fifo` subsystem design in the project, you still need to integrate it into the design by instantiating the module into the appropriate HDL source file. The Vivado Design Suite can generate an instantiation template for the subsystem design to help you with this process.

1. From the **IP Sources** tab of the Sources window, **select** the `char_fifo` module.

2. **Right-click** to open the popup menu, and select the **View Instantiation Template** command.

   The instantiation template, `char_fifo_wrapper.v`, is opened in the Text Editor in the Vivado IDE.

   Scroll to the bottom of the file, to line 33 as seen in Figure 60, and select the text defining the `char_fifo` instance.
3. With the char_fifo text selected in the Text Editor, right-click to open the popup menu, and select the Copy command, or press Ctrl-C.

Now you need to copy the char_fifo instance into the appropriate HDL source file. In this case, you will instantiate the subsystem design into the top-level module, wave_gen.

4. From the Hierarchy tab of the Sources view, double click on wave_gen.v to open the file for editing.

5. Scroll down to line 337, which reads:

   // Instantiate the Character FIFO IP here

6. Position the Text Editor cursor on line 338, and right-click and select Paste, or press Ctrl-V.

7. Click on the Save File icon to save the wave_gen.v file.

   Notice that the Hierarchy, Libraries, and Compile Order tabs are updated to reflect that the IP integrator subsystem design has been instanced into the design.
Elaborate the design to verify that your connections are correct.

8. In the **RTL Analysis** section of the Flow Navigator select, **Open Elaborated Design**.

   An RTL Schematic opens and you can zoom in to see the connections of the **char_fifo** IP subsystem design.

9. From the **RTL Netlist** tab in the Sources view, select **char_fifo**.

---

**Conclusion**

Congratulations! You have successfully started a block diagram, instantiated and customized a FIFO and then added the IP to your top level design. Exit the Vivado IDE or proceed to simulation, synthesis, or implementation.

In this exercise, you learned how to select and customize an IP in the IP integrator design canvas, and how to instantiate the IP Integrator subsystem design into a larger design. You can do this interactively within the Vivado IDE, or via Tcl commands.

This is a simple IP integrator tutorial, which used only one IP. However, IP integrator can be used to create complex IP subsystems, containing multiple IPs, including embedded or hardcore processors. Refer to the *Vivado Design Suite User Guide: Designing with IP* (UG896) for more information on the Vivado IP integrator.
Lab 8: Converting Legacy EDK IP

In this exercise, you will learn how to convert an XPS processor core, or Pcore, to a Vivado Design Suite native IP for use in IP Integrator. For basic Pcores, the conversion process is very similar to the steps outlined in Lab 4.

Step 1: Managing IP

On Linux,
1. Change to the directory where the lab materials are stored:
   cd <Extract_Dir>/lab_8
2. Launch the Vivado IDE: **vivado**
On Windows,

1. Launch the Vivado Design Suite IDE:
   Start > All Programs > Xilinx Design Tools > Vivado 2013.1 > Vivado 2013.1

2. As an alternative, click the Vivado 2013.1 Desktop icon to start the Vivado IDE.
   The Vivado IDE Getting Started page, shown in Figure 62, contains links to open or create projects and to view documentation.

3. Select Manage IP from the Getting Started page and select Open IP Catalog from the drop down menu.

![Figure 63: Manage IP](image)

The Manage IP Initial Settings dialog box displays, as seen in Figure 64.

![Figure 64: Manage IP Initial Settings](image)

---

11 Your Vivado Design Suite installation may be called something different from Xilinx Design Tools on the Start menu.

5. Click **OK** to close the **Manage IP Initial Settings** dialog box.

   The Vivado IDE will load the IP Catalog view layout.

![Image of the IP Catalog view layout](image-url)

**Figure 65: IP Catalog View Layout**

Since this is a Managed IP session and not an RTL project, you cannot do behavioral simulation, elaborate the design, or launch synthesis or implementation. This view is strictly for exploring the IP Catalog, and for packaging designs for use as IP.
Step 2: Running the Package IP Wizard

1. Select Tools > Package IP from the main menu.
   The Package New IP wizard displays and provides some information about the process you are starting.

2. Press Next to continue.
   In the Choose IP Source Location dialog box, Package a specified directory is the only valid option.

3. In the IP Definition Location, select   and browse to <Extract_Dir>/lab_8.

   ![Figure 66: Setting the IP Source location](image)

4. Click Next.
   The Begin IP Creation page displays, showing what work will be performed during the IP packaging flow.

5. Click Finish.
   The Package IP wizard collects the available information from the specified IP location, and the IP Packager Summary screen summarizes what is gathered and populated into the component.xml file, a newly created IP-XACT definition of the IP.
   
   **Note:** If there is a data/*.pao file present in the specified IP location, then this file will be read, and the file and associated library information will be used, as is the case for this EDK Pcore IP.

6. Examine the contents of the IP Packager Summary, as seen in Figure 67.
7. Click **OK** to close the IP Packager Summary dialog box.

   The next step is to review the various sections of the Package IP window, as seen in Figure 68, and update information as required. The Package IP window shows the current IP identification, including the Vendor, Library, Name, and Version (VLNV) attributes of the newly packaged core.
Step 2: Running the Package IP Wizard

8. In the **IP Identification** page make the following changes:
   - Display Name: `my_axi_gpio`
   - Description: **AXI general purpose IO block**
   
   As with any design that you package as an IP, you can provide company identification such as name and web site as well as add supporting documentation, examples, testbench, etc.

9. Closely examine the **IP Customization Parameters** tab.
   
   One item to consider is the address range for the core: `C_S_AXI_ADDR_WIDTH`. Is the address range specified as a fixed value, or is it configurable at the time the IP is customized? If it is configurable, what is the reason? Is the IP acting like a bridge with a variable sized memory window on the slave side? Or, does it have a variable sized memory buffer for storing information?

   Most IP have a fixed set of registers, and this parameter is meaningless in the context of the IP Integrator as the HDL does not really accommodate having a variable memory range. This parameter made sense in XPS/EDK because of how that software worked. However, in the IP Integrator the interconnect takes care of this, and instead the bits are directly related to the range.

10. In the **IP GUI Customization Layout** section you see a preview of how the IP will look, during customization, and when added to an IP Integrator subsystem design.
11. The **Review and Package** page presents a summary of the IP, and provides a list of possible missing information, such as product guide, example, or change log.

    The IP definition file, `component.xml`, is saved automatically as you make changes to the different pages of the Package IP window.

    Exiting Vivado or closing the Package IP window will force a final save of the file.

12. In the **Review and Package** page, press the **Add to Catalog** button.

    The Deploy IP to Catalog window displays, as shown in Figure 70, informing you that the IP directory will be added to the IP Catalog as a local repository to the current project or Manage IP session.

13. Click **OK** to add the IP directory to the catalog.
Adding the IP to the catalog will result in any unsaved edits to be written to the IP definition file, `component.xml`, saved to the location of the existing Pcore, which is now also an IP Integrator IP. Refer to Lab 7 for a brief introduction to using the Vivado IP Integrator.

The Add to Catalog command only adds the IP repository to the IP Catalog in the current project, or Manage IP session. To use the IP repository in other designs, you must add the IP repository to the IP Settings using the **Tools > Project Settings** command from the main menu. Other users can also use the IP from the repository you just created by adding it to the IP Settings of their own projects.

---

**RECOMMENDED:** To use a custom IP repository across multiple design projects, or for use by multiple designers, you can add the repository when the Vivado tool launches by adding it to your `init.tcl` script. Refer to Vivado Design Tcl Command Reference Guide (**UG835**) for more information on the `init.tcl` script.

---

14. **Select** the **IP Catalog** tab and look for `my_axi_gpio` in the **BaseIP** category, and review the details.

You have completed the creation of an IP-XACT definition file for the IP, which can now be used in a Vivado IP Integrator subsystem design.
Conclusion

A different approach to this exercise would have you editing a Vivado project, which included an EDK Pcore, and you decided to create a native Vivado Design Suite IP from the Pcore for use in the Vivado IP Integrator.

You could:

- Launch the Package IP wizard from within the current project,
- Reference the location of the existing Pcore as the IP repository,
- Follow the steps of this lab to Package the IP,
- Add the new IP repository to the IP Catalog,
- Use the new IP in your current project to replace the existing Pcore.

In this exercise, you learned how to create an IP Integrator IP from an existing EDK Pcore. The process involved creating an IP-XACT definition file, `component.xml`, via the Package IP wizard. The process can be done either through the Manage IP flow, working directly with the Pcore, or can be done within your design project.