Notice of Disclaimer

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

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

Revision History

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

Revision History.................................................................................................................................................. 2  
Table of Contents.................................................................................................................................................. 3  
Designing with IP .................................................................................................................................................. 5  
  Overview ......................................................................................................................................................... 5  
  Tutorial Description ....................................................................................................................................... 5  
  Software Requirements ................................................................................................................................. 6  
  Hardware Requirements ................................................................................................................................. 6  
  Locating and Preparing the Tutorial Design Files .......................................................................................... 6  
Lab 1: Designing with the IP Catalog ................................................................................................................ 7  
  Step 1: Opening the Project ............................................................................................................................ 7  
    Launch Vivado ............................................................................................................................................ 7  
    Open the Project ....................................................................................................................................... 8  
  Step 2: Customizing the FIFO Generator ..................................................................................................... 9  
  Step 3: IP Output Products ............................................................................................................................ 13  
  Step 4: Instantiating the IP Template into a Design .................................................................................... 14  
  Step 5: Pre-synthesizing an IP ....................................................................................................................... 18  
  Conclusion ..................................................................................................................................................... 19  
Lab 2: Creating and Managing Reusable IP ...................................................................................................... 20  
  Step 1: Starting a Manage IP session ............................................................................................................... 20  
  Step 2: Customizing the FIFO Generator ..................................................................................................... 22  
  Step 3: Using Third Party Simulators .......................................................................................................... 26  
  Step 4: Generating a Netlist for IP ................................................................................................................. 27  
  Step 5: Writing a Simulation Netlist .............................................................................................................. 29  
  Step 6: Using Third Party Synthesis Tools ................................................................................................... 30  
  Conclusion ..................................................................................................................................................... 31
Designing with IP

Overview

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

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

Figure 1: Vivado Design Suite IP Design Flow

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. You are also shown how to optionally generate a netlist for the IP to reduce synthesis run time of the design.

- **Lab 2:** You will learn how to create and customize IP using the Manage IP flow. Create a project, include an IP from the IP catalog as the top-level source; customize and verify the IP;
then netlist the IP for inclusion in another design. Use the custom IP as a black box in a 3rd party synthesis flow.

- Lab 3: Package a design as an IP module and add it to the Vivado IP catalog. Open a completed Vivado Design Suite project; package the design as an IP core and add it to the IP catalog using IP packager; then verify the new IP through synthesis and implementation.
- Lab 4: Write and run a Tcl script using the Vivado Design Suite Project Mode.
- Lab 5: Write and run a Tcl script using the Vivado Design Suite Non-Project Mode.

---

**Software Requirements**

This tutorial requires that the 2013.2 Vivado Design Suite software release or later is installed. For installation instructions and information, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973)*.

---

**Hardware Requirements**

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

Xilinx recommends a minimum of 2 GB of RAM when using the Vivado tool.

---

**Locating and Preparing the Tutorial Design Files**

There are separate project files and sources for each of the labs in this tutorial. You can find the design files for this tutorial under *Vivado Design Suite -2013.2 Tutorials* on the Xilinx.com website.

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

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

---

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

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

Step 1: Opening the Project

![Vivado IDE - Getting Started Page](image)

**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 2013.2 > Vivado 2013.2**

2. As an alternative, click the **Vivado 2013.2** 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 other than **Xilinx Design Tools** on the **Start** menu.
Step 2: Customizing the FIFO Generator

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

Step 2: Customizing the FIFO Generator

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

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

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

   The search results narrow the list of IP 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.

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

The **Switch to Defaults** resets the configuration options back to the default settings.
9. At the top of the Customize IP dialog box, change the **Component Name** to **char_fifo** from the default name.

   The Basic tab defines the interface type, memory type, and other implementation options of the 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, it will automatically change to 8 bits as well.

   Leave everything else with the default settings on this tab.
16. **Examine** the fields of the **Status Flags** and **Data Counts** tabs.

   These fields configure other options for the FIFO Generator. For this design, leave everything with the default settings.

17. Select the **Summary** tab.

   This displays a summary of all the options selected as well as listing resources used for this configuration. The summary for the FIFO Generator core should look like Figure 7. For this configuration you are using one 18K BRAM.

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

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

   The Generate Output Products dialog box opens, to display a list of source file output products that can be generated as the IP is added to your project.

19. **Click Generate** to add the FIFO core to your project, and generate the supporting output products. Leave the Generate Synthesized Design Checkpoint (.dcp) unchecked at this point. This is used later in this lab in Step 5: Pre-synthesizing an IP.

   The FIFO now appears in the Sources view. Because the FIFO core is not yet instantiated into your design, the core is added 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. Examine the Tcl Console in the Vivado IDE, as shown in Figure 8, and review the **Tcl commands** used to add the FIFO Generator core to your project.
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.

**Step 3: IP Output Products**

The customized FIFO Generator core includes an instantiation template, `char_fifo.veo`, synthesis constraints and VHDL entity and architecture definition, Verilog simulation files, and example design files. Since the FIFO core was originally defined in VHDL, the entity and architecture are added as VHDL source files. However, since Verilog is the target language for the project, the instantiation template is a VEO file for instantiating the VHDL FIFO entity into the Verilog design.

1. In the **IP Sources** tab of the Sources window, examine the **output products** produced for the customized FIFO Generator core, as shown in Figure 9.

Figure 8: Tcl Console

- Figure 9: FIFO Generator Sources
You can generate, or regenerate, the output products from a selected IP at any time, to replace or update the files added to your project.

2. In the **IP Sources** tab select both the **clk_core** and **char_fifo** IP, right click and select **Generate Output Products**.

   The Generate Output Products dialog box displays again, as shown in Figure 10, this time with the output products listed for all selected cores.

   Notice in Figure 10, the output products that are available for the **clk_core** are slightly different from the output products available for the **char_fifo**. The **clk_core** provides an Implementation output product, while the **char_fifo** does not.

   The output products that are available for use with a specific core depend on the definition and packaging of the core in the IP catalog. Lab 3: Packaging an IP for Reuse shows you how to add output products while adding an IP core to the catalog.

   ![Figure 10: Manage Output Products](image)

3. Click **Cancel**, to close the dialog box without generating output products.

### Step 4: Instantiating the IP Template into a Design

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

1. In the IP Sources tab of the Sources window, expand the **Instantiation Template** and double click on the **char_fifo.veo** file to open the template in the Vivado Text Editor.

2. Scroll down to line 56 or 57 of the template file, and select and copy the module instantiation text, as shown in Figure 11.
Step 4: Instantiating the IP Template into a Design

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

3. From the **Hierarchy** tab of the **Sources** view, double click on **wave_gen.v** to open this file for editing.
4. Go down to line 337 which contains a comment stating the Character FIFO should be instanced at this point. Paste the template code into the file as shown in Figure 13.

However, since it is only a template for the module, you will need to do some local editing to make the module work in your design.

5. **Change** the module name from `your_instance_name` to `char_fifo_i0`.

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

```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 ( ) 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.

![Figure 14: Instantiated FIFO IP](image)

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, which you can explore.

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

The cell is cross-selected in the schematic.
Step 5: Pre-synthesizing an IP

All Xilinx IP in the Vivado Design Suite IP Catalog are delivered as RTL source. This provides the benefit of being able to perform behavioral simulations, which are much faster than netlist based simulation. However, having to synthesize the IP every time during development of a design has a runtime impact. To reduce runtimes you can pre-synthesize IP selectively.

1. Creating a netlist for an IP is a simple process and is accomplished in one of two ways; either using the IP Sources or Hierarchy tabs of the Sources window. It is easier to use the IP Sources tab as you do not need to find the IP in the hierarchy.

In the IP Sources tab select the char_fifo IP. Right-click and select Generate Output Products as shown in Figure 15.

![Figure 15: Generating netlist output product](image)


A few things will happen:
- A new file set is created and the IP synthesis sources are copied to it
- A new synthesis Design Run is created
- The synthesize run is launched.
- When the run is completed the Hierarchy view is update to show a netlist is being used, as seen in Figure 17.
3. Now when synthesis of the top level is performed a black box will be inferred for the FIFO generator IP (`char_fifo`) in the design. This would be the same if more than one instances of the same IP existed in the design.

Launch synthesis of the top level by pressing the `Run` button from the Flow Navigator.

4. After synthesis completes, open the Synthesis log in the Log tab. Searching for “blackbox” will take you to the following section which summaries the black boxes found in the design.

```
Writing Synthesis Report

Report BlackBoxes:
<table>
<thead>
<tr>
<th>BlackBox name</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>char_fifo</td>
<td>1</td>
</tr>
</tbody>
</table>
```

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

`<Extract_Dir>/lab_1/project_wave_gen_ip/project_wave_gen_ip.runs/char_fifo_synth_1`

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

**Conclusion**

Congratulations! You have successfully customized the FIFO Generator IP core, and added it to the design. Close the project and exit Vivado tool, or continue and implement the design if you wish to explore further.

In this exercise, you learned how to select and customize an IP from the IP catalog, how to instantiate the customized IP into an HDL design, and how to pre-synthesize an IP. You can do this interactively within the Vivado IDE, or via Tcl scripting.
Lab 2: Creating and Managing Reusable IP

To simplify revision control, and to support the use of custom IPs across multiple projects and designs, you can manage and store the custom IP in a repository, separate from any design projects they are used in. The IP customization file (.xci), and the output products for synthesis, simulation, examples, etc., should be contained together in a unique directory. You can reference these custom IP cores in new projects and designs, to simulate, synthesize, and implement the IP core as part of the design. 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 custom IP core in an IP repository, using the Manage IP flow in the Vivado Design Suite. This flow allows you to browse the IP catalog to create and manage IP customizations for use in either a Project or a Non-Project design flow. You can create a repository of the customized IPs for use in your design(s), managed and maintained outside of a Vivado Project. A special IP project is created at the location specified for the Manage IP flow, to facilitate the creation of an optional synthesis design checkpoint (DCP) in the custom IP repository. You can use the synthesis DCP files in Vivado Design Suite designs to speed synthesis of reused IP cores, and in third party synthesis tools to provide netlists for black box IP.

Step 1: Starting a Manage IP session

On Linux,

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

2. Launch the Vivado IDE: **vivado**

On Windows,

1. Launch the Vivado Design Suite IDE:

   ```
   Start > All Programs > Xilinx Design Tools > Vivado 2013.2 > Vivado 2013.2
   ```

2. As an alternative, click the **Vivado 2013.2** Desktop icon to start the Vivado IDE.

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

---

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

![Manage IP](image)

**Figure 18: Manage IP from the Getting Started Page**

A drop down menu displays, as seen in Figure 18, letting you:

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

*Note:* This second option is disabled if this is your first time running the Manage IP flow.

4. Select **New IP Location**.

The Manage IP Initial Settings dialog box displays.

5. Press **Next**.

6. Change the dialog box settings as shown in **Figure 19**.

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

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

![Manage IP Settings](image)

**Figure 19: Manage IP Settings**
7. Press Finish to proceed.

The IP Catalog displays in an IP Project, which is a simple interface for the creation and management of IP customizations.

### Step 2: Customizing the FIFO Generator

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

1. Type fifo in the search bar.
2. Double-click the **FIFO Generator** from the Memories & Storage Elements group.

   The Customize IP dialog box opens.

![Figure 20: 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](https://www.xilinx.com). This Lab assumes that you have previously completed Lab 1, and are familiar with the concepts covered in it.

3. At the top of the Customize IP dialog box, change the **Component Name** to **char_fifo** from the default name of **fifo_generator_0**, as shown in Figure 20.
4. In the Basic tab:
   - Select the **Interface Type** of **Native**, which is the default.
   - From the **Fifo Implementation** drop down menu, set **Independent Clocks Block RAM**.

5. In the **Native Ports** tab:
   - Set the **Read Mode** to **First Word Fall Through**.
   - Set the **Write Width** to be **8 bits**.
   - **Click** on the **Read Width** field to adjust it automatically to 8 bits as well.

6. Select the **Summary** tab.
   
   The Summary page displays a summary of all the options selected as well as listing resources used for this configuration. The summary for the FIFO Generator core should look like **Figure 21**. For this configuration you will see you are using one 18K BRAM.

![Figure 21: Summary of FIFO Generator Core](image)

7. **Verify** that the information is correct as shown, and click **OK** to generate the customized IP core for use in your design.
   
   The Generate Output Products popup appears, as seen in **Figure 22**.

8. Leave the **Generate Synthesized Design Checkpoint (.dcp)** box **unchecked**, as you will generate this file in a later step.

9. **Click** the **Generate** button to create the output products.
   
   When creating a repository of customized IPs using the Managed IP flow, Xilinx recommends that you generate all output products for each IP. A specific release of the Vivado Design Suite only supports a single version of an IP core. You cannot re-customize or generate output products for a previous version of IP in the Vivado Design Suite. Instead, you would...
need to update the IP to the latest version if you have not preserved the needed output products.

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

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

![Generate Output Products](image)

**Figure 22: Generate Output Products – No DCP**

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

You will notice that two directories have been created, as shown in Figure 23.

- One is the IP customization (`char_fifo`) containing the XCI file, which has all the customization information for the IP, and all the output products generated.
- The second is the IP Project (`managed_ip_project`) which is used to manage the synthesis runs for generation of the synthesized DCP.
11. **Examine the Sources window** to see the generated output products in the Manage IP project.

The generated output products for the `char_fifo` core include the Instantiation Template, Synthesis files, Simulation files, and Example Design files.
Step 3: Using Third Party Simulators

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

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

Xilinx IP delivered in the Vivado Design Suite are encrypted using industry standard IEEE P1735 encryption. Simulators supporting this encryption standard can be used to run behavioral simulation. A list of simulation files and the libraries they belong to is needed.

To get all files that an IP delivers for simulation, use the `get_files` Tcl command.

1. In the Tcl Console, type the following command:
   ```tcl
   get_files -compile_order sources -used_in simulation -of_objects \[
   [get_files char_fifo.xci]
   
   This will produce a list of file names, including the full path, required to simulate the IP. In this case, the list includes:
   ...
   .../my_ip/char_fifo/fifo_generator_v10_0/simulation/fifo_generator_v10_0.v
   ...
   ...
   
   The `-used_in` option lets you specify files that are marked for use in simulation, or marked for use in synthesis.

   The `-of_objects` option lets you extract files that are associated with the specified the IP core file.

   Each simulation file has a LIBRARY property that can be queried. For VHDL files, the library associated with each file is required for simulation.

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

Note: AR# 56487 provides an example Tcl script to collect the file lists for Xilinx IPs that can be used for simulation. You can use the scripts as a template and integrate it into your verification environment.

Step 4: Generating a Netlist for IP

To reduce synthesis runtime for a design using customized IP, the IP can be pre-synthesized as a standalone module. During synthesis of the overall design, a black box is inferred for the IP core, and the pre-synthesized IP netlists are linked into the design during implementation. This method can be used in both Project and Non-Project Mode, and can be scripted using Tcl.

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

1. In the IP Sources tab of the Sources window, right click the char_fifo and select Generate Output Products from the popup menu.

2. Enable the Generate Synthesized Design Checkpoint (.dcp) checkbox, as shown in Figure 25, and click Generate.

![Figure 25: Generate Output Products - DCP](image)
A new Design Run is created which references all the synthesis source files for the IP. This run is automatically launched, and when synthesis is completed the `char_fifo` directory contains a few additional files:

- **char.dcp**: The Synthesis Design Checkpoint which consists of both a netlist and constraints for the IP.
- **char_fifo_synplify_stub.v**: A Verilog port module for use with Synplify Pro, to infer a black box for the IP with no IO buffers inserted.
- **char_fifo_synplify.vho**: A VHDL component instantiation template for use with Synplify Pro, to infer a black box for the IP with no IO buffers inserted.

At this point in the tutorial, you can repeat some of the earlier steps to add some additional IP cores to the project, and customize them as well. In Figure 26, you can see that two additional cores have been added to the Manage IP project.

3. Explore the **IP Catalog**, and create **customizations** for a few additional IP cores.

4. **Generate output products** for the additional cores, or skip generating the output products at this time.

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

![Output products of an IP](image)

**Figure 26:** Output products of an IP

When an IP customization completes, an instantiation template is generated, even if you choose to skip generating output products. This is the minimum output.

When the synthesis DCP is created for an IP core, a corresponding Out-of-Context Design Run is also created and launched. The progress of these synthesis runs can be viewed in the Design Run tab, as shown in Figure 27.
Step 5: Writing a Simulation Netlist

All Xilinx IP delivered in the Vivado Design Suite are HDL (Verilog or VHDL). The HDL is encrypted with IEEE P1735 which is supported by third party simulators. If your simulator does not have multi-language support, you can create a structural simulation netlist. A DCP must be created first as covered in Step 4: Generating a Netlist for IP.

1. In the Design Runs window, right click on the char_fifo_synth_1 run and select Open Synthesized Design from the popup menu, as shown in Figure 28.

2. In the Tcl Console type one of the following commands to write the Verilog or VHDL netlist:

   ```bash
   write_verilog -mode funcsim <Extract_Dir>/lab_2/my_ip/char_fifo/char_fifo_sim.v
   
   or
   
   write_vhdl -mode funcsim <Extract_Dir>/lab_2/my_ip/char_fifo/char_fifo_sim.vhd
   
   Note: Replace <Extract_Dir> with the location of the extracted tutorial files.
   
   All of the outputs for the custom IP should be directed to the custom IP repository you defined, or my_ip in this case. This makes revision control for the custom IP repository straightforward, and keeps the IP directories independent from the installation.
Step 6: Using Third Party Synthesis Tools

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

A Verilog stub or VHDL component declaration for the IP is required so that the third-party synthesis tool can infer black boxes for Xilinx IP cores, and does not insert IO buffers. These files are created for Synopsys Synplify Pro (<ip_name>_synplify_stub.v and <ip_name>_synplify.vho) when the synthesis design checkpoint (DCP) is created.

The following sample Tcl script reads a top-level design netlist, and also reads the char_fifo DCP, then reads the design constraints and runs implementation, in the Vivado tool Non-Project Mode. The synthesis checkpoint file also contains constraints for the IP, as well as the netlist.

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

# Read top-level EDIF and IP DCP
read_edif ./sources/wave_gen.edif
add_files ./my_ip/char_fifo/char_fifo.dcp

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

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

# Logic optimization
opt_design

# Placement
place_design

# routing
route_design

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

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

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

5. **Examine** the Netlist window to see that the char_fifo IP was correctly linked to the top-level design.

![Netlist Window](image)

**Figure 29: Netlist Window**

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

---

**Conclusion**

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

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

In Lab #3, 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:
   
   ```bash
   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 2013.2 > Vivado 2013.2
   ```
2. As an alternative, click the Vivado 2013.2 Desktop icon to start the Vivado IDE.
   
   The Vivado IDE Getting Started page displays with links to open or create projects, and to view documentation.
   
   For either Windows or Linux, continue the lab from this point.
3. Click **Open Project**, and browse to: `<Extract_Dir>/lab_3/my_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.

---

3 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_3/source_files`.

   ![Package New IP](image)

   **Figure 30: Package New IP**

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 31 [Error! Reference source not found.].

---

Designing with IP

UG939 (v 2013.2) June 26, 2013

www.xilinx.com
Step 2: Identifying and Documenting the IP

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.

![IP Packager Summary](image)

Figure 31: IP Packager Summary

Step 2: Identifying and Documenting the IP

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

8. 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 32 shows how this should look.
Step 2: Identifying and Documenting the IP

9. For Categories, select the browse button, \(\text{Browse}\), to open the Choose IP Categories dialog box, as seen in Figure 33.

   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.

10. Press Cancel to close the Choose IP Categories dialog box.
11. In the Package IP window, click **Review and Package**, as shown in Figure 34.
Notice the Possible Missing Information section that identifies missing elements such as documentation and design examples. You will add a PDF Product Guide to the packaged IP at this time.

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

The IP File Groups displays different collections of information, or files, associated with the IP. The standard files groups include Verilog and VHDL source files for synthesis and simulation, and datasheets, product guides, and other documentation to support the use of the IP core.

13. Right-click on the **Standard** section to open the popup menu, and select **Add File Group**, as shown in Figure 35.

14. Select **Product Guide**, as shown in Figure 36, and click **OK**.
15. With the Product Guide file group added, right-click on it to open the popup menu, and select the Add Files command.

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

16. Click Add Files... to open the Add IP Files dialog box.

17. Navigate to `<Extract_Dir>/lab_3/source_files/doc`

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


20. Click OK to close the Add IP Files dialog Box

21. 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: Package 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 the packaged IP to the IP catalog.

In the Package IP window, on the left side pane, click **Review and Package**.

![Figure 37: Package IP](image)

22. In the After Packaging section, click the **settings** link to open the IP tab of the Project Settings dialog box, as seen in Figure 38.
Step 3: Package IP

23. **Examine** the various fields of the **IP Settings** dialog box.

24. Click **Cancel** to close IP Settings.

   The Review and Package pane of the Package IP window, reflects that an archive will be created, and displays the path and filename of the archive. Notice the edit link that lets you change the name and location of the archive file as needed.

25. At the bottom of the Review and Package pane, click the **Package IP** button.

26. **Examine** the `<Extract_Dir>/lab_3` folder to make sure that the **IP archive file** was created.

   The archive will have the ZIP file extension.
The prepared IP is packaged, an archive file is created, and the IP is added to the IP catalog. Adding it to the IP Catalog, means adding the path to the packaged IP to the IP repository, and updating the IP catalog with any IPs found in the new repository.

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

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

29. Select the My Complex Multiplier IP, and examine the Details at the bottom of the IP Catalog window.

![Figure 40: Updated IP Catalog](image)

You will validate that the packaged IP is ready for use in new projects in the next steps.

30. From the main menu, select the File > Close Project command to close the project you used to create the IP definition.
Step 4: 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. Create a new project by selecting **Create New Project** from the Vivado IDE Getting Started page.

![Create New Project](image)

**Figure 41: Create New Project**

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

3. In the Project Name dialog box, set the following options, as seen in **Error! Reference source not found.**:
   - Project name: **test_IP**
   - Project location: `<Extract_Dir>/lab_3`
   - Select the **Create project subdirectory** box.

![New Project Dialog Box](image)

**Figure 42: New Project Dialog Box**

4. Press **Next** to open the Project Type dialog box.
Step 4: Validate the New IP

5. In the Project Type dialog box, select **RTL Project**, and select the **Do not specify sources at this time** checkbox.

6. Press **Next** to open the Default Part dialog box.

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

8. 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.

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

10. 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.

11. **Click** the **Add Repository** button.

12. In the **IP Repositories** dialog box, browse to: `<Extract_Dir>/lab_3/sources`.

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

   In the **Repository Manager** the added path will be displayed in the IP Repositories, as seen in XXX, and the definition of any packaged IPs that are found in that repository are listed in the **IP in Selected Repository** pane.
In this case, the Vivado tool finds and lists the **My Complex Multiplier** IP core, which you packaged earlier in this lab.

14. 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.

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

16. **Select** the **My Complex Multiplier** core and examine the Details pane, as seen in Figure 44.

Notice the details match the information you provided when packaging the IP.
17. 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 45.

18. 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 4: Validate the New IP

19. 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. In addition, the Generate Output Products dialog box opens, as seen in **Figure 46**, to automatically generate the various support files required for this IP in the current project.

20. Click **Generate** to generate the listed output products.
21. **Examine** the Sources window to see that **various design and simulation source files** that have been added to the current project.

22. From the Flow Navigator, click **Run Synthesis**.

23. From the **Synthesis Completed** dialog box, click **Run Implementation**.

24. 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 4: Scripting the Project Mode

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

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

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

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

Step 1: Creating a Project

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

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

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

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

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

   Now you are ready to create your project.

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

   ```
   C:/Data/ug939-design-files/lab_4
   ```
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](https://www.xilinx.com)), or at the Tcl prompt type `help <command_name>`, for more information on the `create_project` command, or any other Tcl command used in this tutorial.

---

**Step 2: Adding RTL Source Files**

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

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

1. Add the following two lines to your script:

   ```tcl
   add_files -scan_for.includes ./sources/HDL
   import_files
   ```

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

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

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

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

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

   ```tcl
   import_files -filesset 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**: 2013.2 version native IP with required output products for synthesis.
- **Clock Wizard**: 2013.2 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/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 'Accumulator' for IP 'Accumulator' has a newer major version in the IP Catalog. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.
   - WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked. Locked reason: IP definition 'Block Memory Generator' for IP 'blk_mem_gen_v7_3_0' has a newer major version in the IP Catalog. No useable outputs are available for this IP. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.
   - WARNING: [IP_Flow 19-3197] Could not import any simulation or synthesis outputs for IP 'blk_mem_gen_v7_3_0'
   - WARNING: [IP_Flow 19-3197] Could not import any simulation or synthesis outputs for IP 'clk_wiz_0'

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

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

   For 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 is current and all output products are present so no warning message is produced.

---

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

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

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 XDC files 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 `{synthesis instantiation_template}`, or you can generated all the available output products by specifying `{all}`.

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

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

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 files that are delivered with the Clock Wizard.

3. **To disable** the **XDC files**, add this line to your script as well:
   ```tcl
   set_property is_enabled false [get_files $clk_wiz_xdc]
   ```

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

   To check what XDC files are evaluated, and in what order, you can use the `report_compile_order` command with the `-constraints` option.
Step 6: Upgrading an IP

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

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

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

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

The following sequence shows you how.

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

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

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

   The Accumulator core is legacy IP that was created with CORE Generator, rather than native IP created in the Vivado Design Suite. The IP has all the necessary output products, for instantiating the HDL module into a design, for synthesis, and for simulation. So it can be used in its current form.
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 `to_step` option that you added to your Tcl script, indicates that implementation should include generating the bitstream for the device. By default, implementation does not include that step. Refer to the *Vivado Design Suite User Guide: Implementation* (UG904) for more information.

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

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

3. Add the `wait_on_run` command to your Tcl script, to wait for the implementation run to complete:
   ```
   wait_on_run impl_1
   ```
   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:

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

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

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

# Step 4: Importing IP
import_ip -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 line⁵, go to: `<Extract_Dir>/ug939-design-files/lab_4`.
   
   ```
   > 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.

---

⁵ 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 5: Scripting the Non-Project Mode

In Lab 4: Scripting the Project Mode, you created a Tcl script to run the Vivado Design Suite in Project Mode. In this lab, you will create a Tcl script to run the Vivado tools in Non-Project Mode. Many of the commands used in this lab are the same commands used in Lab 4. The main difference between Project Mode and Non-Project Mode is that there is no project file or directory 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 in to. 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. In addition, 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_5.

   In Lab 4, 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:

   ```
   cd <Extract_Dir>/lab_5
   ```

   Now you are ready to read the source files.

---

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

   C:/Data/ug939-design-files/lab_5
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:** 2013.2 version native IP with required output products for synthesis.
- **Clock Wizard:** 2013.2 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 IP already have all required output products available, and can be read and used directly from their current location.

**RECOMMENDED:** You should always generate all available output products whenever you create an IP customization. In Non-Project Mode, there is no project part, so the IP output products are generated using the default part for the release when any missing output products are generated. This default part might not be the target part specified by the `synth_design` command, and can result in mismatched synthesis results between the IP and the top-level design in Non-Project Mode designs. See **Lab 2: Creating and Managing Reusable IP** for more information on managing IP customizations.

---

You will upgrade the Accumulator in this lab, though it is not required, since the NGC is present.

1. To read the FIFO Generator IP, including all the output products that are present, add the following line to your script:

   ```
   read_ip ./sources/IP/char_fifo/char_fifo.xci
   ```
For clocking wizard IP, 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, as it is with this IP.

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

_**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, accumulator and the clock wizard IP:

```
file mkdir IP/blk_mem
file mkdir IP/clk_wiz
file mkdir IP/accum
```

3. Add the following to your Tcl script to copy the needed XCI files from the current IP repository into the local directory:

```
file copy -force ./sources/IP/blk_mem/blk_mem_gen_v7_3_0.xci ./IP/blk_mem
file copy -force ./sources/IP/clk_wiz/clk_wiz_0.xci ./IP/clk_wiz
file copy -force ./sources/IP/Accumulator/Accumulator.xco ./IP/accum
```

The _-force_ option causes the file to be overwritten if it 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:

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

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

Unlike in Lab #4, the warnings related to locked IP 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 #4. In Non-Project Mode, the checks are done during synthesis, because that is when the IP 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 files that are included with the Clock Wizard IP as you have constraints in the top-level XDC file that you will apply instead. However, the Clock Wizard IP has not had the output products generated, so you will first generate the synthesis targets, which include the XDC files.

Normally, output products are generated as needed, and do not need to be manually created. However, since you are going to disable the XDC files delivered as an output product, you will need to manually generate the output products 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 files delivered with the Clock Wizard, you need the names of the files. You can query the XDC file(s) that are delivered with an IP, by using the `get_files` command with the `-of_objects` and `-filter` options.

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 files delivered with the Clock Wizard.

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

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

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

   To check what XDC files are evaluated, and in what order, you can use the `report_compile_order` command with the `-constraints` option.
Step 4: Upgrading IP

If you attempt to run synthesis at this time, in your script, Vivado synthesis will return this critical warnings for the accumulator IP:

CRITICAL WARNING: [Designutils 20-1365] Unable to generate target(s) for the following file is locked: c:/Data/ug939-design-files/Test/lab_5/sources/IP/Accumulator/Accumulator.xco
Locked reason: IP definition 'Accumulator' for IP 'Accumulator' has a newer major version in the IP Catalog. Please select 'Report IP Status' from the 'Tools' menu for instructions to unlock.

The accumulator does have an NGC, which will be used during implementation. The block memory IP is not up-to-date and does not have any output products. You will see this warning:

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

Later synthesis will error out due to missing output products for the block memory, it must be upgraded to use.

Both of these IP have updated versions in the Xilinx IP Catalog. The version of the accumulator you read into the in-memory design can be used in its current form, because the NGC required for implementation is present. However, you should upgrade the IP 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 the necessary output products, for instantiating the HDL module into a design, for implementation, 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...
   }
   ```


---

**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`, which is used in both synthesis and implementation.

1. Add the following to your Tcl script:

   ```tcl
   read_xdc ./sources/Constraints/top_timing.xdc
   ```

   The design is now ready for synthesis.
In Non-Project Mode, unlike Project Mode, there are no design runs to launch, and no runs infrastructure managing the strategies used and the state of the design. You will manually launch the various stages of synthesis and implementation.

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

   ```
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:

   ```
write_checkpoint -force post_synth.dcp
   
   The `-force` option is used to overwrite the checkpoint file if it already exists.
   
4. You can also generate any needed reports at this point, such as a post-synthesis timing summary report. Add the following line to your Tcl script

   ```
report_timing_summary -file timing_syn.rpt
   
   This command creates a comprehensive timing report for the design and writes the report to a file.

---

**Step 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 optimization, power optimization, and physical synthesis.

For this lab, you will run the following steps:

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

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

1. Add the following line to your Tcl script:
   ```
   read_xdc ./sources/Constraints/top_physical.xdc
   ```

2. Add optimization and placement commands to your Tcl script:
   ```
   opt_design
   place_design
   write_checkpoint -force post_place.dcp
   report_timing -file timing_place.rpt
   ```

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

3. Add physical synthesis and routing commands to your Tcl script:
   ```
   phys_opt_design
   route_design
   write_checkpoint -force post_route.dcp
   report_timing_summary -file timing_summary
   ```

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

4. Finally, write out a bitstream:
   ```
   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:

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

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

#Step 3: Disable XDC
generate_target synthesis [get_files clk_wiz_0.xci]
setclk_wiz xdc [get_files -of_objects [get_files 
 clk_wiz_0.xci] -filter {FILE_TYPE == XDC}]
set_property is_enabled false [get_files $clk_wiz_xdc]

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

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
```
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 nonproject_run.tcl
   - or-
   - > vivado -mode tcl -source nonproject_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. 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_synthetic.dcp

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

---

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

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

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