Vivado Design Suite Tutorial

Designing with IP

UG939 (v2015.1) April 1, 2015

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
04/01/2015	2015.1	Validated with release.

Table of Contents

Revision History 2

Designing with IP Overview 5
Introduction 5
Tutorial Description 7
Hardware and Software Requirements 8
Locating and Preparing the Tutorial Design Files 8

Lab 1: Customizing the FIFO Generator 10
Introduction 10
Step 1: Opening the Project 10
Step 2: Customize the FIFO Generator 11
Step 3: Generate Output Products 15
Step 4: Instantiating IP into the Design 18
Step 5: Synthesizing the Design 21
Conclusion 24

Lab 2: Creating and Managing Reusable IP 25
Introduction 25
Step 1: Starting a Manage IP session 26
Step 2: Customizing the FIFO Generator 28
Step 3: Customize the Clocking Wizard 33
Step 4: Use Third-Party Simulators 35
Step 5: Add Additional IP 38
Step 6: Use Third-Party Synthesis Tools 39
Step 7: Create a Netlist Project 39
Conclusion 44

Lab 3: Scripting the Project Mode 45
Introduction 45
Step 1: Creating a Project 46
Step 2: Adding RTL Source Files 47
Step 3: Add XDC Constraints 47
Step 4: Add Existing IP 48

Step 5: Disable the IP XDC Files 50
Step 6: Upgrade an IP 51
Step 7: Setting up Design Runs for IP 52
Step 8: Launching Synthesis and Implementation 53
Step 9: Running the Script 54
Conclusion 57

Lab 4: Scripting the Non-Project Mode 58
Introduction 58
Step 1: Reading Design Source Files 59
Step 2: Adding Existing IP 60
Step 3: Disable XDC Files 61
Step 4: Upgrade IP 63
Step 5: Create DCP for IP 64
Step 6: Run Synthesis 65
Step 7: Run Implementation 66
Step 8: Run the Script 67
Conclusion 70

Legal Notices and Important Information 71
Please Read: Important Legal Notices 71

Introduction

IMPORTANT: *This tutorial requires the use of the Kintex®-7 family of devices. You will need to update your Vivado tools installation if you do not have this device family installed. See the Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973) for more information on Adding Design Tools or Devices..*

The Vivado® Design Suite provides multiple ways to use IP in a design. You can customize and add it from the IP Catalog into a project. You can also create and reference a repository of customized IP 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 the following figure, 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.

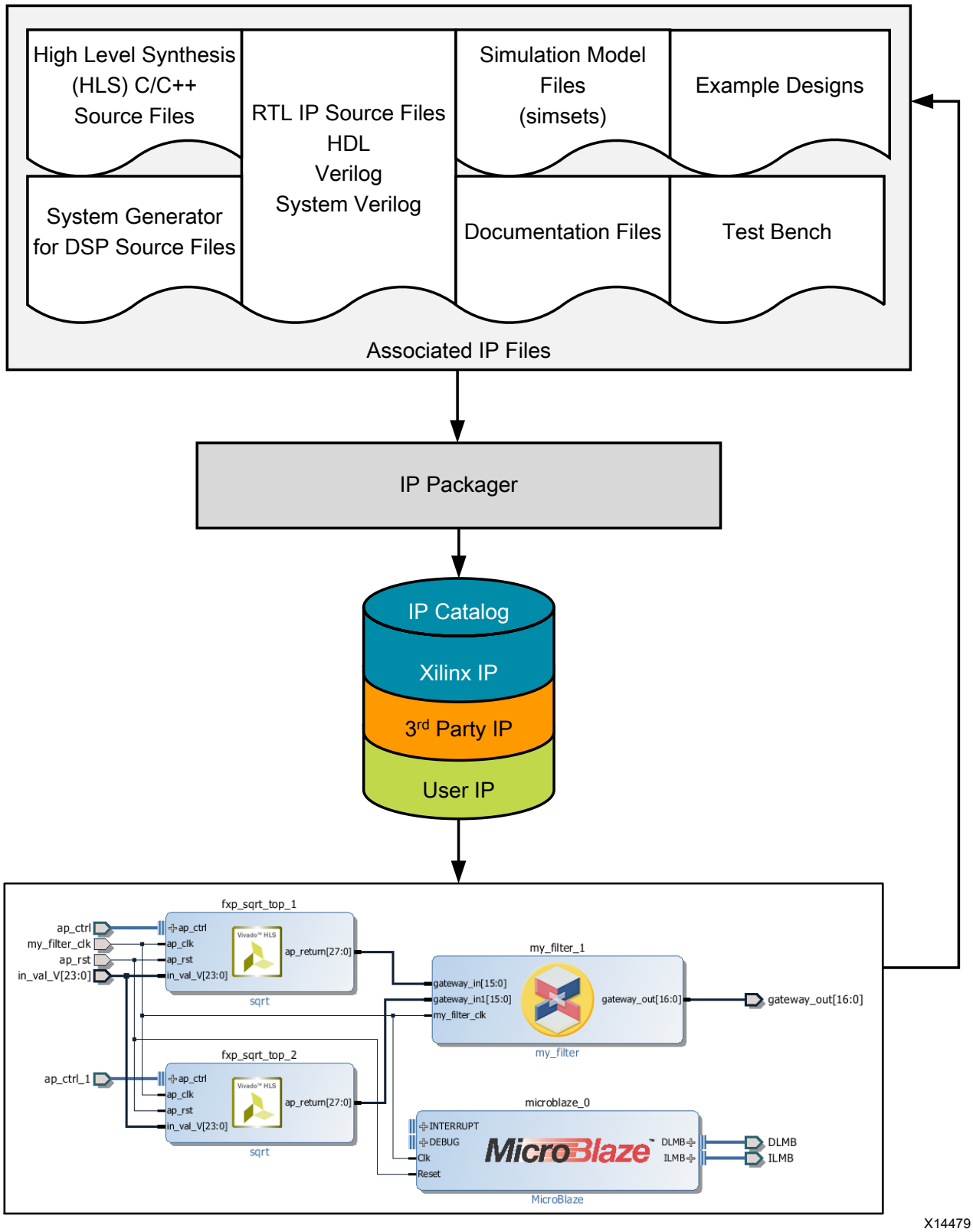


Figure 1: Vivado Design Suite IP Design Flow

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.

VIDEO: *You can also learn more about the creating and using IP cores in Vivado Design Suite by viewing the quick take videos:* Configuring and Managing Custom IP *and* Customizing and Instantiating IP.

TRAINING*: Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use these links to explore related courses:*

- Essentials of FPGA Design
- Embedded Systems Software

Tutorial Description

This tutorial contains several labs as described below:

- **Lab 1**: Open a modified version of the Xilinx `wave_gen` example design that is missing a FIFO; locate and customize the IP in the catalog; and instantiate the IP into the design.
- **Lab 2**: Create and customize IP using the Manage IP flow. Create a project, include an IP from the IP catalog as the top-level source; customize and verify the IP. Optionally, use the customized IP as a black box in a third-party synthesis flow.
- **Lab 3**: Write and run a Tcl script using the Vivado Design Suite to create a project, add IP, upgrade IP, disable IP sources and generate output products including synthesized design checkpoints (DCP).
- **Lab 4**: Write and run a Non-Project Tcl script using the Vivado Design Suite to read in IP sources, upgrade IP, disable IP sources and generate output products including a DCP file.

Hardware and Software Requirements

This tutorial requires that the 2015.1 Vivado Design Suite software release or later is installed.

See the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) for a complete list and description of the system and software requirements.

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 -2015.1 Tutorials on the Xilinx.com website.

1. Download the zipped source files from the Xilinx website:
2. Extract the zip file contents into any write-accessible location on your hard drive, or network location.

   The extracted source directory is referred to as `<Extract_Dir>` throughout this tutorial.

RECOMMENDED: *You will modify the tutorial design data while working through this tutorial. You should use a new copy of* the `<Extract_Dir>` *directory each time you start this tutorial.*

On Windows:

Launch the Vivado Design Suite IDE:

Start > All Programs > Xilinx Design Tools > Vivado 2015.x > Vivado 2015.x[1]

As an alternative, click the **Vivado 2015.x** Desktop icon to start the Vivado IDE.

The Vivado IDE Getting Started page, shown in the following figure contains links to open or create projects and to view documentation.

1 Your Vivado Design Suite installation may be called something other than **Xilinx Design Tools** on the **Start** menu.

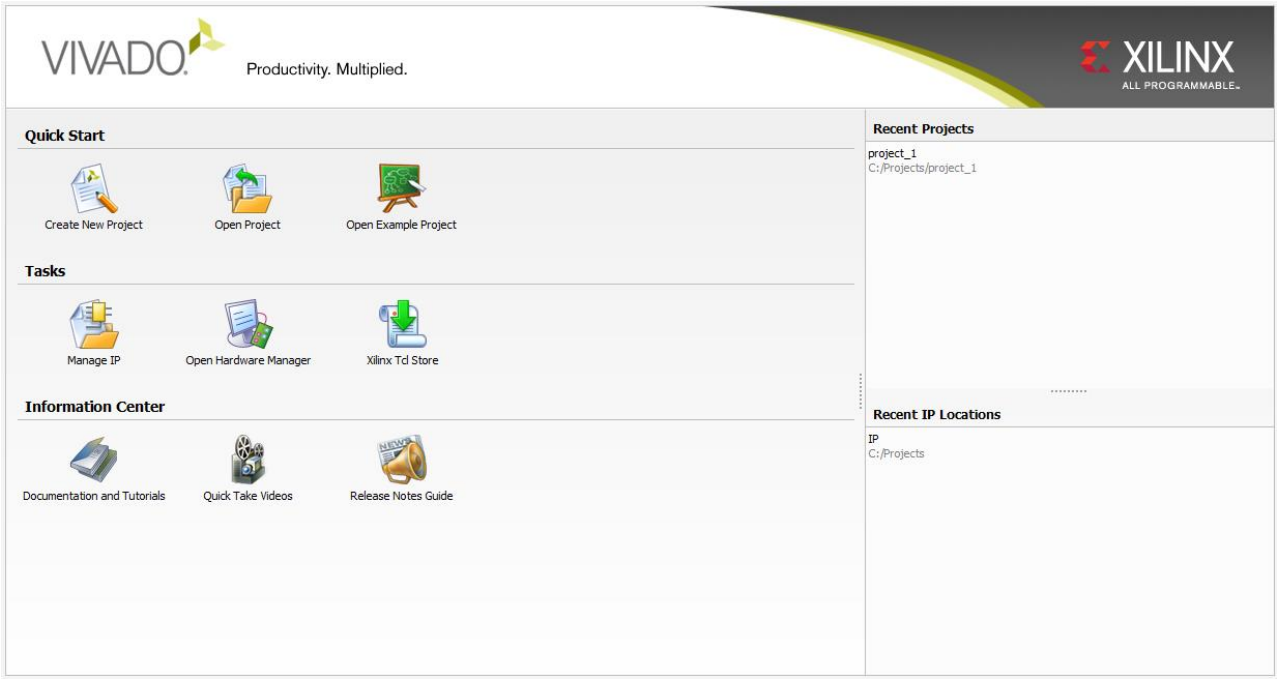


Figure 2: Getting Started Page

Introduction

In this lab you open a Vivado project, and customize the FIFO Generator IP core. You will generate the outpu products for the IP and instantiate it in the design RTL source. Finally, you will synthesize the project.

Step 1: Opening the Project

1. From the Getting Started page, select **Open Project** and browse to: `<Extract_Dir>/lab_1/project_wave_gen_ip`
2. Select `project_wave_gen_ip.xpr`, and click **OK**.

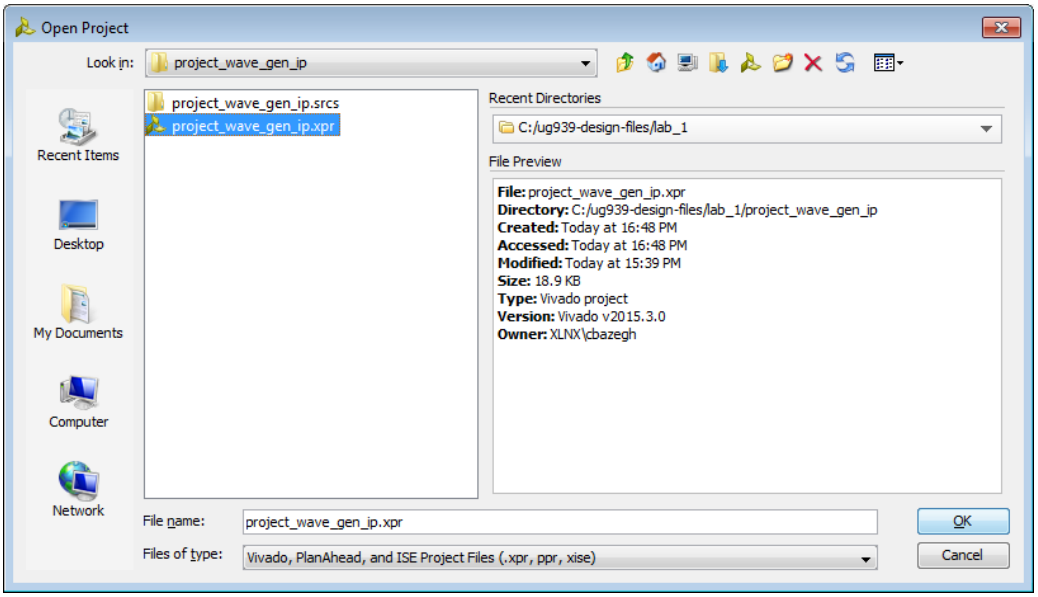


Figure 3: Open Project

The design loads and you see the Vivado IDE in the default layout view, with the Project Summary information as shown in the following figure.

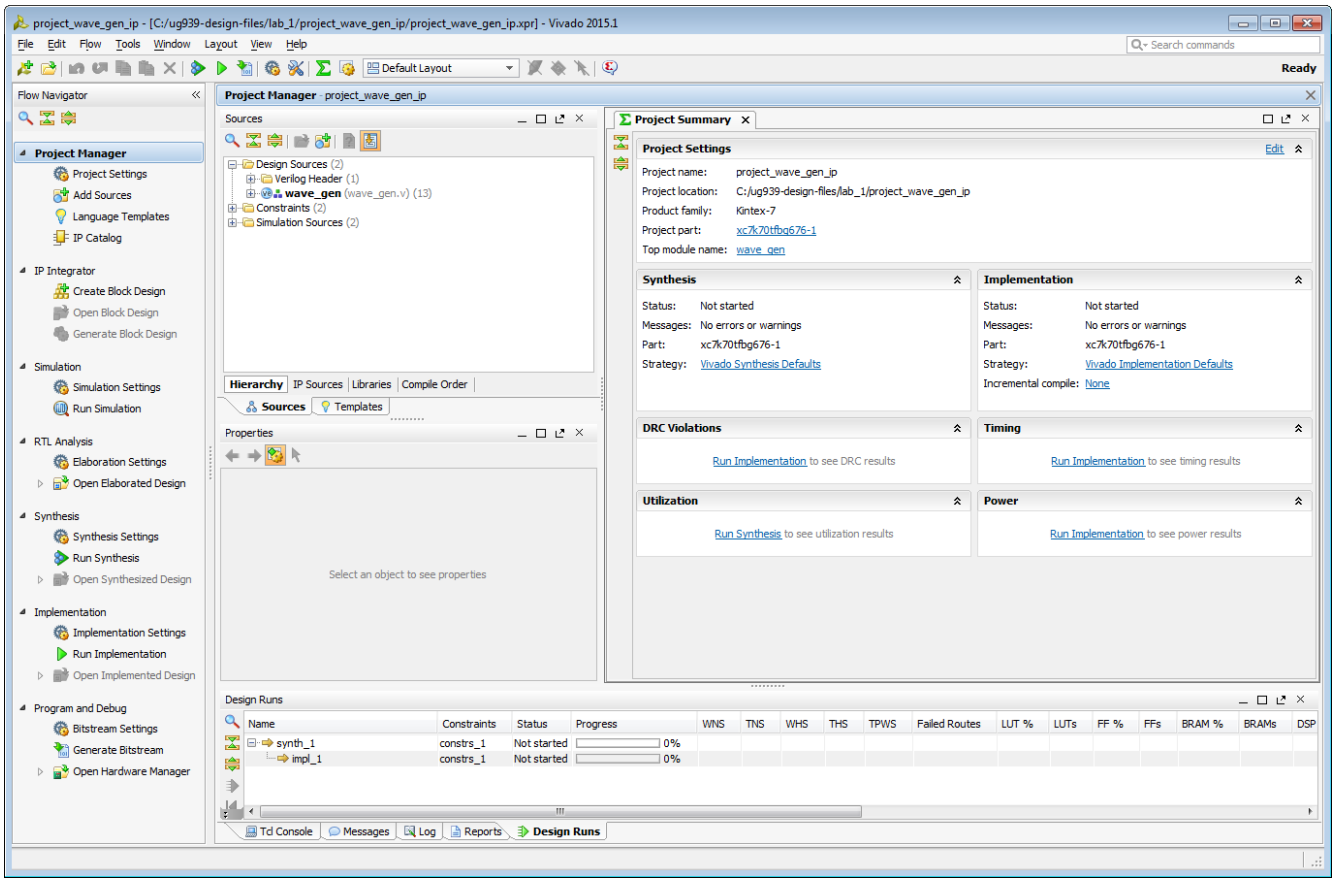


Figure 4: Default Layout

Because 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, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Step 2: Customize the FIFO Generator

1. From the Flow Navigator, select **IP Catalog**. The Xilinx IP Catalog displays in a new tab.
2. 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.
3. Type **fifo** in the search box.

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

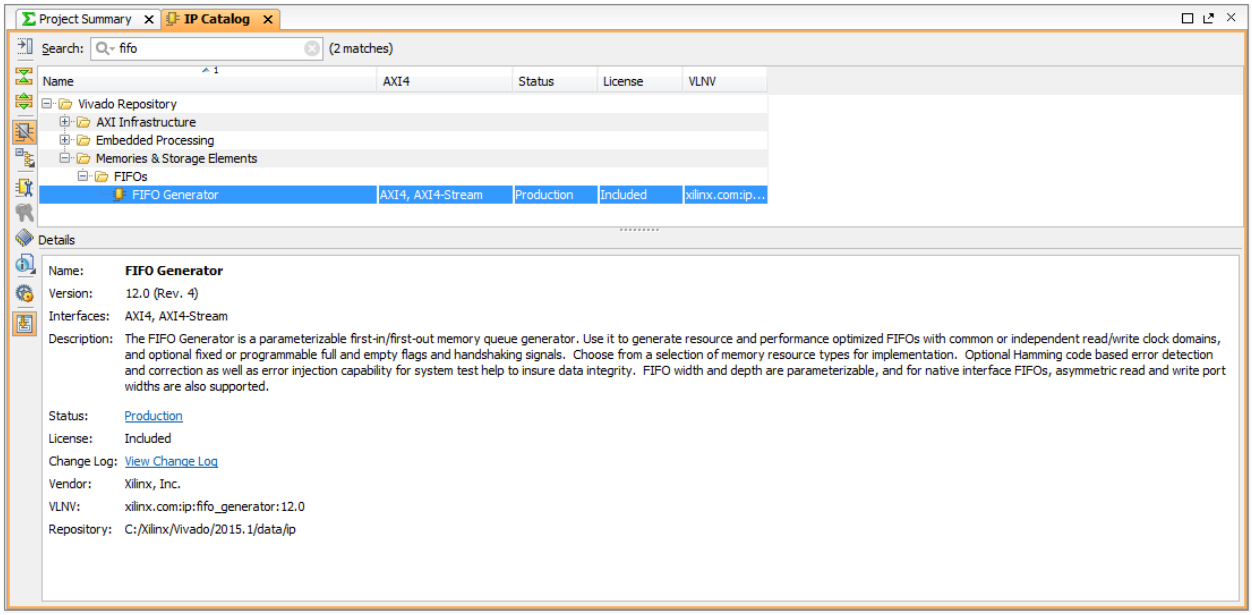


Figure 5: Xilinx IP Catalog – FIFO Core

5. Right-click and select **Customize IP**, or double-click the selected IP.

   The FIFO Generator customization window opens. There is a schematic symbol for the selected core displayed on the left. The schematic symbol changes as you customize the IP.

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

7. If checked, uncheck the **Show Disabled Ports** checkbox to hide unused ports on the symbol.

8. Above the symbol, open the Documentation menu to examine the options for viewing available information.

   The Documentation menu lets you open the PDF file datasheet for the IP, open the change log to review the revision history of the core, or open an Internet browser to navigate to the IP webpage, or view any Answer Records related to the IP.

   The **IP Location** specifies the location to which to save the IP customization (XCI) file and any generated output products. By default, these are saved inside the project directory structure in the `project_wave_gen_ip/project_wave_gen.srcs/sources_1/ip` directory.

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

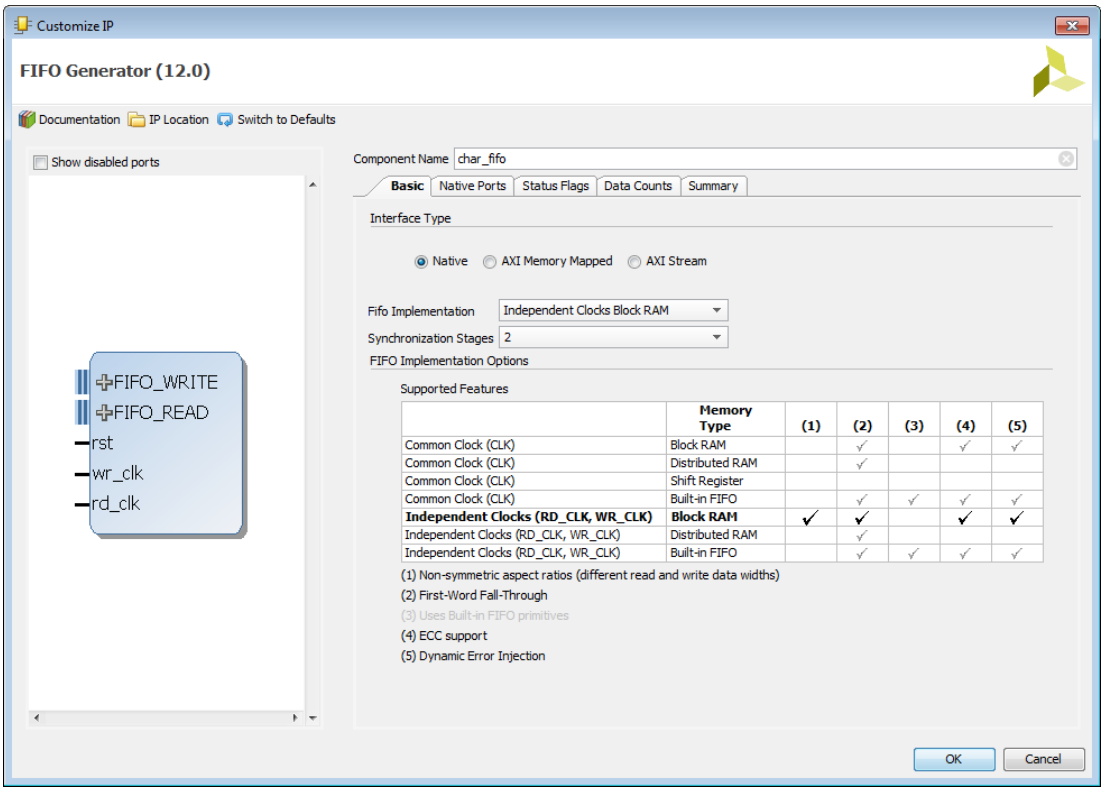


Figure 6: Customize IP

9. In the Customize IP dialog box, change the the following:
   a. **Component Name** to **char_fifo** from the default name.

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

   b. Default Interface Type of **Native**.
   c. **Fifo Implementation**, to **Independent Clocks Block RAM**.

   The customization window should now look like the preceding figure.
10. Select the Native Ports tab to set the **Read Mode**, **Data Port Parameters**, **ECC** and **Output Register** Options, and configure **Initialization**.
    a. Set **Read Mode** to **First Word Fall Through**.
    b. Set the **Write Width** to **8** bits.

       Setting the **Write Width** automatically changes the **Read Width** to match when you click in the **Read Width** field.

    c. Click the **Read Width** field to automatically change it to **8** bits.
11. Leave everything else with the default settings on this tab.
12. 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.

13. Select the Summary tab.

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

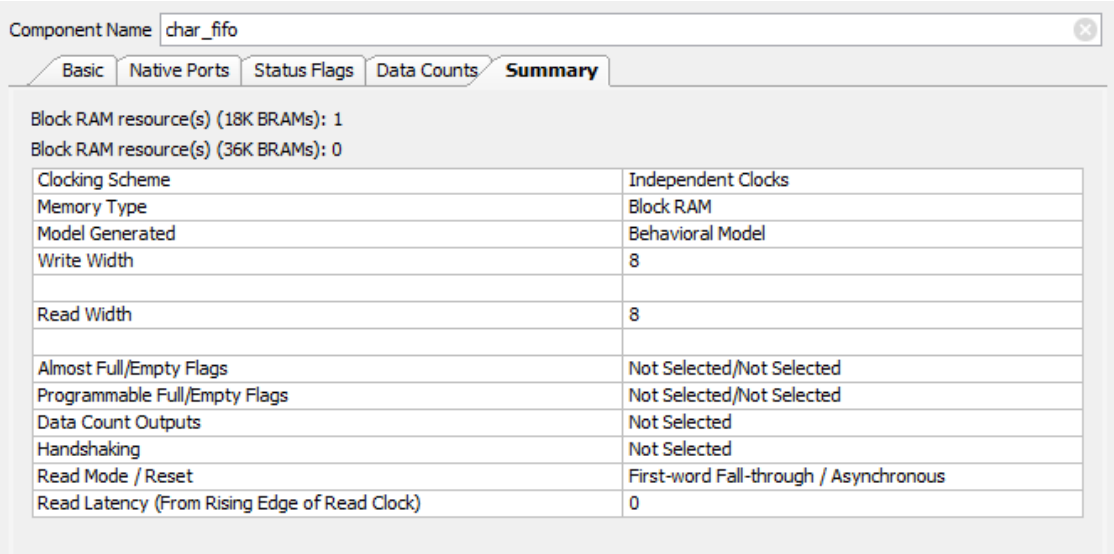


Figure 7: Summary of FIFO Generator Core

14. Verify that the information is correct as shown in the summary, and click **OK** to add the IP customization to your design.

Step 3: Generate Output Products

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

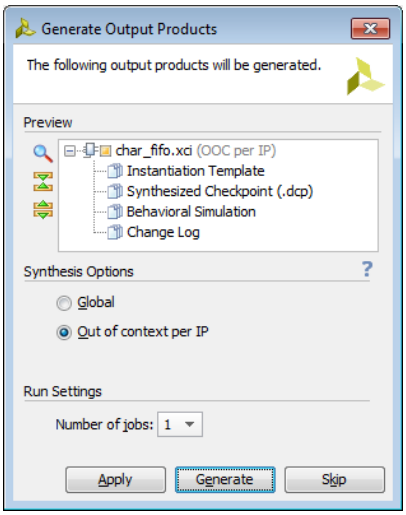


Figure 8: Generate Output Products

1. Click **Generate** to generate the required output products.

   By default, the Vivado IDE generates the Synthesized Checkpoint (DCP) file. To create this DCP, an Out-of-Context Module Run for the **char_fifo** IP is added to the Design Runs window, and the run is launched to synthesize the IP as shown in the following figure.

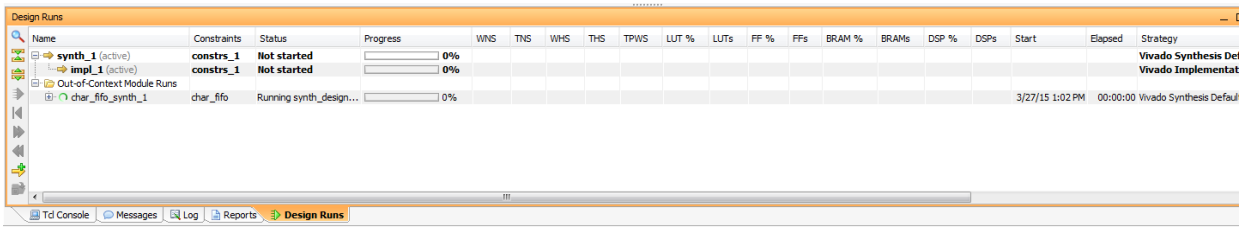


Figure 9: Out-of-Context Synthesis Run

The Out-of-Context (OOC) run creates the DCP for the specified IP customization. This does the following:

- Allows the IP to be complete with regard to synthesis.
- Ensures that the integrity of the core is preserved in the current design.
- Reduces synthesis time for the top-level design in future iterations of the design flow.

For more information on OOC runs, and the use of DCP files, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

IMPORTANT: *Generate the DCP file to reduce synthesis time for the top-level design. Do not change the default synthesis setting for the IP Design Runs.*

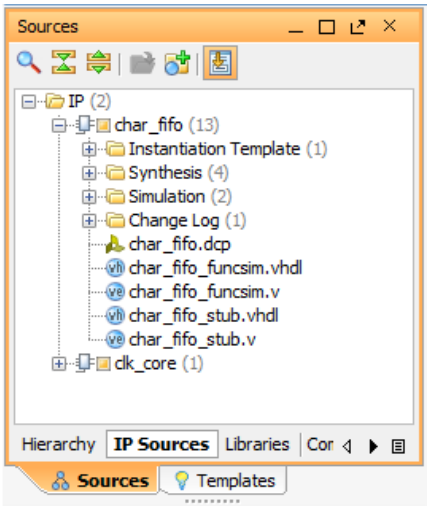


Figure 10: FIFO Generator Sources

The FIFO now appears in the Sources view, as shown in the following figure.

2. In the IP Sources tab of the Sources window, examine the output products produced for the FIFO Generator customization. If you had not generated the output products, they would generate automatically when you launch the top-level synthesis run.

   The FIFO IP is added to the design sources, but is not yet instantiated into your design. It is added at the same level as the top-level `wave_gen` module in the Hierarchy tab, and appears in the `/Block Sources` folder in the Libraries and Compile Order tabs.

   The FIFO Generator customization includes an instantiation template, `char_fifo.veo`, synthesis constraints and VHDL entity and architecture definition, Verilog simulation files, and the synthesized design checkpoint.

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

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

3. Examine the Tcl Console in the Vivado IDE, as shown in the following figure and review the Tcl commands used to add the FIFO Generator core to your project.

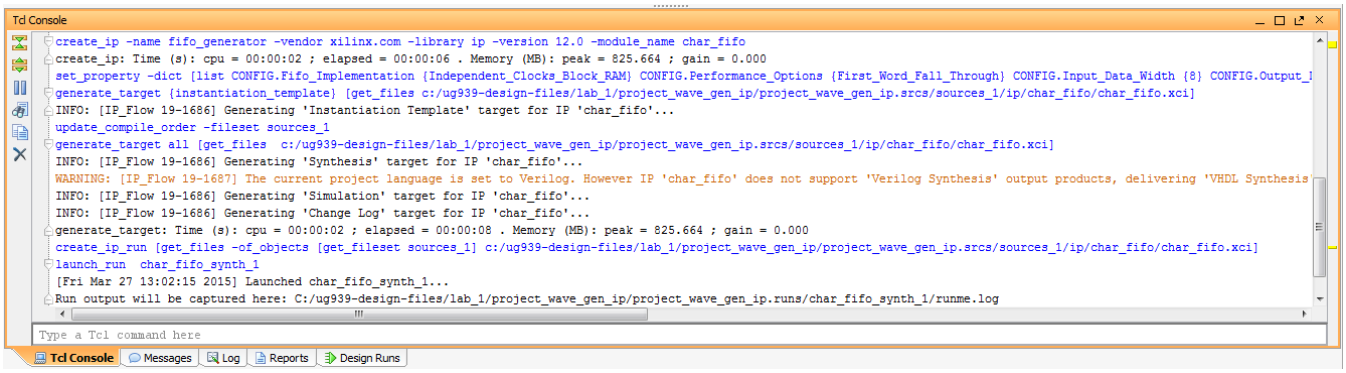


Figure 11: Tcl Console Commands for Adding IP

- The `create_ip` command adds the IP into the current project.
- The `set_property` command sets the various configuration options selected in the Customize IP dialog box.
- The `generate_target` command creates the specified output products for the IP customized.
- The `create_ip_run` command creates the Out-of-Context synthesis run for the IP customized.
- The `launch_run` command runs synthesis on the IP customization to produce the synthesis DCP file, functional simulation netlists, and stub files for third-party synthesis tools to infer a black box for the IP.

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

Step 4: Instantiating IP into the Design

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

1. In the IP Sources tab of the Sources window, expand the **Instantiation Template** and double click the `char_fifo.veo` file to open the template in the Vivado Text Editor.
2. Scroll down to line 57 of the template file, and select and copy the module instantiation text, as shown in the following figure.

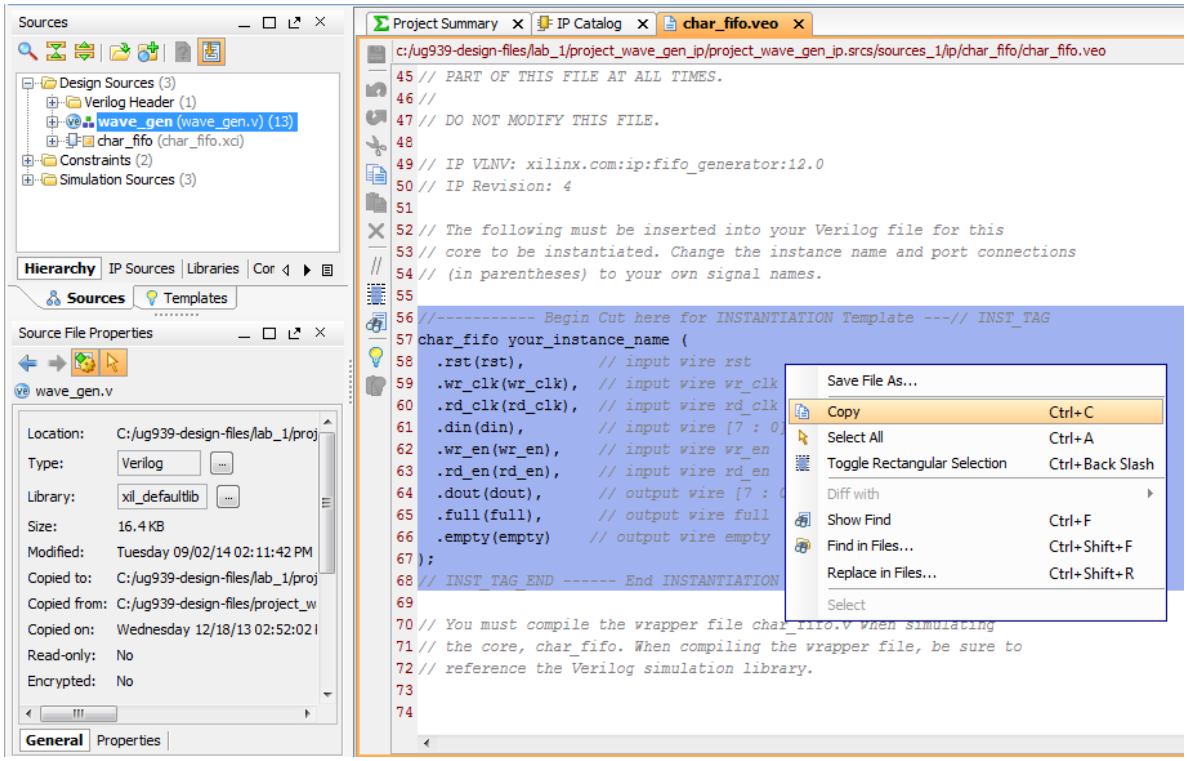


Figure 12: FIFO Core Generator - Instantiation Template

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

3. Open this file for editing from the Hierarchy tab of the Sources view, by double-clicking `wave_gen.v`. The following figure shows the file.

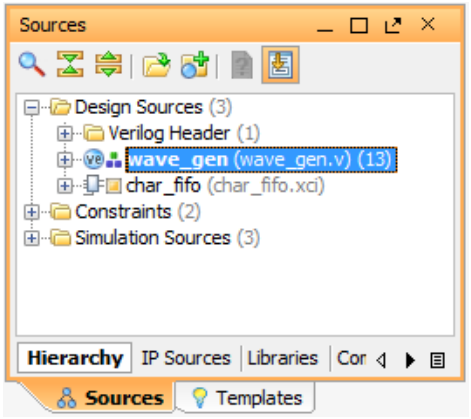


Figure 13: wave_gen.v

4. Go 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 the following figure.

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

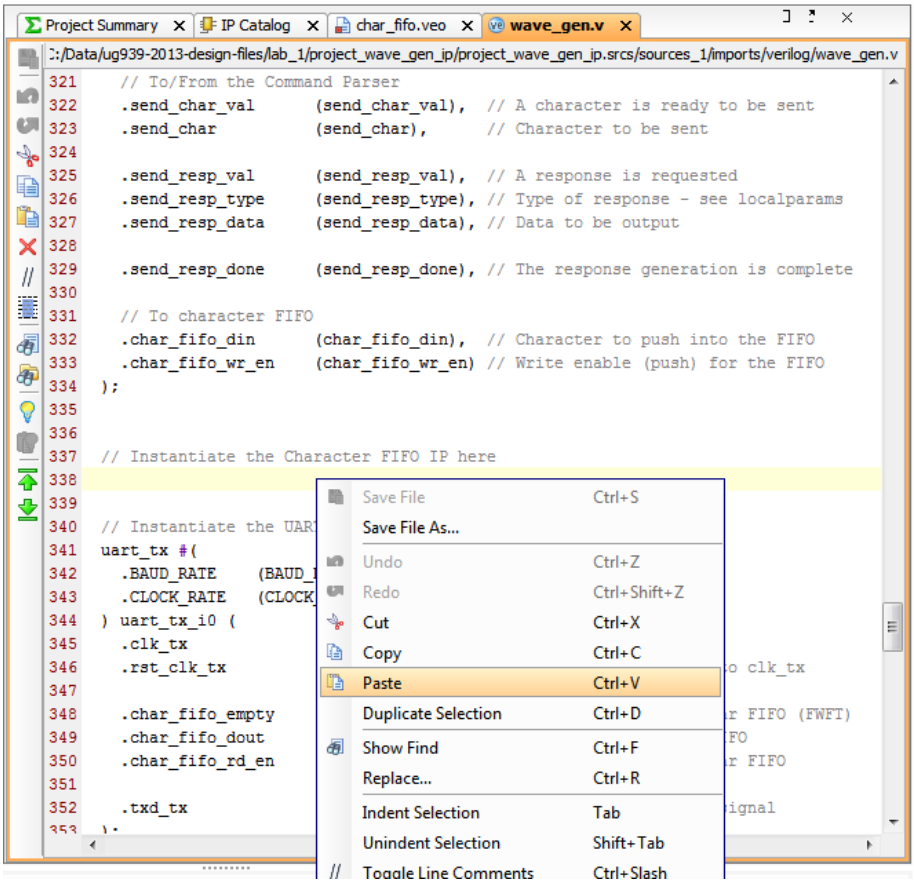


Figure 14: Paste the FIFO IP into the Top-level Design

a. Change the module name from your_instance_name to `char_fifo_i0`.

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

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

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

   Notice that the Hierarchy, Libraries, and Compile Order tabs update to indicate that the IP is instanced into the design, as seen in the following figure.

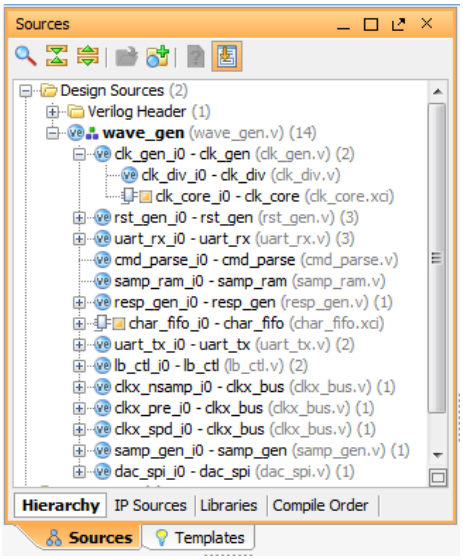


Figure 15: IP Integrated into the Design

Step 5: Synthesizing the Design

With the customized IP integrated into your design, you can now synthesize the design.

Xilinx IP in the IP Catalog of the Vivado Design Suite are delivered as RTL source. This provides the benefit of being able to perform behavioral simulation, which faster than netlist-based simulation. However, synthesizing each IP along with the overall design, with every design iteration, can add significant synthesis time to the project development.

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

The default behavior of the Vivado Design Suite is to generate the necessary output products, including the DCP file, when you create an IP customization.

You can disable generation of the DCP file, as well as defer generating output products until later.

1. In the Hierarchy Tab of the Sources window, look at the `clk_core` IP customization instantiated under the top-level of the design, `wave_gen`, as seen in Figure 15, page 21.

   Notice that the `clk_core` IP cannot be expanded with the '+' icon like you can expand the `char_fifo` IP. This indicates that no output products exist for the `clk_core` when it was instanced in the project.

2. In the Sources window, click the Plus (**+**) icon next to the `char_fifo` IP, as shown in the following figure.

The Vivado tool displays the Show IP Hierarchy dialog box, (following figure), warning you that expanding the hierarchy of very large IP cores can add significant delay to updating the hierarchy in the Sources window.

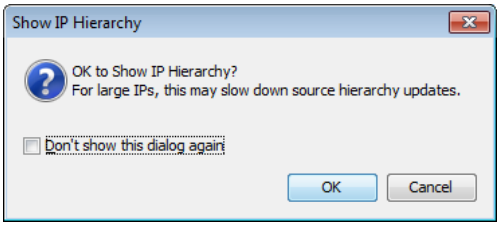
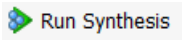


Figure 16: Show IP Hierarchy

3. Click **Cancel** to close the Show IP Hierarchy dialog box.

   If you skip generation of output products when the IP is customized, the Vivado Design Suite will automatically generate the required output products at the point in the design flow they become necessary, such as during synthesis or simulation.

   By default, a DCP file for an IP core is created unless you disable this in the Out-of-Context Settings dialog box. See the *Vivado Design Suite User Guide: Designing with IP* (UG896) for more information. If DCP file generation is disabled, the IP RTL synthesizes along with the top-level design.

4. In the **Flow Navigator**, click the **Run Synthesis** button. 

   The Vivado tool automatically creates a new Out-of-Context Module synthesis run for the `clk_core`, and launches that synthesis run, as seen in the following figure. This synthesis run creates the DCP file for the IP customization.

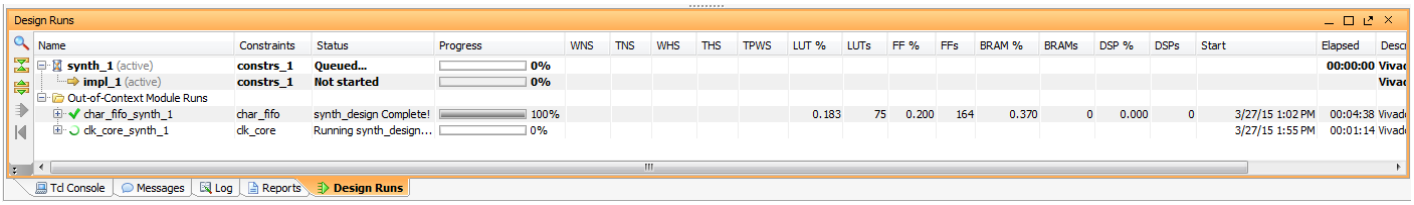


Figure 17: Out-of-Context Synthesis Runs

5. When the `clk_core` synthesis run is finished, you can examine the contents of the IP Sources tab of the Sources window. You will see the output products generated by the Vivado Design Suite for the IP, (following figure).

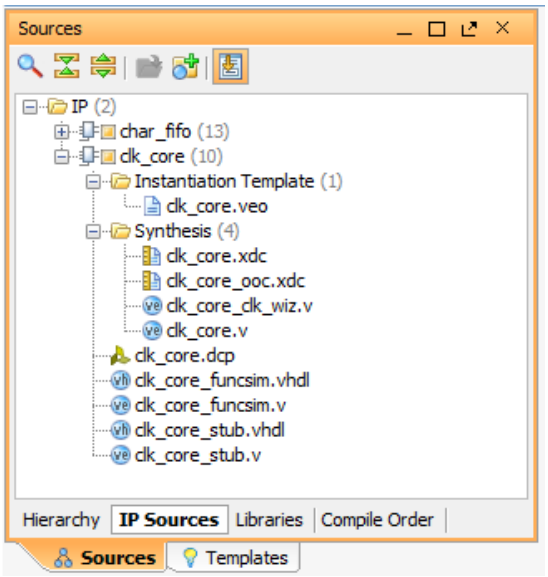


Figure 18: IP Sources

With the required output products for the `clk_core` created, the Vivado synthesis tool runs on the top-level of the design, (following figure). When the top-level of the design is synthesizing, Vivado synthesis infers a black box for the FIFO generator IP (`char_fifo`) and the Clock generator IP (`clk_core`) in the design.

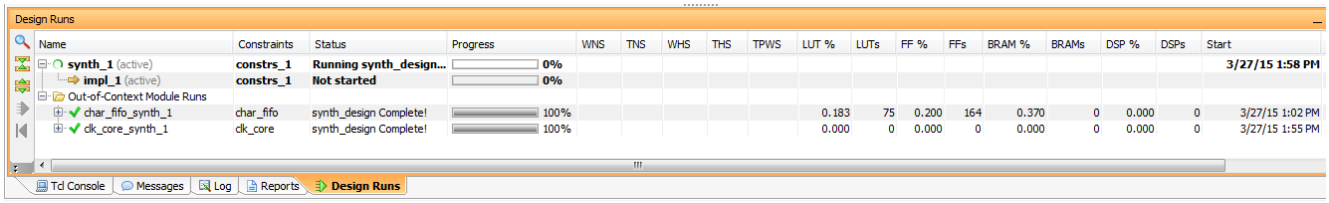


Figure 19: Top-level Synthesis

6. When the Synthesis Completed dialog box opens, select the **View Reports** option and click **OK**.

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

7. Switch to the Reports window and open the Synthesis > synth_1 > Vivado Synthesis Report.

8. Use the **Show Find** command, , to search the Log window for **"blackbox"**.

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

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

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

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

    You can use the IP customizations created in other projects by adding the .xci file as a source. All the output products for the IP, including the DCP, are used automatically.

    If you change the part, you must update the IP and regenerate the output products.

Conclusion

This concludes Lab 1.

You have successfully created a FIFO Generator IP customization, and instanced it in a design. Close the project and exit the Vivado tool, or continue and implement the design to explore further.

In this Lab, you learned how to:

- Open the Vivado IDE.
- Select and customize an IP from the IP catalog.
- Instantiate the customized IP into an HDL design.
- Use some details of the output products required to support the IP in the design flow.

You can add and manage IP in a design interactively within the Vivado IDE, or using Tcl scripting.

Introduction

To simplify revision control, and to support the use of customized IP across multiple projects and designs, you can manage and store the customized IP in a repository, separate from any design projects in which they are used. The IP customization file (XCI), and the output products for synthesis, simulation, and other output, are contained together in a unique directory.

You can reference these IP customizations in new projects and designs, to simulate, synthesize, and implement the IP as part of the design. Having all the generated output products available also preserves that customized version of the IP for use in a future release of the Vivado Design Suite, even if the IP is updated in the Xilinx IP Catalog. See the *Vivado Design Suite User Guide: Designing with IP* (UG896) for more information on managing IP.

In this lab, you create and verify an IP customization in an IP repository, using the Manage IP flow in the Vivado Design Suite. This flow lets you browse the IP Catalog to create and manage IP customizations for use in either a Project or a Non-Project design flow.

You can create a repository of the customized IP for use in your design(s), managed and maintained outside of a Vivado Project. A special IP project is created at the location specified for the Manage IP flow. This special project facilitates the creation of a synthesis design checkpoint (DCP) and structural simulation models for the IP.

When using an IP customization, in a project or non-project flow, all output products, including a DCP if present, are used in the design flow. The use of IP synthesis DCP file speeds synthesis of the top-level design because the IP have been pre-synthesized. In addition, a stub file is produced for use in third party synthesis tools to infer a black box for IP.

Step 1: Starting a Manage IP session

On Linux:

1. Change to the directory where the lab materials are stored:

   ```
   cd <Extract_Dir>/lab_2
   ```

2. Launch the Vivado IDE by typing:

   ```
   vivado
   ```

   On Windows, launch the Vivado Design Suite IDE:

   Start > All Programs > Xilinx Design Tools > Vivado 2015.x[2]

   As an alternative, click the Vivado 2015.x Desktop icon to start the Vivado IDE.

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

3. On the Getting Started page, click the **Manage IP** link, as shown in the following figure.

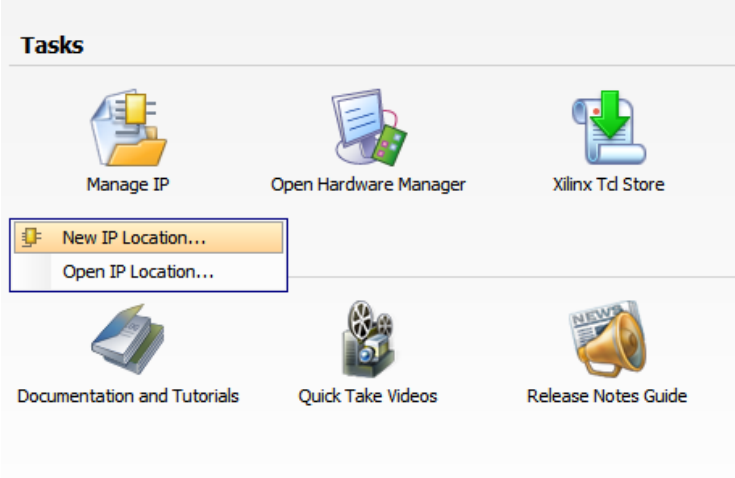


Figure 20: Manage IP from the Getting Started Page

   A menu displays that lets you:

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

4. Select **New IP Location**.

2 Your Vivado Design Suite installation could be called something different than **Xilinx Design Tools** on the **Start** menu.

5. Click **Next** to move to the Manage IP Settings dialog box shown in the following figure (this shows to what you change the default setting).

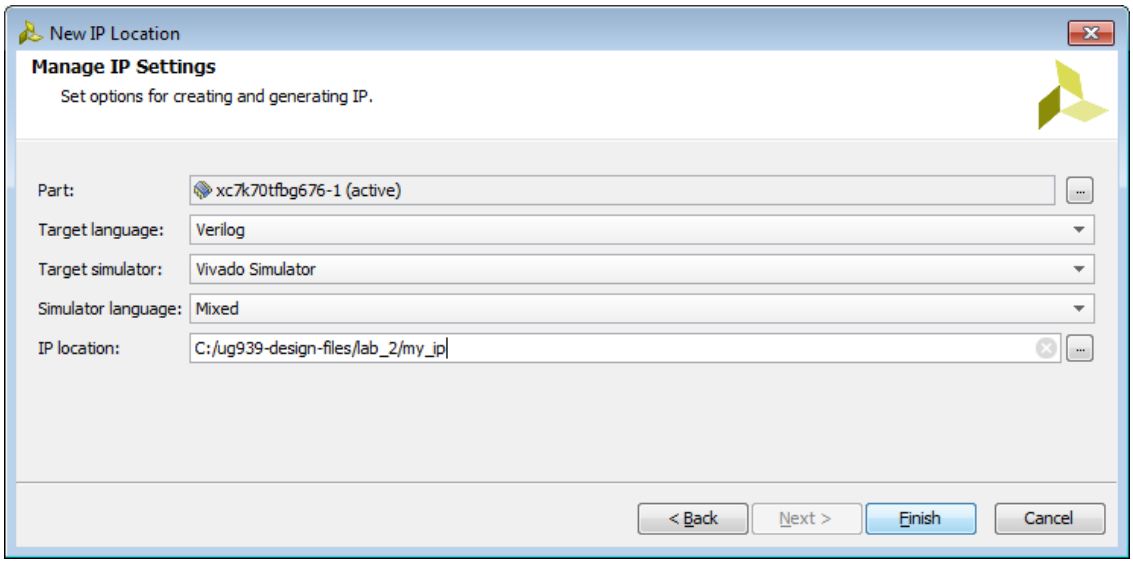


Figure 21: Manage IP Settings

6. Configure the dialog box settings as follows:
   - Part: **xc7k70tfbg676-1**

     Note: Use the browse button to select the specified target part.
   - Target Language: **Verilog**
   - Target Simulator: **Vivado Simulator**
   - Simulator Language: **Mixed**
   - **IP location**: `<Extract_Dir>/lab_2/my_ip`
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. In the IP Catalog search bar, type **fifo**.
2. Double-click the **FIFO Generator** from the **Memories & Storage Elements** group.

   The Customize IP dialog box opens, as shown in the following figure (this is not the original setting, it shows what you are changing the customization to be).

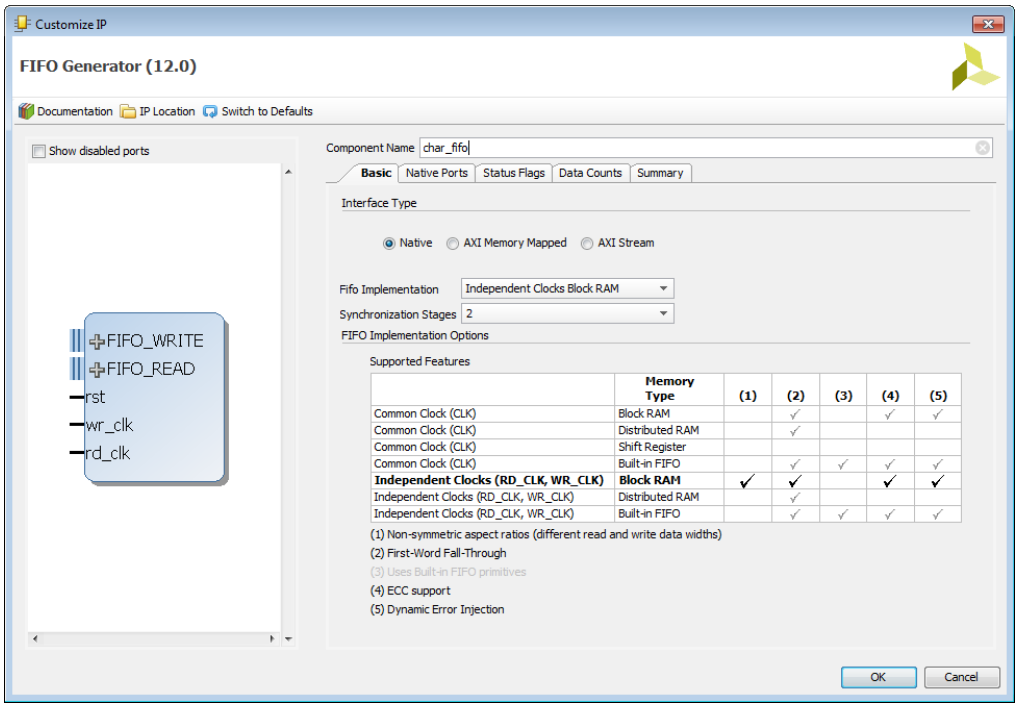


Figure 22: Customize IP Window

TIP: For a complete description of the Customize IP dialog box, and its use, see Lab 1: Customizing the FIFO Generator. This Lab assumes that you have previously completed Lab 1, and are familiar with the concepts covered in that lab.

3. At the top of the Customize IP dialog box, change the **Component Name** to **`char_fifo`**.
4. In the Basic tab:
   - Select the default **Interface Type** of **Native**.
   - From the **Fifo Implementation** 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** 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 the following figure. For this configuration you will see you are using one 18K BRAM.

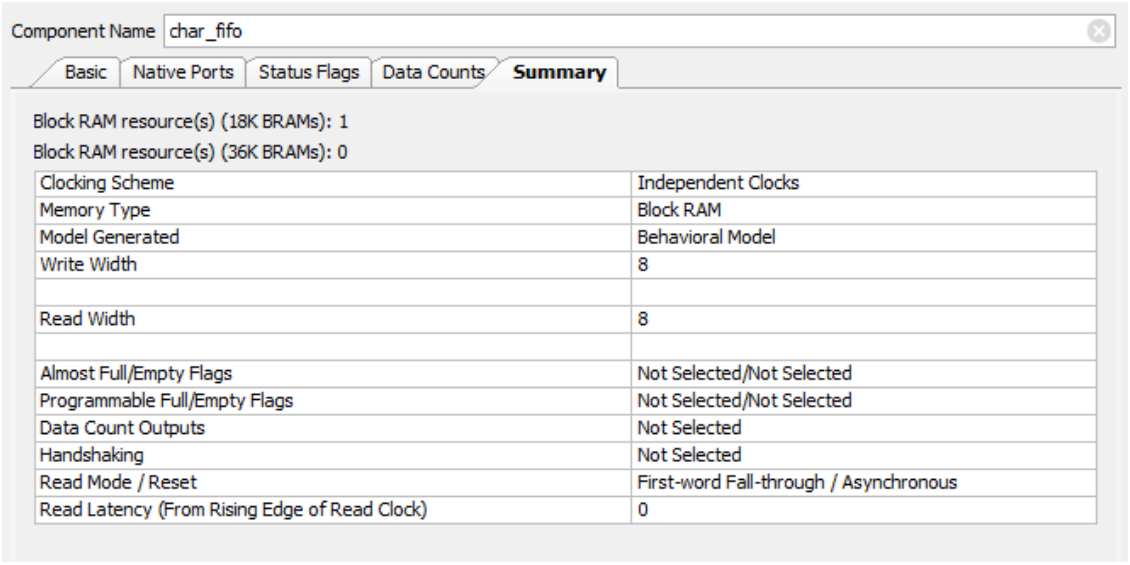


Figure 23: Summary of FIFO Generator Core

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

   The Generate Output Products popup appears, as seen in the following figure. When creating a repository of customized IP using the Managed IP flow, generate all output products, including the design checkpoint (DCP), for each IP. A specific release of the Vivado tools only supports a single version of an IP core. You cannot re-customize or generate outputs for a prior 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 tool. To use older versions of an IP, ALL output products must be available in your custom IP repository.*

8. Click the **Generate** button to create the output products, as shown in the following figure.

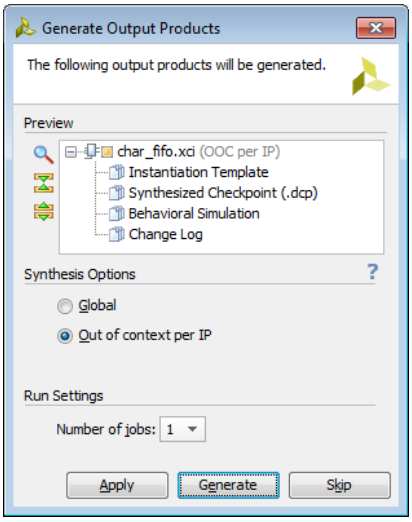


Figure 24: Generate Output Products

To reduce synthesis runtime for a design using one or more customized IP cores, the IP are pre-synthesized as a standalone module by default, with the netlist saved in a synthesized checkpoint file (DCP).

During synthesis of the overall design, the Vivado synthesis infers a black box for the IP.

During implementation, the netlists from the DCP files are linked into the design. This flow is scriptable in Tcl for both Project and Non-Project Mode.

To create the synthesized checkout file, the Vivado IDE creates and launches a corresponding Out-of-Context Module Run, as seen in the following figure.

This run automatically launches, and when synthesis completes, the `char_fifo` directory contains a few additional files:

- `char_fifo.dcp`: The Synthesis Design Checkpoint, which contains a netlist for the IP.
- `char_fifo_stub.v`: A Verilog port module for use with third-party synthesis tools to infer a black box for the IP. The stub contains directives for Synplify Pro to not insert IO buffers, this may require editing for other third-party synthesis tools.
- `char_fifo_funcsim.v/char_fifo_funcsim.vhd`: A Verilog and VHDL netlist for functional simulation of the IP core.

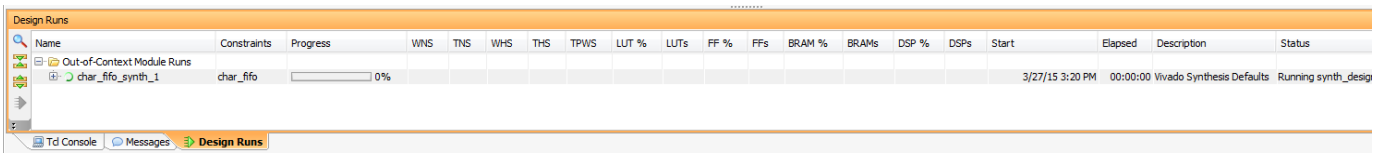


Figure 25: Out-of-Context Module Run to Create the DCP

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 to infer the black box.

The Vivado Design Suite automatically creates this file along with the synthesis design checkpoint (DCP) when generating the output products.

Customized IPs can be referenced using the XCI file 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, see the *Vivado Design Suite User Guide: Design Flows Overview* (UG892).

9. Examine the `<Extract_Dir>/lab_2/my_ip` location.

   Notice that two directories are created, as shown in the following figure.

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

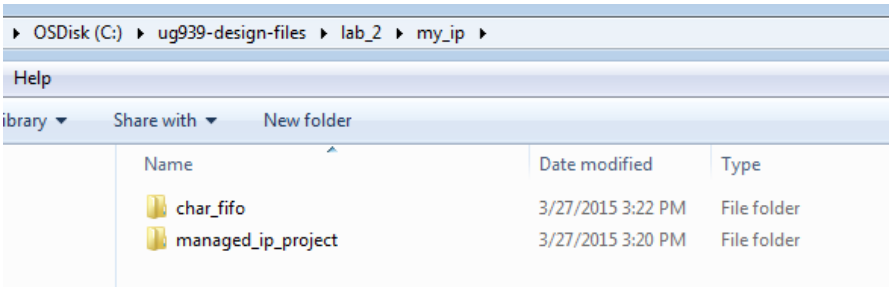


Figure 26: Directory and IP Project

10. Examine the Sources window as shown in the following figure, to see the generated output products in the Manage IP project.

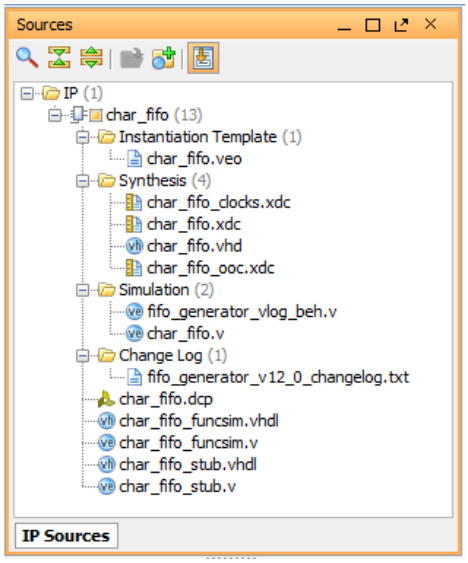


Figure 27: IP Sources

The generated output products for the `char_fifo` core include the Instantiation Template, Synthesis files, Simulation files, synthesized design checkpoint, functional simulation netlists as well as stub files for use with third-party synthesis tools.

> ***Note:*** *The FIFO customization created in this lab is the same as the one from Lab #1. If you want to add existing IP into a Managed IP project you can do so using the* **File > Add Existing IP** *menu.*

Step 3: Customize the Clocking Wizard

Now create a customization for the Clocking Wizard IP.

1. Follow the same processes you used for finding the FIFO Generator IP in the IP Catalog to find and launch a Clocking Wizard IP customization.
2. Change the **Customization Name** to `clk_core`.
3. In the Clocking Options tab, leave all the settings the default except for the Input Clock Information as shown in following figure.
   - Primary Clock Input Frequency: **200**
   - Source: **Differential clock capable pin**

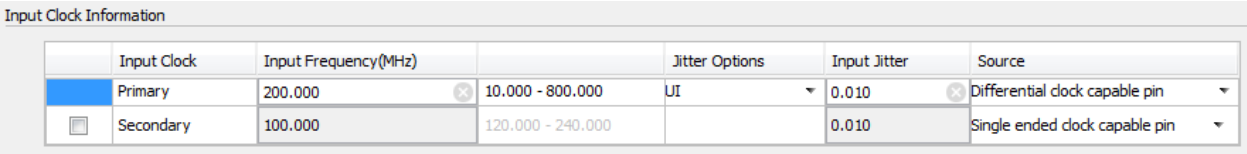


Figure 28: Input Clock Settings

4. In the Output Clocks tab, there are two output clocks required for the design, shown in the following figure.
   - `clk_out1`: **200**
   - `clk_out2`: **166.667**

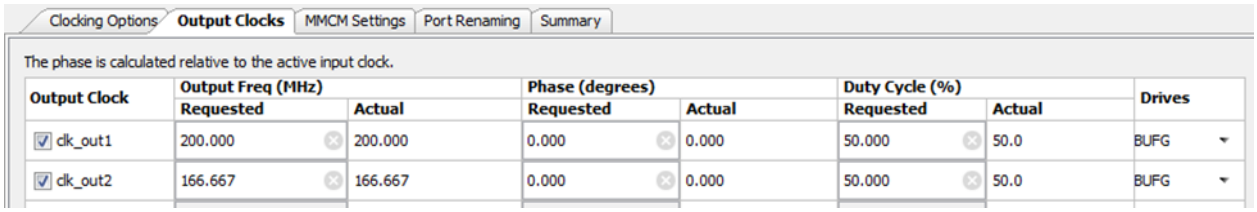


Figure 29: Output Clock Settings

5. In the Port Renaming tab, change the input and output clock names as shown in the following figure:
   - **Input Clock Port Name**: **clk_pin**
   - **Output Clock Port Name** for `clk_out1`: **clk_rx**
   - **Output Clock Port Name** for `clk_out2`: **clk_tx**

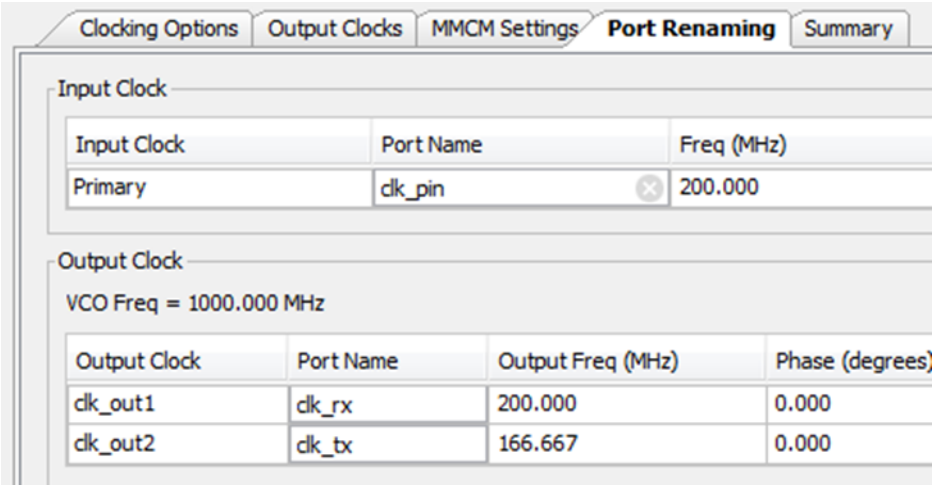


Figure 30: Naming Clock Ports

6. Click **OK** to bring up the Generate Output Product window, and click **Generate**.

   Vivado creates and launches a design run for the Clocking Wizard IP as shown in the following figure.

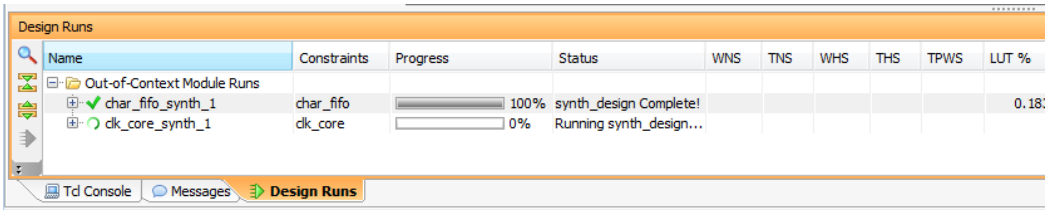


Figure 31: Clocking wizard Design Run

7. Verify the output products and directory created for the Clocking Wizard IP customization (`clk_core`) you just created, just as you did with the customized FIFO Generator IP.

Step 4: Use 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.

Xilinx IP delivered in the Vivado Design Suite are encrypted using industry standard IEEE P1735 encryption. Supported simulator use this encryption standard, and can run behavioral simulation.

The Vivado Design Suite includes the Vivado simulator, for mixed language simulation, as well as direct support for Mentor Graphics® QuestaSim/ModelSim, Cadence Incisive Enterprise Simulator (IES), Synopsys Verilog Compiler Simulator (VCS), and Aldec Riviera-PRO/Active-HDL,

You can use the following Tcl command to generate a Tcl script for the target simulator:

```
launch_simulation -scripts_only
```

You can also use the **Tools > Project Setting >Simulation** to specify the simulator of your choice.

You can change the target simulator, using a Tcl command, as follows:

```
set_property target_simulator <target_simulator> [current_project]
```

The valid options for `<target_simulator>` are: XSim, VCS, IES, Questa, ModelSim, Riviera, ActiveHDL

See the *Vivado Design Suite User Guide: Logic Simulation* (UG900) for more information on using third-party simulators, including creating scripts for running simulations outside of the Vivado Design Suite.

Getting Required Simulation Files

As an alternative to the `launch_simulation` command, you can manually gather the required files to support third-party simulation.

1. To get all files that an IP delivers for simulation, use the `get_files` Tcl command. In the Tcl Console, type the following command:

   ```
   get_files -compile_order sources -used_in simulation -of_objects \
   [get_ips char_fifo]
   ```

   Where:

   `-used_in` Lets you specify files that are marked for use in simulation, or marked for use in synthesis.

   `-of_objects` Lets you extract files that are associated with the specified IP customization file.

   This produces a list of file names, including the full path, required to simulate the IP.

   In this case, the list includes:

   ```
   …/char_fifo/fifo_generator_v12_0/simulation/fifo_generator_vhdl_beh.v
   …/char_fifo/fifo_generator_v12_0/hdl/fifo_generator_v12_0_rfs.vhd
   …/char_fifo/fifo_generator_v12_0/hdl/fifo_generator_v12_0_rfs.v
   …/char_fifo/sim/char_fifo.v
   ```

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

2. To extract the `LIBRARY` property, type the following Tcl command:

   ```
   get_property LIBRARY [get_files char_fifo.vhd]
   ```

   This returns the `xil_defaultlib` library.

3. Use the following Tcl script to print out each file used for simulation, including the path, and its associated library:

   ```
   # Get the list of files required for simulation
   set ip_files [get_files -compile_order sources -used_in simulation \
   -of_objects [get_ips <IP name>]]
   # For each of these files, get the library information
   foreach file $ip_files {
   puts "[get_property LIBRARY $file] $file"
   }
   ```

   In the preceding script, replace `<IP_Name>` with the name of the customized IP to extract files from. In this case, you would use `char_fifo`.

Structural Netlists for Simulation

Depending on which Simulator Language you select during the Manage IP project creation (see the following figure) the results of the `get_files` commands above differs.

With some IP you might not be able to do behavioral simulation if you did not specify support for a **Mixed** simulator and instead selected **Verilog** or **VHDL**.

You would need to run simulations using a structural netlist, which the Vivado Design Suite produces automatically when a synthesized design checkpoint is available.

If the IP can deliver behavioral simulation files based upon your selected simulator language, when generating the output products you see **Behavioral Simulation** listed as an output product as was shown in the following figure.

However, if the IP does not deliver simulation files for the selected simulator language you see **Structural Simulation** as shown in the following figure.

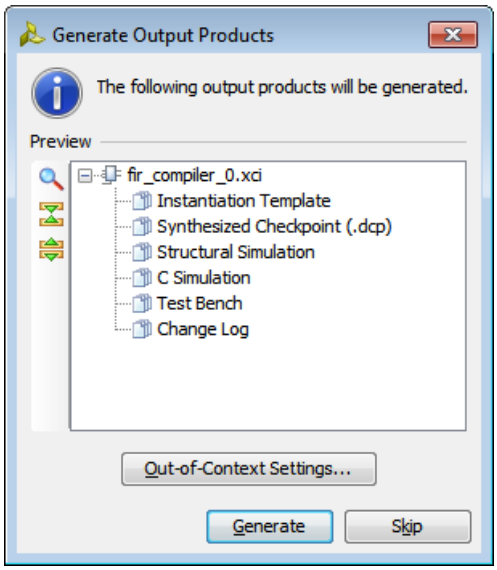


Figure 32: Structural Simulation Output

If you queried the FIR IP customization shown in the following figure for simulation files using the following command:

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

The only file returned is:

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

TIP: *Structural simulation output products, for both Verilog and VHDL, are always created when a synthesized design checkpoint is produced. Querying the simulation files with* `get_files` *will vary depending on whether behavioral simulation was possible with the selected simulator language setting.*

Step 5: Add Additional IP

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

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

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

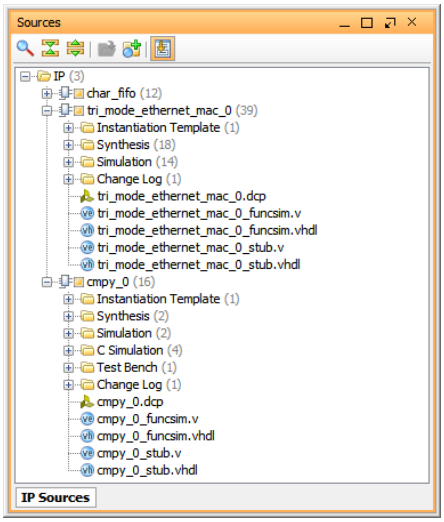


Figure 33: Output products of an IP

When you create an IP customization, the Vivado Design Suite adds an Instantiation Template to the project, even if you choose to skip generating other output products. The Instantiation Template is the minimum output product.

By default, the Vivado Design Suite creates a synthesized design checkpoint for each IP customization added to the project, adding Out-of-Context module runs for each of the IP cores. You can view the progress of these synthesis runs in the Design Run tab, as shown in the following figure.

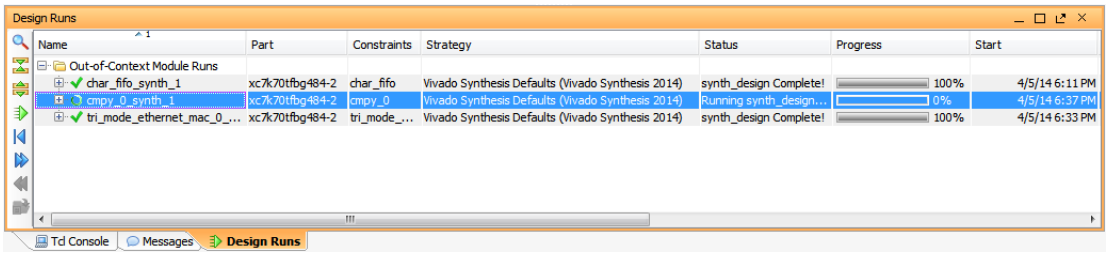


Figure 34: Synthesis Runs for IP

Step 6: Use Third-Party Synthesis Tools

Xilinx IP delivered with the Vivado Design Suite supports synthesis using the Vivado synthesis tool only. The user logic can be synthesized using a supported third-party synthesis tools, such as Synopsys® Synplify Pro or Mentor Graphics® Precision.

The Vivado Design Suite generates a Verilog/VHDL stub file for each IP customization that can be used by third-party synthesis tools to infer a black box for the Vivado IP.

Vivado automatically creates the stub file if a synthesized DCP is generated for the IP, which is the default behavior of the tool. Because it is important that the third- party synthesis tool does not insert I/O buffers for ports that connect to the Vivado IP, the `<ip_name>_stub.v` contains synthesis directives to prevent I/O buffer insertion.

This lab has you use a Manage IP project to create and customize two IP used in the sample design.

- To avoid requiring access to a third-party synthesis tool you are provided an EDIF produced using Synplify® Pro.
- You are also provided two XDC files for constraining the design during implementation.
- An optional stage is provided on how to perform simulations with the IP.
- You create a netlist project and combine the IP with the netlist produced by the third-party synthesis tool.
- A script is provided to demonstrate how you would use a non-project flow to bring the sources together for implementation.

Step 7: Create a Netlist Project

The process to implement the design consists of bringing the netlist from the third-party synthesis tool into Vivado, the two IP customizations created in the Manage IP project, and top-level constraints. The EDIF netlist and the top-level XDC files are provided for you.

1. If open, close the Manage IP project or launch Vivado.
2. From the Getting Started page of Vivado, select **Create New Project** (shown in the following figure) and then select **Next**.

Figure 35: Create New Project

3. In the Project Name dialog box, provide a project name and specify a location.
   - **Project name**: `wave_gen_netlist`
   - **Project location**: `<Extract_Dir>/lab_2/`
4. Leave the **Create project subdirectory** checked, and click **Next**.
5. Set the Project type to be a **Post-synthesis project** and leave the **Do not specify sources at this time** box unchecked.

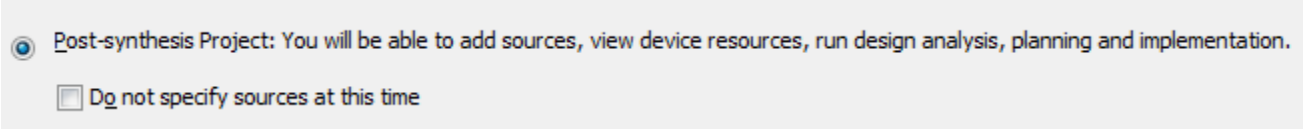


Figure 36: Selecting a Post-Synthesis Project

6. In the Add Netlist Sources window, browse for and select the following files:
   - `<Extract_Dir>/lab_2/sources`, and select `wave_gen.edf`.
   - `<Extract_Dir>/lab_2/my_ip/char_fifo` (or wherever you put the Manage IP project) and select `char_fifo.xci`.
   - `<Extract_Dir>/lab_2/my_ip/clk_core` and select `clk_core.xci`.

   You can either remotely reference the source files or copy them into the netlist project.
7. Ensure that the `wave_gen.edf` is set as top as shown in the following figure, then click **Next** to continue.

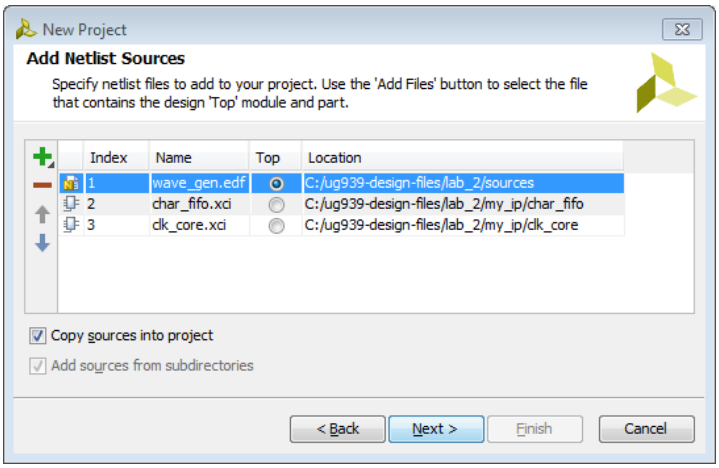


Figure 37: Add Sources to Netlist Project

8. In the Add Constraints window, browse to the `<Extract_Dir>/lab_2/`sources area and select the `wave_gen_timing.xdc` and `wave_gen_pins.xdc` files.
   a. Again, either reference them remotely or add them to your project.
   b. Click **Next** when done selecting the two files.
9. In the Default Part window, select the **xc7k70tfbg676-1** and click **Next**.

   The New Project Summary window opens, as shown in the following figure.
10. Click **Finish**.

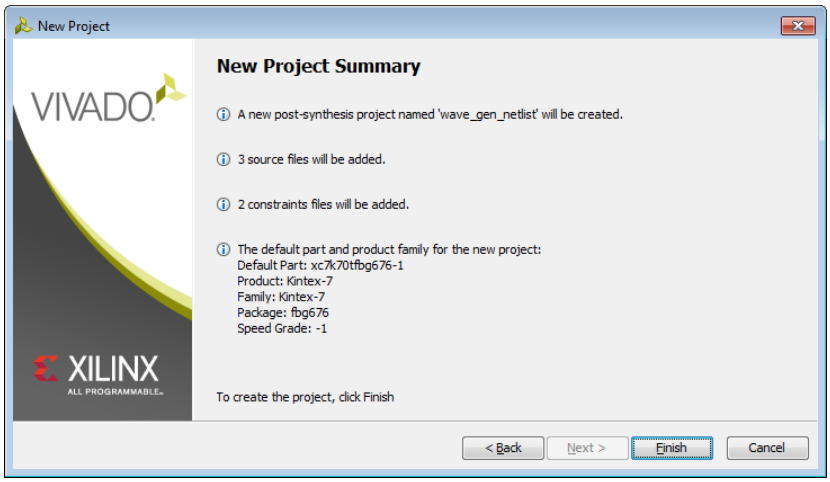


Figure 38: Netlist project summary

The newly created netlist project opens.

11. Expand the sources window as shown in the following figure to see the netlist, IP, and constraints that comprise the project.

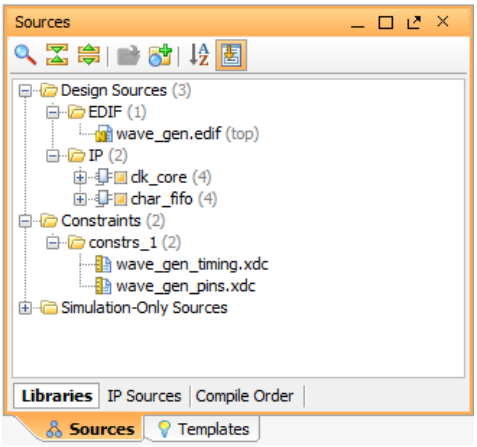


Figure 39: Project Sources

In a netlist project you can check the status of the IP that are present.

12. Go to **Tools >Report IP Status**, you see the IP are up-to-date.

    Even if the IP were not current, because you have all the output products you can proceed through implementation.

13. Select the **Compile Order** tab and select **Implementation** to see the order design source files and constraints are processed in.

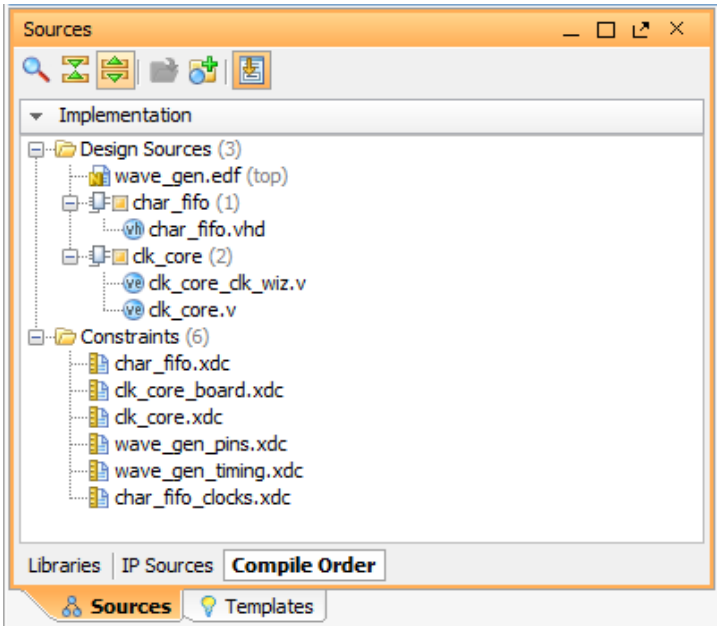


Figure 40: Compile Order

14. In the Tcl Console, type:

    ```
    report_compile_order
    ```

    You see three sections:

    - **Source compile order for implementation**: Lists the EDIF and DCP sources that are used during implementation. The only thing used from the DCPs for the IP are the EDIF netlists.
    - **Source compile order for simulation**: RTL sources for the IP set to be used during implementation. There is no top level simulation files for this project so simulation cannot be performed at this time on the project.
    - **Constraint evaluation order for implementation**: IP XDC output product files as well as the two top-level constraint files. The constraints in the IP DCP files are not used.

15. Launch implementation by pressing **Run Implementation**.

16. After implementation completes, open the implemented design and report timing using the **Report Timing Summary** command in the Flow Navigator to analyze the timing.

This flow can also be scripts. See the `.jou` file for the script commands for a project flow. Continue to the next section for information on how to create a non-project based flow script.

Creating a Non-Project Third-Party Script

You can create a non-project script to read a top-level design netlist, the IP, the design constraint files, and then implements the design.

Modify this script accordingly to point to your source file locations.

```
# Set the target part to be used
set_part xc7k70tfbg676-1

# Read top-level EDIF netlist produced by the third party synthesis tool
read_edif ./sources/wave_gen.edf

# Read the two IP
# Read the IP and use any required output products generated (XDC, DCP, etc.)
read_ip ./my_ip/char_fifo/char_fifo.xci
read_ip ./my_ip/clk_core/clk_core.xci

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

# Link the netlists to build the in-memory design database
link_design -top wave_gen

# Implement the design in Non-Project Mode
opt_design
place_design
route_design

# write out an implemented design checkpoint
write_checkpoint -force wave_gen_post_route.dcp
```

All black boxes for that IP customization are replaced with the netlist from the synthesized DCP when the design is linked, and the design constraints will be applied to each instance.

Conclusion

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

In this lab, you learned 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 IP to share between users and projects and to manage the IP under revision control.
- From the Manage IP Flow you can use the customized IP in a script (Project or Non-Project Mode), from an RTL project, or for inferring a black box for use with a third-party synthesis tool.
- For convenience, Vivado also generates structural netlist Verilog/VHDL files for simulation.
- Perform behavioral simulation with an IP, use the IP in a project, or generate scripts for the Vivado simulator or ModelSim/QuestaSim for standalone simulation.
- Use other third party simulators, such as Cadence® Incisive Enterprise Simulator (IES) or Synopsys® VCS MX, where you either query both the HDL files required for simulation, and the libraries with which the files are associated, or use the `export_simulation` Tcl command.

Introduction

In this lab, you write a Project Mode Tcl script, creating a new project and adding source RTL and customized Xilinx IP.

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 the run status.

In Lab 4: Scripting the Non-Project Mode, you explore creating a Tcl script 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, see 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_3`.

   You start your script by creating a new project using the `create_project` command.

   This results in a new project directory 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_3
   ```

   Now you are ready to create your project.
4. Add the following Tcl command to your `project_run.tcl` script:

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

   This command creates a directory called `my_project` is created, and adds a project named `my_project` is added to that directory. 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 specified by the `-part xc7k70t` option. If `-part` is not specified, the default part for the Vivado release is used.

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

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

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

Step 2: Adding RTL Source Files

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

Because all the HDL files that you need are located in `<Extract_Dir>/lab_3/sources`, you can add the entire directory directly.

Add the following two lines to your script:

```
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 use of `./sources` provides a relative path for locating the source files from where the Tcl script is being run. Recall that the `project_run.tcl` script is being created in the `<Extract_Dir>/lab_3` 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.

TIP: *You can reference source files from their original location by not importing the files by using the* `import_files` *command.*

Step 3: Add XDC Constraints

For this design, two XDC files that are required:

- top_physical.xdc
- top_timing.xdc

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

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

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

By default, all XDC files are used in both synthesis and implementation. However, in this case, you 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:

   ```
   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: Add Existing IP

IMPORTANT: *NGC format files are not supported in the Vivado Design Suite for UltraScale devices. It is recommended that you regenerate the IP using the Vivado Design Suite IP customization tools with native output products. Alternatively, you can use the NGC2EDIF command to migrate the NGC file to EDIF format for importing. However, Xilinx recommends using native Vivado IP rather than XST-generated NGC format files going forward.*

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: An outdated version of native Vivado Design Suite IP with no output products generated.
- FIFO Generator: Vivado 2014.4 version with all output products, including DCP.
- Clock Wizard: Vivado 2014.4 version with no output products.

All of these IP, 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`), which is a legacy IP.

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, the Vivado Design Suite returns two warning messages:

```
WARNING: [IP_Flow 19-2162] IP 'Accumulator' is locked:
* IP definition 'Accumulator (11.0)' for IP 'Accumulator' has a newer major
version in the IP Catalog.
Please select 'Report IP Status' from the 'Tools/Report' menu or run Tcl command
'report_ip_status' for more information.

WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked:
* IP definition 'Block Memory Generator (7.3)' for IP 'blk_mem_gen_v7_3_0' has a
newer major version in the IP Catalog.
Please select 'Report IP Status' from the 'Tools/Report' menu or run Tcl command
'report_ip_status' for more information.
```

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

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

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

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

RECOMMENDED: *When there is a major version change to an IP core in the catalog, changes such as parameter or port name changes, may make upgrading the IP to the latest version requ changes to the design.*

Step 5: Disable the IP XDC Files

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

In a Project Mode you are not required to generate output products manually. The Vivado tools generate output products from IP automatically as needed in the design flow, including the generation of a Vivado synthesis DCP. However, because you are changing a property on the XDC files delivered with the `clk_wiz` IP, you must manually generate the output products to create the constraints file or files to change the property.

Also, because you are going to disable the use of an IP XDC file and provide the constraints during synthesis of the top-level design, you must disable to generation of the `clk_wiz` synthesis DCP (or netlist) as well.

1. To disable the automatic generation of a Synthesis Design Checkpoint (DCP) file add the following to your script:

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

   Now when the design is synthesized, an Out-of-Context Module (OOC) synthesis run is not automatically created and launched for the `clk_wiz_0` IP. Instead, the `clk_wiz_0` IP is synthesized as part of the top-level design.

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

   ```
   generate_target all [get_files clk_wiz_0.xci]
   ```

   Multiple output products can be generated by passing a list to the command, such as {synthesis instantiation_template}, or you can generate all the available output products by specifying `all`.

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

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

3. 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.

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

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

   This returns the names of the XDC files that are delivered with the Clock Wizard.

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

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

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

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

Step 6: Upgrade 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:
* IP definition 'Block Memory Generator (7.3)' for IP 'blk_mem_gen_v7_3_0' has a
newer major version in the IP Catalog.
Please select 'Report IP Status' from the 'Tools/Report' menu or run Tcl command
'report_ip_status' for more information.
```

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 to do this.

1. Check to see if the IP is locked, and store the state in a Tcl variable. Add the following line to your Tcl script:

   ```
   set locked [get_property IS_LOCKED [get_ips blk_mem_gen_v7_3_0]]
   ```

2. Next, 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:

   ```
   set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
   ```

   This returns 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. 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:

```
if {$upgrade != "" && $locked} {
   upgrade_ip [get_ips blk_mem_gen_v7_3_0]}
```

   This results in an upgrade to 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 from the ISE Design Suite, 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; consequently, it can be used in its current form although it does not have any timing constraints.

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

4. Following the steps in 1-3 above, add a sequence of commands to your Tcl script to check if the Accumulator IP is locked, has an available upgrade, and upgrade the IP if so. You will see a warning during upgrade for this IP. This can be ignored, as the warning is that the IP uses upper case for the instantiation template, which the source HDL uses.

```
WARNING: [IP_Flow 19-3501] Upgraded Accumulator from Accumulator 11.0 to
Accumulator 12.0, with warnings. Please review the upgrade log.
```

Step 7: Setting up Design Runs for IP

Now that all the IP have been upgraded to the latest version you can optionally setup and launch out-of-context synthesis runs for the IP. You already configured the Clocking Wizard IP to not use the out-of-context flow in Step 5.

The FIFO Generator IP already had all the output products, including the DCP, generated when it was imported. However, the Block Memory Generator and Accumulator IP have not had any output products generated.

In a project flow, if the IP is current then all the output products, including the DCP, are generated automatically. However, you can also script this step. One reason to manually create and launch design runs for the IP cores in a project is to have the out-of-context synthesis processes run concurrently. By default they are launched serially.

Create and Launch a Design Run for the Accumulator IP.

For a Synthesis Design Checkpoint to be generated an IP Design Run must first be created.

1. Add the following to your Tcl script:

```
create_ip_run [get_ips Accumulator]
```

   A new run is created for the IP, with the name `<IP_name>_synth_1`. The IP design run is launched using the `launch_runs` command.

2. Add the following to your script:

   ```
   launch_runs [get_runs Accumulator_synth_1]
   ```

   The run is now launched and when completed a Synthesis Design Checkpoint is added to the project. If you do not launch the IP run, it launches automatically when synthesizing the top-level.

   If you have multiple IP runs that you created, those runs are launched serially. To launch in parallel you need to use the `launch_runs` command.

   During synthesis of the top level logic a black box will be inferred for the IP. During implementation the DCP will be opened and the netlist read and constraints applied.

   When launching the top level synthesis run it waits automatically for any IP runs to complete so there is no need to put `wait_on_run commands` for each IP run.

   Note: *If you wanted to have a number of IP generate DCPs in parallel you can either:*

   - Create all the runs first and then launch all the runs
   - Create an IP run and then launch it, create another IP run and then launch it

Step 8: Launching Synthesis and Implementation

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

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

1. Add the following line to your Tcl script:

   ```
   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:

   ```
   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_bitsteam` *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, wait for your Tcl script to allow 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 9: Running the Script

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

```
#Step 1: Create Project
cd C:/ug939-design-files/lab_3
create_project -force -part xc7k70t-fbg484-3 my_project my_project

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

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

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

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

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

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

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

#Step 8: Launching Synthesis and Implementation launch_runs synth_1
launch_runs synth_1
wait_on_run synth_1
launch_runs -to_step write_bitstream impl_1
wait_on_run impl_1
```

Sourcing the Tcl Script

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

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

- On Linux:

  Change to the directory where the lab materials are stored:

  ```
  cd <Extract_Dir>/lab_3
  ```

  Launch the Vivado Design Suite Tcl shell, and source a Tcl script to create the tutorial design:

  ```
  vivado -mode tcl -source project_run.tcl
  ```

- On Windows:

  Launch the Vivado Design Suite Tcl shell:

  ```
  Start > All Programs > Xilinx Design Tools > Vivado 2015.x > Vivado 2015.x Tcl Shell[3]
  ```

1. In the Tcl shell, change to the directory where the lab materials are stored:

   ```
   Vivado% cd <Extract_Dir>/lab_3
   ```

2. Source the Tcl script to create the design:

[3] Your Vivado Design Suite installation might be called something different from **Xilinx Design Tools** on the **Start** menu.

```
Vivado% source project_run.tcl
```

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

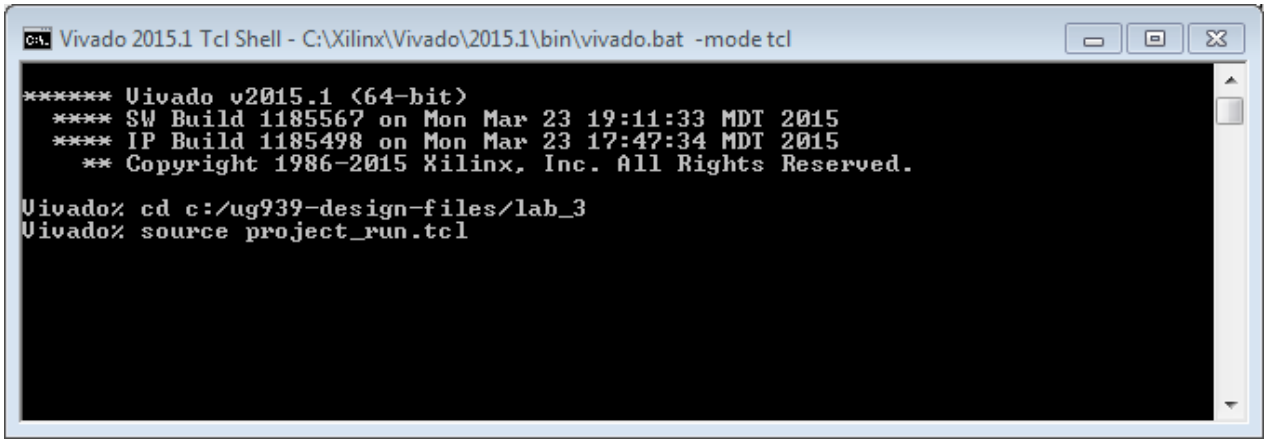


Figure 41: Vivado Tcl Mode and Sourced Tcl Script

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

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

3. To open the GUI from the Tcl prompt type:

```
start_gui.
```

Conclusion

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

- Creating a project and adding HDL sources.
- Adding IP sources to a project, both native XCI files and legacy XCO files.
- Generating IP Output Products.
- Disabling IP Output Products, such as an XDC file.
- Disabling the creation of a Synthesis Design Checkpoint (DCP) file.
- Creating and launching IP design runs.
- Querying the properites of the IP.
- Updating an IP to the latest version.

Lab 4: Scripting the Non-Project Mode

Introduction

In Lab 3: 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 3. The main difference between Project Mode and Non-Project Mode is that there is no project file or directory structure created on disk. Instead, Vivado Design Suite manages the design directly in an in-memory database.

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

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

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

Step 1: Reading Design Source Files

1. Invoke a text editor of your choice, such as Emacs, VI, or Notepad; or launch the Text Editor from within the Vivado IDE.
2. Save a new file called `non_project_run.tcl` in `<Extract_Dir>/lab_4`.

   In Lab 4, you started your project by creating a project; here you will begin by creating an in-memory design, and reading source files. However, you want to first make sure that the Tcl script is in the right location to find source files referenced by the script.
3. Add the following line to your Tcl script to change to the appropriate directory for this lab:

   ```
   cd <Extract_Dir>4/lab_4[4]
   ```

   A default target part is used unless you specify another. This target part is used for all IP that are added to the design, otherwise the default part for the Vivado Design Suite is used. Additionally, setting the target part removes the need to specify a part when synthesizing the top-level design, or any IP for out-of-context synthesis.
4. Add the following to your script to set the part to be used:

   ```
   set_part xc7k70t-fbg484-3
   ```

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

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

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

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

```
C:/ug939-design-files/lab_4
```

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

1. Read the following IP cores into the design:
   - Accumulator: A legacy CORE Generator IP, with no output products or NGC.
   - FIFO Generator: 2014.4 version native IP with all output products, including a DCP.
   - Clock Wizard: 2014.4 version native IP with no output products.
   - Block memory generator: An outdated version of a native Vivado Design Suite IP with no output products generated.

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

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

2. 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
   ```

3. 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
   ```

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

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

   The `-force` option causes the file to be overwritten if it already exists.

IMPORTANT: *Without the* `-force` *option, an error could be returned and the script will quit!*

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

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

IMPORTANT: *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.*

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

Unlike in Lab 3, the warnings related to locked IP do not display when the IP are processed into the design using the `read_ip` command. In Project Mode, the Vivado Design Suite performs checks as an IP is added to a project, resulting in the warning messages seen in Lab 3, Step 4: Adding Existing IP. In Non-Project Mode, the checks are only performed when the IP are processed during synthesis.

Step 3: Disable XDC Files

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

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

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

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

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

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

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

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

Because you copied the XCI file from the source IP repository into a local directory, the output products are written to the local directory.

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

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

Multiple output products can be generated by passing a list to the command, such as {synthesis instantiation_template}.
To disable the XDC files delivered with the Clock Wizard, you need the names of the files. You can query the XDC file(s) that are delivered with an IP, by using the `get_files` command with the `-of_objects` and `-filter` options.

---

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

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

This returns the names of the XDC files delivered with the Clock Wizard.

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

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

The XDC files delivered with `clk_wiz` IP 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 4: Upgrade IP

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

```
WARNING: [IP_Flow 19-2162] IP 'blk_mem_gen_v7_3_0' is locked:
* IP definition 'Block Memory Generator (7.3)' for IP 'blk_mem_gen_v7_3_0'
has a newer major version in the IP Catalog.
Please select 'Report IP Status' from the 'Tools/Report' menu or run Tcl
command 'report_ip_status' for more information..
```

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

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

- To anticipate and prevent this, you can use your script to:
- Determine if an IP is locked.
- Check for a newer version of the IP in the catalog.
- Upgrade an IP if it is locked, and a new version is available.
- Generate the output products for the IP.

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

1. First, check to see if the IP is locked, and store the state in a Tcl variable. Add the following line to your Tcl script:

   ```
   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:

   ```
   set upgrade [get_property UPGRADE_VERSIONS [get_ips blk_mem_gen_v7_3_0]]
   ```

   This returns the VLNV (Vendor/Library/Name/Version) identifier of the upgrade, if there is one available.

   If there is no upgrade, the variable contains 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:

   ```
   if {$locked && $upgrade != ""} {
      upgrade_ip [get_ips blk_mem_gen_v7_3_0]}
   ```

This will result in upgrading the block memory generator IP from the current version in the design, to the latest version in the IP catalog.

4. Now that the IP is current, add the following to your script:

   ```
   generate_target all [get_ips blk_mem_gen_v7_3_0]
   ```

   The Accumulator core is legacy IP created with CORE Generator, rather than native IP created in the Vivado Design Suite. If the IP had the necessary output products, for instantiating the HDL module into a design, for implementation, and for simulation it could be used in its current form.

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

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

Step 5: Create DCP for IP

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

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

Add the following two lines to your script to create DCP for the Accumulator and Block Memory Generator IP:

```
synth_ip [get_ips Accumulator]
synth_ip [get_ips blk_mem_gen_v7_3_0]
```

This results in a DCP file being created and stored in the directory containing the IP XCI file.

Step 6: Run 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:

   ```
   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 -top sys_integration_top
   ```

   Because you created an in-memory project and set the target part, defining the target part is not needed here; however, you must provide the top-level module name with the `synth_design` command.

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

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

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

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

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

   You can also generate any needed reports at this point, such as a post-synthesis timing summary report.

4. 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 7: Run 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.

In this lab, you run the following steps:

- Logic optimization: opt_design
- Placement: place_design
- Physical synthesis: phys_opt_design
- Routing: route_design
- Bitstream generation: write_bitstream

For a complete description of each of these steps, see the *Vivado Design Suite User Guide: Implementation* (UG904).

Between each of these steps, you can generate reports, and write checkpoints to save the design in different stages of implementation.

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

1. Add the following line to your Tcl script:

```
read_xdc ./sources/Constraints/top_physical.xdc
```

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

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

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

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

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

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

4. Finally, write out a bitstream by adding the following:

```
write_bitstream -force sys_integration_top.bit
```

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

Step 8: Run 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:/ug939-design-files/lab_4
set_part xc7k70t-fbg484-3
read_verilog [glob ./sources/HDL/*.v]


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


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


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

```
set locked [get_property IS_LOCKED [get_ips Accumulator]]
set upgrade [get_property UPGRADE_VERSIONS [get_ips Accumulator]]
if {$upgrade != "" && $locked} {
   upgrade_ip [get_ips Accumulator]}
generate_target all [get_ips Accumulator]
#Step 5: Creating DCP for IP
synth_ip [get_ips Accumulator]
synth_ip [get_ips blk_mem_gen_v7_3_0]


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


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

Source the Tcl Script

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

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

On Linux:

- Change to the directory where the lab materials are stored:

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

- Launch the Vivado Design Suite Tcl shell, and source a Tcl script to create the tutorial design:

```
vivado -mode tcl -source nonproject_run.tcl
```

On Windows:

- Launch the Vivado Design Suite Tcl shell:

```
Start > All Programs > Xilinx Design Tools > Vivado 2015.x > Vivado 2015.x Tcl Shell[5]
```

In the Tcl shell:

- Change to the directory where the lab materials are stored:

```
Vivado% cd <Extract_Dir>/lab_4
```

- Source the Tcl script to create the design:

```
Vivado% source non_project_run.tcl
```

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

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

[5] Your Vivado Design Suite installation may be called something different from **Xilinx Design Tools** on the **Start** menu.

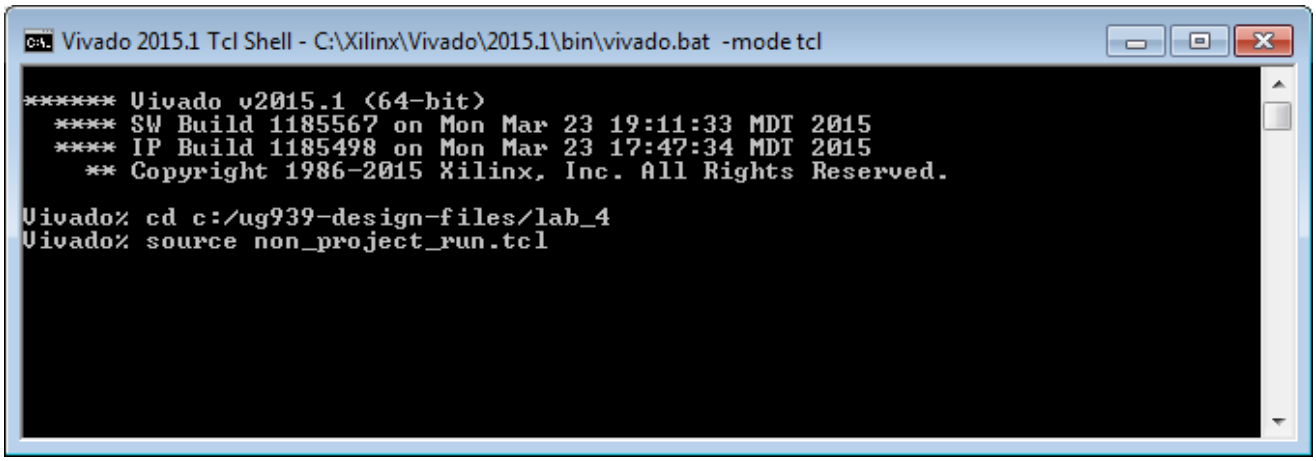


Figure 42: Vivado Tcl Mode and Sourced Tcl Script

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

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

Conclusion

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

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

Please Read: Important Legal Notices

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 Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; 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 such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

© Copyright 2013-2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, 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.