

Using the Tools

2021.1

Abstract

This lab introduces the VHDL development environment within the Vivado® Design Suite.

This lab should take approximately 45 minutes.

Objectives

After completing this lab, you will be able to:

- Launch and navigate the RTL and technology schematic viewers
- Locate and insert snippets from the Language Templates
- Demonstrate the capabilities of the text editor

Introduction

This lab will introduce the bulk of the features required to successfully complete the Designing with VHDL course labs.

A typical FPGA RTL design process is as follows:

- | | |
|---|---|
| 1. Create new project. | 10. Run DRC checker. |
| 2. Add/create design source files. | 11. Synthesize. |
| 3. Integrate systems (embedded or DSP). | 12. Add timing constraints and check timing. |
| 4. Add IP. | 13. Re-synthesize. |
| 5. Elaboration of RTL. | 14. Review design reports. |
| 6. RTL simulation. | 15. Implement. |
| 7. RTL analysis. | 16. Review design reports and perform static timing analysis. |
| 8. Evaluate clocking resources. | 17. Evaluate design with routing resources and/or with the Vivado logic analyzer. |
| 9. Plan I/O pin layout. | |

In order to get their design to work properly, many designers would say that they need to backtrack in this process and repeat certain steps. Many designers would also lock their pins earlier in the design flow. This flow is to illustrate what the lab is leading to.

Here you will use the Vivado Design Suite to create a new design project, create a new VHDL design source file using the Language Templates provided in the Vivado Design Suite (i.e.,

design examples), synthesize the design by using Vivado synthesis, and analyze the synthesized schematic.

The circuit designed in this lab is a simple 8-bit register. You will use the VHDL code of a specific type of D flip-flop, provided in the Language Templates, to complete this design.

Note: Do not concern yourself with the specific VHDL statements—they will be covered in great detail during the course.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform. Many of the labs and demos can be successfully executed in the Windows environment or a native Linux environment as well.

The instructions found in this lab are expressed using the Linux notation. This includes the forward slash ('/') as the hierarchy separator instead of the Windows backslash ('\'). Students who want to run the labs directly under Windows must use the correct hierarchy separator.

Customizable environment variables enable you to tailor your environment for specific machine configurations. The only environment variable (shown below) used in the customer training environment (CustEd_VM) points to the training directory where all the lab files are located. This reduces the amount of typing you need to do when entering directory paths.

This environment variable can be customized according to your specific location and can be set for Linux systems in the `/etc/profile` file and for Windows systems by entering "env" from the search bar.

The following is the environment variable used in the customer training VM:

Environment Variable Name	Description
<code>\$TRAINING_PATH</code>	<p>Points to the space allocated for students to work through the labs. This directory includes prebuilt images and starting points for the labs and demos.</p> <p>The customer training VM sets <code>\$TRAINING_PATH</code> to the <code>/home/xilinx/training</code> directory.</p> <p>Typically, Windows users will install the training directory under C: to keep the path names as short as possible.</p>

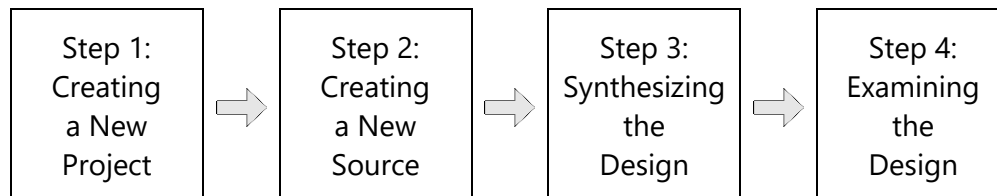
Note: Environment variables are not supported from the Vitis IDE GUI. When using this tool, you must manually replace `$TRAINING_PATH` with the value of the variable, which in the customer training virtual machine, is `/home/xilinx/training`. Other tools, such as the Vivado Design Suite, will properly expand the environment variable.

Additional note about environments: Both the Vivado Design Suite and Vitis platform offer a Tcl environment. The contents of this environment are NOT preserved with the project. When the

tools launch, they start with a pristine Tcl environment with none of the procs or variables remaining from a previous launch of the tools.

This means that if you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool (and perhaps even reopen the last project), you will need to source the Tcl script again and set any variables that the lab requires.

General Flow



Creating a New Project

Step 1

Here are two popular mechanisms for opening the Vivado Design Suite.

1-1. Open the Vivado Design Suite.

1-1-1. Click the **Vivado** icon (🚀) from the taskbar.

OR

Open a Linux terminal window (press <Ctrl + Alt + T>) and enter the following command:

```
[host]$ source /opt/Xilinx/Vivado/2021.1/settings64.sh
[host]$ vivado
```

Note: The customer training environment (CustEd_VM) sets the Vivado Design Suite install path to /opt/Xilinx/Vivado. If the Vivado Design Suite is installed in a different location in your environment, use that install path.

The Vivado Design Suite opens to the Welcome window. From the Welcome window you can create a new project, open an existing project, or enter Tcl commands directly into the Vivado Design Suite as well as access documentation and examples.

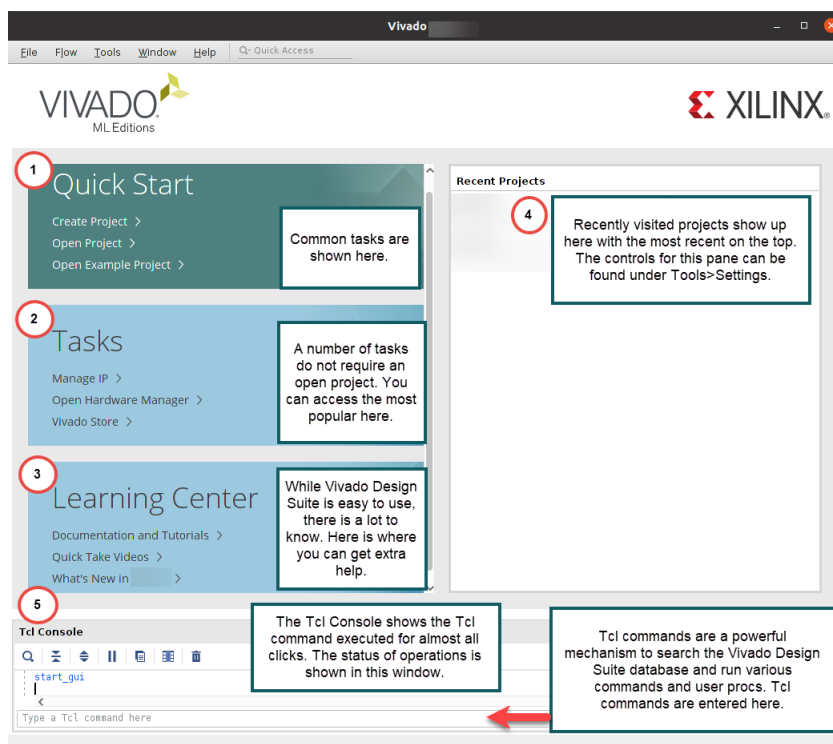


Figure 1-1: Vivado Design Suite Welcome Screen

"Create Project" is the starting point for all designs. Projects contain sources, settings, graphics, IP, and other elements that are used to build a final bitstream and analyze a design. The Create New Project Wizard in the Vivado Design Suite allows you to specify HDL and other project resource files that will be included in the project.

1-2. Create a new, blank Vivado Design Suite project.

1-2-1. Click **Create Project** to begin the process (1).

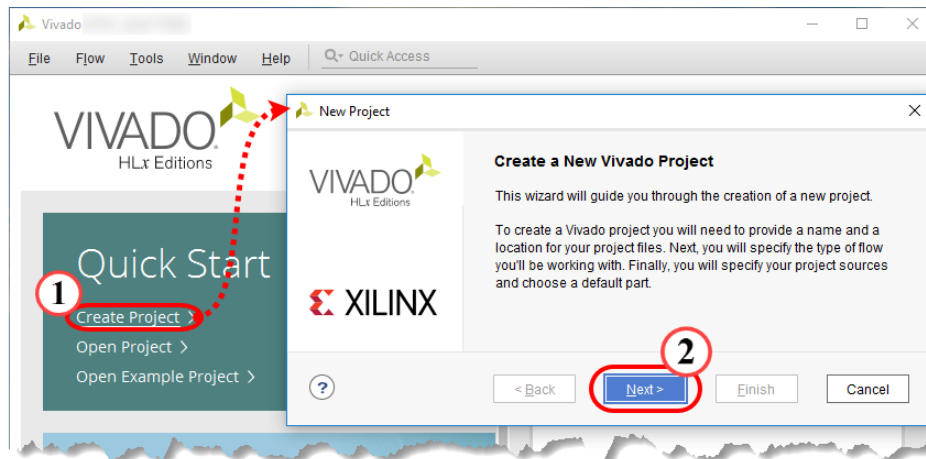


Figure 1-2: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

1-2-2. Click **Next** to exit the introductory dialog box and begin entering in project-specific information (2).

1-3. Describe the various aspects of the project.

1-3-1. Enter **useTools** in the Project name field (1).

1-3-2. Enter the following location in the Project location field (2):

\$TRAINING_PATH/usingTools/lab/KCU105

Important: You need to expand the `path` to its full length as explained in the Introduction section.

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

1-3-3. Deselect the **Create Project Subdirectory** option (3).

Leaving this checked will create an unnecessary level of hierarchy for this lab.

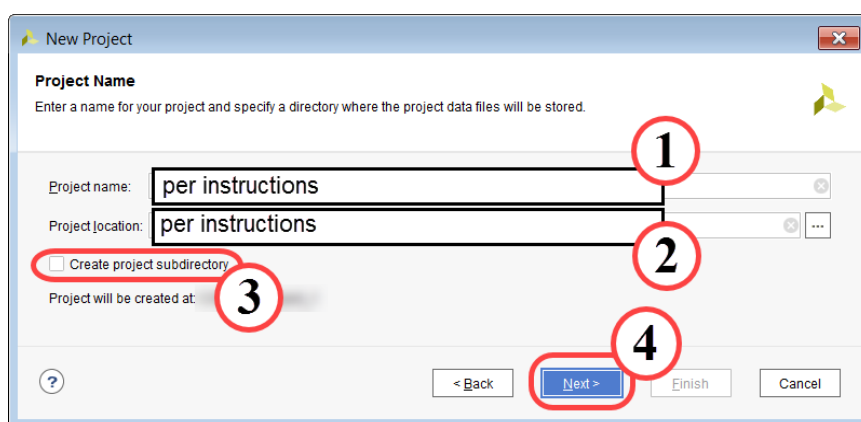


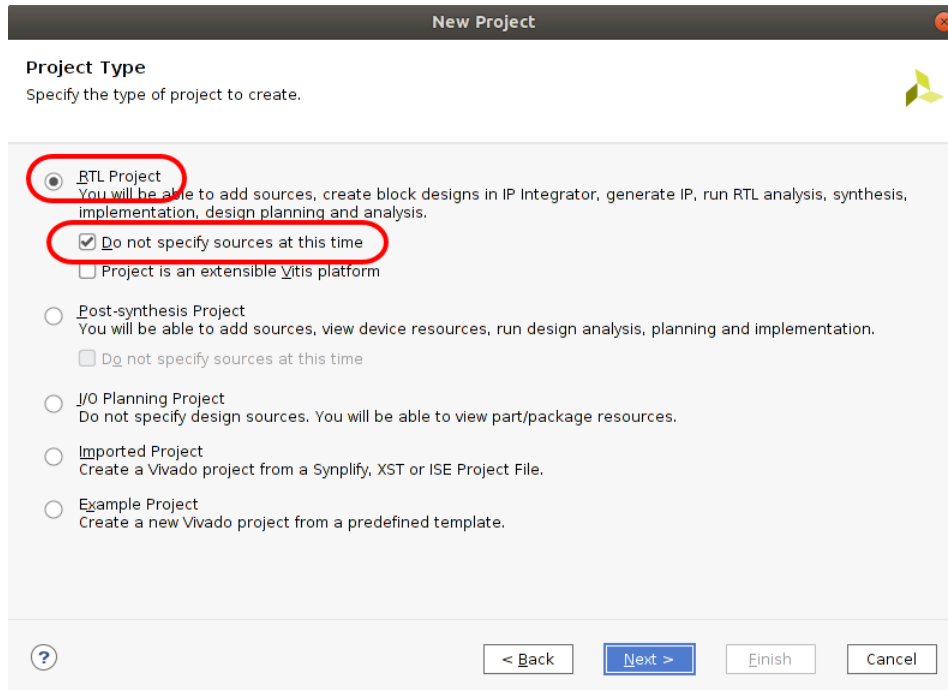
Figure 1-3: Entering the Project Name and Location

1-3-4. Click **Next** to advance to the next dialog box (4).

Here you will specify your project type as either an RTL project or a post-synthesis project. Simply put, an RTL project enables you to add or create new HDL files and synthesize them, whereas the post-synthesis project requires pre-synthesized files. When an empty design is created, an RTL project is used.

1-3-5. Select RTL Project.**1-3-6. Select Do not specify sources at this time**, which creates a blank project.

While existing sources could be entered at this time, you will enter them later so that you can move through this portion of the project creation process more quickly.



New Project

Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☒ **Do not specify sources at this time**

☐ Project is an extensible Vitis platform

☐ **Post-synthesis Project**
You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ Do not specify sources at this time

☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**
Create a new Vivado project from a predefined template.






    

Figure 1-4: Specifying Project Options

1-3-7. Click Next to advance to the target device/platform selection.

1-4. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.

1-4-1. Click **Boards** from the *Default part* area to filter by board rather than by the specific part (1).

1-4-2. Select **xilinx.com** from the Vendor drop-down list in the Filter area (2).

This limits the number of boards seen to those manufactured by the specified vendor.

1-4-3. Select **Kintex-UltraScale KCU105 Evaluation Platform** from the board list.

If you accidentally double-clicked the entry, a web page will open for that board. You can close the browser page.

Note: While this page contains important information and resources for the board, these details are not needed to complete this lab.

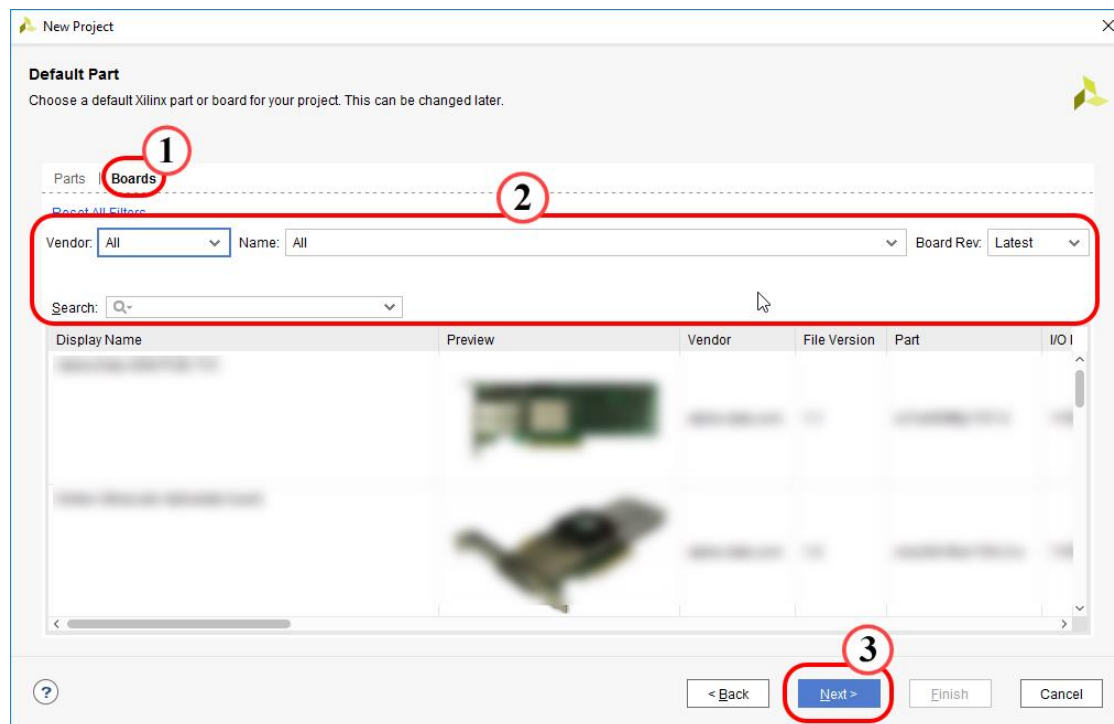


Figure 1-5: Selecting the Board for the Project

1-4-4. Click **Next** to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box. Once the project is created, the project properties can still be edited.

1-4-5. Click **Finish** to accept these settings and build the project.

Your project is constructed and leaves you in the operational portion of the Vivado Design Suite GUI.

1-5. Set VHDL as the target language.

- 1-5-1. Click **Settings** under Project Manager in the Flow Navigator.
- 1-5-2. Click **General** under the Project settings, if not already selected.
- 1-5-3. Select **VHDL** from the Target Language field drop-down list.
- 1-5-4. Click **OK** in the Settings dialog box.

Creating a New VHDL Source

Step 2

Now that the project has been created, it is time to begin creating the design. The design is a simple 8-bit register with an enable.

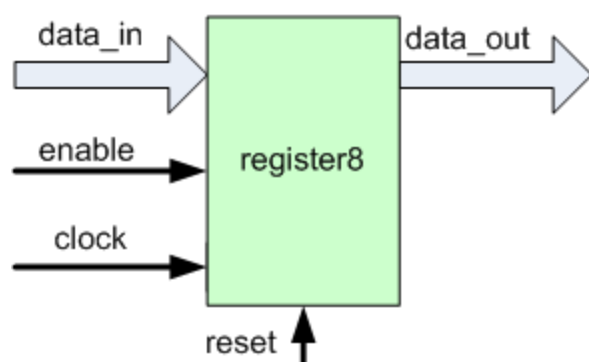


Figure 1-6: register8 Block Diagram

You will begin by creating a VHDL module template, then filling the template with a snippet from the Language Templates and making two simple modifications.

2-1. Create a new HDL source file called *register8*.

- 2-1-1. Select **Add Sources** in the Flow Navigator under Project Manager.

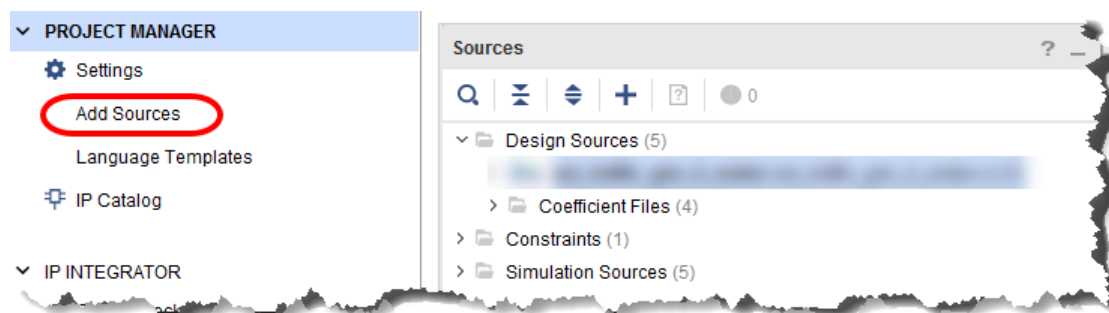
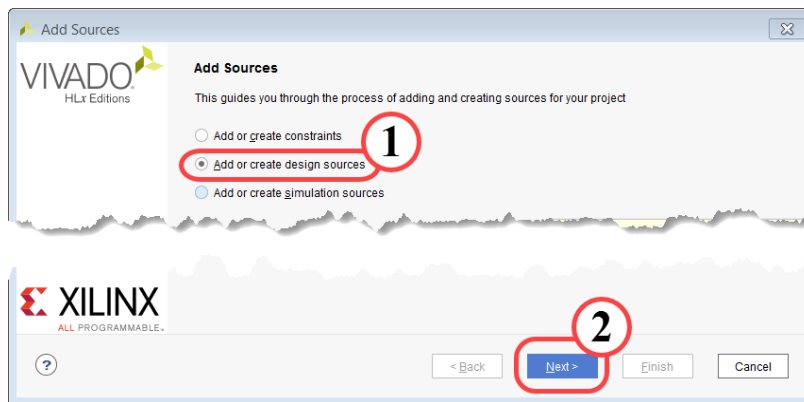
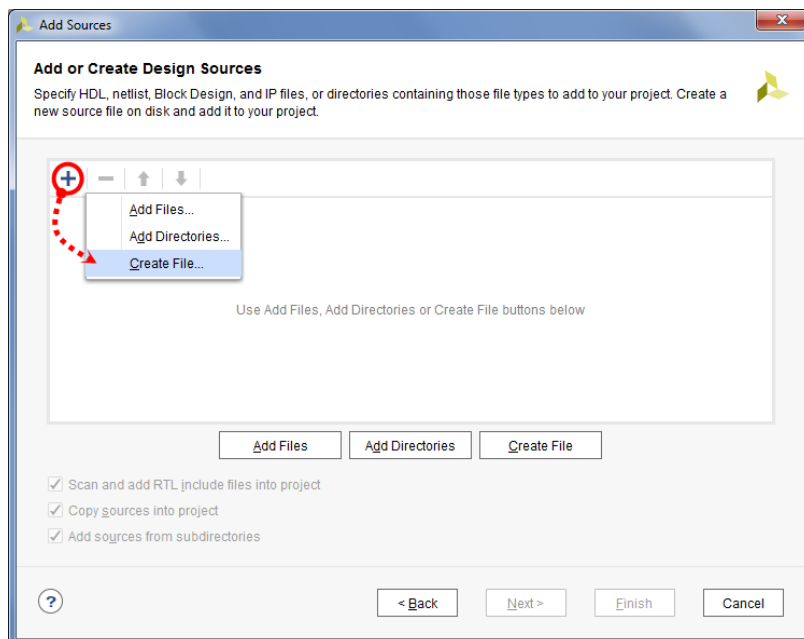


Figure 1-7: Selecting Add Sources

2-1-2. Select Add or create design sources (1).**Figure 1-8: Selecting Add or Create Design Sources****2-1-3. Click Next (2).**

The Add or Create Design Sources dialog box opens.

2-1-4. Click the Plus (+) icon and select Create File.**Figure 1-9: Selecting Create File**

The Create Source File dialog box opens.

2-1-5. Select **VHDL** from the File type drop-down list.

2-1-6. Enter **register8** as the file name.

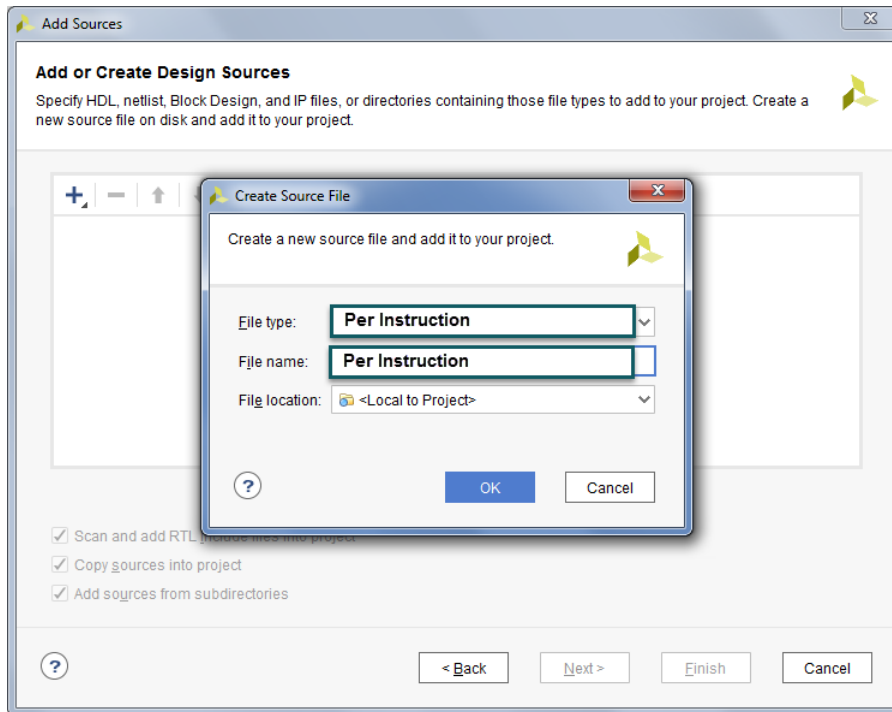


Figure 1-10: Entering File Name and Type

2-1-7. Click **OK** in the Create Source File dialog box.

2-1-8. Click **Finish** to add the new source file(s).

The Define Module dialog box opens.

2-2. Specify the inputs and outputs for the *register8.vhd* module in the Define Module dialog box.

- **clock** – in – std_logic
- **reset** – in – std_logic
- **enable** – in – std_logic
- **data_in** – in - std_logic_vector – width 8 (i.e., 7 downto 0)
- **data_out** – out - std_logic_vector – width 8 (i.e., 7 downto 0)

2-2-1. Fill in the dialog box as shown in the figure below.

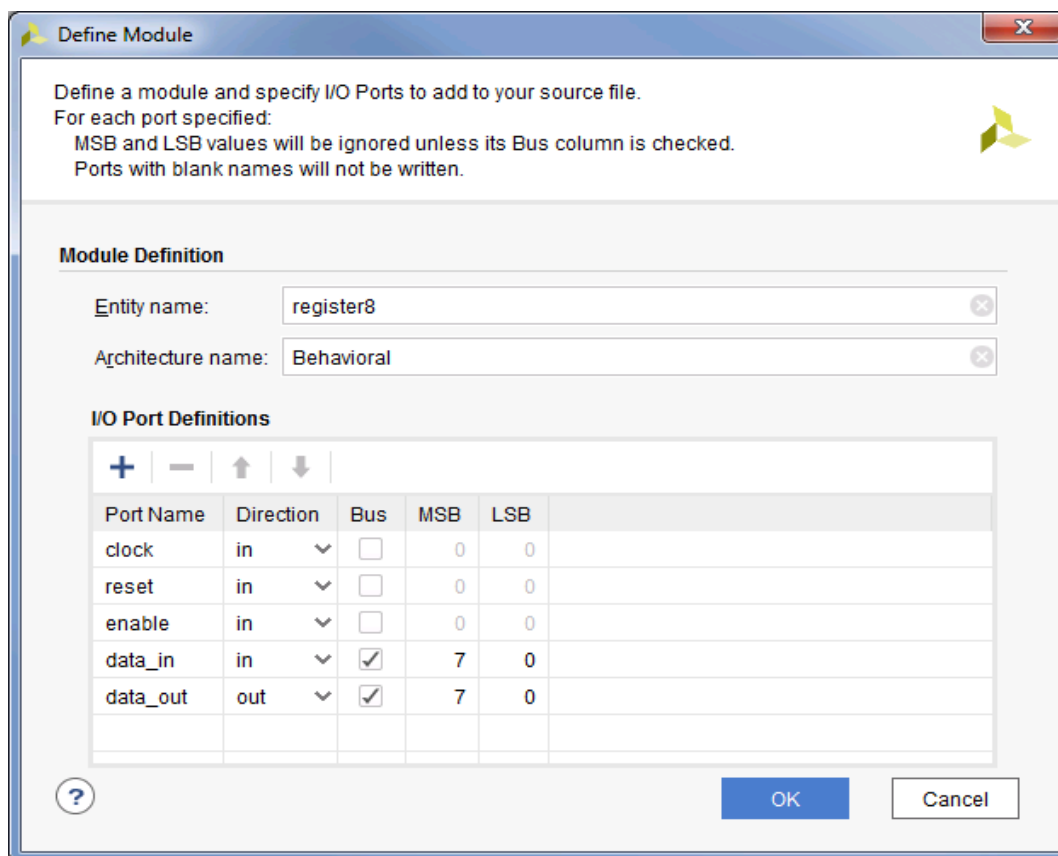


Figure 1-11: Define Module Dialog Box

Hint: Use the **+** button to add more ports to the module.

2-2-2. Click **OK**.

The newly formed code will appear in the Sources view.

2-2-3. Double-click the **register8.vhd** module in the Hierarchy tab of the Sources window to open the file in a text editor window of the Vivado IDE.

2-3. Uncomment the IEEE.NUMERIC_STD package in the code.

The VHDL template automatically includes the IEEE library and uses the STD_LOGIC_1164 package, which contains the definitions for std_logic and std_logic_vector.

Another package in the IEEE library that is frequently used is the NUMERIC_STD package, which is typically used for managing std_logic_vector and integer conversions. This approved standard package replaces the STD_LOGIC_ARITH, STD_LOGIC_UNSIGNED, and STD_LOGIC_SIGNED packages that you may see in older code.

Although the NUMERIC_STD package is not needed for this lab you can uncomment this statement as there are no negative effects.

One more library is provided in the template: the UNISIM library. This library contains the package Vcomponents and is used when Xilinx primitives such as buffers and clock management resources are used.

Again, none of these primitives are going to be explicitly declared in this file and you can leave this code commented out. If you choose to uncomment this block, there are no negative effects.

- 2-3-1. Remove the - - comment symbols before the **use IEEE.NUMERIC_STD.ALL** on line 27 to include the NUMERIC_STD package.

```
21 |
22 | library IEEE;
23 | use IEEE.STD_LOGIC_1164.ALL;
24 |
25 | -- Uncomment the following library declaration if using
26 | -- arithmetic functions with Signed or Unsigned values
27 | --use IEEE.NUMERIC_STD.ALL;
28 |
29 | -- Uncomment the following library declaration if instantiating
30 | -- any Xilinx leaf cells in this code.
31 | --library UNISIM;
32 | --use UNISIM.VComponents.all;
33 |
34 | entity register8 is
35 |     Port ( clock : in STD_LOGIC;
36 |           reset : in STD_LOGIC;
37 |           enable : in STD_LOGIC;
38 |           data_in : in STD_LOGIC_VECTOR (7 downto 0);
39 |           data_out : out STD_LOGIC_VECTOR (7 downto 0));
40 | end register8;
41 |
42 | architecture Behavioral of register8 is
```

Figure 1-12: Source Code with the NUMERIC_STD Package Commented

Notice how the editor shows that line 27 is no longer a comment by changing from the dull gray color to purple (used for keywords) and black (used for names).

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
```

Figure 1-13: Current State of the Source

This is an indicator that this is a predefined library, package, or type defined in a package.

Also notice that the keywords are colored purple. These visual cues act as a quick confirmation that you entered the information as intended.

2-4. Using the Language Templates, select the snippet for a simple register with enable.

Now that you have defined the inputs and outputs for this module, the behavior must be described.

2-4-1. Select **Tools > **Language Templates** to open the Language Templates window.**

The Language Templates icon will also be present on the toolbar to the top of the text editor. The Language Templates window contain a large number of code snippets that demonstrate how various constructs can be built.

2-4-2. Expand **VHDL > **Synthesis Constructs** > **Coding Examples** > **Flip Flops** > **D Flip Flop** > **Posedge** so that you can see the various examples.**

2-4-3. Open the **w/Synchronous Active High Reset and CE** example.

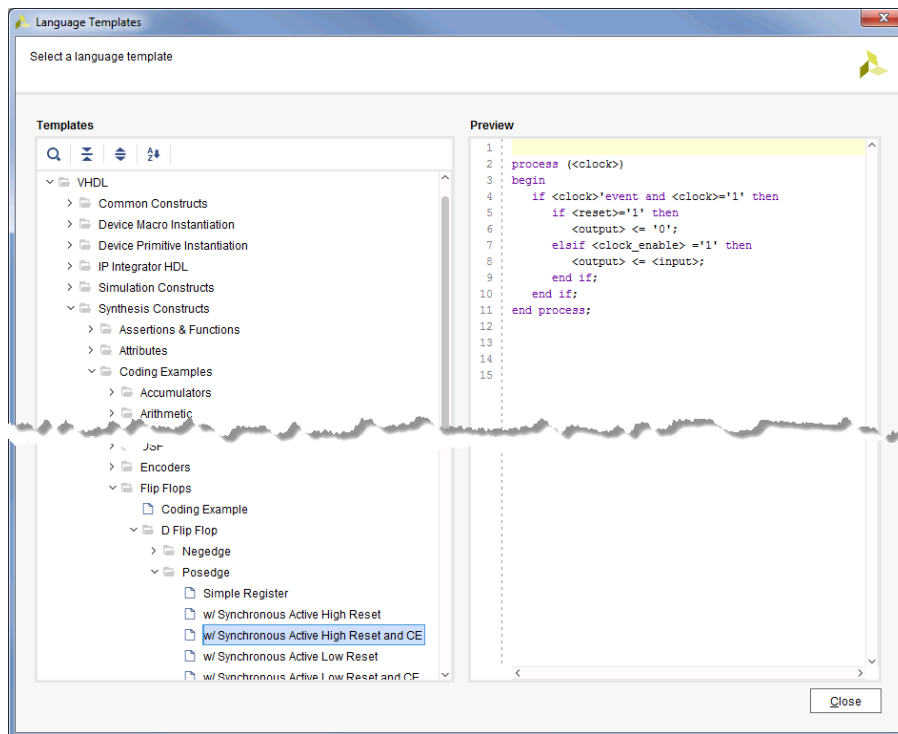


Figure 1-14: Partially Expanded Language Template List

Notice how code appears in the Preview section. This code represents a template for creating the object selected from the tree.

2-5. **Copy and paste the entire snippet into the source code between the "begin" and "end Behavioral" statements. Indent the snippet.**

2-5-1. Select the text in the Preview section of the Language Templates window.

2-5-2. Right-click the highlighted area to bring up the pop-up menu and select **Copy**.

You can also use the keyboard shortcut **<Ctrl + C>**.

2-5-3. Close the Language Templates window.

2-5-4. Click the **register8.vhd** tab to switch back to the source code.

The asterisk means that this file is unsaved.

2-5-5. Click line 45 or 46 (anywhere between the "begin" and "end Behavioral" statements) to mark where you want to place the text that have in the copy buffer.

2-5-6. Right-click and select **Paste**.

You can also use the keyboard shortcut **<Ctrl + V>**.

2-5-7. Indent the block of code just inserted by selecting it and pressing the **<Tab>** key.

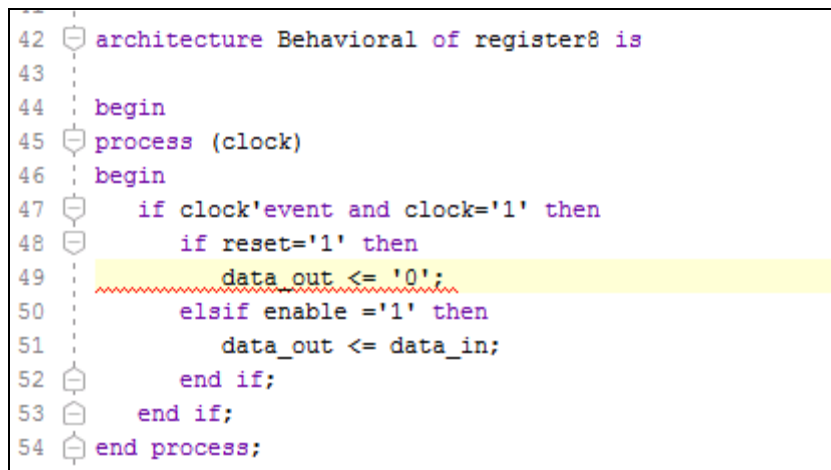
Good design practice calls for intelligent use of white space.

2-6. The snippet represents generic code. Change the items enclosed in the angle brackets with signals that were defined in the entity.

2-6-1. Make the following substitutions:

- <clock> becomes clock
- <reset> becomes reset
- <clock_enable> becomes enable
- <input> becomes data_in
- <output> becomes data_out

The code should appear as in the following figure.



```
42 architecture Behavioral of register8 is
43
44 begin
45 process (clock)
46 begin
47     if clock'event and clock='1' then
48         if reset='1' then
49             data_out <= '0';
50         elsif enable='1' then
51             data_out <= data_in;
52         end if;
53     end if;
54 end process;
```

Figure 1-15: Completed Source Code

2-6-2. Observe how the assignment of '0' to data_out is highlighted in the text editor. By hovering the cursor near the line, you will see why the statement is highlighted.

You will fix this issue in the following step.

2-6-3. Select **File > Text Editor > Save File**.

2-7. Optional: Return to the Language Templates and explore the code snippets for creating specific structures as well as the "Common Constructs" which is a syntax helper.

Synthesizing the Design

Step 3

Now that the code has been entered, you must verify that it is what you intended.

3-1. Launch synthesis. Observe how an error is reported in the Console window. This is expected and you will fix this in the next sub-step.

3-1-1. Select **Synthesis > Run Synthesis** in the Flow Navigator and click **OK** to launch runs.

A dialog box opens indicating that synthesis has failed.

3-1-2. Click **OK** to close the Synthesis Failed dialog box.

Notice the errors that appear in the Messages console.

3-1-3. Click the "**register8.vhd:<line-number>**" hyperlink, which will open the *register8.vhd* file.

Notice the line number where the error has occurred.

3-2. The error relates to the `data_out` signal being comprised of eight signals, but the initialization only covers one signal.

Correct the error and resynthesize.

3-2-1. Change the line that reads: "`data_out <= '0';`" which refers to a single value being assigned to "`data_out <= (others=>'0');`" which will set the entire bus to zero.

The snippet was valid for non-bus signals. By changing it, you made it compatible with bus signals.

The corrected code should appear as in the figure below.

```
42 architecture Behavioral of register8 is
43
44     begin
45     process (clock)
46     begin
47         if clock'event and clock='1' then
48             if reset='1' then
49                 data_out <= (others=>'0');
50             elsif enable ='1' then
51                 data_out <= data_in;
52             end if;
53         end if;
54     end process;
55
56 end Behavioral;
```

Figure 1-16: Completed Source Code

3-2-2. Select **File > Text Editor > Save File**.

3-2-3. Click **Run Synthesis** in the Flow Navigator.

This time, the code should complete without errors.

Examining the Design with Viewers

Step 4

How do you know if the code built the logic that was expected? Simulation would be a good choice and will be covered in subsequent labs. In this lab, you will view the generated code using the schematic viewers available in the Vivado IDE.

4-1. Use the Schematic viewer to explore the design.

The Schematic viewer shows the logic and register relationships and how it is to be implemented.

- 4-1-1. Select **Open Synthesized Design** in the Synthesis Completed dialog box after synthesis has finished.
- 4-1-2. Click **OK**.
- 4-1-3. Select **Layout > I/O Planning**, if it is not already selected, to open the I/O Planning view.

Notice that the Vivado IDE view changes; the Package and Device views should now be visible.

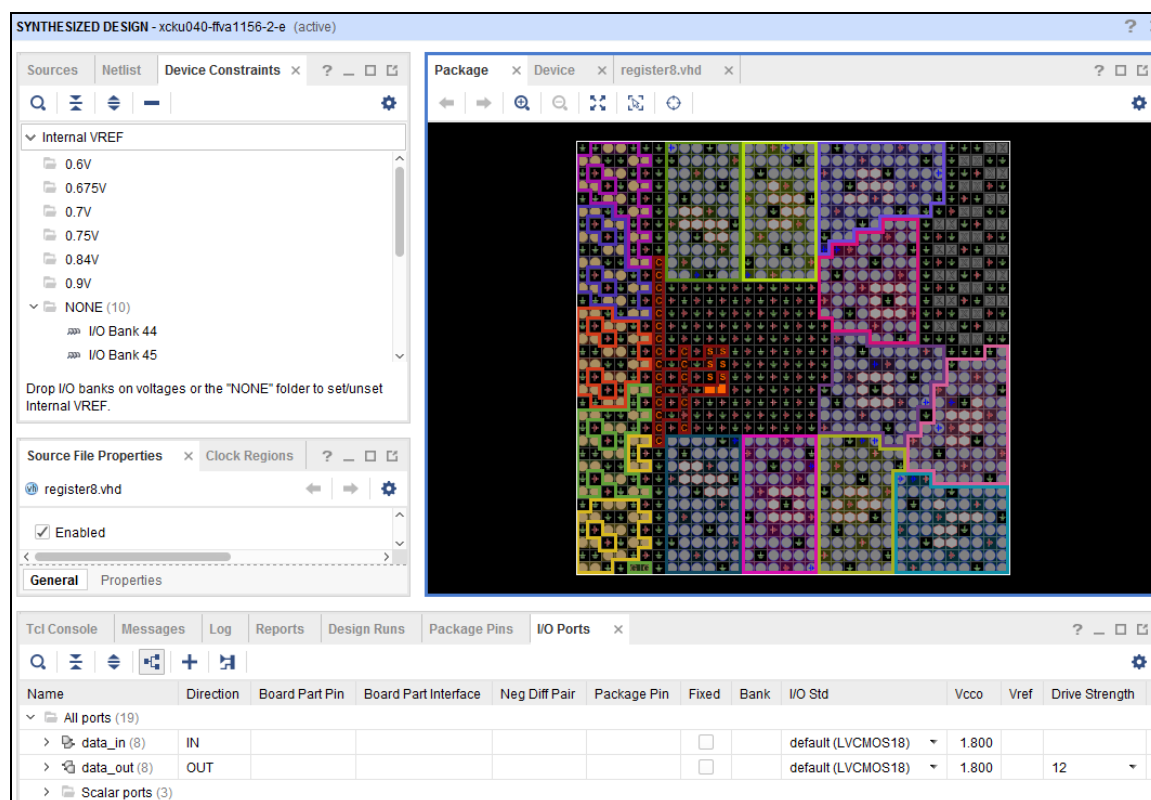


Figure 1-17: Synthesized Design View

- 4-1-4. Select the **Netlist** tab to the left of Device Constraints tab to see the Netlist view.

4-1-5. Right-click **register8** in the Netlist view and select **Schematic**.

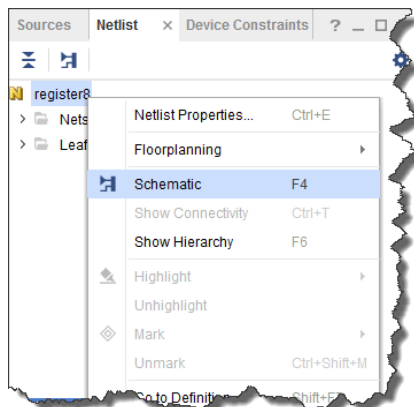


Figure 1-18: Selecting the Schematic of the register8 Netlist

The final design should appear as shown in the figure below, depending on the board you are using.

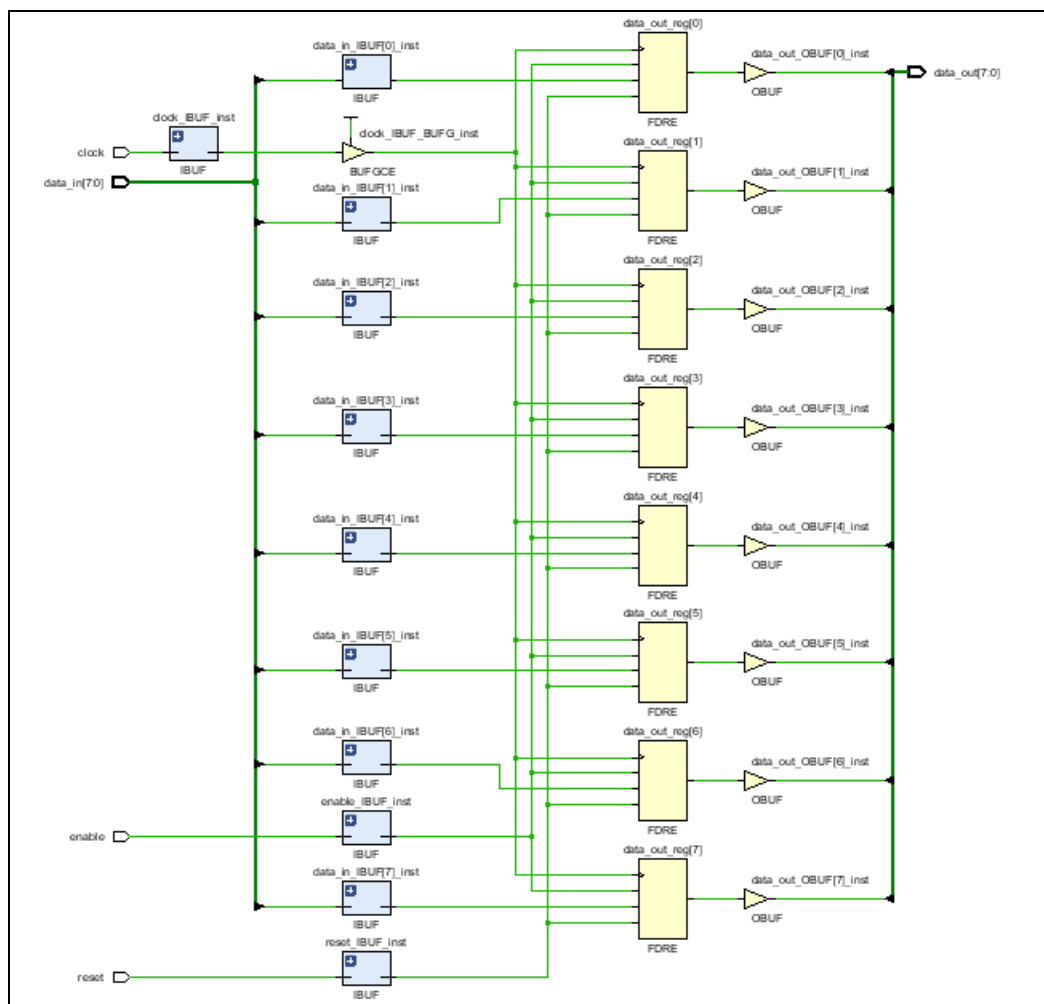


Figure 1-19: Schematic View (KCU105)

Question 1

Does the final design in the Schematic view appear as you would expect? Why or why not?

4-2. Close the Vivado Design Suite.

4-2-1. Select **File > Exit**.

The Exit Vivado dialog box opens.



Figure 1-20: Exit Vivado Dialog Box

4-2-2. Click **OK**.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/usingTools` directory.

4-3. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

4-3-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/usingTools`.

4-3-2. Select **usingTools**.

4-3-3. Press <**Delete**>.

-- OR --

Using the command line:

4-3-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**;
Linux: press <**Ctrl** + **Alt** + **T**>).

4-3-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/usingTools`

[Linux users]: `rm -rf $TRAINING_PATH/usingTools`

Summary

This lab demonstrated the basic capabilities of the Vivado Design Suite, covering design entry, compilation, and resultant schematic viewing.

Additional features and capabilities will be shown in subsequent labs.

Answers

1. Does the final design in the Schematic view appear as you would expect? Why or why not?

It should be pretty close to what you expected. Eight registers, each with an enable, clock, data in, data out, and a reset were built. The tools have also inserted the buffers for the Input and Output signals and have used some clock buffers for the clock pins.