

---

## **Tutorial on Vivado 2017.2 Using PYNQ**

---

### **Objectives**

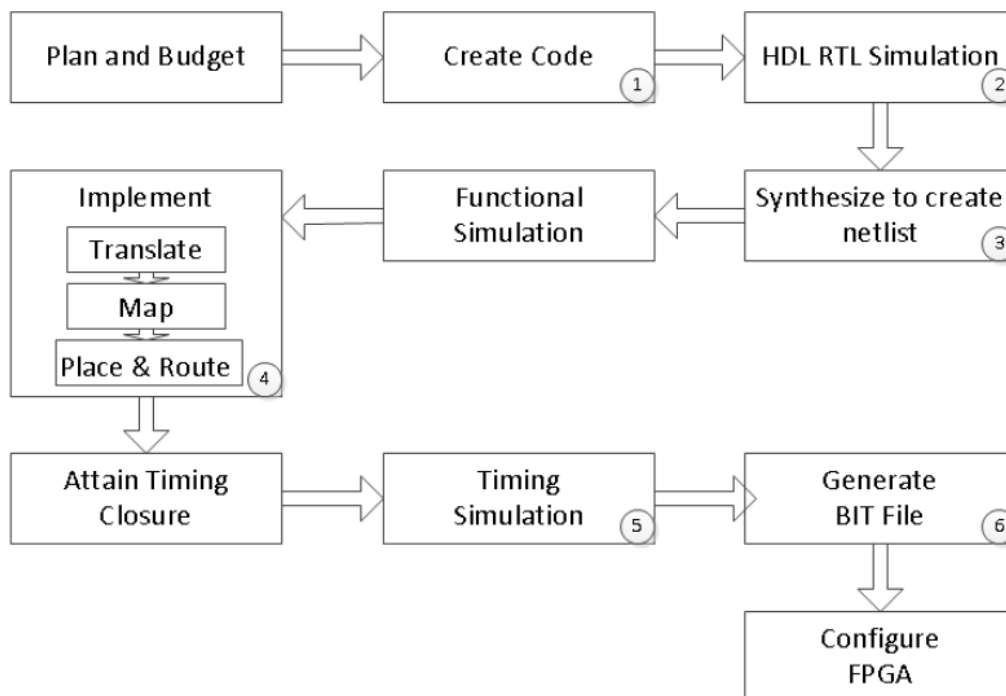
After completing this tutorial, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the PYNQ board
- Simulate the design using XSim simulator
- Synthesize and implement the design
- Generate the bitstream
- Download the design and verify the functionality

---

### **Introduction**

This tutorial guides you through the design flow using Xilinx Vivado software to create a simple digital circuit using Verilog. A typical design flow consists of creating model(s), creating user constraint file(s), creating a Vivado project, importing the created models, assigning created constraint file(s), optionally running behavioral simulation, synthesizing the design, implementing the design, generating the bitstream, and finally verifying the functionality in the hardware by downloading the generated bitstream file. You will go through the typical design flow targeting the ZYNQ-7000 based PYNQ board. The typical design is shown below.

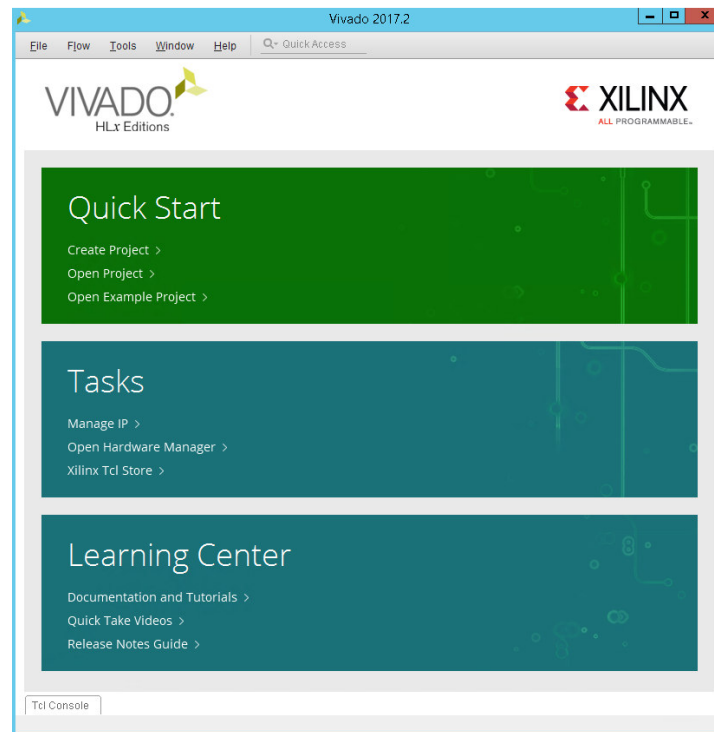


**Figure 1: A typical design flow**

## Part I – Drawing Schematics

The Xilinx Vivado allows users to design circuits for Xilinx FPGA's and CPLD's. It involves the use of Vivado IDE, a user interface that helps users manage the entire design process including design entry, simulation, synthesis, implementation and finally downloading the design onto an FPGA or CPLD.

1. If you want to use **Remote Desktop**, then read the *RemoteDesktopGuide.pdf* in order to connect to the Remote Desktop. After connecting you can open Vivado from the Start menu by pressing the **Windows Key** -> type **Vivado** -> select **Vivado 2017.2**. The Vivado IDE opens as shown in Figure 2. The Vivado IDE lets you manage the sources and processes in your project.

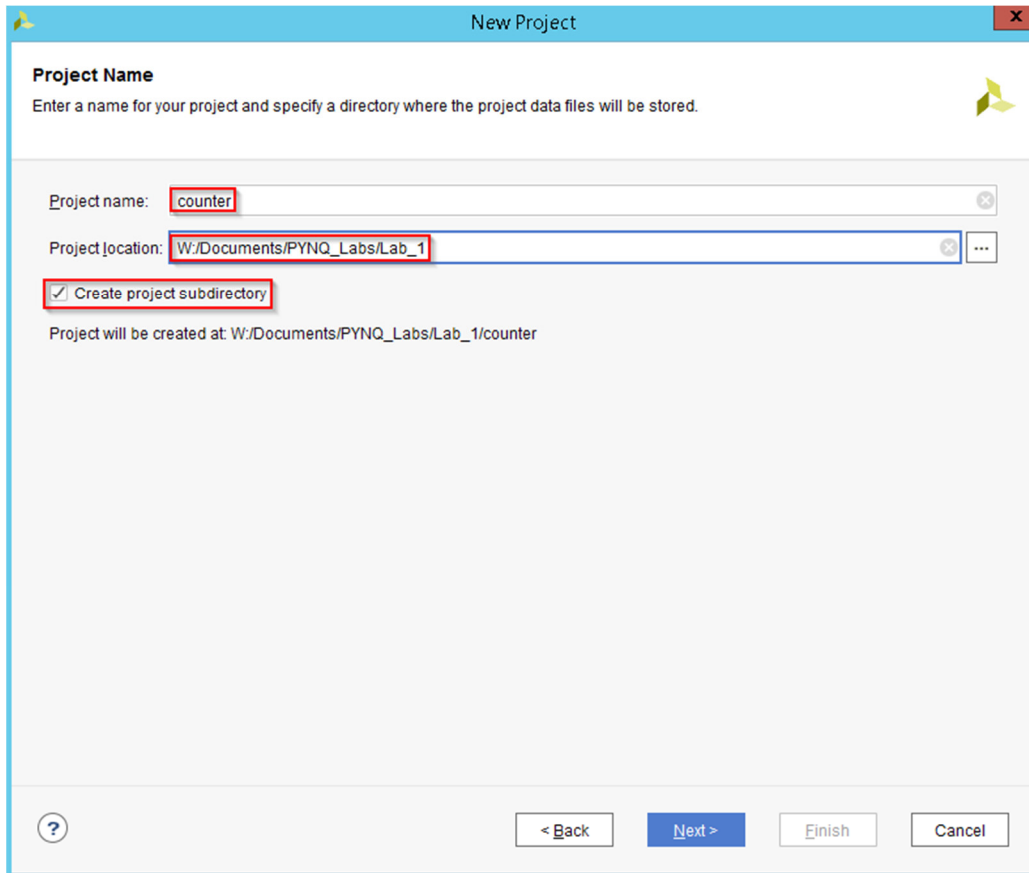


**Figure 2: Launching Vivado IDE**

- The next step is to create a new project. To create a new project for this tutorial: Click **Create Project** to start the wizard. You will see *Create a New Vivado Project* dialog box. Click **Next**.
- First, enter a Project Location (directory path) for the new project. You can either choose a location on your local **C:** drive (if you have Vivado installed on your laptop) or choose a location on **W:** drive (if you plan to use the Blade Server).

**NOTE: DO NOT EVER SPECIFY A DIRECTORY OR FILENAME WITH SPACES.** This will cause Xilinx to mysteriously fail on certain operations.

- Type **counter** in the Project Name field. Do not forget to check *Create project subdirectory* box and a **counter** subdirectory is created automatically in the directory path you selected as shown in Figure 3. Now click **Next**.



**Figure 3: Creating a new project**

2. Select **RTL Project** option and then click **Next**.
3. Select **Verilog** as the *Target Language* and as the *Simulator language* in the *Add Sources* form and then click **Next**.
4. Click **Next** to avoid adding a constraints file in *Add Constraints* form for now.
5. In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section, select the **xc7z020clg400-1** part as shown in Figure 4 below. Or to keep it simple, you can select **Boards** and then select **PYNQ-Z1**. Click **Next**.
6. Click **Finish** to create the Vivado project as shown in Figure 5.

Use the Windows Explorer and look at the **project** directory. You will find that the **counter.xpr** (Vivado) project file has been created.

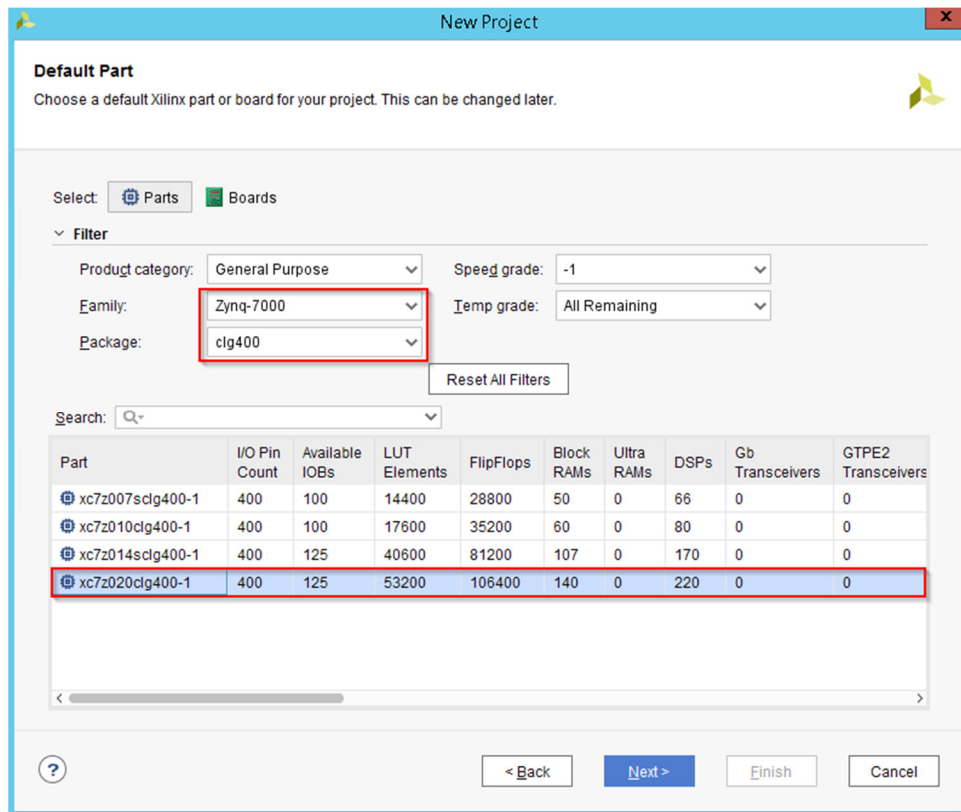


Figure 4: Part selection

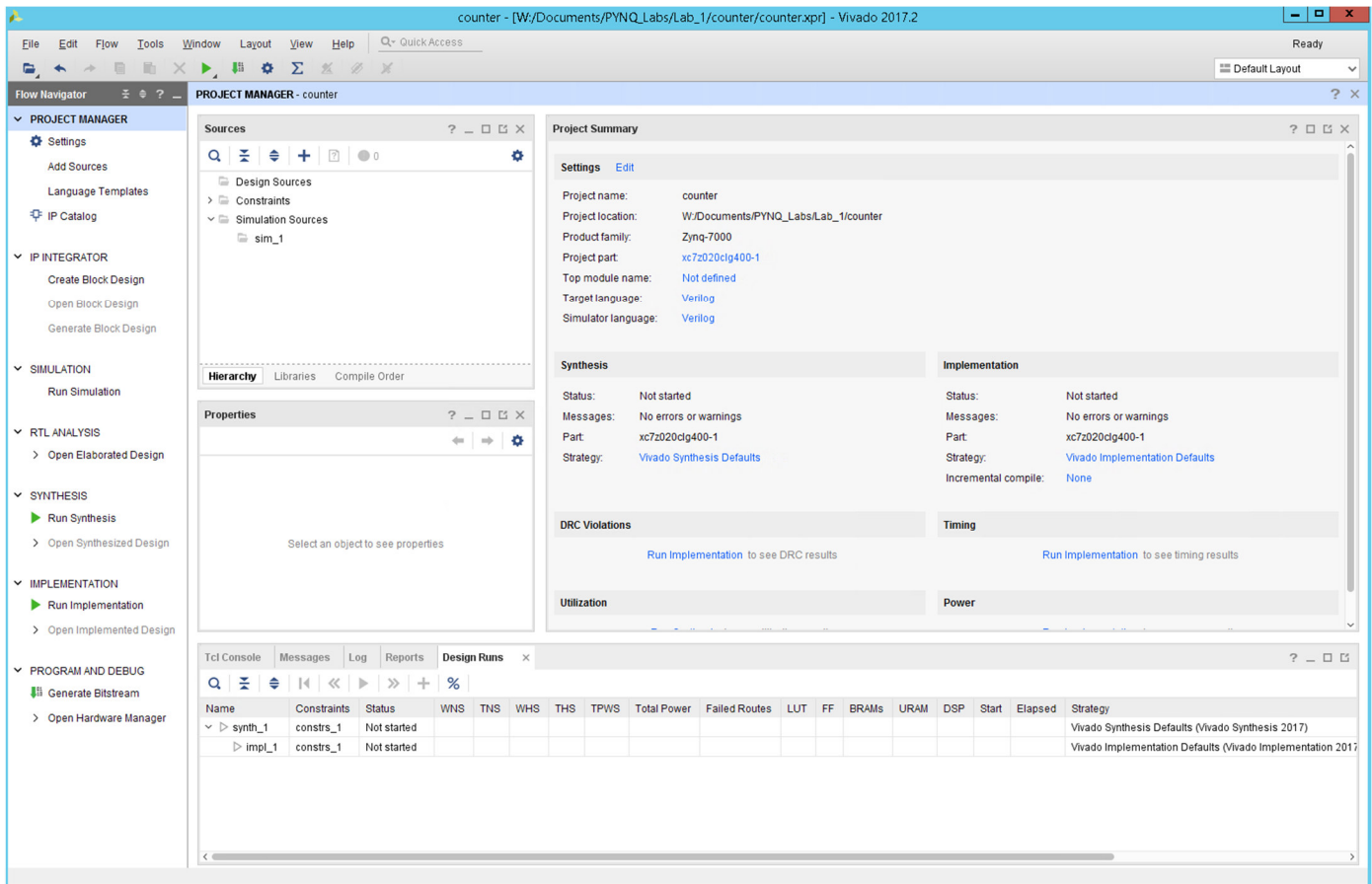
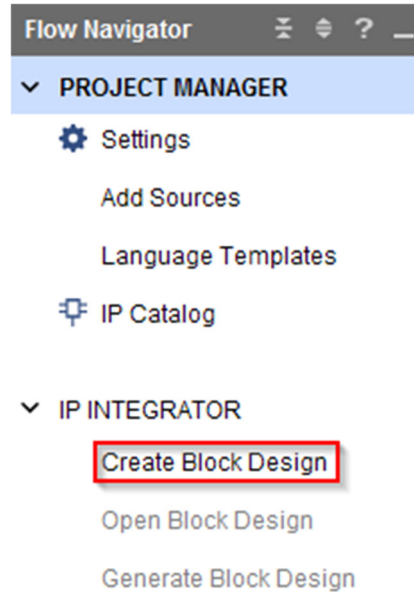


Figure 5: Successful creation of project

7. Now create schematic in the form of **Block Design**. In the **Flow Navigator** window, click on **Create Block Design** under the **IP Integrator** block as shown in Figure 6 below.

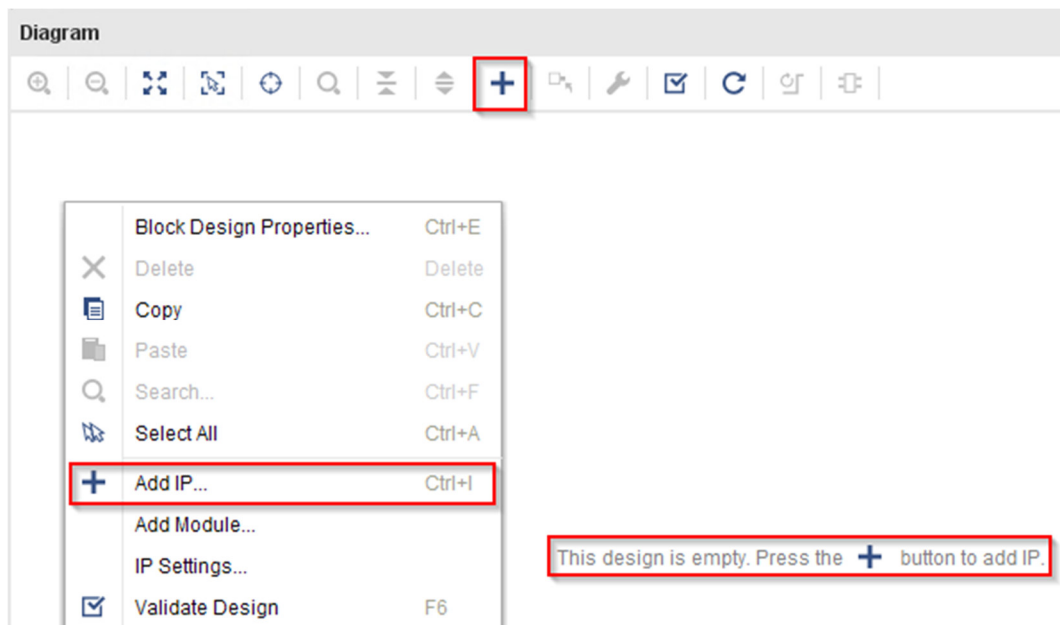


**Figure 6: Invoking IP Integrator to create a Block Design**

8. Under **Create Block Design** window, enter **counter** under **Design name** and click **OK**.

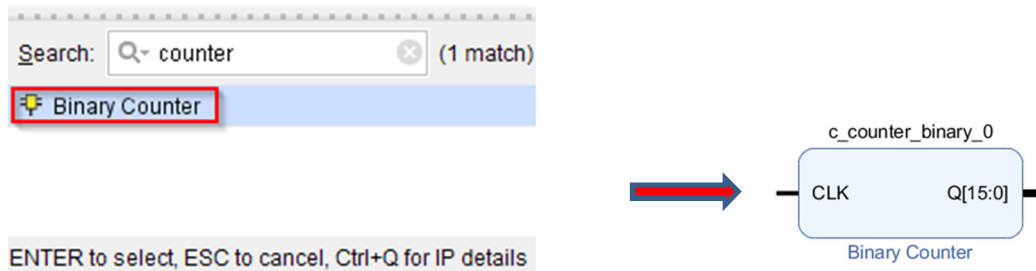
You will find that the **counter.srscs** directory has been created. The **counter.srscs** directory is a place holder for the Vivado program database.

9. IP from the catalog can be added in several ways. Click the **Add IP icon** **+** in the block diagram side bar, or right-click anywhere in the **Diagram workspace** and select **Add IP** as shown in Figure 7 below.



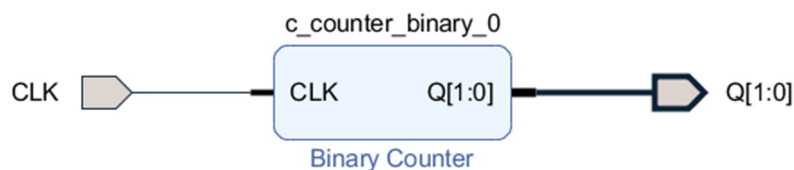
**Figure 7: Add IP to Block Diagram**

10. Once the **IP Catalog** is open, type “**counter**” into the Search bar, find and double click on **Binary Counter** entry, or click on the entry and hit the **Enter** key to add it to the design as shown in Figure 8 below



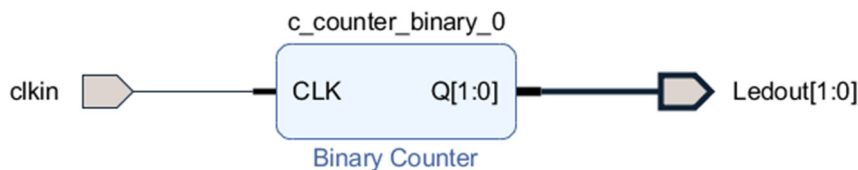
**Figure 8: Add a counter to the design**

11. Now double-click on counter symbol to customize it. Under **Basic** tab, change **Output Width** field to 2 to convert it into a 2-bit counter and then click **OK**.
12. Right-click on the **c\_counter\_binary\_0** instance’s input port (**CLK**) and select **Make External**. Similarly, make the output port (**Q[1:0]**) of the same instance and make it external as shown below.



**Figure 9: Making ports external**

13. Click on the **CLK** port and under **External Port Properties** form, change the name to **clkin** and hit **Enter**. Similarly, change the output port **Q** to **Ledout** as shown in Figure 10 below.



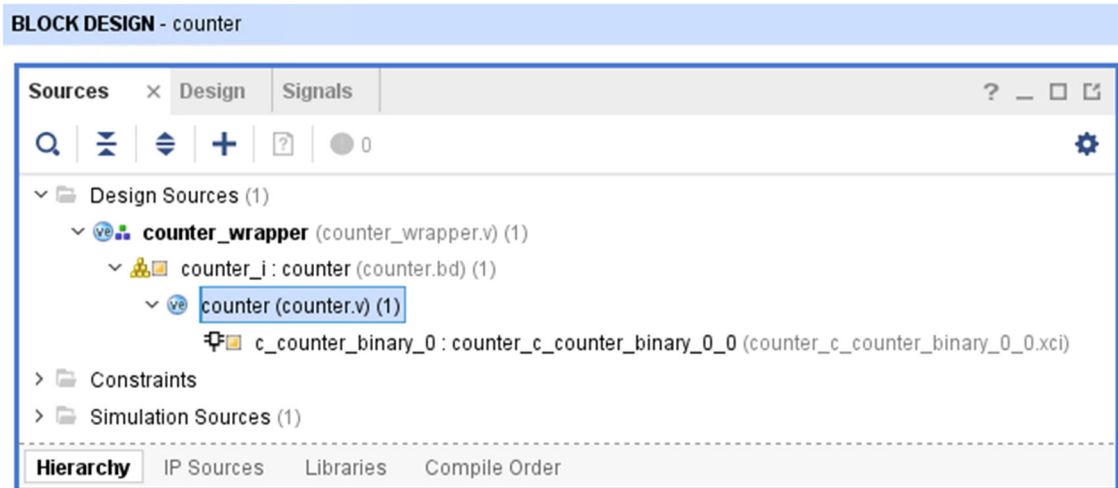
**Figure 10: Modified ports name**

14. Now click on **Ctrl + S** or select **File → Save Block Design** to save the Block design.


Use the Windows Explorer again and look at the **project** directory. Deep within the **counter.srcs** directory you will find **counter.bd** file (which is your schematic file).

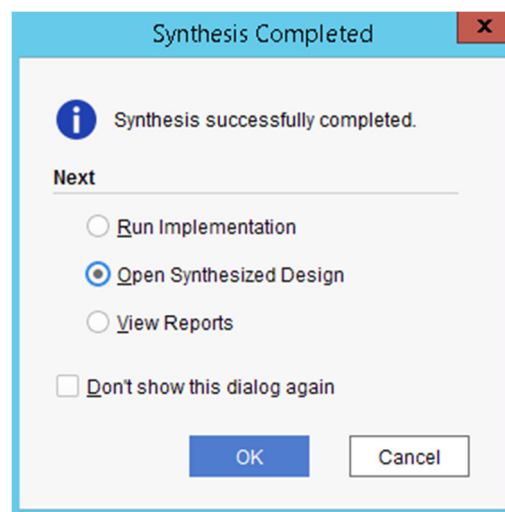
15. Next step is to create a HDL wrapper file as Vivado does not allow schematic to be the top-level file. In the **Sources** view, right-click on the block diagram file, **counter.bd**, and select **Create HDL Wrapper** to create the HDL wrapper file. When prompted, select **Let Vivado manage wrapper and auto-update**, click **OK**.

16. In the **Sources** pane, expand the hierarchy. Notice the **counter\_wrapper** file instantiates **counter.bd** which in turn instantiates **counter.v**, which in turn instantiates the Binary Counter symbol as shown in Figure 11 below.

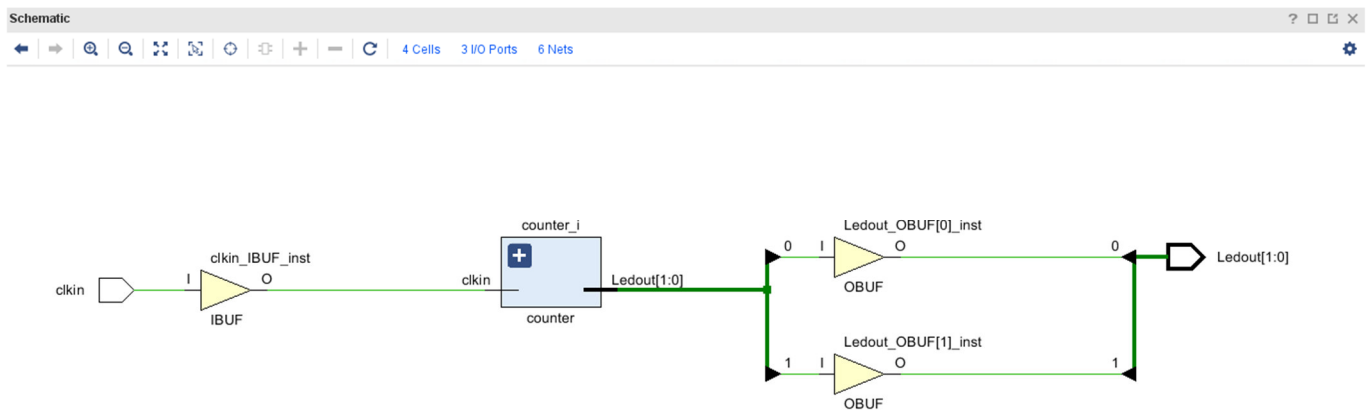


**Figure 11: Hierarchical design**

17. The **counter** design must now be *synthesized*, i.e., converted to a representation that maps to actual hardware resources on the Xilinx FPGA that you selected. To do so:
- Select **counter\_wrapper.v** file under **Sources** pane
  - Then click on **Run Synthesis** option under **Flow Navigator** window and **Synthesis** block.
  - Leave the default options in the **Launch Runs** window and click **OK**.
18. Wait for Synthesis process to be completed. On successful completion, a dialog box will appear as shown in Figure 12.
19. Select **Open Synthesized Design** and click **OK**. You will see the synthesized view of your design (Figure 13).
- If you do not see the synthesized view, select **Schematic**  under **Netlist** window or press **F4**.



**Figure 12: Synthesis completion dialog box**

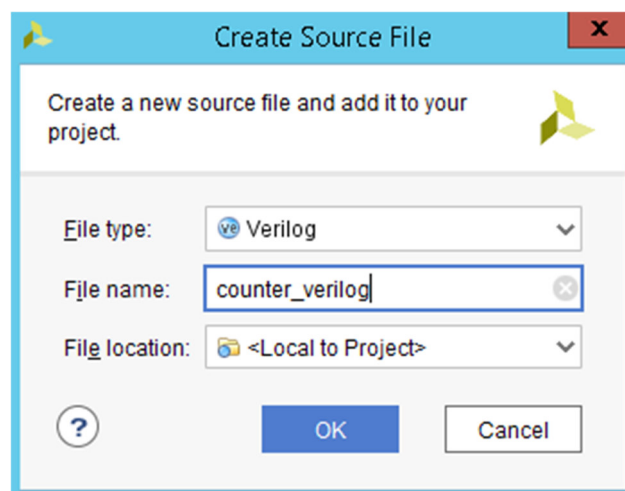


**Figure 13: Synthesized view of the design**

20. You can now close the synthesized design. The select **File → Close Project** to close the current project.

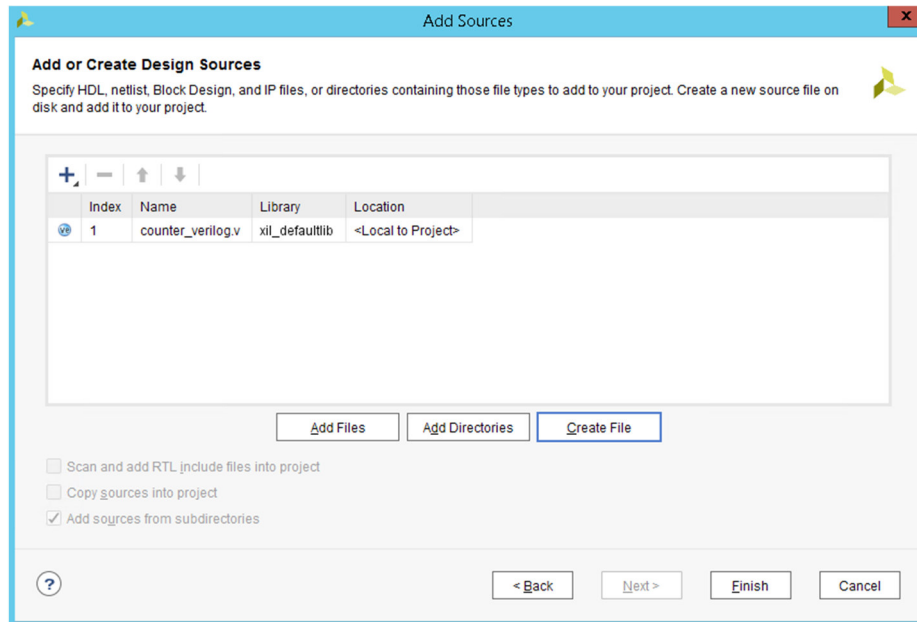
## **Part II – Circuit Design with Verilog**

1. Create a new Vivado project (just like you did in Part 1) and name the project as ***counter\_verilog***. The device that we are using is a **xc7z020clg400-1** device.
2. Once the project is created, the next step is to create a Verilog source file. In order to do that, click on **Add Sources** option under **Project Manager** form in **Flow Navigator** window.
3. In **Add Sources** dialog box, select **Add or Create Design Sources** and then click on **Next**.
4. Now click on **Create File** option and under **File name** field type “**counter\_verilog**” and click on **OK** as shown in Figure 14.
5. You should now see **counter\_verilog** file listed under **Add Sources** dialog box as shown in Figure 15. Now click on **Finish**.



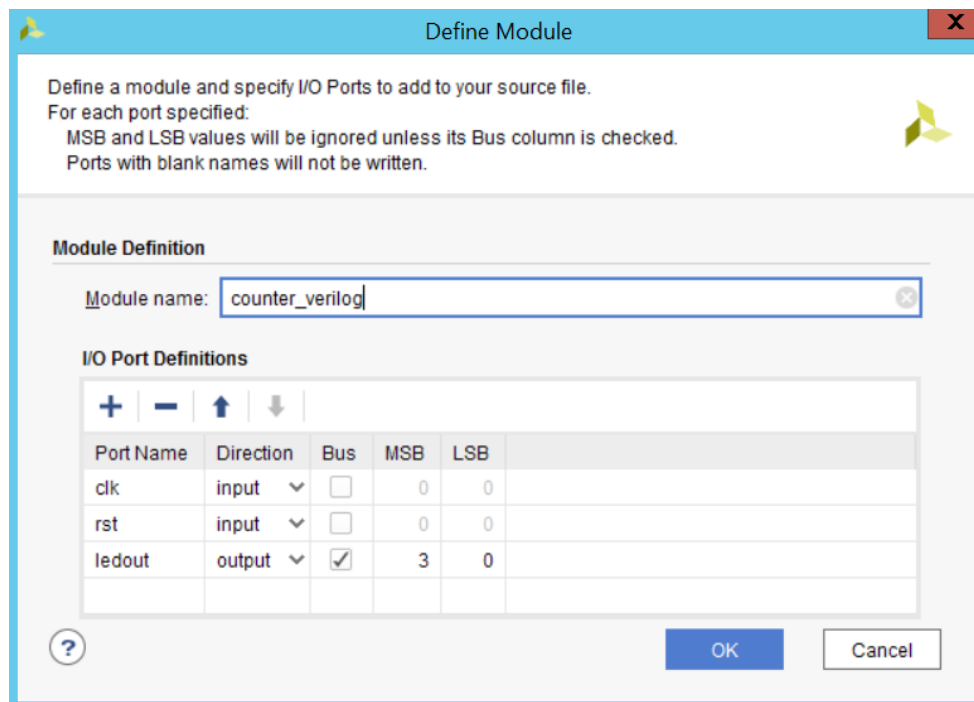
**Figure 14: Creating a Verilog source file**





**Figure 15: Adding a Verilog source file to the project**

6. The **Define Module** dialog box appears. In this dialog you will set the names of the input and output “pins” of the circuit. Set the fields of this dialog to appear as in Figure 16 and click **OK**.
7. In the **Project Manager** window, you should see *counter\_verilog.v* source file under **Sources** form. Double-click on the file to open it. You will see a pre-defined template in Verilog as shown in Figure 17.



**Figure 16: Defining the inputs and outputs of a Verilog module**

```

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:
7  // Design Name:
8  // Module Name: counter_verilog
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////
21
22
23 module counter_verilog(
24     input clk,
25     input rst,
26     output [3:0] ledout
27 );
28 endmodule

```

**Figure 17: Verilog Template**

8. Some of the Verilog code has already been written for you. Finish the design by modifying the source code as shown in Figure 18. Press **Ctrl-S** to save your work. Once you save the file, it will automatically perform syntax check and display any errors in the **Messages** window.

```

15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////
21
22
23 module counter_verilog(clk,rst,ledout);
24
25     input clk,rst;
26     output [3:0] ledout;
27
28     reg [3:0] ledout;
29     reg [3:0] ledtemp;
30
31     always @(posedge clk or negedge rst)
32     begin
33         if(rst == 0)
34             ledtemp <= 0;
35         else
36             ledtemp <= ledtemp + 1; //Increment ledtemp only on rising edge of clk
37
38         ledout <= ledtemp;
39     end
40
41 endmodule
42

```

**Figure 18: Verilog code implementation of a 4-bit counter**

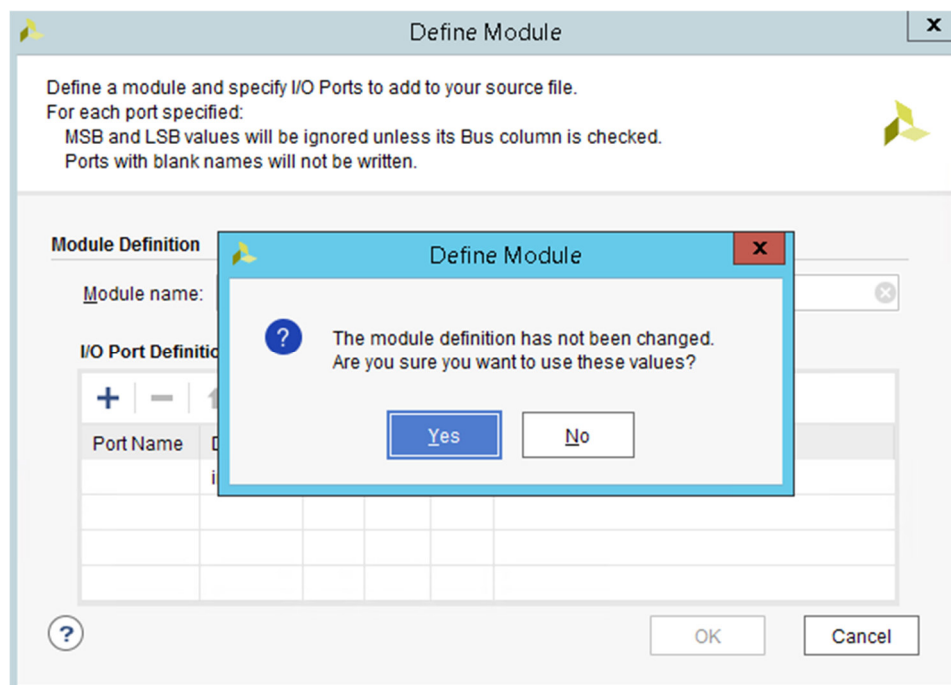
9. Once the code is correct, synthesize the design by clicking on **Run Synthesis** option under **Synthesis** form in **Flow Navigator** window.
10. *What do you see? Did it synthesize successfully?* If it did not synthesize successfully, correct the errors and synthesize again. Repeat until a successful synthesis is achieved.
11. After synthesizing the design, let's perform a **behavioral simulation** of the design in the next section.

---

### **Part III – Verilog Test Benches**

Although we can construct waveform stimulus for our designs using the graphical editor (**not possible in Vivado**), doing so is cumbersome, time consuming, and error-prone. Furthermore, the graphical simulation results must be verified “by eye”, which is also cumbersome, time-consuming, and error-prone. We can instead use Verilog to both specify the input stimulus (just a clock and reset so far) and check the output results.

1. In order to create a testbench file, click on **Add Sources** option under **Project Manager** form in **Flow Navigator** window.
2. In **Add Sources** dialog box, select **Add or Create Simulation Sources** and then click on **Next**.
3. Now click on **Create File** option and under **File name** field type “**counter\_tb**” and click on **OK** followed by **Finish** to create the testbench file.
4. The **Define Module** dialog box appears. Since we do not have to define I/O ports as it is a testbench just click on **OK** followed by **Yes** to create the file (see Figure 19).



**Figure 19: Creating testbench file**

5. The testbench file gets created. Double-click and open the testbench file which is shown in Figure 20.

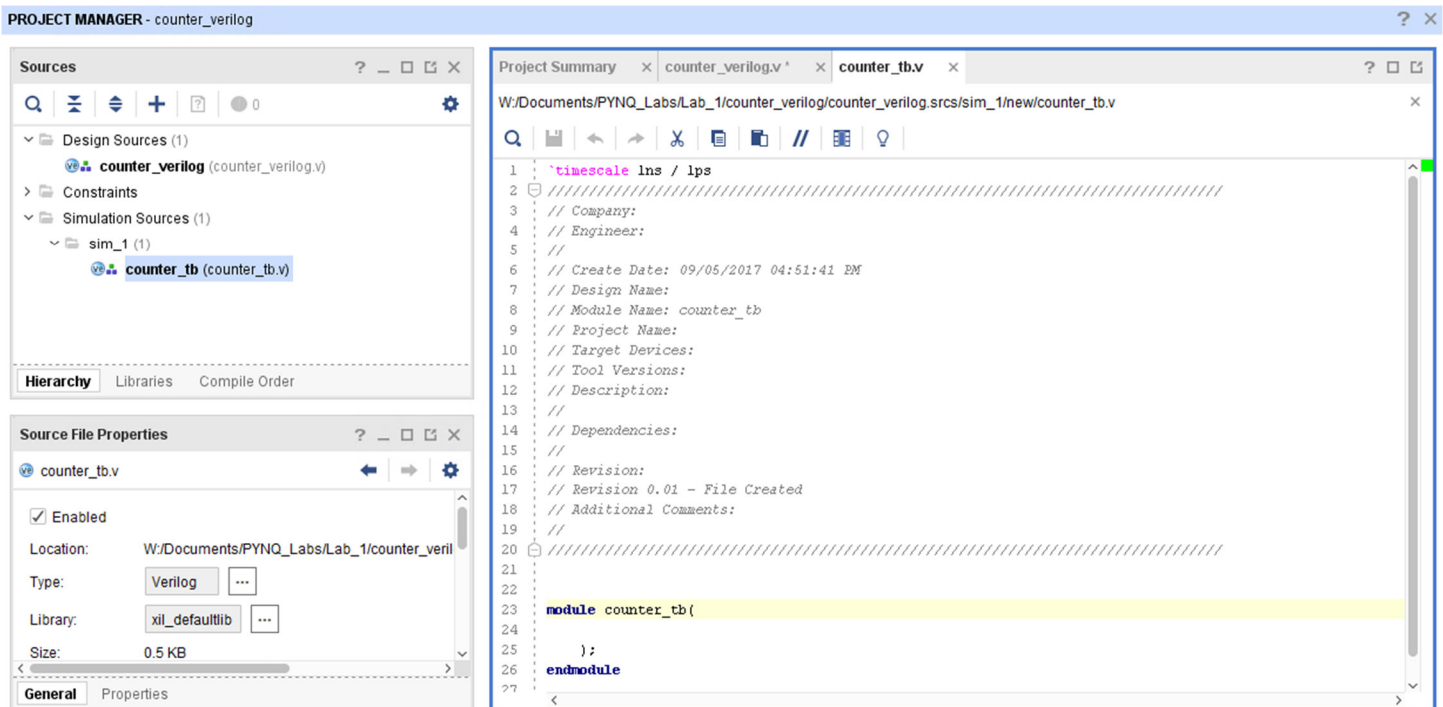


Figure 20: Testbench template

**Note:** This is just a template and you need to provide the stimulus.

6. Now you need to make changes to the **counter\_tb.v** file (as shown in Figure 21) that will allow you to create clock stimulus to 4-bit counter design.

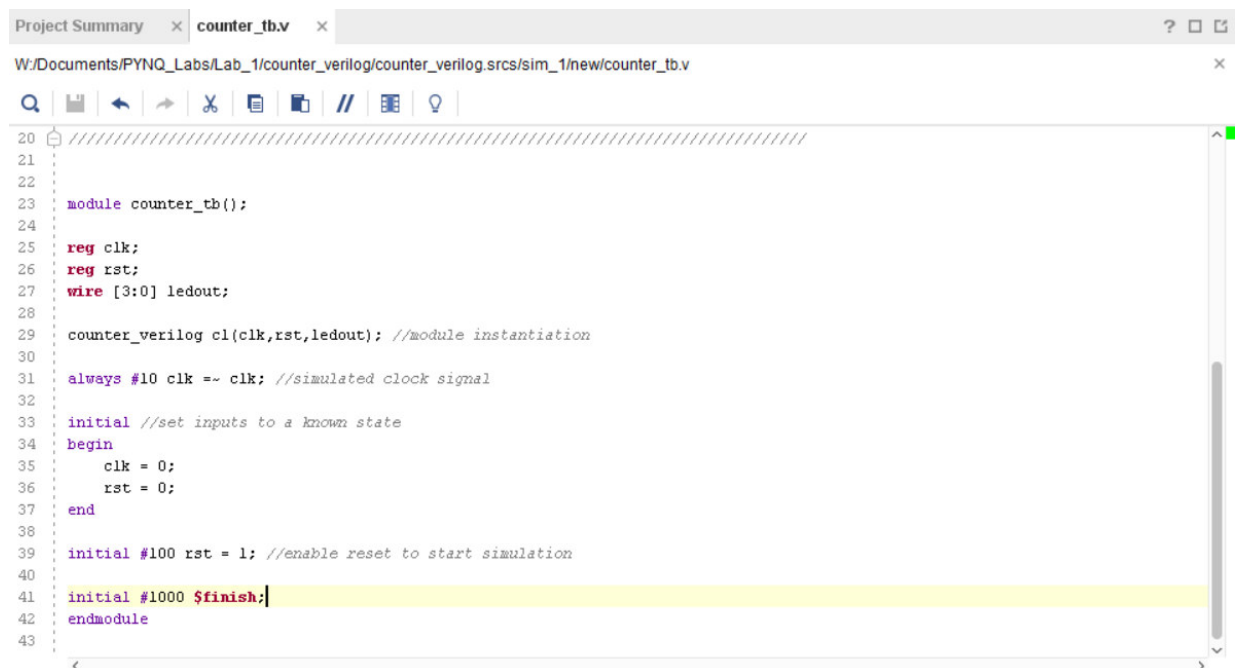
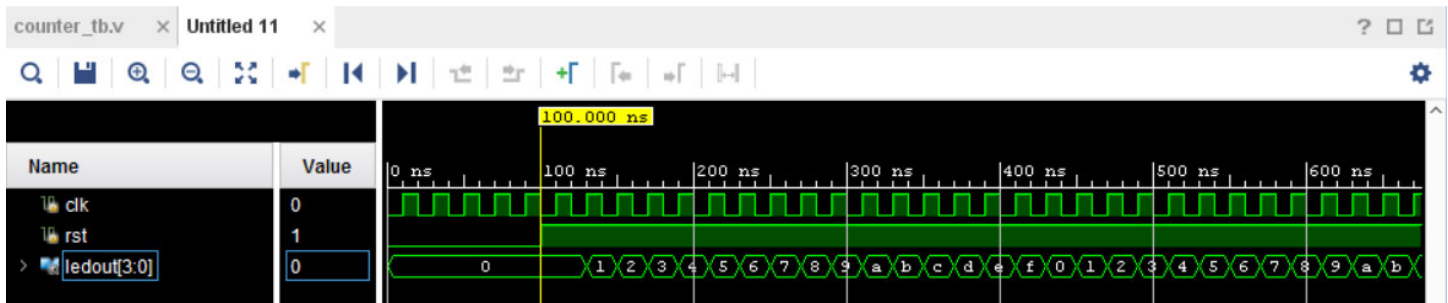


Figure 21: Verilog Test bench code to provide clock stimulus

7. Now save the testbench file. Under **Flow Navigator** window, select **Run Simulation** → **Run Behavioral Simulation** option under **Simulation** form.
8. The simulation window should open and display the simulation results as shown in Figure 22 below. Verify that your circuit functions as a 4-bit counter.



**Figure 22: Results of 4-bit counter simulation**

9. You might need to perform the following steps to be able to see the different count values:
  - a. Select **View -> Zoom Fit** to be able to see all the different count values on the same screen.
  - b. In the simulation output window, right-click on **ledout[3:0]** signal and then select **Radix -> Unsigned Decimal** to change the count value representation to Decimal.
  - c. You can then **Zoom in** to see the different count values clearly. You will see that the counter counts up to 15 and then repeats until 1000ns is reached.
    - i. Remember the “*initial #1000 \$finish;*” line in the testbench? This line tells the simulator to stop after 1000ns. Adjust the finish value and observe the changes.
10. You can now close the simulation window to go back to your project. Make sure you **EXIT** the simulation application and **DISCARD** saving simulation file when asked.

## **Part IV – Downloading to the Hardware**

After successfully verifying the behavior of the design, now we need to download the design onto the actual hardware. But before that we need to slow down the clock, so that each count is visible to the user through LEDs on the board.

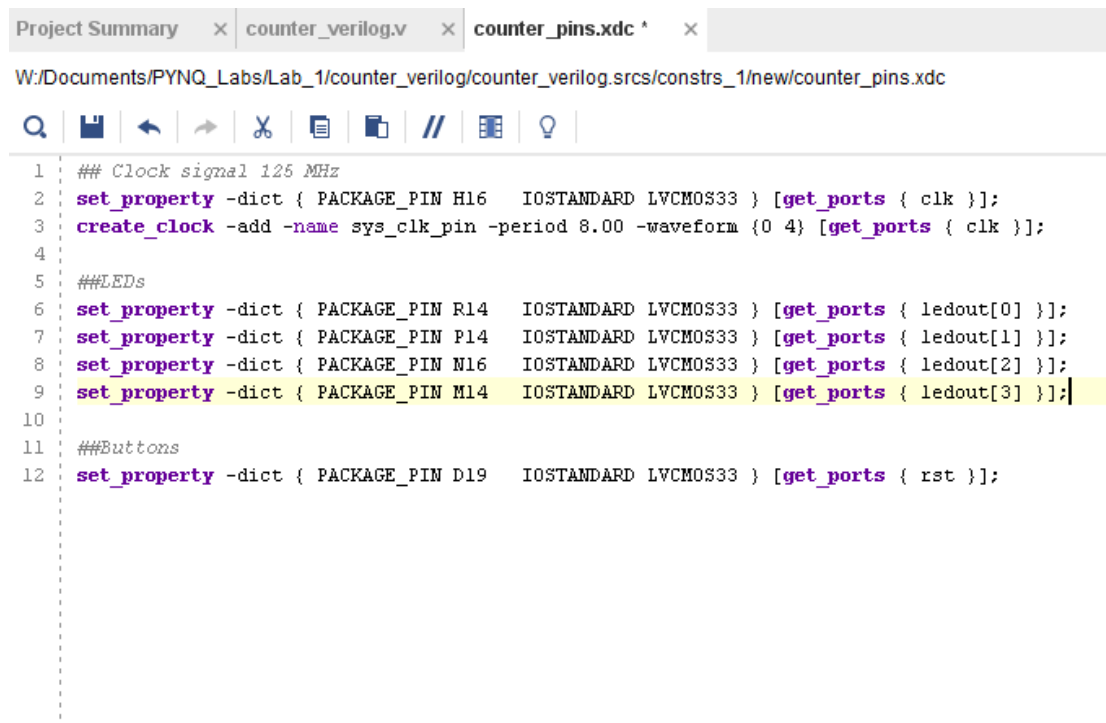
1. Now go back to edit the original ‘**counter\_verilog.v**’ source file. Figure below shows the changes you need to make to your code to include a clock divider process. You will need to:
  - a. Adjust the **ledtemp** register at line 29 size to 30.
  - b. Change the **rst** IF statement at line 33 to equal **1** to account for the active HIGH buttons on the PYNQ board.
  - c. Delete the “**ledout <= ledtemp + 1;**” statement.
  - d. Add the **always** block at line 41 and ending at line 47 to add the delay functionality.

```
counter_verilog.v
W:/Documents/PYNQ_Labs/Lab_1/counter_verilog/counter_verilog.srscs/sources_1/new/counter_verilog.v

21
22
23 module counter_verilog(clk,rst,ledout);
24
25     input clk,rst;
26     output [3:0] ledout;
27
28     reg [3:0] ledout;
29     reg [30:0] ledtemp;
30
31     always @(posedge clk or negedge rst)
32     begin
33         if(rst == 1)
34         begin
35             ledtemp <= 0;
36         end
37     else
38         ledtemp <= ledtemp + 1; //Increment ledtemp only on rising edge of clk
39     end
40
41     always @(posedge ledtemp[26] or negedge rst)
42     begin
43         if(rst == 1)
44             ledout <= 0;
45         else
46             ledout <= ledout + 1;
47     end
48
49 endmodule
50
```

**Figure 23: Modification to counter\_verilog design to implement a clock divider**

2. Now, synthesize your design again by clicking on the **Run Synthesis** option. Make sure your design synthesizes successfully.
3. After converting your design into a hardware representation, the next step is to assign actual device pins to your design input/output ports through a constraint file. You can do that by selecting **Add Sources** option under **Project Manager** form in **Flow Navigator** window.
4. In **Add Sources** dialog box, select **Add or Create Constraints** and then click on **Next**.
5. Now click on **Create File** option and under **File name** field type “**counter\_pins**” and click on **OK** followed by **Finish** to create the constraints file.
6. Now we can see the new file “**counter\_pins.xdc**” added under **Constraints** folder in **Sources** tab. Double-click on the constraints file, add the constraints as shown in Figure 24 below and then Save it.

The image shows a screenshot of a Verilog IDE with three tabs: 'Project Summary', 'counter\_verilog.v', and 'counter\_pins.xdc'. The active tab is 'counter\_pins.xdc', which contains Verilog code for pin assignments. The code includes comments for clock signal, LEDs, and buttons, with corresponding 'set\_property' and 'create\_clock' commands. The file path is 'W:/Documents/PYNQ\_Labs/Lab\_1/counter\_verilog/counter\_verilog.srcs/constrs\_1/new/counter\_pins.xdc'.

```
1  ## Clock signal 125 MHz
2  set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { clk }];
3  create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];
4
5  ##LEDs
6  set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { ledout[0] }];
7  set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { ledout[1] }];
8  set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { ledout[2] }];
9  set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { ledout[3] }];
10
11 ##Buttons
12 set_property -dict { PACKAGE_PIN D19    IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

Figure 24: Assigning I/O pins

7. After assigning the pins, you now need to convert and map your design to the logic blocks inside the FPGA. Click on **Run Implementation** option to implement the design. Make sure your design implements successfully.
8. Now click on **Generate Bitstream** to create a **bit** file (the file that will be downloaded onto your hardware). On successful completion, **counter\_verilog.bit** file generated under **impl\_1** directory which was generated under the **counter\_verilog.runs** directory.
9. The next step is to power your **PYNQ** board. Make sure that the power is set of OFF before making any adjustments to the jumpers. Make sure that the power supply source is jumpered to **USB** (not REG) and the provided Micro-USB cable is connected between the board and the PC (see Figure 25).

**Note:** You do not need to connect the power jack and the board can be powered and configured via USB alone.



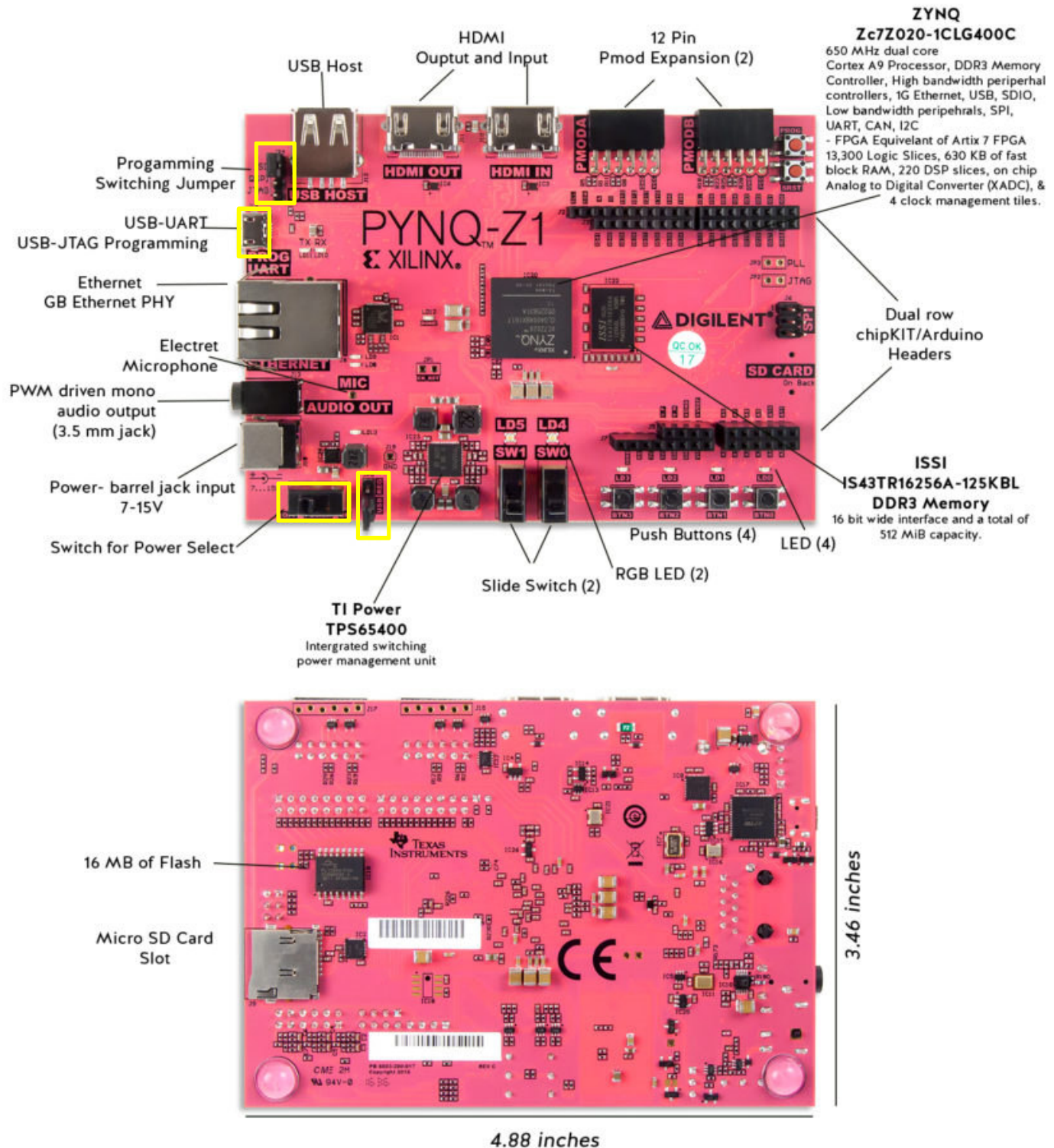
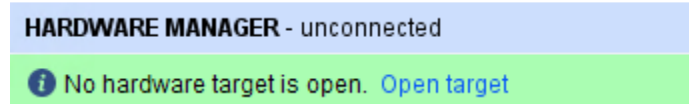


Figure 25: PYNQ board settings

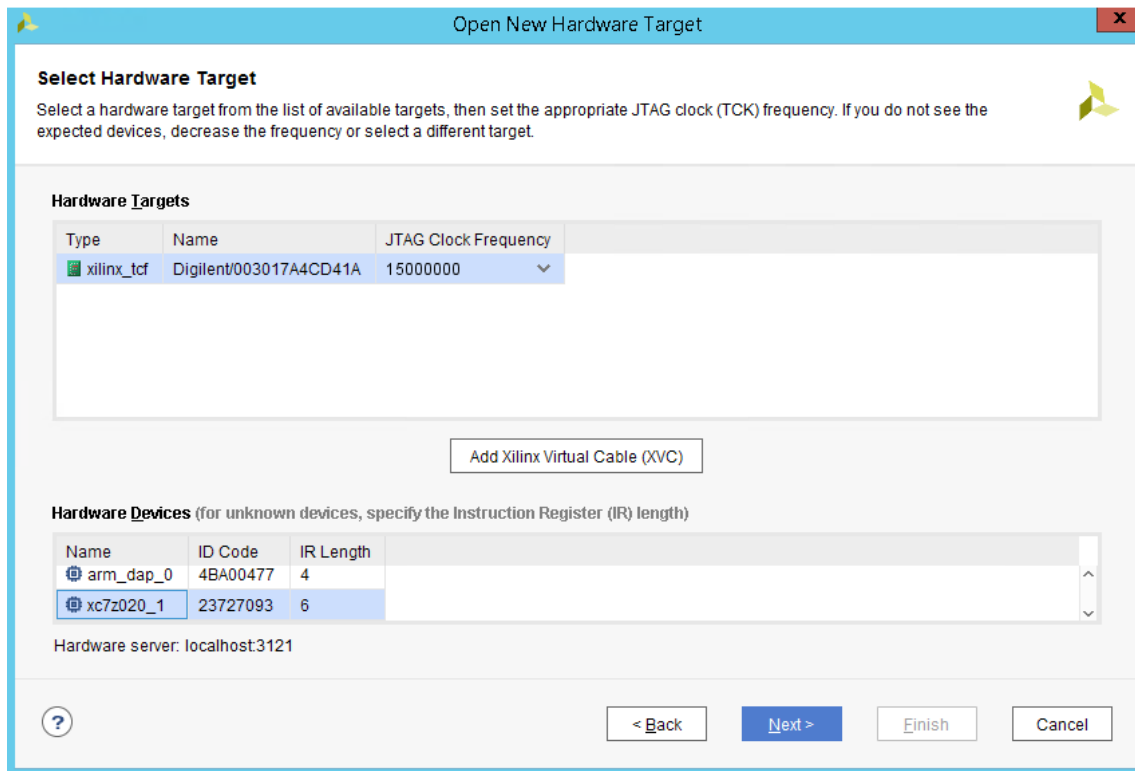


10. Move the jumper **JP4** such that it will allow the user to program FPGA using **JTAG**. **Allow the drivers to be installed**. Now, slide the **Power** switch to **ON** position.
11. Now click on **Open Hardware Manager** option under **Program and Debug** form.
12. In the green information bar as shown in Figure 26, select **Open target** followed by **Open New Target**. **Open New Hardware Target** wizard will be launched. Click **Next**.



**Figure 26: Hardware Manager window**

13. Click **Next**, accepting the default **Local server** connection. A progress bar showing the connection to the local host server will increment to completion, after which the target hardware will be identified.
14. In the **Open New Hardware Target** form, select the only option for the hardware target, set the JTAG clock frequency to 150MHz, and select the hardware device as shown in Figure 27 below. Click **Next**.



**Figure 27: Selecting the FPGA to program**

15. Verify the **Target Settings** in the summary window and click **Finish**.
16. Notice the green information bar at the top indicating there are no debug cores. Click on **Program Device**.
17. In the **Program Device** window, make sure **counter\_verilog.bit** file is selected to configure the PYNQ board. Click on **Program**.

18. The BIT file is downloaded onto FPGA. You should see the 4 LEDs turn on in a binary counting pattern from 0-15 at 1Hz, and then repeating. Pressing **BTN0** (our programmed reset button), you will notice that the counting pattern starts over at 0.
19. Once you verify that the program works, try modifying the '**counter\_verilog.v**' file to achieve a faster count rate. You may have to think and experiment a bit!
20. Once you are done, slide the POWER switch to OFF position and then disconnect the USB cable from the computer. Close the **Hardware Manager** and **Vivado** applications.