Vivado Design Flow

Introduction

This lab guides you through the process of using Vivado IDE to create a simple HDL design targeting the Zynq device. You will simulate, synthesize, and implement the design with default settings. Finally, you will generate the bitstream and download it in to the hardware to verify the design functionality

Objectives

After completing this lab, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the ZedBoard or Zybo
- Use the provided Xilinx Design Constraint (XDC) file to constrain the pin locations
- Simulate the design using the Vivado simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA using the generated bitstream and verify the functionality

Procedure

This lab is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab1.

Design Description

The design consists of some inputs directly connected to the corresponding output LEDs. Other inputs are logically operated on before the results are output on the remaining LEDs as shown in **Figure 1**.

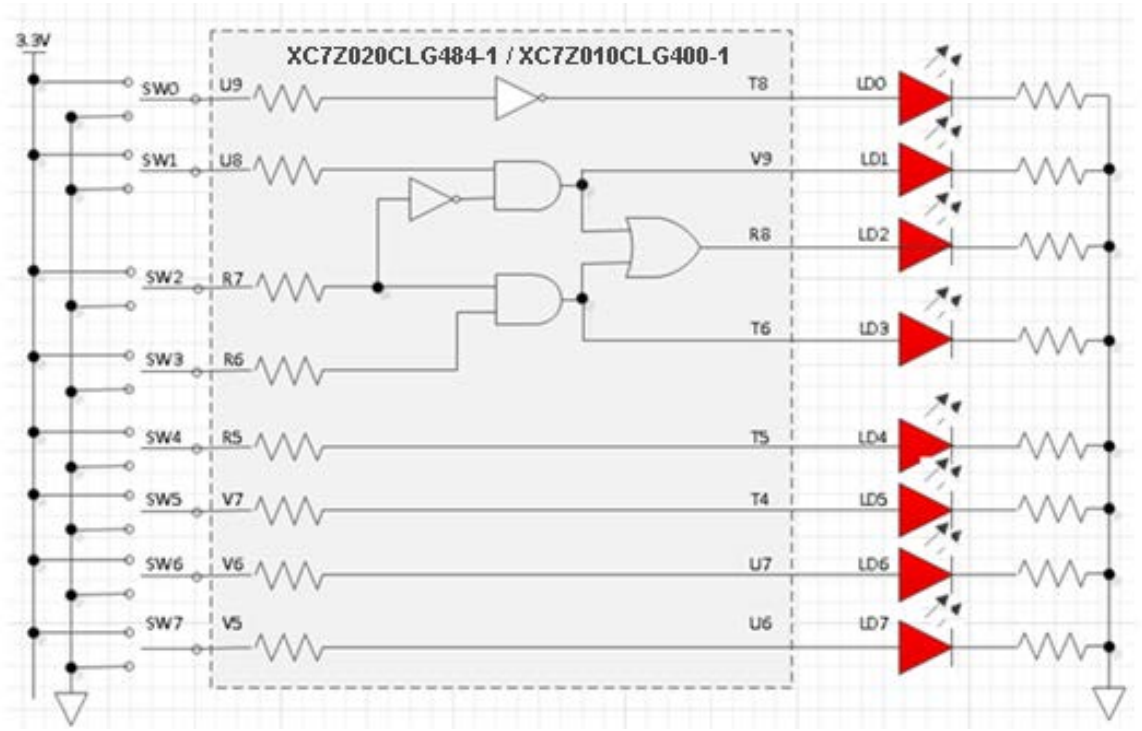
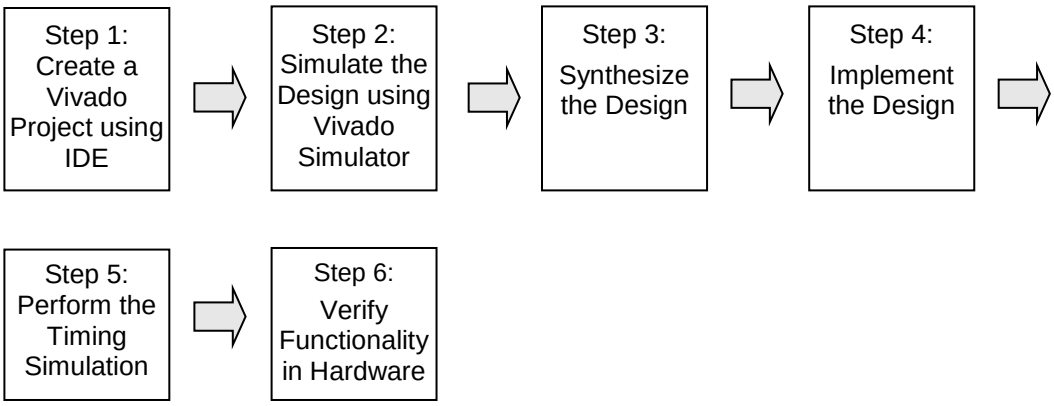


Figure 1. The Completed Design

General Flow



Create a Vivado Project using IDE — Step 1

1-1. Launch Vivado and create a project targeting the appropriate Zynq device and using the Verilog HDL. Use the provided lab1.v and lab1.xdc. The files are added to the project from the *<2014_2_zynq_sources>\<board>\lab1* directory.

> References to **<2014_2_zynq_labs>** is a placeholder for the **c:\xup\fpga_flow\2014_2_zynq_labs** directory and **<2014_2_zynq_sources>** is a placeholder for the **c:\xup\fpga_flow\2014_2_zynq_sources** directory.
>
> Reference to **<board>** means either the **ZedBoard** or the **Zybo.**

1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2**

1-1-2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.

1-1-3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **<2014_2_zynq_labs>**, and click **Select**.

1-1-4. Enter **lab1** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

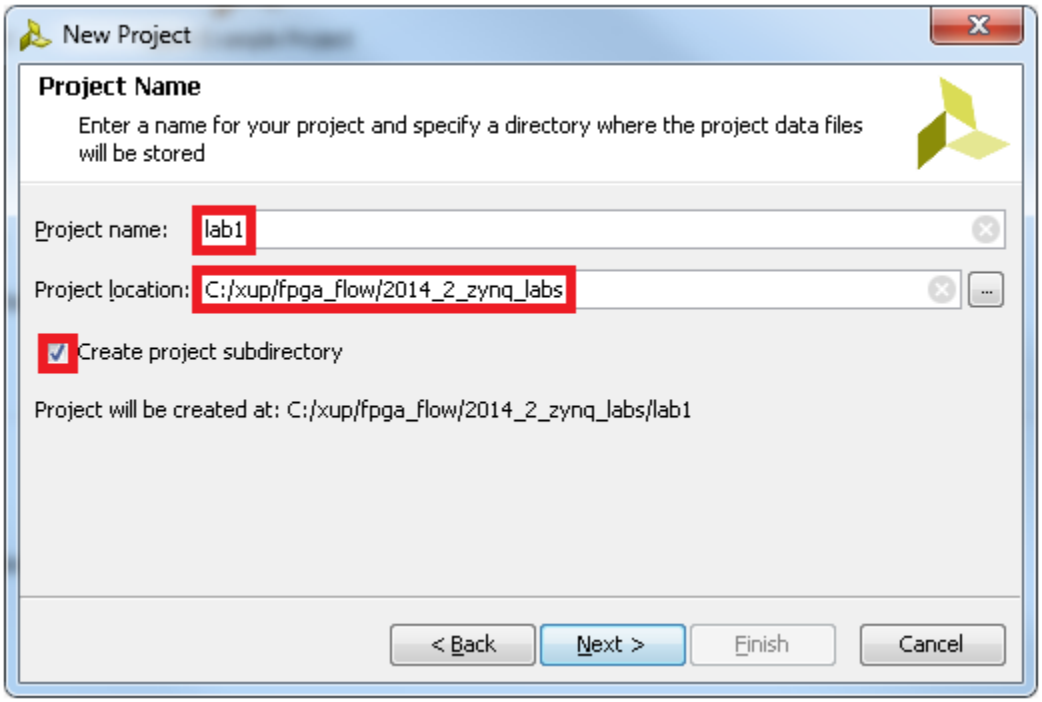


Figure 2. Project Name and Location entry

1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.

1-1-6. Select **Verilog** as the *Target Language* in the *Add Sources* form. Select **Verilog** as the *Simulator Language*.

1-1-7. Click on the **Add Files…** button, browse to the **<2014_2_zynq_sources>** directory, double click the appropriate **<board>\lab1** subdirectory and select *lab1.v,* click **OK**. Make sure the source file is **copied** into your project directory (via the check box) and then click **Next**.

1-1-8. Click **Next** again at the *Add Existing IP (optional)* form to get to the *Add Con*straints form.

1-1-9. The constraints file *lab1_zedboard.xdc* or *lab1_zybo.xdc* has automatically been added depending on the board directory previously selected while adding sources. Click **Next.**

This Xilinx Design Constraints file assigns the physical IO locations on FPGA to the switches, buttons and LEDs located on the board. This information can be obtained either through the board's schematic or the board's user guide.

1-1-10. In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section (as seen in the screen shot below), select the **XC7Z020CLG484-1** device for the ZedBoard or the **XC7Z010CLG400-1** device for the Zybo.

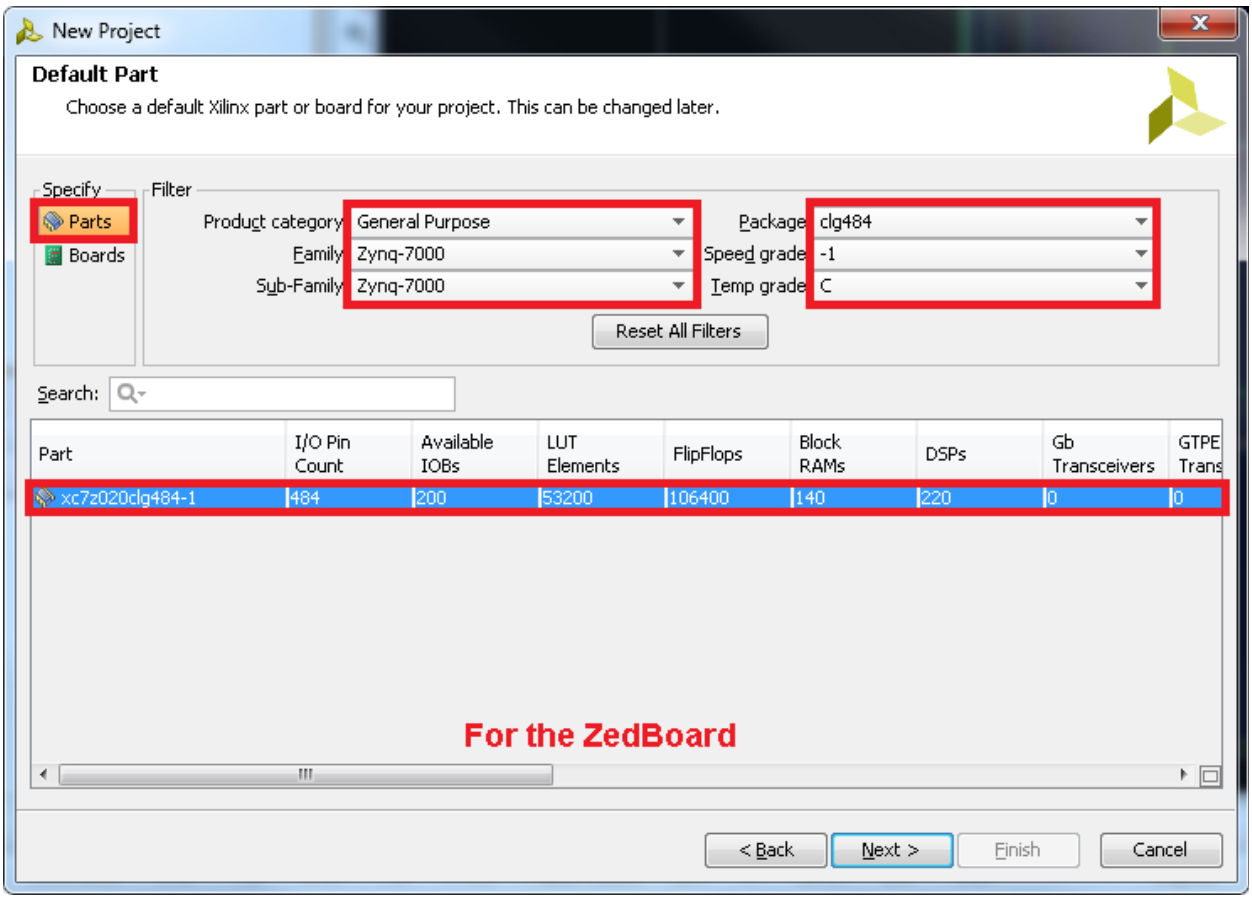


Figure 3. Part Selection to select the XC7Z020-1CLG484C device for the ZedBoard

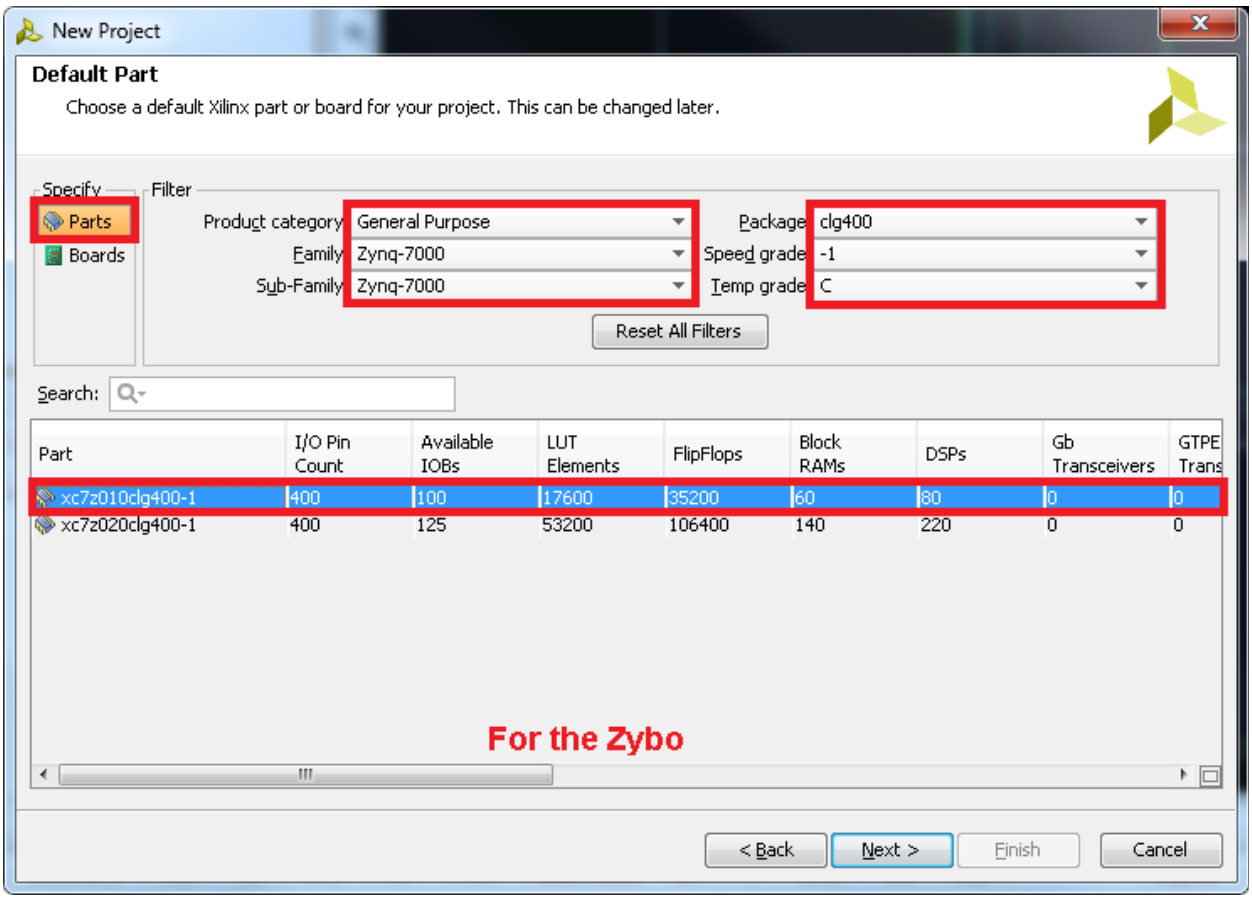


Figure 3. Part Selection to select the XC7Z010-1CLG400C device for the Zybo

1-1-11. Click **Next** to review the *Project Summary* page.

1-1-12. Click **Finish** to create the Vivado project.

Use the Windows Explorer and look at the **<2014_2_zynq_labs>\lab1** directory. You will find that the **lab1.cache** and **lab1.srcs** directories and the **lab1.xpr** (Vivado) project file have been created. The **lab1.cache** directory is a place holder for the Vivado program database. Two directories, **constrs_1** and **sources_1**, are created under the **lab1.srcs** directory; deep down under them, the copied **lab1.xdc** (constraint) and **lab1.v** (source) files respectively are placed.

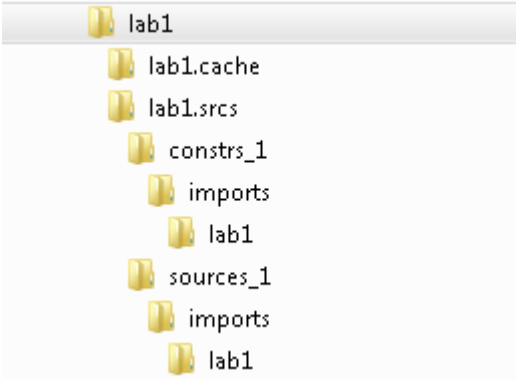


Figure 4. Generated directory structure

1-2. Open the lab1.v source and analyze the content.

1-2-1. In the *Sources* pane, double-click the **lab1.v** entry to open the file in text mode.

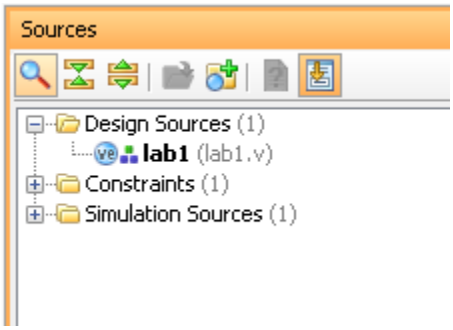


Figure 5. Opening the source file

1-2-2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. Lines 2-4 are comment lines describing the module name and the purpose of the module.

Line 7 defines the beginning (marked with keyword **module**) and **endmodule** denotes the end of the module.

Lines 8-9 defines the input and output ports whereas the assignment statements defines the combinatorial functionality of the design.

1-3. Open the lab1.xdc source and analyze the content.

1-3-1. In the *Sources* pane, expand the *Constraints* folder and double-click the **lab1.xdc** entry to open the file in text mode.

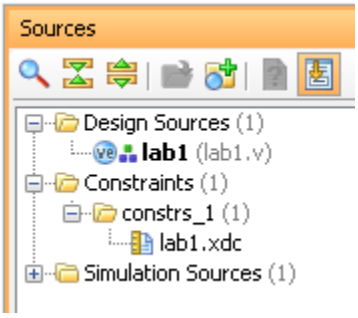


Figure 6. Opening the constraint file

1-3-2. The design takes input from the onboard slide switches and outputs to the onboard LED. The XDC file is different for the ZedBoard and Zybo due to pinout considerations.

1-4. Perform RTL analysis on the source file.

1-4-1. Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**.

The model (design) will be elaborated and a logic view of the design is displayed.

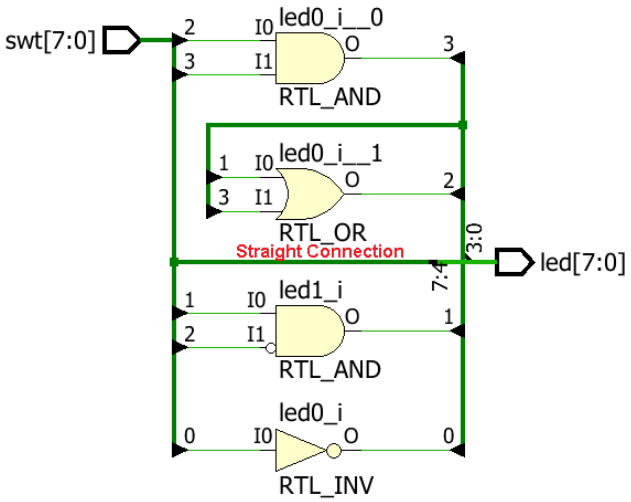


Figure 7. A logic view of the design for the ZedBoard

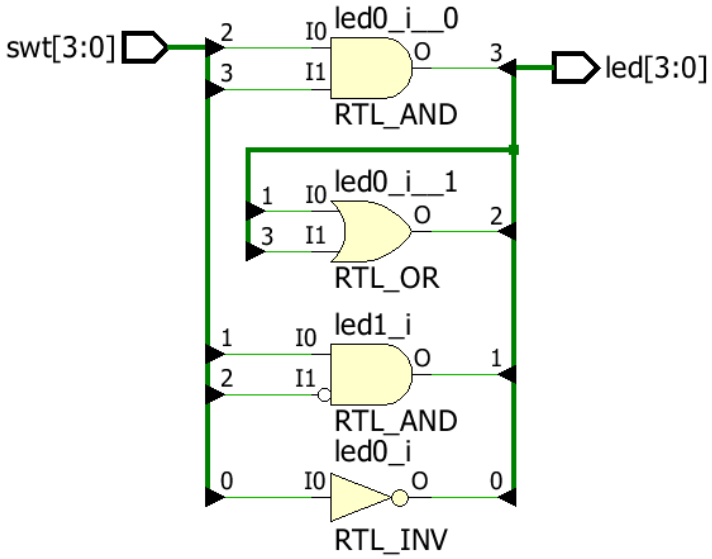


Figure 7. A logic view of the design for the Zybo

Notice that the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs as modeled in the file (that functionality is for the ZedBoard only).

Simulate the Design using the Vivado Simulator — Step 2

2-1. Add the lab1_tb.v testbench file.

2-1-1. Click **Add Sources** under the *Project Manager* tasks of the *Flow Navigator* pane.

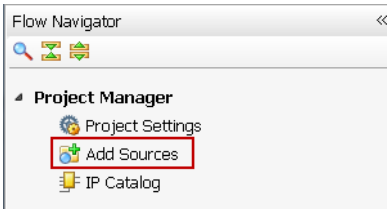


Figure 8. Add Sources

2-1-2. Select the *Add or Create Simulation Sources* option and click **Next**.

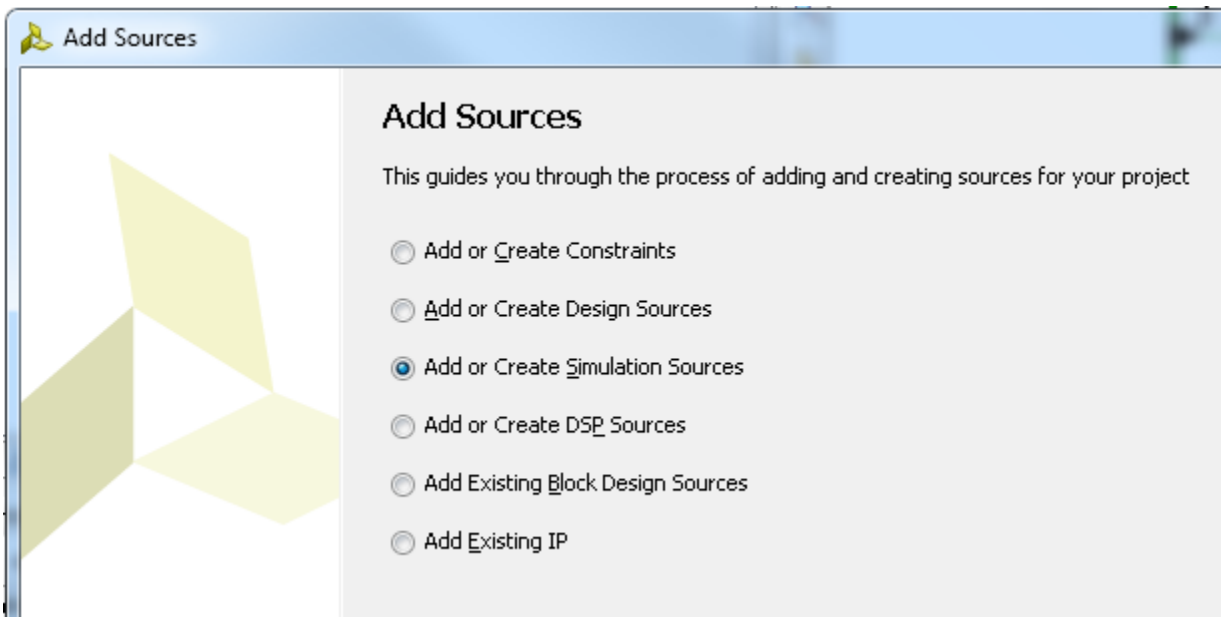


Figure 9. Selecting Simulation Sources option

2-1-3. In the *Add Sources* form, click the **Add Files…** button.

2-1-4. Browse to the **<2014_2_zynq_sources>** folder and the double click on the appropriate **\<board>\lab1** subdirectory. Select *lab1_tb.v,* and click **OK**.

2-1-5. Make sure the file is **copied** into the project and click **Finish**.

2-1-6. Select the *Sources* tab and expand the *Simulation Sources* group.

The lab1_tb.v file is added under the *Simulation Sources* group, and **lab1.v** is automatically placed in its hierarchy as a dut1 instance.

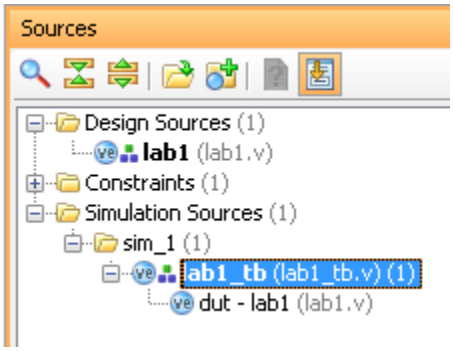


Figure 10. Simulation Sources hierarchy

2-1-7. Using the Windows Explorer, verify that the **sim_1** directory is created at the same level as constrs_1 and sources_1 directories under the lab1.srcs directory, and that a copy of lab1_tb.v is placed under **lab1.srcs > sim_1 > imports > lab1**.

2-1-8. Double-click on the **lab1_tb** in the *Sources* pane to view its contents.

```
1 `timescale 1ns / 1ps
2 ////////////////////////////////////////////////////////////////////////////////
3 // Module Name: lab1_tb
4 ////////////////////////////////////////////////////////////////////////////////
5 module lab1_tb(
6
7     );
8
9     reg [7:0] switches;
10    wire [7:0] leds;
11    reg [7:0] e_led;
12
13    integer i;
14
15    lab1 dut(.led(leds),.swt(switches));
16
17    function [7:0] expected_led;
18       input [7:0] swt;
19    begin
20       expected_led[0] = ~swt[0];
21       expected_led[1] = swt[1] & ~swt[2];
22       expected_led[3] = swt[2] & swt[3];
23       expected_led[2] = expected_led[1] | expected_led[3];
24       expected_led[7:4] = swt[7:4];
25    end
26    endfunction
27
28    initial
29    begin
30        for (i=0; i < 255; i=i+2)
31        begin
32            #50 switches=i;
33            #10 e_led = expected_led(switches);
34            if(leds == e_led)
35                $display("LED output matched at", $time);
36            else
37                $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
38        end
39    end
40
41 endmodule
42
```

Figure 11. The self-checking testbench for the ZedBoard

```
1 `timescale 1ns / 1ps
2 ////////////////////////////////////////////////////////////////////////////////
3 // Module Name: lab1_tb
4 ////////////////////////////////////////////////////////////////////////////////
5 module lab1_tb(
6
7     );
8
9     reg [3:0] switches;
10    wire [3:0] leds;
11    reg [3:0] e_led;
12
13    integer i;
14
15    lab1 dut(.led(leds),.swt(switches));
16
17    function [3:0] expected_led;
18       input [3:0] swt;
19    begin
20       expected_led[0] = ~swt[0];
21       expected_led[1] = swt[1] & ~swt[2];
22       expected_led[3] = swt[2] & swt[3];
23       expected_led[2] = expected_led[1] | expected_led[3];
24    end
25    endfunction
26
27    initial
28    begin
29        for (i=0; i < 15; i=i+1)
30        begin
31            #50 switches=i;
32            #10 e_led = expected_led(switches);
33            if(leds == e_led)
34                $display("LED output matched at", $time);
35            else
36                $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
37        end
38    end
39
40 endmodule
41
```

Figure 11. The self-checking testbench for the Zybo

The testbench defines the simulation step size and the resolution in line 1. The testbench module definition begins on line 5.

For the ZedBoard, line 15 instantiates the DUT (device/module under test). Lines 17 through 26 define the same module functionality for the expected value computation. Lines 28 through 39 define the stimuli generation, and compare the expected output with what the DUT provides. Line 41 ends the testbench. The $display task will print the message in the simulator console window when the simulation is run.

For the Zybo, line 15 instantiates the DUT (device/module under test). Lines 17 through 25 define the same module functionality for the expected value computation. Lines 27 through 38 define the stimuli generation, and compare the expected output with what the DUT provides. Line 40 ends the testbench. The $display task will print the message in the simulator console window when the simulation is run.

2-2. Simulate the design for 200 ns using the Vivado simulator.

2-2-1. Select **Simulation Settings** under the *Project Manager* tasks of the *Flow Navigator* pane.

A **Project Settings** form will appear showing the **Simulation** properties form.

2-2-2. Select the **Simulation** tab, and set the **Simulation Run Time** value to 200ns and click **OK**.

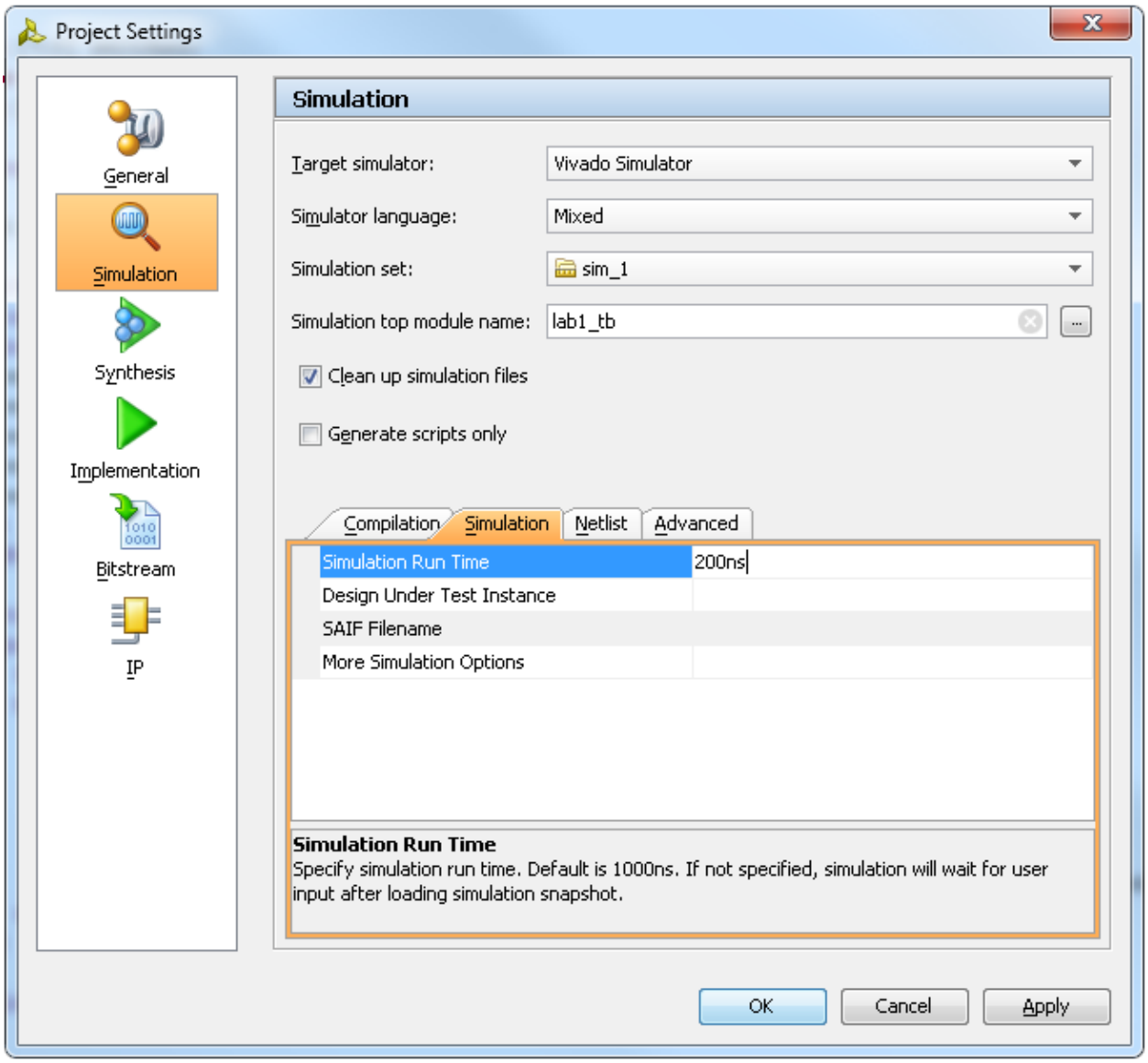


Figure 12. Setting simulation run time

2-2-3. Click on **Run Simulation > Run Behavioral Simulation** under the *Project Manager* tasks of the *Flow Navigator* pane.

The testbench and source files will be compiled and the Vivado simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below.

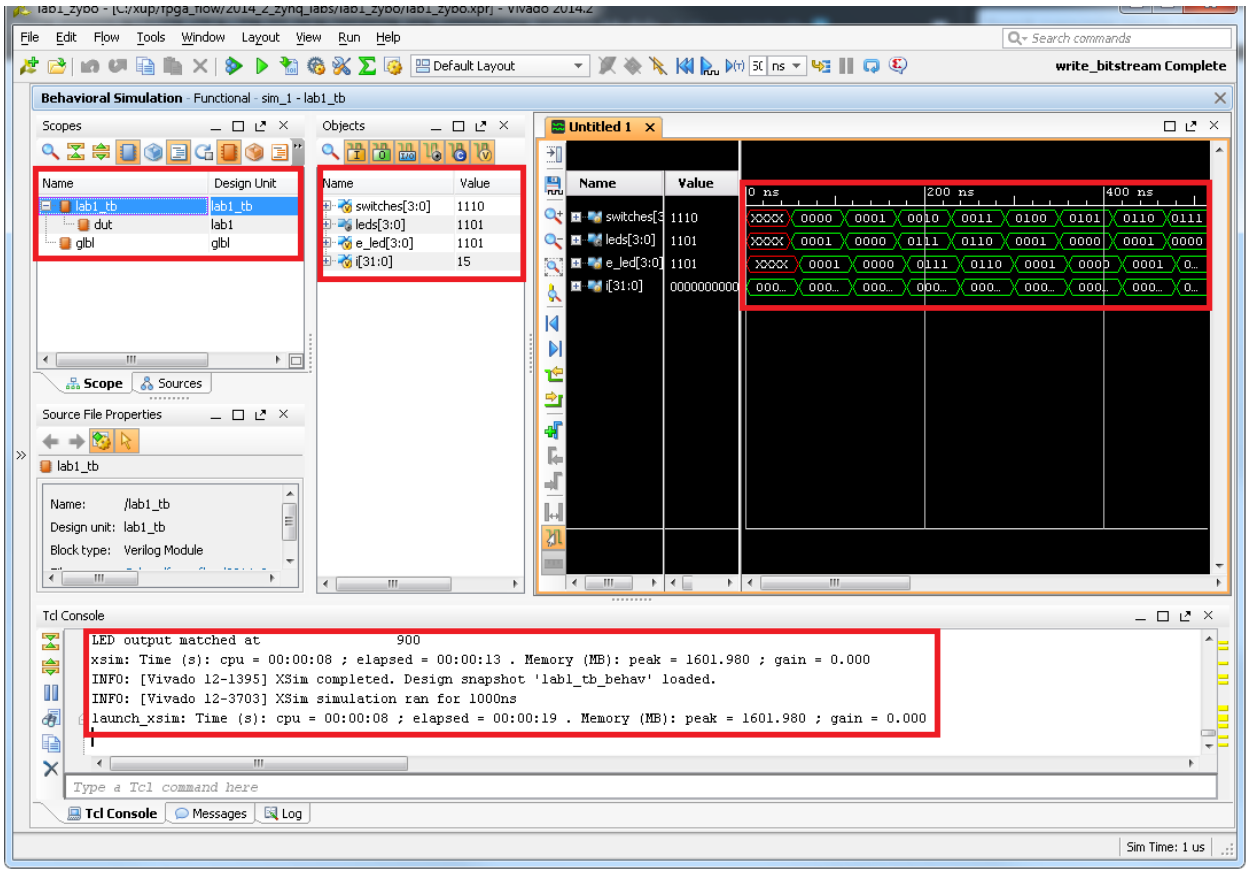


Figure 13. Simulator output

You will see four main views: (i) *Scopes,* where the testbench hierarchy as well as glbl instances are displayed, (ii) *Objects,* where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

Notice that the **lab1.sim** directory is created under the **lab1** directory, along with several lower-level directories.

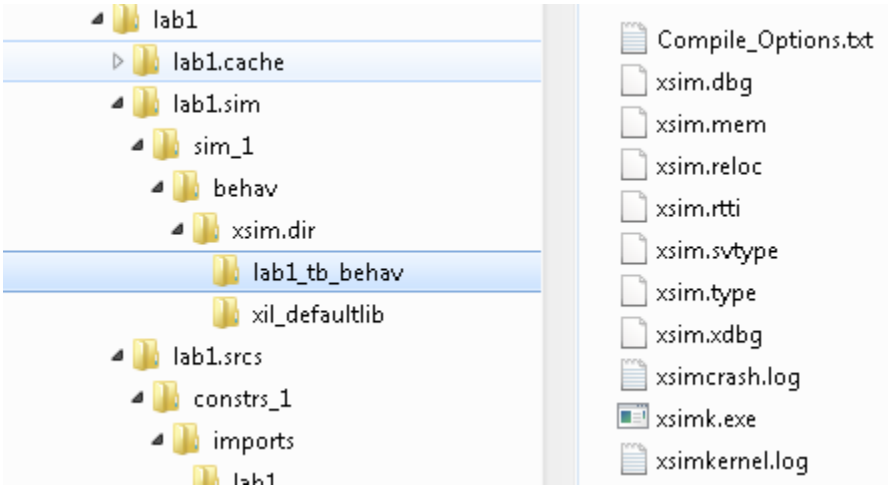


Figure 14. Directory structure after running behavioral simulation

You will see several buttons next to the waveform window which can be used for the specific purpose as listed in the table below.

Table 1: Various buttons available to view the waveform

	Waveform options
	Save the waveform
	Zoom In
	Zoom Out
	Zoom Fit
	Zoom to cursor
	Go to Time 0
	Go to Last Time
	Previous Transition
	Next Transition
	Add Marker
	Previous Marker
	Next Marker
	Swap Cursors
	Snap to Transition
	Floating Ruler

2-2-4. Click on the *Zoom Fit* button ( ) to see the entire waveform.

Notice that the output changes when the input changes.

You can also float the simulation waveform window by clicking on the Float button on the upper right hand side of the view. This will allow you to have a wider window to view the simulation waveforms. To reintegrate the floating window back into the GUI, simply click on the Dock Window button.

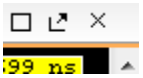


Figure 15. Float Button

Figure 16. Dock Window Button

2-3. Change display format if desired.

2-3-1. Select **i[31:0]** in the waveform window, right-click, select *Radix*, and then select *Unsigned Decimal* to view the for-loop index in *integer* form. Similarly, change the radix of **switches[7:0]** (for the ZedBoard) / **switches[3:0]** (for the Zybo) to *Hexadecimal*. Leave the **leds[7:0]** (for the ZedBoard) / **leds[3:0]** (for the Zybo) and **e_led[7:0]** (for the ZedBoard) / **e_led[3:0]** (for the Zybo) radix to *binary* as we want to see each output bit.

2-4. Add more signals to monitor the lower-level signals and continue to run the simulation for 500 ns.

2-4-1. Expand the **lab1_tb** instance, if necessary, in the *Scope* window and select the **dut** instance.

For the ZedBoard, the swt[7:0] and led[7:0] signals will be displayed in the *Objects* window.

For the Zybo, the swt[3:0] and led[3:0] signals will be displayed in the *Objects* window.

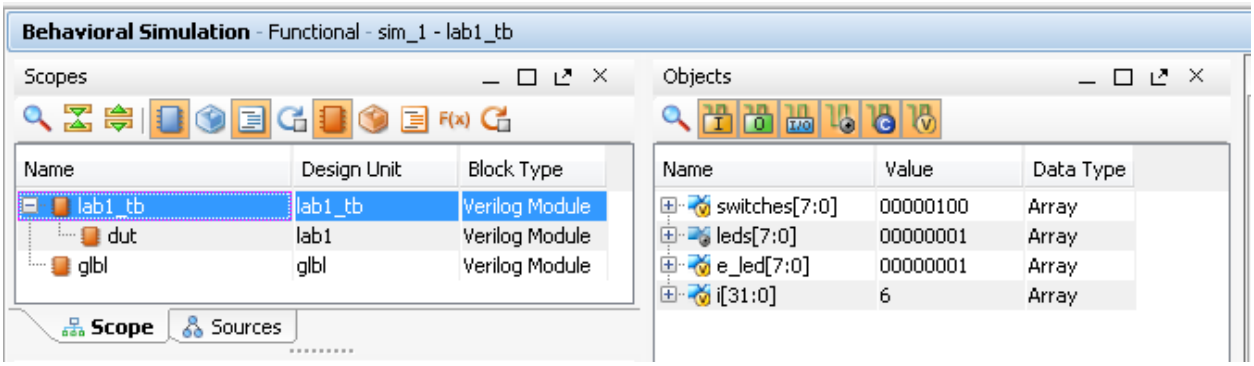


Figure 17. Selecting lower-level signals for the ZedBoard

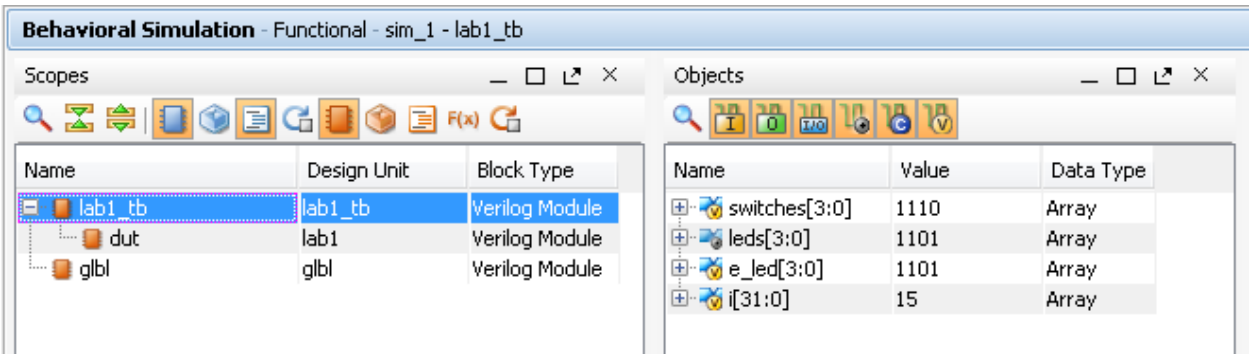


Figure 17. Selecting lower-level signals for the Zybo

2-4-2. For the ZedBoard, select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals. For the Zybo, do so for **swt[3:0]** and **led[3:0]**.

2-4-3. On the simulator tool buttons ribbon bar, type 500, click on the drop-down button of the units field and select ns () if we want to run for 500 ns (total of 700 ns), and click on the ( ) button.

The simulation will run for an additional 500 ns.

2-4-4. Click on the *Zoom Fit* button and observe the output.

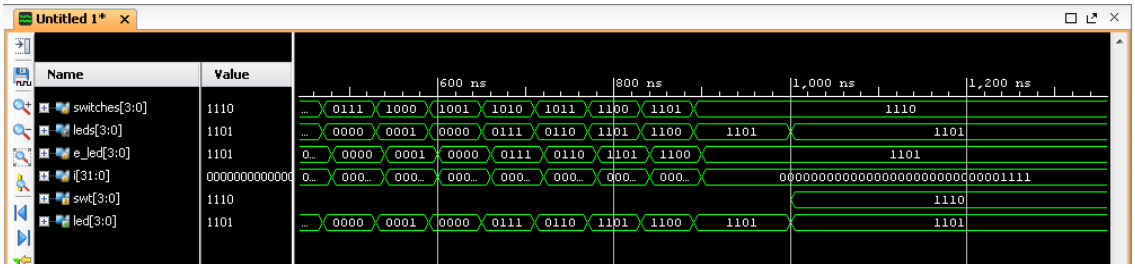


Figure 18. Running simulation for additional 500 ns

Observe the Tcl Console window and see the output is being displayed as the testbench uses the $display task.

```
xsim: Time (s): cpu = 00:00:06 ; elapsed = 00:00:11 . Memory (MB): peak = 1397.586 ; gain = 17.715
INFO: [Vivado 12-1395] XSim completed. Design snapshot 'lab1_tb_behav' loaded.
INFO: [Vivado 12-3703] XSim simulation ran for 200ns
launch_xsim: Time (s): cpu = 00:00:07 ; elapsed = 00:00:14 . Memory (MB): peak = 1397.586 ; gain =
run 500 ns
LED output matched at                    240
LED output matched at                    300
LED output matched at                    360
LED output matched at                    420
LED output matched at                    480
LED output matched at                    540
LED output matched at                    600
LED output matched at                    660
```

Figure 19. Tcl Console output after running the simulation for additional 500 ns

2-4-5. Close the simulator by selecting **File > Close Simulation**.

2-4-6. Click **OK** and then click **No** to close it without saving the waveform.

Synthesize the Design — Step 3

3-1. Synthesize the design with the Vivado synthesis tool and analyze the Project Summary output.

3-1-1. Click on **Run Synthesis** under the *Synthesis* task of the *Flow Navigator* pane.

The synthesis process will be run on the lab1.v file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

3-1-2. Select the *Open Synthesized Design* option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage.

Click **Yes** to close the elaborated design if the dialog box is displayed.

3-1-3. Select the **Project Summary** tab and understand the various windows.

If you don't see the Project Summary tab then select **Layout > Default Layout,** or click the **Project Summary** icon **.**

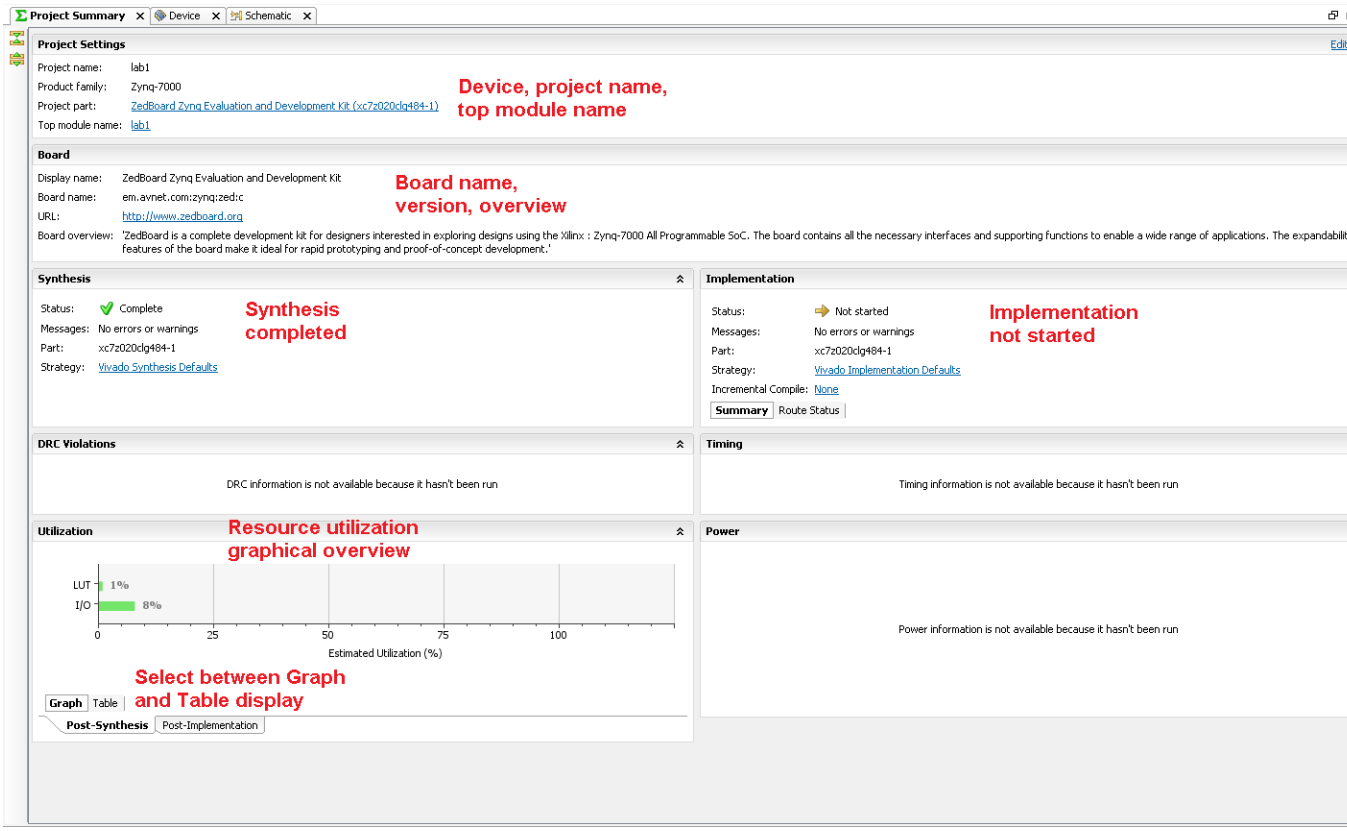


Figure 20. Project Summary view

Click on the various links to see what information they provide and which allows you to change the synthesis settings.

3-1-4. Click on the **Table** tab in the **Project Summary** tab.

Notice that there are an estimated three LUTs and 16 IOs (8 input and 8 output) that are used for the ZedBoard. There are only 8 IOs used for the Zybo (4 input and 4 output). The number of total available pins differ based on the target device used with the XC7Z010 having less pins than the XC7Z020.

Resource	Estimation	Available	Utilization %
LUT	3	53200	0.01
I/O	16	200	8.00

Figure 21. Resource utilization estimation summary for the ZedBoard

Resource	Estimation	Available	Utilization %
LUT	3	17600	0.02
I/O	8	100	8.00

Figure 21. Resource utilization estimation summary for the Zybo

3-1-5. In The *Flow Navigator*, under *Synthesis* (expand *Synthesized Design* if necessary), click on **Schematic** to view the synthesized design in a schematic view.

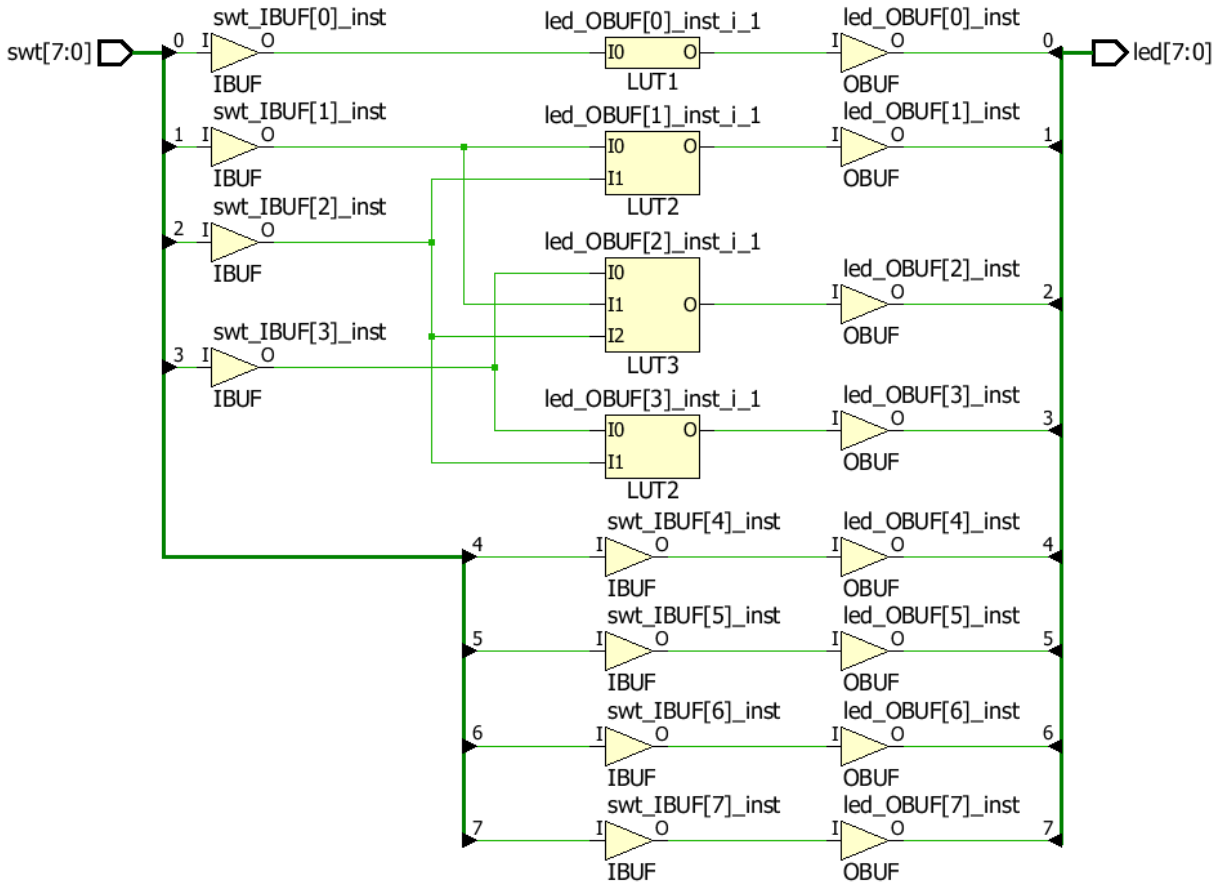


Figure 22. Synthesized design's schematic view for the ZedBoard

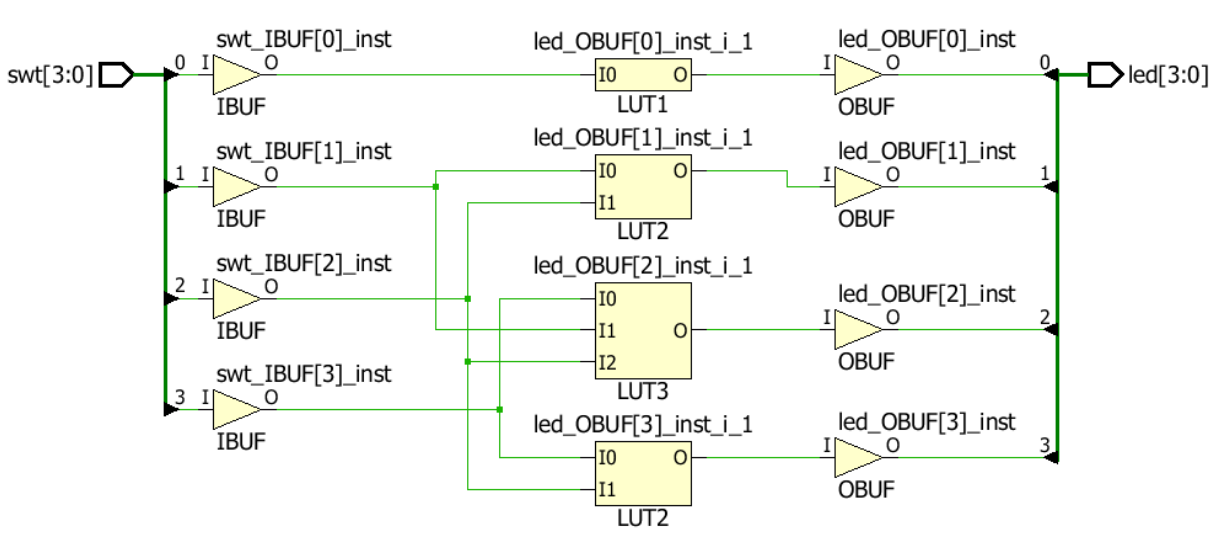


Figure 22. Synthesized design's schematic view for the Zybo

Notice that IBUFs and OBUFs are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs (1 input is listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). The four gates in RTL analysis output are mapped onto four LUTs in the synthesized output. Also notice that the design is effectively split in

two between the two boards, with one half being available both the ZedBoard and Zybo (LUTs) while the other half toggled by the switches (straight through connection) being only available to the ZedBoard. This is due to the way the two designs are coded.

Using Windows Explorer, verify that **lab1.runs** directory is created under **lab1**. Under the **runs** directory, **synth_1** directory is created which holds several files related to synthesis.

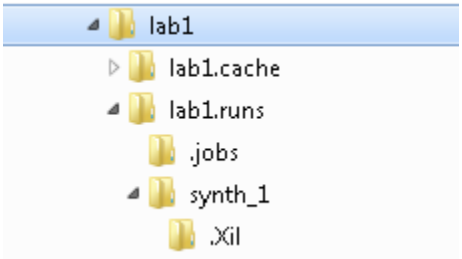


Figure 23. Directory structure after synthesizing the design

Implement the Design Step 4

4-1. Implement the design with the Vivado Implementation Defaults (Vivado Implementation 2014) settings and analyze the Project Summary output.

4-1-1. Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane.

The implementation process will be run on the synthesized design. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.

4-1-2. Select **Open implemented design** and click **OK** as we want to look at the implemented design in a Device view tab.

4-1-3. Click **Yes,** if prompted, to close the synthesized design.

The implemented design will be opened.

4-1-4. In the *Netlist* pane, select one of the nets by expanding the Nets branch (e.g. led_OBUF[0]) and notice that the net displayed in the Device view tab (you may have to zoom fit in to see it).

4-1-5. If it is not selected, click the *Routing Resources* icon  to show the routing resources.

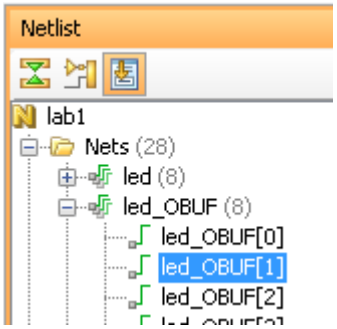


Figure 24. Selecting a net

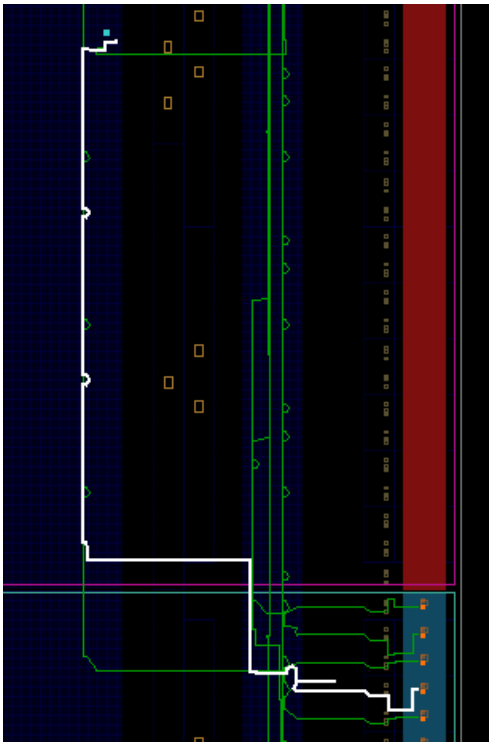


Figure 25. Viewing the implemented ZedBoard design

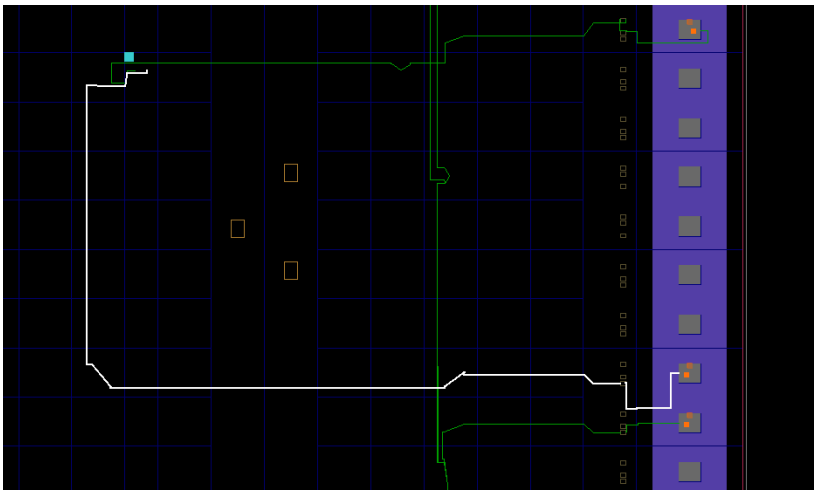


Figure 25. Viewing the implemented Zybo design

4-1-6. Close the implemented design view and select the **Project Summary** tab (you may have to change to the Default Layout view) and observe the results.

Select the Post-Implementation tab in the **Utilization** pane.

Notice that the actual resource utilization is 3 LUTs and 16 IOs for the ZedBoard and 3 LUTs and 8 IOs for the Zybo. Also, it indicates that no timing constraints were defined for this design (since the design is combinatorial).

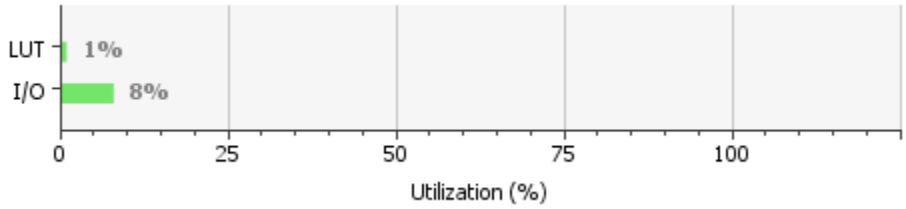


Figure 26. Post implementation resources chart for both boards

Using the Windows Explorer, verify that **impl_1** directory is created at the same level as **synth_1** under the **lab1.runs** directory. The **impl_1** directory contains several files including the implementation report files.

4-1-7. In Vivado, select the **Reports** tab in the bottom panel (if not visible, click *Window* in the menu bar and select **Reports**), and double-click on the *Utilization Report* entry under the *Place Design* section. The report will be displayed in the auxiliary view pane showing resource utilization. Note that since the design is combinatorial no registers are used.

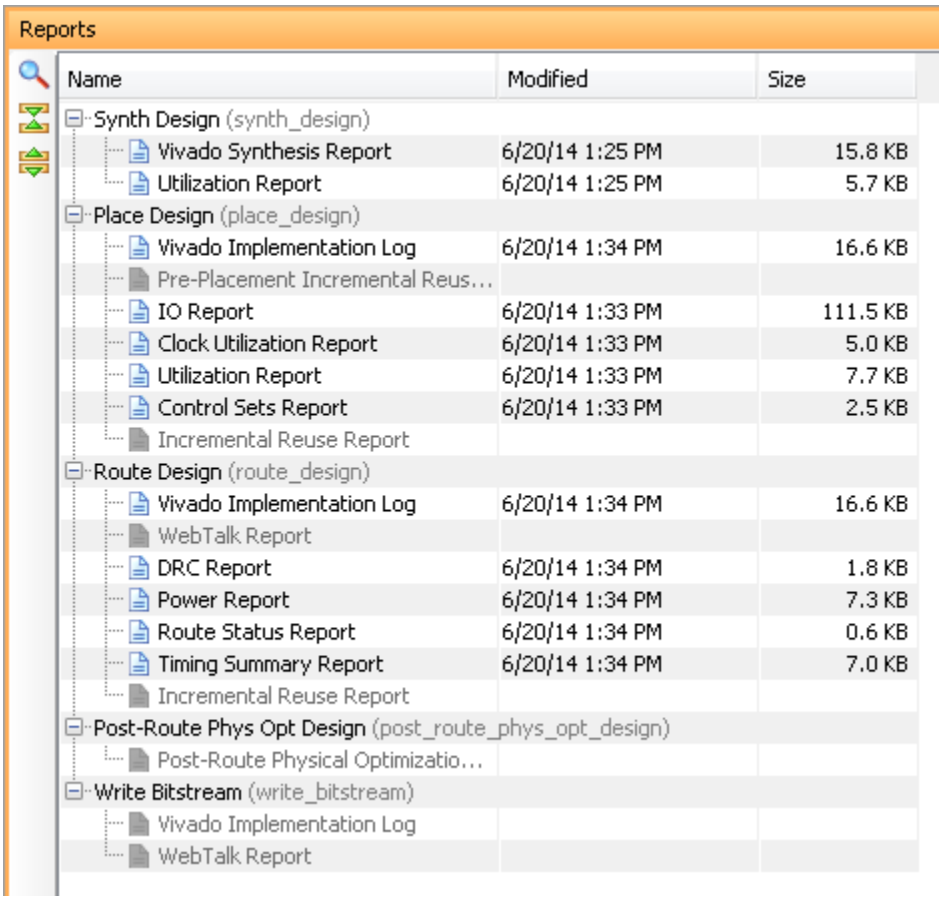


Figure 27. Available reports to view

Perform Timing Simulation — Step 5

5-1. Run a timing simulation.

5-1-1. Select **Run Simulation > Run Post-Implementation Timing Simulation** process under the *Simulation* tasks of the *Flow Navigator* pane.

The Vivado simulator will be launched using the implemented design and **lab1_tb** as the top-level module.

Using the Windows Explorer, verify that **timing** directory is created under the **lab1.sim > sim_1 > impl** directory. The **timing** directory contains generated files to run the timing simulation.

5-1-2. Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.

5-1-3. Click on the 50 ns mark (where the switch input is set to 0000000b for the ZedBoard or 0000b for the Zybo) and a marker is automatically placed at the location.

5-1-4. You can also add a marker by clicking on the Add Marker button ( ). Click on the **Add Marker** button and left-click at around 60 ns where **e_led** changes.

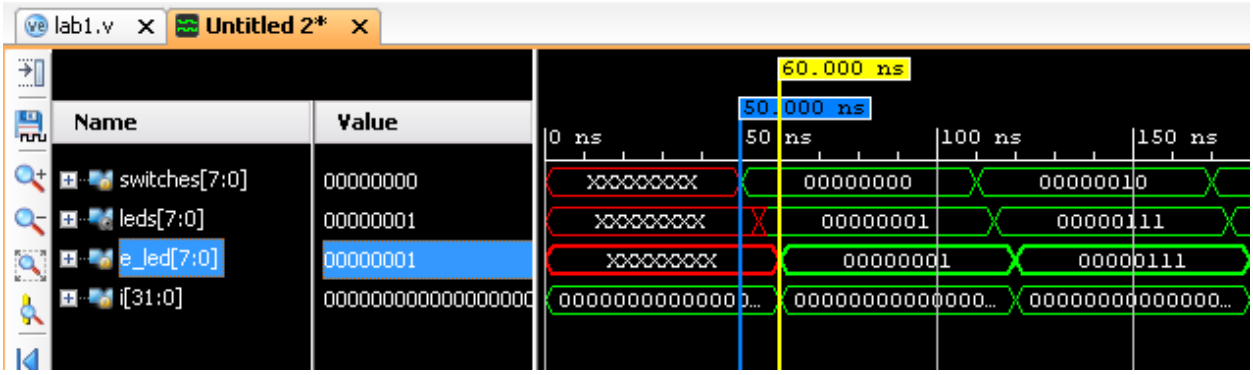


Figure 28. Timing simulation output for the ZedBoard

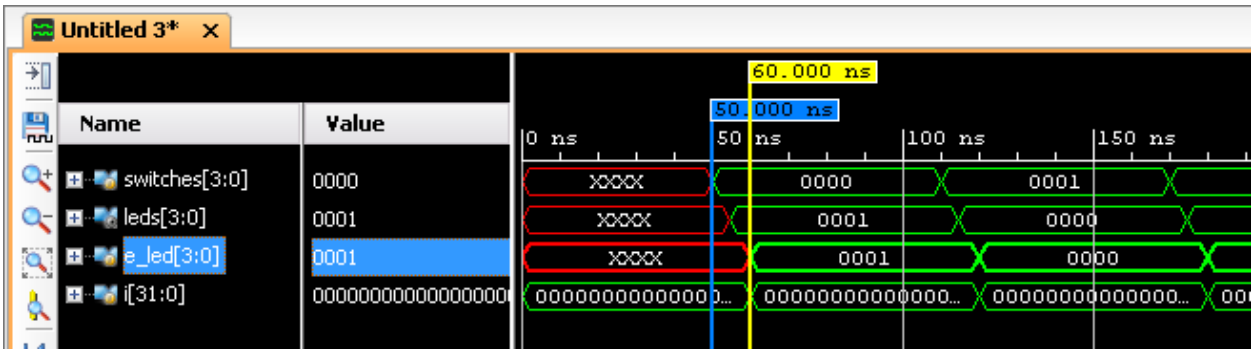


Figure 28. Timing simulation output for the Zybo

Notice that we monitored the expected led output at 10 ns after the input is changed (see the testbench) whereas the actual delay is about 5.000 ns where leds[7:0] or leds[3:0] changes.

5-1-5. Close the simulator by selecting **File > Close Simulation** without saving any changes.

Generate the Bitstream and Verify Functionality Step 6

6-1. Connect the board and power it ON. Generate the bitstream, open a Hardware Manager, and program the FPGA.

6-1-1. Make sure that the Micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Connect the power jack for the ZedBoard (none needed for the Zybo).

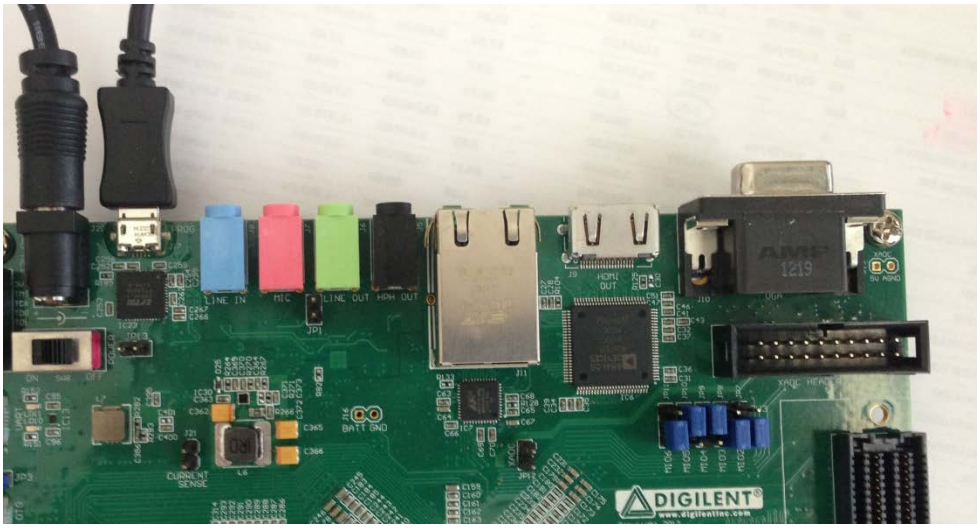


Figure 29. ZedBoard connections

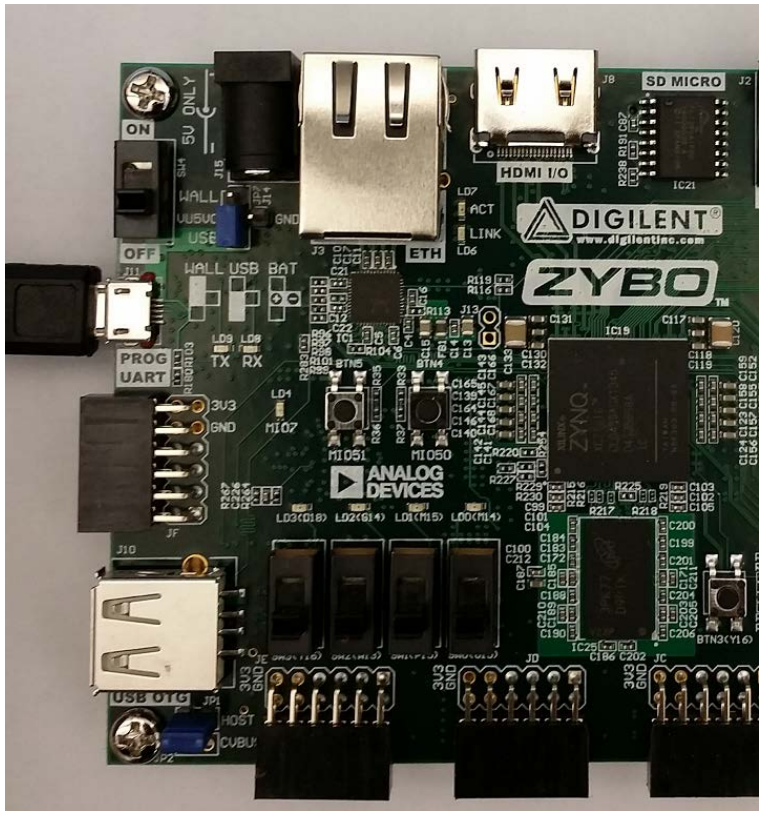


Figure 29. Zybo connection

6-1-2. Power **ON** the board.

6-1-3. Click on the **Generate Bitstream** entry under the *Program and Debug* tasks of the *Flow Navigator* pane.

The bitstream generation process will be run on the implemented design. When the process is completed a *Bitstream Generation Completed* dialog box with two (or three) options will be displayed.

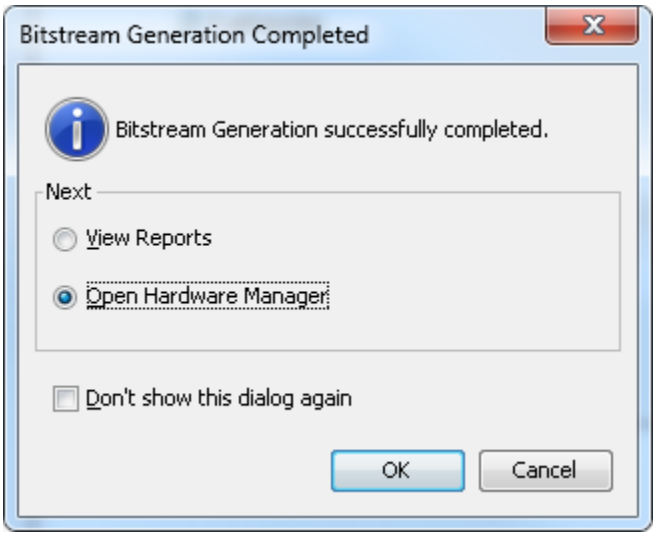


Figure 30. Bitstream generation

This process will have generated a **lab1.bit** file under **impl_1** directory in the **lab1.runs** directory.

If the dialog box is not displayed, to perform the next step, simply select the **Open Hardware Manager** option under *Program and Debug* in the *Flow Navigator* pane.

6-1-4. Select the *Open Hardware Manager* option and click **OK**.

The Hardware Manager window will open indicating "unconnected" status.

6-1-5. Click on the **Open New Hardware Target** link.

You can also click on the Open Recent Hardware Target link if the board was already targeted before.

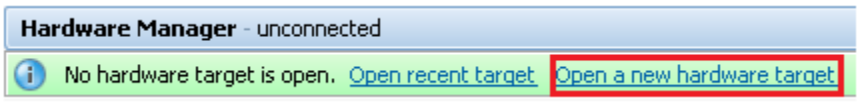


Figure 31. Opening new hardware target

6-1-6. Click **Next** to see the Hardware Server Settings form.

6-1-7. Click **Next** with the Hardware Target selected.

The JTAG cable which should be detected and identified as a hardware target. It will also show the hardware devices detected in the chain.

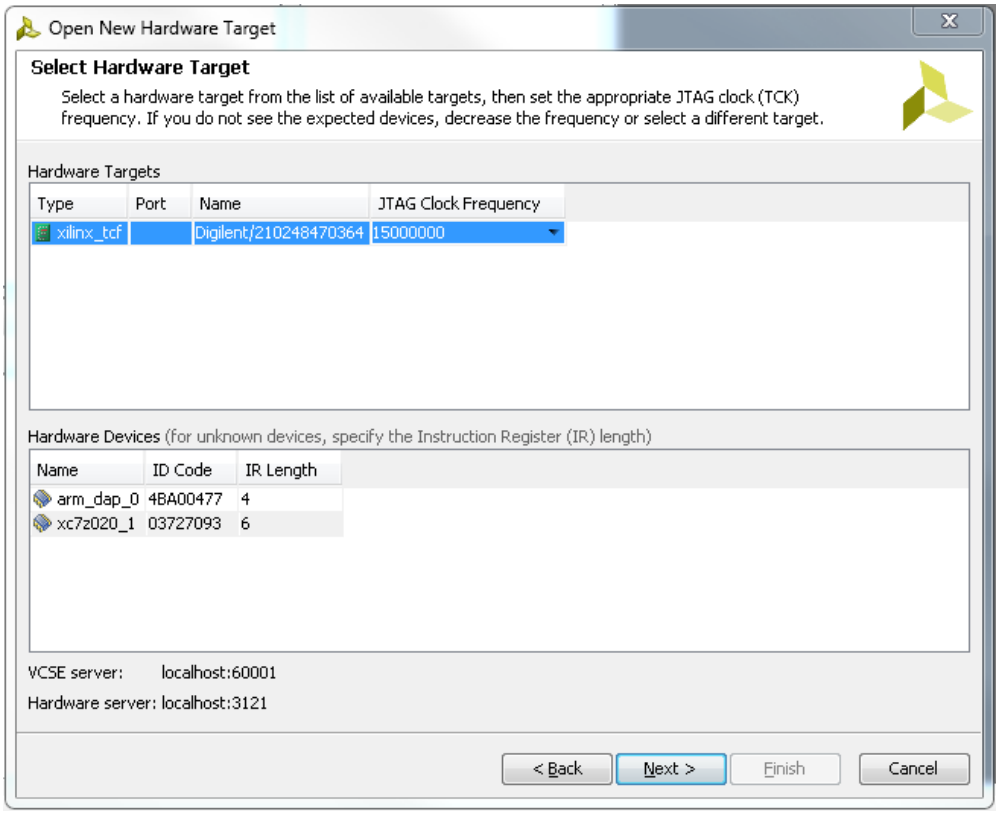


Figure 32. New hardware target detection for the ZedBoard

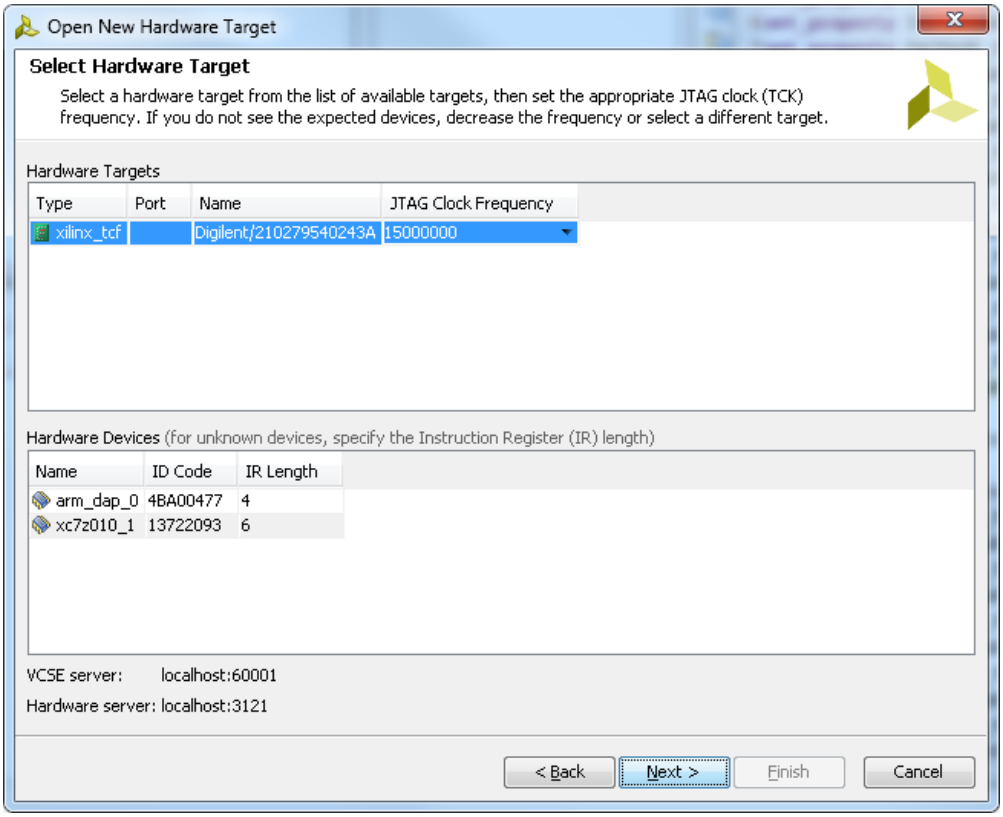


Figure 32. New hardware target detection for the Zybo

6-1-8. Click **Next** and then **Finish**.

The Hardware Manager status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.

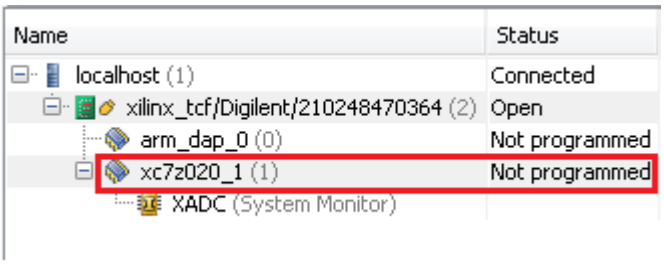


Figure 33. Opened Hardware Manager for the ZedBoard

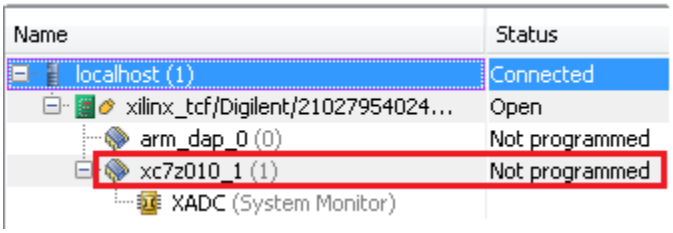


Figure 33. Opened Hardware Manager for the Zybo

6-1-9. Select the XC7Z010 or XC7Z020 device and verify that the lab1.bit is selected as the programming file in the General tab.

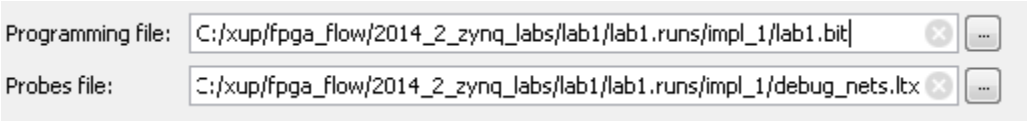


Figure 34. Programming file to be configured into the target device

6-1-10. **Right-click** on the device and select *Program Device…* to program the target FPGA device.

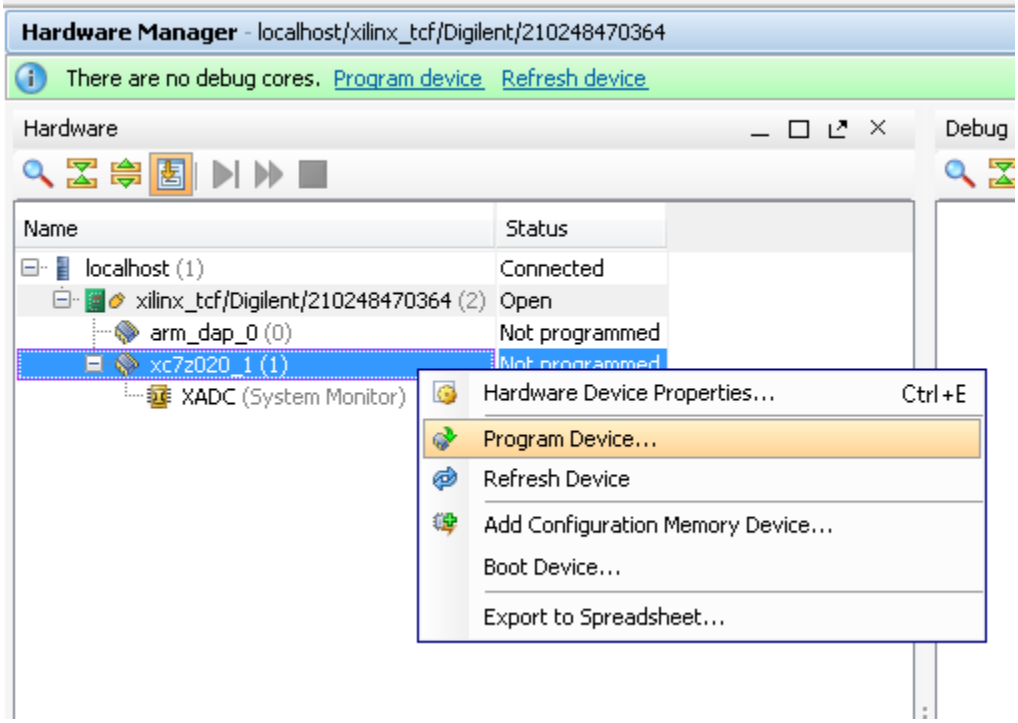


Figure 35. Selecting to program the FPGA

6-1-11. Click **Program** to program the FPGA.

The DONE light (Blue LED on the ZedBoard and Green LED on the Zybo) will light when the device is programmed. You may see some other LEDs lit depending on switch positions.

6-1-12. Verify the functionality by flipping switches and observing the output on LEDs (Refer to the earlier logic diagram/elaboration schematics).

6-1-13. When satisfied, power **OFF** the board.

6-1-14. Close the Hardware Manager by selecting **File > Close Hardware Manager.**

6-1-15. Click **OK** to close the session.

6-1-16. Close the **Vivado** program by selecting **File > Exit** and click **OK**.

Conclusion

The Vivado software tool can be used to perform a complete design flow. The project was created using the supplied source files (HDL model and user constraint file). A behavioral simulation using the provided testbench was done to verify the model functionality. The model was then synthesized, implemented, and a bitstream was generated. The timing simulation was run on the implemented design using the same testbench. The functionality was verified in hardware using the generated bitstream.