



Embedded Systems Design

14 : 332 : 493 : 03 / 16 : 332 : 579 : 05

Lab Intro: Vivado Design Flow

Introduction

This lab guides you through the process of using Vivado IDE to create a simple HDL design targeting the Zybo board (Zybo or Zybo Z7-10 versions). You will simulate, synthesize, and implement the design with default settings. Finally, you will generate the bitstream and download it into 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 the ZYNQ device located on the 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 ZYNQ 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 lab.

Note: You will notice certain procedures have different variations depending on development board being ZedBoard or Zybo. It will be explicitly mentioned in notes when such variation is encountered. This manual will be targeting the Zybo (or Zybo Z7-10) board specifically for Rutgers students.

Design Description

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

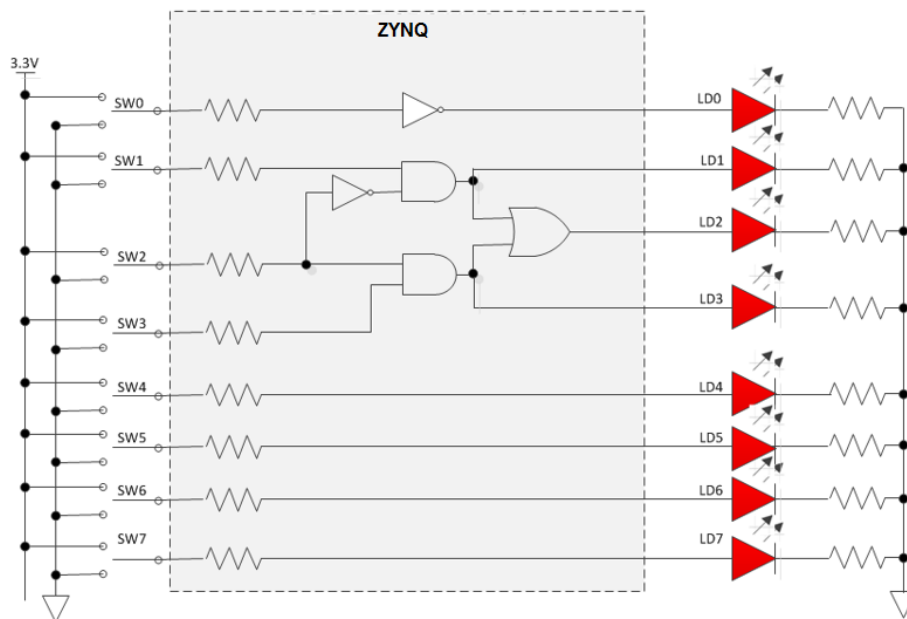
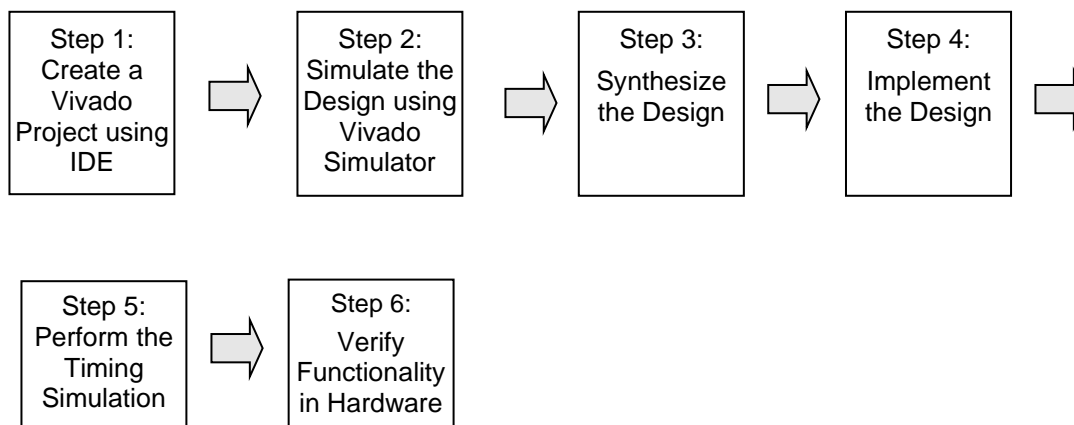


Figure 1. The Completed Design

Note: SW4 – SW7 and LD4 – LD7 do not exist on Zybo and hence will be routed to PMOD pins instead. ZedBoard will use the completed design in Figure 1 as is. HDL source files will remain common for both ZedBoard and Zybo. Zybo routing changes to external PMOD for pins SW4 – SW7 and LD4 – LD7 will be made through their respective XDC files.

General Flow



Create a Vivado Project using IDE

Step 1

- 1-1. **Launch Vivado and create a project targeting the XC7Z010clg400-1 (Zybo), and using the VHDL HDL. Use the provided lab_intro.vhd and lab_intro_Zybo.xdc files from the imported directory posted on CANVAS.**
 - 1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2021.1 > Vivado 2021.1** or **double clicking on the Vivado icon on the Rutgers Virtual ECE Workstation**. (Or select the Vivado version installed on your computer)
 - 1-1-2. Click **Create 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 a **predefined working directory of your choice**, and click **Select**.
 - 1-1-4. Enter **lab_intro** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

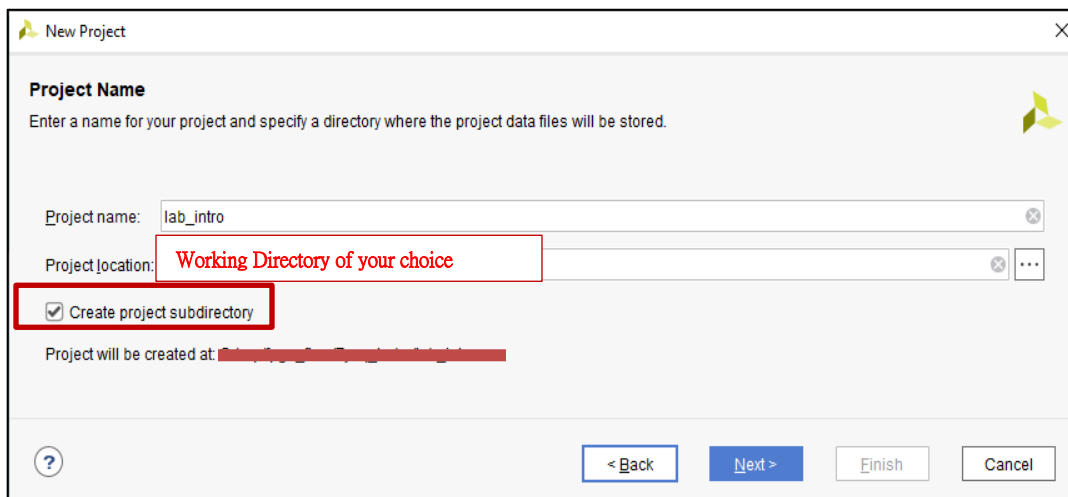


Figure 2. Project Name and Location entry

- 1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.
- 1-1-6. Using the drop-down buttons, select **VHDL** as the *Target Language* and **Mixed Simulator Language** in the *Add Sources* form.

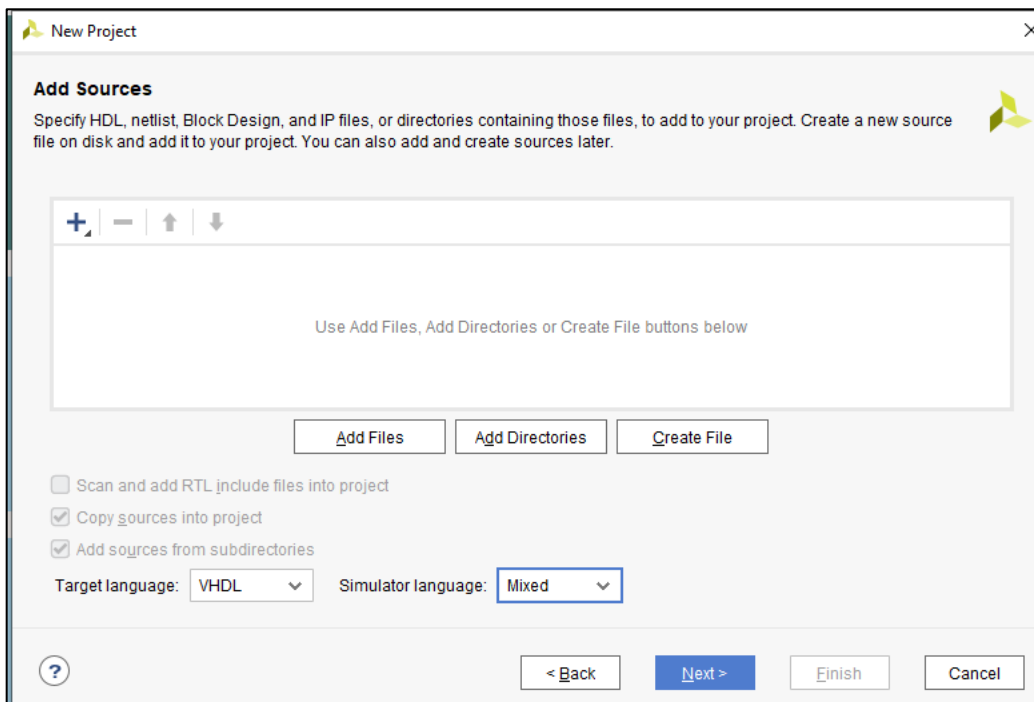


Figure 3. Selecting Target and Simulator language

- 1-1-7.** Click on the **Plus (+)** button, then **Add Files...** and browse to **<Your Working Directory>lab_intro** directory or directory where you have the files downloaded from Canvas for this lab, select *lab_intro.vhd*, click **OK**.

If it isn't already checked, check **Copy sources into project** and then click **Next** to get to the *Add Existing IP* form.

- 1-1-8.** Since we do not have any IP to add, click **Next** to get to the *Add Constraints* form.

- 1-1-9.** Click on the **Plus (+)** button, then **Add Files...** and browse to **<Your Working Directory>lab_intro** directory (if necessary), select *lab_intro_Zybo.xdc* and click **OK** (if necessary), and then click **Next**.

This Xilinx Design Constraints file assigns the physical IO locations on FPGA to the switches 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, use the **Parts** option and various drop-down fields of the **Filter** section. If using the Zybo, select the **XC7Z010clg400-1**. Or Select **Boards** to select Zybo.<Part XC7Z010clg400-1>

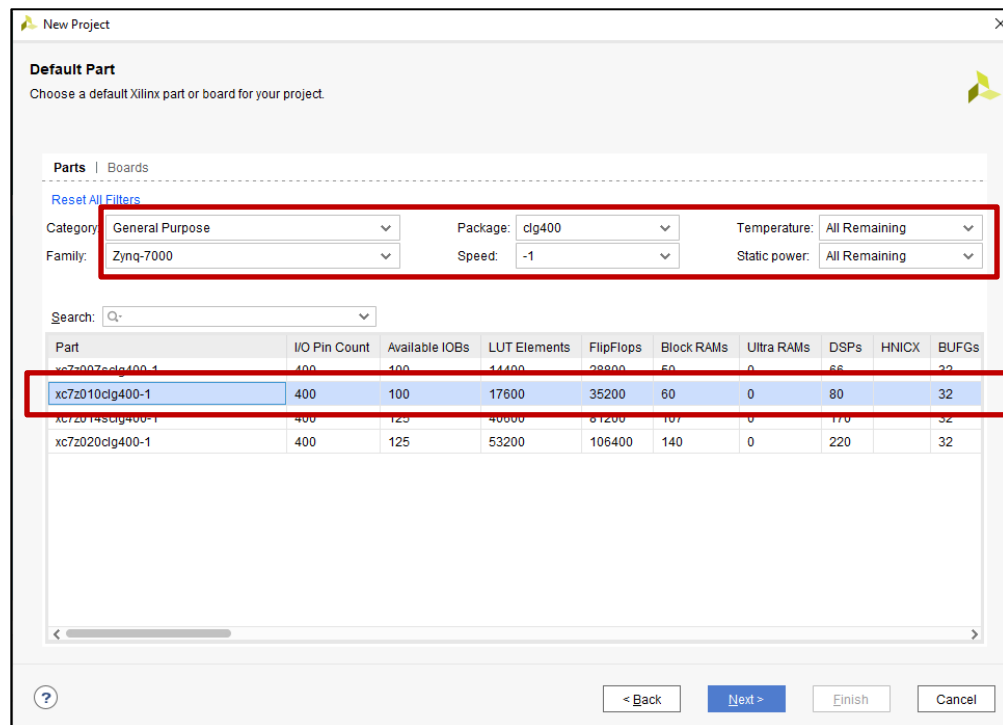


Figure 4. Part Selection for the Zybo

You may also select the **Boards** option, select **digilentinc.com** under the Vendor filter and select the appropriate board. Notice that Zybo may not be listed as they are not in the tools database. If not listed then you can download the board files for the desired boards either from Digilent Inc website: https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-vitis#install_digilent_s_board_files (tool restart may be needed to refresh board options). These boards are already options available on the Vivado tool on vlab.rutgers.edu.

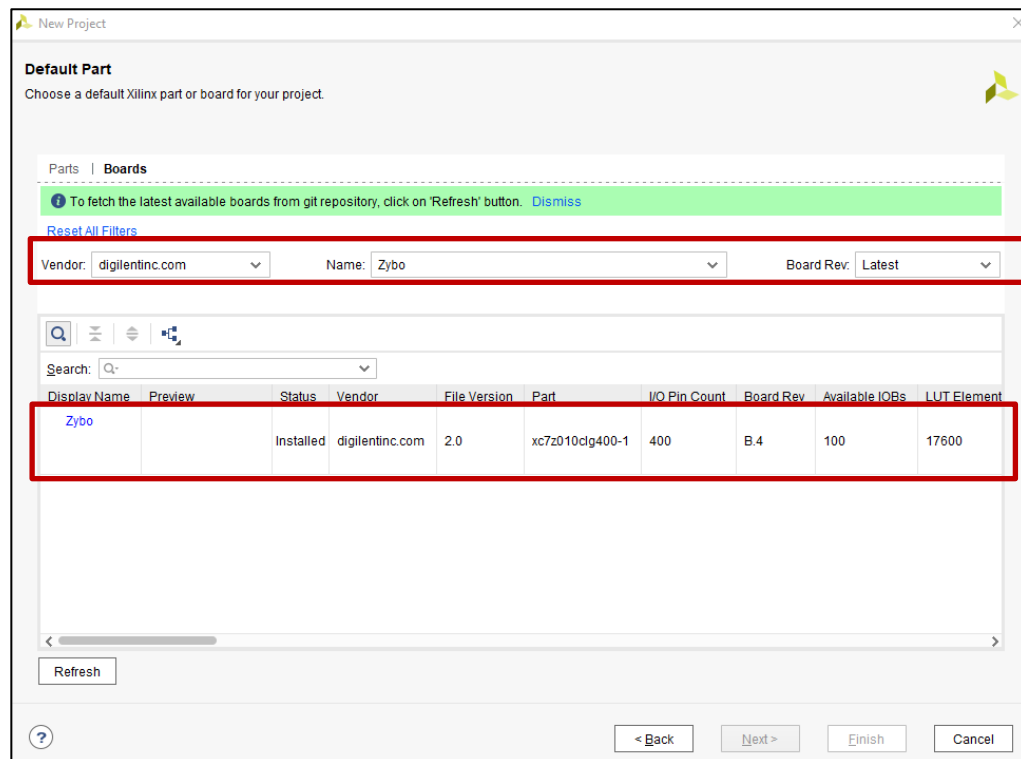


Figure 5. Selecting the target board

1-1-11. Click **Next**.

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

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

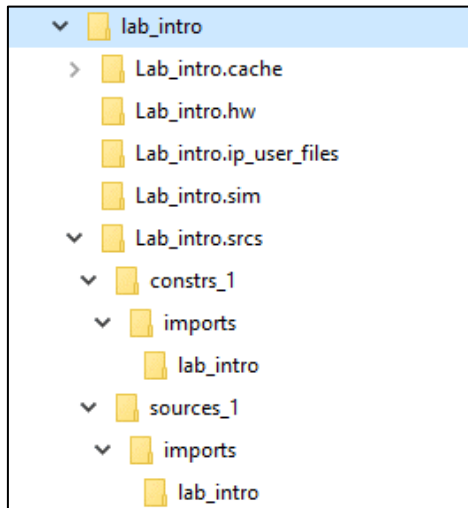


Figure 6. Generated directory structure

1-2. Open the lab_intro.vhd source and analyze the content.

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

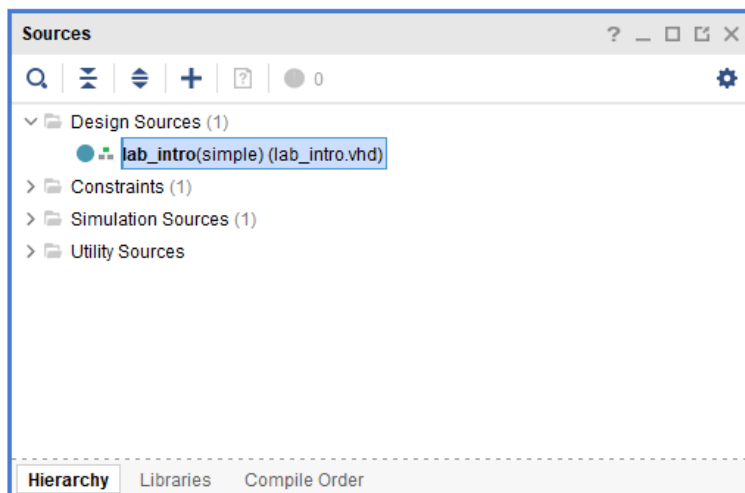


Figure 6. Opening the source file

1-2-2. Notice in the VHDL code the libraries required for proper compilation of the code (Lines 11-12)

1-2-3. Line 17 defines the beginning (marked with keyword **entity**) and Line 22 defines the end of the entity declaration (marked with keyword **end lab_intro**).

- 1-2-4.** Lines 19-20 defines the input and output ports whereas lines 27-36 defines the actual functionality (architecture).

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

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

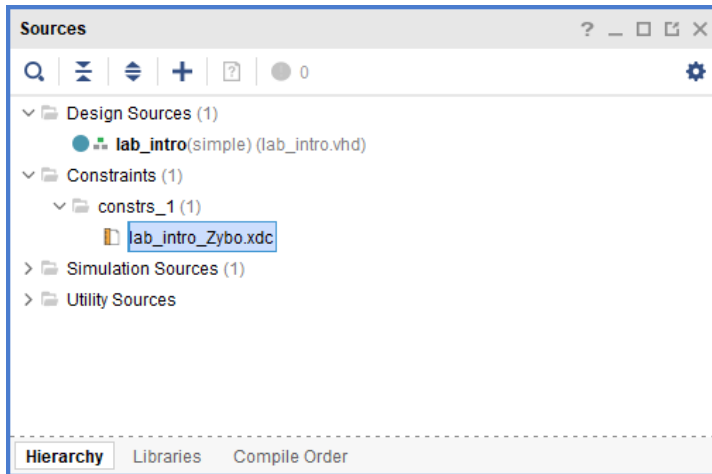


Figure 7. Opening the constraint file

- 1-3-2.** For the lab_intro_Zybo.xdc file lines 5-8 and 13-16 define the pin locations for the input switches and lines 21-24 and 29-32 define pin locations for output LEDs.

Note: SW4 – SW7 and LD4 – LD7 do not exist on Zybo and hence will be routed to PMOD pins instead. Zybo routing changes to external PMOD for pins SW4 – SW7 and LD4 – LD7 can be seen on lines 13-16 and 29-32 in the lab_intro_Zybo.xdc file.

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 (left most pane) and click on **Schematic**.

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

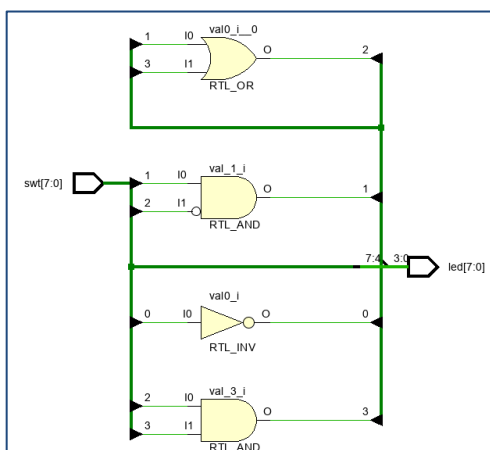


Figure 8. A logic view of the design

Notice that some of the switch inputs go through gates before being output to LEDs through output buffers and the rest go through input buffers and output buffers to LEDs as modeled in the file.

Simulate the Design using the Vivado Simulator

Step 2

2-1. Add the lab_intro_tb.vhd testbench file.

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

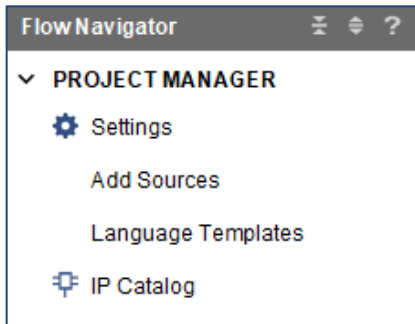


Figure 9. Add Sources

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

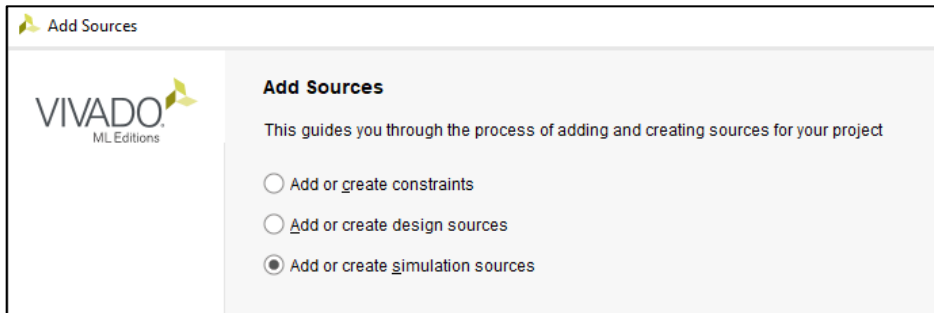


Figure 10. Selecting Simulation Sources option

2-1-3. In the *Add Sources Files* form, click the **Plus (+)** button and then **Add Files...**

2-1-4. Browse to the **<Your Working Directory>lab_intro** folder and select *lab_intro_tb.vhd* and click **OK**.

2-1-5. Click **Finish**.

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

The lab_intro_tb.vhd file is added under the *Simulation Sources* group, and **lab_intro.vhd** is automatically placed in its hierarchy as a dut (device under test) instance.

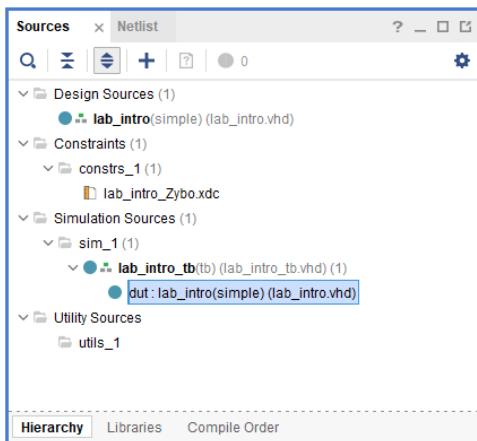


Figure 11. Simulation Sources hierarchy

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 **lab_intro.srscs** directory, and that a copy of **lab_intro_tb.vhd** is placed under **lab_intro.srscs > sim_1 > imports > lab_intro**.

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

```

10  library ieee;
11  use ieee.std_logic_1164.all;
12  use ieee.numeric_std.all;
13
14  -----
15  -- empty entity for testbench
16  -----
17  entity lab_intro_tb is
18  end lab_intro_tb;
19
20  -----
21  -- logic for testing lab intro entity
22  -----
23  architecture tb of lab_intro_tb is
24
25      -- component declaration to use lab intro design in our testbench
26      component lab_intro
27      port (
28          swt : in std_logic_vector (7 downto 0);
29          led : out std_logic_vector (7 downto 0)
30      );
31  end component;
32
33      -- internal signals to interface with lab1 component
34      signal swt : std_logic_vector(7 downto 0) := (others => '0');
35      signal led : std_logic_vector(7 downto 0) := (others => '0');
36
37      -- signal for expected output
38      signal e_led : std_logic_vector(7 downto 0) := (others => '0');
39
40  begin
41
42      -----
43      -- instantiate component to be tested
44      -----
45      dut: lab_intro port map (
46          swt => swt,
47          led => led
48      );
49
50      -----
51      -- calculate expected outputs
52      -----

```

Figure 12. Snapshot of the self-checking testbench

The testbench architecture definition begins on line 23. Line 45 instantiates the DUT (device/module under test) declared as a component in Line 26-31. Lines 50 through 94 define the stimuli generation, and compare the expected output with what the DUT provides. Line 97 ends the testbench. The “report” 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 200 ns and click **OK**.

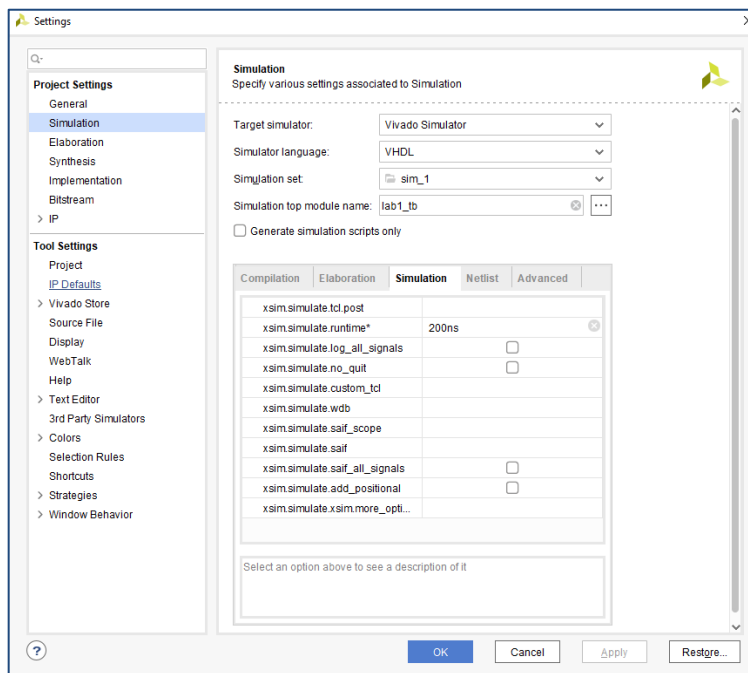


Figure 13. Setting simulation run time

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.

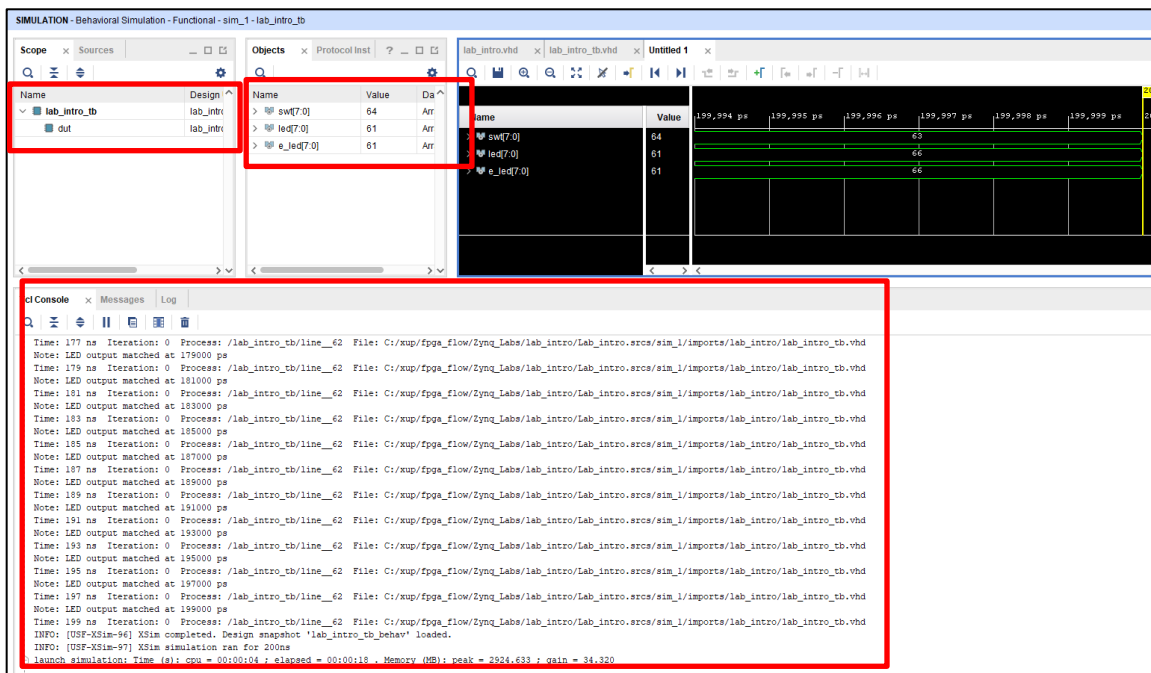


Figure 14. Simulator output

You will see four main views: (i) *Scopes*, where the testbench hierarchy as well as gbl 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 **lab_intro.sim** directory is created under the **lab_intro** directory, along with several lower-level directories.

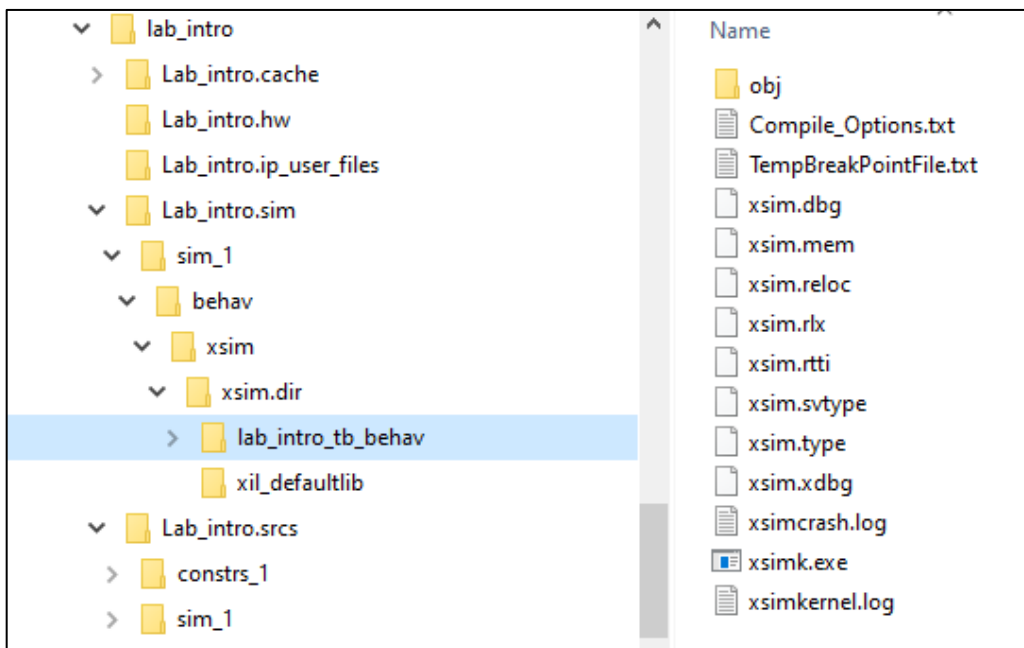


Figure 15. Directory structure after running behavioral simulation

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


















	Search
	Save the waveform
	Zoom In
	Zoom Out
	Zoom Fit
	Unselect All
	Go to Time Cursor
	Go to Time 0
	Go to Last Time
	Previous Transition
	Next Transition
	Add Marker
	Previous Marker
	Next Marker
	Delete all Markers
	Swap Cursors

Table 1: Various buttons available to view the waveform

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

Notice that the output (led 7:0) changes when the input (swt 7:0) 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.

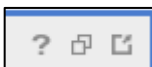


Figure 16. Float Button



Figure 17. Dock Window Button

2-3. Change display format if desired.

Select **switches[7:0]** in the waveform window, right-click, select *Radix*, and then select *Hexadecimal*. Leave the **leds[7:0]** and **e_led[7:0]** radix to *binary* (or change if not in 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 **lab_intro_tb** instance, if necessary, in the *Scopes* window and select the **dut** instance.

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

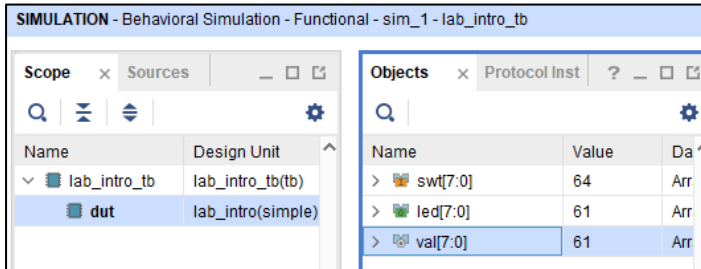


Figure 18. Selecting lower-level signals

2-4-2. Select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals.

2-4-3. On the simulator tool buttons ribbon bar, type 500 over in the simulation run time field, click on the drop-down button of the units field and select ns () since 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.

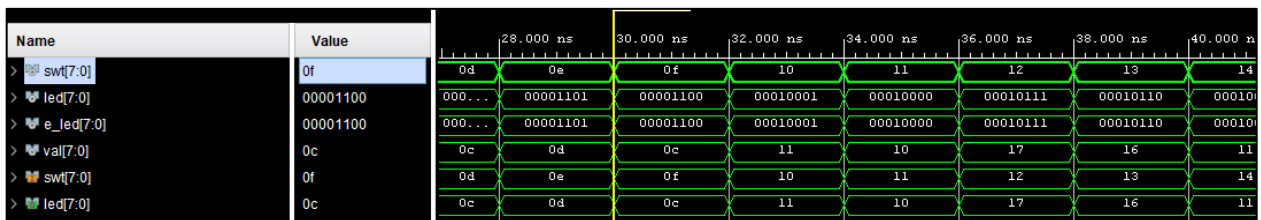


Figure 19. Snapshot of Running simulation for additional 500 ns

Observe the Tcl Console window and see the output is being displayed as the testbench uses the “report” task.

```
Time: 691 ns Iteration: 0 Process: /lab_intro_tb/line_62 File: C:/xup/fpga_flow/Zynq_Labs/lab_intro/Lab_intro.srcs/sim_1/imports/lab_intro/lab_intro_tb.vhd
Note: LED output matched at 693000 ps
Time: 693 ns Iteration: 0 Process: /lab_intro_tb/line_62 File: C:/xup/fpga_flow/Zynq_Labs/lab_intro/Lab_intro.srcs/sim_1/imports/lab_intro/lab_intro_tb.vhd
Note: LED output matched at 695000 ps
Time: 695 ns Iteration: 0 Process: /lab_intro_tb/line_62 File: C:/xup/fpga_flow/Zynq_Labs/lab_intro/Lab_intro.srcs/sim_1/imports/lab_intro/lab_intro_tb.vhd
Note: LED output matched at 697000 ps
Time: 697 ns Iteration: 0 Process: /lab_intro_tb/line_62 File: C:/xup/fpga_flow/Zynq_Labs/lab_intro/Lab_intro.srcs/sim_1/imports/lab_intro/lab_intro_tb.vhd
Note: LED output matched at 699000 ps
Time: 699 ns Iteration: 0 Process: /lab_intro_tb/line_62 File: C:/xup/fpga_flow/Zynq_Labs/lab_intro/Lab_intro.srcs/sim_1/imports/lab_intro/lab_intro_tb.vhd
```

Figure 20. 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 **Discard** 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* tasks of the *Flow Navigator* pane.

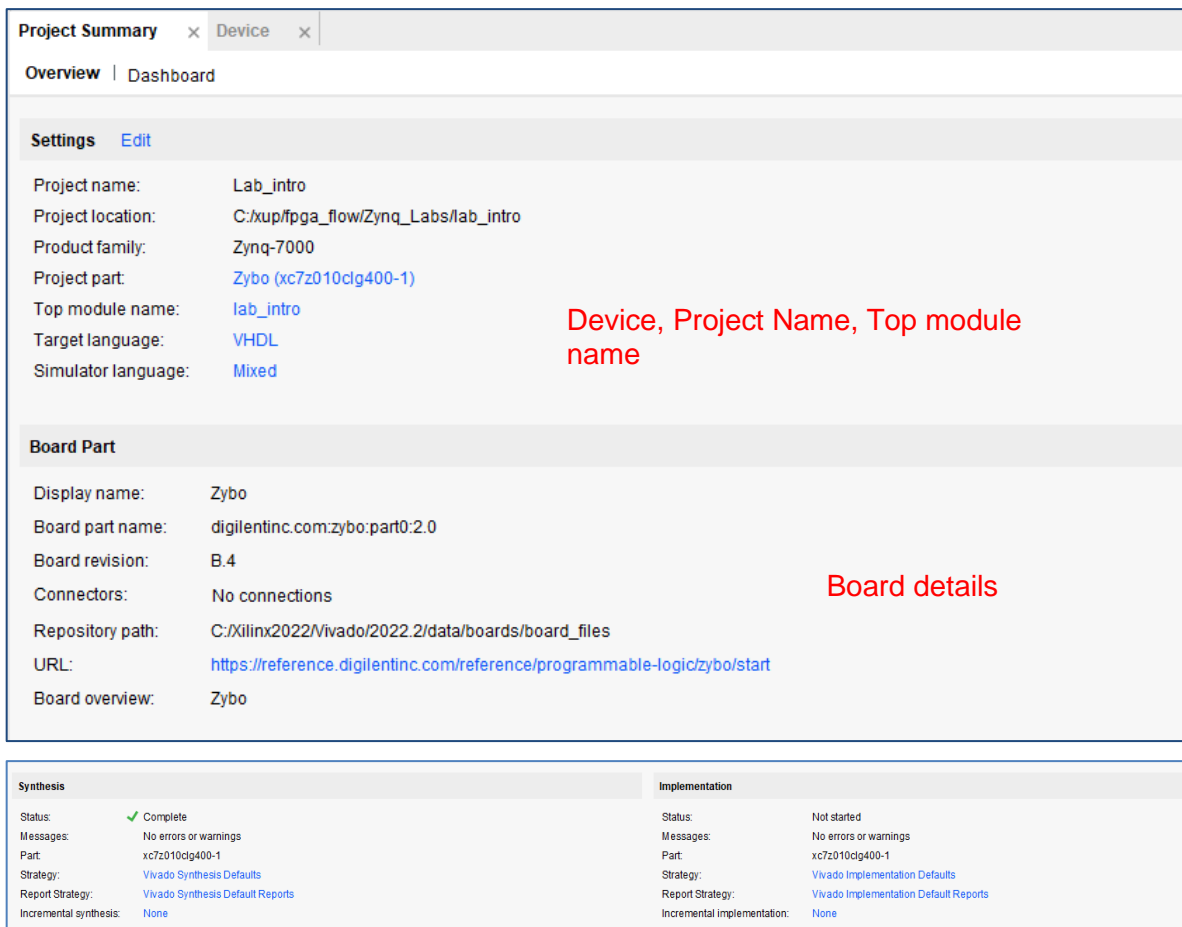
The synthesis process will be run on the lab_intro.vhd file (and all its hierarchical files if they exist). When the process is completed, a *Synthesis Completed* dialog box with three options will be displayed.

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 .



The screenshot shows the Vivado Project Summary window. It has tabs for 'Project Summary' (selected), 'Device', and 'Implementation'. The 'Project Summary' tab is divided into 'Overview' and 'Dashboard' sections. The 'Overview' section contains 'Settings' and 'Board Part' subsections.

Settings:

- Project name: Lab_intro
- Project location: C:/xup/fpga_flow/Zynq_Labs/lab_intro
- Product family: Zynq-7000
- Project part: Zybo (xc7z010clg400-1)
- Top module name: lab_intro
- Target language: VHDL
- Simulator language: Mixed

Board Part:

- Display name: Zybo
- Board part name: digilentinc.com:zybo:part0:2.0
- Board revision: B.4
- Connectors: No connections
- Repository path: C:/Xilinx2022/Vivado/2022.2/data/boards/board_files
- URL: <https://reference.digilentinc.com/reference/programmable-logic/zybo/start>
- Board overview: Zybo

Synthesis and Implementation Status:

Synthesis	Implementation
Status: ✔ Complete	Status: Not started
Messages: No errors or warnings	Messages: No errors or warnings
Part: xc7z010clg400-1	Part: xc7z010clg400-1
Strategy: Vivado Synthesis Defaults	Strategy: Vivado Implementation Defaults
Report Strategy: Vivado Synthesis Default Reports	Report Strategy: Vivado Implementation Default Reports
Incremental synthesis: None	Incremental implementation: None

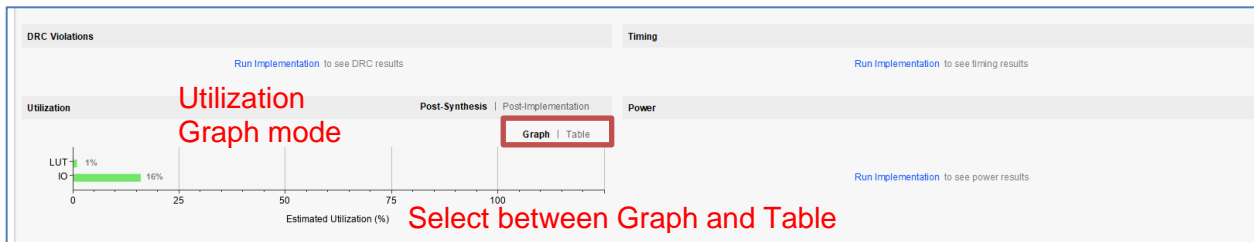


Figure 21. Project Summary view

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.

Utilization		Post-Synthesis Post-Implementation	
		Graph Table	
Resource	Estimation	Available	Utilization %
LUT	3	17600	0.02
IO	16	100	16.00

Figure 22. 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.

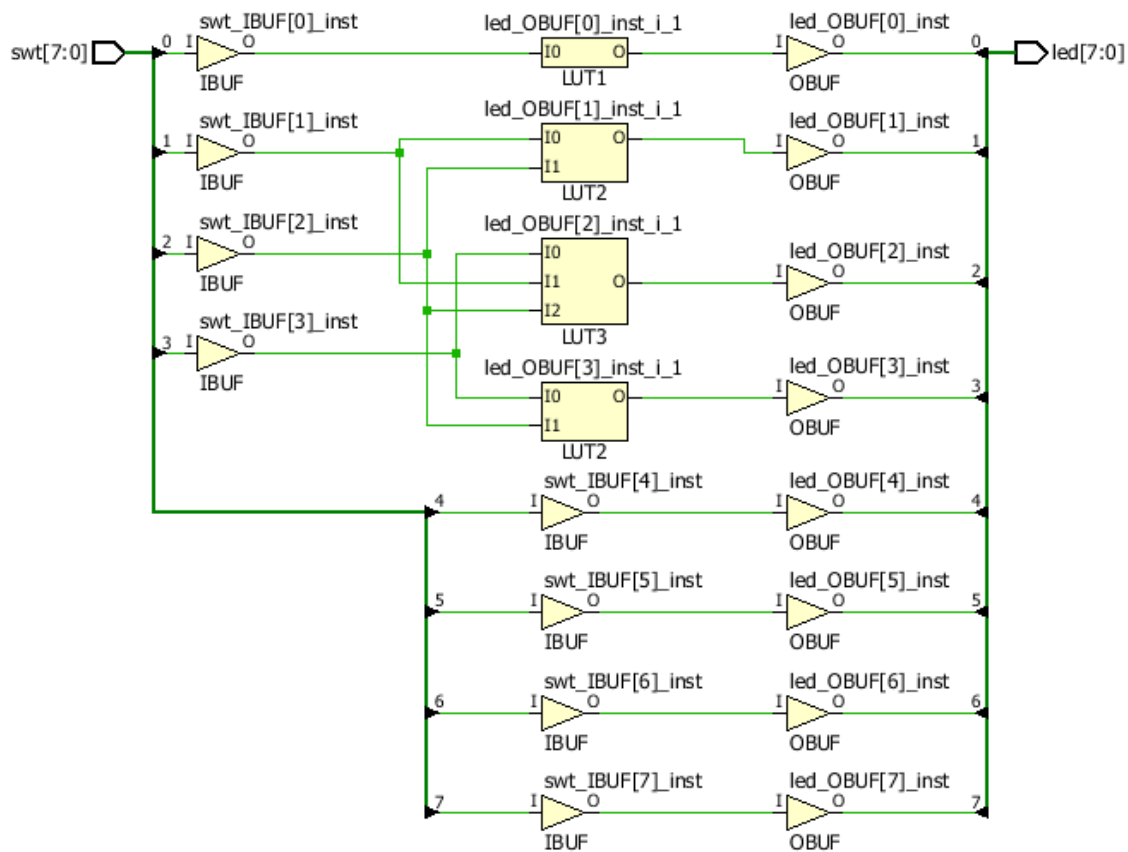


Figure 23. Synthesized design's schematic view

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). Four gates in RTL analysis output are mapped onto four LUTs in the synthesized output.

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

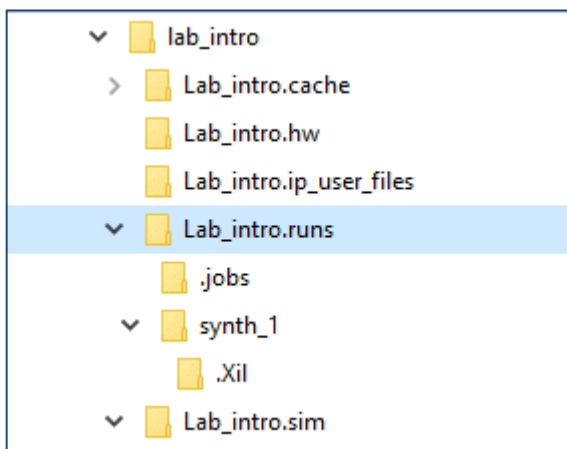


Figure 24. Directory structure after synthesizing the design

Implement the Design

Step 4

4-1. Implement the design with the Vivado Implementation Defaults (Vivado Implementation 2021) 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 (e.g. led_OBUF[1]) and notice that the net displayed in the X1Y1 (Zybo) clock region in the Device view tab (you may have to zoom in to see it).

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

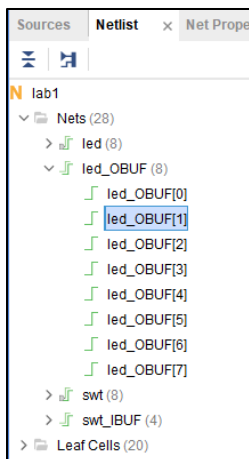


Figure 25. Selecting a net

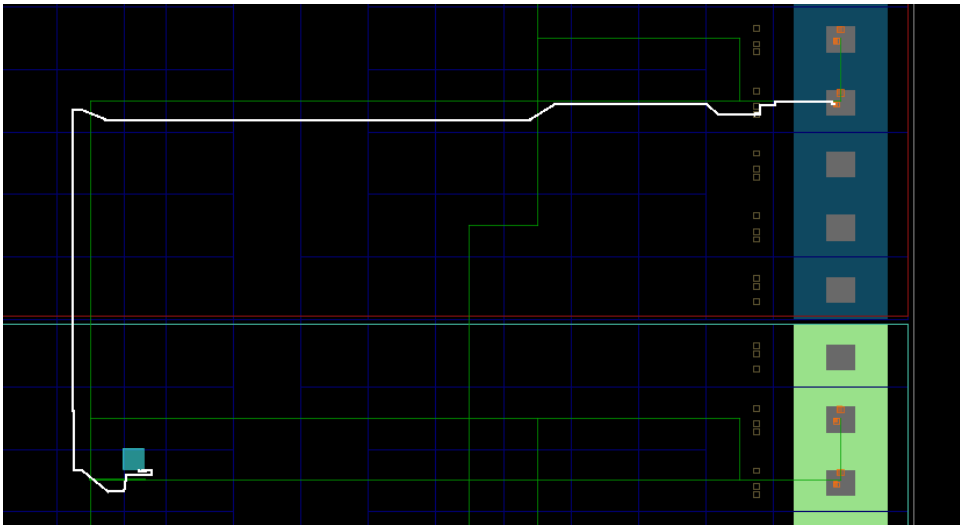


Figure 26. Viewing implemented design for the Zybo

- 4-1-6. Close the implemented design view by selecting **File > Close Implemented Design**, 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.

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

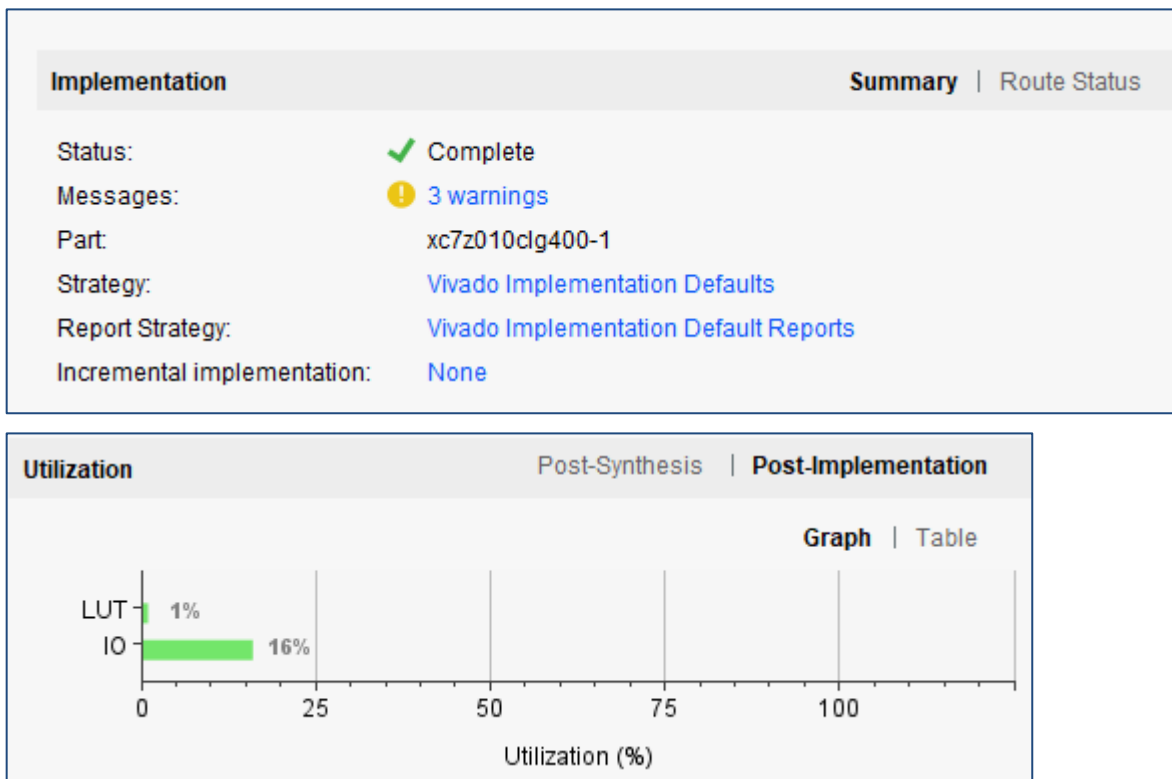


Figure 27. Implementation results for the Zybo

Using the Windows Explorer, verify that **impl_1** directory is created at the same level as **synth_1** under the **lab_intro.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.

Tcl Console Messages Log Reports x Design Runs					
Report	Type	Options	Modified	Size	
▼ Synthesis					
▼ Synth Design (synth_design)					
Utilization - Synth Design	report_utilization		1/21/22, 6:33 PM	7.3 KB	
synthesis_report			1/21/22, 6:33 PM	14.4 KB	
▼ Implementation					
▼ impl_1					
▼ Design Initialization (init_design)					
Timing Summary - Design Initialization	report_timing_summary	max_paths = 10;			
▼ Opt Design (opt_design)					
DRC - Opt Design	report_drc		1/21/22, 6:42 PM	1.4 KB	
Timing Summary - Opt Design	report_timing_summary	max_paths = 10;			
▼ Power Opt Design (power_opt_design)					
Timing Summary - Power Opt Design	report_timing_summary	max_paths = 10;			
▼ Place Design (place_design)					
IO - Place Design	report_io		1/21/22, 6:42 PM	117.6 KB	
Utilization - Place Design	report_utilization		1/21/22, 6:42 PM	9.2 KB	
Control Sets - Place Design	report_control_sets	verbose = true;	1/21/22, 6:42 PM	3.4 KB	
Incremental Reuse - Place Design	report_incremental_reuse				
Incremental Reuse - Place Design	report_incremental_reuse				
Timing Summary - Place Design	report_timing_summary	max_paths = 10;			
> Post-Place Power Opt Design (post_place_power_opt_design)					
▼ Post-Place Phys Opt Design (phys_opt_design)					
Timing Summary - Post-Place Phys Opt Design	report_timing_summary	max_paths = 10;			
▼ Route Design (route_design)					
DRC - Route Design	report_drc		1/21/22, 6:42 PM	1.5 KB	
Methodology - Route Design	report_methodology		1/21/22, 6:42 PM	1.3 KB	
Power - Route Design	report_power		1/21/22, 6:42 PM	8.7 KB	
Route Status - Route Design	report_route_status		1/21/22, 6:42 PM	0.6 KB	
Timing Summary - Route Design	report_timing_summary	max_paths = 10;	1/21/22, 6:42 PM	7.9 KB	
Incremental Reuse - Route Design	report_incremental_reuse				
Clock Utilization - Route Design	report_clock_utilization		1/21/22, 6:42 PM	5.8 KB	
Bus Skew - Route Design	report_bus_skew	warn_on_violation = true;	1/21/22, 6:42 PM	0.8 KB	
implementation_log			1/21/22, 6:42 PM	25.5 KB	
> Post-Route Phys Opt Design (post_route_phys_opt_design)					
> Write Bitstream (write_bitstream)					

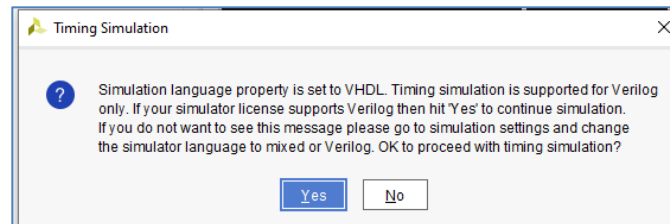
Figure 28. Available reports to view

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.
- 5-1-2.** You will see a message for Timing simulation only supported for Verilog (if mixed-mode selection was not selected in the initial simulator language options). Click **Yes** to continue.



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

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

- 5-1-3.** Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.
- 5-1-4.** Right-click at 8 ns (where the switch input is set to 00000100b) and select **Markers > Add Marker**.
- 5-1-5.** Similarly, right-click and add a marker at around 8.500 ns where the **led** changes.
- 5-1-6.** 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.

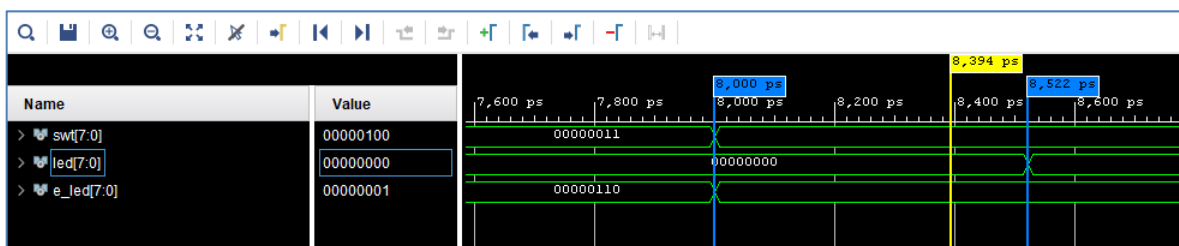


Figure 29. Timing simulation output

Notice that we monitored the expected led output at 1 ns after the input is changed (see the testbench) whereas the actual delay is about 500 ps (depending on the board). Also, check the TCL console where some Errors may show up at the beginning of the simulation (signals are propagating/stabilizing with timing – Red/undefined waveforms) but outputs will start to match eventually.

- 5-1-7.** 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 session, 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 switch for the Zybo.
- 6-1-2. The Zybo is powered through USB power via the JTAG PROG.

Make sure that the board is set to use USB power for the Zybo (via the Power Select jumper JP7)



Figure 30. Board connection for the Zybo

- 6-1-3. Power **ON** the board.
- 6-1-4. 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 three options will be displayed.

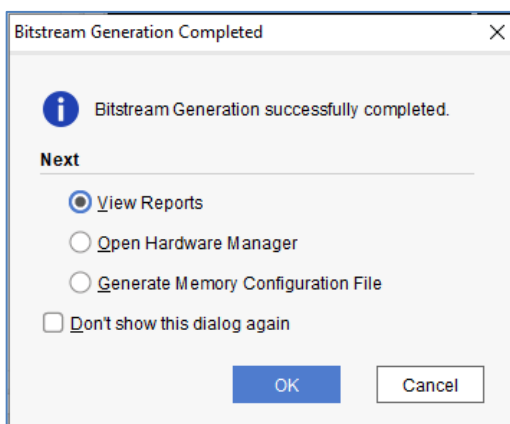


Figure 31. Bitstream generation

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

- 6-1-5.** Select the *Open Hardware Manager* option and click **OK**.

The Hardware Manager window will open indicating “unconnected” status.

- 6-1-6.** Click on the **Open target** link.

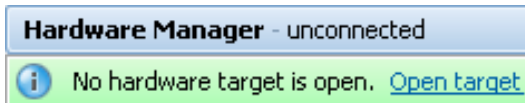


Figure 32. Opening new hardware target

- 6-1-7.** From the dropdown menu, click **Auto Connect**.

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

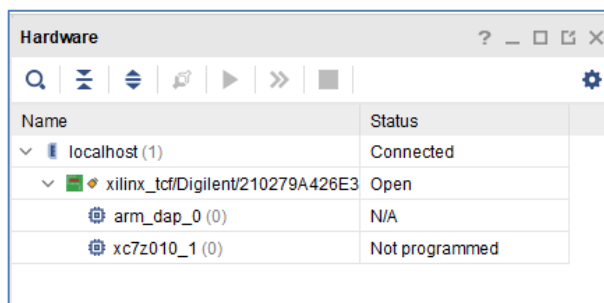


Figure 33. Opened hardware session for the Zybo

Select the device, and verify that the lab_intro.bit is selected as the programming file in the General tab below.

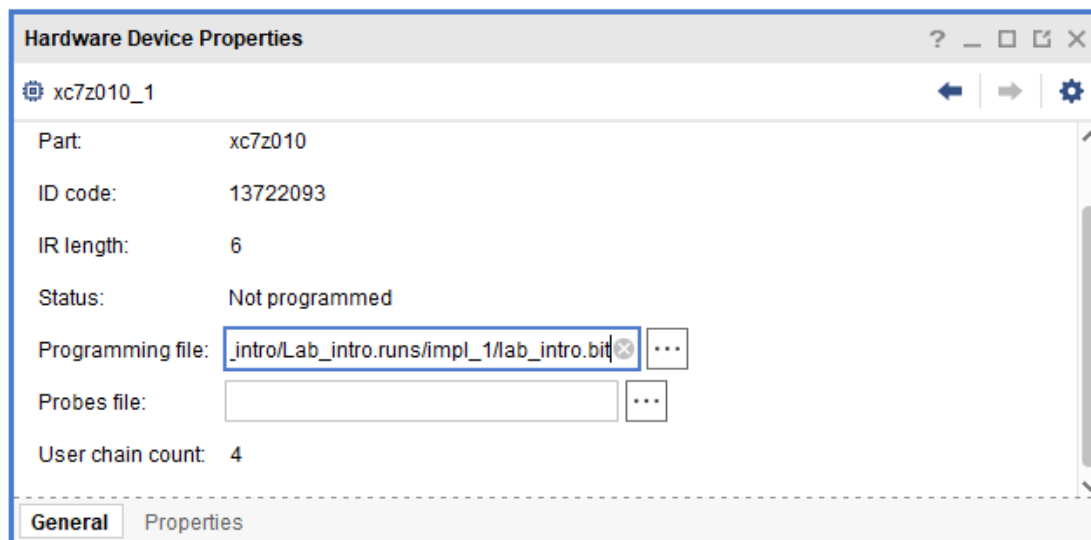


Figure 34. Programming file

- 6-1-8.** Click on the *Program device > XC7Z010_1* link in the green information bar to program the target FPGA device.

Another way is to right click on the device and select *Program Device...*

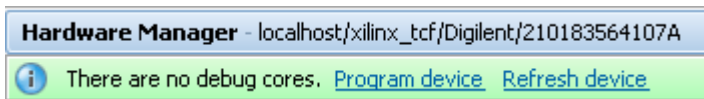


Figure 35. Selecting to program the FPGA

- 6-1-9.** Click **Program** to program the FPGA.

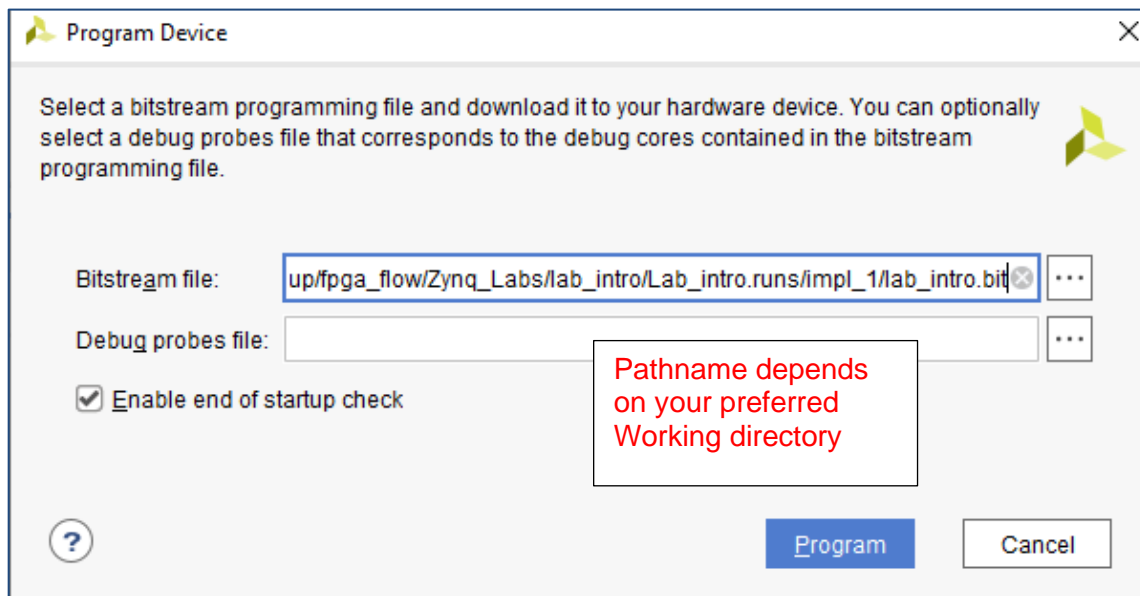


Figure 36. Program Device window

The DONE light will light when the device is programmed. You may see some other LEDs lit depending on switch positions.

- 6-1-10.** Verify the functionality by flipping switches and observing the output on the LEDs (Refer to the earlier logic diagram).
- 6-1-11.** When satisfied, power **OFF** the board.
- 6-1-12.** Close the hardware session by selecting **File > Close Hardware Manager**.
- 6-1-13.** Click **OK** to close the session.
- 6-1-14.** Close the **Vivado** program by selecting **File > Exit** and click **OK**.

Questions

1. Is there a combination of switches where you can light up all four output LEDs?
2. What combination of switches lights up LEDs: LD3, LD2 and LD0 while LD1 is off?
3. Which Xilinx part did you select as the target device?

Conclusion

The Vivado software tool can be used to perform a complete HDL based 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.

Reference

Xilinx – www.xilinx.com/support/university - Copyright 2016