

## Laboratory Assignment 1 - Vivado Design Flow

### Introduction

This lab guides you through the process of using Vivado IDE to create a simple HDL design targeting the Basys3 boards. 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:

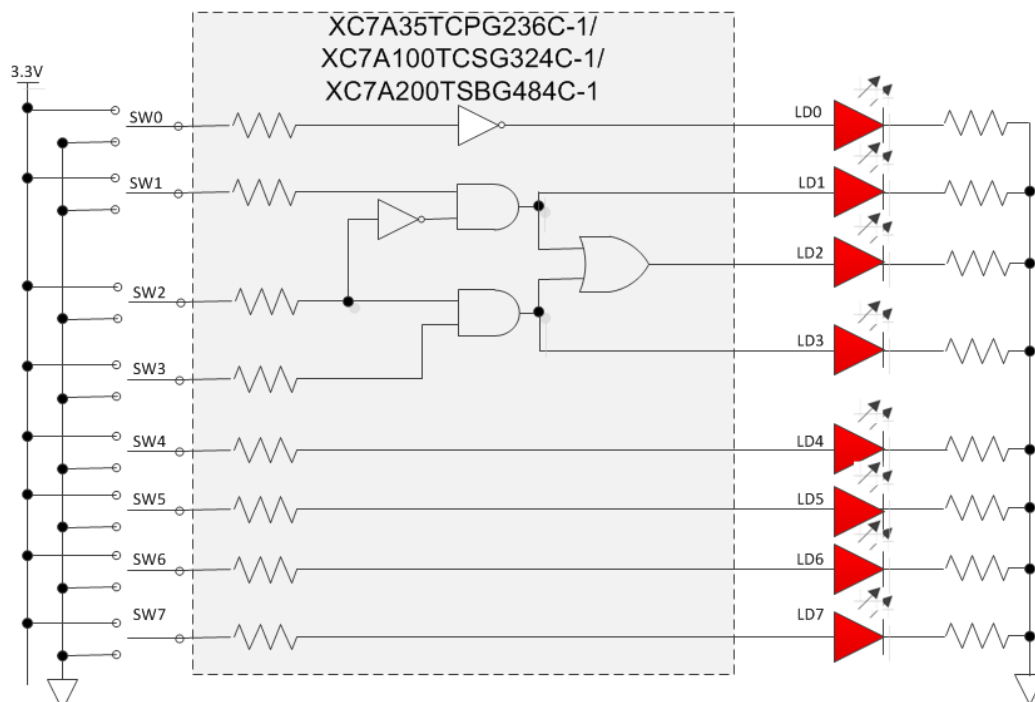
- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device
- Use the provided Xilinx Design Constraint (XDC) file to constrain the pin locations
- Simulate the design using the Vivado simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA 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 lab.

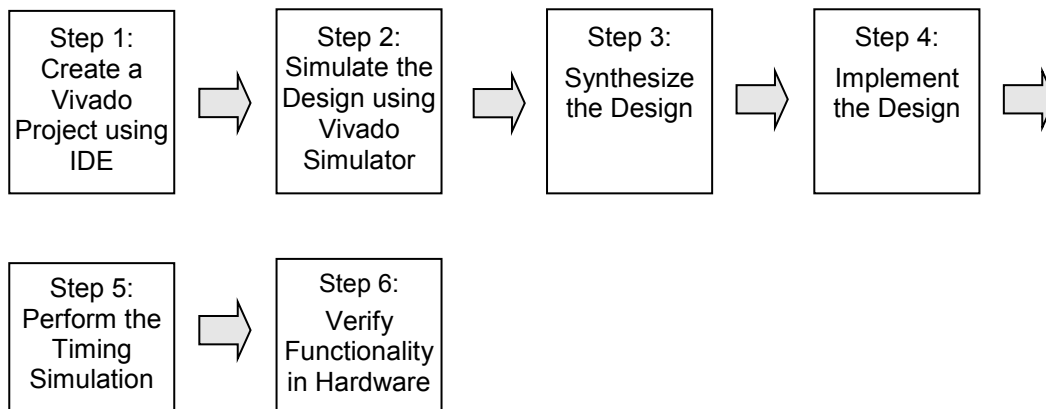
### Design Description

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



In the instructions below;

{**sources**} refers to: C:\Users\MEE\_CE\sources

{**labs**} refers to : C:\ Users\MEE\_CE\labs

Board support for the Basys3 is not included in Vivado 2016.2 by default. The relevant zip files need to be extracted and saved to: {Vivado installation}\data\boards\board\_files\.

These files can be downloaded either from the Digilent, Inc. webpage

(<https://reference.digilentinc.com/vivado/boardfiles2015>) or the XUP webpage

(<http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-fpga-design-flow.html> )

where this material is also hosted.

## Create a Vivado Project using IDE

### Step 1

#### 1-1. Create a New Project

- 1-1-1. Start downloading the source files provided (Moodle) and save them in {sources}/lab1
- 1-1-2. Open Vivado: **Start > All Programs > Xilinx Design Tools > Vivado 2016.2 > Vivado 2016.2**
- 1-1-3. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
- 1-1-4. Click the Browse button of the *Project location* field of the **New Project** form, browse to {c:/Users/MEE\_EC/labs}, and click **Select**.
- 1-1-5. Enter **lab1** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

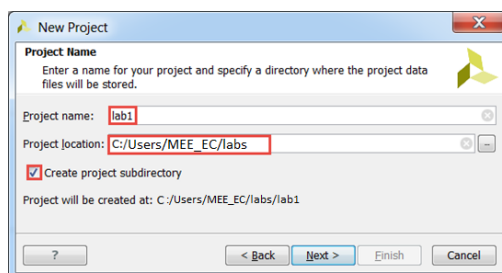
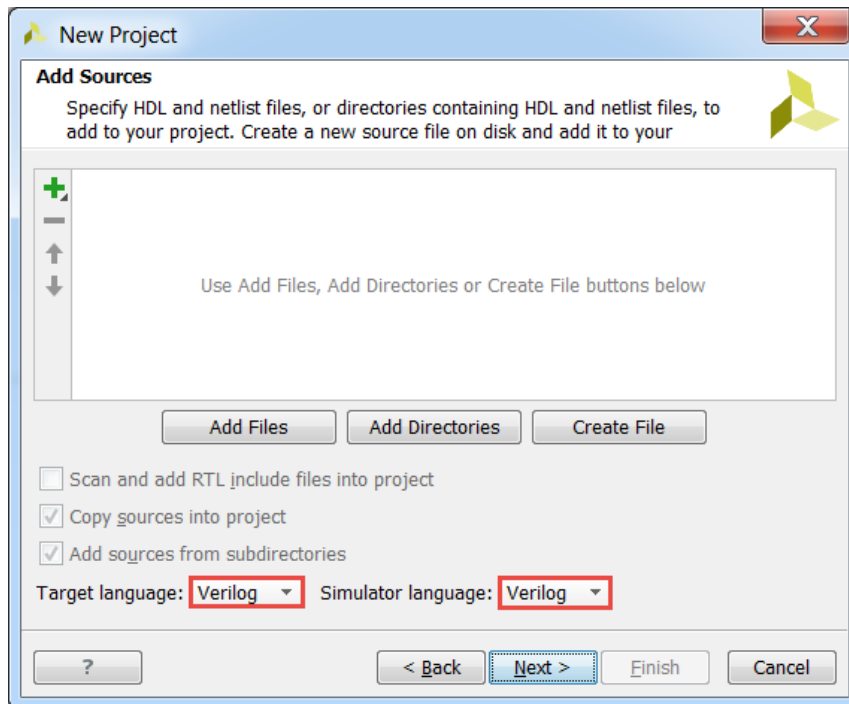


Figure 2. Project Name and Location entry

- 1-1-6.** Select **RTL Project** option in the *Project Type* form, and click **Next**.
- 1-1-7.** Using the drop-down buttons, select **Verilog** as the *Target Language* and *Simulator Language* in the *Add Sources* form.



**Figure 3. Selecting Target and Simulator language**

- 1-1-8.** Click on the **Green Plus** button, then **Add Files...** and browse to the **{sources}\lab1** directory, select *lab1.v*, 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.
- 1-1-9.** Since we do not have any IP to add, click **Next** to get to the *Add Constraints* form.
- 1-1-10.** Click on the **Green Plus** button, then **Add Files...** and browse to the **{sources}\lab1** directory (if necessary), select *lab1\_basys3.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.
- 1-1-11.** In the *Default Part* form, use the **Parts** option and various drop-down fields of the **Filter** section. Select the **XC7A35TCPG236-1** for *Basys3* board.

You may also select the Boards option, select digilentinc.com under the Vendor filter and select the appropriate board. Notice that Basys3 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 or from the XUP website's workshop material pages and install them in the Vivado installation. You must restart Vivado to see the boards.

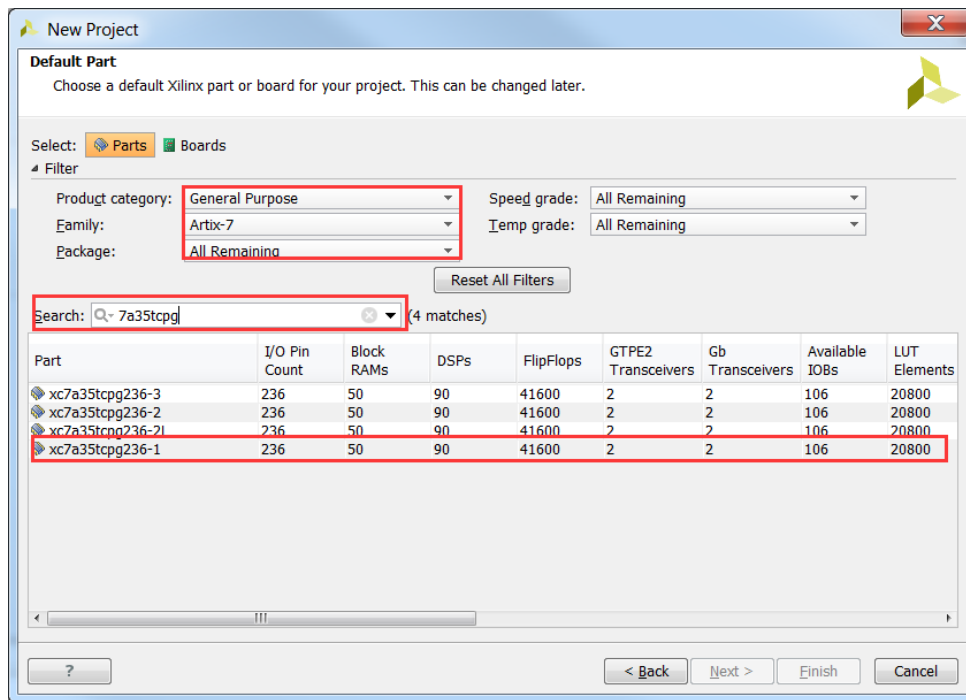


Figure 4. Part Selection for the Basys3

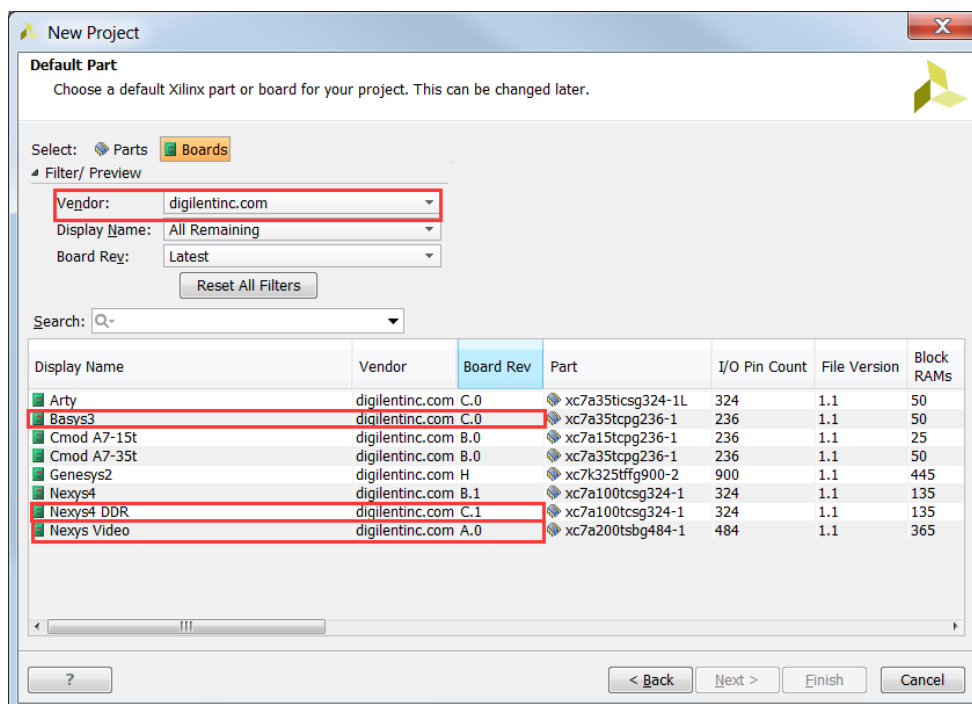


Figure 5. Selecting the target board

**1-1-12.** Click **Next**.

**1-1-13.** Click **Finish** to create the Vivado project.

Use the Windows Explorer and look at the {labs}\lab1 directory. You will find that the lab1.cache and lab1.srds 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.srds directory; deep down under them, the copied lab1\_<board>.xdc (constraint) and lab1.v (source) files respectively are placed.

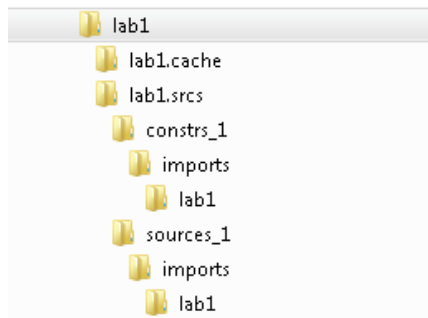


Figure 6. Generated directory structure

## 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.

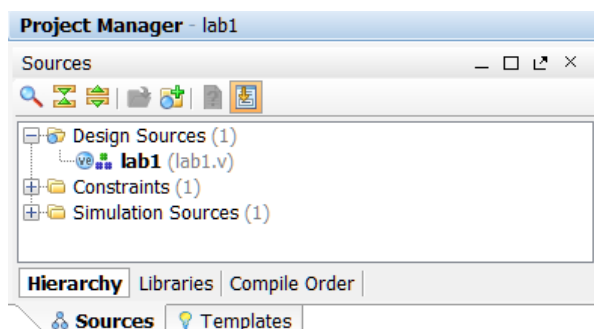


Figure 7. Opening the source file

- 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.
- 1-2-3. Line 7 defines the beginning (marked with keyword **module**) and Line 19 defines the end of the module (marked with keyword **endmodule**).
- 1-2-4. Lines 8-9 defines the input and output ports whereas lines 12-17 defines the actual functionality.

## 1-3. Open the lab1\_basys3.xdc source and analyze the content.

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

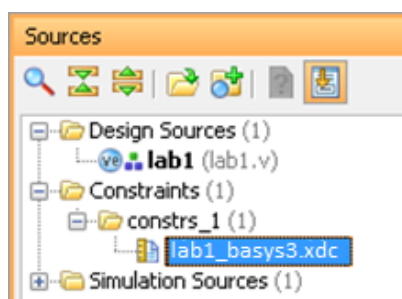
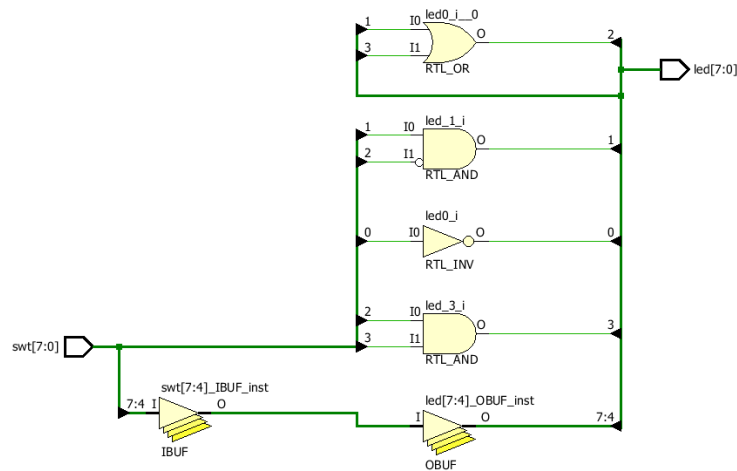


Figure 8. Opening the constraint file

- 1-3-2. For the lines 5-12 defines the pin locations of the input switches [7:0] and lines 17-24 defines the pin locations of the output LEDs [7:0].
- 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 9. A logic view of the design**

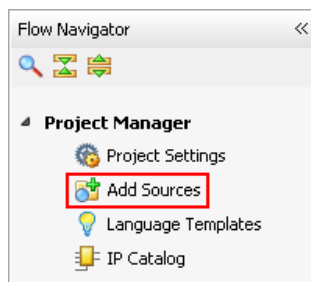
Notice that some of the switch inputs go through gates before being output to LEDs through output buffers and the rest go through input buffers and output buffers to LEDs as modeled in the file.

## Simulate the Design using the Vivado Simulator

## Step 2

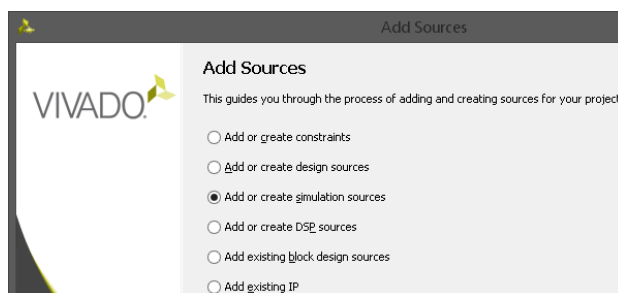
- 2-1. Add the lab1\_tb.v testbench file.**

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



**Figure 10. Add Sources**

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



**Figure 11. Selecting Simulation Sources option**

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

- 2-1-4. Browse to the {sources}\lab1 folder and select *lab1\_tb.v* and click **OK**.
- 2-1-5. 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 **dut** (device under test) instance.

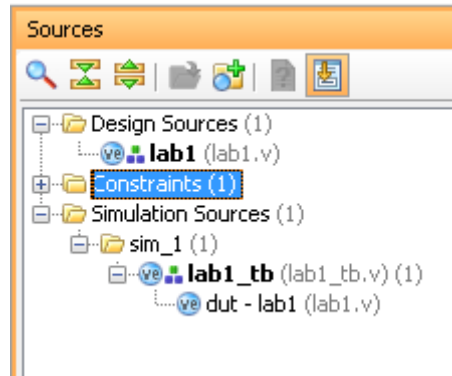


Figure 12. Simulation Sources hierarchy

- 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    end
41 endmodule
42

```

Figure 13. The self-checking testbench

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

## 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**.

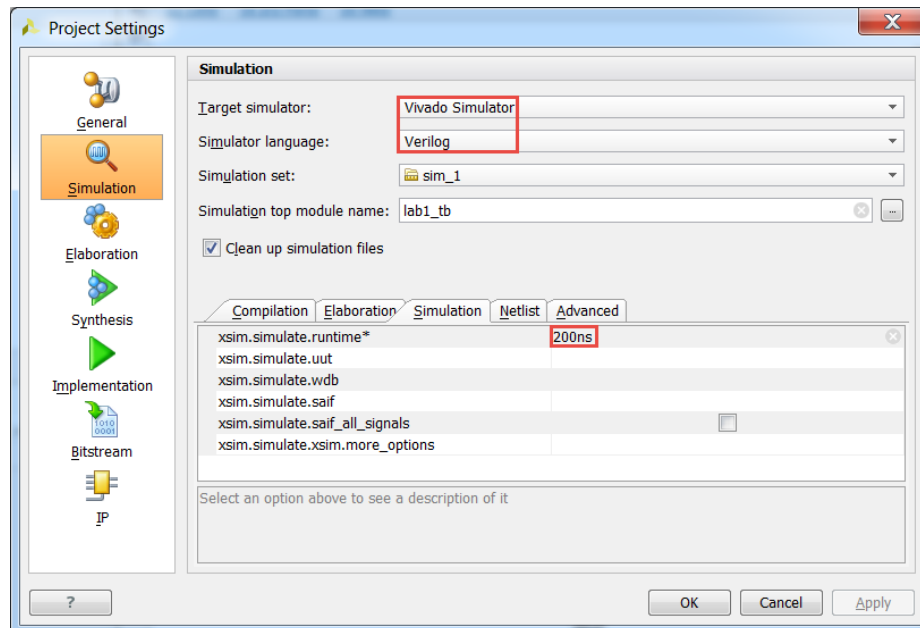



Figure 14. Setting simulation run time

### 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. Click on the Zoom Fit (  ) button and you will see output similar to the one shown below.

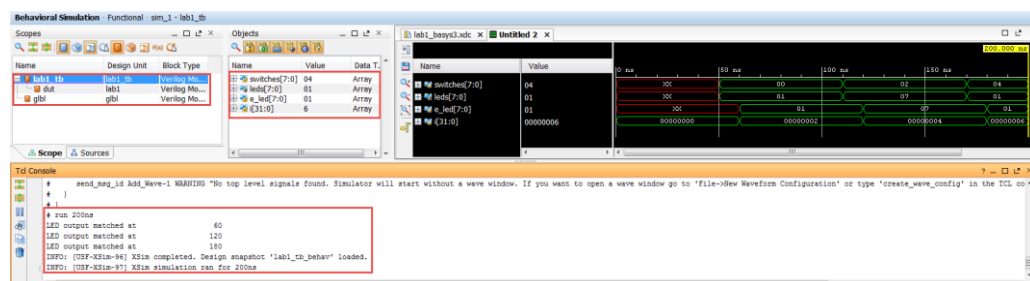
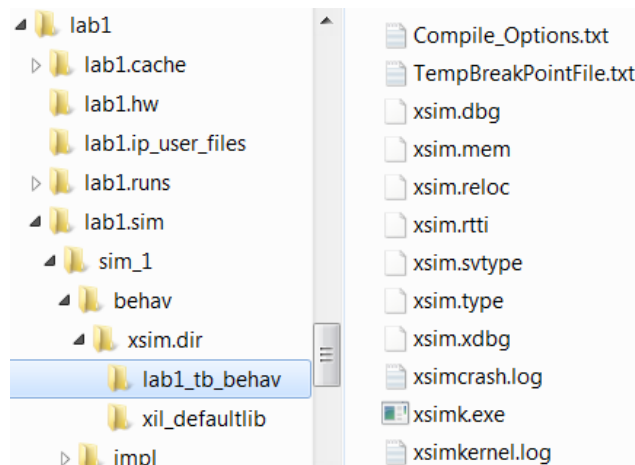


Figure 15. Simulator output

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



**Figure 16. 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.

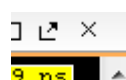
**Table 1: Various buttons available to view the waveform**

	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 17. Float Button**



**Figure 18. Dock Window Button**

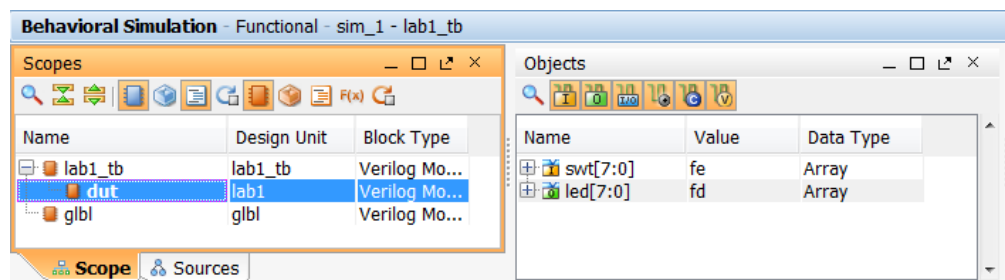
**2-3. Change display format if desired.**

- 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]** to *Hexadecimal*. Leave the **leds[7:0]** and **e\_led[7:0]** 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 *Scopes* window and select the **dut** instance.

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

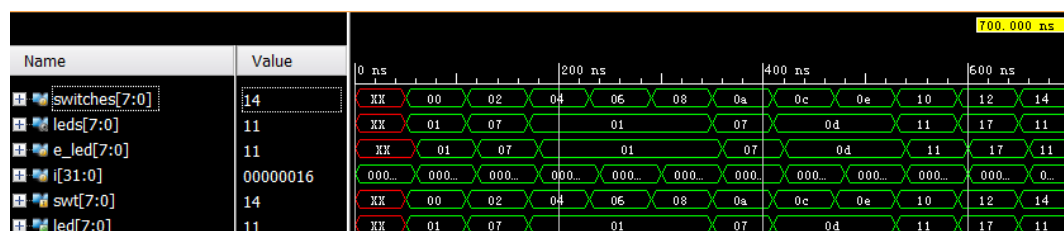


**Figure 19. Selecting lower-level signals**

- 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 ( 500 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 20. 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.

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'lab1_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 200ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:08 . Memory (MB): peak = 1159.863 ;
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 21. 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 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.

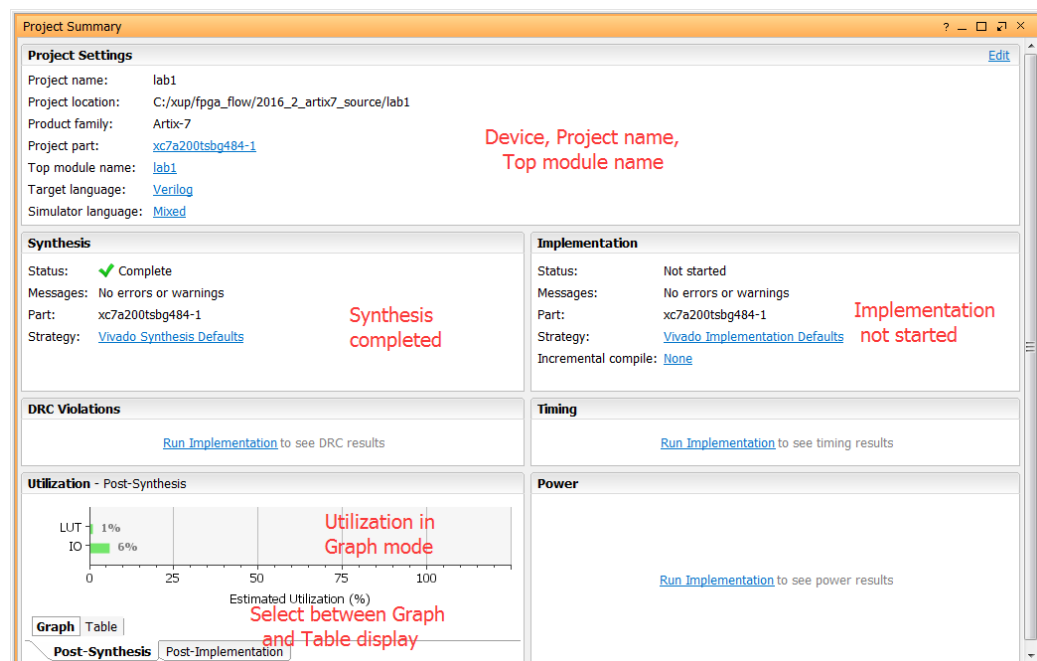
**3-1-2.** Select the **Open Synthesized Design** option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage.

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 22. 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.

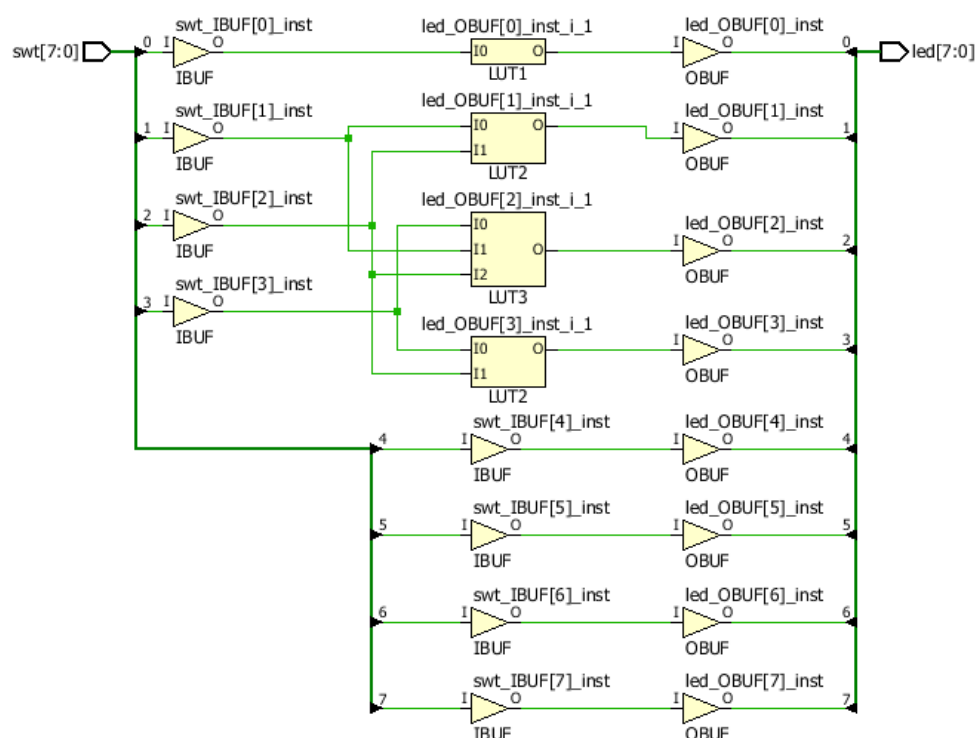
**Utilization - Post-Synthesis**

Resource	Estimation	Available	Utilization %
LUT	3	20800	0.01
I/O	16	106	15.09

Graph Table

**Figure 23. Resource utilization estimation summary for the Basys3**

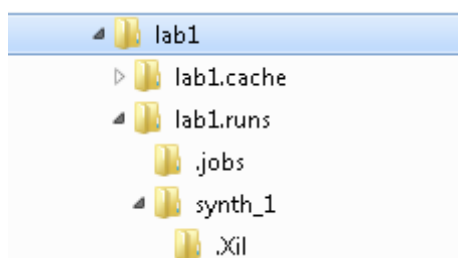
- 3-1-5.** In The *Flow Navigator*, under *Synthesis* (expand *Synthesized Design* if necessary), click on **Schematic** to view the synthesized design in a schematic view.



**Figure 24. 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 **lab1.runs** directory is created under **lab1**. Under the **runs** directory, **synth\_1** directory is created which holds several files related to synthesis.



**Figure 25. Directory structure after synthesizing the design**

## Implement the Design

## Step 4

### 4-1. Implement the design with the Vivado Implementation Defaults 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 X0Y0 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.

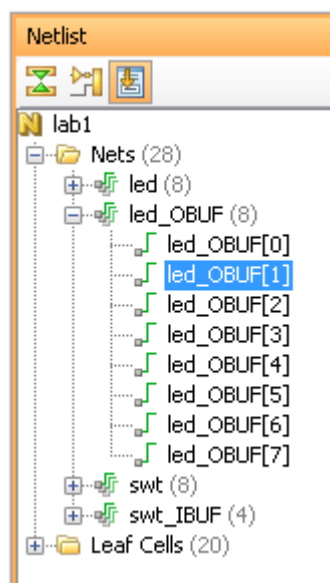


Figure 26. Selecting a net

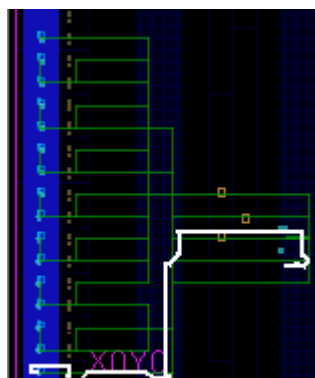
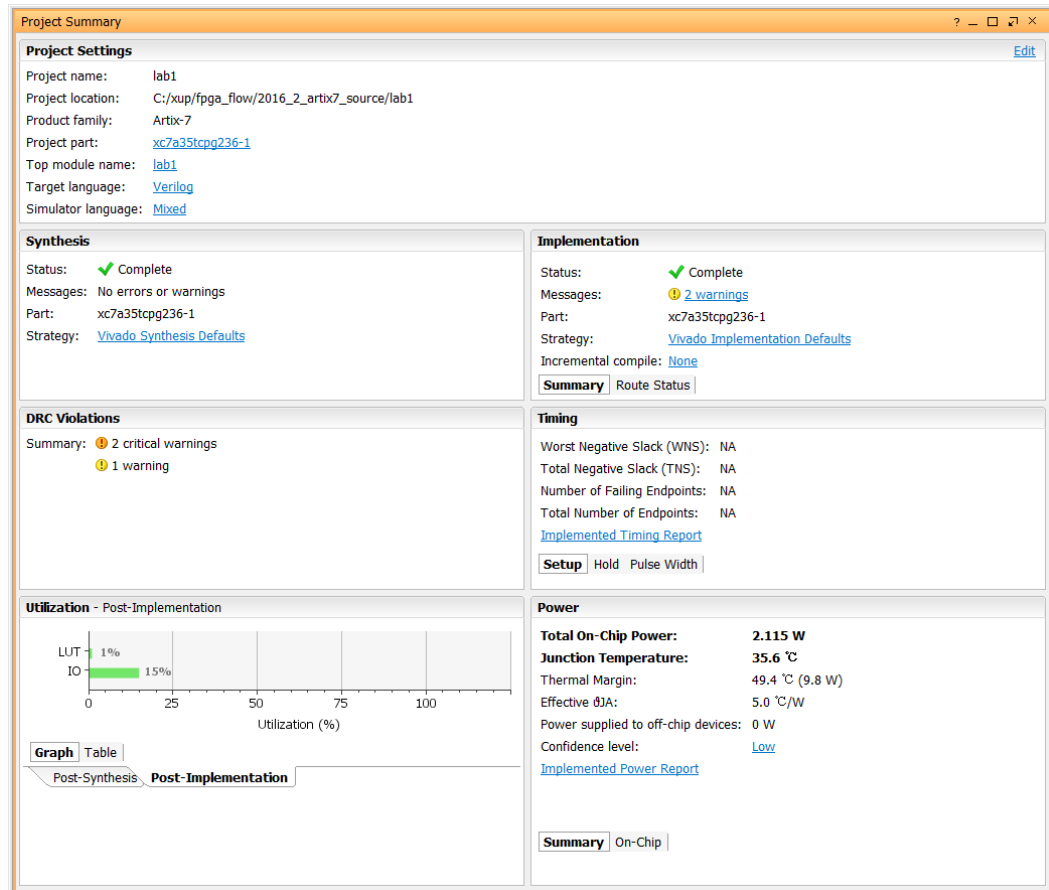


Figure 27. Viewing implemented design for the Basys3

- 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 28. Implementation results for the Basys3**

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.

Name	Modified	Size	GUI Report
[-] Synth Design (synth_design)			
[-] Vivado Synthesis Report	6/5/16 10:22 PM	14.2 KB	
[-] Utilization Report	6/5/16 10:22 PM	6.7 KB	
[-] Design Initialization (init_design)			
[-] Timing Summary Report			
[-] Opt Design (opt_design)			
[-] Post opt_design DRC Report	6/17/16 4:46 PM	3.6 KB	
[-] Post opt_design Methodolo...			
[-] Timing Summary Report			
[-] Power Opt Design (power_opt_design)			
[-] Timing Summary Report			
[-] Place Design (place_design)			
[-] Vivado Implementation Log	6/17/16 4:46 PM	16.3 KB	
[-] Pre-Placement Incremental...			
[-] IO Report	6/17/16 4:46 PM	60.1 KB	
[-] Utilization Report	6/17/16 4:46 PM	7.8 KB	
[-] Control Sets Report	6/17/16 4:46 PM	2.5 KB	
[-] Incremental Reuse Report			
[-] Timing Summary Report			
[-] Post-Place Power Opt Design (post_place_power_opt_design)			
[-] Timing Summary Report			
[-] Post-Place Phys Opt Design (phys_opt_design)			
[-] Timing Summary Report			
[-] Route Design (route_design)			
[-] Vivado Implementation Log	6/17/16 4:46 PM	16.3 KB	
[-] WebTalk Report			
[-] DRC Report	6/17/16 4:46 PM	3.6 KB	
[-] Methodology DRC Report			
[-] Power Report	6/17/16 4:46 PM	6.8 KB	
[-] Route Status Report	6/17/16 4:46 PM	0.6 KB	
[-] Timing Summary Report	6/17/16 4:46 PM	7.1 KB	Open
[-] Incremental Reuse Report			
[-] Clock Utilization Report	6/17/16 4:46 PM	7.3 KB	
[-] Post-Route Phys Opt Design (post_route_phys_opt_design)			
[-] Post-Route Physical Optimi...			
[-] Write Bitstream (write_bitstream)			
[-] Vivado Implementation Log			
[-] WebTalk Report			

Figure 29. 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. Right-click at 50 ns (where the switch input is set to 0000000b) and select **Markers > Add Marker**.
- 5-1-4. Similarly, right-click and add a marker at around 59.260 ns where the **leds** changes.
- 5-1-5. 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.

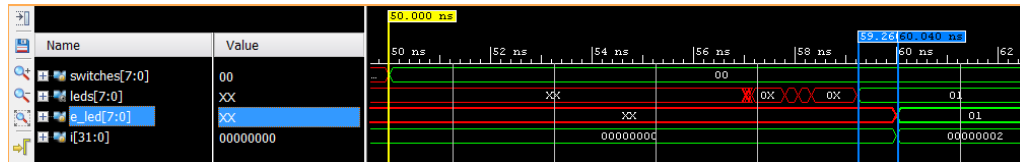


Figure 30. Timing simulation output

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 to 5.9 ns (depending on the board).

- 5-1-6. 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 connector).
  - 6-1-2. Make sure that the board is set to use USB power (via the Power Select jumper JP2)



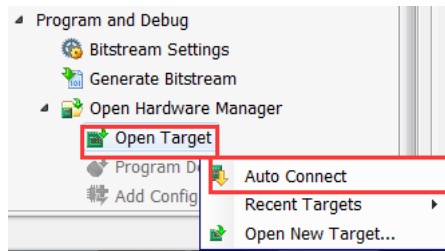
Figure 31. Board connection for the Basys3

- 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. This process will have generated a **lab1.bit** file under **impl\_1** directory in the **lab1.runs** directory.

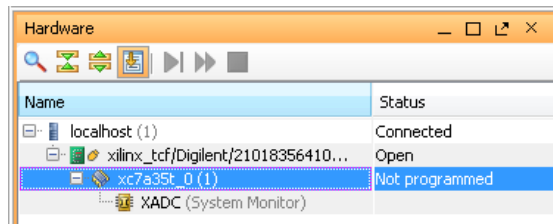
- 6-1-5. Expand the *Open Hardware Manager* option and click *Open Target* option.
- 6-1-6. Click on the *Select Auto Connect* option.





**Figure 32. Selecting *Auto Connect***

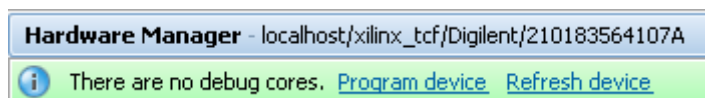
- 6-1-7.** The Hardware Manager Window will open automatically. You can find your device listed in the *Hardware Window*. Also notice that the Status indicates that it is not programmed.



**Figure 33. Opened hardware session for the Basys3**

- 6-1-8.** Select the device and verify that the lab1.bit is selected as the programming file.
- 6-1-9.** Click on the *Program device* > XC7A100T\_0 or the XC7A35T\_0 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 34. Selecting to program the FPGA**

- 6-1-10.** Click **Program** to program the FPGA.

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

- 6-1-11.** Verify the functionality by flipping switches and observing the output on the LEDs (Refer to the earlier logic diagram).
- 6-1-12.** When satisfied, power **OFF** the board.
- 6-1-13.** Close the hardware session by selecting **File > Close Hardware Manager**.
- 6-1-14.** Click **OK** to close the session.
- 6-1-15.** Close the **Vivado** program by selecting **File > Exit** and click **OK**.

## 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.