

Vivado Tutorial

Introduction

This tutorial guides you through the design flow using Xilinx Vivado software to create a simple digital circuit using Verilog HDL. A typical design flow consists of creating model(s), creating user constraint file(s), creating a Vivado project, importing the created models, assigning created constraint file(s), optionally running behavioral simulation, synthesizing the design, implementing the design, generating the bitstream, and finally verifying the functionality in the hardware by downloading the generated bitstream file. You will go through the typical design flow targeting the Artix-100 based Nexys4 board. The typical design flow is shown below. The circled number indicates the corresponding step in this tutorial.

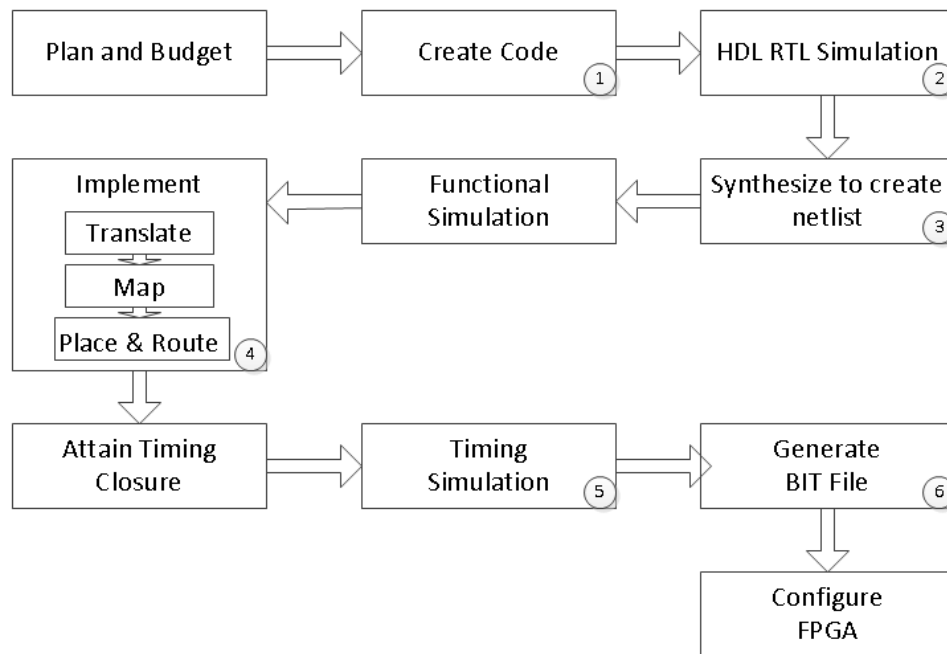


Figure 1. A typical design flow

Objectives

After completing this tutorial, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the Nexys4 board
- Use the provided partially completed Xilinx Design Constraint (XDC) file to constrain some of the pin locations
- Add additional constraints using the Tcl scripting feature of Vivado
- Simulate the design using the XSim simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA using the generated bitstream and verify the functionality
- Go through the design flow in batch mode using the Tcl script

Procedure

This tutorial 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 tutorial.

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**.

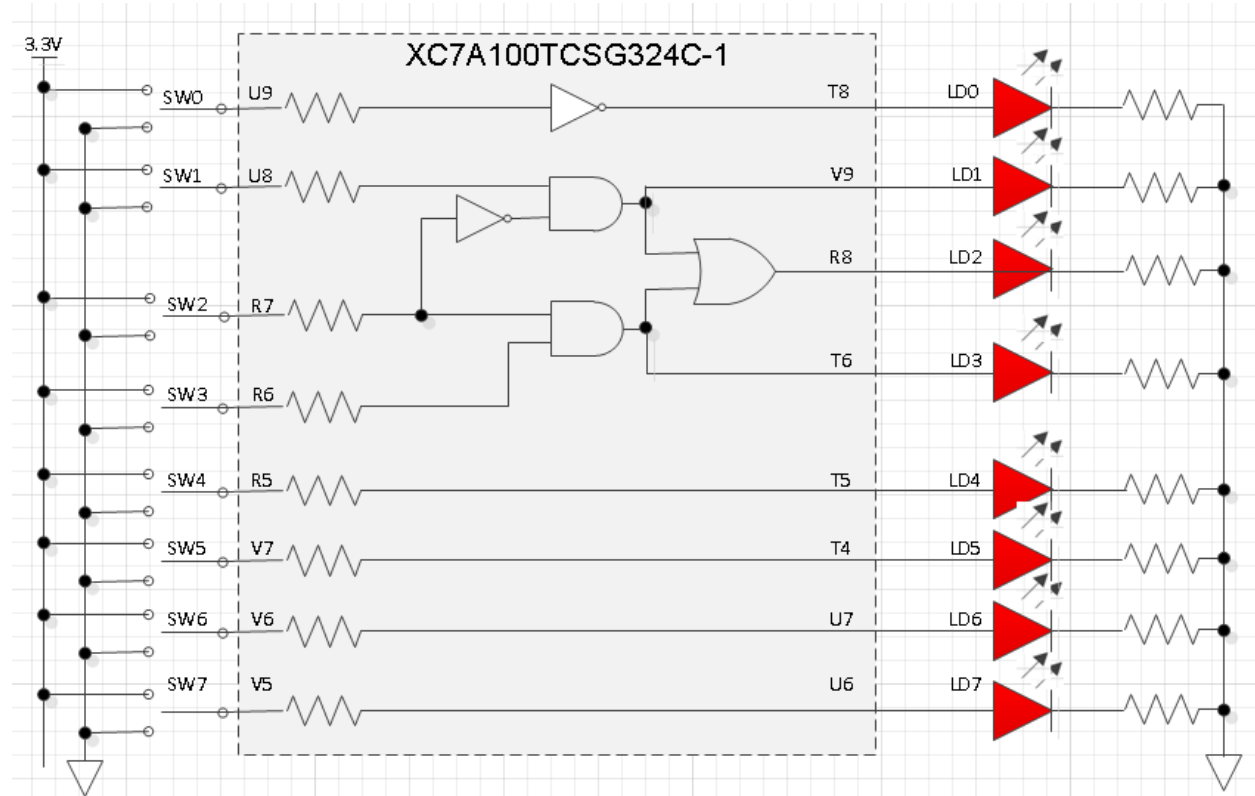


Figure 2. Completed Design

General Flow for this tutorial

- Create a Vivado project and analyze source files
- Simulate the design using XSim simulator
- Synthesize the design
- Implement the design
- Perform the timing simulation
- Verify the functionality in hardware using the Nexys4 board
- Run the tools in batch mode using the provided Tcd script

1 Create a Vivado Project using IDE

Step 1

- 1.1. **Launch Vivado and create a project targeting the XC7A100TCSG324-1 device and using the Verilog HDL. Use the provided tutorial.v and tutorial.xdc files from the sources directory.**
 - 1.1.1. Download vivadotutorial.zip from Piazza > Resources and extract to **W:\tutorial_1**
 - 1.1.2. Open Vivado by selecting **Start > All Programs > Xilinx 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 **W:\tutorial_1**, and click **Select**.
 - 1.1.5. Enter **tutorial** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

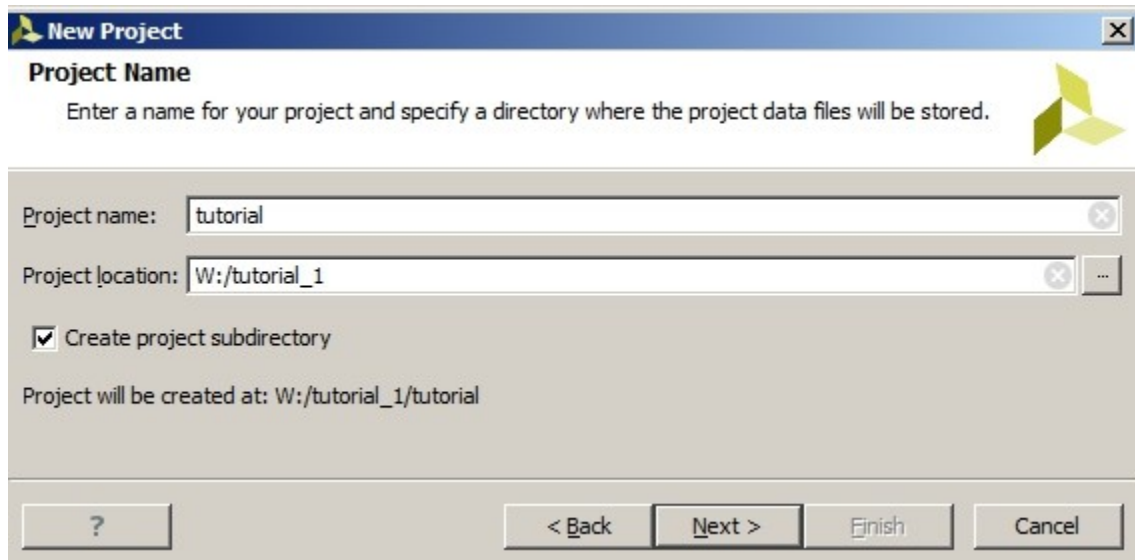


Figure 3. Project Name and Location entry

- 1.1.6. Select **RTL Project** option in the *Project Type* form, and click **Next**.
- 1.1.7. Select **Verilog** as the *Target Language* in the *Add Sources* form.
- 1.1.8. Click on the **Add Files...** button, browse to the **W:\tutorial_1** directory, select *tutorial.v*, click **Open**. Check the box that says *Copy sources into project* and then click **OK**.
- 1.1.9. Click **Next** to get to *Add Existing IP* form and **Next** again to reach *Add Constraints*

- 1.1.10.** Click on the **Add Files...** button, browse to the **W:\tutorial_1** directory (if necessary), select *tutorial.xdc* and click **Open** (if necessary). Ensure that *Copy constraints files into project* is selected and then click **OK**.

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 a board's schematic or board's user guide.

- 1.1.11.** In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section, select the **XC7A100TCSG324-1** part. Click **Next**.

New Project

Default Part
Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☒ Parts ☐ Boards

Filter

Product category: All Speed grade: All Remaining
 Family: Artix-7 Temp grade: C
 Package: csg324

Reset All Filters

Search: (3 matches)

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs
xc7a100tcsg324-3	324	135	240	126800	0	0	210
xc7a100tcsg324-2	324	135	240	126800	0	0	210
xc7a100tcsg324-1	324	135	240	126800	0	0	210

? < Back Next > Finish Cancel

Figure 4. Part Selection

- 1.1.12.** Click **Finish** to create the Vivado project.

Use the Windows Explorer and look at the **W:\tutorial_1** directory. You will find that the **tutorial.srcs** directories and the **tutorial.xpr** (Vivado) project file have been created. Two more directories, **constrs_1** and **sources_1**, are created under the **tutorial.srcs** directory; deep down under them, the copied **tutorial.xdc** (constraint) and **tutorial.v** (source) files respectively are placed.

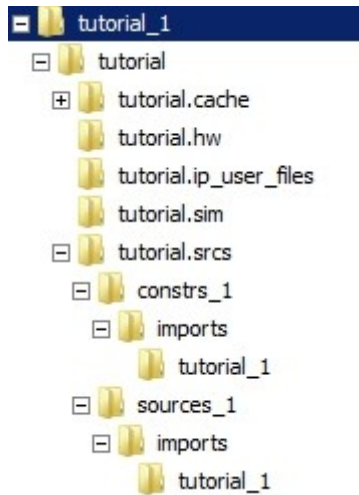


Figure 5. Generated directory structure

1.2. Open the tutorial.v source and analyze the content.

1.2.1. In the *Sources* pane, double-click the **tutorial.v** entry to open the file in text mode.

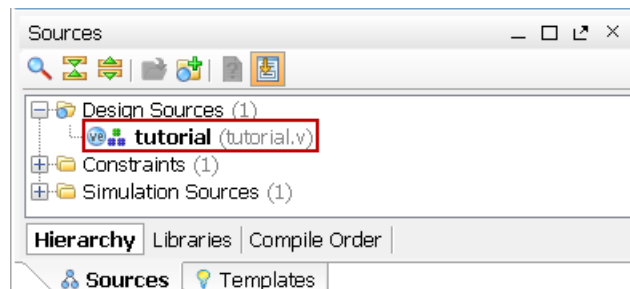


Figure 6. Opening the source file

- 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 tutorial.xdc source and analyze the content.

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

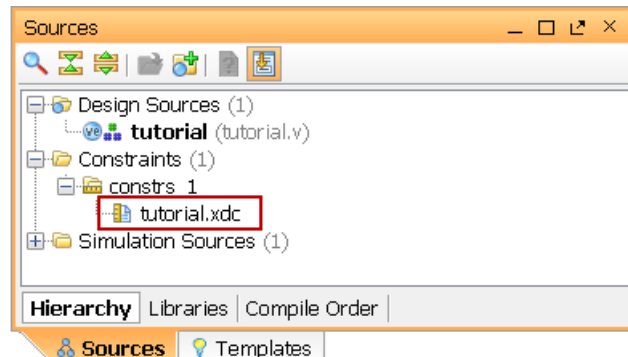


Figure 7. Opening the constraint file

- 1.3.2. Lines 1-14 defines the pin locations of the input switches [6:0] and lines 16-29 defines the pin locations of the output LEDs [6:0]. The swt[7] and led[7] are deliberately not defined so you can learn how to enter them using other methods in Step 1-5

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.

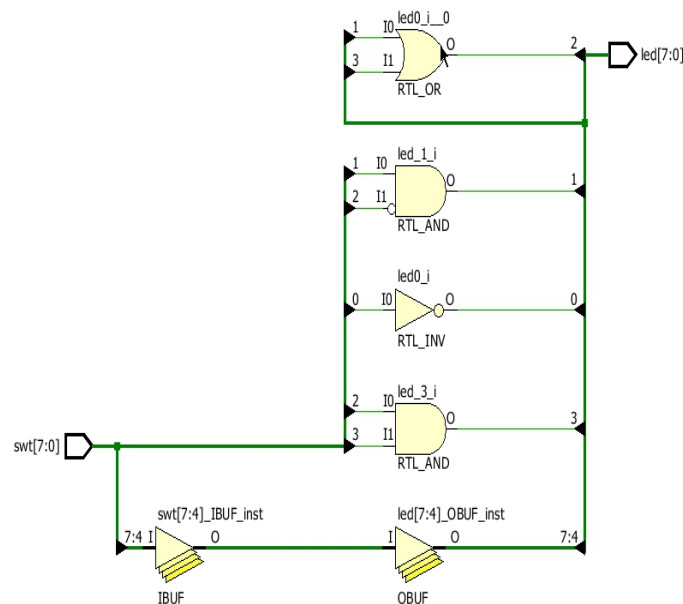


Figure 8. A logic view of the design

Notice that some of the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs (through I/O buffers) as modeled in the file.

1.5. Add I/O constraints for the missing LED and switch pins.

- 1.5.1. Once RTL analysis is performed, another standard layout called the I/O Planning layout is available. Click on the drop-down button and select the I/O Planning layout.

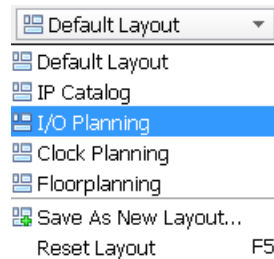


Figure 9. I/O Planning layout selection

Notice that the Package view is displayed in the Auxiliary View area, Device Constraints tab is selected, and I/O ports tab is displayed in the Console View area. Also notice that design ports (led and swt) are listed in the I/O Ports tab with both having multiple I/O standards.

Move the mouse cursor over the Package view, highlighting different pins. Notice the pin site number is shown at the bottom of the Vivado GUI, along with the pin type (User IO, GND, VCCO...) and the I/O bank it belongs to.

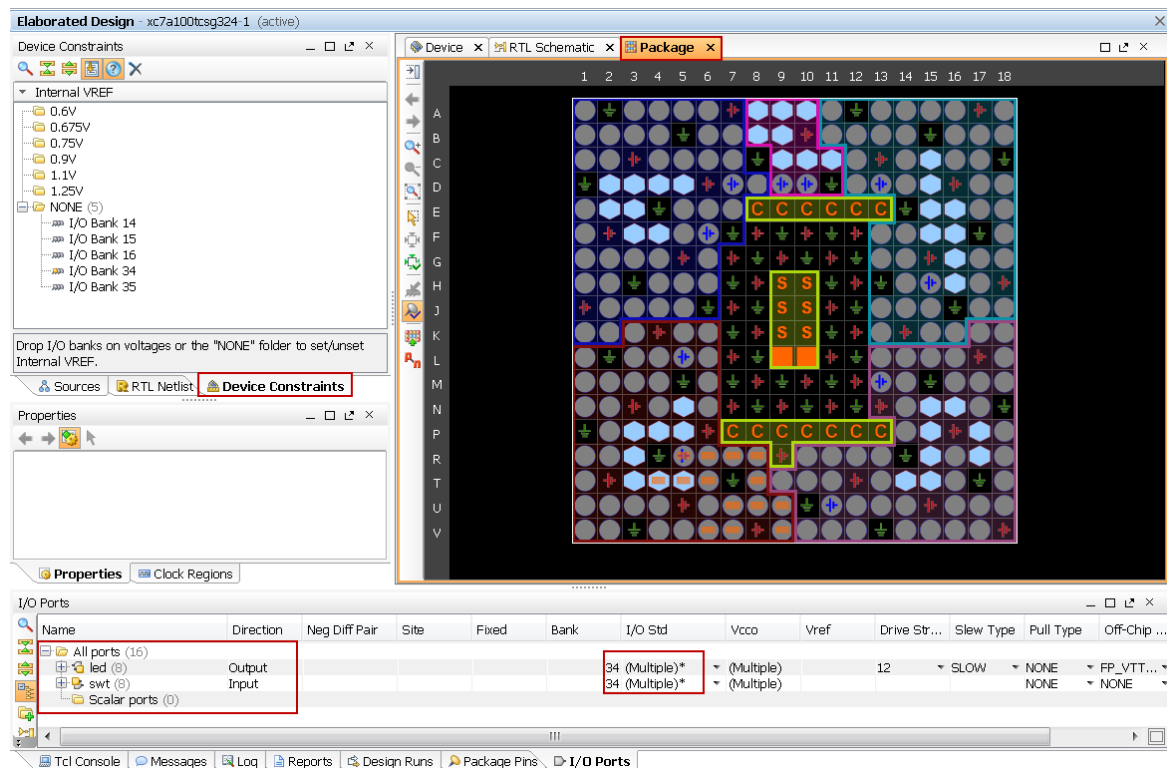


Figure 10. I/O Planning layout view

- 1.5.2. Expand the **led** and **swt** ports by clicking on the + box and observe that led [6:0] and swt[6:0] have assigned pins and uses the LVCMOS33 I/O standard whereas led[7] and swt[7] do not have

assigned pins and defaults to LVCMOS18; hence you can see multiple I/O standard in the collapsed view.

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std
All ports (16)						
led (8)						34 (Multiple)*
led[7]	Output			<input type="checkbox"/>		default (LVCMOS18)
led[6]	Output		U7	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[5]	Output		T4	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[4]	Output		T5	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[3]	Output		T6	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[2]	Output		R8	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[1]	Output		V9	<input checked="" type="checkbox"/>		34 LVCMOS33*
led[0]	Output		T8	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt (8)						34 (Multiple)*
swt[7]	Input			<input type="checkbox"/>		default (LVCMOS18)
swt[6]	Input		V6	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[5]	Input		V7	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[4]	Input		R5	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[3]	Input		R6	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[2]	Input		R7	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[1]	Input		U8	<input checked="" type="checkbox"/>		34 LVCMOS33*
swt[0]	Input		U9	<input checked="" type="checkbox"/>		34 LVCMOS33*
Scalar ports (0)						

Figure 11. I/O Ports tab

- 1.5.3. Click under the *Package Pin* column across **led[7]** row to see a drop-down box appear. Type **U** in the field to jump to Uxx pins, scroll-down until you see U16, select U16 and hit the *Enter* key to assign the pin. Notice after selecting the pin U16, the *Package Pin* column automatically places **led[7]** lower down in the column since it alphabetically arranges the site/pin names.

Name	Direction	Package Pin
led (8)		
led[7]	OUT	U16
led[6]	OUT	U7
led[5]	OUT	U8
led[4]	OUT	U9
led[3]	OUT	U11
led[2]	OUT	U12
led[1]	OUT	U13
led[0]	OUT	U14
swt (8)	IN	

Figure 12. Assigning pin

- 1.5.4. Similarly, click under the *I/O Std* column across the **led[7]** row and select LVCMOS33. This assigns the LVCMOS33 standard to the site.
- 1.5.5. You can also assign the pin constraints using tcl commands. Type in the following two commands in the Td Console tab to assign the R13 pin location and the LVCMOS33 I/O standard to **swt[7]** hitting the Enter key after each command.

```
set_property package_pin R13 [get_ports {swt[7]}]
set_property iostandard LVCMOS33 [get_ports [list {swt[7]}]]
```

Observe the pin and I/O standard assignments in the I/O Ports tab.

You can also assign the pin by selecting its entry (swt[7]) in the I/O ports tab, and dragging it to the Package view, and placing it at the R13 location. You can assign the LVCMOS33 standard by selecting its entry (swt[7]), selecting Configure tab of the I/O Port Properties window, followed by clicking the drop-down button of the I/O standard field, and selecting LVCMOS33.

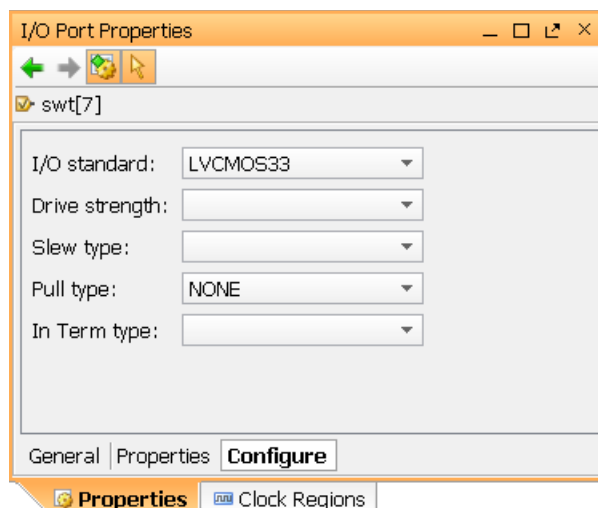


Figure 13. Assigning I/O standard through the I/O Port Properties form

- 1.5.6. Select **File > Save Constraints** and click **OK** to save the constraints in the **tutorial.xdc** file.

Note that the constraints are updated in the tutorial.xdc file under the tutorial project directory and not under the sources directory.

2 Simulate the Design using the XSim Simulator

Step 2

- 2.1. Add the tutorial_tb.v testbench file.

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

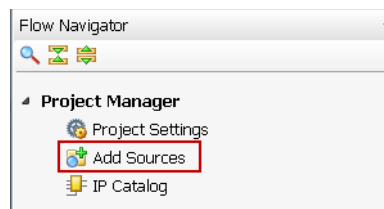


Figure 14. Add Sources

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

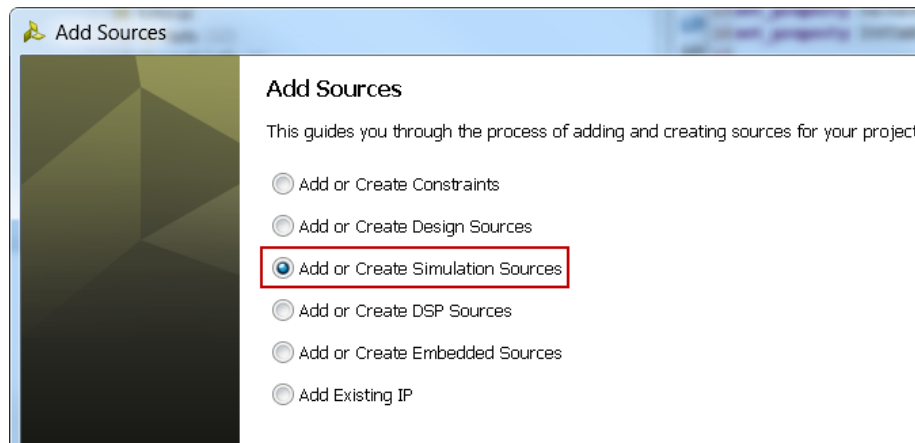


Figure 15. Selecting Simulation Sources option

2.1.3. In the *Add Sources Files* form, click the **Add Files...** button.

2.1.4. Browse to the **W:\tutorial_1** folder and select *tutorial_tb.v* and click **OK**.

2.1.5. Click **Finish**.

2.1.6. Select the *Sources* tab and expand the *Simulation Sources* group.

The *tutorial_tb.v* file is added under the *Simulation Sources* group, and **tutorial.v** is automatically placed in its hierarchy as a **tut1** instance.

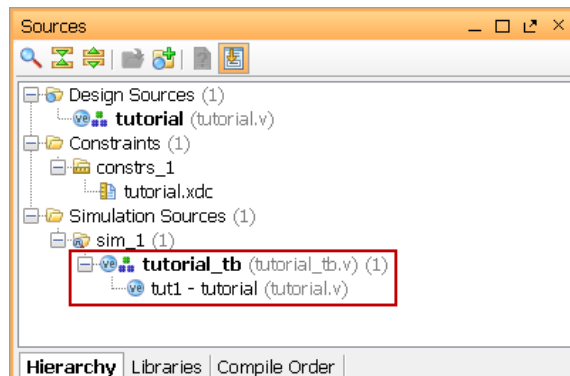


Figure 16. Simulation Sources hierarchy

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 **tutorial.srcs** directory, and that a copy of *tutorial_tb.v* is placed under **tutorial.srcs > sim_1 > imports > sources**.

2.1.8. Double-click on the **tutorial_tb** in the *Sources* pane to view its contents.

```

1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////
3 // Module Name: tutorial_tb
4 //////////////////////////////////////////////////
5 module tutorial_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    tutorial tut1(.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

```

Figure 17. 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 compares 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 XSim simulator.

2.2.1. Select **Simulation Settings** under the *Simulation* 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**.

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 XSim simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below.

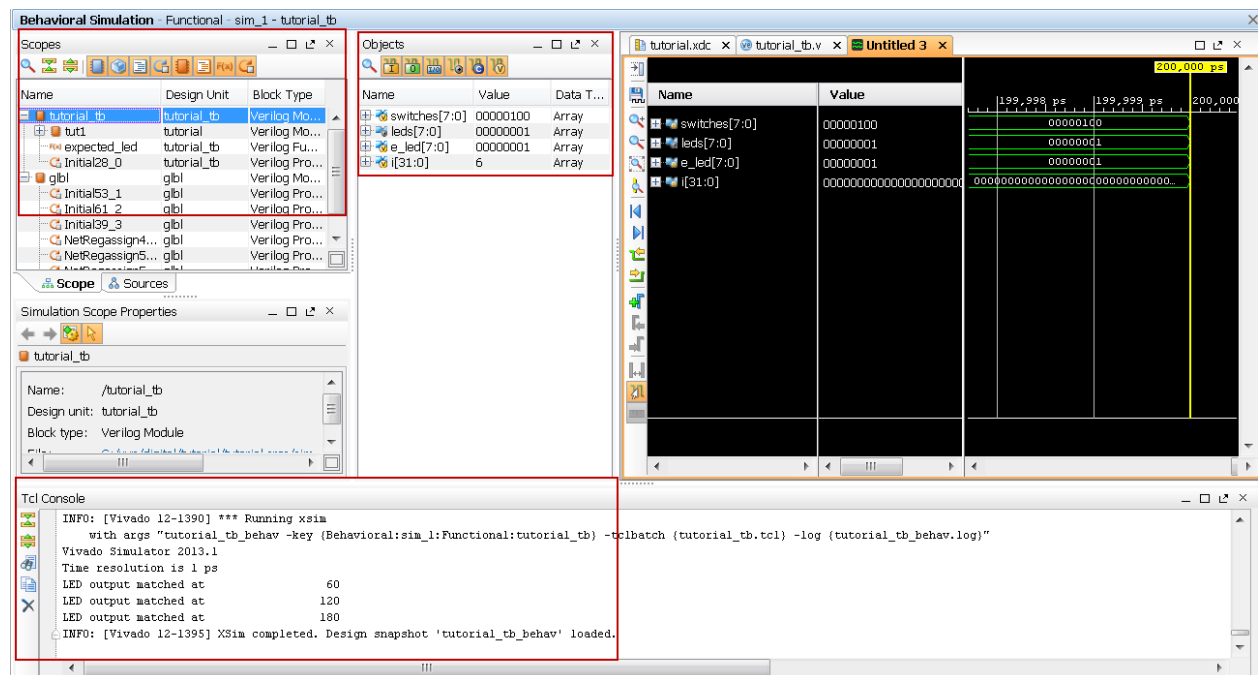


Figure 18. Simulator output

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

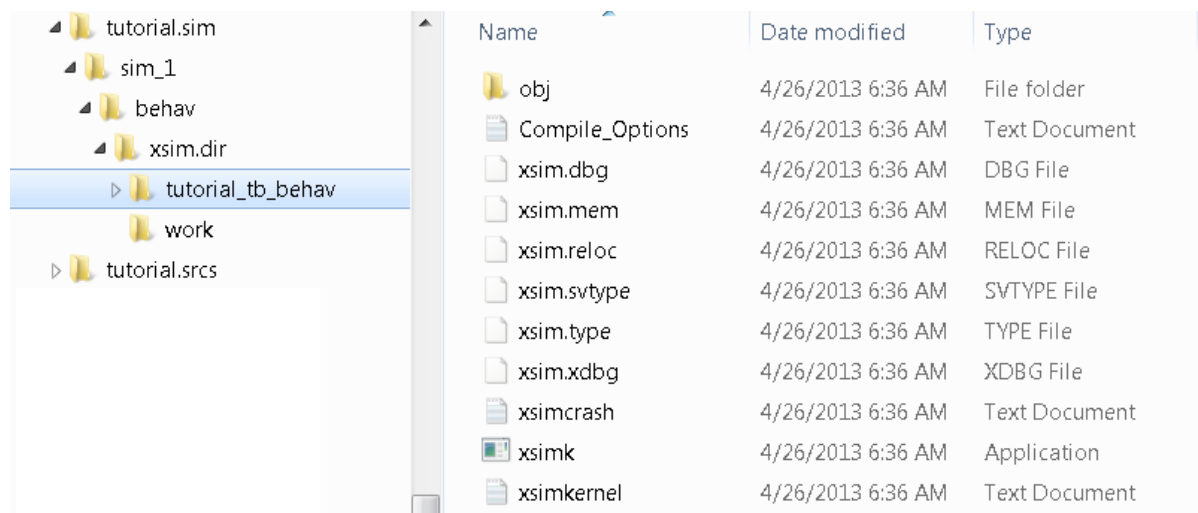



Figure 19. Directory structure after running behavioral simulation

2.2.4. Click on the **Zoom Fit** button () located left of the waveform window 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.

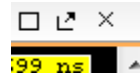


Figure 20. Float Button



Figure 21. 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 lower-level signals and continue to run the simulation for 500 ns.

- 2.4.1. Expand the **tutorial_tb** instance, if necessary, in the *Scopes* window and select the **tut1** instance.

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

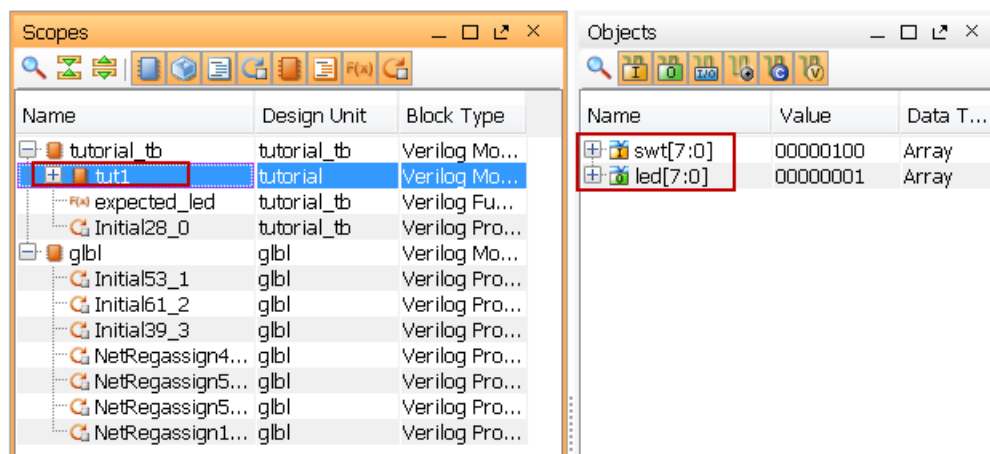









Figure 22. Selecting lower-level signals

- 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, select 10 and type over 500, click on the drop-down button of the units field and select ns (   500 ns   ) if we want to run for 500 ns (total of 700 ns), and click on the () button – button to left of 500.

The simulation will run for an additional 500 ns.

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

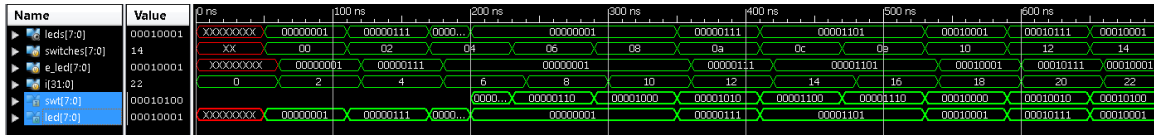


Figure 23. Running simulation for additional 500 ns

- 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.

3 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 tutorial.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. You may see "Default Layout" from the drop-down menu in the toolbar.

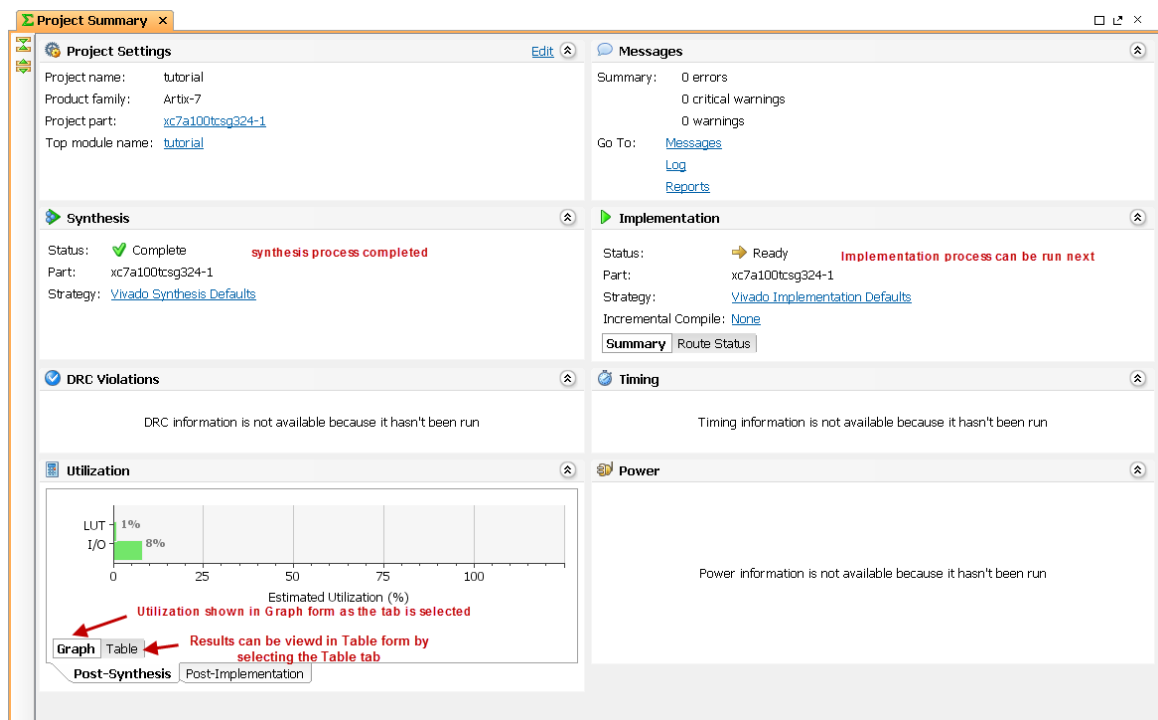


Figure 24. Project Summary view

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.

Resource	Estimation	Available	Utilization %
LUT	3	63400	1
I/O	16	210	8

Graph Table

Post-Synthesis Post-Implementation

Figure 25. Resource utilization estimation summary

3.1.5. Click on **Schematic** under the *Synthesized Design* tasks of *Synthesis* tasks of the *Flow Navigator* pane to view the synthesized design in a schematic view.

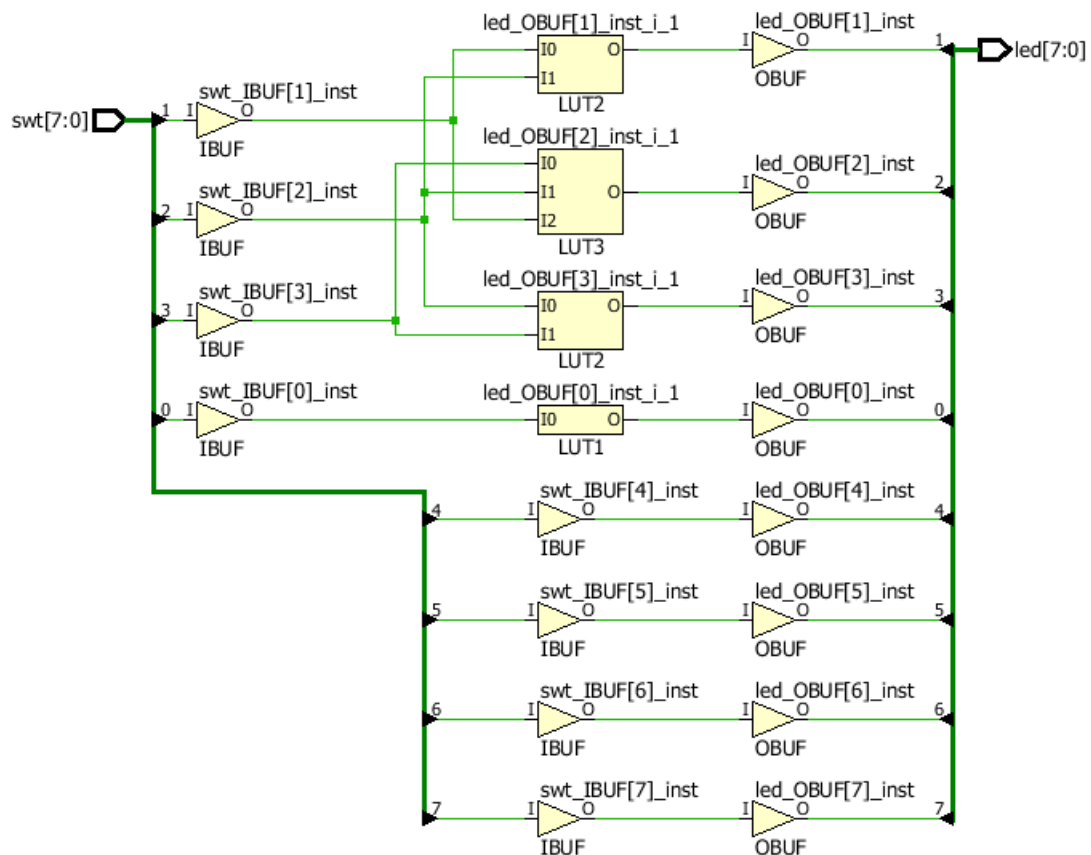


Figure 26. Synthesized design's schematic view

Notice that IBUF and OBUF 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 is mapped into four LUTs in the synthesized output.

Using the Windows Explorer, verify that **tutorial.runs** directory is created under **tutorial**. Under the **runs** directory, **synth_1** directory is created which holds several temporary sub-directories.

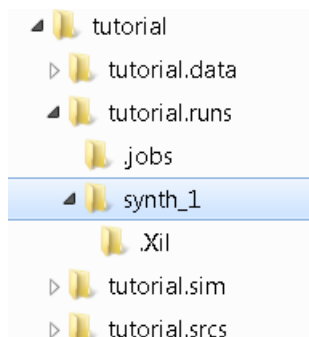


Figure 27. Directory structure after synthesizing the design

4 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 synthesis output files. 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** to close the synthesized design.

The implemented design will be opened. Click **OK** to see the device view.

4.1.4. In the *Netlist* pane, select one of the nets (e.g. led_OBUF[1]) and notice the net displayed in the Device view tab (you may have to zoom in to see it).

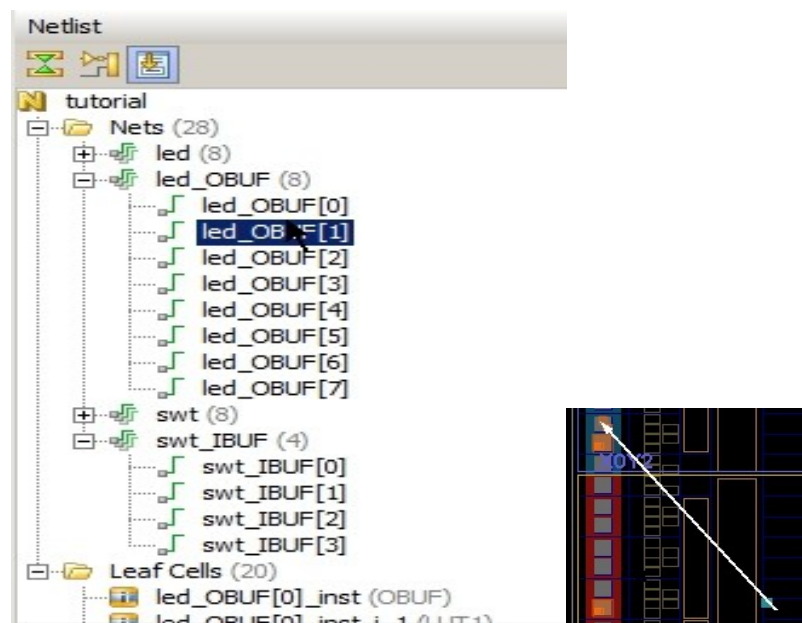


Figure 28. Viewing implemented design

4.1.5. 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.

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 combinational).

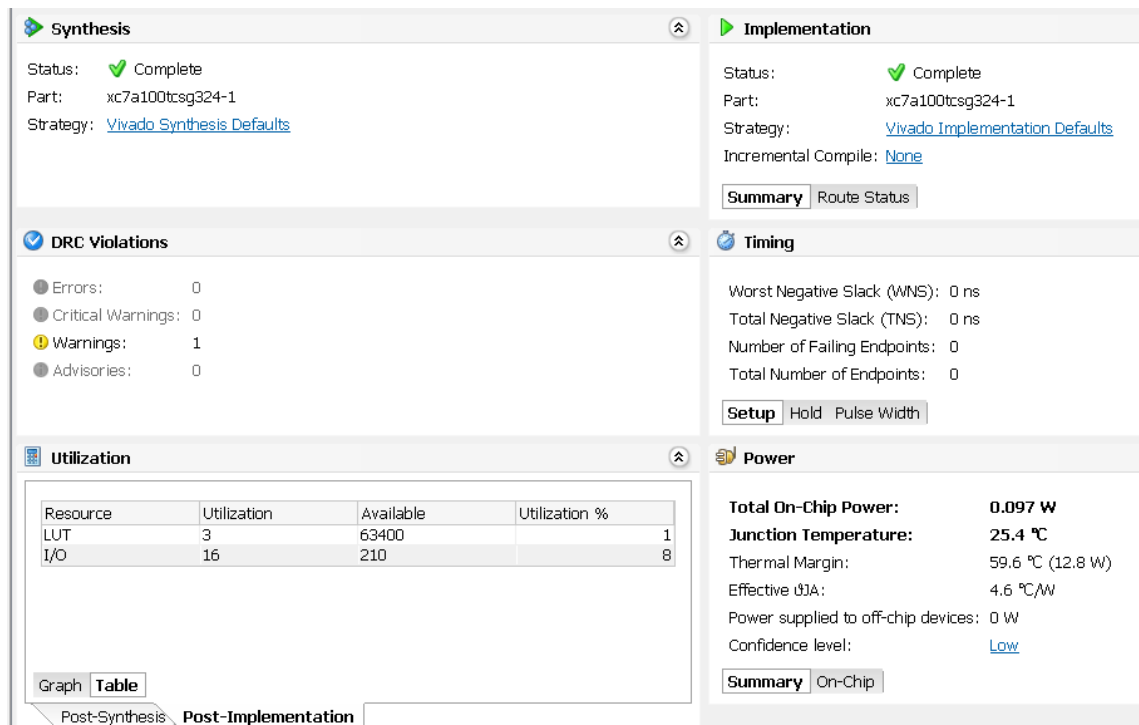


Figure 29. Implementation results

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

- 4.1.6. Select the **Reports** tab, and double-click on the *Utilization Report* entry under the *Place Design* section. The report will be displayed in the auxiliary view pane showing resources utilization. Note that since the design is combinatorial no registers are used.

5 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 XSim simulator will be launched using the implemented design and **tutorial_tb** as the top-level module..

Using the Windows Explorer, verify that **timing** directory is created under the **tutorial.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 55.000 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.

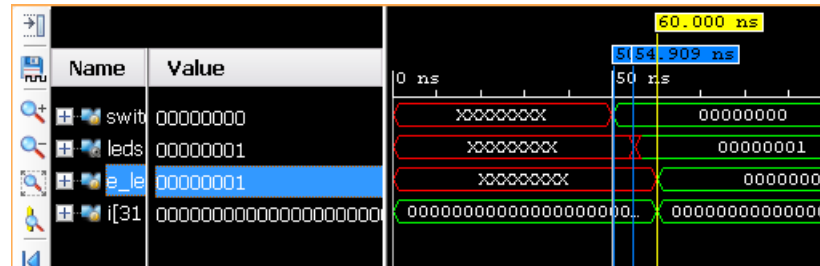


Figure 30. Timing simulation output

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 5.000 ns.

5.1.6. Close the simulator by selecting **File > Close Simulation** without saving any changes.

6 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. 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 four options will be displayed.

This process will have **tutorial.bit** file generated under **impl_1** directory which was generated under the **tutorial.runs** directory.

6.1.2. Make sure that the power supply source is jumper to *USB* and the provided Micro-USB cable is connected between the board and the PC. Note that you do not need to connect the power jack and the board can be powered and configured via USB alone

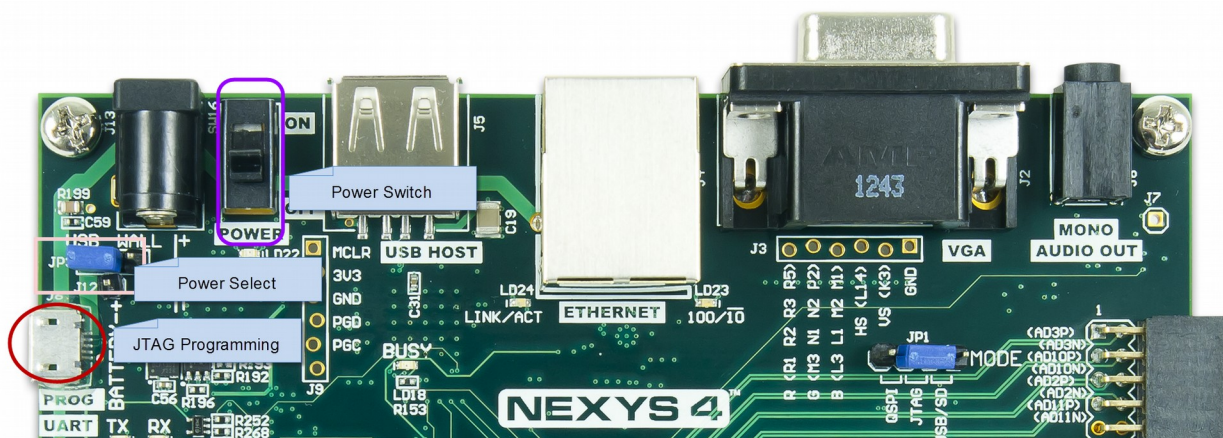


Figure 31. Board settings

6.1.3. Power **ON** the switch on the board.

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 target** link and **Auto Connect**.

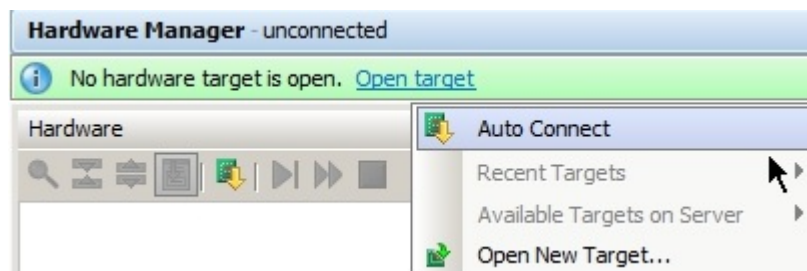


Figure 32. Auto Connect

6.1.6. Select the *Program device* option and click on the FPGA part *xc7a100t_0*.

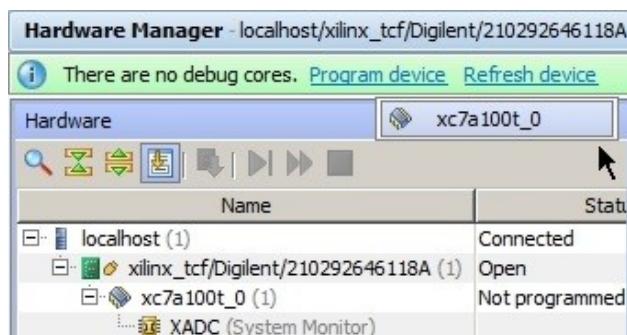


Figure 33. Programming file

6.1.7. A dialog will appear showing the path the bitstream file.

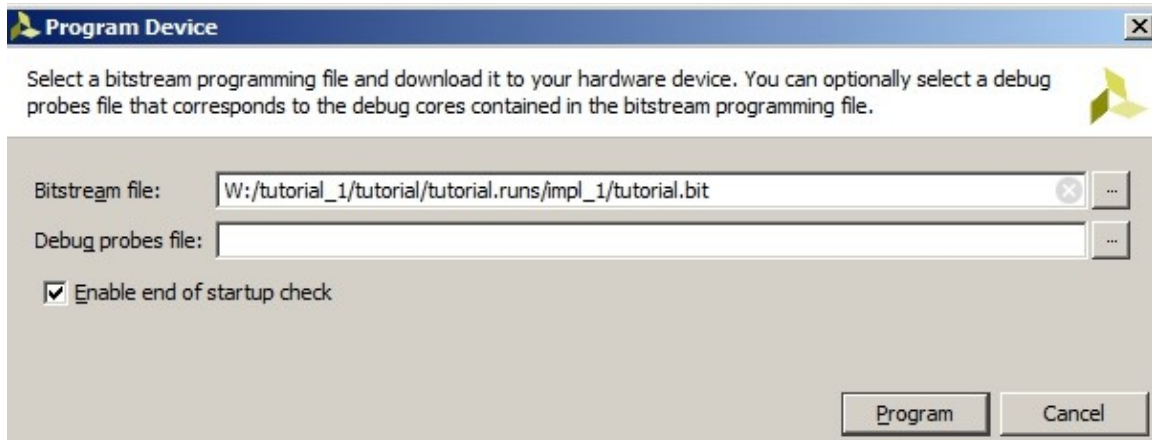


Figure 34. Programming Bitstream File

6.1.8. Click **Program** to program the FPGA.

The DONE light will lit when the device is programmed. You may see some LEDs lit depending on the switches position.

6.1.9. Verify the functionality by flipping switches and observing the output on the LEDs.

6.1.10. When satisfied, power **OFF** the board.

6.1.11. Close the hardware session by selecting **File > Close Hardware Manager**.

6.1.12. Click **OK** to close the session.

6.1.13. Close the **Vivado** program by selecting **File > Exit** and click **OK**.

7 Run the Tools in Batch Mode Using Tcl Script

Step 7

7.1. **Open a command prompt window, change the directory to W:\tutorial_1, run the settings64.bat from the <Vivado_2016_2_install_directory>. Run the tools in a batch mode using the provided tcl script file by executing the following command.**

```
vivado -mode batch -source sources\tutorial_tcl_with_sim.tcl
```

7.1.1. Select **Start > All Programs > Command Prompt** to open the command prompt window.

7.1.2. In the command prompt window, change the working directory to W:\tutorial_1

```
> W:
> cd W:\tutorial_1
```

- 7.1.3.** Execute the settings64.bat from the Vivado install directory by executing one of the following commands.

```
> C:\Xilinx\Vivado\2016.2\settings64.bat
```

- 7.1.4.** Run the provided Tcl script in the batch mode by executing the following command

```
vivado -mode batch -source tutorial_tcl_with_sim.tcl
```

The tools will be run and various directories will be created. Note that there may be some errors due to files that cannot be found. This could occur depending on the original directory names where the script is invoked and where sources are found. If the errors occur, it should be clear what paths are incorrect and the script can be modified accordingly

The Tcl script file is shown below.

```
1 set outDir .
2 create_project tutorial_tcl_with_sim $outDir/tutorial_tcl_with_sim -part xc7a100tcsq324-1
3 add_files -norecurse $outDir/sources/tutorial.v
4 import_files -fileset constrs_1 -force -norecurse $outDir/sources/tutorial.xdc
5 update_compile_order -fileset sources_1
6 synth_design -rtl -name rtl_1
7 set_property package_pin U6 [get_ports {led[7]}]
8 set_property iostandard LVCMOS33 [get_ports [list {led[7]}]]
9 set_property package_pin V5 [get_ports {swt[7]}]
10 set_property iostandard LVCMOS33 [get_ports [list {swt[7]}]]
11 set_property target_constrs_file \
12 $outDir/tutorial_tcl_with_sim/tutorial_tcl_with_sim.srcs/constrs_1/imports/sources/tutorial.xdc [current_fileset -constrset]
13 save_constraints -force
14 set_property SOURCE_SET sources_1 [get_filesets sim_1]
15 import_files -fileset sim_1 -norecurse $outDir/sources/tutorial_tb.v
16 update_compile_order -fileset sim_1
17 set_property runtime 200ns [get_filesets sim_1]
18 launch_xsim -simset sim_1 -mode behavioral
19 launch_runs synth_1
20 wait_on_run synth_1
21 launch_runs impl_1
22 wait_on_run impl_1
23 launch_runs impl_1 -to_step write_bitstream
```

Figure 37. The Tcl source file

Line 1 sets the output directory path

Line 2 creates the project directory tutorial_tcl_with_sim under the c:\xup\digital (\$outDir) directory targeting Artix-7 100 part.

Line 3 adds the source file.

Line 4 imports the constraints files in which all except two I/O pins are defined

Line 5 sets the top module file

Line 6 executes the rtl analysis command

Lines 7 through 10 adds the missing I/O pins constraints

Lines 11 through 13 saves the constraints in the target xdc file located under the created project directory

Line 14 through 17 sets up, reads, and compiles the testbench

Line 18 runs the behavioral simulation

Line 19 through 22 synthesizes and implements the design

Line 23 generates the bitstream

Note that wait_on_run on lines 20 and 22 are essential as the tools run in multi-thread mode, and since the following command uses the results generated by the previous command, the previous command should be completed.

- 7.1.5.** Using the Windows Explorer, browse through the generated project directory and verify that the bitstream file is generated in the impl_1 directory.

7.1.6. Close the command window.

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 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. The design flow was also carried out in the batch mode using the provided Tcl script.