

# Vivado Tutorial

## Introduction

This tutorial guides you through the design flow using Xilinx Vivado software to create a simple digital circuit using VHDL. 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-7 based Nexys4 board. The typical design flow is shown below. The circled number indicates the corresponding step in this tutorial.

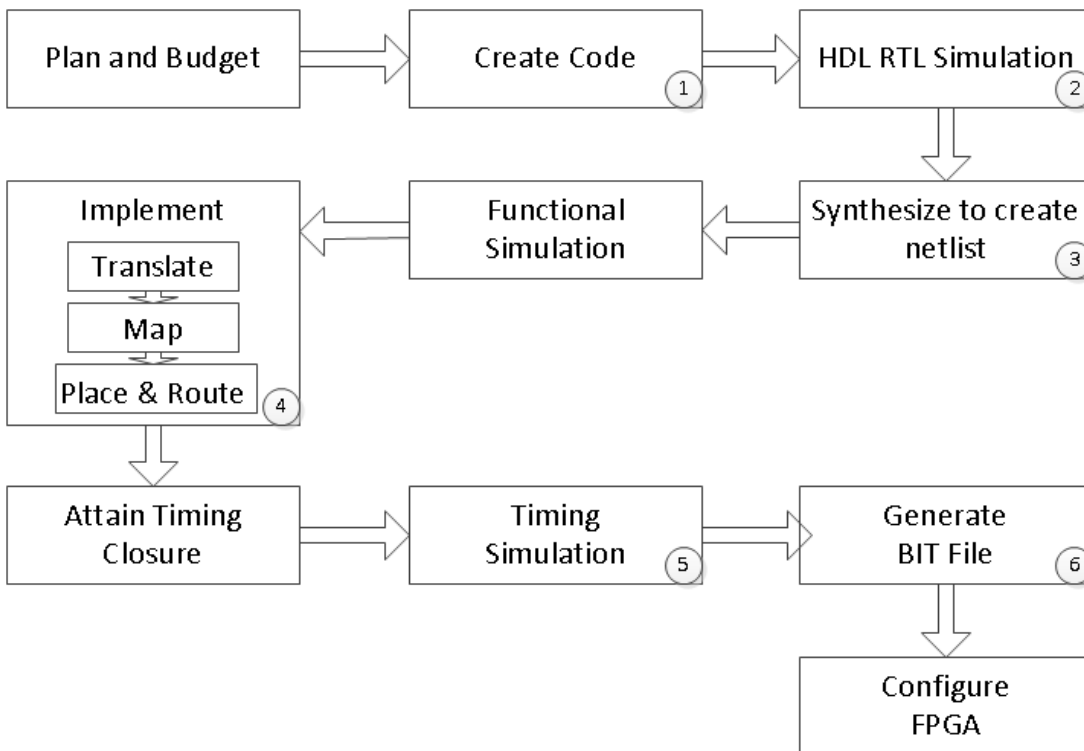


Figure 1. A typical design flow

## 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 user constraint file (XDC) to constrain pin locations
- Simulate the design using the XSIM simulator
- Synthesize and implement the design
- Generate the bitstream
- Download the design and verify the functionality

## Procedure

This tutorial is broken into steps that consist of general overview statements providing information on 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**.

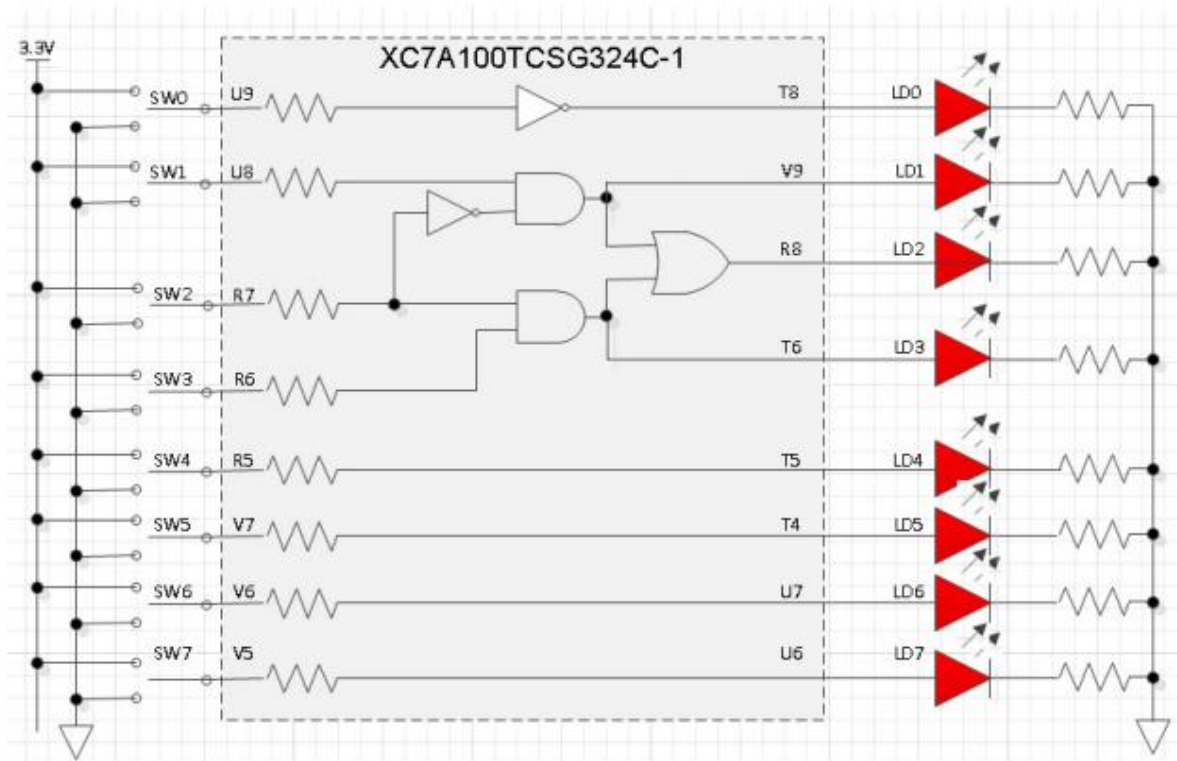


Figure 2. Completed Design

## 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 Nexys-4

## Create a Vivado Project

### Step 1

**1-1. Launch Vivado and create a project targeting the XC7A100T-1CSG324C device and using the VHDL. Use the provided tutorial.vhd and tutorial.xdc files in sources directory.**

**1-1-1.** Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3**.

- 1-1-2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
- 1-1-3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **c:\xup\digital**, and click **Select**.
- 1-1-4. Enter **tutorial** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

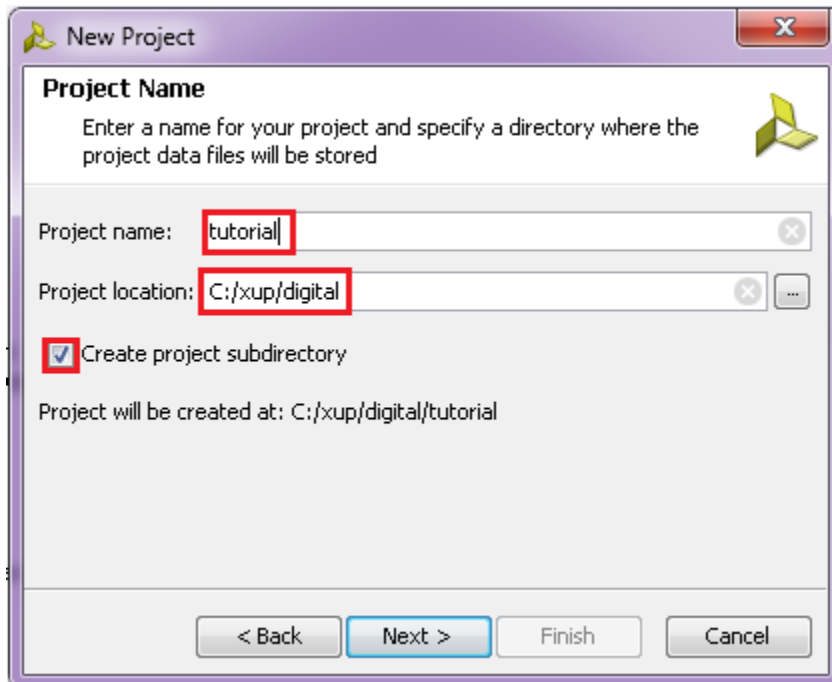
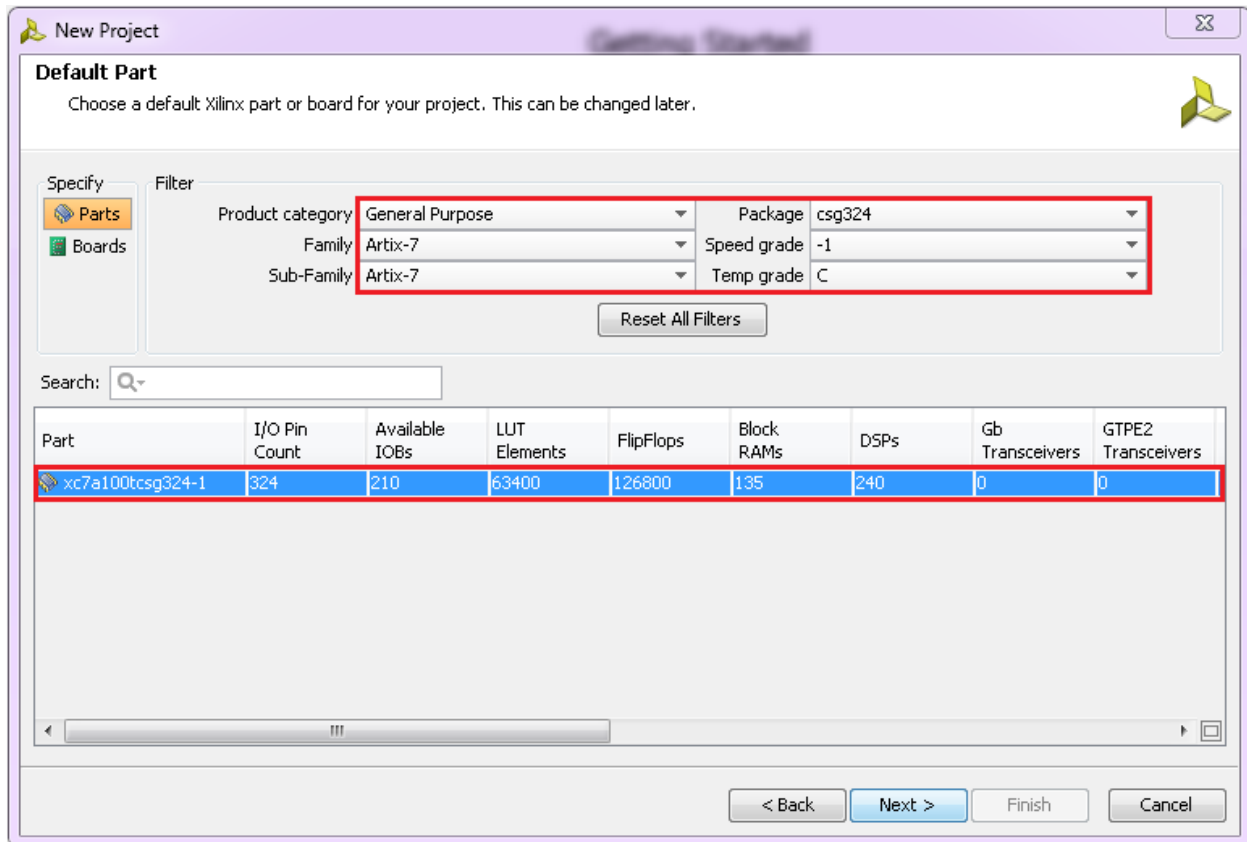


Figure 3. Project Name and Location entry

- 1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.
- 1-1-6. Select **VHDL** as the *Target Language* and as the *Simulator language* in the *Add Sources* form.
- 1-1-7. Click on the **Add Files...** button, browse to the **c:\xup\digital\sources** directory, select *tutorial.vhd*, click **Open**, and then click **Next**.
- 1-1-8. Click **Next** at the *Add Existing IP* form.
- 1-1-9. Click on the **Add Files...** button, browse to the **c:\xup\digital\sources** directory, select *tutorial.xdc* click **Open**, and then click **Next**.

The XDC constraint 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.

- 1-1-10. In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section, select the **xc7a100tcsq324-1** part. Click **Next**.



**Figure 4. Part Selection**

**1-1-11.** Click **Finish** to create the Vivado project.

**1-1-12.** Use the Windows Explorer and look at the **c:\xup\digital\tutorial** directory. You will find that the **tutorial.data** and **tutorial.srcs** directories and the **tutorial.xpr** (Vivado) project file have been created. The **tutorial.data** directory is a place holder for the Vivado program database. 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.vhd** (source) files respectively are placed.

## **1-2. Open the tutorial.vhd source and analyze the content.**

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

The design takes input from slide switches 0 to 7 on the Nexys4 board and toggles the LEDs on the board. Since combinatorial logic is inserted between some switches, the LEDs will turn on/off depending on the pattern of the switches. This is a very basic combinatorial logic demo.

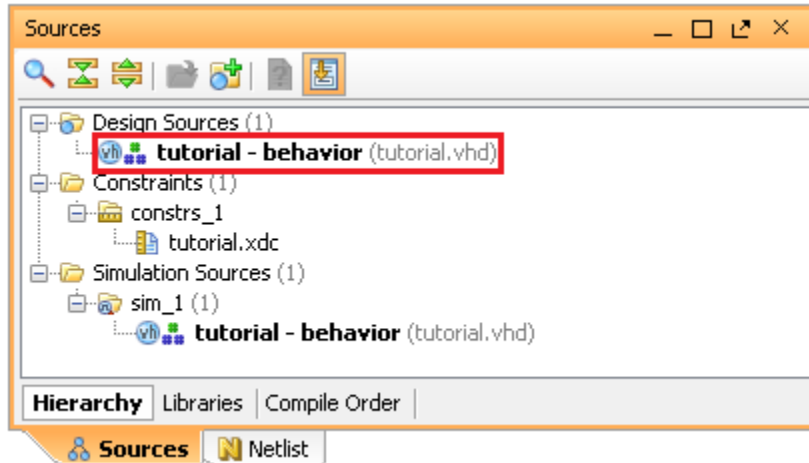


Figure 5. Opening the source file

### 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.

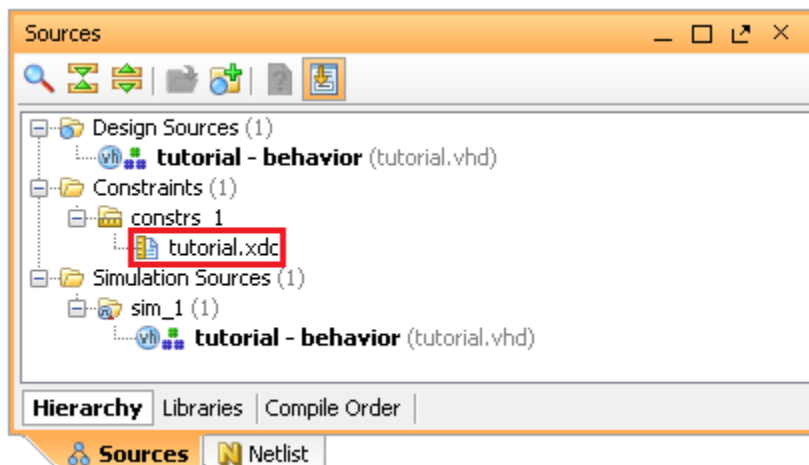


Figure 6. Opening the constraint file

- 1-3-2. Lines 1-22 defines the pin locations of the input switches [6:0] and lines 26-48 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. In the *Sources* pane, select the **tutorial.vhd** entry, and click on **Schematic** (you may have to expand the *Open Elaborated Design* entry) under the *RTL Analysis* tasks of the *Flow Navigator* pane.

A logic view of the design is displayed.

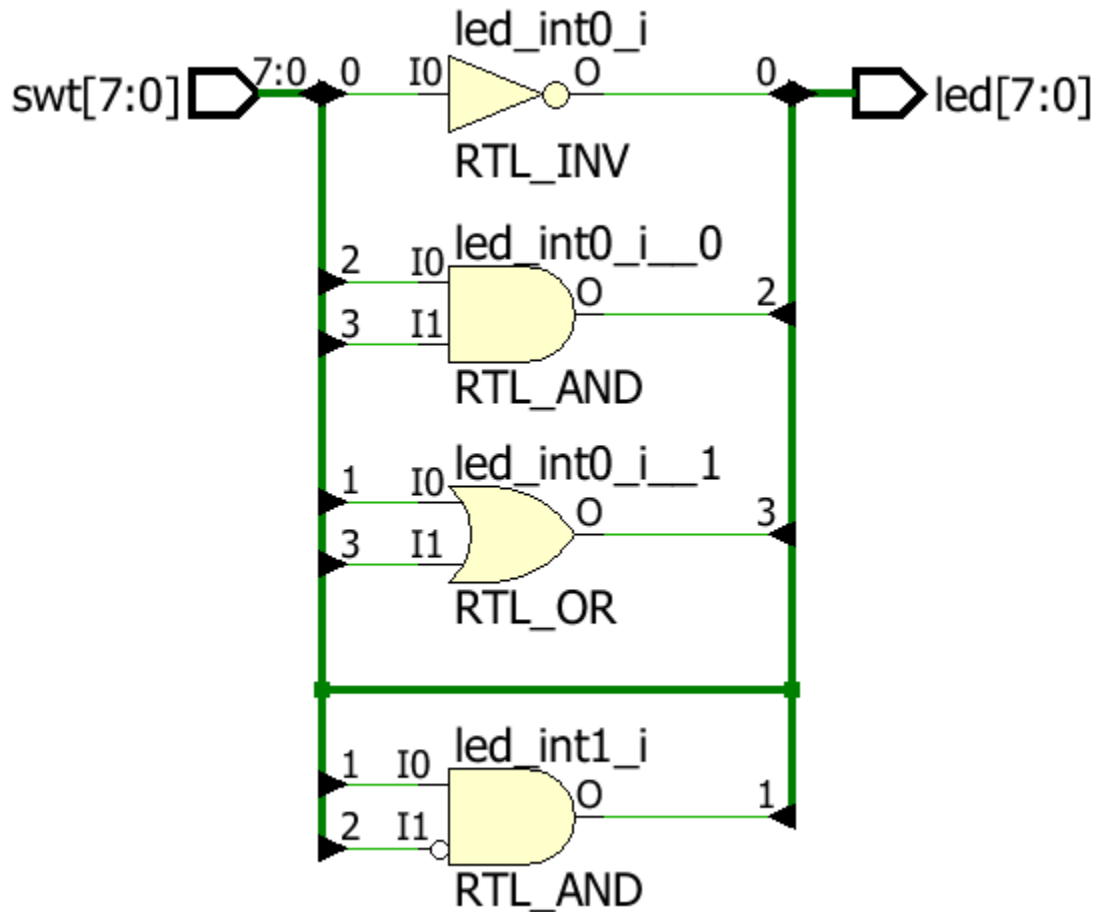


Figure 7. A logic view of the design

Notice that some of the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs 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.

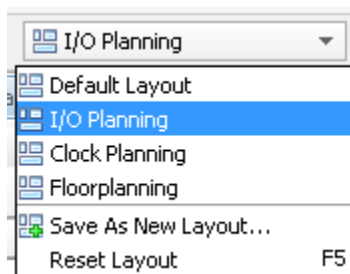


Figure 8. I/O Planning layout selection

Notice that the Package view is displayed in the Auxiliary View area, RTL Netlist 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.

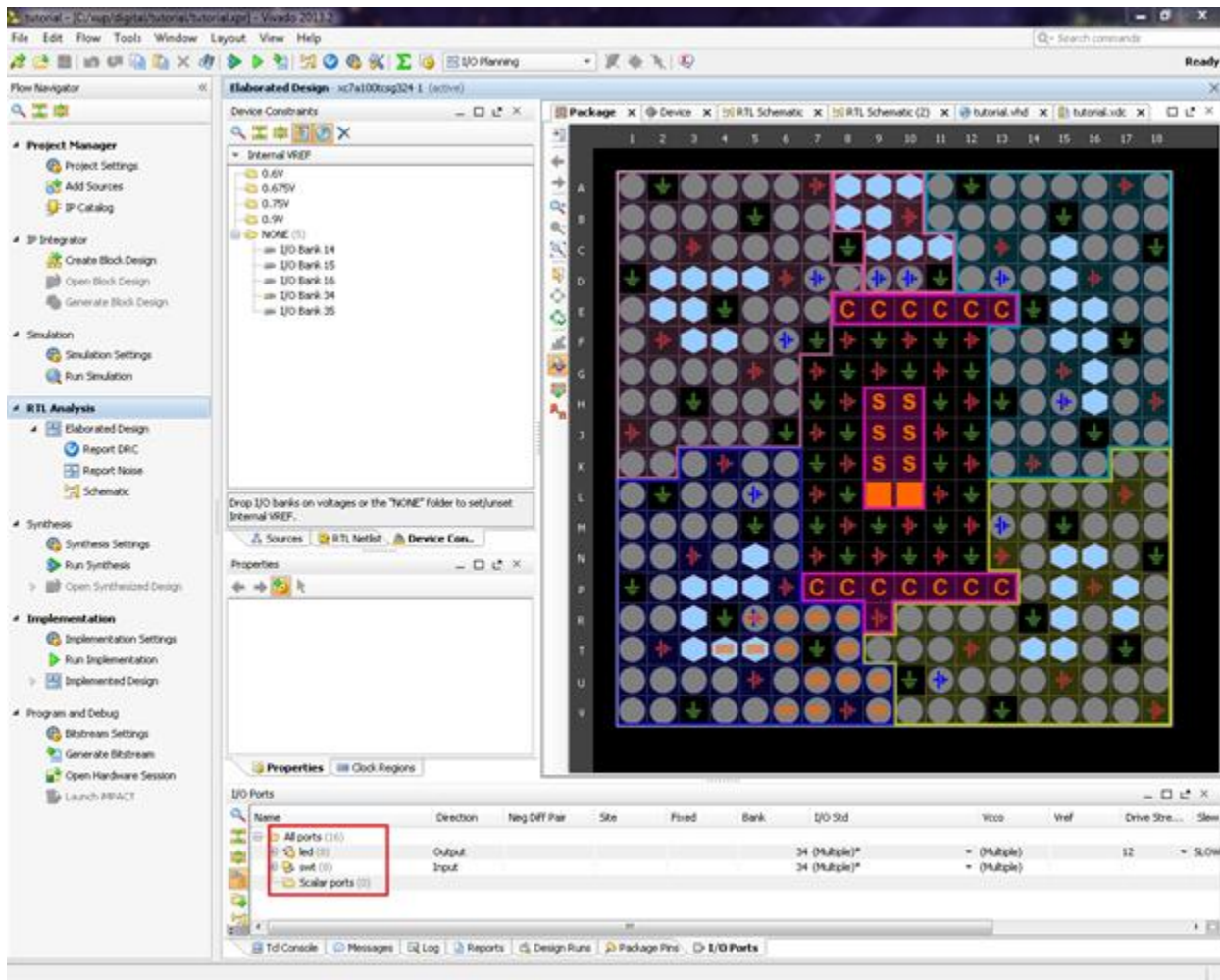


Figure 9. I/O Planning layout view

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 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)	Output					34 (Multiple)*
led[7]	Output					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)	Input					34 (Multiple)*
swt[7]	Input					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 10. I/O Ports tab

- 1-5-3.** Click under the *Site* column across **led[7]** row to see a drop-down box appear. Type **V** in the field to jump to Vxx pins, scroll-down until you see V5, and then select V5 and hit the *Enter* key to assign the pin.

Name	Direction	Neg Diff Pair	Site
All ports (16)			
led (8)	Output		
led[7]	Output		v
led[6]	Output		
led[5]	Output		
led[4]	Output		
led[3]	Output		
led[2]	Output		
led[1]	Output		
led[0]	Output		
swt (8)	Input		
swt[7]	Input		

Figure 11. Assigning pin

- 1-5-4.** Similarly, click under the *I/O Std* column across the **led[7]** row and select *LVCOMS33*.

- 1-5-5.** Similarly, assign the *U6* pin location and the *LVCSMOS33* I/O standard to **swt[7]**.

You can assign the pin by selecting its entry in the I/O ports tab, and dragging it to the Package view, and placing it at the *U6* location.

You can also assign the LVCMOS33 standard by selecting its entry, selecting the Configure tab of the I/O Port Properties window, followed by clicking the drop-down button of the I/O standard field, selecting LVCMOS33.



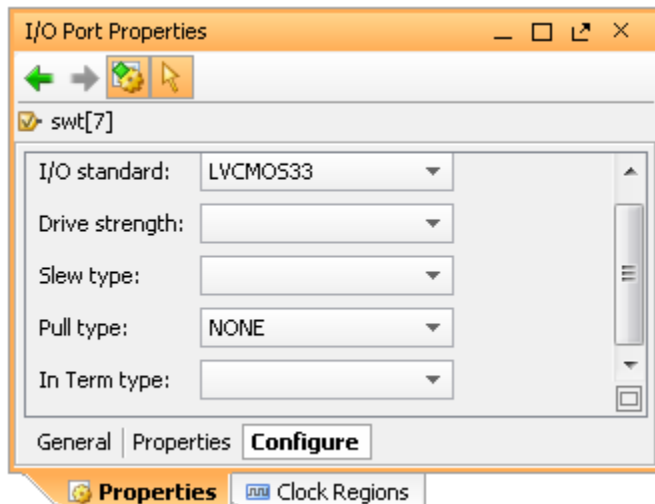


Figure 12. Assigning I/O standard through the I/O Port Properties form

You can also assign the pin constraints using tcl commands. Type in the following two commands in the Tcl Console tab to assign the V5 pin location and the LVCMOS33 I/O standard to **swt[7]** hitting the Enter key after each command.

```
set_property package_pin V5 [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.

**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.

## Simulate the Design using the ISim Simulator

## Step 2

### 2-1. Add the tutorial\_tb.vhd testbench file.

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

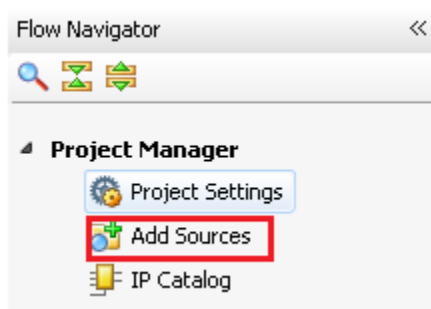


Figure 13. Add Sources

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

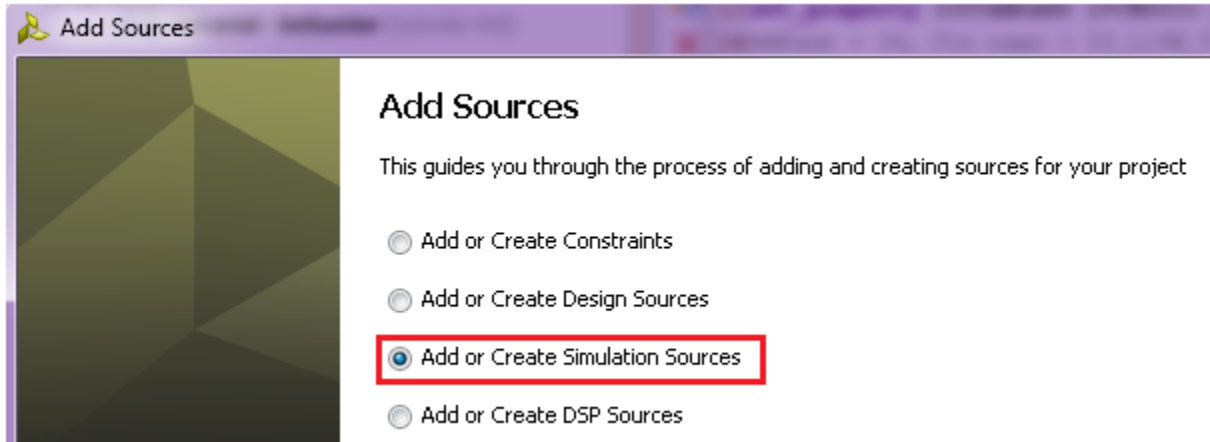


Figure 14. Selecting Simulation Sources option

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

2-1-4. Browse to the `c:\xup\digital\sources` folder and select `tutorial_tb.vhd` and click **OK**.

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

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

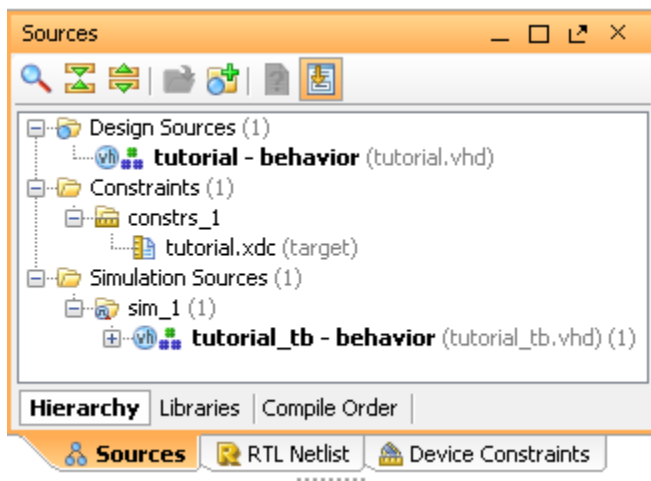


Figure 15. Simulation Sources hierarchy

2-1-6. Using the Windows Explorer, verify that the `sim_1` directory is created at the same level as `constrs_1` and `sources_1` directories and that a copy of `tutorial_tb.vhd` is placed.

2-1-7. Double-click on the **tutorial\_tb** to view its contents.

The VHDL testbench has the same structure as any VHDL design source code. There are a few exceptions that need some explanation. After the library declarations, note that the Entity declaration is left empty on Lines 16 and 17. The Unit Under Test (UUT; or the VHDL code being simulated) is instantiated as a component declaration from Lines 20 to 25. To generate the expected results during simulation, Lines 38 through 48 emulate the behavior of the UUT. Lines

49 to 52 is the port declaration for the UUT. Lines 56 through 86 define the stimuli generation and compares the expected output against the UUT output. Line 87 ends the testbench.

To provide feedback to the user via the Vivado simulator console window, examine Lines 74 through 80. Note multiple lines have been concatenated into one line, separated by the VHDL end of line character “;”.

## 2-2. Simulate the design for 100 ns using the XSIM simulator.

**2-2-1.** Select tutorial\_tb under the *Simulation Sources* group, and click on **Run Simulation** under the *Project Manager* tasks of the *Flow Navigator* pane. Select the **Run Behavioral Simulation** option.

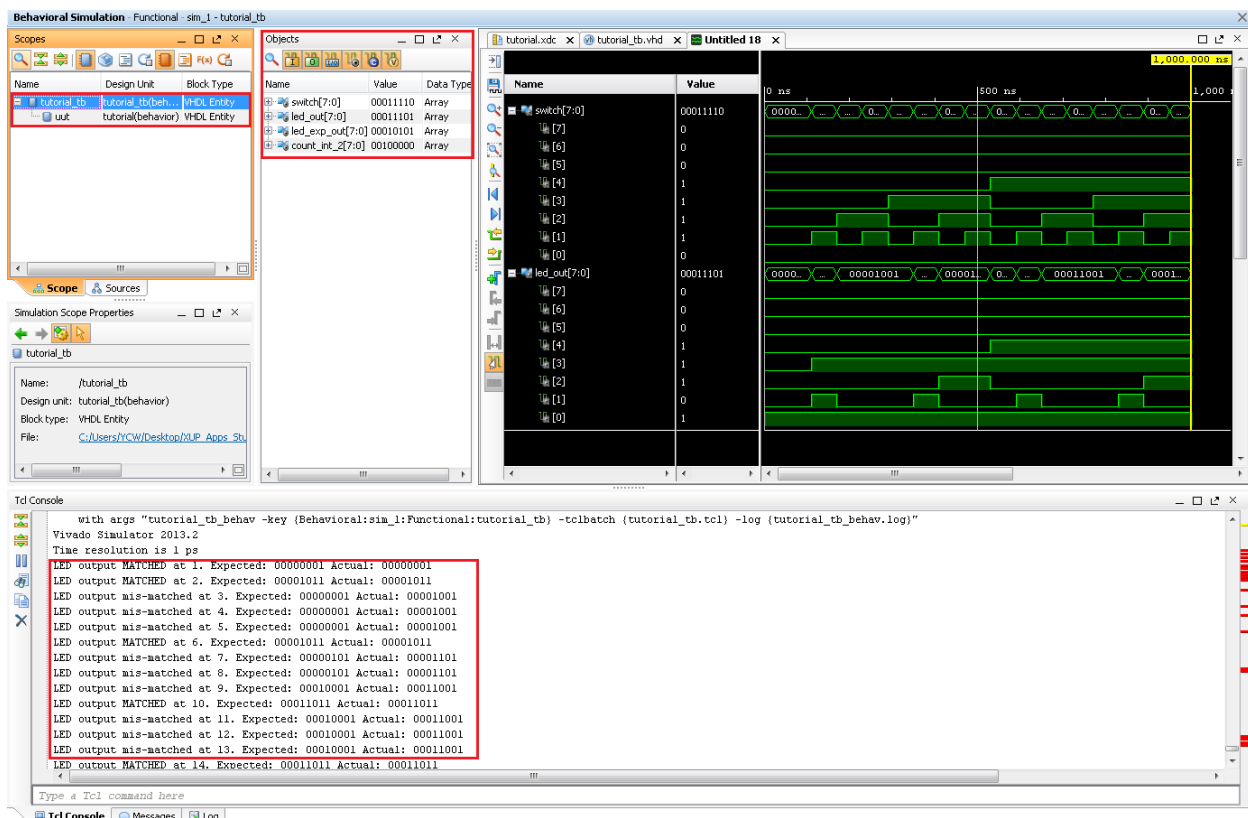
A **Launch Behavioral Simulation** dialog box will appear.

**2-2-2.** Click on the **Options...** button.

**2-2-3.** Select the *Simulation* tab and change the simulation time to **100 ns**.

**2-2-4.** Click **OK** and then click **Launch**.

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. You will see matching and mismatching results from the UUT and the procedure in the testbench. The mismatches were deliberately inserted to demonstrate the *TEXTIO* functionality of the VHDL testbench.

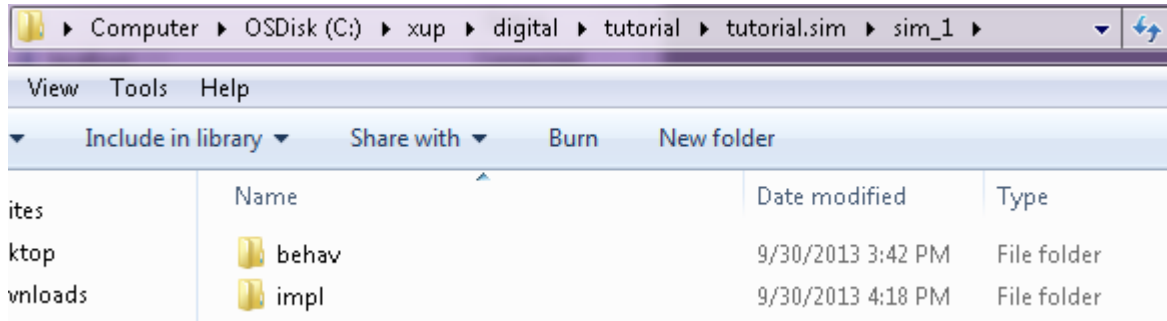


**Figure 16. Simulator output**

You will see four main views: (i) *Scopes*, where the testbench hierarchy (in collapsed form), (ii) *Objects*, where top-level signals are displayed, (iii) the waveform window, and (iv) *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 17. Directory structure after running behavioral simulation**

Please refer to the following URL:

[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2013\\_2/ug900-vivado-logic-simulation.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug900-vivado-logic-simulation.pdf)

to learn more about the Vivado simulator.

**2-2-5.** 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.

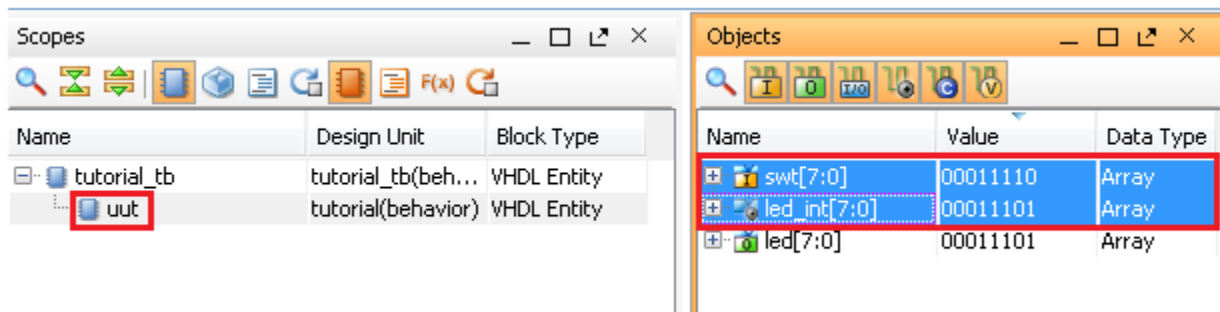
## 2-3. Change display format if desired.

**2-3-1.** Highlight **switch[7:0]** to select it. Change the radix to *Hexadecimal*. Leave the **led\_out[7:0]** and **led\_exp\_out[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 in the *Instances and Processes* window and select the **UUT** instance.

**swt[7:0]** and **led[7:0]** 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.

Notice that the signals are added, but the content is not refreshed.

- 2-4-3.** In the waveform window, select 100 ns and type over 500 ns (  ns ) if we want to run for 500 ns (a total of 600 ns) and hit enter.

The simulation will run for an additional 500 ns.

- 2-4-4.** Click on the *zoom-fit* button and notice that **swt[7:0]** is updated from 100 ns to 600 ns period. This is because we added an additional 500 ns from the previous step.

**2-4-5.**

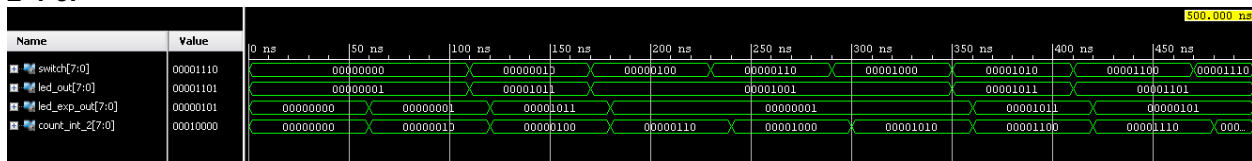


Figure 19. Running simulation for additional 5000 ns

- 2-4-6.** Close the simulator by selecting **File > Exit** in the Vivado simulator window 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.** Select **tutorial** under the *Design Sources* group, and click on **Run Synthesis** under the *Synthesis* tasks of the *Flow Navigator* pane.

The synthesis process will be run on the tutorial.vhd file (and all its hierarchical files if 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.

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

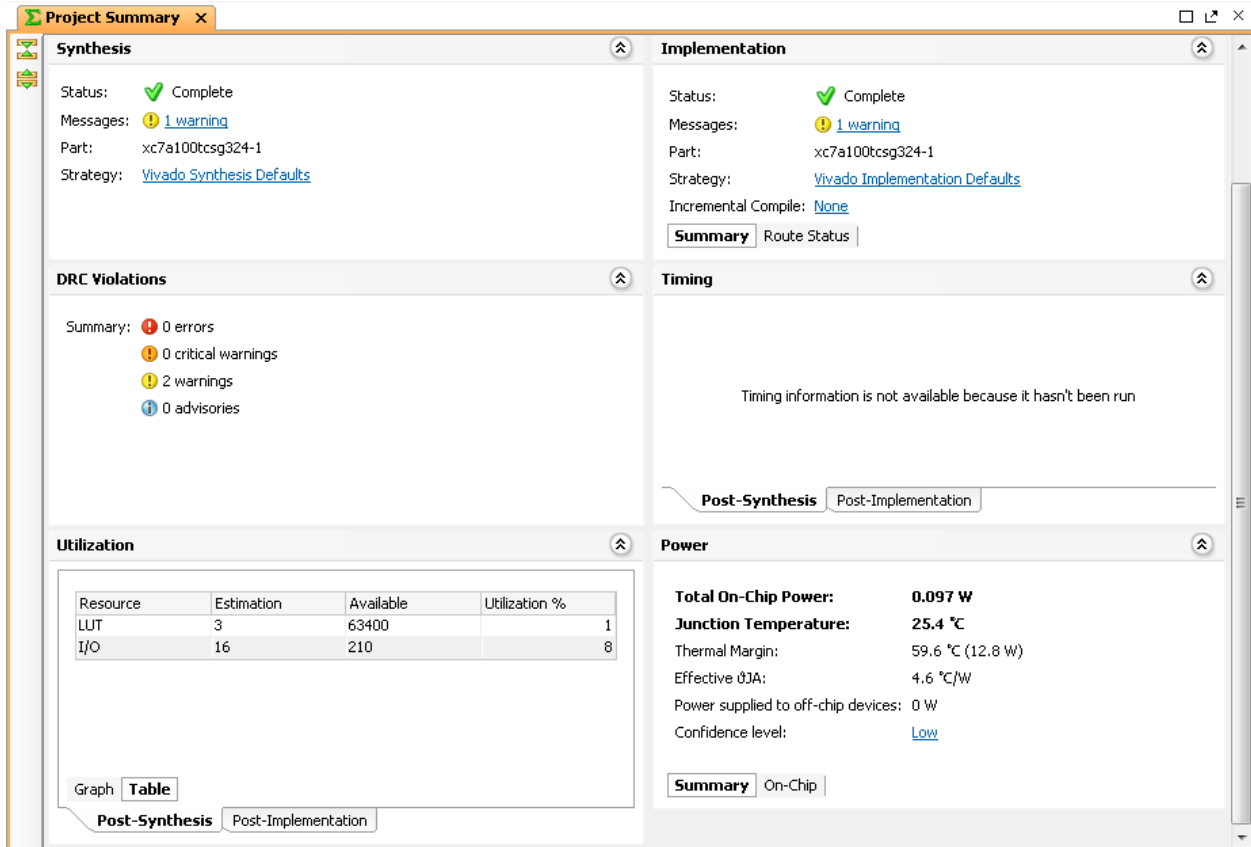


Figure 20. Project Summary view

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

3-1-4. Click on the **Show Graph** link in the **Project Summary** tab.

Notice that there are an estimated three LUTs and 16 IOs (8 input and 8 output) that are used.

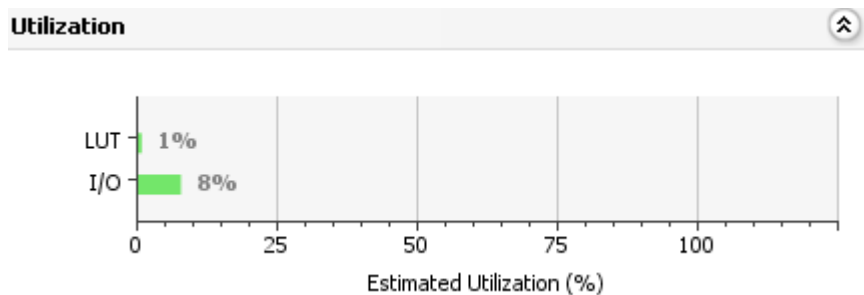


Figure 21. Resource utilization estimation summary

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 schematic view.

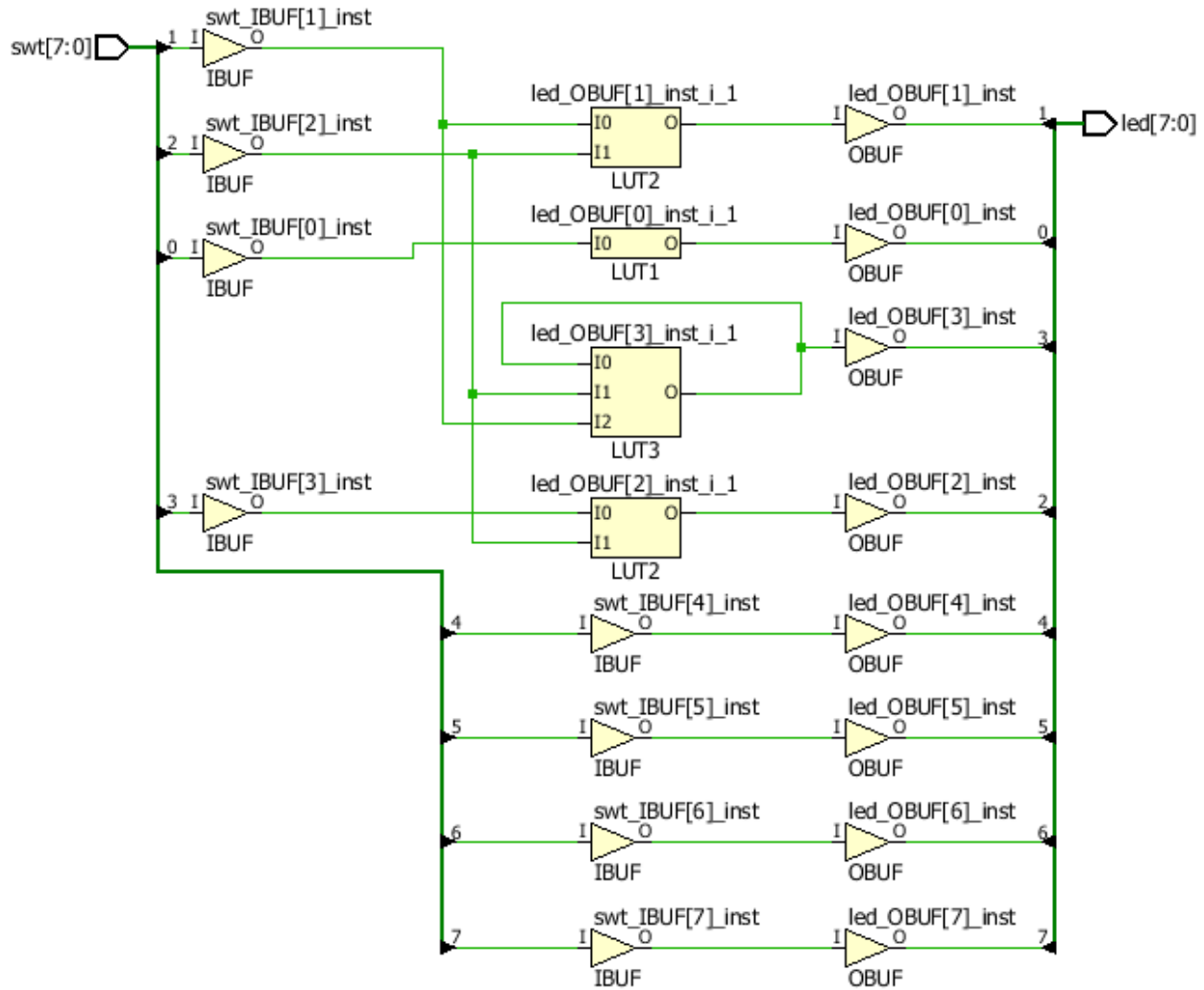


Figure 22. Synthesized design’s schematic view

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 us listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). Five 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 along with the Vivado Checkpoint File (**tutorial.dcp**).



Figure 23. 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.** Select **tutorial** under the *Design Sources* group, and 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.

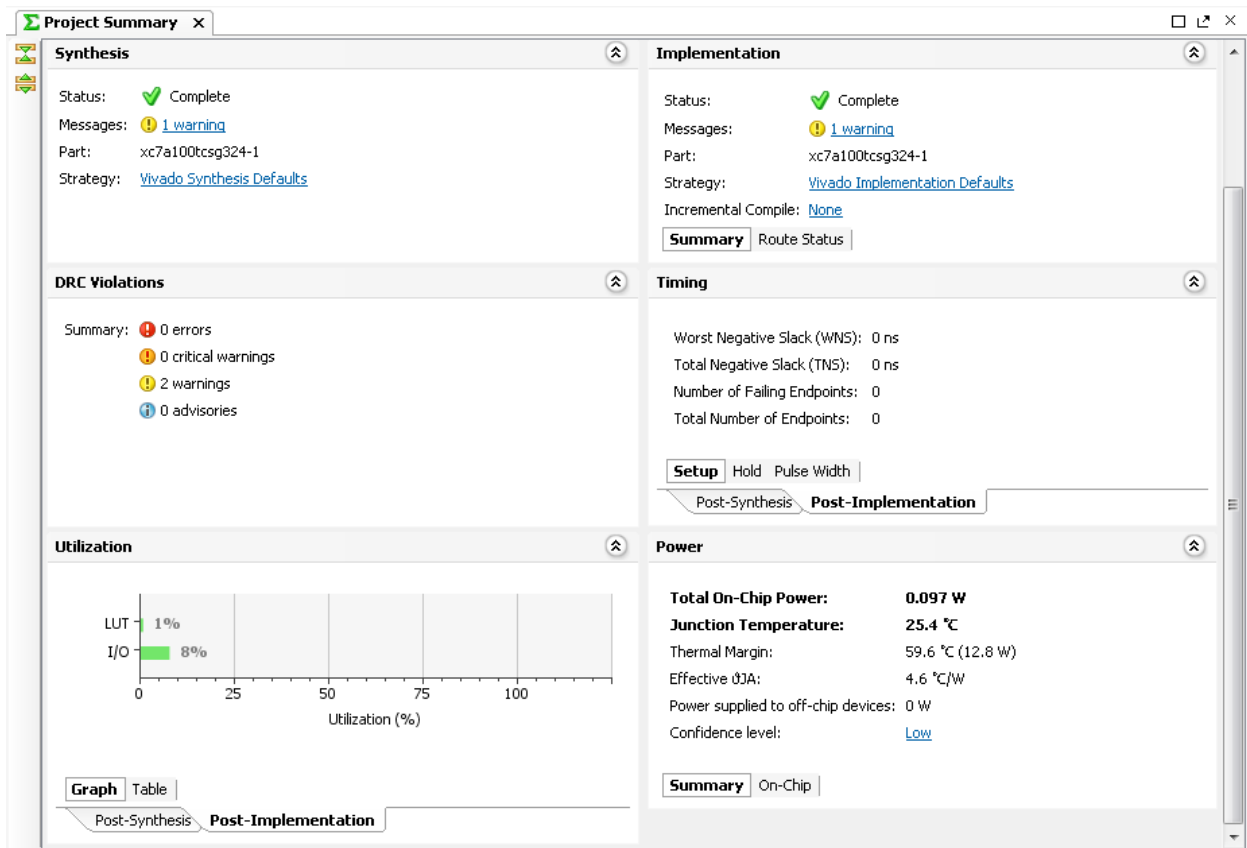
- 4-1-2.** Select **Open Implemented Design** and click **OK** as we want to look at the implementation output before progressing to the bitstream generation stage.

- 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.** Select the **Project Summary** tab and observe the results.

Notice that the resource utilization is shown in the Graph form as that was the last type of view used to see the synthesis output. To see the actual resource utilization in the tabular form, select Show Table. Note that three LUTs and 16 IOs. Also, since there were no timing constraints defined for this design, timing analysis could not be performed.



**Figure 24. 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 various report files (\*.RPT).

## Perform Timing Simulation

## Step 5

### 5-1. Expand Implemented Design tasks group and run the timing simulation.

5-1-1. Click on **Run Simulation** under *Simulation* in the *Flow Navigator* pane.

5-1-2. Click on the **Run Post-Implementation Timing Simulation** process to run simulation on the implemented design.

The simulator windows will appear as seen during behavioral simulation.

Using the Windows Explorer, verify that **impl** directory is created under **sim\_1** which is under **tutorial.sim** directory. The **impl** 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 1000 ns.

5-1-4. Left click at 50 ns (where the switch input is set to 0000000b. Click on the **Add Marker** button.

5-1-5. Drag the marker to where **leds** change (at the 110 ns mark).

5-1-6. Click on the **Add Marker** button again and move to where **led\_exp\_out** changes (120 ns).

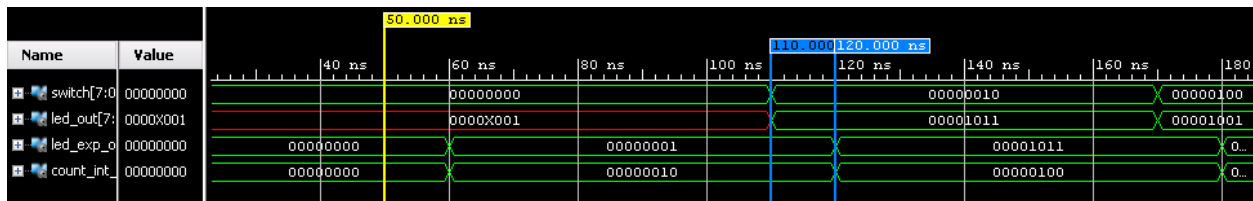


Figure 25. Timing simulation output

Notice that we monitored the expected led output at 10 ns after the input is changed (see the testbench)..

5-1-7. Close the simulator by selecting **File > Exit** without saving any changes.

## Generate the Bitstream and Verify Functionality

## Step 6

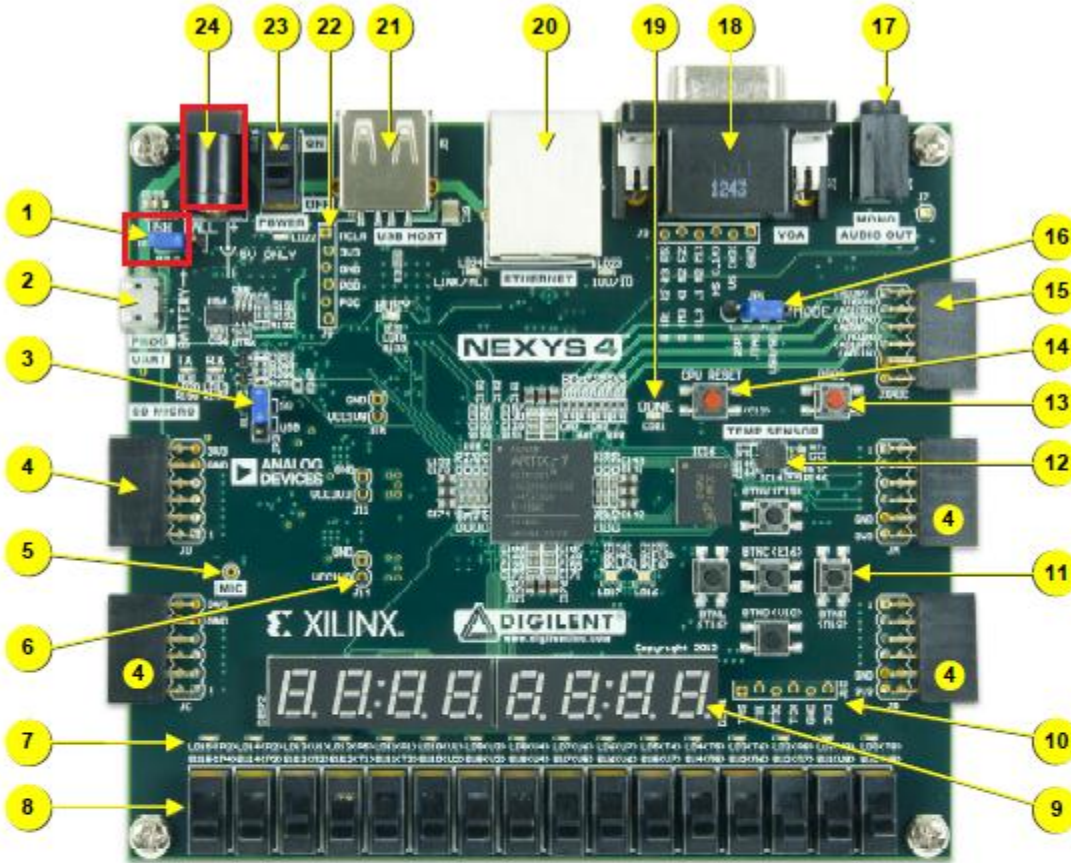
### 6-1. Connect the board and power it ON. Generate and download the bitstream.

6-1-1. Select **tutorial** under the *Design Sources* group, and click on the **Generate Bitstream** entry under the *Program and Debug* tasks of the *Flow Navigator* pane.

The bitstream generation process will be run on the implemented design. When the process is completed a *Bitstream Generation Completed* dialog box with two options will be displayed. Click **Cancel** to close the box.

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 jumpered 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.



Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Adept USB (JTAG and UART)	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

**Figure 26. Board settings**

**6-1-3.** Power **ON** the switch on the board.

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

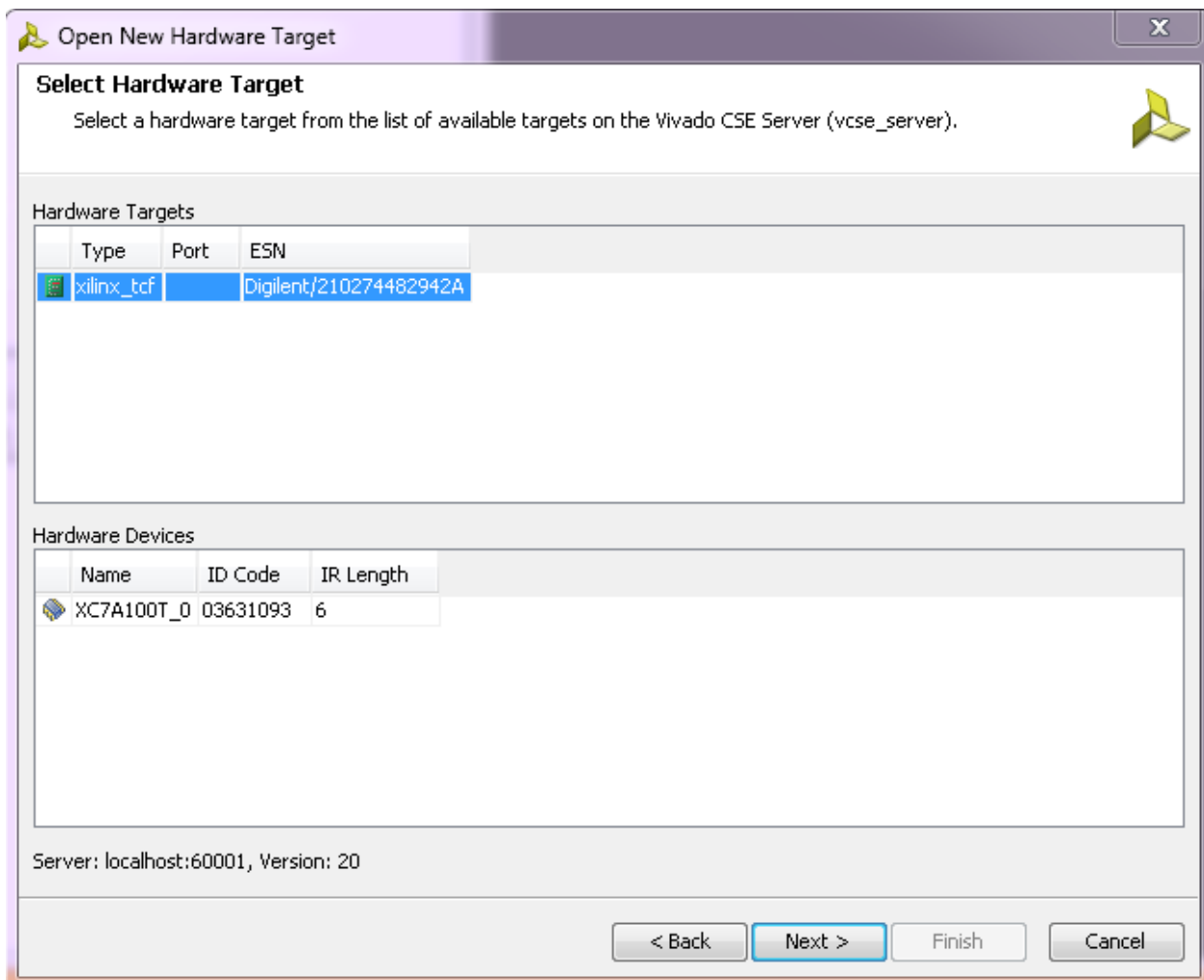
2013.32013.3 Select in the green information bar **Open a new hardware target**. The **Open New Hardware Target** wizard will be launched. Click **Next**.

 No hardware target is open. [Open recent target](#) [Open a new hardware target](#)

**Figure 27. Opening a hardware session**

**6-1-5.** Click **Next**, accepting the default *localhost:60001* setting. A progress bar showing the connection to the local host server will increment to completion, after which the target hardware will be identified.

**6-1-6.** Click **Next** to select the only device identified for the Nexys4 board.



**Figure 28. Selecting the FPGA to program**

**6-1-7.** Click **Next** to select the default target frequency of (15000000 Hz).

**6-1-8.** Verify the **Target Settings** in the summary window and click **Finish**.

**6-1-9.** Notice the green information bar at the top indicating there are no debug cores. Click on **Program Device** to program the FPGA.

**6-1-10.** Click **Program** at the *Program Device* window to configure the Artix-7 FPGA with the *tutorial.bit* file.

The *DONE LED* should light up after the device is configured. *LED 0* and *LED 3* will be lit by default. Toggle the switches to see the design in action (corresponding LEDs turning on or off depending on the switch settings).

**6-1-11.** When satisfied, power **OFF** the board.

**6-1-12.** Close the **Vivado** program by selecting **File > Exit**.

## Run the Tools in Batch Mode using Tcl Scripts

## Step 7

**7-1.** Open a Vivado 2013.3 Tcl Shell window, Change the directory to **c:\xup\digital\sources\tutorial** (using **cd: /xup/digital/sources/tutorial** command). Run the tools in a batch mode using the provided tcl script file by executing the following command.

```
Source tutorial_tcl_with_sim.tcl
```

**7-1-1.** Select **Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3 Tcl Shell** to open the Tcl shell.

**7-1-2.** In the Tcl prompt window, change the working directory to **c:\xup\digital\sources\tutorial** by executing the following command.

```
cd c:/xup/digital/sources/tutorial
```

**7-1-3.** Run the provided Tcl script in the batch mode by executing the following command.

```
Source tool_tcl_with_sim.tcl
```

The tools will be run and various directories will be created. 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
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
24

```

**Figure 29. 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.

**7-1-4.** Close the Tcl Shell window when the execution is 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.** Go to the `tutorial_tcl_with_sim.sim > sim_1 > behave` directory and view the content of the `tutorial_tb_behav.log` file. Note that it contains the simulator output.

**7-1-7.** 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 functionality was verified in hardware using the generated bitstream.