

PlanAhead Tutorial

Introduction

This tutorial guides you through the design flow using Xilinx PlanAhead 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 PlanAhead 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 Spartan-6 based Nexys3 board. The typical design flow is shown below. The circled number indicates the corresponding step in this tutorial.

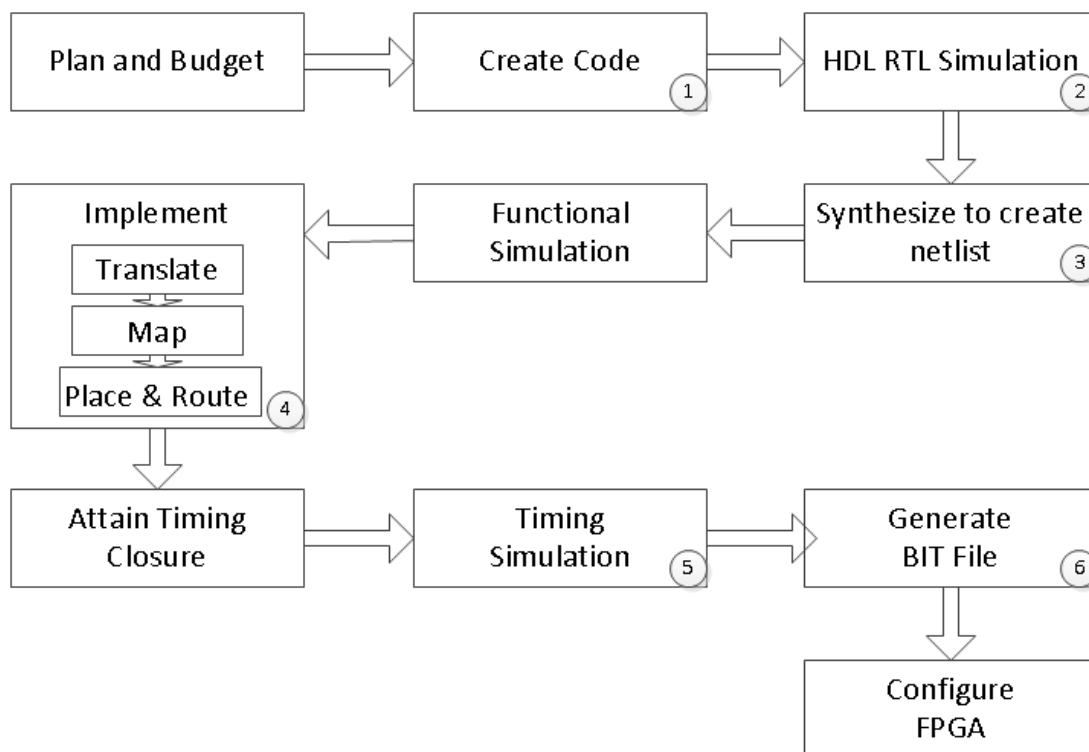


Figure 1. A typical design flow

Objectives

After completing this tutorial, you will be able to:

- Create a PlanAhead project sourcing HDL model(s) and targeting a specific FPGA device located on the Nexys3 board
- Use the provided user constraint file (UCF) to constrain pin locations
- Simulate the design using the ISim simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the Spartan-6 FPGA with the generated bitstream 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**.

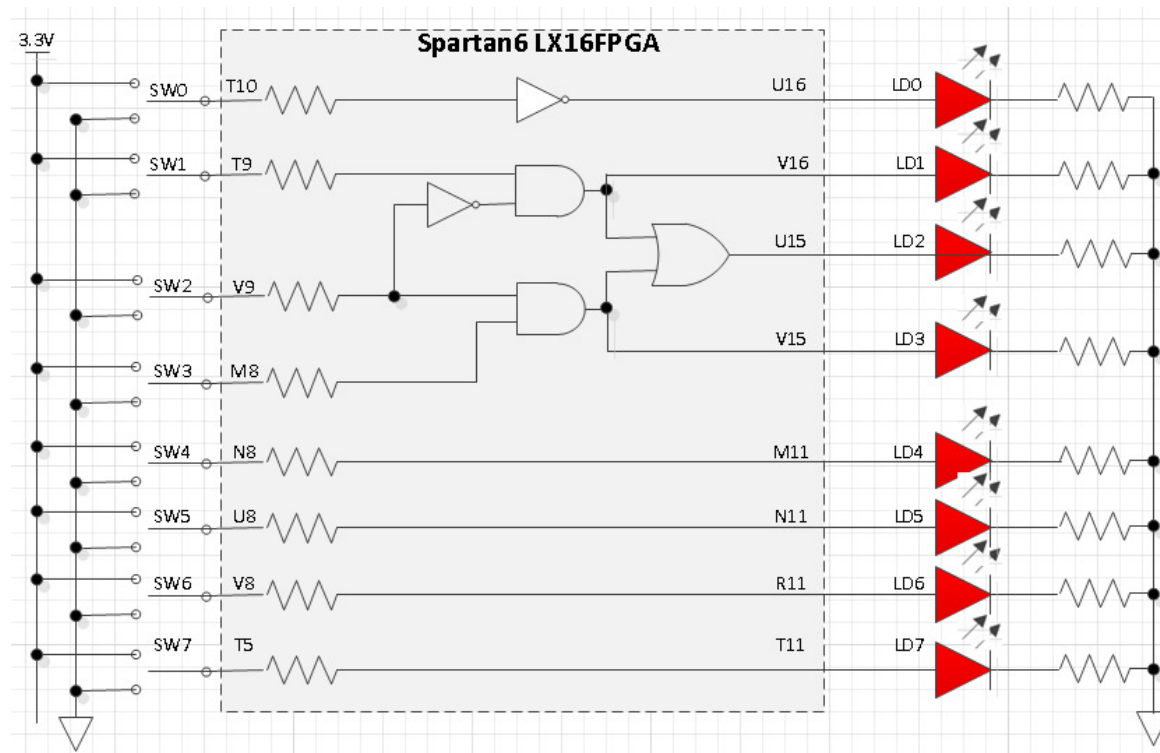


Figure 2. Completed Design

General Flow for this tutorial

- Create a PlanAhead project and analyze source files
- Simulate the design using ISim simulator
- Synthesize the design
- Implement the design
- Perform the timing simulation
- Verify the functionality in hardware using Nexys3

Create a PlanAhead Project

Step 1

1-1. Launch PlanAhead and create a project targeting the xc6slx16csg324-2 device and using the Verilog HDL. Use the provided tutorial.v and tutorial.ucf files in the *sources* directory.

1-1-1. Open PlanAhead by selecting **Start > All Programs > Xilinx Design Tools > ISE Design Suite 14.6 > PlanAhead > PlanAhead**.

1-1-2. Click **Create New Project** to start the wizard. You will see *Create A New PlanAhead 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**.

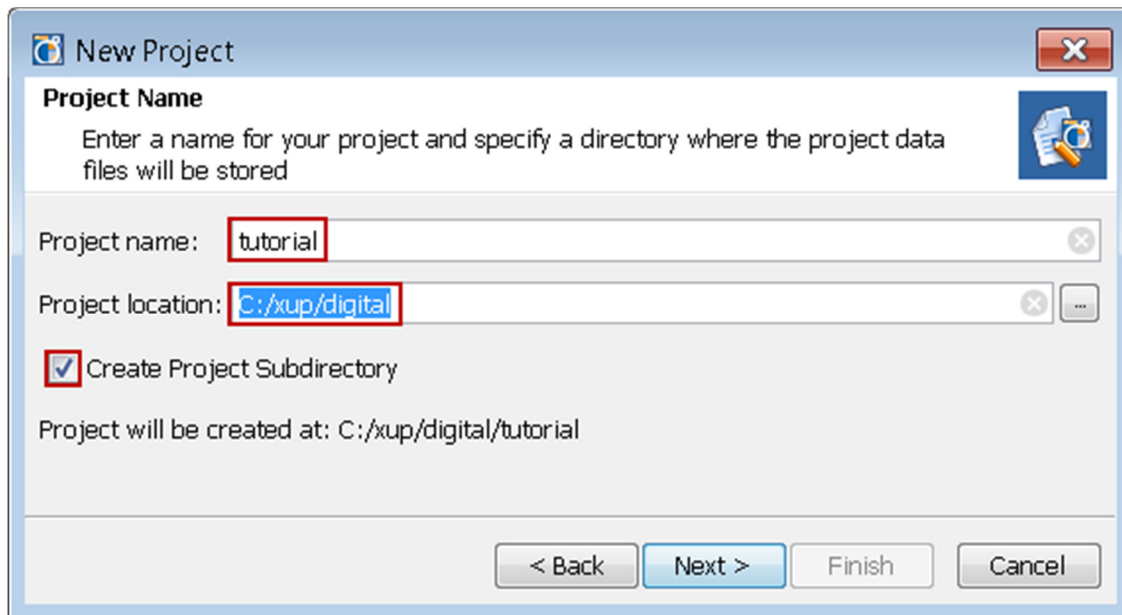


Figure 3. Project Name and Location entry

- 1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.
 - 1-1-6. Select **Verilog** as the *Target Language* in the *Add Sources* form.
 - 1-1-7. Click on the **Add Files...** button, browse to the **c:\xup\digital\sources\tutorial** directory or where the file was extracted, select *tutorial.v*, click **OK**, and then click **Next**.
 - 1-1-8. Click **Next** to get to the *Add Constraints* form.
 - 1-1-9. Click on the **Add Files...** button, browse to the **c:\xup\digital\sources\tutorial** directory or where the file was extracted, select *tutorial.ucf* click **OK**, and then click **Next**.
- The user 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 **xc6slx16csg324-2** part. Click **Next**.

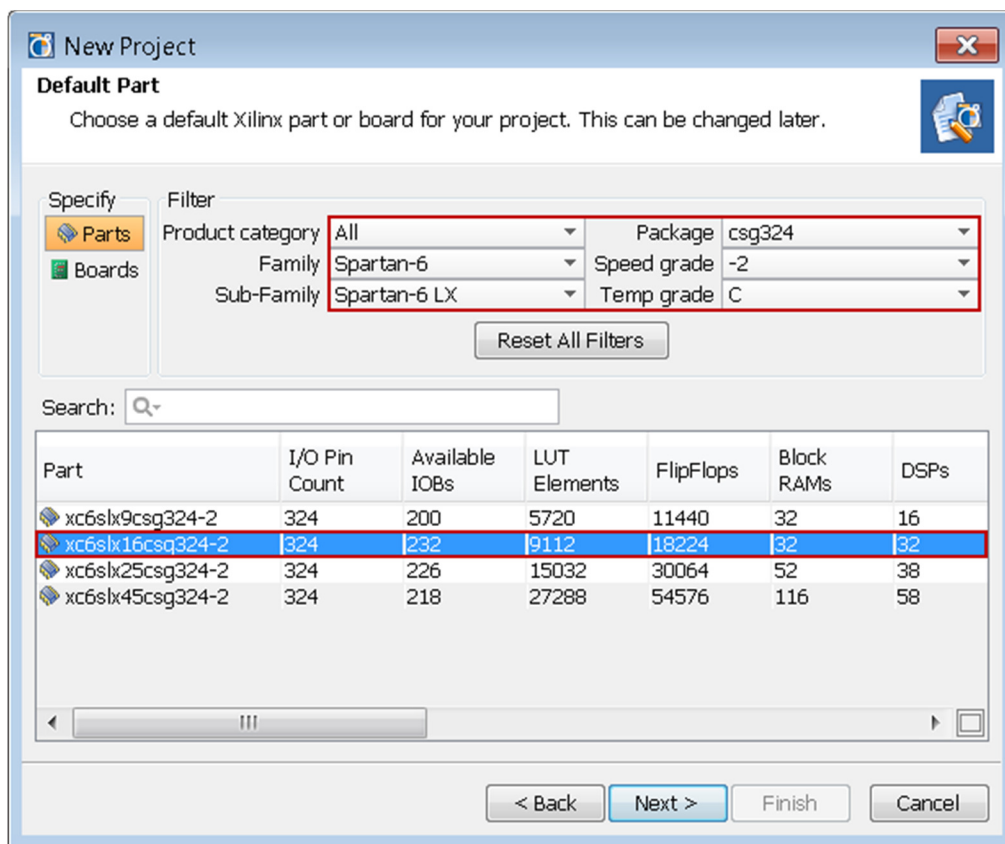


Figure 4. Part Selection

1-1-11. Click **Finish** to create the PlanAhead 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.ppr** (PlanAhead) project file have been created. The **tutorial.data** directory is a place holder for the PlanAhead program database. Two more directories, **constrs_1** and **sources_1**, are created under the **tutorial.srcs** directory; deep down under them, the copied **tutorial.ucf** (constraint) and **tutorial.v** (source) files respectively are placed.

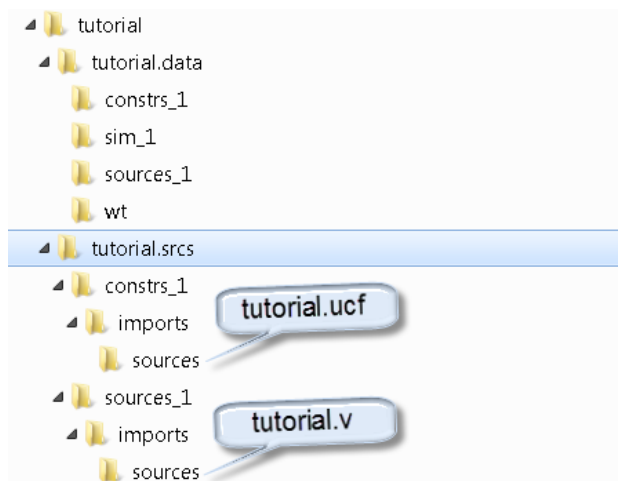


Figure 5. Generated directory structure

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.

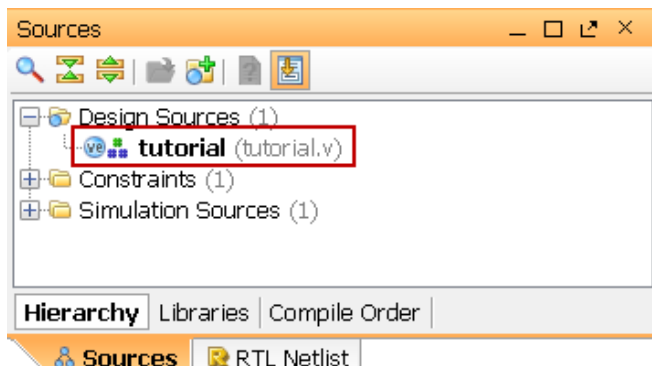


Figure 6. Opening the source file

1-2-2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. Lines 2-5 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.ucf source and analyze the content.

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

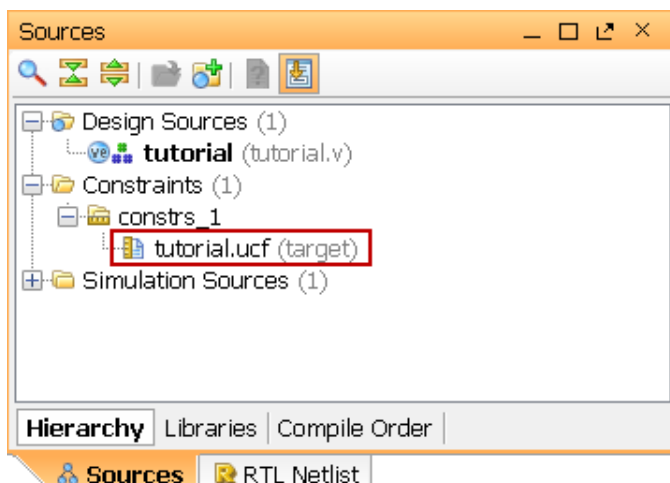


Figure 7. Opening the constraint file

1-3-2. Lines 4-10 defines the pin locations of the input switches [6:0] and lines 12-18 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, expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane, and click on **Schematic**.

A logic view of the design is displayed.

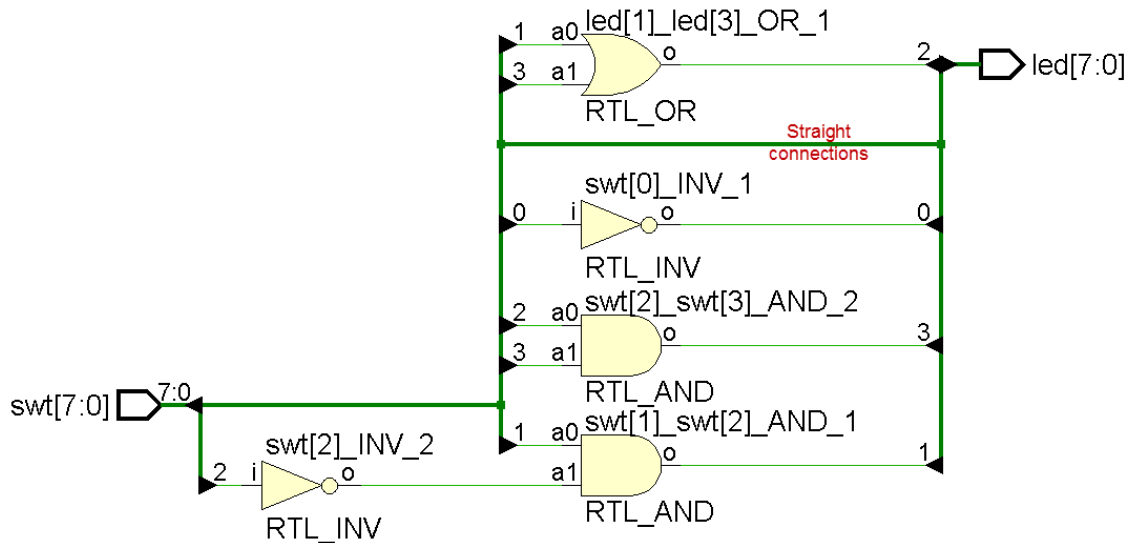


Figure 8. A logic view of the design

Notice that some of the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs 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 layout, called the I/O Planning, is available. Click on the drop-down button and select the I/O Planning layout.

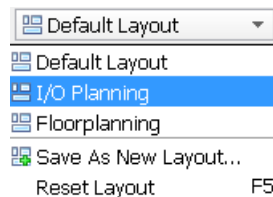


Figure 9. I/O Planning layout selection

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

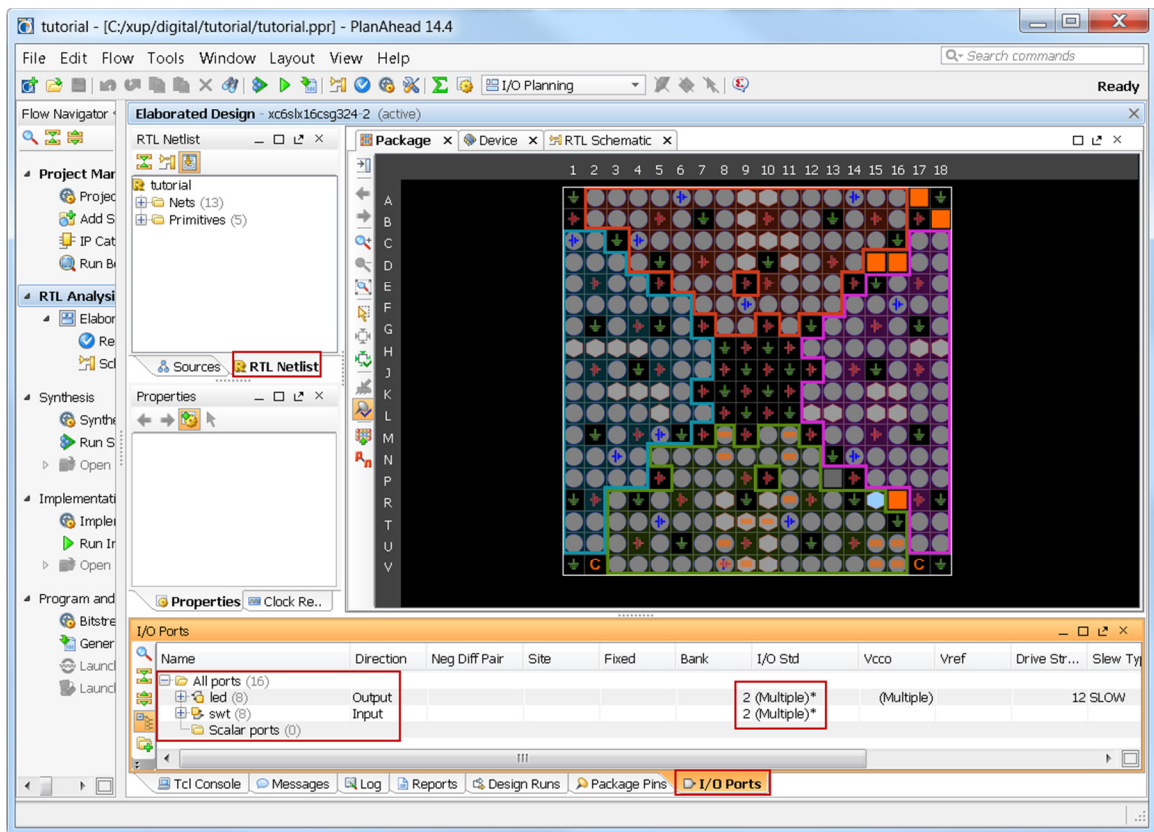


Figure 10. I/O Planning layout view

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 LVCMOS33 I/O standard whereas led[7] and swt[7] do not have assigned pins and defaults to LVCMOS25; hence you can see multiple I/O standard in the collapsed view.

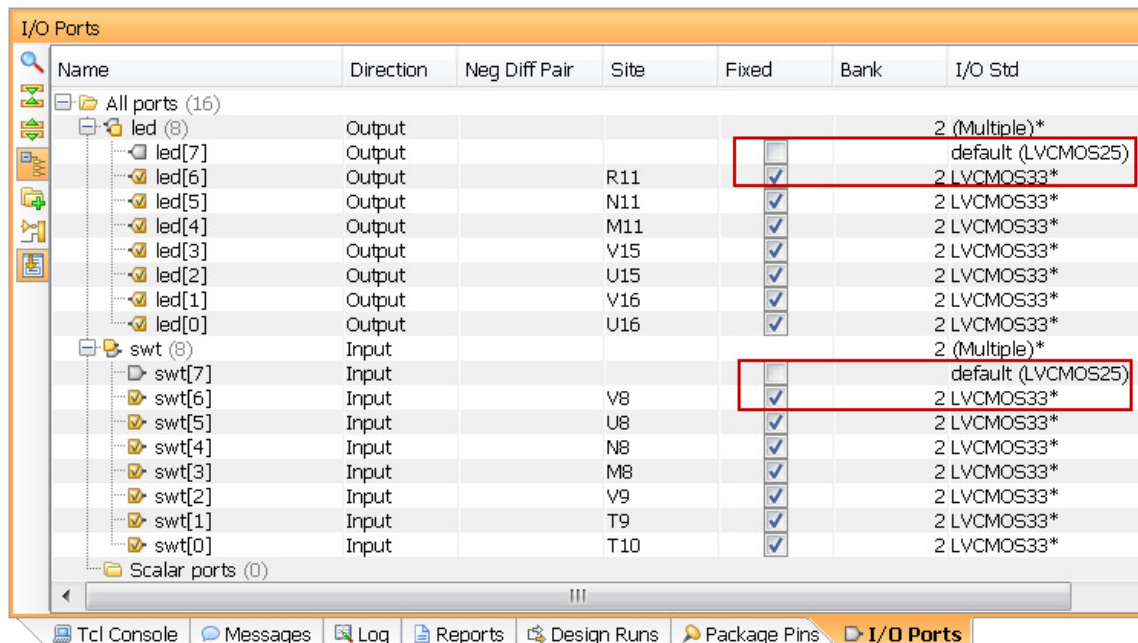


Figure 11. I/O Ports tab

- 1-5-3.** Click under the *Site* column across **led[7]** row to see a drop-down box appear. Type **T** in the field to jump to Txx pins, scroll-down until you see T11, then select T11, and hit the *Enter* key to assign the pin.

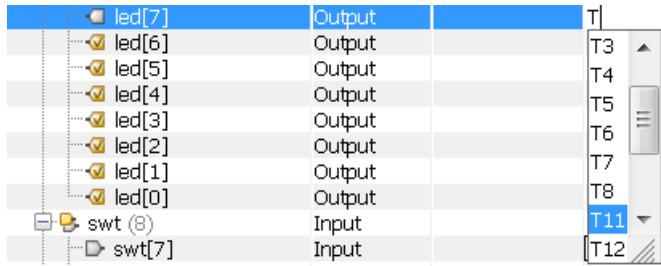


Figure 12. Assigning pin

- 1-5-4.** Similarly, click under the *I/O Std* column across the **led[7]** row and select **LVCMOS33**.
- 1-5-5.** Similarly, assign the **T5** pin location and the **LVCMOS33** I/O standard to **swt[7]**.

You can also 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 **T5** 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, and clicking on the **Apply** button.

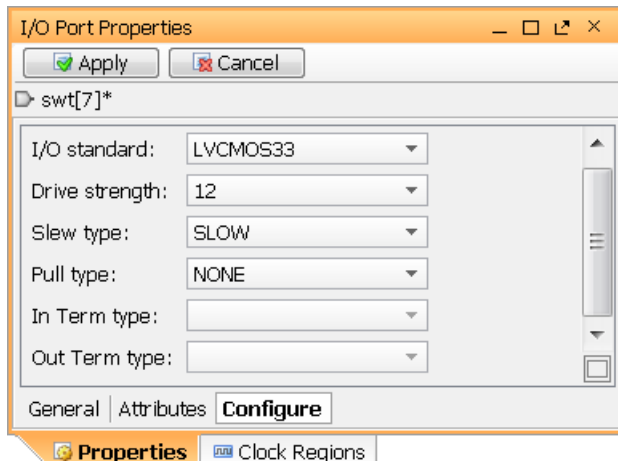


Figure 13. Assigning I/O standard through the I/O Port Properties form

- 1-5-6.** Select **File > Save Constraints** and click **OK** to save the constraints in the **tutorial.ucf** file.

Note that the constraints are updated in the tutorial.ucf 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.v testbench file.

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

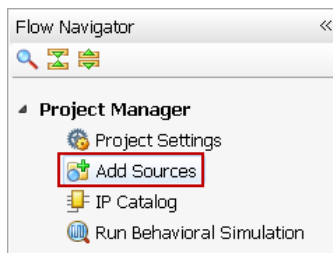


Figure 14. Add Sources

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

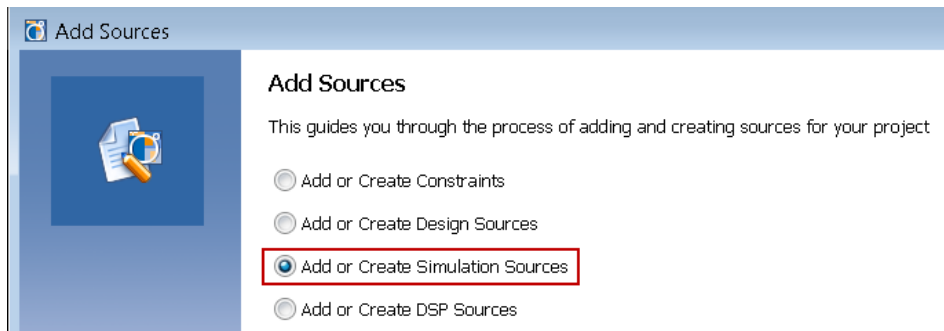


Figure 15. Selecting Simulation Sources option

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

2-1-4. Browse to the `c:\xup\digital\sources\tutorial` folder or where the file was extracted, and select `tutorial_tb.v` and click **OK**.

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

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

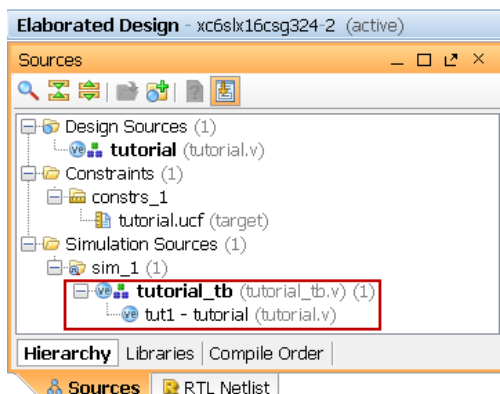


Figure 16. Simulation Sources hierarchy

2-1-6. 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.v` is placed.

2-1-7. Double-click on the `tutorial_tb` 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 ISim simulator.

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

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 **200 ns**.

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

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

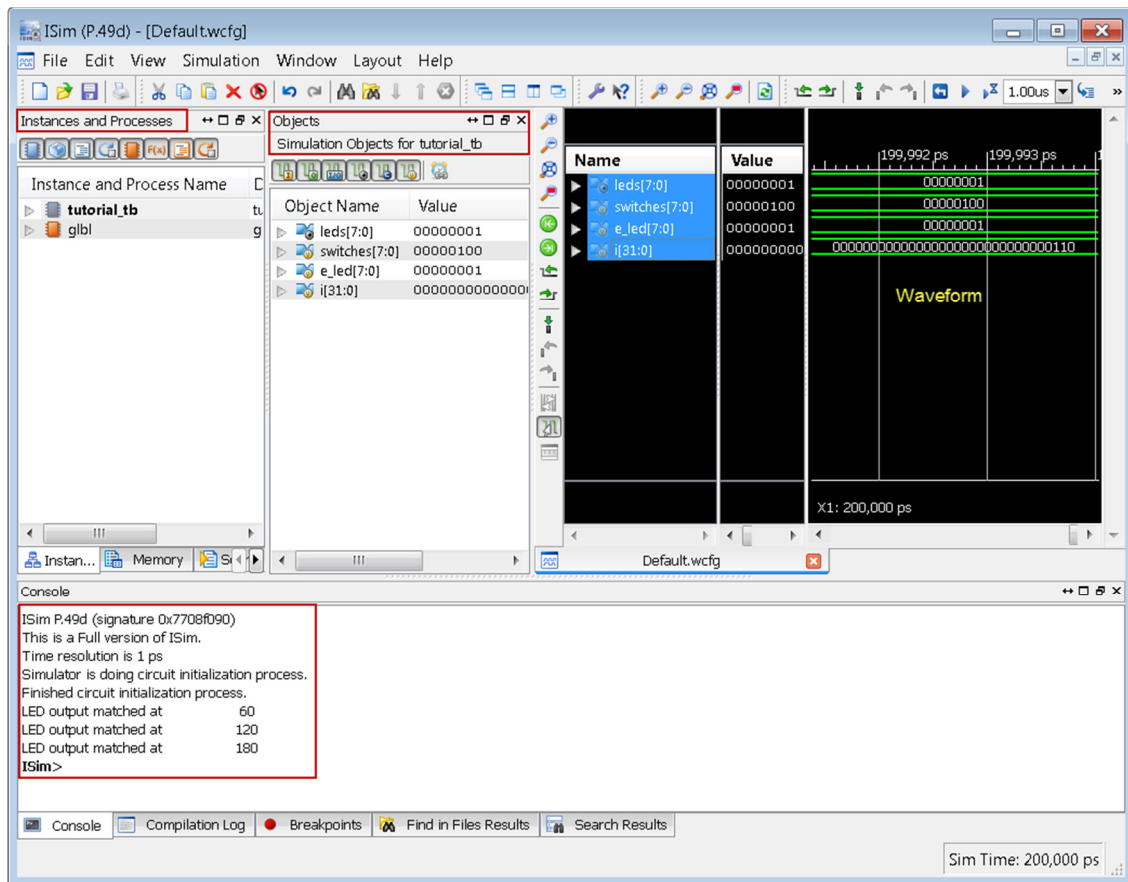


Figure 18. Simulator output

You will see four main views: (i) *Instances and Processes*, where the testbench hierarchy (in collapsed form) as well as gbl instances are displayed, (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. The executable simulator file, **tutorial_tb** is created under the **tutorial_tb.exe.sim** directory which is executed to run the simulation.

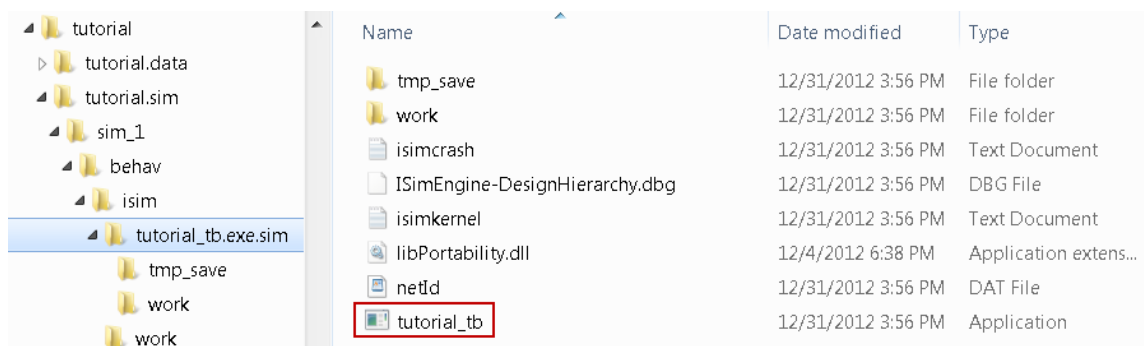



Figure 19. Directory structure after running behavioral simulation

Please refer to the following URL

http://www.xilinx.com/support/documentation/sw_manuels/xilinx14_4/plugin_ism.pdf to learn more about the ISim simulator.

- 2-2-5.** Click on the full-zoom 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.** 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.** Click on the glbl instance in the *Instances and Processes* window to see the global signals, in the *Objects* window available to the system. Do not add these signals.

- 2-4-2.** Expand the **tutorial_tb** instance in the *Instances and Processes* window and select the **tut1** instance.

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

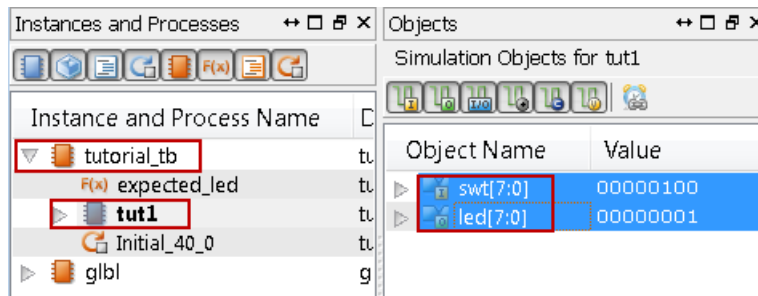



Figure 20. Selecting lower-level signals

- 2-4-3.** 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.

- 2-4-4.** In the waveform window, select 1.00 us and type over 500 ns () if we want to run for 500 ns (total of 700 ns), and hit enter.

The simulation will run for an additional 500 ns.

- 2-4-5.** Click on the *Zoom Full* button and notice that **swt[7:0]** is updated from 200 ns to 700 ns period. This is because we added an additional 500 ns from the previous step.

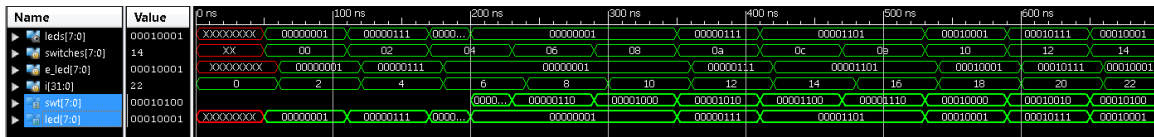


Figure 21. Running simulation for additional 500 ns

2-4-6. Close the simulator by selecting **File > Exit** in the ISim window without saving the waveform.

Synthesize the Design

Step 3

3-1. Synthesize the design with the XST 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 (the top-level module 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 *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. Click **Yes** to close the elaborated design.

3-1-4. Select the **Project Summary** tab and understand various windows

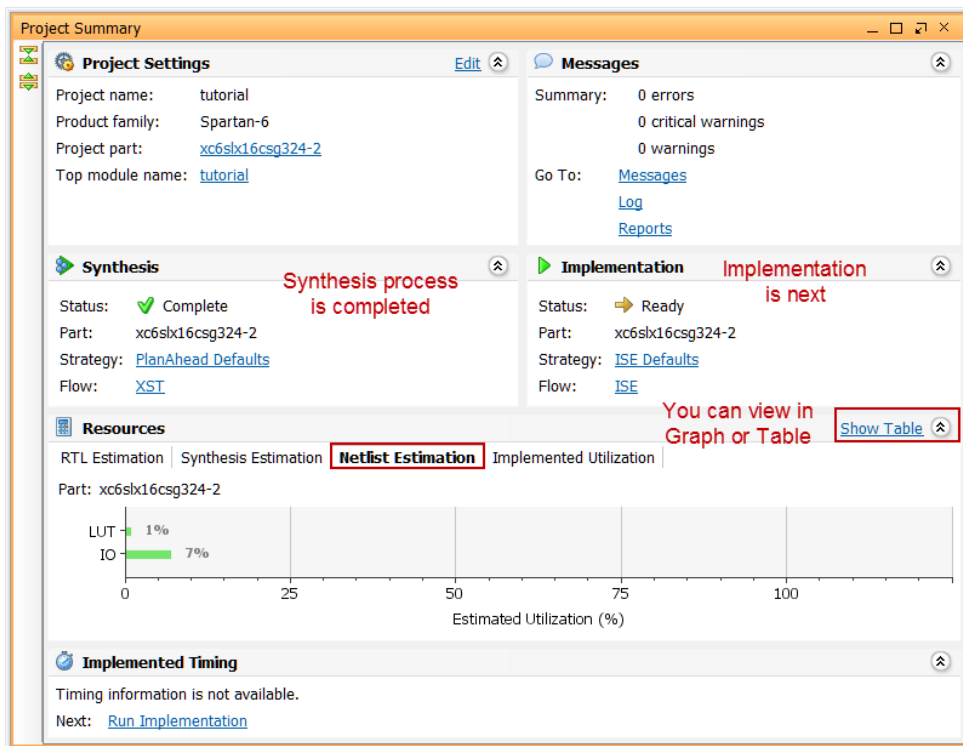


Figure 22. Project Summary view

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

3-1-5. Click on the **Show Table** link in the **Project Summary** tab.

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

Resources			
RTL Estimation Synthesis Estimation Netlist Estimation Implemented Utilization			
Part: xc6slx16csg324-2			
Resource	Estimation	Available	Utilization
LUT	4	27336	1%
IO	16	232	7%

Figure 23. Resource utilization estimation summary

3-1-6. Click on **Schematic** under the *Synthesized Design* tasks of *Synthesis* tasks of the *Flow Navigator* pane to view the synthesized design in schematic view.

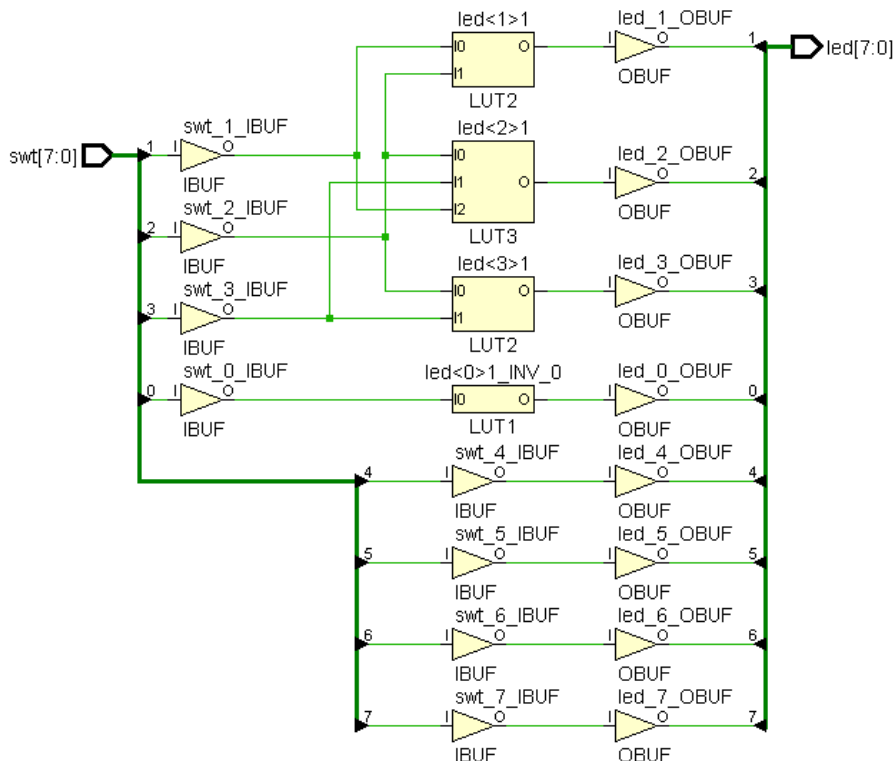


Figure 24. Synthesized design's schematic view

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). 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 synthesized netlist file (**tutorial.ngc**).

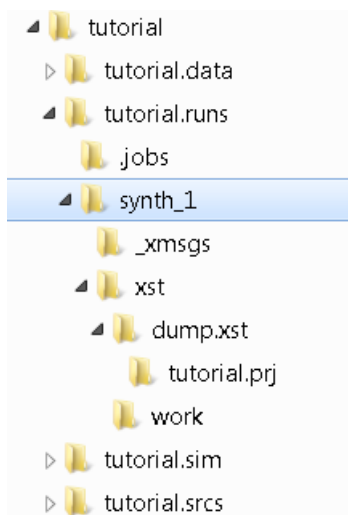


Figure 25. Directory structure after synthesizing the design

Implement the Design

Step 4

4-1. Implement the design with ISE 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 a *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 Table form as that was the last type of view used to see the synthesis output. Note that two LUTs, one slice (in which the two LUTs have been packed), and 16 IOs. Also, it indicates that no timing constraints were defined for this design.

Project Summary

Project Settings

Project name: tutorial
 Product family: Spartan-6
 Project part: xc6slx16csg324-2
 Top module name: tutorial

Messages

Summary: 0 errors
 0 critical warnings
 0 warnings
 Go To: [Messages](#)
[Log](#)
[Reports](#)

Synthesis

Status: ✔ Complete
 Part: xc6slx16csg324-2
 Strategy: [PlanAhead Defaults](#)
 Flow: [XST](#)

Implementation

Status: ✔ Complete
 Part: xc6slx16csg324-2
 Strategy: [ISE Defaults](#)
 Flow: [ISE](#)

Resources

RTL Estimation | Synthesis Estimation | Netlist Estimation | **Implemented Utilization** [Show Graph](#)

Part: xc6slx16csg324-2

Resource	Utilization	Available	Utilization
LUT	2	9112	1%
Slice	1	2278	1%
IO	16	232	6%

Implemented Timing

Timing constraints were not defined for this implementation.
 Go to: [Timing Constraints](#)

Figure 26. 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 **tutorial.ncd**, **tutorial.mrp**, **tutorial.map** files.

Perform Timing Simulation

Step 5

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

5-1-1. Expand **Implemented Design** under the *Implementation* tasks of the *Flow Navigator* pane, if necessary, and click on the **Run Timing Simulation** process to run simulation on the implemented design.

A *Launch Timing Simulation* dialog box will appear with Simulation Top Module as **tutorial_tb**.

5-1-2. Click **Launch** to run the simulation.

Click **OK** to close the warning message, if appears, indicating that no timing constraints were found.

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 to Full View** button to see the waveform window from 0 to 200 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 (around the 57.198 ns).
- 5-1-6. Click on the **Add Marker** button again and move to where **e_led** changes (60 ns).

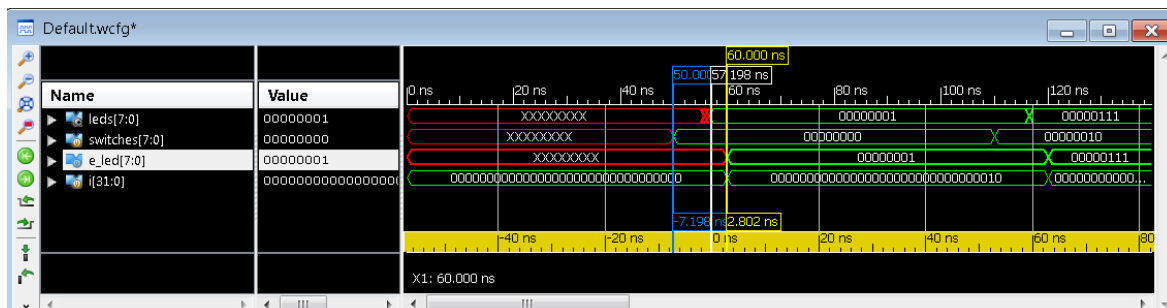


Figure 27. Timing simulation output

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 7.198 ns.

- 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. 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.

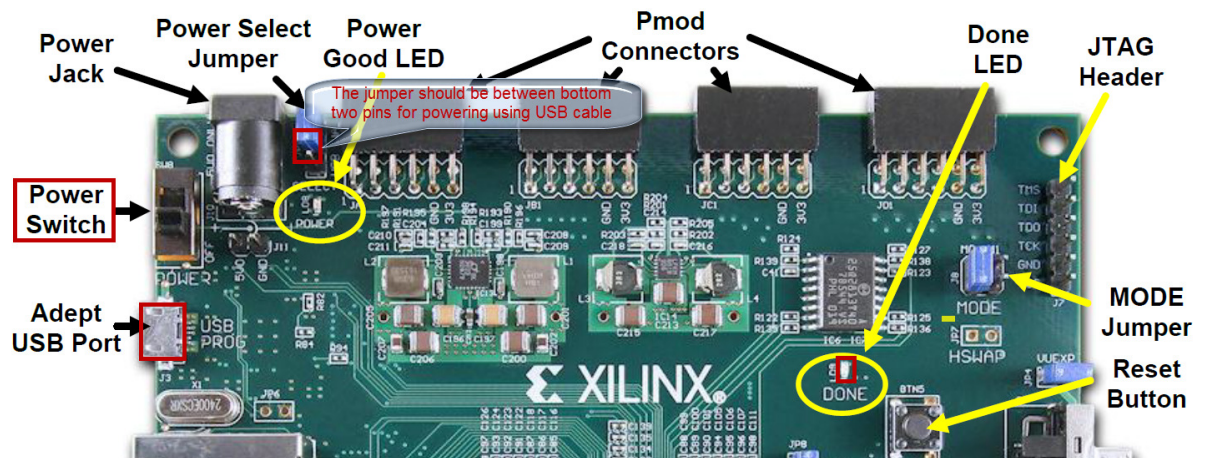


Figure 28. Board settings

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

6-1-4. Select the *Launch iMPACT* option and click **OK**.

The iMPACT program will be launched, the JTAG chain will be scanned and the xc6slx16 device should be detected, the generated **tutorial.bit** file will then be assigned, and *Save Project File* dialog will be opened.

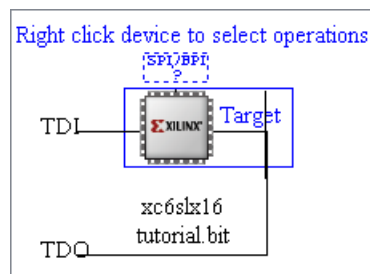


Figure 29. FPGA device is assigned the bit file

6-1-5. Click **Cancel** to close the dialog box.

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

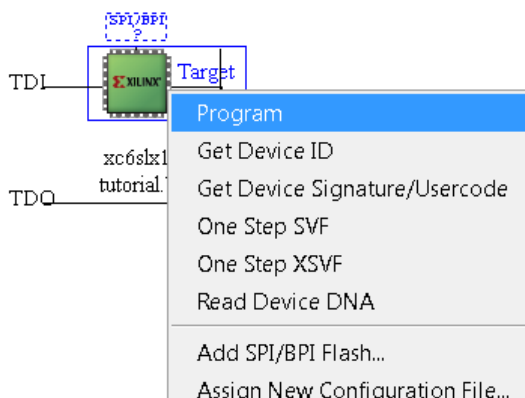


Figure 30. Selecting to program the FPGA

6-1-7. Click **OK** 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.

6-1-8. Verify the functionality by flipping switches and observing the output on the LEDs.

6-1-9. When satisfied, power **OFF** the board.

6-1-10. Close the **IMPACT** program by selecting **File > Exit**.

6-1-11. Close the **PlanAhead** program by selecting **File > Exit**.

Conclusion

The PlanAhead software tool can be used to perform a complete FPGA 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.