ECE532S Digital Systems Design

Tutorial 1 - Vivado Project Flow Basics

Last Updated: July, 2019

This tutorial guides you through the design flow of the 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 in Figure 1.

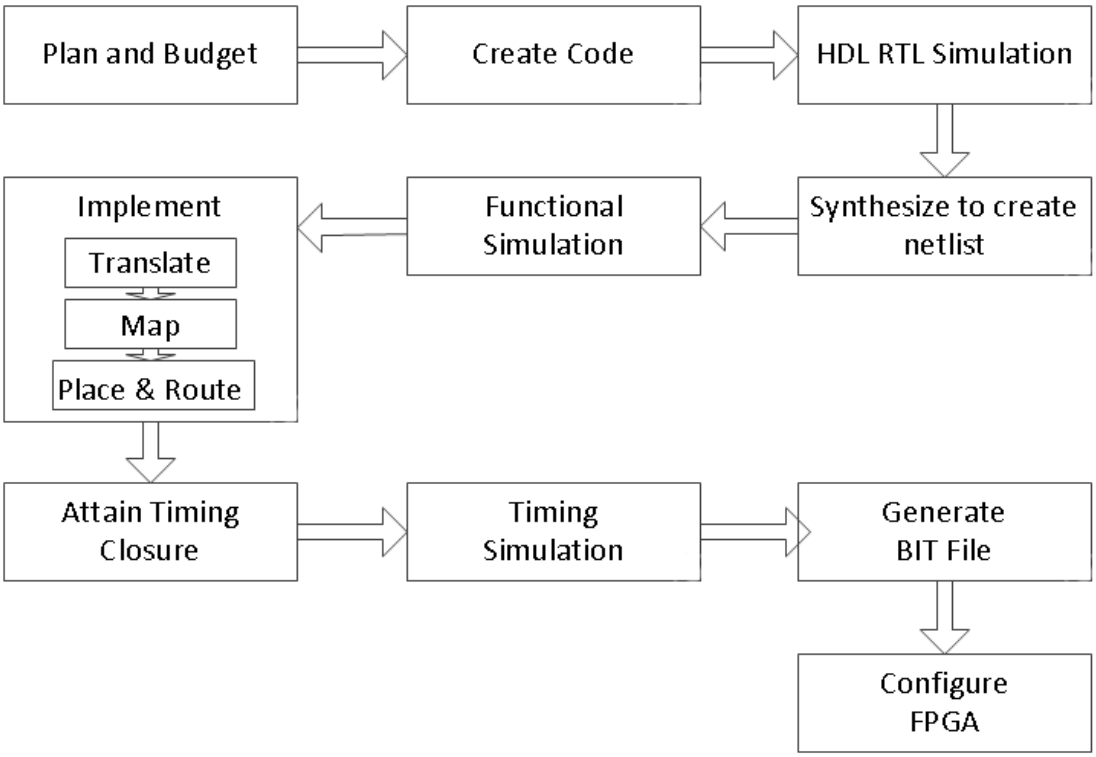

Figure 1: A typical Vivado design flow

1 Creating a Vivado Project

Launch Vivado in order to create a new project. Note, if launching Vivado from a terminal, you may need to source a script located at:

```
<Vivado Install Directory>/Xilinx/Vivado/<Vivado Version>/settings64.sh
```

For the UG machines for example, the script will be located in the following location:

```
/cad1/Xilinx/Vivado/2018.1/settings.sh
```

To *source* the script file, type the following command into the terminal and hit enter:

```
source /cad1/Xilinx/Vivado/2018.1/settings.sh
```

After sourcing the script, launch Vivado by typing vivado into the terminal:

```
vivado
```

If you're launching Vivado from a Windows machine, you should be able to find Vivado from the Start Menu and don't need to launch Vivado from the terminal. Note, if you do want to launch Vivado from the Command Shell on a Windows machine (perhaps to launch Vivado using a provided TCL script), follow the above steps but note that the extension for the script file will be *.bat* rather than *.sh*.

```
<Vivado Install Directory>\Xilinx\Vivado\<Vivado Version>\settings64.bat
```

Once the Vivado GUI has opened, select **Create New Project** in order to start the project creation wizard. You will see a *Create A New Vivado Project* dialog box; click **Next**. A zip file containing some necessary source files for this tutorial was distributed alongside this tutorial document, make sure the contents of that zip file are extracted to some directory in which you want to create your new project. In the *project location* field of the current dialog, browse to the directory to which you extracted the project files. Enter **tutorial** in the *project name* field and make sure that the *Create project subdirectory* box is checked; click **Next**. See Figure 2 for an example.

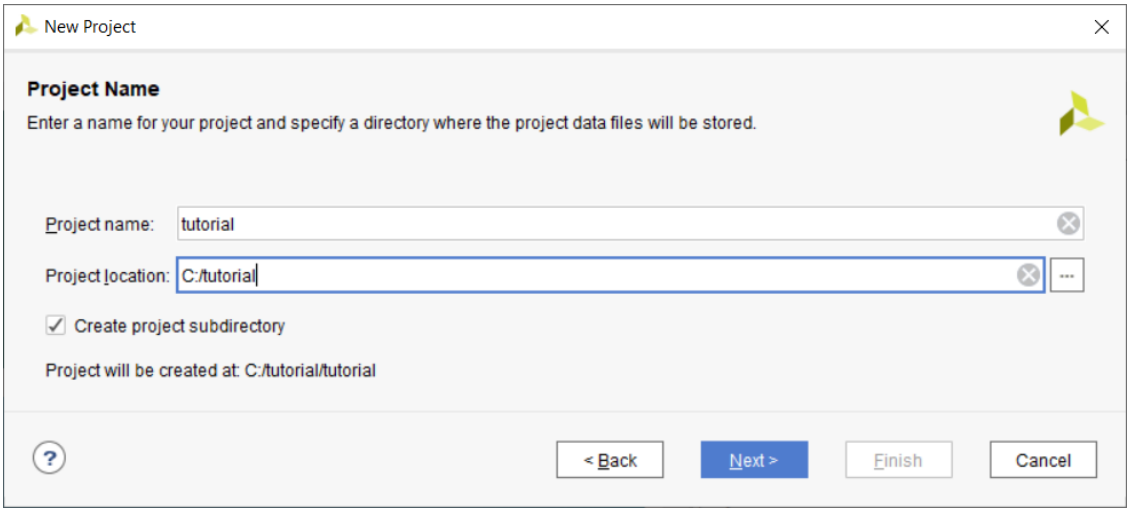

Figure 2: Project name and location entry

In the next window of the project creation wizard, Select the **RTL Project** option in the *Project Type* form, and click **Next**. Now we have the *Add Sources* window. On this page, we add any source files that we have already created to the project. Select **Verilog** as the *Target Language* and then

click on the **Add Files...** button. From here you can browse to the location of any source you wish to add. For this tutorial, browse to the extracted files and select *tutorial.v* and click **OK** to add that source to the project. Make sure to check the box labelled *Copy sources into project* before proceeding to the next pane. This check box option copies any of the selected source files into the newly created project, which means changes to the source within the project will not effect the original file and, conversely, changes to the original file will not be seen by the project. In future projects, keep in mind whether or not this is the behaviour you'd like for your project and select the option accordingly. click **Next** to move to the next page. See Figure 3 for an example.

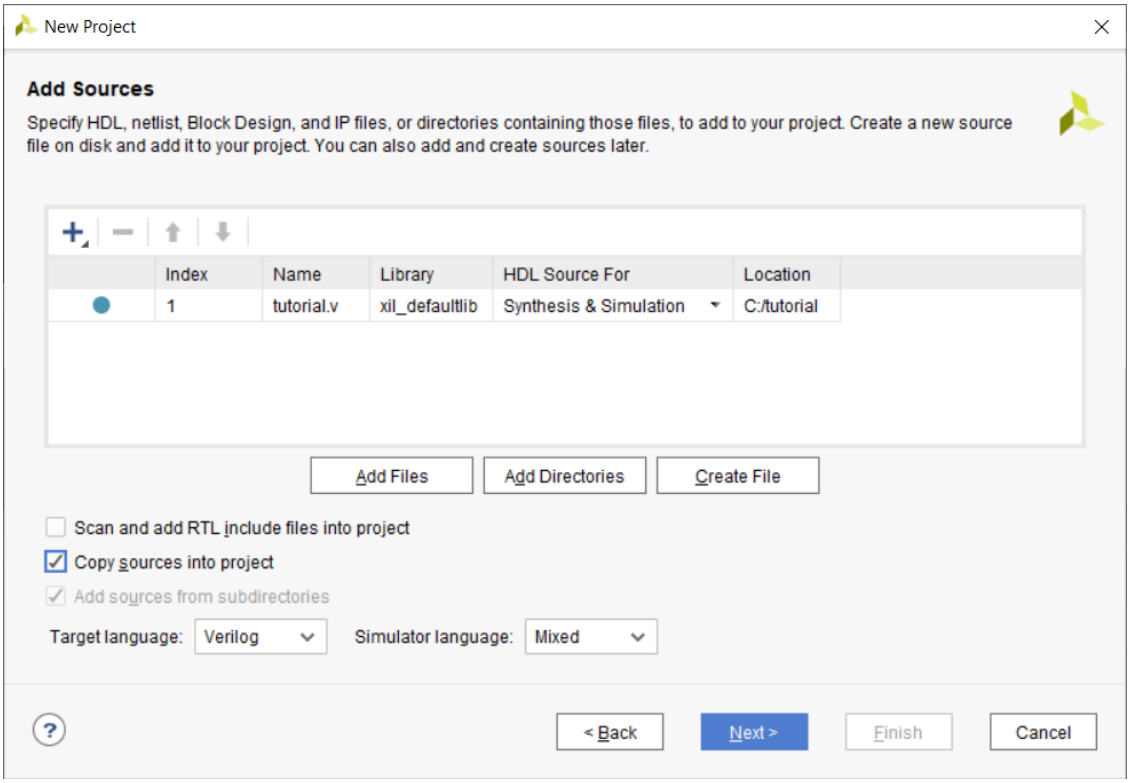

Figure 3: Project source selection window

Now we have the *Add Constraints* window. In FPGA projects, constraints are used to specify things like which signals ought to connect to specific pins on the FPGA, and the frequency of our external clock sources. For this tutorial, we've provided a constraints file (extension *.xdc*). Click on the **Add Files...** button and browse to the location of the extracted files. Select the *tutorial.xdc* file and click **Open** (if necessary). Ensure that *Copy constraints files into project* check box is selected (this option serves the same purpose as we discussed with the source files earlier). Click **Next** and proceed to *Default Part* selection.

In the *Default Part* form, using the **Parts** option and the various drop-down fields of the Filter section (or the search box), select the **XC7A100TCSG324-1** part. This corresponds to the FPGA part of the Nexsys DDR4 board (if you're using a Nexsys Video board, adjust this accordingly). See Figure 4 for an example. Note, while we used the *Parts* option to select our part, a *Boards* option is also available, which will select the appropriate part for the board you select in that list. The Nexsys boards aren't included in Xilinx's list of boards, though in later tutorials we'll go through how to add board files to the Vivado install such that Nexsys devices appear in the list. Note also that using the board option often allows us to avoid using a constraints file, since many of

the constraints are defined in the board files themselves. For this tutorial, we'll stick to the *Part* selection option. Click **Next** and then **Finish** in the last pane of the wizard to create the project.

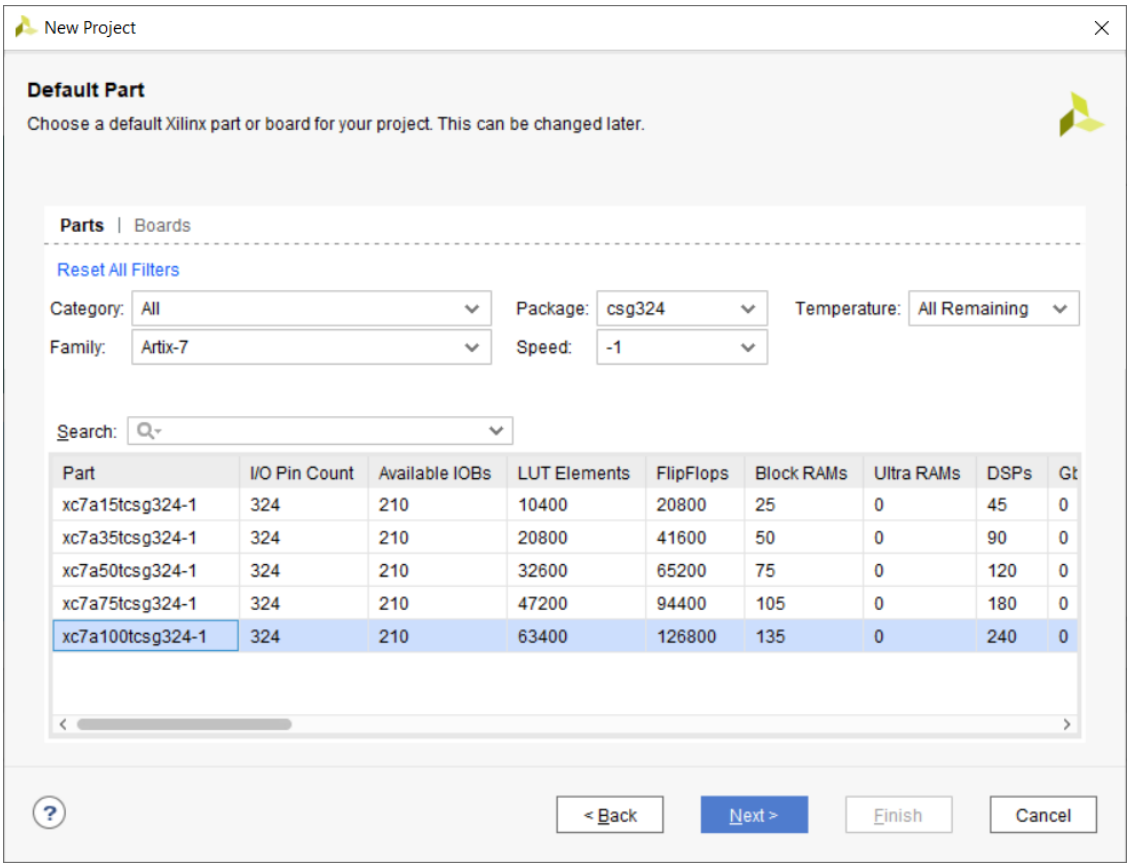

Figure 4: Default part selection window

2 Navigating the Vivado Project View

Once you have finished the project creation wizard, the Vivado *Project Manager* view should be open for your newly created project. There are a number of different panes within this view that expose different features of Vivado or different aspects of the project on which you are working. Figure 5 shows the Vivado *Project Manager* view with all of the panes labelled with a number (for ease of discussion here).

The first section, labelled with a **1**, is called the *Flow Navigator*. In the *Flow Navigator* you can find the various steps you would need in developing your project. For example, the *Flow Navigator* has subsections for *Simulation*, *Synthesis*, and *Implementation*. To the right of the *Flow Navigator* we have various panes that make up the *Project Manager* view. Note, the *Flow Navigator* has actions that may open up a new view in place of the *Project Manager* view. For example, the actions *Open Elaborated Design*, *Open Synthesized Design*, and *Open Implemented Design* will open the *Elaborated Design* view, the *Synthesized Design* view, and the *Implemented Design* view respectfully. These new views expose different information about the state of the project, and can be closed to return to the *Project Manager* view (Note, closing the *Project Manager* view will close the whole project).

The next section, labelled with a **2**, simply lists all of the sources included in the project. By

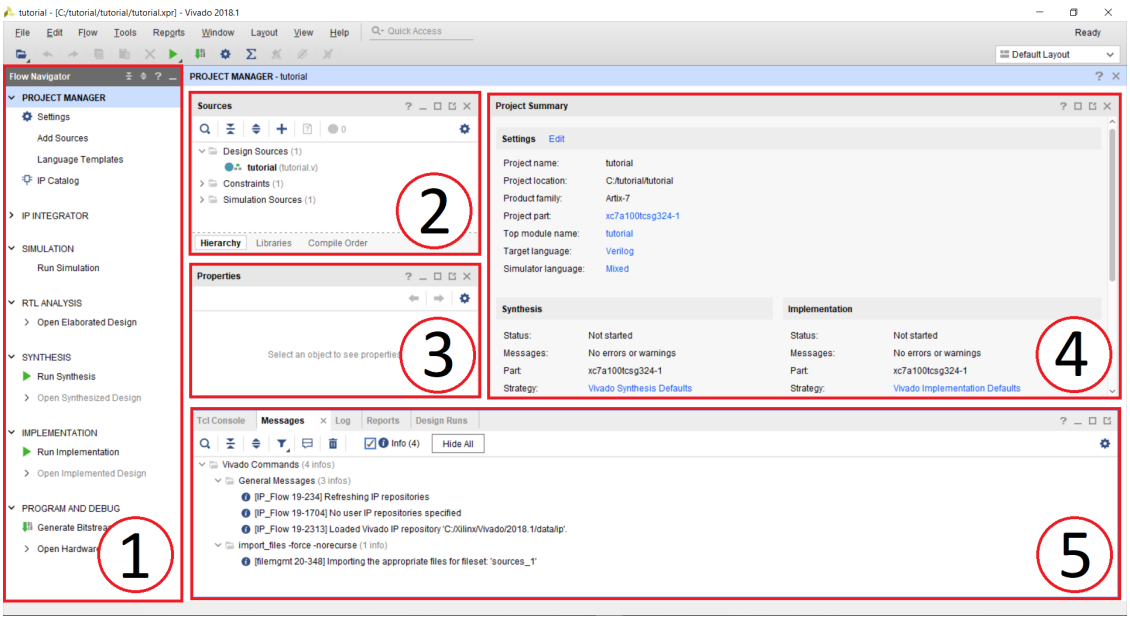

Figure 5: Vivado Project Manager view with panes labelled

default the *Hierarchy* view is selected for the sources, which displays all of your sourced based on their connection hierarchy within the project. The *top level* module in this view will be displayed in bold face. The section labelled with a **3** is the properties window and it displays the properties of whatever you have selected, such as the properties of a selected file in the Sources window.

Next, the main content window is labelled with a **4**. Here is where the contents of any open files will be displayed, and where their contents can be edited. By default, a *Project Summary* page is displayed here that summarizes the current state of the project. The subsections *Utilization* and *Timing* are particularly useful in this summary, showing the project's utilization of the FPGA resources (which is useful to determine whether you have enough resources to implement the project) and whether the routed wires of the project have delays compatible with the clock frequencies within the project (wires with too large delays will likely cause the project to fail).

Finally, the last section, labelled with a **5**, shows various information organized in tabs. The first tab, called *TCL Console*, is a console that allows Vivado commands to be issued. Vivado commands are specified in a language called TCL (hence the name TCL console), and are often used to perform advanced tasks or for scripting simpler tasks. Note, any commands you initiate with the GUI will appear in the TCL console as well. The *Messages* tab shows all of the **Errors**, **Critical Warnings**, **Warnings**, and **Info Messages** for the steps run within Vivado. Generally speaking, most projects will have a very large amount of Warnings, and so we tend to focus on Errors and Critical Warnings here when we have a problem, though it is sometimes useful to comb through the Warning messages to ensure they aren't indicating an unexpected problem. The *Log* tab shows the log output of the various running steps, and the *Reports* tab lists all of the reports generated by the steps already run. The final tab lists the *Design Runs*, though we'll likely not deviate from the default Design Runs setup in Vivado, so this tab is not that important for us.

3 RTL Analysis

In the *Sources* pane, double-click the **tutorial.v** entry to open the file in text mode. 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. Line 7 defines the beginning (marked with keyword **module**) and Line 19 defines the end of the module (marked with keyword **endmodule**). Lines 8-9 define the input and output ports whereas lines 12-17 define the actual functionality. See Figure 6 for reference. Remember, any changes made to the source will only apply to the version of the source file we copied to this project (since we chose to copy source files into the project). We do not need to make any changes to the source for this tutorial.

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Module Name: tutorial
4  //////////////////////////////////////////////////////////////////////////////////
5
6
7  module tutorial (
8      input [7:0] swt,
9      output [7:0] led
10     );
11
12     assign led[0] = ~swt[0];
13     assign led[1] = swt[1] & ~swt[2];
14     assign led[3] = swt[2] & swt[3];
15     assign led[2] = led[1] | led[3];
16
17     assign led[7:4] = swt[7:4];
18
19 endmodule
```

Figure 6: The Verilog source file

In the *Sources* pane, expand the *Constraints* folder and double-click the **tutorial.xdc** entry to open the file in text mode. Lines 1-14 define the pin locations of the input switches [6:0] and lines 16-29 define 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 later in the tutorial.

Now that we have our sources setup and are happy with their contents, we can see what hardware will be generated from the source. Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**. Click **OK** on any pop-ups that come up. The model (design) will be elaborated and the *Elaborated Design* view will be opened with a logic view of the design displayed. See Figure 7 for the logic view that we expect. 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.

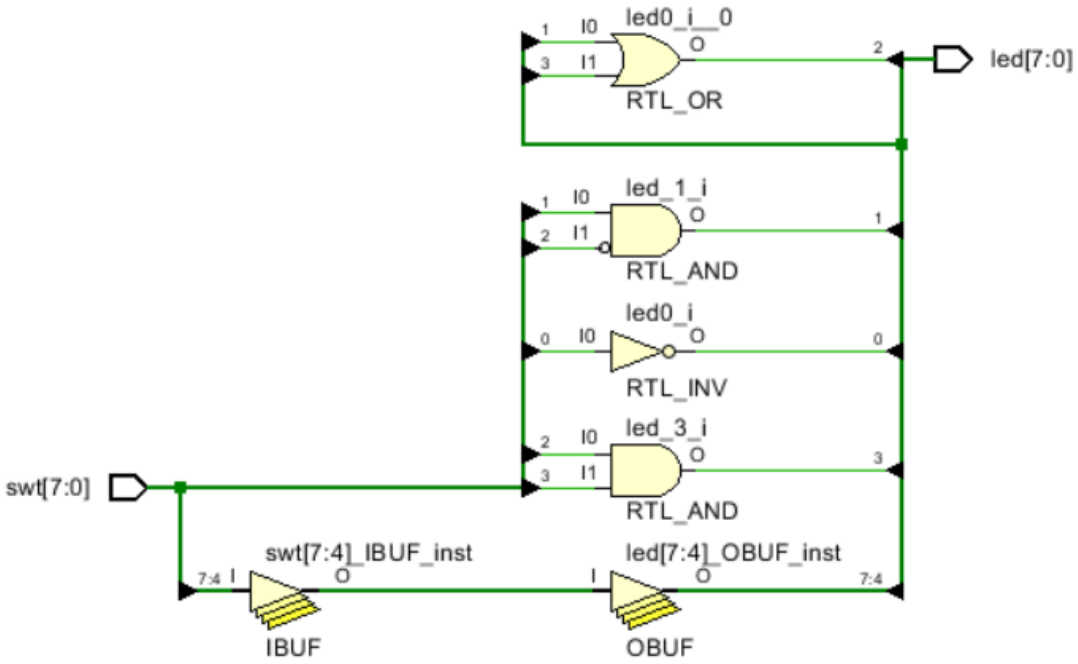

Figure 7: The schematic created from the elaborated design

4 Adding I/O Constraints

In Section 3 we noted that we had some missing constraints in our project. The pin locations for led[6:0] and swt[6:0] are defined in our *constraints.xdc*; we could add the constraints for pins led[7] and swt[7] to that file, though for the sake of demonstration we'll go over a number of other ways to add pin constraints to a project.

Once RTL analysis is performed and we have our *Elaborated Design* view open, another standard layout called the I/O Planning layout is available. Click on the drop-down list on the top-right of the Vivado interface and select the I/O Planning layout. See Figure 8 for details. Notice that the *Package* view is displayed in the Content View area and the *I/O ports* tab is displayed in the Console View area at the bottom of the Vivado interface. Also notice that design ports (led and swt) are listed in the I/O Ports tab with both having multiple I/O standards.

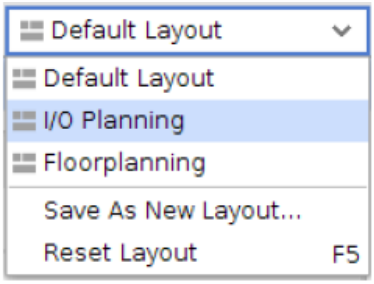

Figure 8: The Layout dropdown menu

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 (See Figure 9).

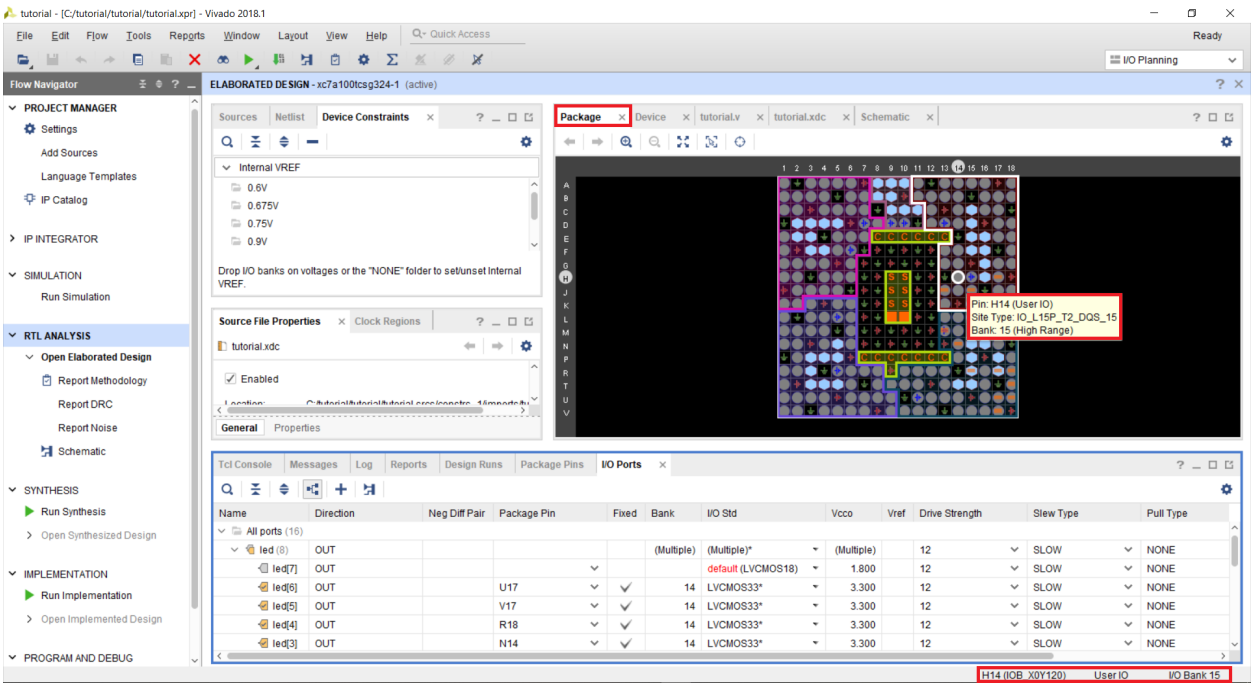

Figure 9: The Package view with some IO site highlighted

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 (see Figure 10).

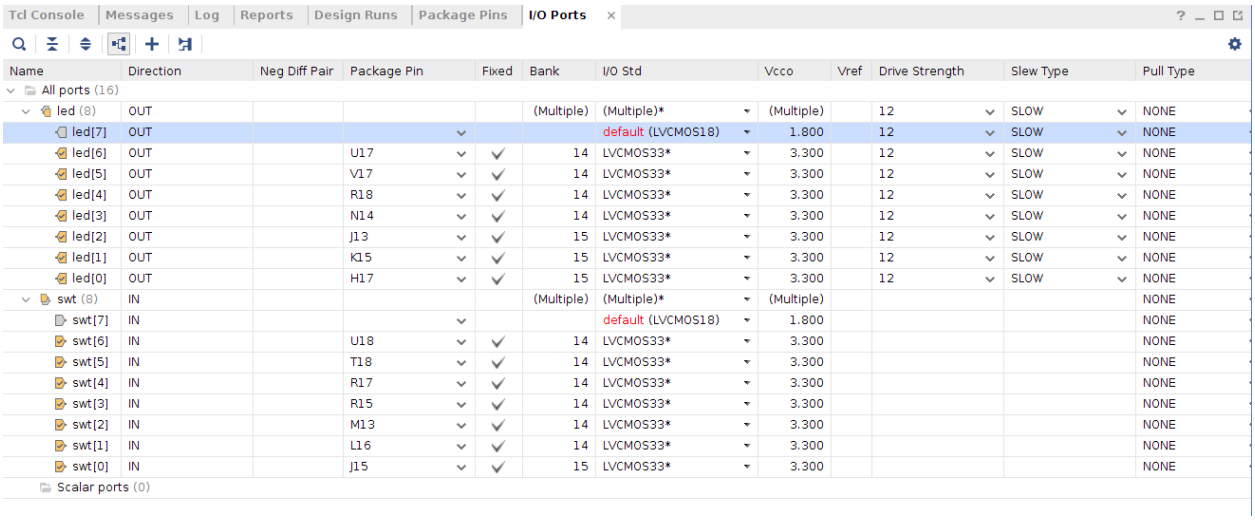

Figure 10: The pin sites assigned to each of the top level signals

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. Similarly,

click under the *I/O Std* column across the **led[7]** row and select LVCMOS33. This assigns the LVCMOS33 standard to the site. We have now assigned a pin location using the *I/O Planning* methodology, though you can also assign pins using TCL commands. Open the TCL Console and type the following commands to select the pin and I/O standard for **swt[7]**:

```
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 changed in the I/O Ports tab.

Finally, You can also assign the pin constraints 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 the *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 (see Figure 11).

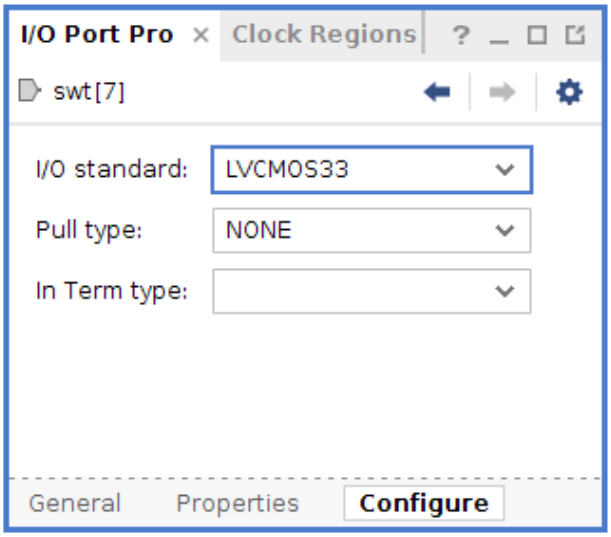

Figure 11: Selecting the IO Standard in the pin property window

Once we have assigned all of our pin locations, we can save our assignments to the *tutorial.xdc* file. Select save on upper left part of Vivado to save the constraints in the *tutorial.xdc* file. Click **Update** when warning is prompted. Note, the changes were made to the project's local copy of the *constraints.xdc* file. Close the *Elaborated Design* view to return to the *Project Manager*.

5 Simulating with XSim

We usually want to test our projects using the Vivado simulator before synthesizing the project and downloading the project to the board. In order to test the project we use a testbench, which is simply a Verilog source file designed to test our project. Click **Add Sources** under the *Project Manager* tasks of the *Flow Navigator* pane. Select the *Add or Create Simulation Sources* option and click **Next** (See Figure 12). In the *Add Source Files* form, click the **Add Files...** button. Navigate to the directory to which you extracted the files and select *tutorial_tb.v* and click **OK** and then click **Finish**. In the *Sources* pane, the hierarchy under *Design Sources* should remain

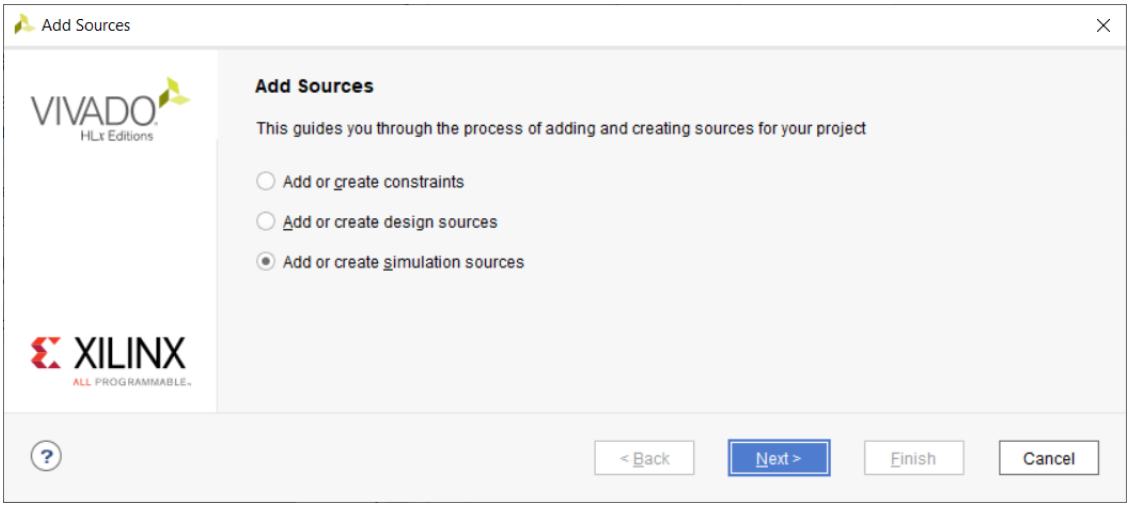

Figure 12: Adding simulation sources to the project

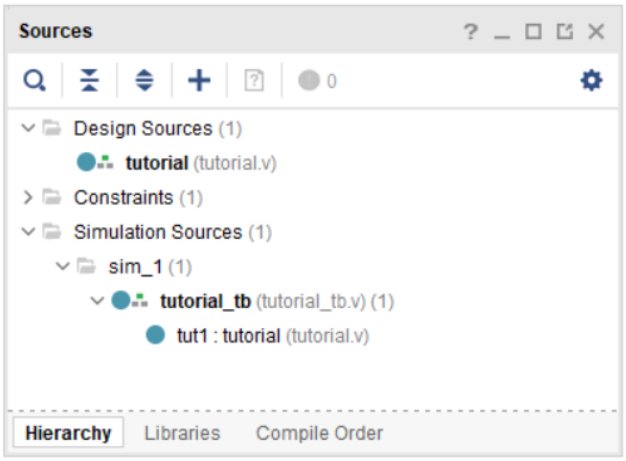

Figure 13: Source hierarchy after adding a testbench

unchanged, though if you expand the *Simulation Sources* folder, you should now see a *tutorial_tb* module under which the *tutorial* module is instantiated, as per Figure 13.

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

Before running the simulation, let's set some of the simulation settings. Right Click **Simulation** in the Flow Navigator pane and select Simulation Settings. A *Settings* form will appear showing the *Simulation* properties form. Select the Simulation tab, and set the **Simulation Run Time** value to 200 ns and click **OK** (See Figure 14). Click on **Run Simulation** → **Run Behavioral Simulation** under the *Simulation* tasks of the *Flow Navigator* pane. The testbench and source files will be compiled and the XSim simulator will be run (assuming no errors).

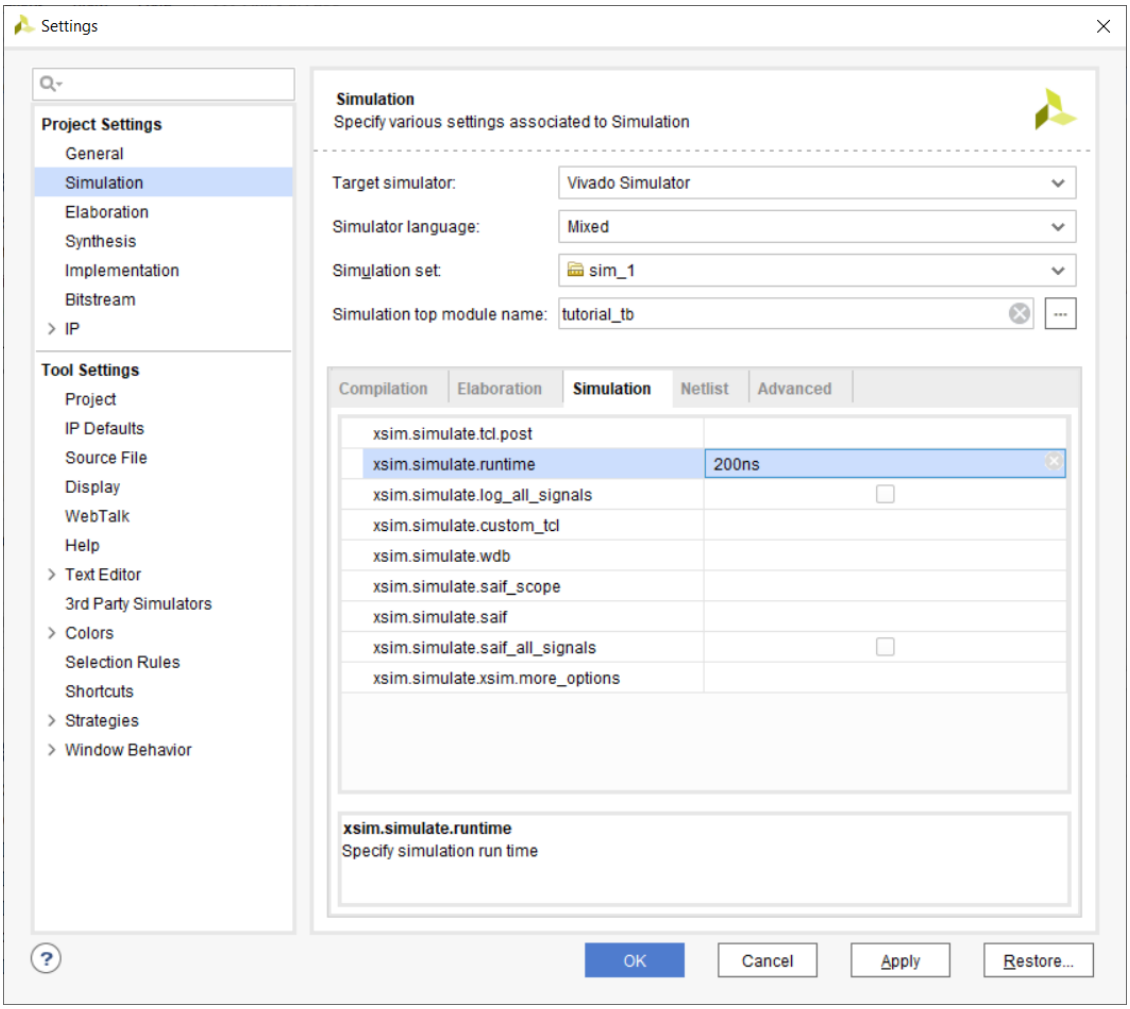

Figure 14: Changing the simulation settings

You will see a simulator output similar to the one shown in Figure 15. You will see four main views: (i) Scopes, where the testbench hierarchy as well as global instances are displayed, (ii) Objects, where top-level signals are displayed, (iii) the waveform window, and (iv) the TCL Console where the simulation outputs are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

Click on the **Zoom Fit** button ( ) located above the waveform window to see the entire waveform within the current waveform window. Also, to get a better view of the waveform, you can click the **Float button** in the top-right-most corner of the waveform window to detach the waveform window from the rest of the simulation view and re-size it independently. Figure 16 shows the expected waveform output from our simulation.

Within the wavefrom viewer, there are a number of things we can change to make it easier to examine the output. 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*. Finally, change the radix for the **leds[7:0]** and **e_led[7:0]** to binary as we want to see each output bit.

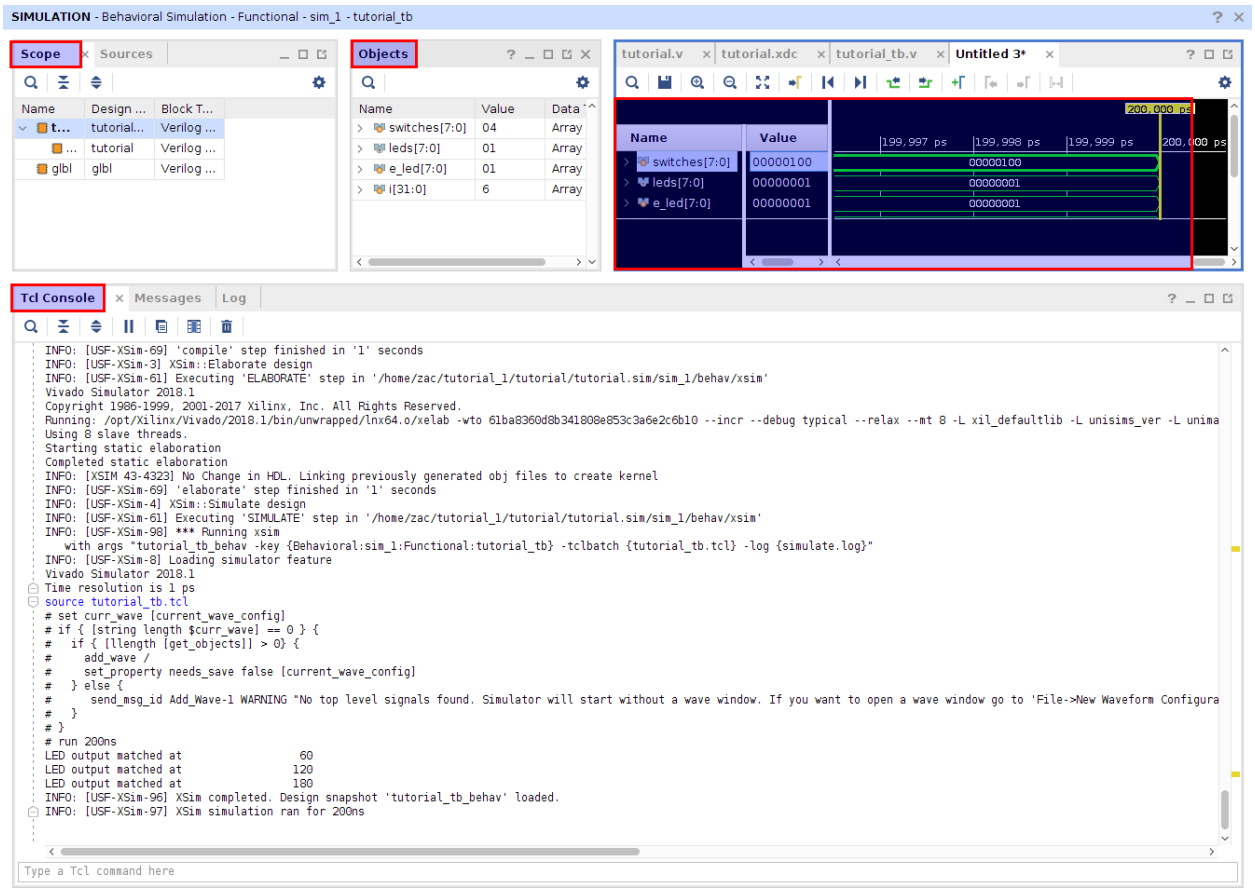

Figure 15: Simulator output

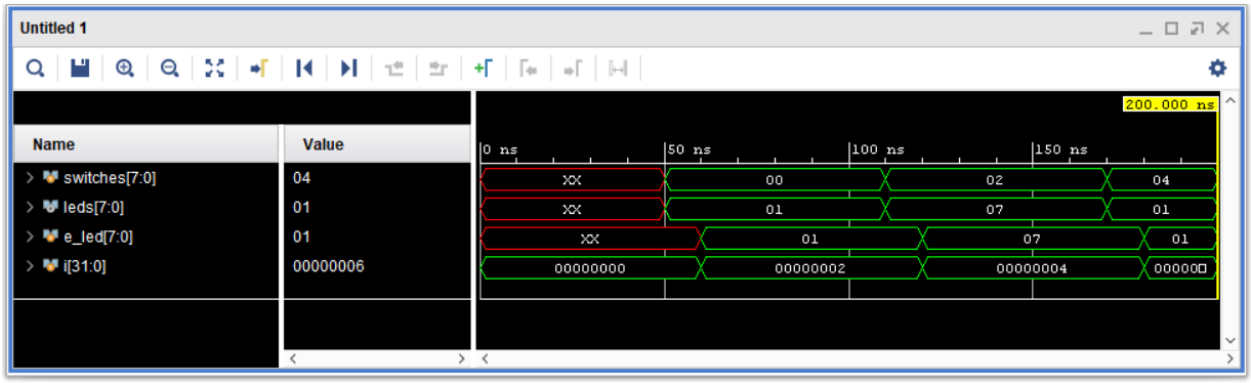

Figure 16: Simulator waveform

We can also add more signals to the waveform viewer for monitoring. 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 (see Figure 17). Select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals.

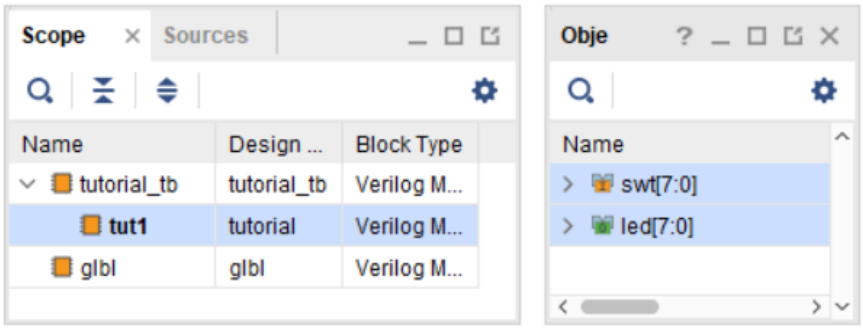

Figure 17: Simulator Scopes and Objects windows

In order to see values for our newly added signals, we need to run the simulator again. To do this, examine the simulator ribbon bar. Specifically, the buttons in Figure 18 are of note. The first button rewinds the simulation to time zero; if we want to rerun the simulation, we must rewind it first. Next, the play button will run the simulator indefinitely until it receives a stop signal or the testbench has nothing else to do; for most uses cases this is not useful (unless we explicitly have a $finish command in our testbench) as it will cause the simulator to run forever. Finally, the play button annotated with a (T) will run the simulator for an additional amount of time, specified in the entry box beside it. **Enter** the value 500ns into the box, press the **Rewind** button, and then press the **Timed Start** button to rerun the simulator for 500ns from time zero. We expect to see the waveform from Figure 19 (remember to press the **Zoom Fit** button to fit the entire waveform in the window).

Figure 18: Simulator ribbon bar to control simulation

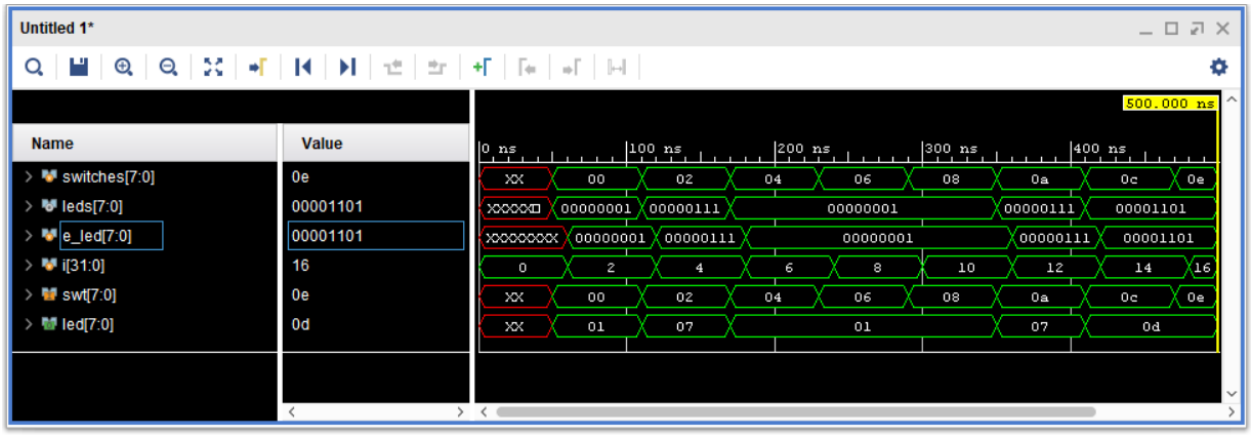

Figure 19: Simulator waveform updated

Close the simulator by selecting **File** → **Close Simulation**. Click **OK** and then click **Discard** to close without saving the waveform.

6 Design Synthesis

Click on **Run Synthesis** under the *Synthesis* tasks of the *Flow Navigator* pane and then click **OK** to start the synthesis process. 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. Usually we would proceed to the **Run Implementation**, though we don't want to run the implementation yet, so press **Cancel** rather then selecting any of the three options.

Open the *Project Summary* page to see summarized information from the synthesis run. The main information we're generally interested in after synthesis is the utilization, so scroll down to that section of the summary. Click on the **Table** tab to see the utilization results in tabular form. We notice that our project requires 3 LUTs and 16 IOs (see Figure 20), which represents a very low amount of of the total resources available on our FPGA (only 0.01% of the LUTs).

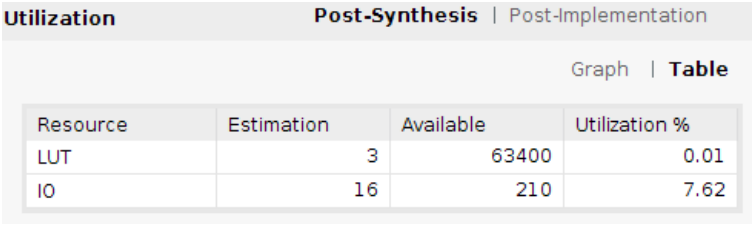

Figure 20: Post-synthesis utilization results

Next, we can take a look at the post-synthesis schematic of our design. Click on **Schematic** under the *Open Synthesized Design* of *Synthesis* tasks of the Flow Navigator pane to view the synthesized design in a schematic view. This will open the *Synthesized Design* view and then bring up the schematic of the synthesized design (see Figure 21). Notice that IBUF and OBUF are automatically instantiated (added) to the design as the inputs and outputs are buffered. The logical gates are implemented in LUTs (1 input LUTs are listed as LUT1, 2 input LUTs are listed as LUT2, and 3 input LUTs are listed as LUT3). The four gates from the RTL analysis schematic (Figure 7) are mapped to four LUTs in this synthesized output. Close the *Synthesized Design* view to return to thr *Project Manager*.

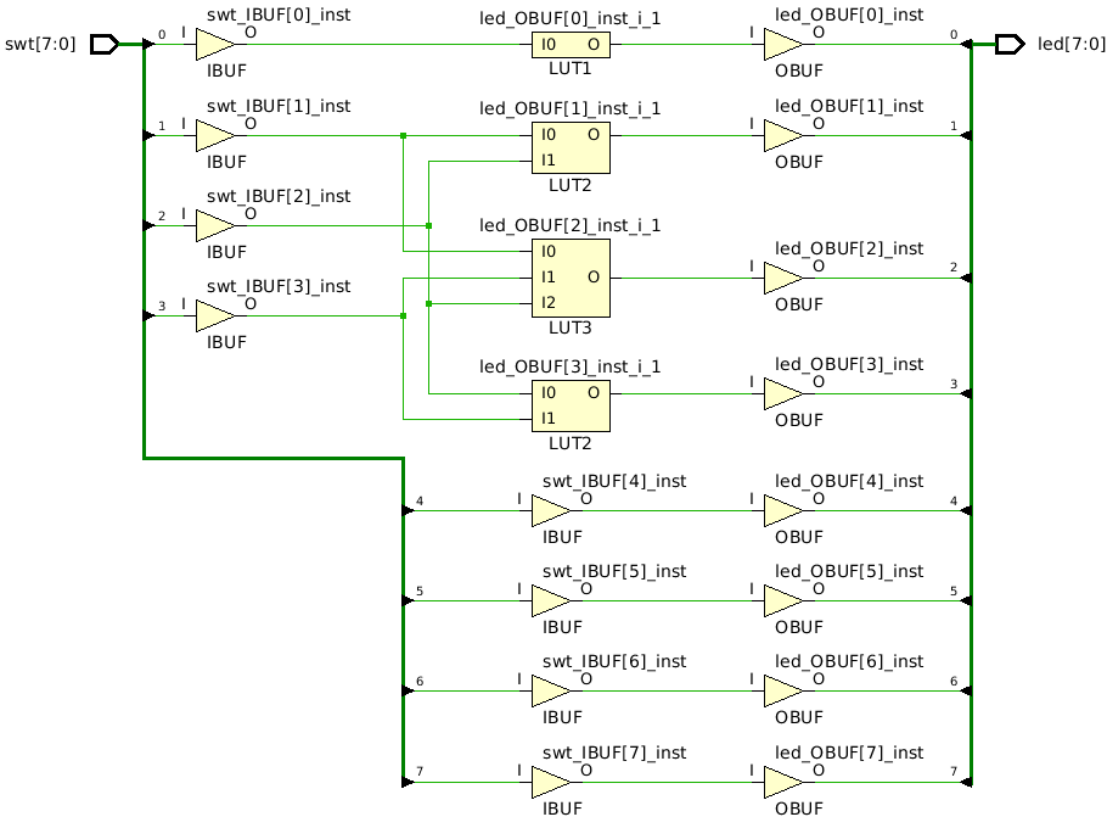

Figure 21: Post-synthesis schematic

7 Design Implementation

Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane and then click **OK** to start the implementation process. The implementation process will run on the results of the synthesis. When the process is completed a *Implementation Completed* dialog box with three options will be displayed. Select *Open implemented design* and click **OK** as we want to look at the implemented design in a Device view tab. The *Implemented Design* view will be opened.

In the *Netlist* pane on the left side, 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). Here we can see the wire used to route that net in the device. This can be done for any net in your design. In addition, under the *Leaf Cells* section in the *Netlist* pane, you can select any of the nodes in your design to see where it was placed. See Figure 22 for an example of a routed net displayed on the *Device* view.

Close the *Implemented Design* view to return to the *Project Manager*. Select the *Project Summary* tab to see the summarized information from the implementation run. In general, we're interested is viewing the post-implementation utilization results (i.e. the final actual utilization) and the timing report summary. First, the timing report should have no information since we don't have any clocks or sequential elements in our design. Looking at the utilization results, we can again look at the utilization *Table* view to see that 3 LUTs and 16 IOs are used. The utilization hasn't changed since synthesis, since the design is pretty simple, though often there is some change in utilization once implementation is run due to additional optimization passes and other factors in implementation. Note, under *Implementation* in the *Flow Navigator*, when you expand *Open*

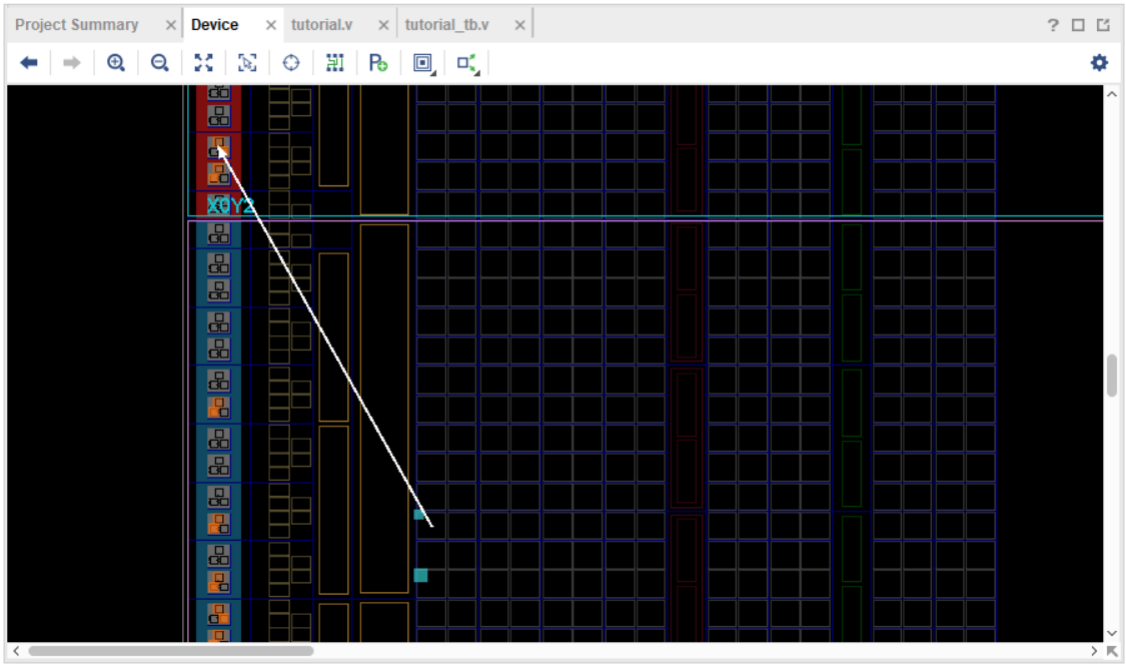

Figure 22: Routed net shown in Implemented Design Device view

Implemented Design, there are options to get more detailed reports for both timing and utilization, which may be helpful in future projects.

8 Timing Simulation with XSim

Select the **Run Simulation → Run Post-Implementation Timing Simulation** process under the *Simulation* section of the *Flow Navigator* pane. The XSim simulator will be launched using the implemented design and tutorial_tb as the top-level module. This simulation is different than the one presented in Section 5 as it considers the real delays of the routed design produced by the implementation run. Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.

We can add markers to the waveform to allow us to more easily see the relative difference between signals. **Right-click** at 50ns (where the switch input is set to 0000000b) and select **Markers → Add Marker**. Similarly, right-click and add a marker at around 55ns where the **leds** complete their transition. You can also add a marker by clicking on the Add Marker button ( ) . Click on the **Add Marker** button and left-click at around 60ns where e_led changes. Note that the leds change 5ns after the input swt changes, which represents the actual delays of our IBUF, OBUF, and net routing. Close the simulation by selecting **File → Close Simulation**; there is no need to save the waveform.

9 Bitsream Generation and Verification on Device

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. The process will create a *tutorial.bit* file in the *impl_1* directory of the *tutorial.runs* directory within your project folder. When the process is completed a *Bitstream Generation Completed* dialog box with four

options will be displayed. If you are ready to program the device, select *Open Hardware Manager* and click **OK**, otherwise click **Cancel**. As long as you have your board ready, you should opt to *Open Hardware Manager*.

The board can be powered by USB or the external power jack. Since we are connecting the USB in order to program the board, we can use this as the power source, though we need to change the *Power Select* jumper on the board to the *USB* setting in order to allow this. If you'd rather leave the board powered through the power jack, leave the jumper and make sure to plug in the board's power. Connect the provided micro-USB cable between the board's *JTAG Programming* port and the PC and power **ON** the board. See Figure 23 for reference.

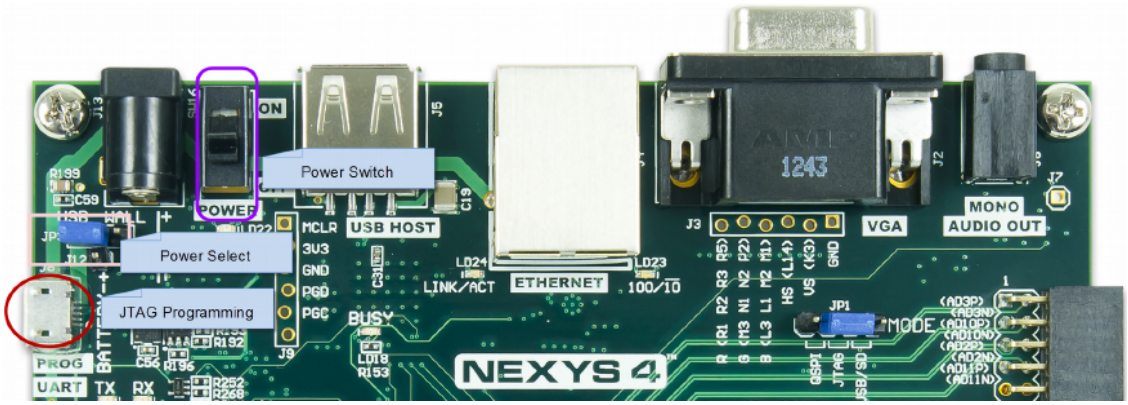

Figure 23: Board power configuration and programming/power port locations

The Hardware Manager window will open indicating "unconnected" status. Click on the **Open target** link and **Auto Connect** (see Figure 24). Select the *Program device* link and click on the FPGA part **xc7a100t_0**. A dialog will appear showing the path to the bitstream file; by default the prompt will point to the bistream file we just generated. Click **Program** to program the FPGA. The DONE light on the FPGA will be lit when the device is programmed.

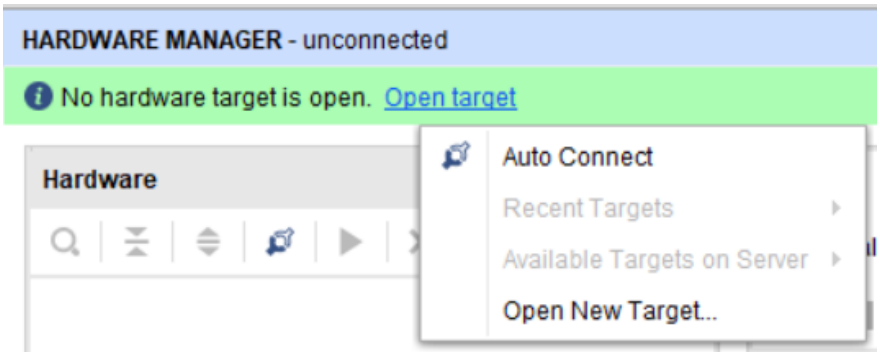

Figure 24: Auto connect to board from Hardware Manager

Verify the functionality by flipping switches and observing the output on the LEDs. When satisfied, power **OFF** the board. Close the hardware session by selecting **File** → **Close Hardware Manager**. Click **OK** to close the session. Close the Vivado program by selecting **File** → **Exit** and click **OK**.

10 Running Vivado in Batch Mode

In the previous sections we have run Vivado in what is called *Project Mode*. You can also run Vivado in *Batch Mode*, where Vivado is given a TCL script and simply runs the steps specified in that TCL script. Batch mode is run from the terminal (or Command Prompt on Windows), so first **open** a Terminal (or Command Prompt) session. From Section 1 recall that we need to source a specific script file in order to use Vivado, follow those steps to do so. Next, change to the directory where you extracted the tutorial files. In this directory you should see a TCL file called *tutorial_tcl_with_sim.tcl*.

The contents of this TCL file are shown in Figure 25. Line 1 sets the output directory path; line 2 creates the project directory tutorial_tcl_with_sim under the $outDir directory targeting the Artix-7 100 part; line 3 adds the source file; line 4 imports the constraints files; line 5 sets the top module file; line 6 executes the rtl analysis command; lines 7 through 10 add the missing I/O pins constraints; lines 11 through 13 save the constraints in the target xdc file; lines 14 through 17 set up, read, and compile the testbench; line 18 runs the behavioral simulation; lines 19 through 22 synthesizes and implements the design; and line 23 generates the bitstream. Note that wait_on_run on lines 20 and 22 are essential as the results from the steps that precede them are needed before executing the next command.

```
1  set outDir .
2  create_project tutorial_tcl_with_sim $outDir/tutorial_tcl_with_sim -part xc7a100tcsg324-1
3  add_files -norecurse $outDir/tutorial.v
4  import_files -fileset constrs_1 -force -norecurse $outDir/tutorial.xdc
5  update_compile_order -fileset sources_1
6  synth_design -rtl -name rtl_1
7  set_property package_pin U16 [get_ports {led[7]}]
8  set_property iostandard LVCMOS33 [get_ports [list {led[7]}]]
9  set_property package_pin R13 [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/tutorial_1/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/tutorial_tb.v
16 update_compile_order -fileset sim_1
17 set_property runtime 200ns [get_filesets sim_1]
18 launch_simulation -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 wait_on_run impl_1
```

Figure 25: The TCL script for batch mode

To run Vivado in batch mode using this script, type the following command into the terminal or command prompt:

```
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. Once the script is done executing, you can close the terminal and examine the generated folder structures.

11 Summary

The Vivado software tools 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 batch mode using the provided TCL script.