EE3220 System-on-Chip Design

Tutorial 7: Vivado Design Flow for SoC

Please go to https://github.com/xupgit/FPGA-Design-Flow-using-Vivado for more details. We are doing the tutorial for the first step.

Objectives

After completing this tutorial, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting the ZYNQ device located on the PYNQ-Z2
- Use the provided Xilinx Design Constraint (XDC) file to constrain the pin locations
- Simulate the design using the Vivado simulator (optional)
- Synthesize and implement the design
- Generate the bitstream
- Configure ZYNQ using the generated bitstream and verify the functionality

Tools

The Tools required for this tutorial includes:

- Software: Xilinx Vivado
- Hardware: PYNQ-Z2 FPGA board

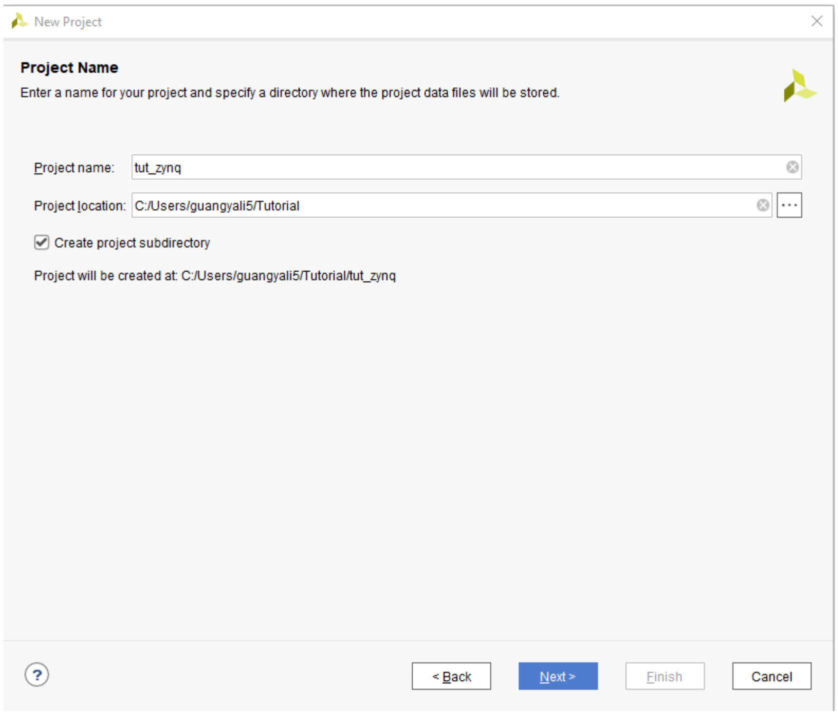
Students could complete this tutorial with remotely logging into laboratory. You can also download the Xilinx Vivado https://www.xilinx.com/support/download.html on your PC if you are interested.

Steps

1. Create a Vivado Project using IDE

1.1 Launch Vivado and create an empty project targeting the PYNQ-Z2 board, selecting Verilog (or VHDL) as a target language. Use the provided tut_zynq.v and tut_zynq.xdc files.

1. Open Vivado by selecting **Start > Xilinx Design Tools > Vivado**
2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
3. Click the Browse button of the *Project location* field of the **New Project** form, browse to a suitable location **(C:/Users/guangyali5/Tutorial in the figure)**, and click **Select**.
4. Enter **tut_zynq** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: tut_zynq

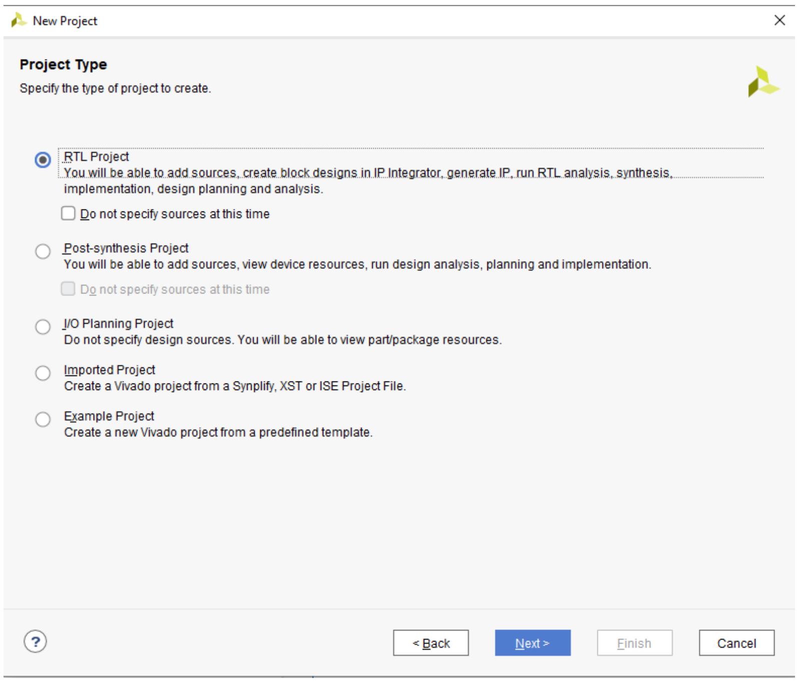
Project location: C:/Users/guangyali5/Tutorial

[✓] Create project subdirectory

Project will be created at: C:/Users/guangyali5/Tutorial/tut_zynq

Project Name and Location entry

5. Select **RTL Project** option in the *Project Type* form, and do **NOT** select **Do not specify sources at this time**, click **Next**.



Project Type

Specify the type of project to create.

[●] RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
[ ] Do not specify sources at this time

( ) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
[ ] Do not specify sources at this time

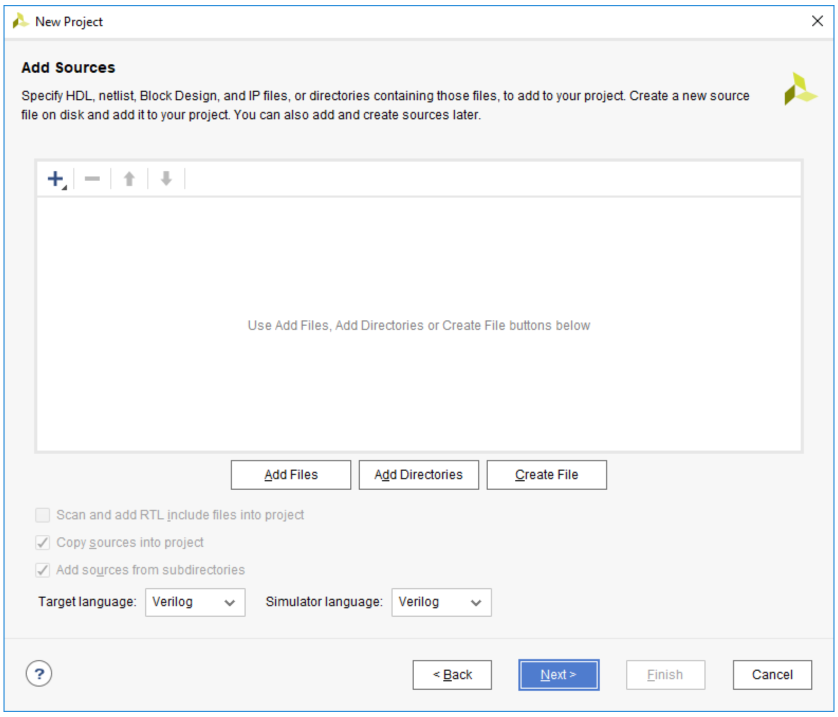
( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

Selecting Target and Simulator language

6. Using the drop-down buttons, select **Verilog** as the *Target Language* and *Simulator Language* in the *Add Sources* form.


New Project

Add Sources

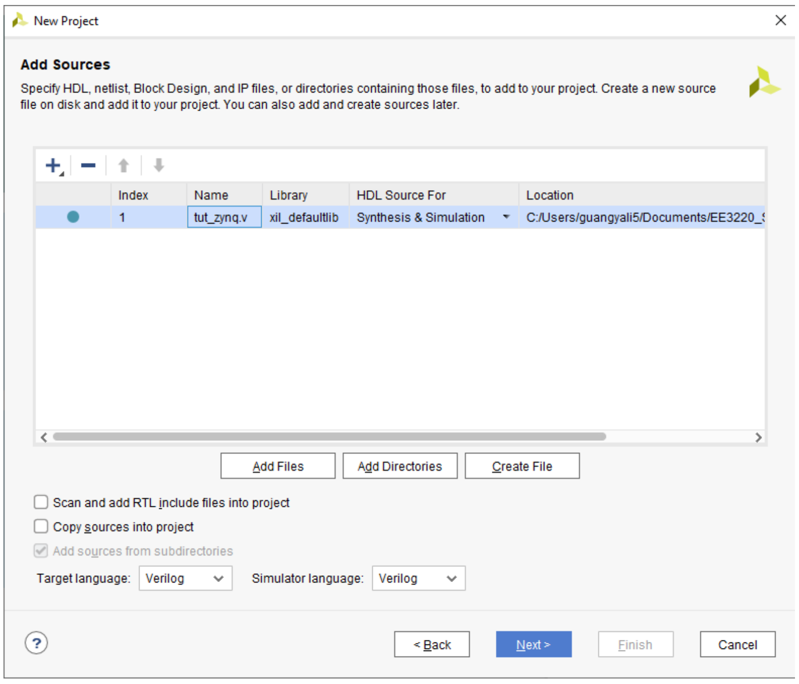
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
Target language: Verilog
Simulator language: Verilog

Selecting Target and Simulator language

7. Click on the **Blue Plus** button, then **Add Files...** and select *tut_zynq.v,* click **OK**.


New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
●	1	tut_zynq.v	xil_defaultlib	Synthesis & Simulation	C:/Users/guangyali5/Documents/EE3220_

Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
Target language: Verilog
Simulator language: Verilog

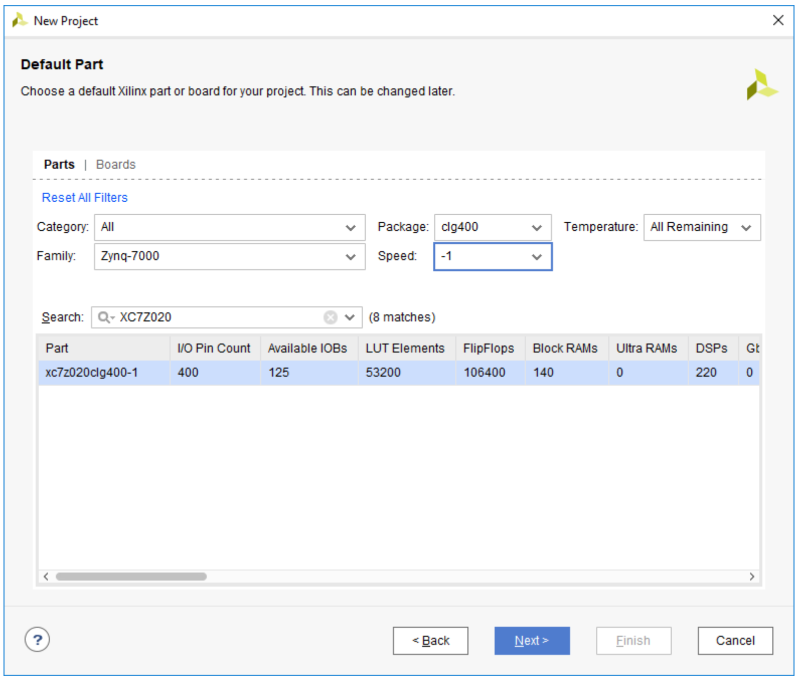
If it is not already checked, check **Copy sources into project**.

8. Click **Next** to get to the *Add Constraints* form.
9. Click on the **Blue Plus** button, then **Add Files...** and select *tut_zynq.xdc* and click **OK** (if necessary), and then click **Next.**

   If it is not already checked, check **Copy sources into project**.

The Xilinx Design Constraints file assigns the physical IO locations on FPGA to the switches and LEDs located on the board. This information can be obtained either through the board's schematic or the board's user guide.

10. In the *Default Part* form, use the **Parts** option and various drop-down fields of the **Filter** section. Select the **XC7Z020clg400-1**.



New Project

Default Part

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

Parts | Boards

Reset All Filters

Category: All
Family: Zynq-7000
Package: clg400
Speed: -1
Temperature: All Remaining

Search: XC7Z020 (8 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7z020clg400-1	400	125	53200	106400	140	0	220	0

Part Selection for the PYNQ

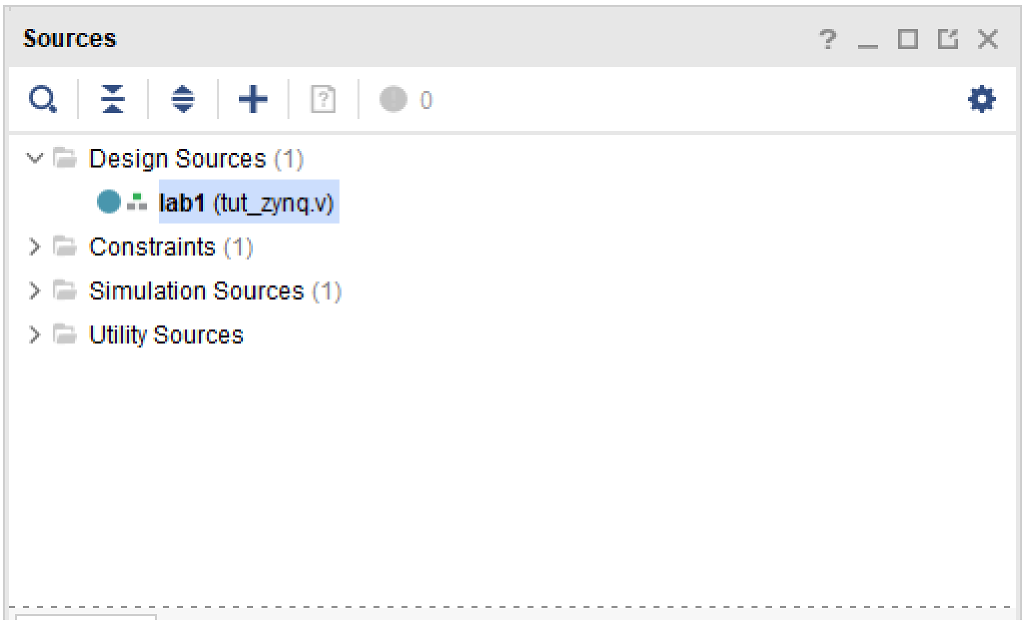
You may also select the **Boards** option, select www.digilentinc.com for the PYNQ-Z1 board, tul.com.tw for the PYNQ-Z2 board under the Vendor filter and select the appropriate board. Notice that PYNQ-Z1 and PYNQ-Z1 may not be listed as they are not in the tools database. If not listed then you can download the board files for the desired boards either from Digilent PYNQ-Z1 webpage or TUL PYNQ-Z2 webpage. Click **Next**.

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

Use the Windows Explorer and look at the **your created project** directory. You will find that the lab1.cache and lab1.srcs directories and the lab1.xpr (Vivado) project file have been created. The lab1.cache directory is a place holder for the Vivado program database. Two directories, constrs_1 and sources_1, are created under the lab1.srcs directory; deep down under them, the copied lab1_.xdc (constraint) and lab1.v (source) files respectively are placed.

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

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

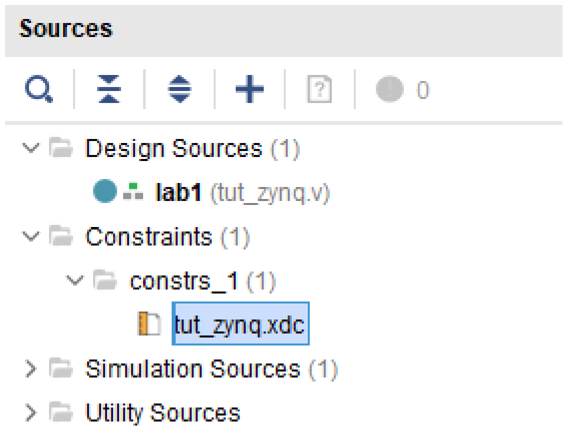


Opening the source file

2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. Lines 2-4 are comment lines describing the module name and the purpose of the module.
3. Line 7 defines the beginning (marked with keyword **module**) and Line 17 defines the end of the module (marked with keyword **endmodule**).
4. Lines 8-9 defines the input and output ports whereas lines 12-15 defines the actual functionality.

1.3 Open the tut_zynq.xdc source and analyze the content.

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



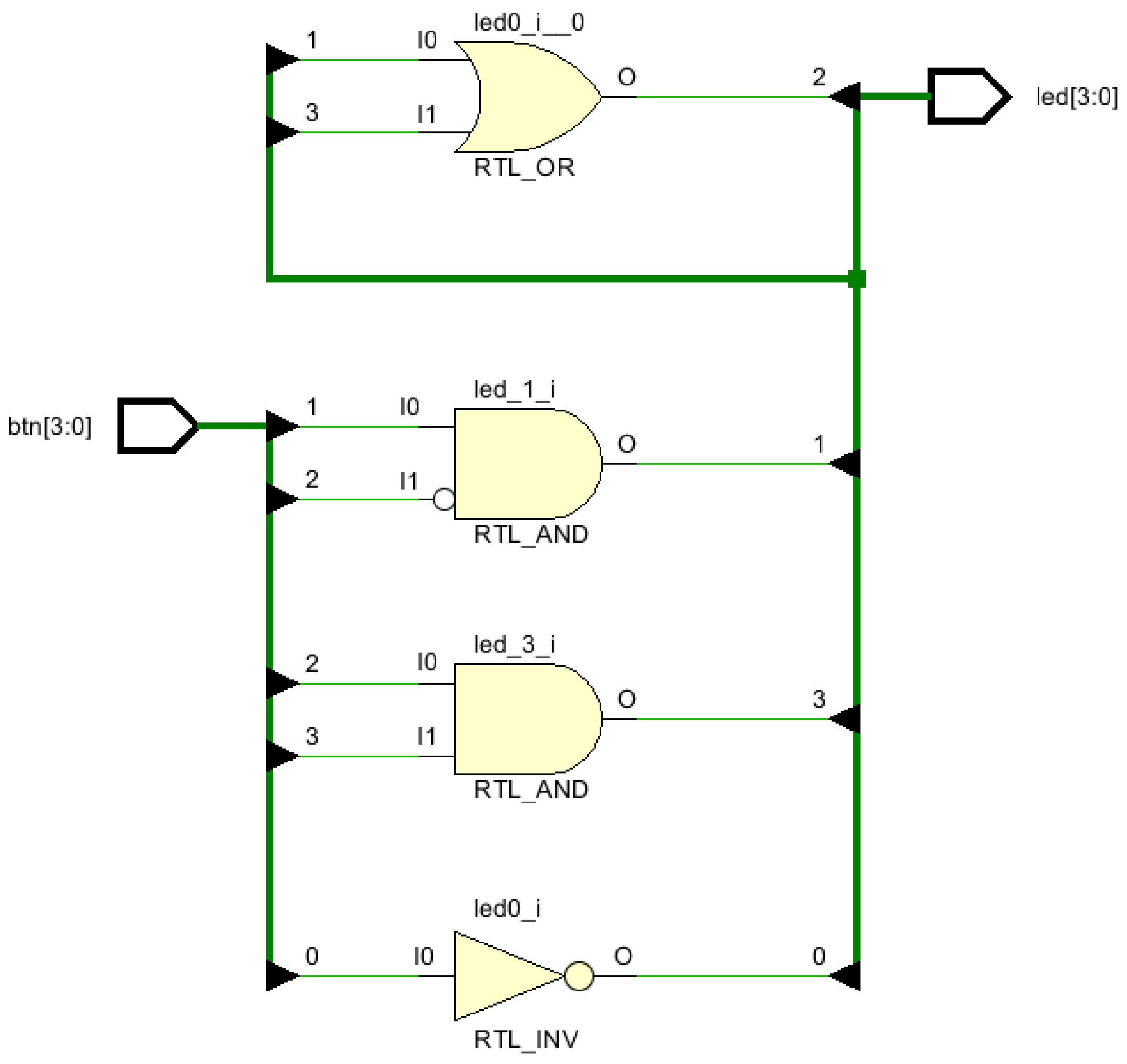
Opening the constraint file

2. Lines 5-8 define the pin locations for the input buttons and lines 13-16 define pin locations for output LEDs.

2. Perform RTL analysis on the source file.

Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**.

The model (design) will be elaborated and a logic view of the design is displayed.

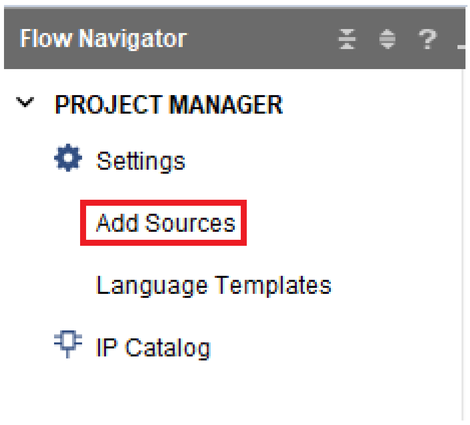


A logic view of the design

3. Simulate the Design using the Vivado Simulator (Optional)

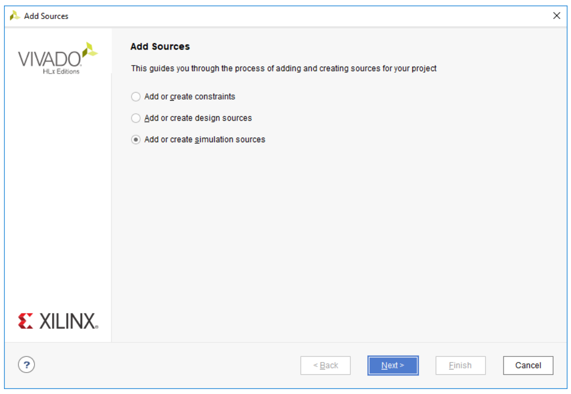
3.1 Add the tut_zynq_tb.v testbench file.

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



Add Sources

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



Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

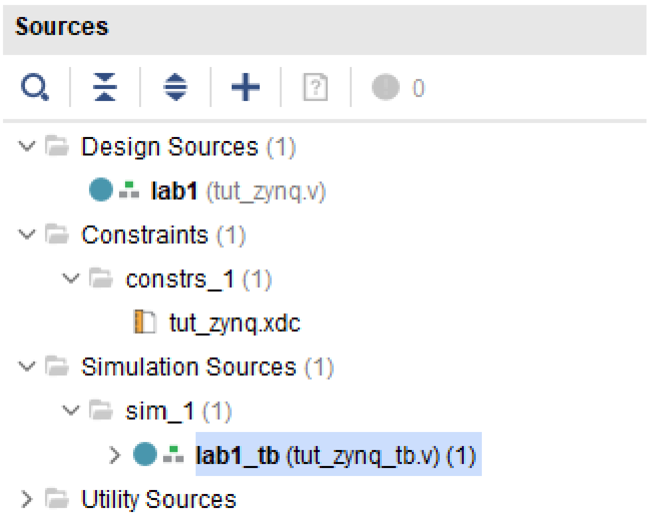
- ○ Add or create constraints
- ○ Add or create design sources
- ● Add or create simulation sources

Selecting Simulation Sources option

3. In the *Add Sources Files* form, click the **Blue Plus** button and then **Add Files...**.
4. Select *tut_zynq_tb.v* and click **OK**.

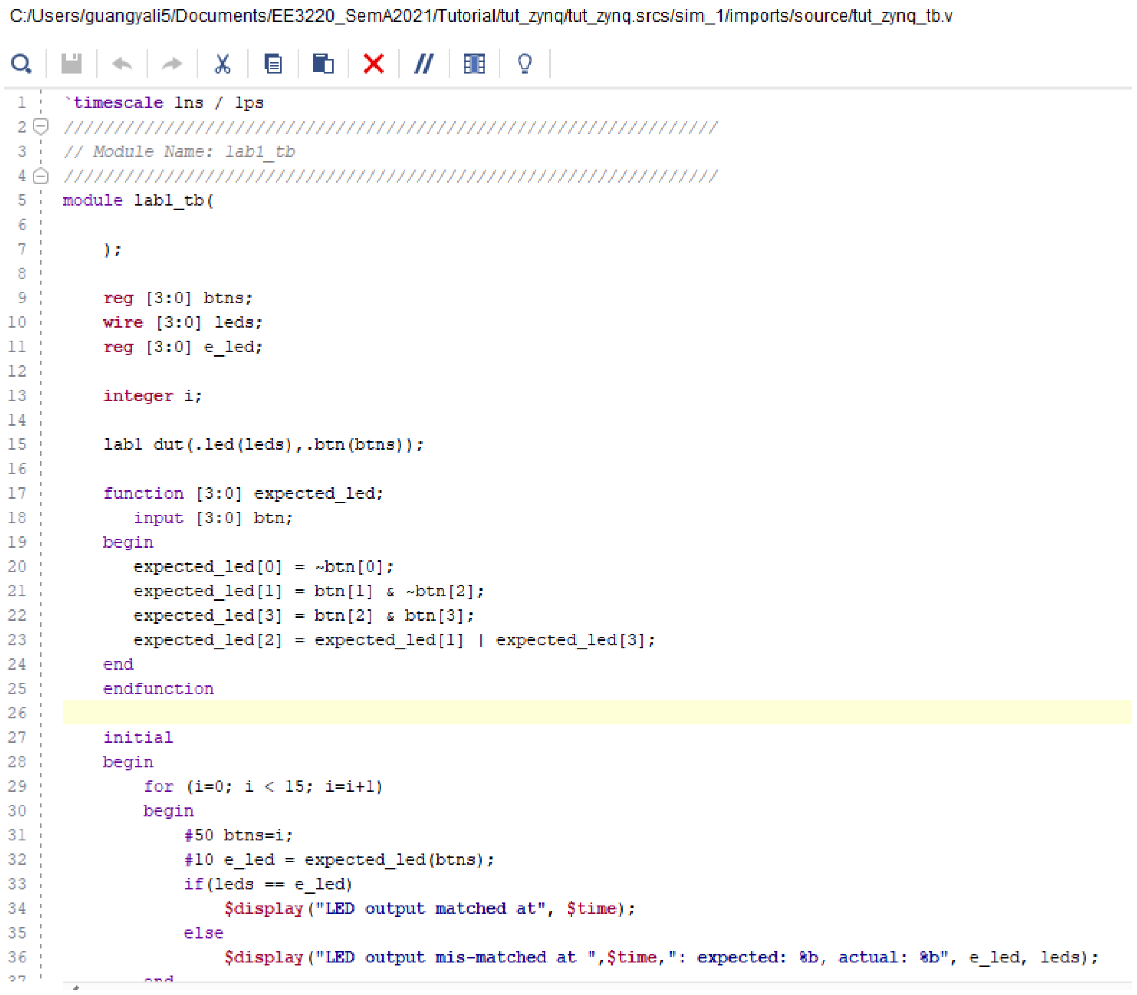
   If it is not already checked, check **Copy sources into project**.
5. Click **Finish**.
6. Select the *Sources* tab and expand the *Simulation Sources* group.

   The *tut_zynq_tb.v* file is added under the *Simulation Sources* group, and **tut_zynq.v** is automatically placed in its hierarchy as a dut (device under test) instance.


```
Sources
Design Sources (1)
    lab1 (tut_zynq.v)
Constraints (1)
    constrs_1 (1)
        tut_zynq.xdc
Simulation Sources (1)
    sim_1 (1)
        lab1_tb (tut_zynq_tb.v) (1)
Utility Sources
```

Simulation Sources hierarchy

7. Using the Windows Explorer, verify that the **sim_1** directory is created at the same level as constrs_1 and sources_1 directories under the lab1.srcs directory, and that a copy of *tut_zynq_tb.v* is placed under **tut_zynq.srcs > sim_1 > imports > source**.
8. Double-click on the **lab1_tb** in the *Sources* pane to view its contents.


C:/Users/guangyali5/Documents/EE3220_SemA2021/Tutorial/tut_zynq/tut_zynq.srcs/sim_1/imports/source/tut_zynq_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Module Name: lab1_tb
//////////////////////////////////////////////////////////////////
module lab1_tb(

    );

    reg [3:0] btns;
    wire [3:0] leds;
    reg [3:0] e_led;

    integer i;

    lab1 dut(.led(leds),.btn(btns));

    function [3:0] expected_led;
       input [3:0] btn;
    begin
       expected_led[0] = ~btn[0];
       expected_led[1] = btn[1] & ~btn[2];
       expected_led[3] = btn[2] & btn[3];
       expected_led[2] = expected_led[1] | expected_led[3];
    end
    endfunction

    initial
    begin
        for (i=0; i < 15; i=i+1)
        begin
            #50 btns=i;
            #10 e_led = expected_led(btns);
            if(leds == e_led)
                $display("LED output matched at", $time);
            else
                $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
```

The self-checking test bench

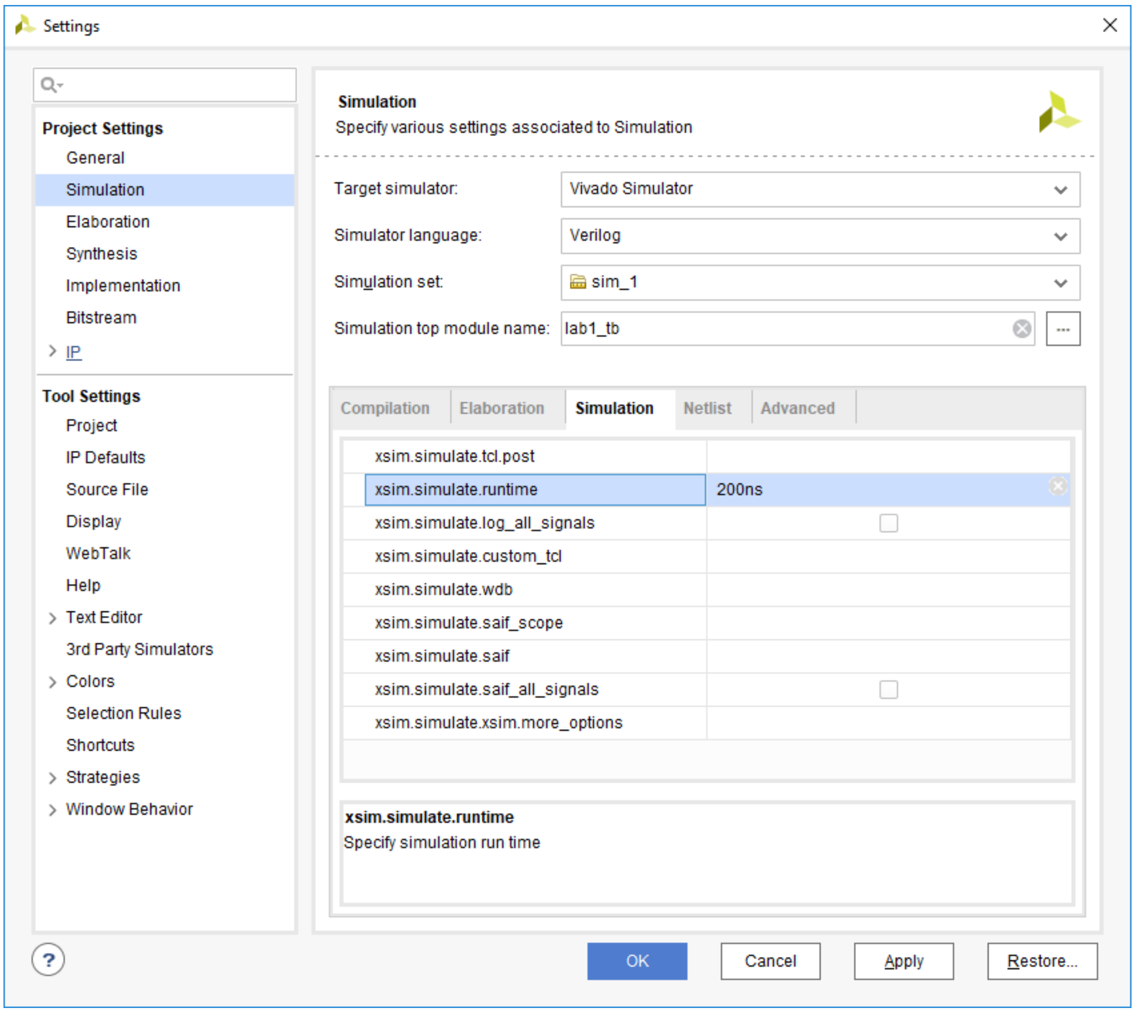
The test bench defines the simulation step size and the resolution in line 1. The test bench module definition begins on line 5. Line 15 instantiates the DUT (device/module under test). Lines 17 through 25 define the same module functionality for the expected value computation. Lines 27 through 38 define the stimuli generation, and compare the expected output with what the DUT provides. Line 40 ends the test bench. The **$display** task will print the message in the simulator console window when the simulation is run.

3.2 Simulate the design for 200 ns using the Vivado simulator.

1. Select **Settings** under the *Project Manager* tasks of the *Flow Navigator* pane.

   A **Settings** form will appear showing the **Simulation** properties form.

2. Select the **Simulation** tab, and set the **Simulation Run Time** value to 200 ns and click **OK**.



Settings

Simulation
Specify various settings associated to Simulation

Target simulator:	Vivado Simulator
Simulator language:	Verilog
Simulation set:	sim_1
Simulation top module name:	lab1_tb

Compilation | Elaboration | Simulation | Netlist | Advanced

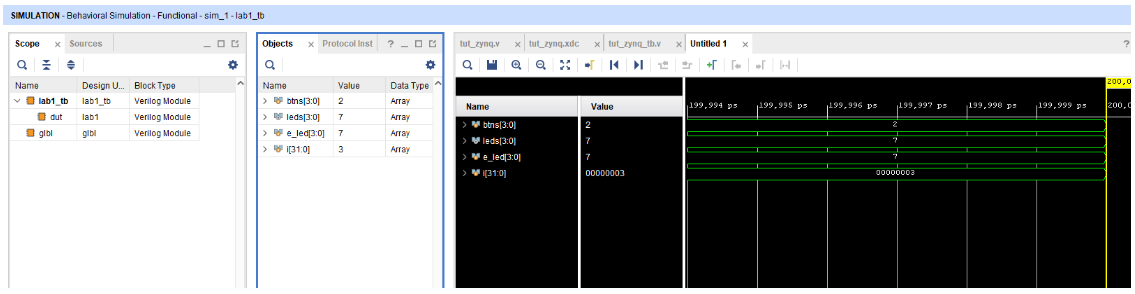
xsim.simulate.tcl.post	
xsim.simulate.runtime	200ns
xsim.simulate.log_all_signals	☐
xsim.simulate.custom_tcl	
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

xsim.simulate.runtime
Specify simulation run time

Setting simulation run time

3. Click on **Simulation > Run Simulation > Run Behavioral Simulation** under the *Project Manager* tasks of the *Flow Navigator* pane.

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


SIMULATION - Behavioral Simulation - Functional - sim_1 - lab1_tb

Name	Design U...	Block Type
lab1_tb	lab1_tb	Verilog Module
dut	lab1	Verilog Module
glbl	glbl	Verilog Module

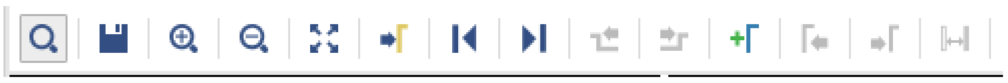
Name	Value	Data Type
btns[3:0]	2	Array
leds[3:0]	7	Array
e_led[3:0]	7	Array
i[31:0]	3	Array

Name	Value
btns[3:0]	2
leds[3:0]	7
e_led[3:0]	7
i[31:0]	00000003

Simulator output

You will see four main views: (i) *Scopes,* where the testbench hierarchy as well as glbl instances are displayed, (ii) *Objects,* where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

You will see several buttons next to the waveform window which can be used for the specific purpose as listed in the table below.



Various buttons available to view the waveform

4. Click on the *Zoom Fit* button ( ) to see the entire waveform.

   Notice that the output changes when the input changes.

   You can also float the simulation waveform window by clicking on the Float button ( ) on the upper right hand side of the view. This will allow you to have a wider window to view the simulation waveforms. To reintegrate the floating window back into the GUI, simply click on the Dock Window button ( ).

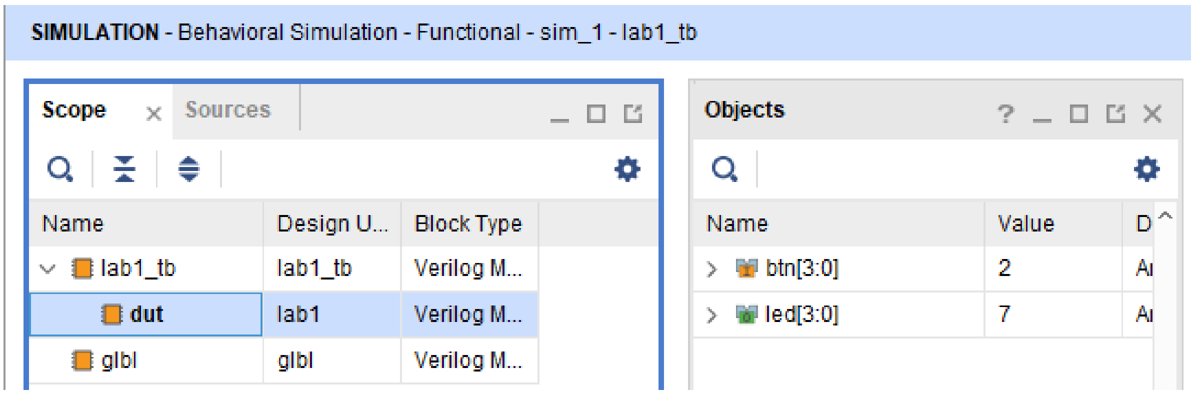
3.3 Change display format if desired.

Select **i[31:0]** in the waveform window, right-click, select *Radix*, and then select *Unsigned Decimal* to view the for-loop index in an unsigned *integer* form. Similarly, change the radix of **btn[3:0]** to *Hexadecimal*. Leave the **leds[3:0]** and **e_led[3:0]** radix to *binary* as we want to see each output bit.

Add more signals to monitor the lower-level signals and continue to run the simulation for 500 ns.

1. Expand the **lab1_tb** instance, if necessary, in the *Scopes* window and select the **dut** instance.

   The btn[3:0] and led[3:0] signals will be displayed in the *Objects* window.



SIMULATION - Behavioral Simulation - Functional - sim_1 - lab1_tb

Scope | Sources

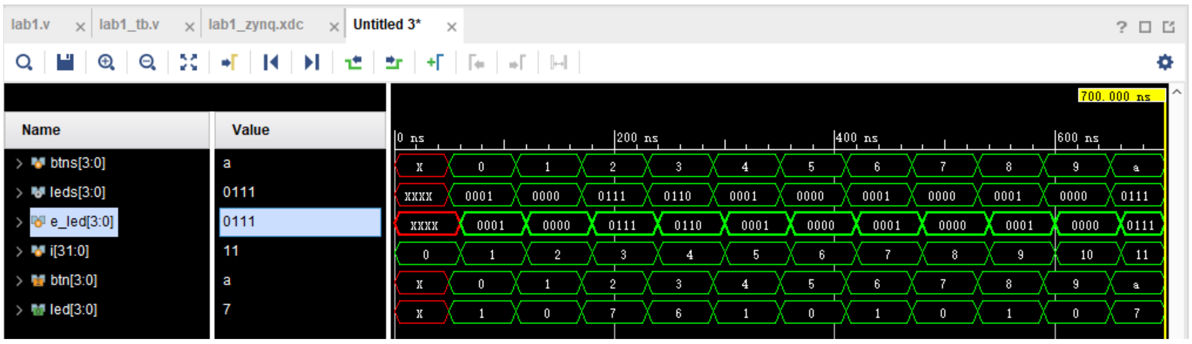
Name	Design U...	Block Type
lab1_tb	lab1_tb	Verilog M...
dut	lab1	Verilog M...
glbl	glbl	Verilog M...

Objects

Name	Value	D
btn[3:0]	2	A
led[3:0]	7	A

Selecting lower-level signals

2. Select **btn[3:0]** and **led[3:0]** and drag them into the waveform window to monitor those lower-level signals.
3. On the simulator tool buttons ribbon bar 500 ns, type 500 over in the simulation run time field, click on the drop-down button of the units field and select ns since we want to run for 500 ns (total of 700 ns), and click on the ( ) button. The simulation will run for an additional 500 ns.
4. Click on the *Zoom Fit* button and observe the output.

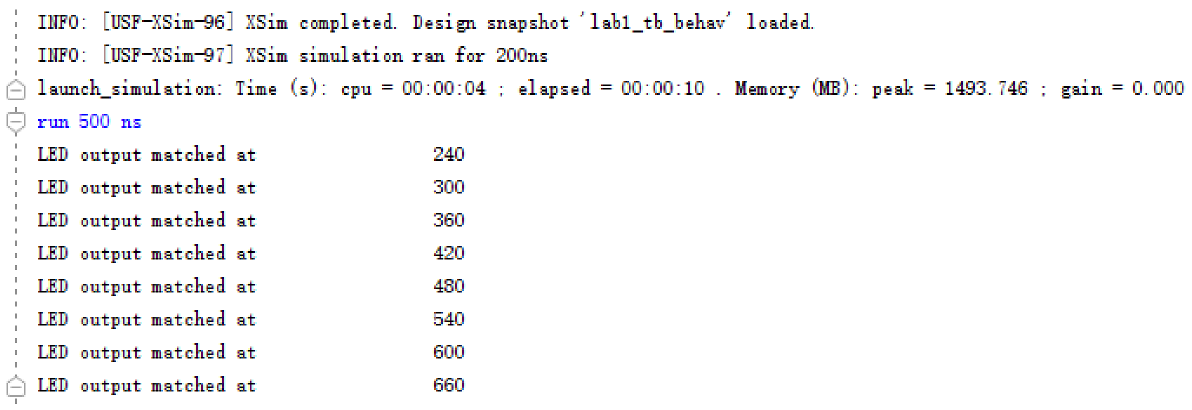


Name	Value
btns[3:0]	a
leds[3:0]	0111
e_led[3:0]	0111
i[31:0]	11
btn[3:0]	a
led[3:0]	7

X 0 1 2 3 4 5 6 7 8 9 a
XXXX 0001 0000 0111 0110 0001 0000 0001 0000 0001 0000 0111
XXXX 0001 0000 0111 0110 0001 0000 0001 0000 0001 0000 0111
0 1 2 3 4 5 6 7 8 9 10 11
X 0 1 2 3 4 5 6 7 8 9 a
X 1 0 7 6 1 0 1 0 1 0 7

Running simulation for additional 500 ns

Observe the Tcl Console window and see the output is being displayed as the testbench uses the $display task.



```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'lab1_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 200ns
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:10 . Memory (MB): peak = 1493.746 ; gain = 0.000
run 500 ns
LED output matched at          240
LED output matched at          300
LED output matched at          360
LED output matched at          420
LED output matched at          480
LED output matched at          540
LED output matched at          600
LED output matched at          660
```

Tcl Console output after running the simulation for additional 500 ns

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

4. Synthesize the Design

4.1 Synthesize the design with the Vivado synthesis tool and analyze the Project Summary output.

1. Click on **Run Synthesis** under the *SYNTHESIS* tasks of the *Flow Navigator* pane.

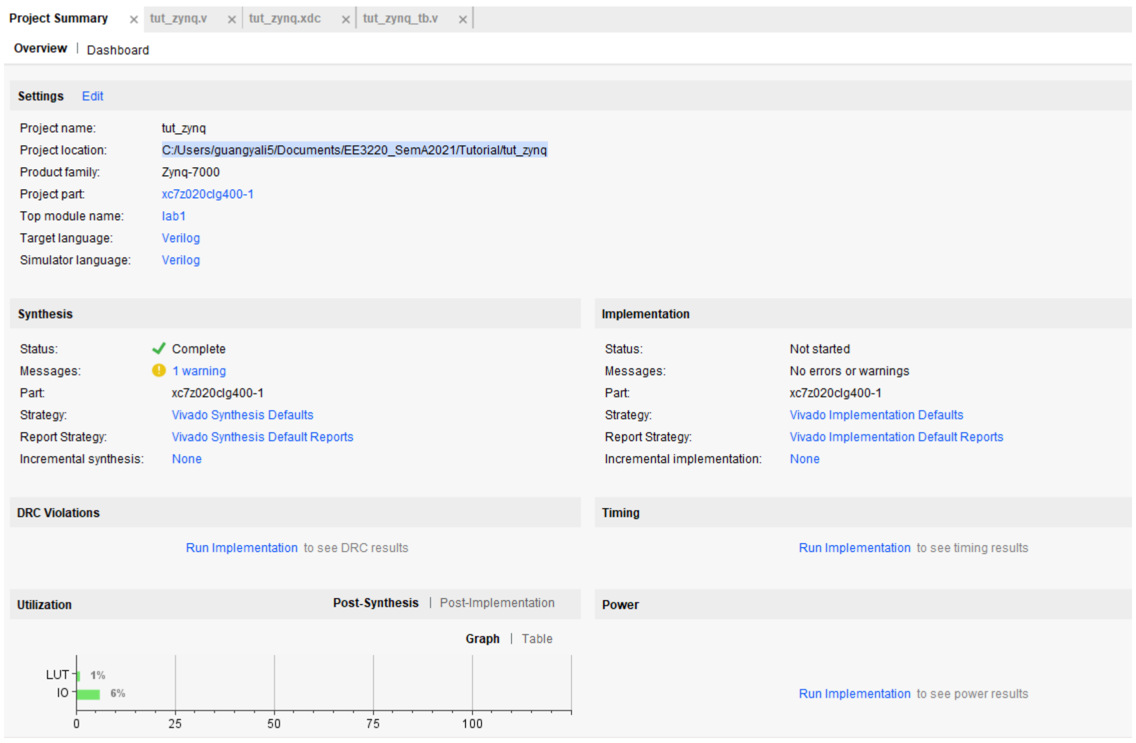
   The synthesis process will be run on the tut_zynq.v file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

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

   Click **Yes** to close the elaborated design if the dialog box is displayed.

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

   If you don't see the Project Summary tab then select **Window > Project Summary** or click the **Project Summary** icon Σ .



Settings

Project name:	tut_zynq
Project location:	C:/Users/guangyali5/Documents/EE3220_SemA2021/Tutorial/tut_zynq
Product family:	Zynq-7000
Project part:	xc7z020clg400-1
Top module name:	lab1
Target language:	Verilog
Simulator language:	Verilog

Synthesis

Status:	✓ Complete
Messages:	1 warning
Part:	xc7z020clg400-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z020clg400-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization

Power

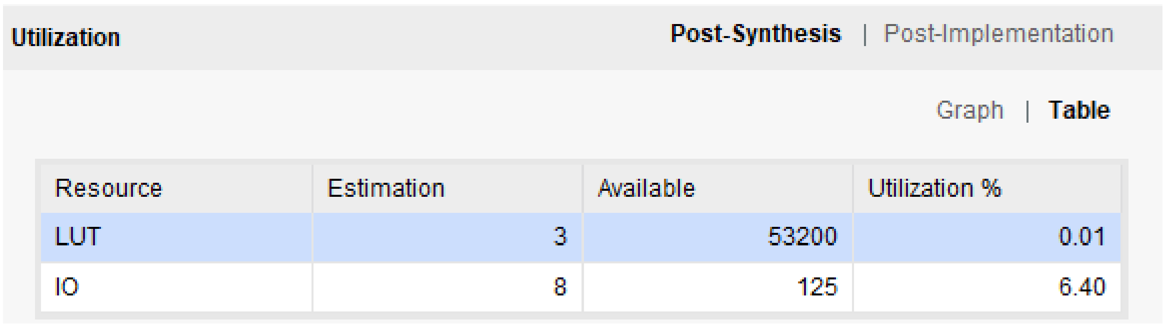
Run Implementation to see power results

Project Summary view

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

4. Click on the **Table** tab in the **Project Summary** tab.

   Notice that there are an estimated 3 LUTs and 8 IOs (4 input and 4 output) that are used.



Utilization

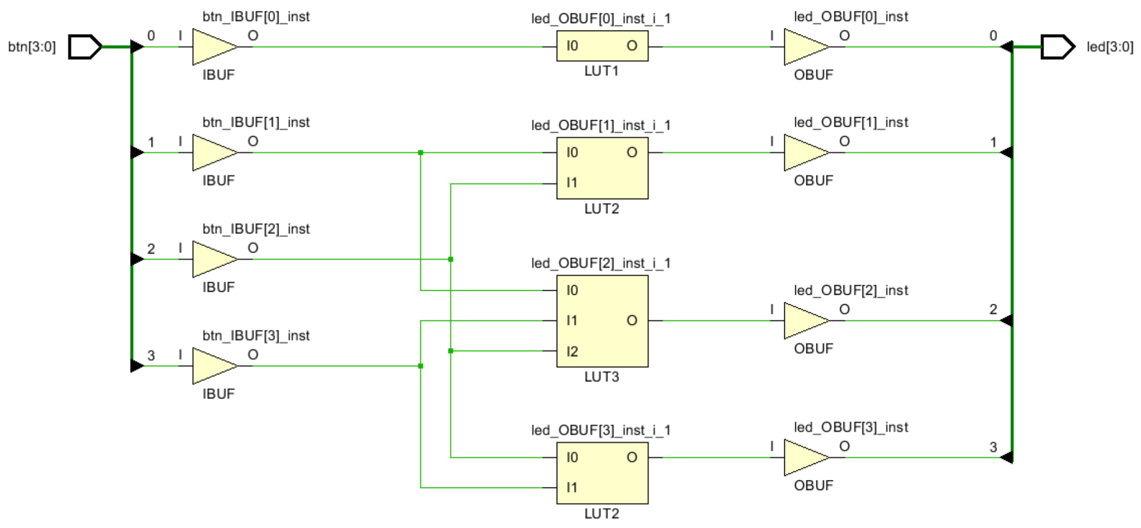
Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	3	53200	0.01
IO	8	125	6.40

Resource utilization estimation summary

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



Synthesized design's schematic view

Notice that IBUFs and OBUFs are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs (1 input is listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). Four gates in RTL analysis output are mapped onto four LUTs in the synthesized output.

5. Implement the Design

5.1 Implement the design with the Vivado Implementation Defaults settings and analyze the Project Summary output.

1. Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane.

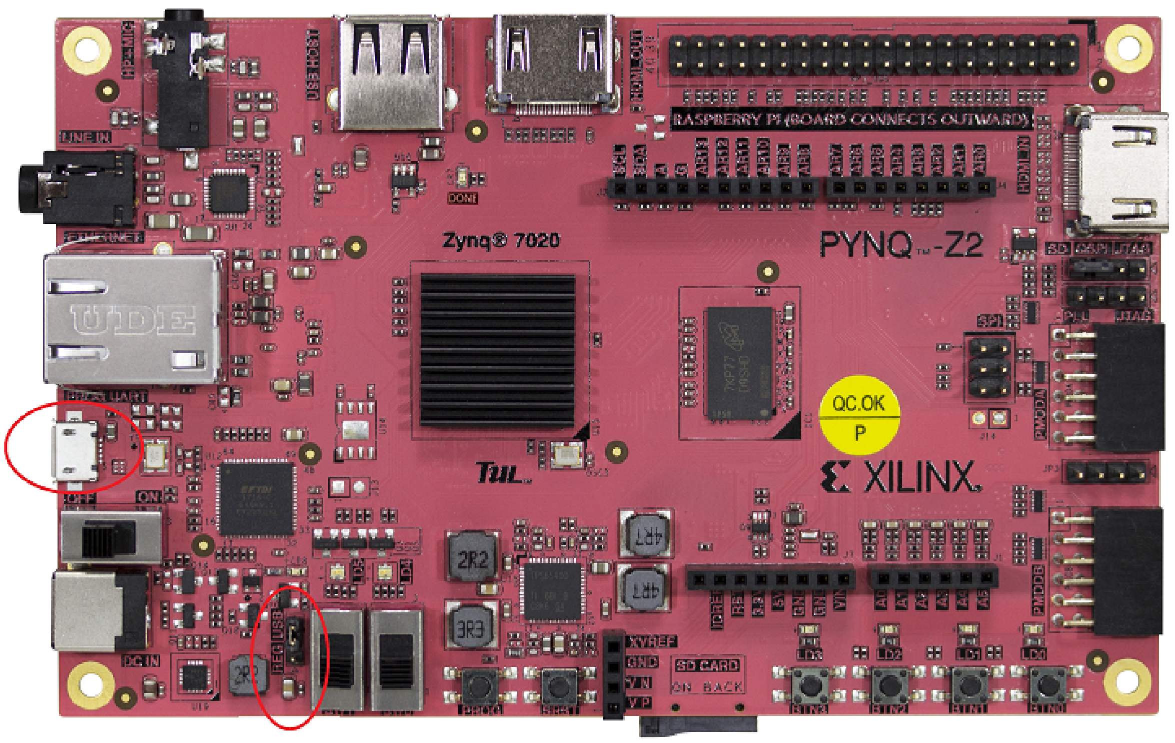
The implementation process will be run on the synthesized design. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.

6. Generate the Bitstream and Verify Functionality

6.1 Connect the board and power it ON. Generate the bitstream, open a hardware session, and program the FPGA.

1. Make sure that the Micro-USB cable is connected to the JTAG PROG connector.
2. The PYNQ-Z2 can be powered through USB power via the JTAG PROG.

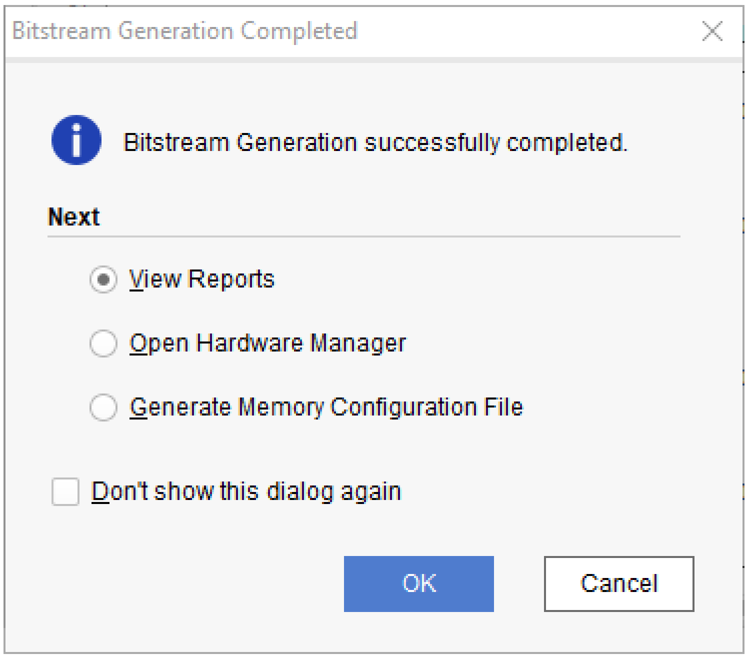
   Make sure that the board is set to use USB power.



Board connection for the PYNQ-Z2

3. Power **ON** the board.
4. Click on the **Generate Bitstream** entry under the *PROGRAM AND DEBUG* tasks of the *Flow Navigator* pane.

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



Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

- [✓ View Reports]
- Open Hardware Manager
- Generate Memory Configuration File

Don't show this dialog again

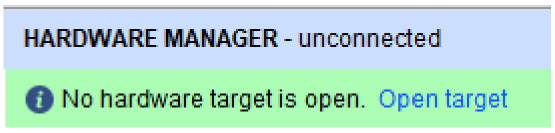
Bitstream generation

This process will have generated a **tut_zynq.bit** file under the **impl_1** directory in the **lab1.runs** directory.

5. Select the *Open Hardware Manager* option and click **OK**.

   The Hardware Manager window will open indicating "unconnected" status.

6. Click on the **Open target** link.



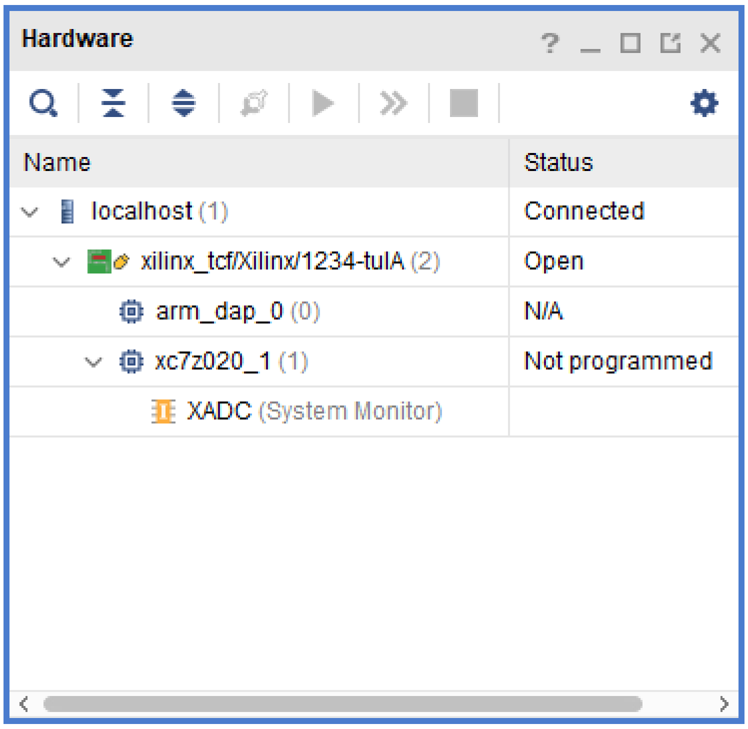
HARDWARE MANAGER - unconnected

No hardware target is open. Open target

Opening new hardware target

7. From the dropdown menu, click **Auto Connect.**

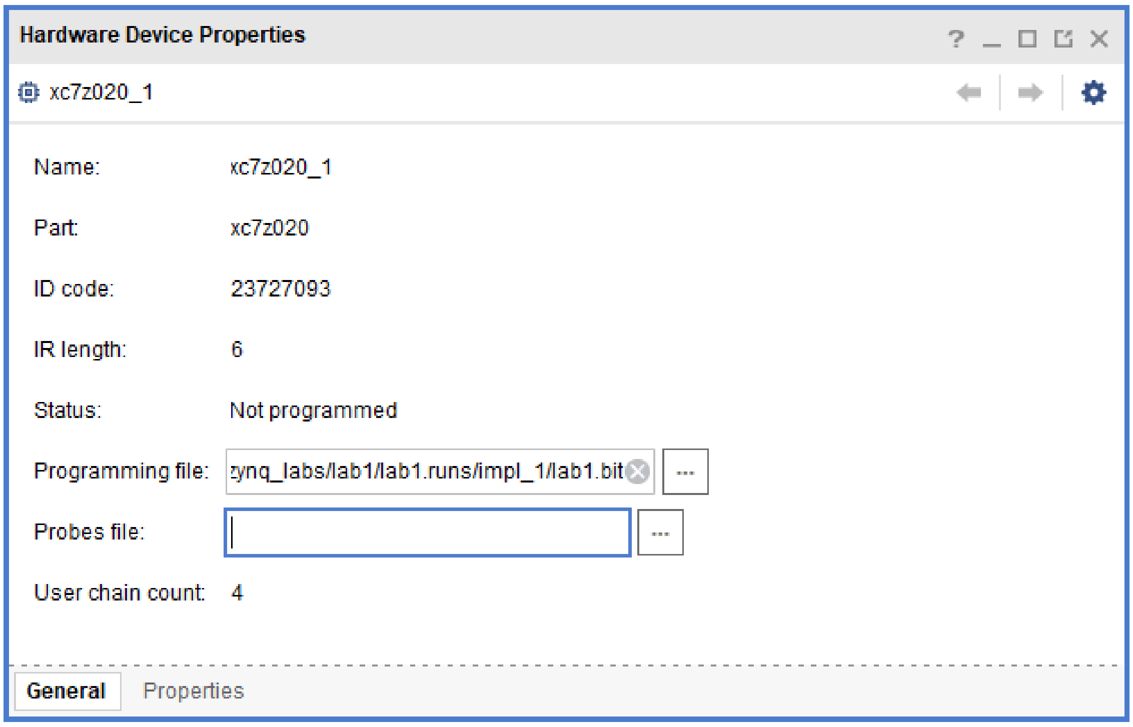
   The Hardware Session status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.


Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/1234-tulA (2)	Open
arm_dap_0 (0)	N/A
xc7z020_1 (1)	Not programmed
XADC (System Monitor)	

Opened hardware session

Select the device and verify that the lab1.bit is selected as the programming file in the General tab.

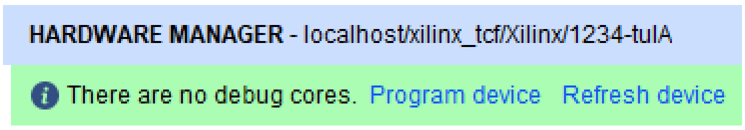

Hardware Device Properties

xc7z020_1

Name:	xc7z020_1
Part:	xc7z020
ID code:	23727093
IR length:	6
Status:	Not programmed
Programming file:	ynq_labs/lab1/lab1.runs/impl_1/lab1.bit
Probes file:	
User chain count:	4

Programming file

8. Click on the *Program device* link in the green information bar to program the target FPGA device. Another way is to right click on the device and select *Program Device.*


HARDWARE MANAGER - localhost/xilinx_tcf/Xilinx/1234-tulA

There are no debug cores. Program device Refresh device

Selecting to program the FPGA

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

   The DONE LED will lit when the device is programmed. You may see some other LEDs lit depending on switch positions.
10. Verify the functionality by flipping the switches and observing the output on the LEDs (Refer to the earlier logic diagram).
11. When satisfied, power **OFF** the board.
12. Close the hardware session by selecting **File > Close Hardware Manager.**
13. Click **OK** to close the session.
14. Close the **Vivado** program by selecting **File > Exit** and click **OK**.

Conclusion

The Vivado software tool can be used to perform a complete HDL based design flow. The project was created using the supplied source files (HDL model and user constraint file). A behavioral simulation using the provided testbench was done to verify the model functionality. The model was then synthesized, implemented, and a bitstream was generated. The timing simulation was run on the implemented design using the same testbench. The functionality was verified in hardware using the generated bitstream.

The dedicated hardware components could be integrated into the **IP cores**, which is also an important part in System-on-Chip design. The Vivado software tool could help us to integrate the IP cores with CPUs. In the following tutorials, we would focus on the integration of IP cores and CPU.