

City University of Hong Kong

Department of Electrical Engineering

EE 2000 – Lab Manual

**Lab 1: Getting familiar with the Vivado
Simulation Tool and BASYS 3 FPGA
Board**

NOTE:

- **There are seven checkpoints on Pages 2, 13, 18, 19, 21 and 23. For each checkpoint, please record your results either by taking photos, capturing screenshots, or registering some findings where it is appropriated.**
- **Show Checkpoints 2 and 5 to the lab tutor or demonstrator.**

Objectives:

- To get familiar with the VHDL and FPGA implementation procedure for digital circuit design;
- To get familiar with the Xilinx Vivado design and simulation packages;
- To practice the Artix®-7 FPGA chip programming.
- To learn how to access the I/O components on the BASYS 3 FPGA board.

Procedures

Part 1: Getting Familiar with BASYS3 Board

Before starting the experiment, we examine the BASYS 3 FPGA board shown in the Figure 1. The BASYS 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®.

CHECKPOINT 1: Give the full name and functionality of the following terms: VGA, XADC, UART, JTAG, Pmod? Please search the relevant materials on the Internet.

Before starting the programming of the FPGA chip on the Basys 3, we need to setup the hardware connections as follows:

1. Connect the micro USB port, PROG (J1), using a MicroUSB cable, to a PC. (Please ask for help if you cannot find this cable.)
2. Turn the board's power switch (Number 15 in Fig.1) to the ON position.

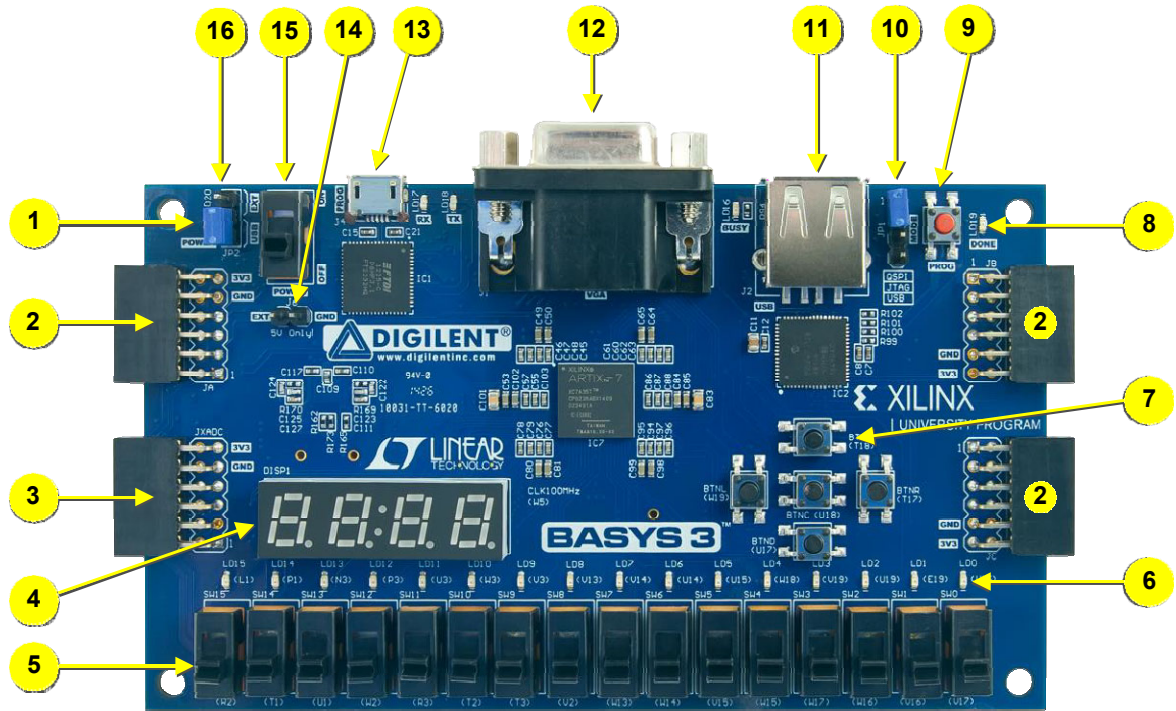


Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Push buttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Table 1. Basys 3 Callouts and component descriptions.

Part 2: Vivado Tour

This section guides to the use of the Xilinx Vivado design and simulation package. A typical design flow consists of creating a Vivado project, creating a model, creating user constraint files, running the behavioral simulation, synthesizing the design, implementing the design, generating the bitstream, and finally verifying the functionality with the FPGA board by downloading the generated bitstream file to the board.

Following the procedures below, you should be able to perform a simple digital circuit using Hardware Description Language (HDL) through the Xilinx Vivado software package. The

major tasks of a typical design include: 1) creating models, 2) producing user constraint files, 3) launching a new Vivado project, 4) importing the created models, 5) assigning created constraint files, 6) optionally running behavioral simulation, 7) synthesizing the design, 8) implementing the design, 9) generating a bitstream (.bit) file for programming the FPGA chip, and finally, 10) verifying the functionality in the hardware by downloading the generated bitstream file. The design flow is shown below.

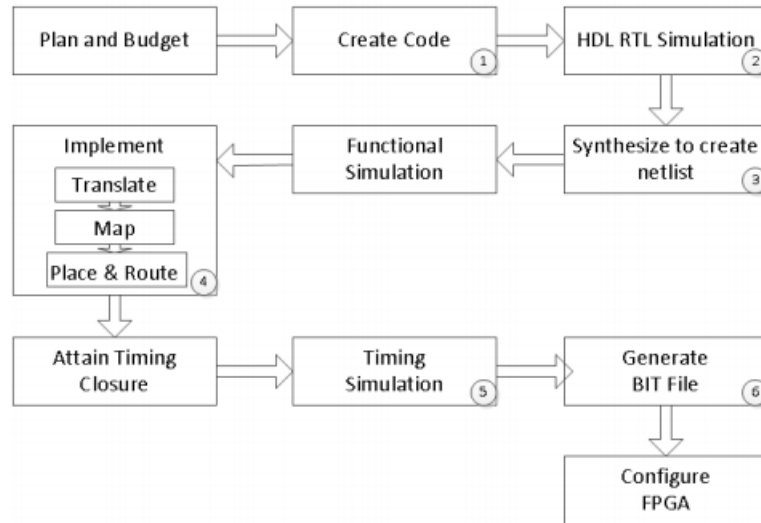
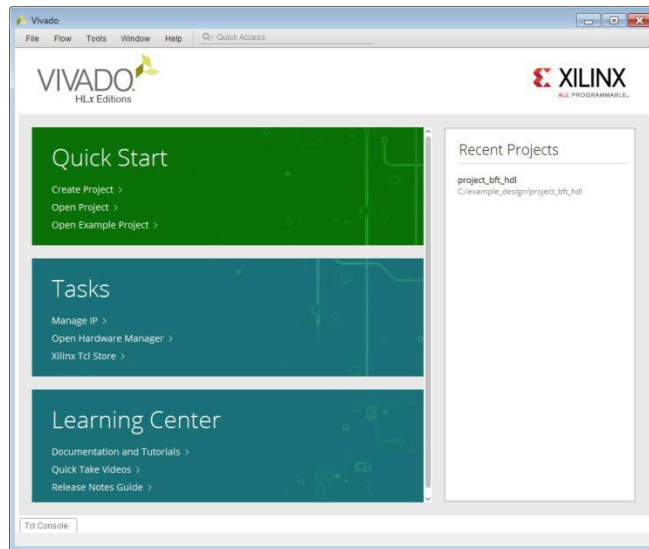


Figure 2. Flow of HDL design.

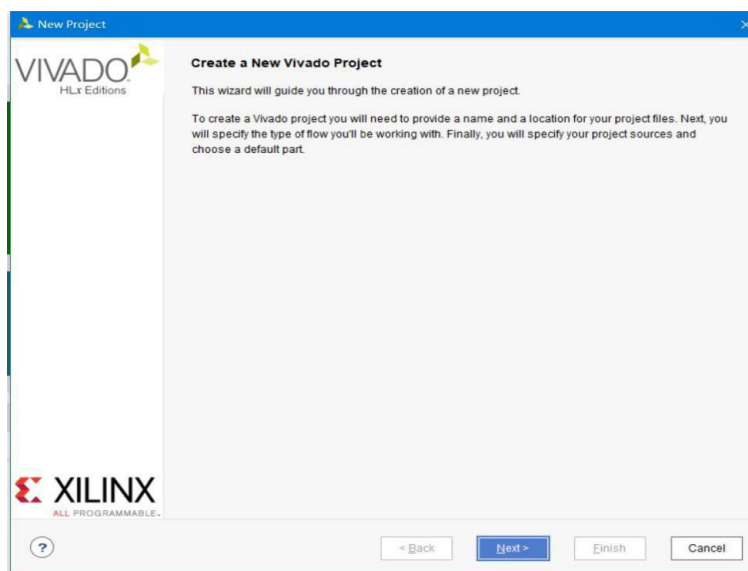
Running Vivado

1. Login to one of the lab machines. You should use the Lab's windows password shown on the whiteboard to log in.
2. Launch the Vivado software and create a new project targeting the Artix®-7 FPGA Chip – **xc7a35ticpg236-1L** and using the VHDL.

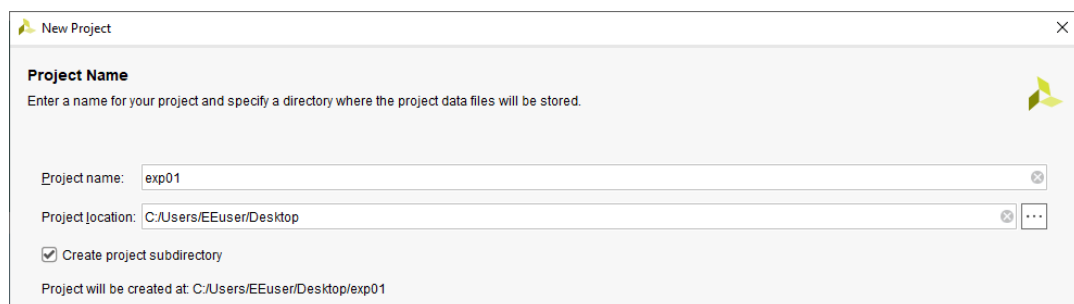
Once you launch the **Vivado** software, the getting started page will appear as follows.

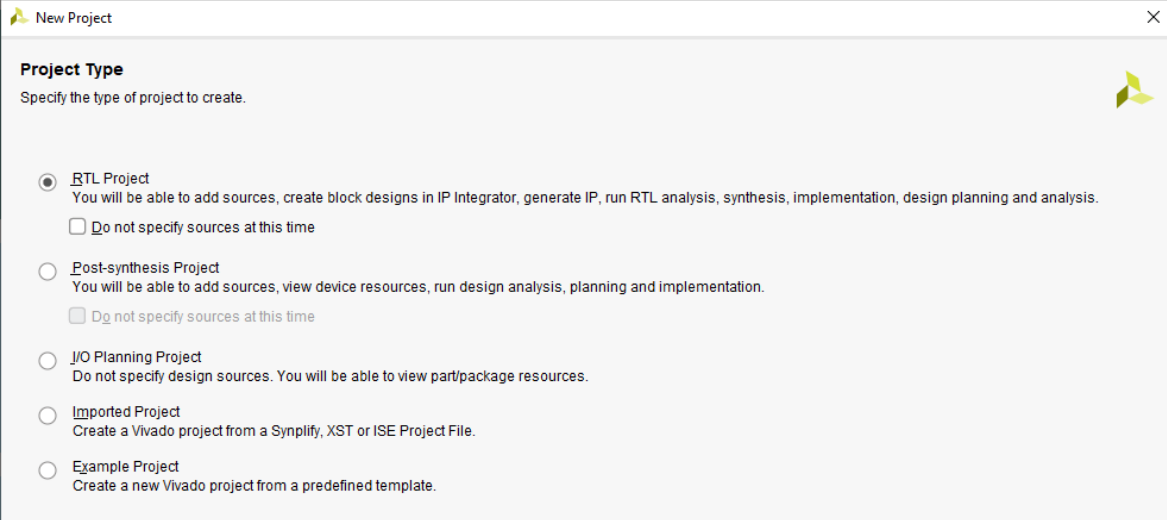


- i. Choose **Create Project** to open the New Project Wizard. Click **Next**.



- ii. Enter a **Project name** and **Project location**. Click **Next**. Select **RTL Project**, Click **Do not specify sources at this time** and click **Next**.



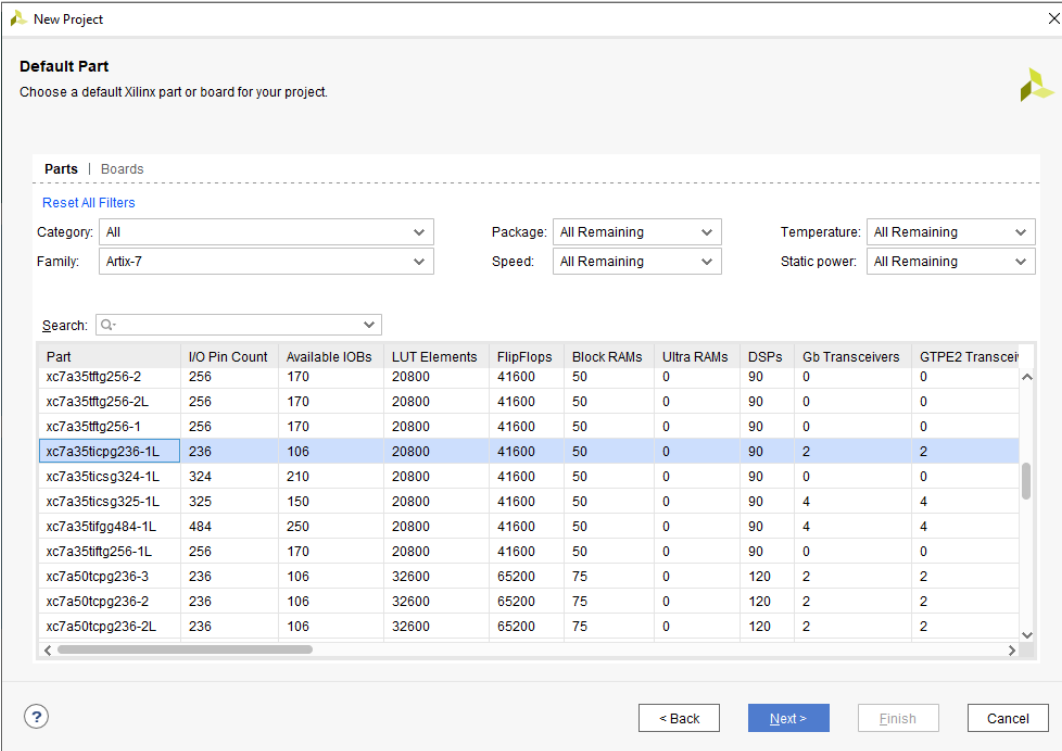


New Project

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.

- iii. Select Parts as **Artix-7 xc7a35t1cp236-1L** and click **Next**. Alternatively, you can select the chip model. Then click **Finish** to create the project.



New Project

Default Part
Choose a default Xilinx part or board for your project.

Parts | Boards

[Reset All Filters](#)

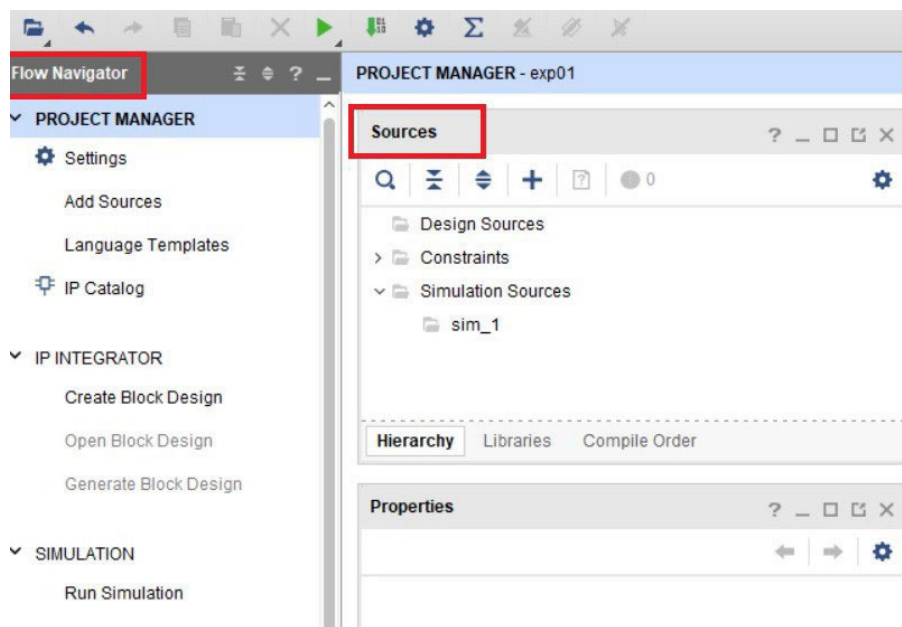
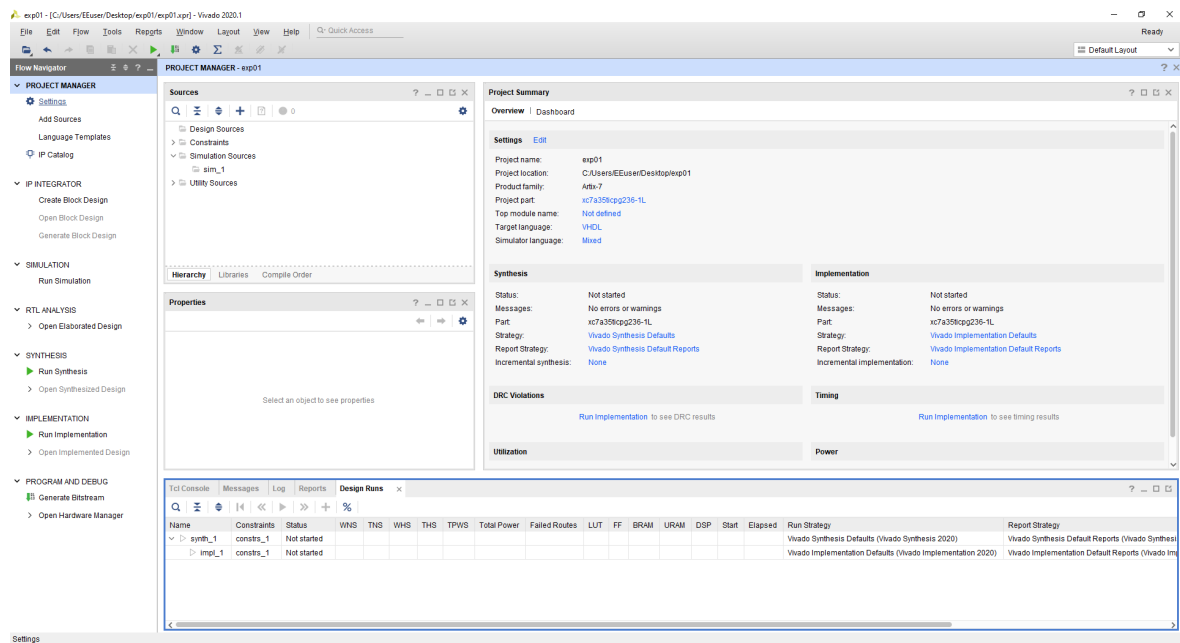
Category: All Package: All Remaining Temperature: All Remaining
Family: Artix-7 Speed: All Remaining Static power: All Remaining

Search: Q-

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers
xc7a35t1fg256-2	256	170	20800	41600	50	0	90	0	0
xc7a35t1fg256-2L	256	170	20800	41600	50	0	90	0	0
xc7a35t1fg256-1	256	170	20800	41600	50	0	90	0	0
xc7a35t1cp236-1L	236	106	20800	41600	50	0	90	2	2
xc7a35t1csg324-1L	324	210	20800	41600	50	0	90	0	0
xc7a35t1csg325-1L	325	150	20800	41600	50	0	90	4	4
xc7a35t1fg484-1L	484	250	20800	41600	50	0	90	4	4
xc7a35t1fg256-1L	256	170	20800	41600	50	0	90	0	0
xc7a50t1cp236-3	236	106	32600	65200	75	0	120	2	2
xc7a50t1cp236-2	236	106	32600	65200	75	0	120	2	2
xc7a50t1cp236-2L	236	106	32600	65200	75	0	120	2	2

Navigation: ? < Back Next > Finish Cancel

- iv. You will see a screen that looks like this. You can find Flow Navigator on the left side with a few click buttons. Besides, there is a **Sources** window showing your project hierarchy.

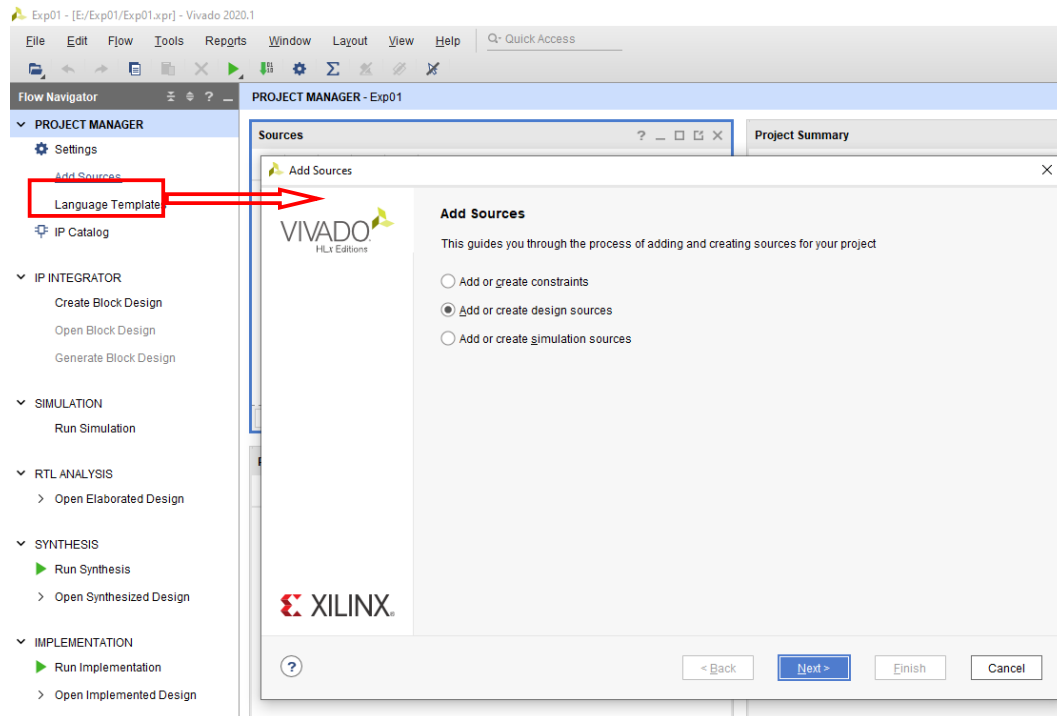


3. Create and edit two VHDL Files:

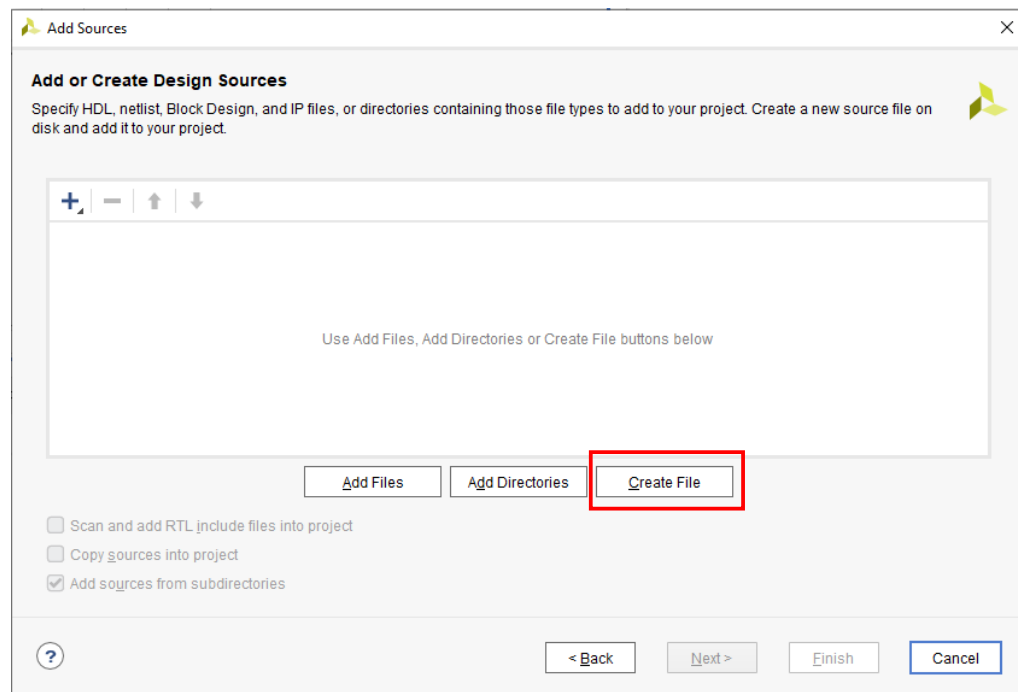
- exp01.vhd (Design model source HDLfile)

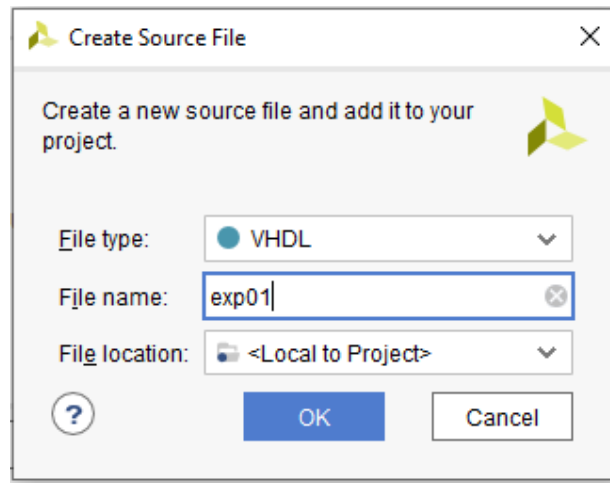
- exp01_tb.vhd (Design Testbench HDL file)

- i. In the **Sources** window, right click **Design Sources** and click **Add Sources**.
Alternatively, you can click **Add Sources** from **Flow Navigator**. You will open a new window.

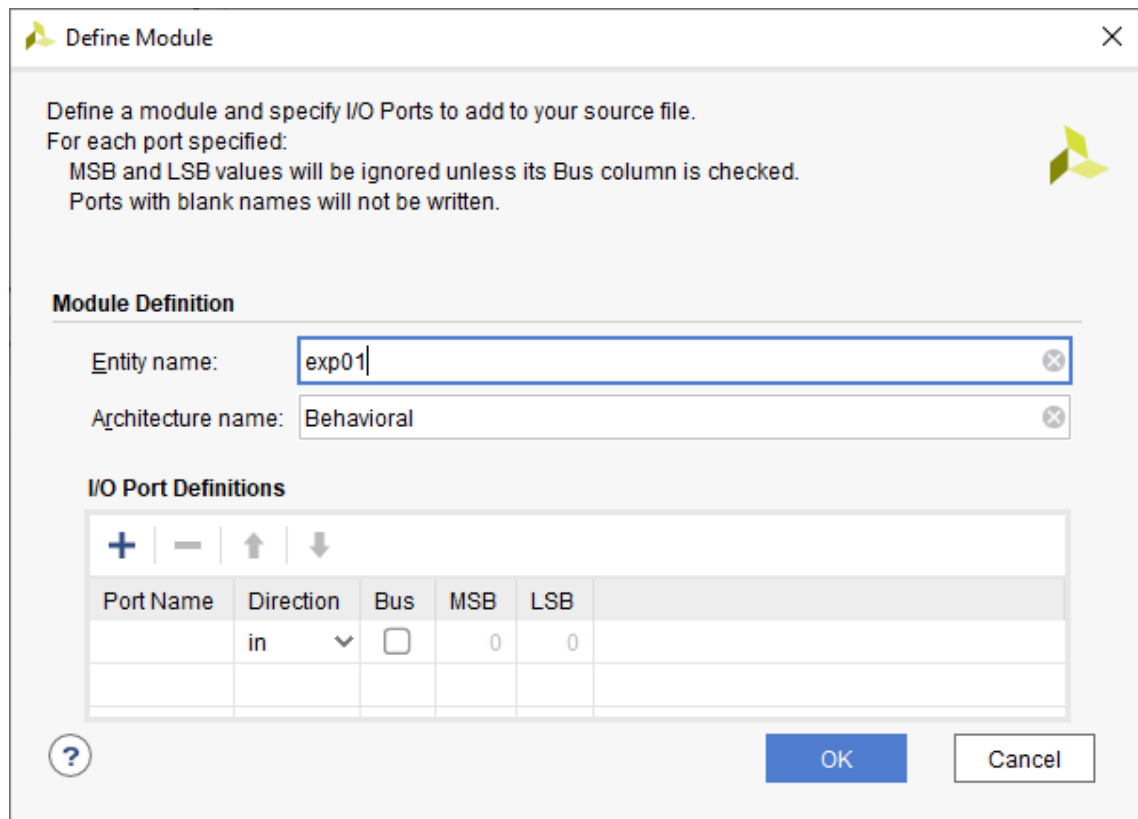


- ii. Select **Add or create design sources** and click **Next**. Select **Create File**. Make sure the **File type** is **VHDL**, and enter the **File name**: **exp01**. Click **OK**, and then click **Finish**.

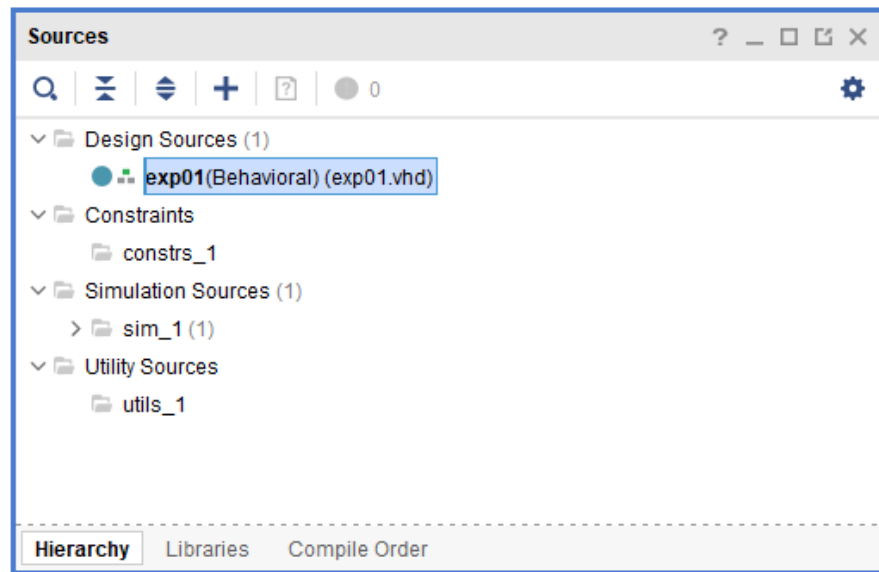




In the new window **Define Module**, click **OK** directly, then click **Yes**.



iii. Find your newly created file.

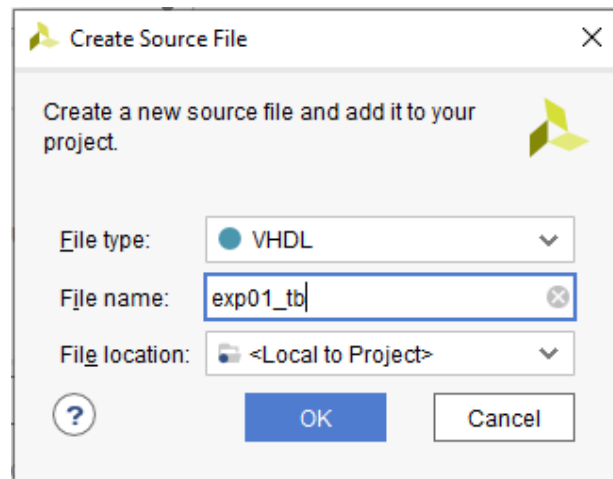
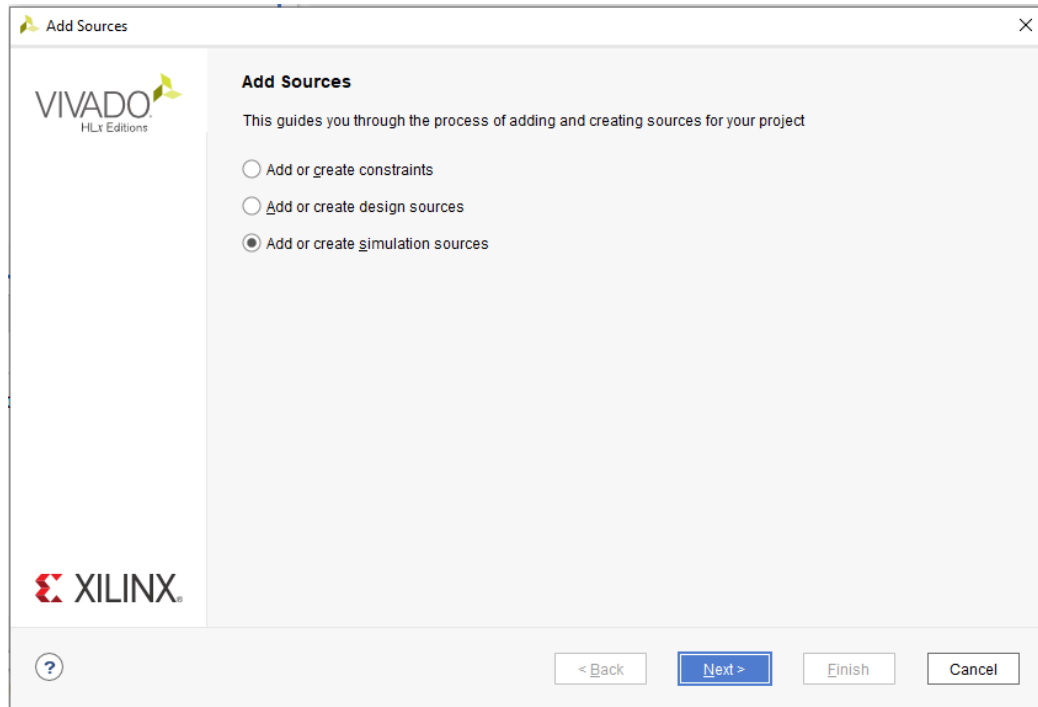


Edit exp01.vhd, and then save it. You are required to add your own code to the file.

```
entity exp01 is
    Port (
        sw: IN BIT;
        led: OUT BIT
    );
end exp01;

architecture Behavioral of exp01 is begin
-- Put your own code here
end Behavioral;
```

- iv. Create a simulation file in the testbench. Right click on **Simulation Sources** in the source window and select **Add Sources**. Make sure you select **Add or create simulation sources** and click **Next**. Select **Create File**, choose **File Type** as VHDL, and enter **File name:** exp01_tb. The operation is similar to creating exp01.vhd.



- v. Edit exp01_tb.v as follows and save it.

```

entity exp01_tb is
end exp01_tb;

architecture Behavioral of exp01_tb is

-- component declaration
component exp01 is
    Port (

        sw: IN BIT;
        led: OUT BIT
    );
end component;

-- signal declaration
signal sw: BIT;
signal led: BIT;
begin
exp01_inst: exp01

    port map (

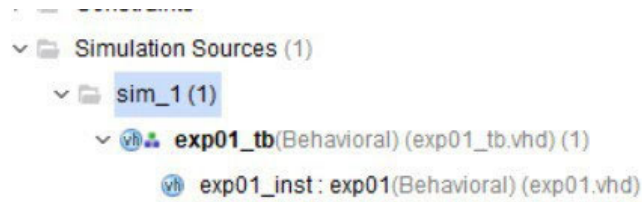
        sw =>
        sw, led
        => led
    );

simgen: process
begin
    sw <= '0';

    wait for 50ns;
    sw <= '1';
    wait for
    100ns; sw <=
    '0';
    wait for
50ns; end
process;
end Behavioral;

```

After saving the file, you will find that the hierarchy has been changed since exp01_tb instantiates exp01.



4. Run simulation.

- i. Find and click **Run Simulation** in Flow Navigator, and Select **Run Behavioral Simulation**. The default simulator is XSim, and you can see the simulation waveform in Untitled 1. The example design is simple; you will learn more simulation skills later.

Open Block Design

Generate Block Design

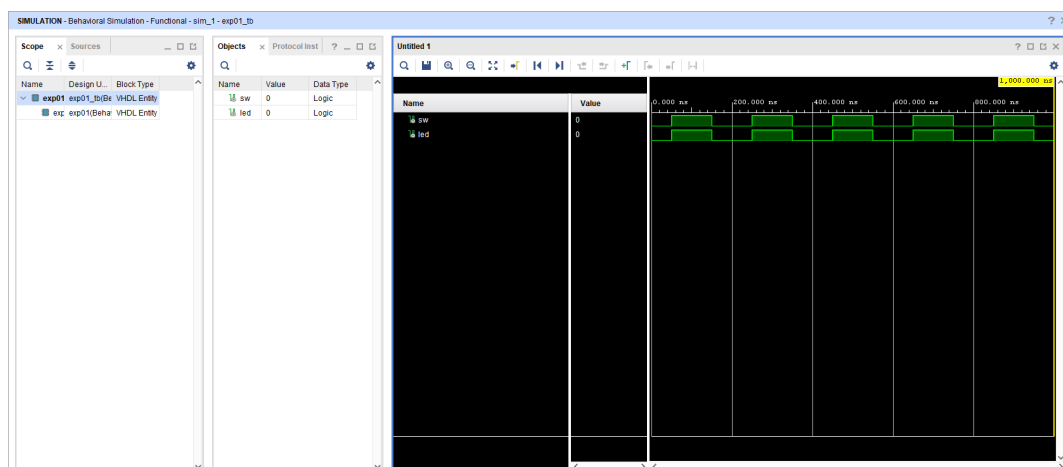
▼ SIMULATION

Run Simulation

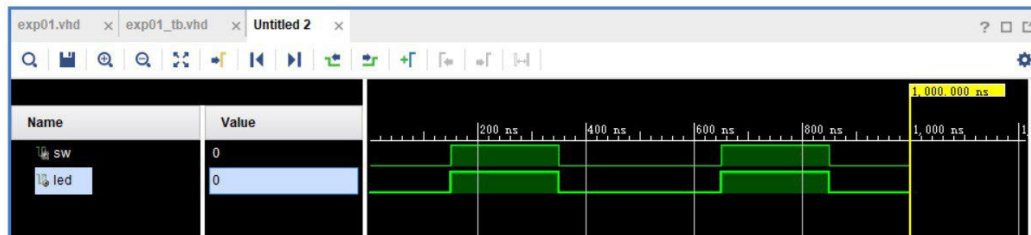
▼ RTL ANALYSIS

> Open Elaborated Design

You can use the three buttons in the red box to scale your waveform.

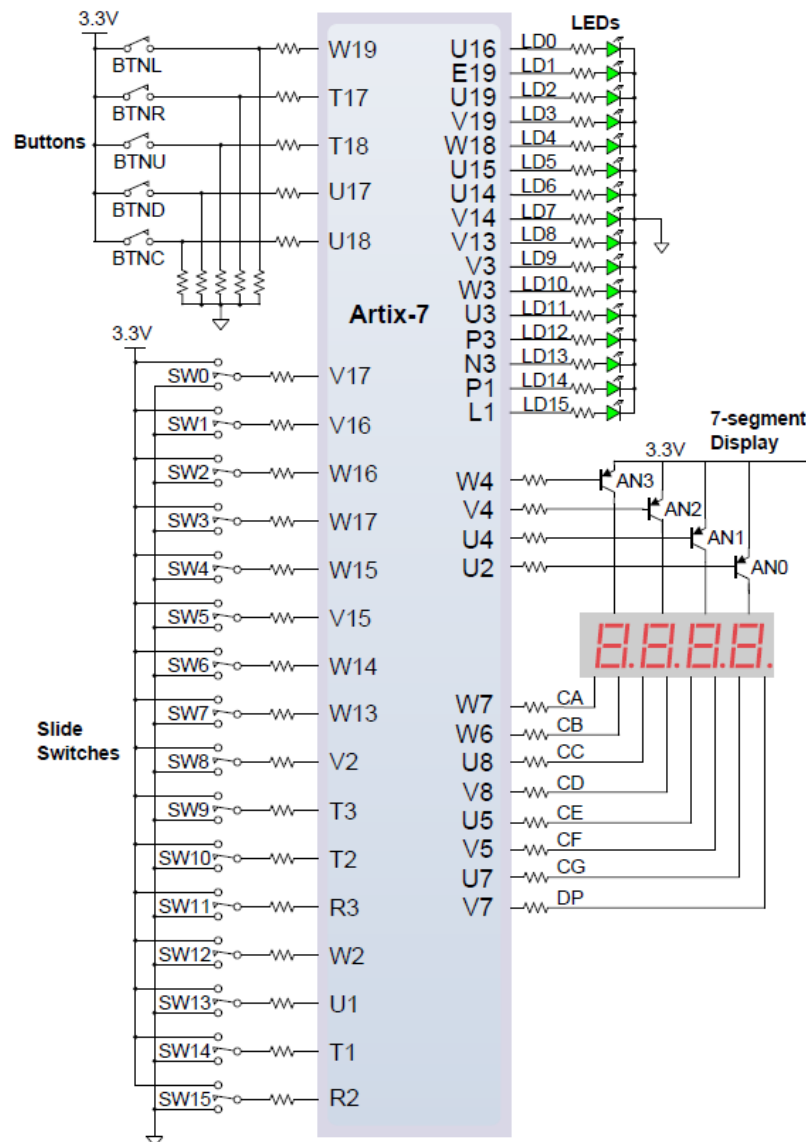


CHECKPOINT 2: How do you generate the waveform, as shown in Figure 3, by modifying the exp01_tb.vhd file? (Hint: Please note that the duration of 1 and 0, as shown in the figure, have changed.)



5. Synthesize, implement the design, and generate bitstream file.

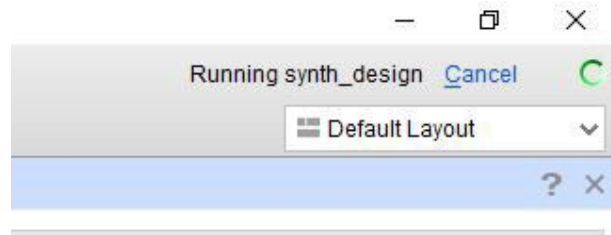
At first, the pins of the FPGA chip connected with the above peripheral are shown below.



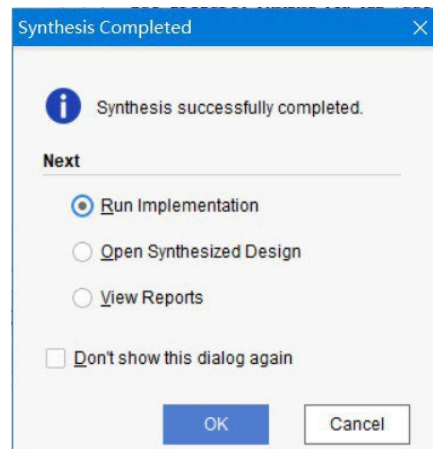
- i. Close the Simulation Window, right click **Constraints**, and select **Add Sources**. Click **Next** and **Create File**. Enter the **File name** as Basys3_board.xdc. Please make sure that the file type is .xdc. Click **OK**, and then click **Finish**.
- ii. Edit Basys3_board.xdc

```
set_property PACKAGE_PIN V17 [get_ports {sw}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw}]
set_property PACKAGE_PIN U16 [get_ports {led}]
set_property IOSTANDARD LVCMOS33 [get_ports {led}]
```

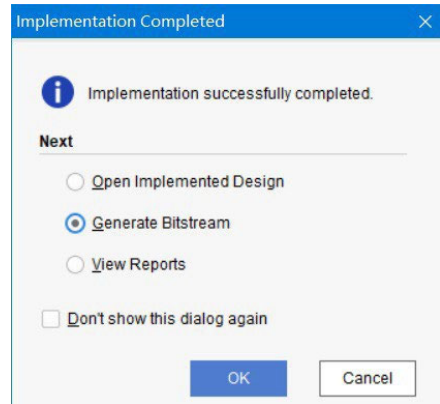
- iii. Click **Run Synthesis**, it may take some time, and you can see the process in the top right corner.



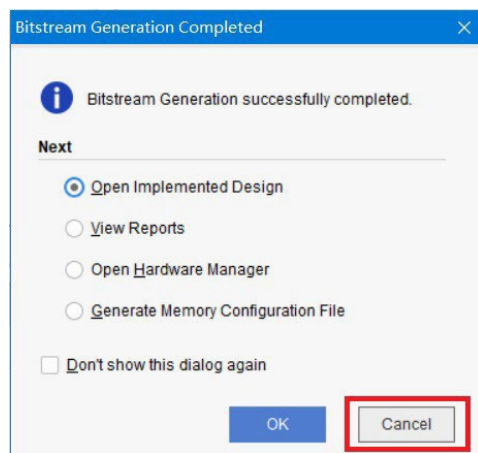
After synthesis is finished, you will see the following dialog, select **Run Implementation** and click **OK**. This process may take a few minutes.



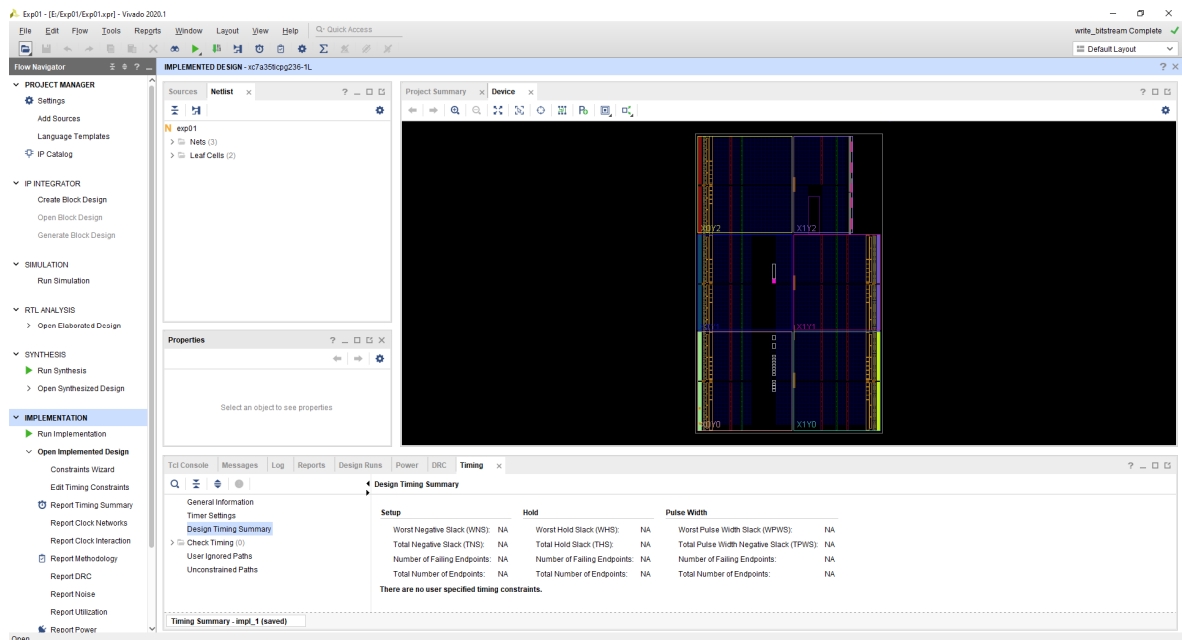
After implementation is finished, you will see the following dialog, select **Generate Bitstream** and click **OK**.



Finally, the bitstream is generated. You will see the following dialog box, click “Cancel”.

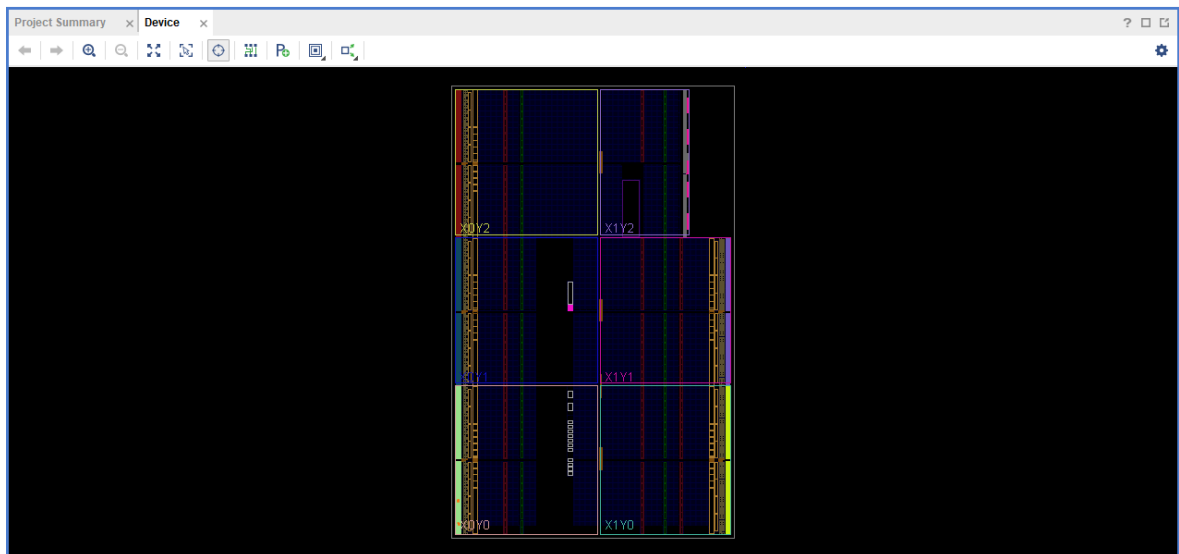


- iv. Click **Open Implementation**. After the synthesis is finished, you will see the occupation of the device in Vivado.

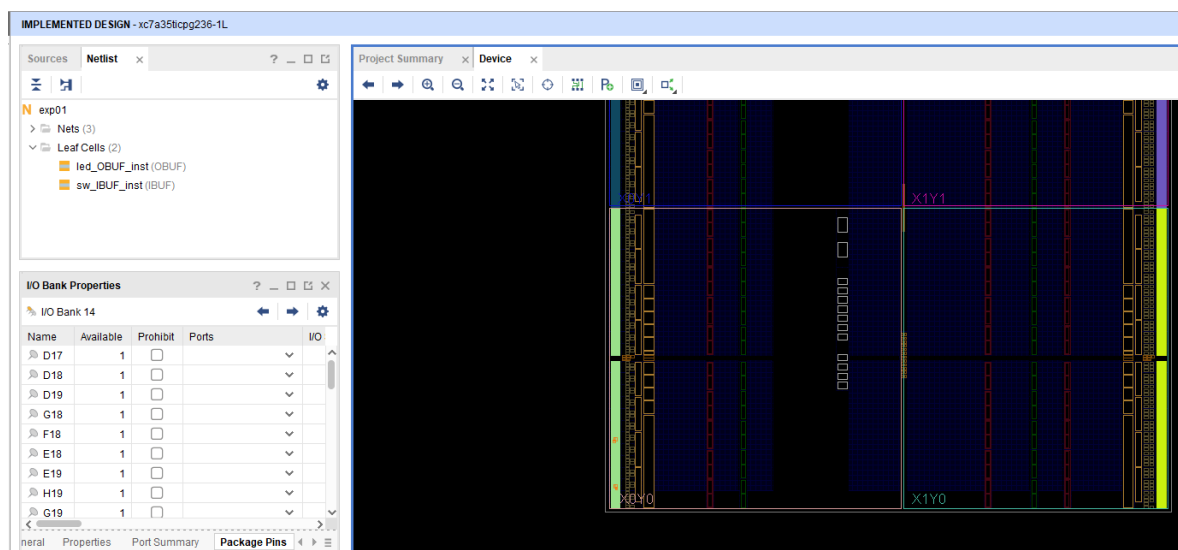


Press “ctrl” on the keyboard and turn the “scroll wheel” of the mouse. Then we

can zoom in/ zoom out the floorplan as shown below.



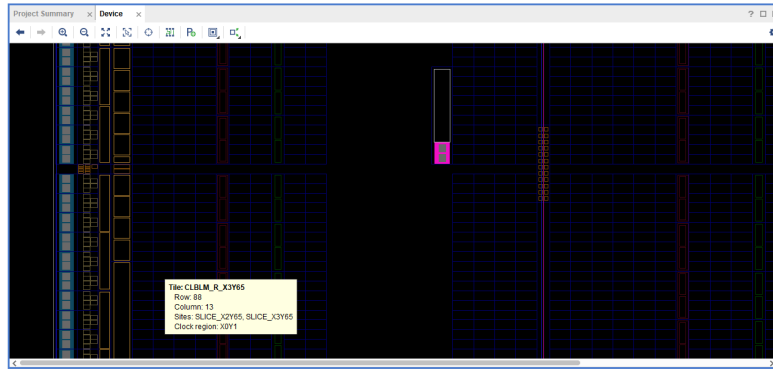
Zoom in the floorplan and scroll to the bottom left corner, as shown below.



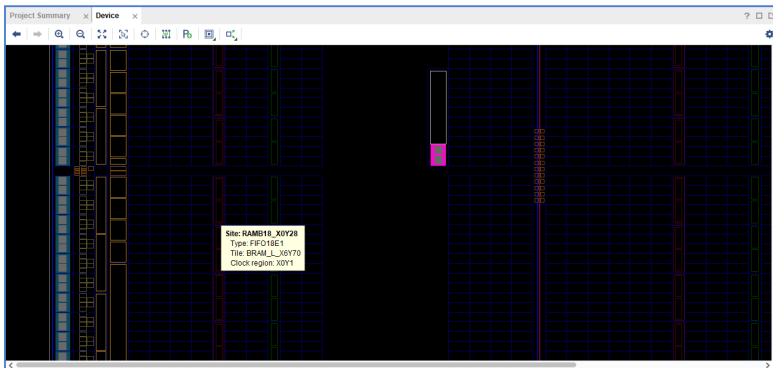
Click the yellow bar, and we can find “I/O Bank Properties” in the left panel; we can see the detailed information of the selected component.

Checkpoint 3: Try to select the blue box, pink box, and green box. And find out what these components are.

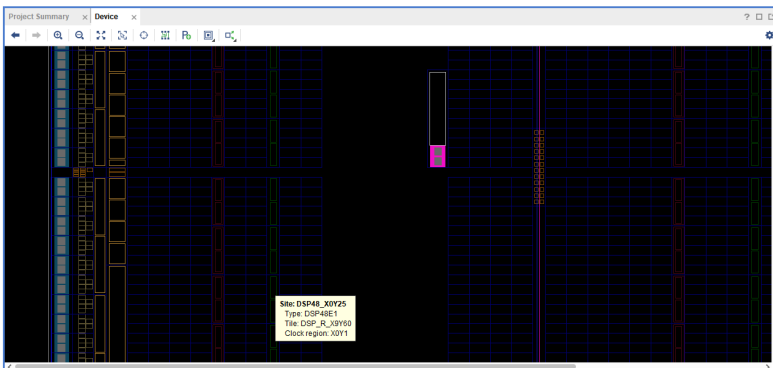
Blue box



Pink box

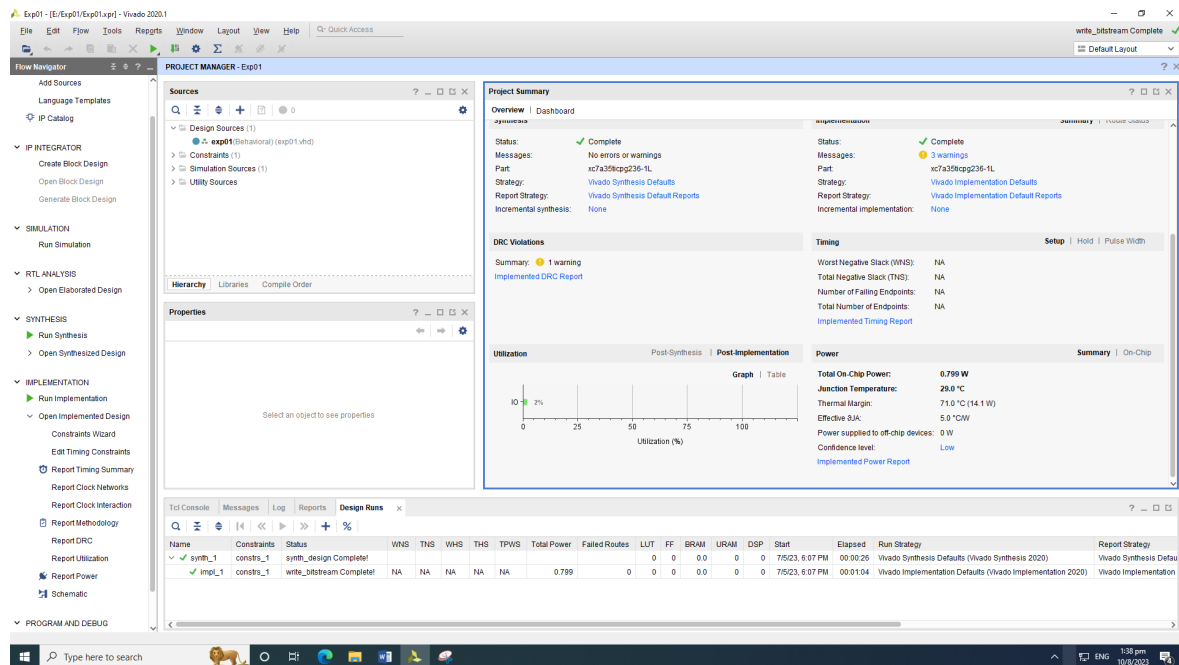


Green box



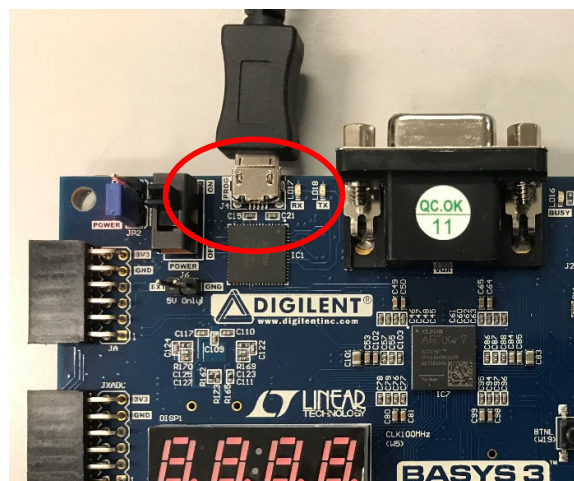
Checkpoint 4: Please have a look at the synthesized results and the report.
How many LUTs are being used? Why? How about IOB?

Then, click report utilization in the flow navigator under implementation to get the report.



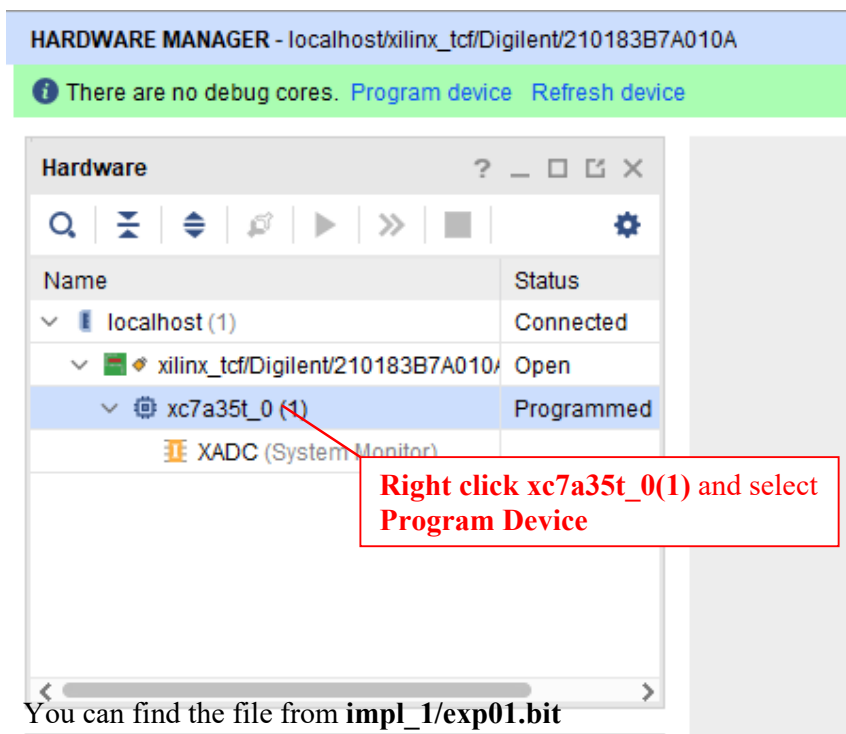
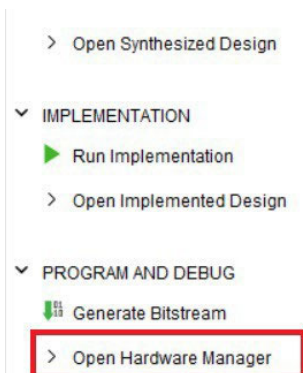
Besides the utilization, the power consumption and temperature are also shown in the picture.

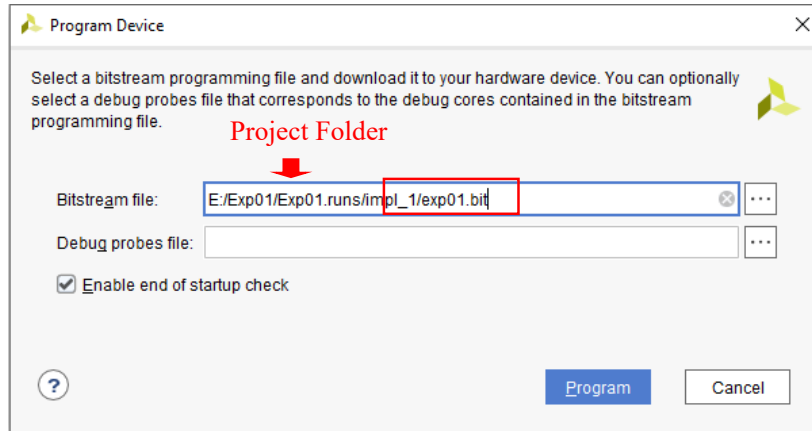
- v. Make sure the Basys 3 board has been correctly connected to the PC (The USB cable is connected to the PC).



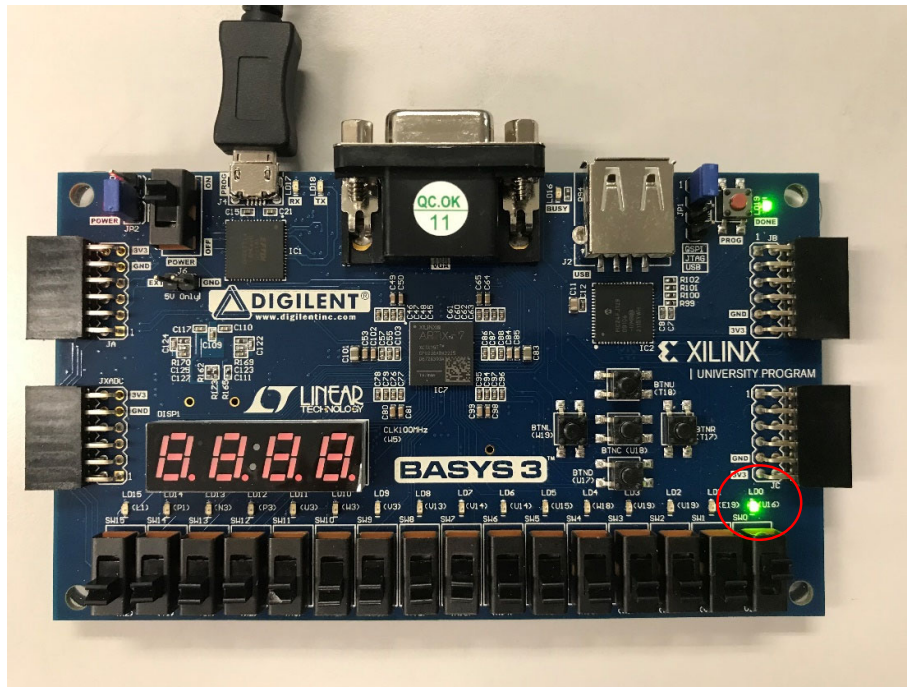
- vi. Open **Hardware Manager**, Click **Open Target**, and select **Auto Connect**. If Basys 3 Board is correctly connected to the PC, you will see the FPGA chip from the

Vivado software. **Right-click xc7a35t_0(1)** and select **Program Device**. Then click **Program**.





- vii. Turn on one of the LEDs, LED0.



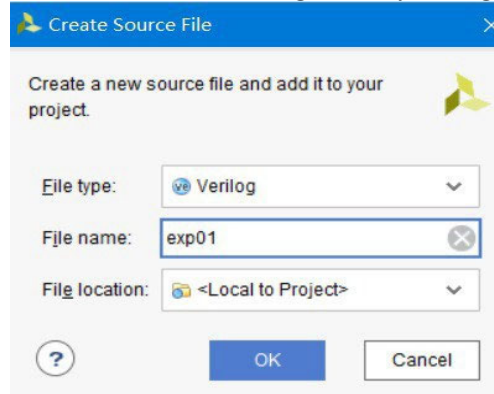
CHECKPOINT 5: Show the results to the demonstrator in your lab session.

CHECKPOINT 6: Discuss how you can implement a configurable logic design that can increase the utilization percentage of the FPGA chip (e.g., 10% and more).

6. Create a Verilog design

The Verilog design flow is similar, but there are some differences that need to be noticed.

- i. When creating the design file and testbench file, remember to choose the File type as **Verilog**. Then you can create two files, **exp01.v**, and **exp01_tb.v**.
- ii. If you are unfamiliar with Verilog HDL syntax, google it.



Edit exp01.v

```
`timescale 1ns / 1ps

module exp01 (
    input sw,
    output led
);

    -- Put your own code here

endmodule
```

Edit exp01_tb.v

```
`timescale 1ns / 1ps

module exp01_tb();
```

```

reg switch;

wire led;

exp01 exp01_inst (.sw(switch), .led(led));

initial

begin

    #0 switch = 1'b0; // # means a time delay, so at 5ns, switch will be set to 1

    #5 switch = 1'b1;

    #10 switch = 1'b0;

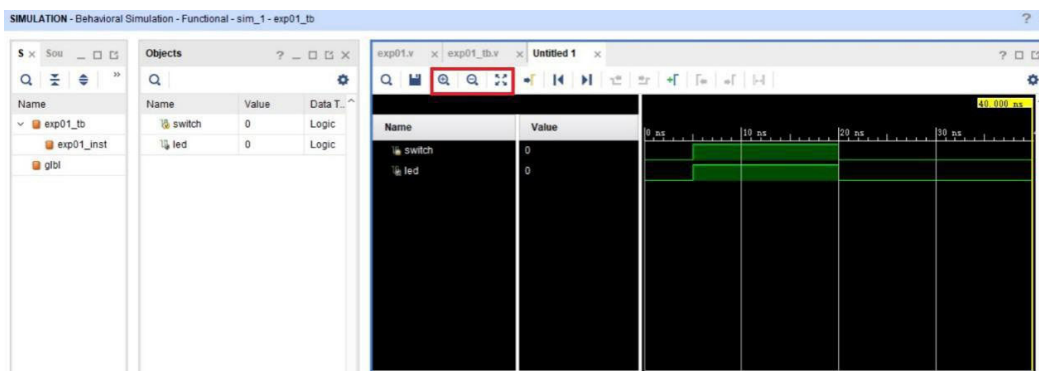
    #15 $stop; // $stop will stop the simulation

end

endmodule

```

iii. You can find the **simulation waveform** like this:



CHECKPOINT 7: What are the basic differences between VHDL and Verilog syntax? Take a screenshot of your generated waveform using the Verilog.

Supplementary Materials:

VHDL Code Structure- Three Steps:

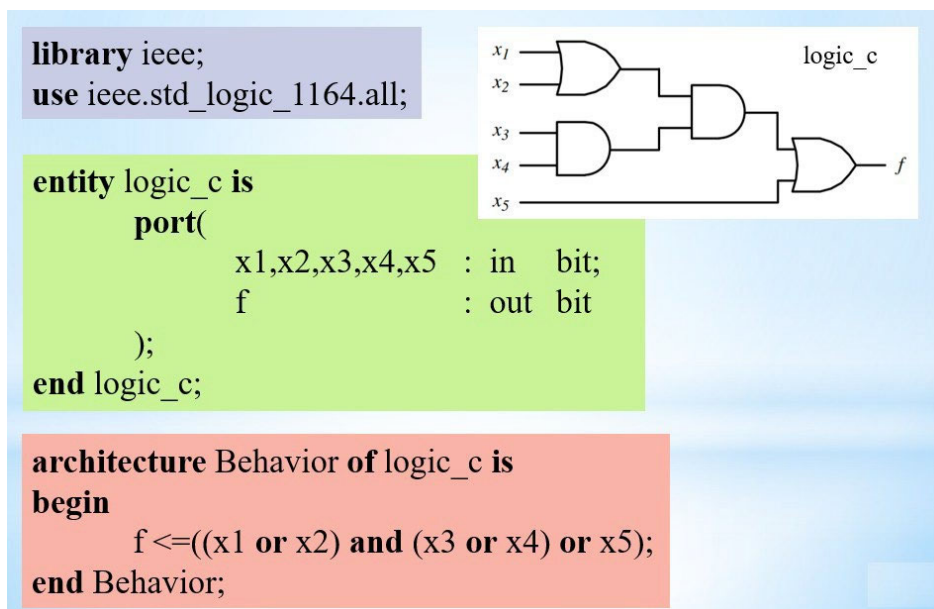
- i. Library Declaration
- ii. Entity Declaration
- iii. Architecture Declaration

Library declaration: locate the system library, IEEE Standard library is often included in the VHDL code. You can create your own library.

Entity declaration: defines the external interface to the current design [input (port) / output (port)].

Architecture declaration: describes the internal circuit of the corresponding entity [logic circuit].

Example:



VHDL Simulation:

Testbench

- Performing simulation to verify through simulation waveform;
- Provide a stimulus for the UUT (unit under test) to check the

output. Template:

```

entity circuit_tb is
-- port (empty) (not required for simulation)
end entity circuit_tb;

architecture behavioral of circuit_tb is

    component declarations

    signal declarations

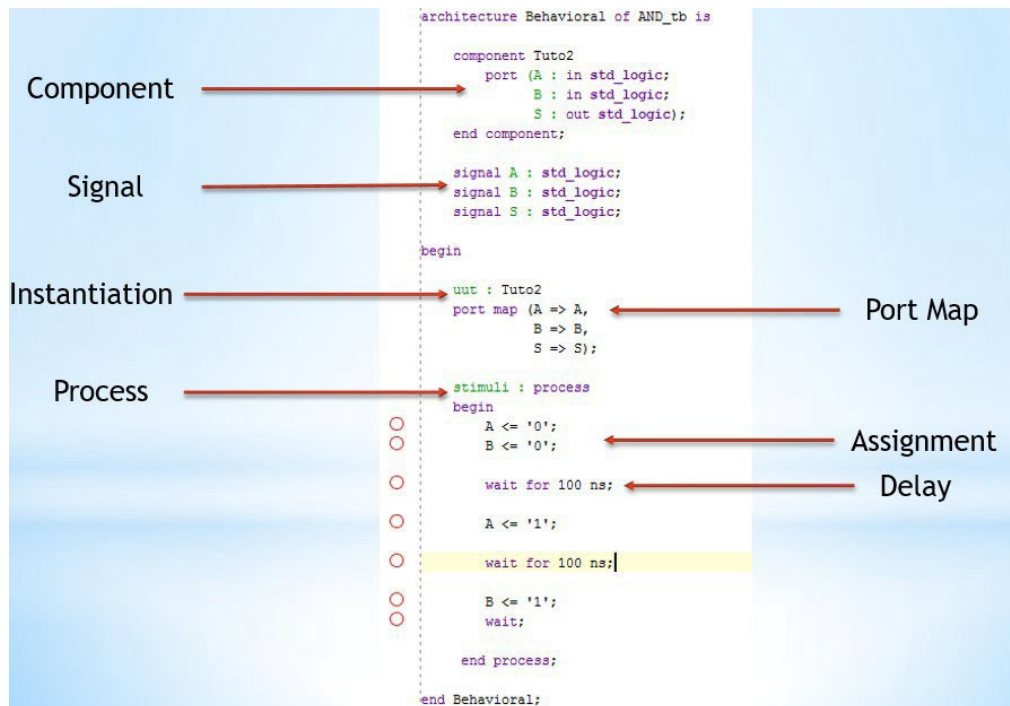
begin

    component instantiations

    process

end behavioral;
Example:

```



- Component declaration declares a virtual circuit template, which must be instantiated to take effect during the design.
- A port map is required for component instantiation. Port mapping is for connecting signals of the design in which the components are instantiated with the ports of the component itself.
- Multiple processes may exist in an architecture. All processes in a VHDL description file (.VHD) are executed in parallel – concurrent process blocks. Statements within a process are executed sequentially.