# EXPERIMENT: 1

**Objective:** To make hands-free with FPGA Board (Spartan 6) using Xilinx 14.7.
When the push button is pressed, led turns ON and otherwise it remains OFF.

**Software:** Xilinx ISE Design Suite14.7

**Target Hardware:** FPGA Board (Spartan 6)

## Procedure:

1. To start the **Xilinx ISE design Suite 14.7**, double-click on the **ISE Design Suite 14.7** icon on your desktop, or select Start => All Programs => Xilinx ISE Design Suite => Xilinx ISE Design Suite 14.7 => ISE =>Design Tools => Project Navigator.

2. The **ISE Design Suite 14.7** interface will be opened (Fig. 1.1). In **ISE Design Suite**, select **File => New Project.**

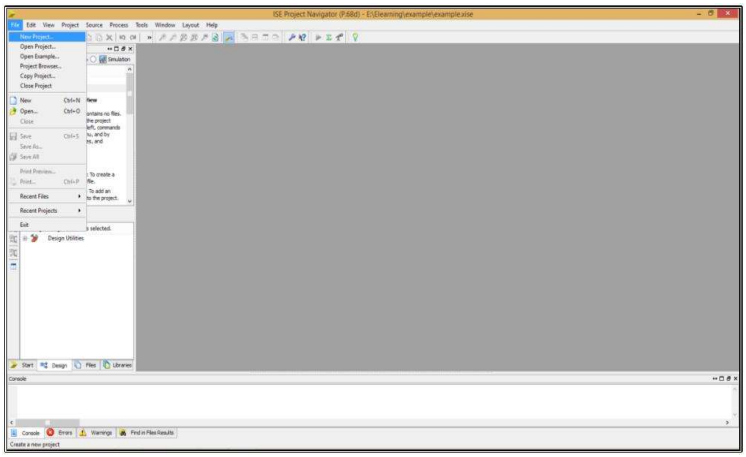

**Fig 1.1:** ISE Project navigator interface

3. The **New Project Wizard** window will be displayed. In the **Name** field type switch(i.e. name of the project) and in the **Location** field, choose the directory in which you have to store the project. For the current project it is /home/ise/switch(Fig. 1.2)

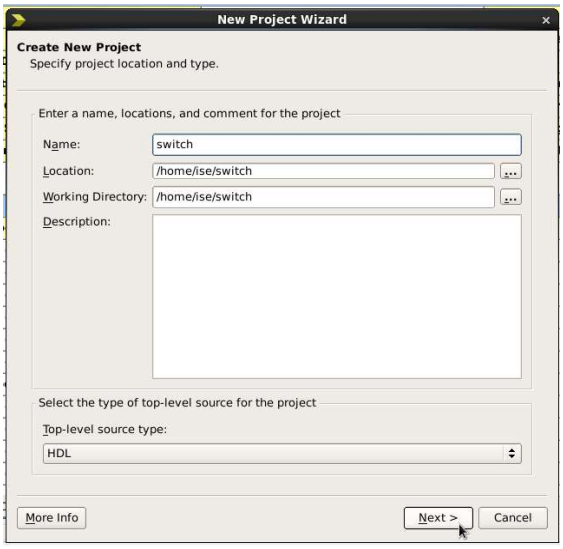


**Fig 1.2:** New Project Wizard window

Also, select HDL as the **Top-Level Source type.** You can also add project **description** if you need.

4. Click **Next. New Project Wizard—Project Settings** page will be display

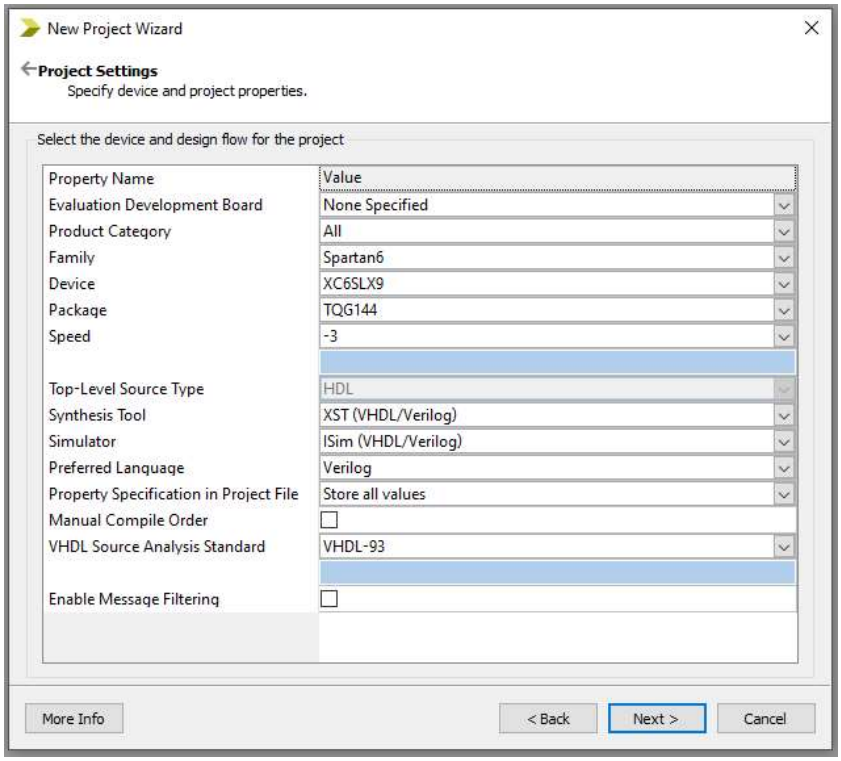


**Fig 1.3:** Project setting page

Select the following properties in the **Project Settings** to specify project and device properties:

- **Product Category:** All
- **Family:** Spartan6
- **Device:** XC6SLX9
- **Package:** TQG144
- **Speed:** -3
- **Synthesis Tool:** XST (VHDL/Verilog)
- **Simulator:** ISim (VHDL/Verilog)
- **Preferred Language:** VHDL or Verilog depending on the preference. For the current project choose **Verilog**. This will determine the default language for all the processes that generate HDL files.

Other properties can be left at their **default** values. Click **Next**.

5. The **New Project Wizard—Project Summary** page appears.

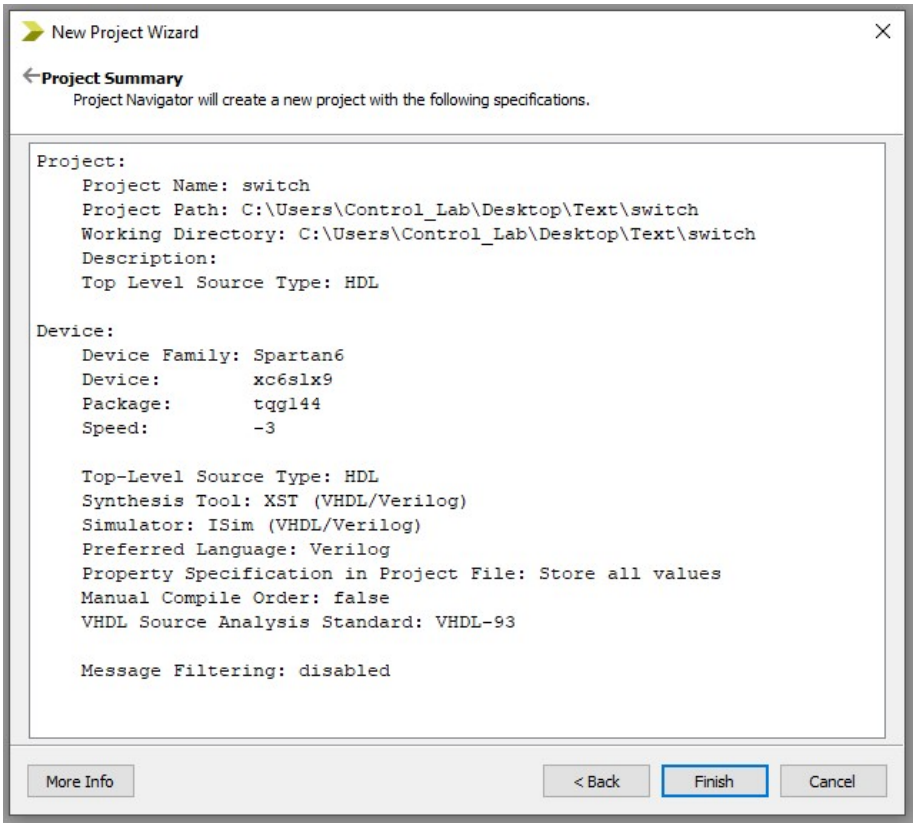


**Fig 1.4:** Project Summary page

Verify the **Project Summary** and click **Finish**.

6. The ISE Project Navigator interface will be as shown in Fig.1.5.

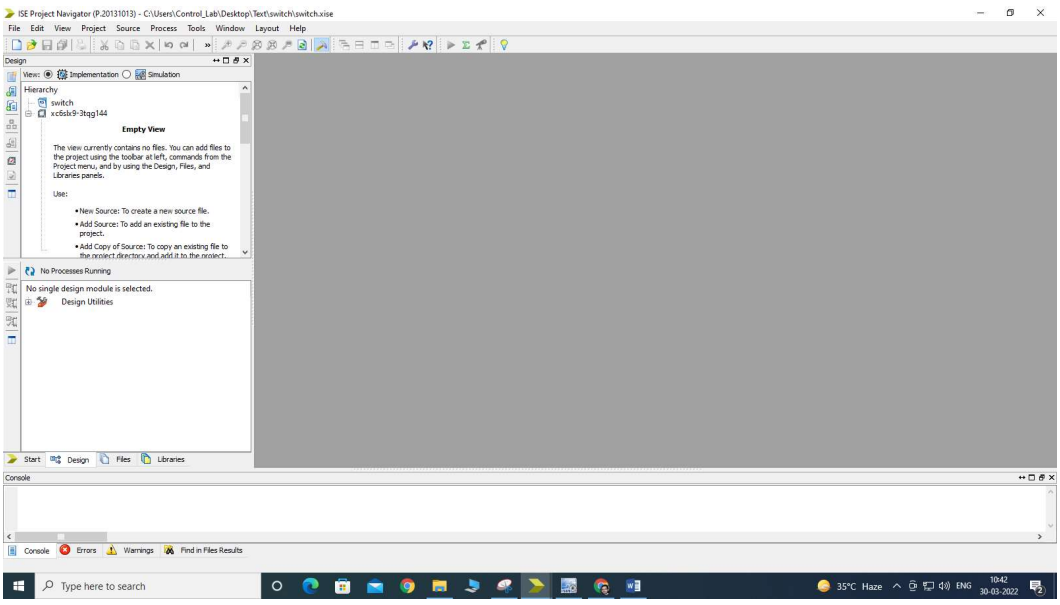

**Fig 1.5:** ISE Project Navigator

7. To create new source file, select **Project => New Source**. The **New Source Wizard** window will open in which you have to specify the type of source you want to create. Select **Verilog Module** as the **Source Type** and enter a name “switch” for the new source in the **File Name** field. Click **Next.**

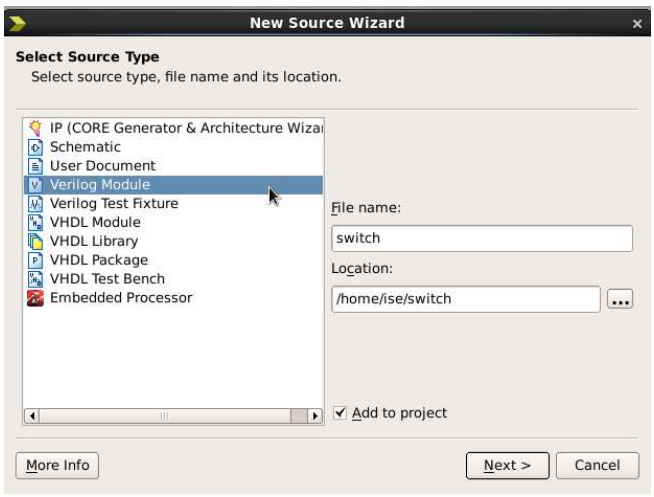


**Fig 1.6:** Select Source Type

8. In the **Define Module** page, enter the port information for the “switch” as follows (Fig 1.7):
   **a.** In the first two **Port Name** field, enter ‘y’ and ‘a’.
   **b.** Set the **Direction** field to output for ‘y’ and to input for ‘a’. Leave the **Bus** check boxes as it is.

Click **Next**.

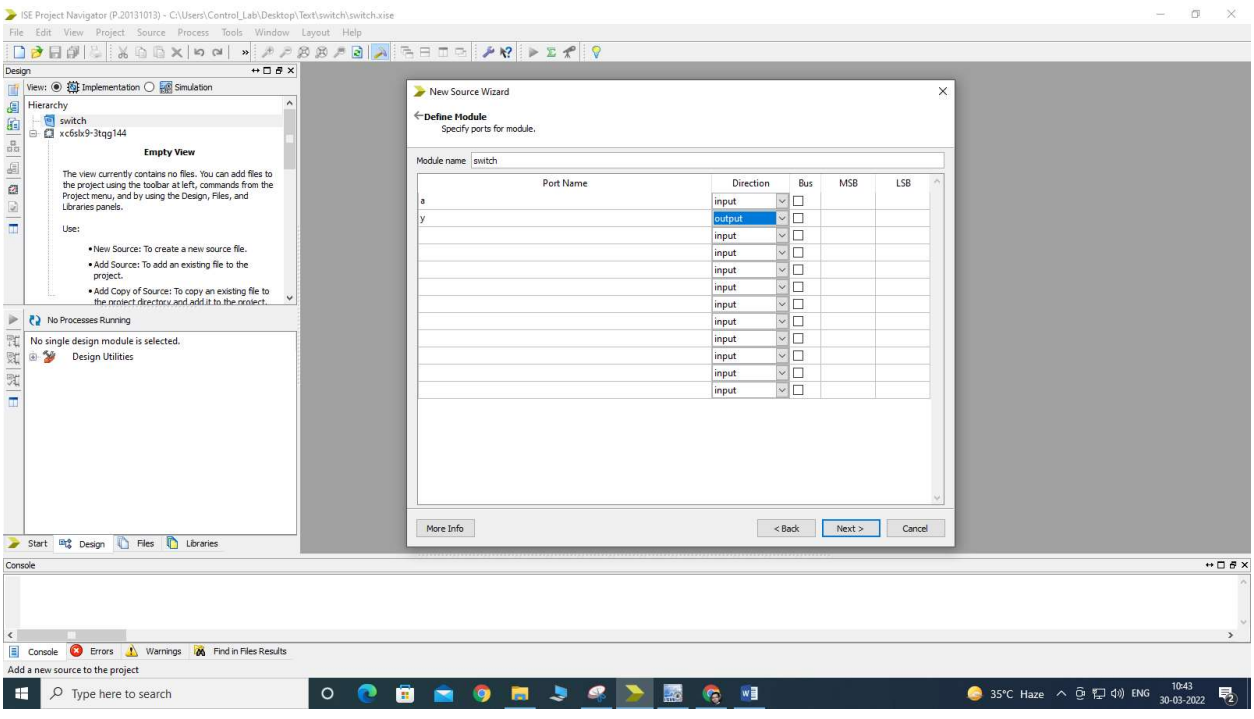


**Fig 1.7:** Define Module window

9. **Summary** window of the **source** file will be displayed (Fig 1.8). Check the source summary and click on Finish.

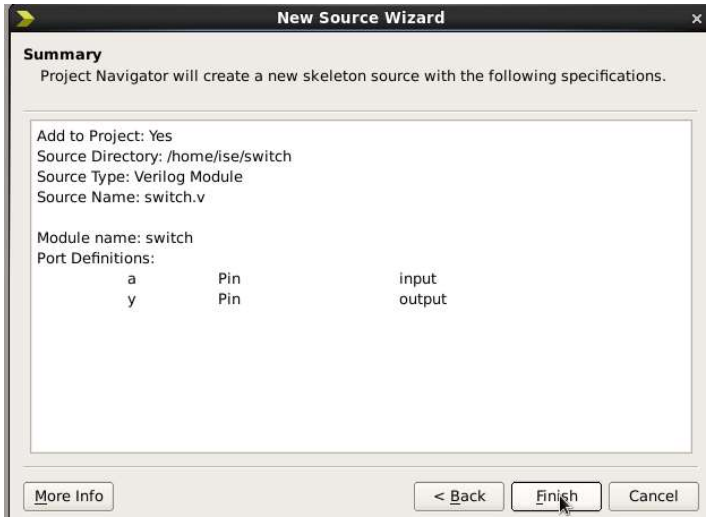


**Fig 1.8:** Summary Window

10. In the Xilinx ISE interface you can see that the new source file "switch.v" has been added to the project.

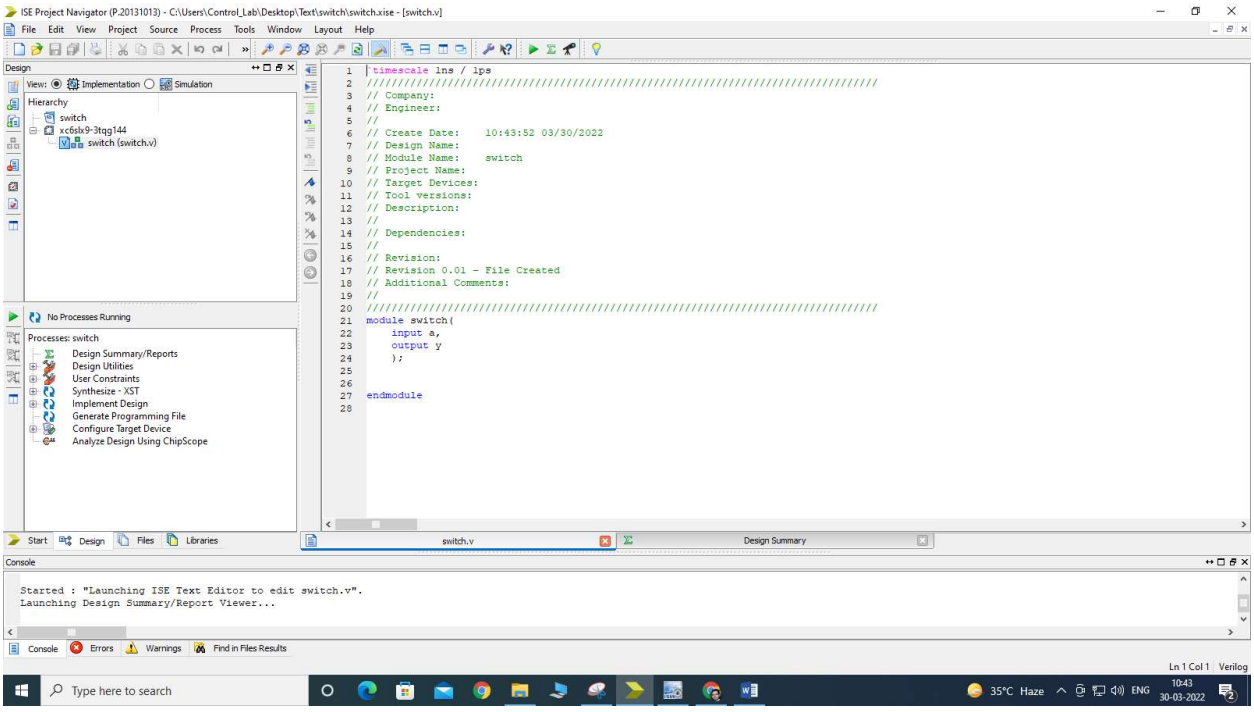


**Fig 1.9:** Switch Structure in the workspace

## Verilog code for the Switch

```
module switch(
   output y,
   input a,
   );
assign y=a;
endmodule
```

11. In the **ISE Text Editor**, a template of the Verilog source file, based on the information you provided while creating the new source, is already generated by Xilinx ISE Design Suite (Fig 1.9). Now you need to complete the source code for the switch. For the current experiment, data flow modeling style is used. You can also write the desired logic by using gate level or behavioral modeling style.

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    21:49:53 01/08/2019
7  // Design Name:
8  // Module Name:    switch
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module switch(
22     input a,
23     output y
24     );
25 assign y=a;
26
27 endmodule
28
```

**Fig1.10:** Complete Verilog code Structure

12. After entering the Verilog code , save the file. The source file containing the Verilog code displays in the Workspace, and the complete "switch" displays in the Source tab (Fig 1.11).

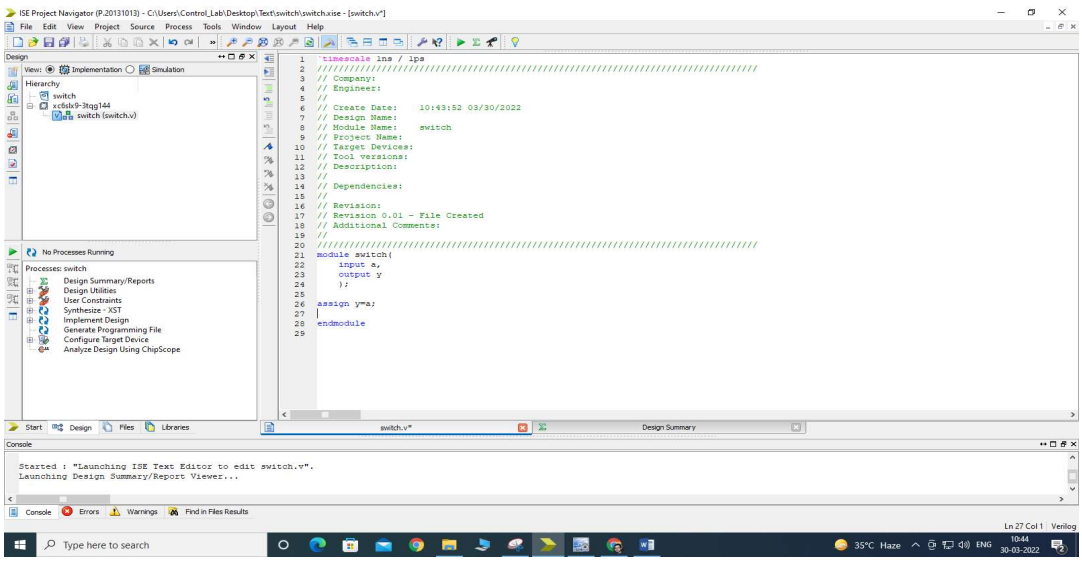


**Fig 1.11:** Source Tab with complete Verilog code

13. After completing the code next step is to check its syntax and functionality. For this, select **Simulation** Button in the **Hierarchy window** on the **left pane** as shown in the Fig 1.12.

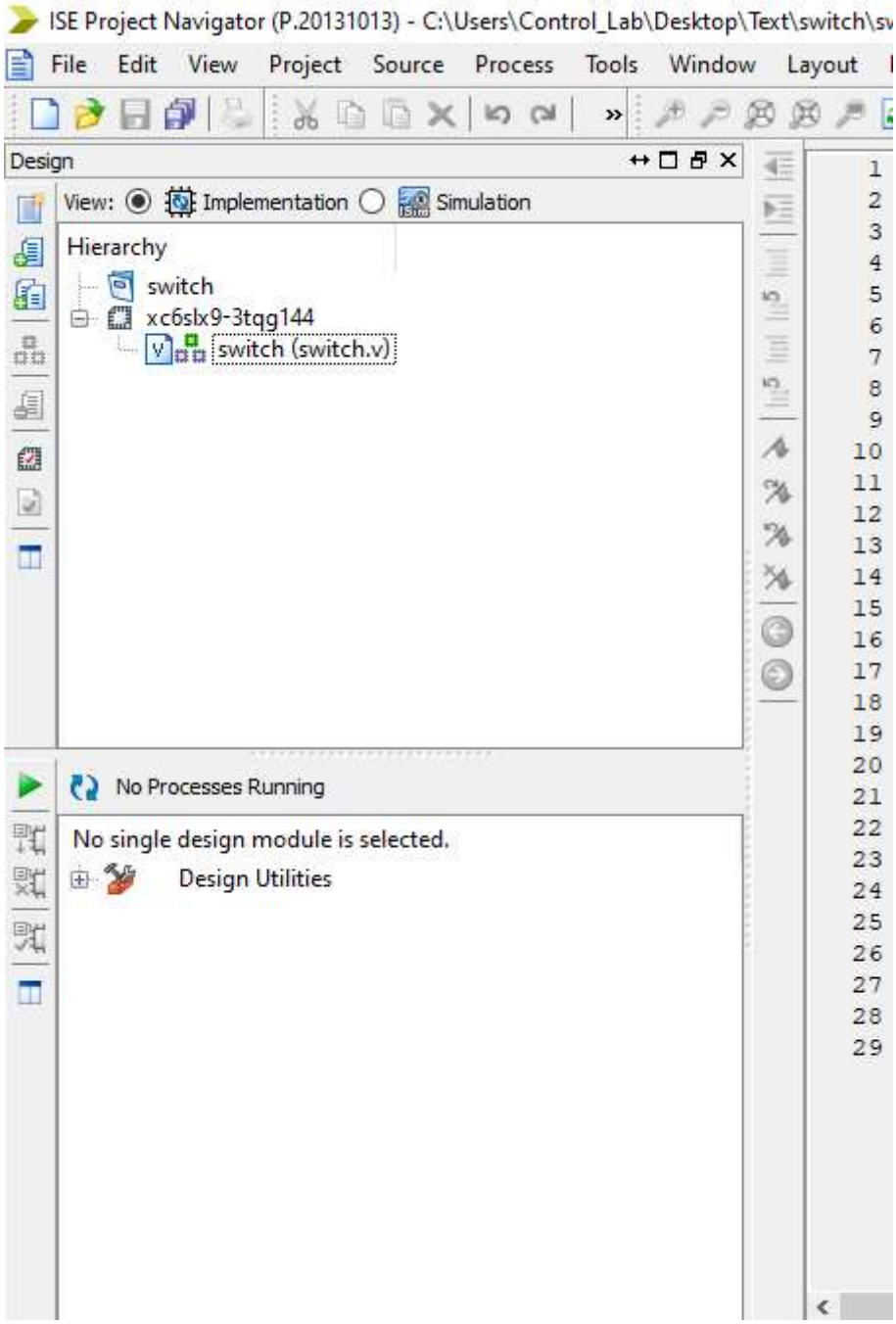


**Fig 1.12:** Hierarchy window

14. Now in **Simulation window** select the source file "switch.v" and click on **Behavioral Check Syntax** in the **ISim Simulator** in the **Processes window** to check for syntax errors. If syntax errors are present, then they will be displayed in the Error window at the bottom.
    If the design contains no errors, then a green check will appear near the Behavioral Check Syntax.

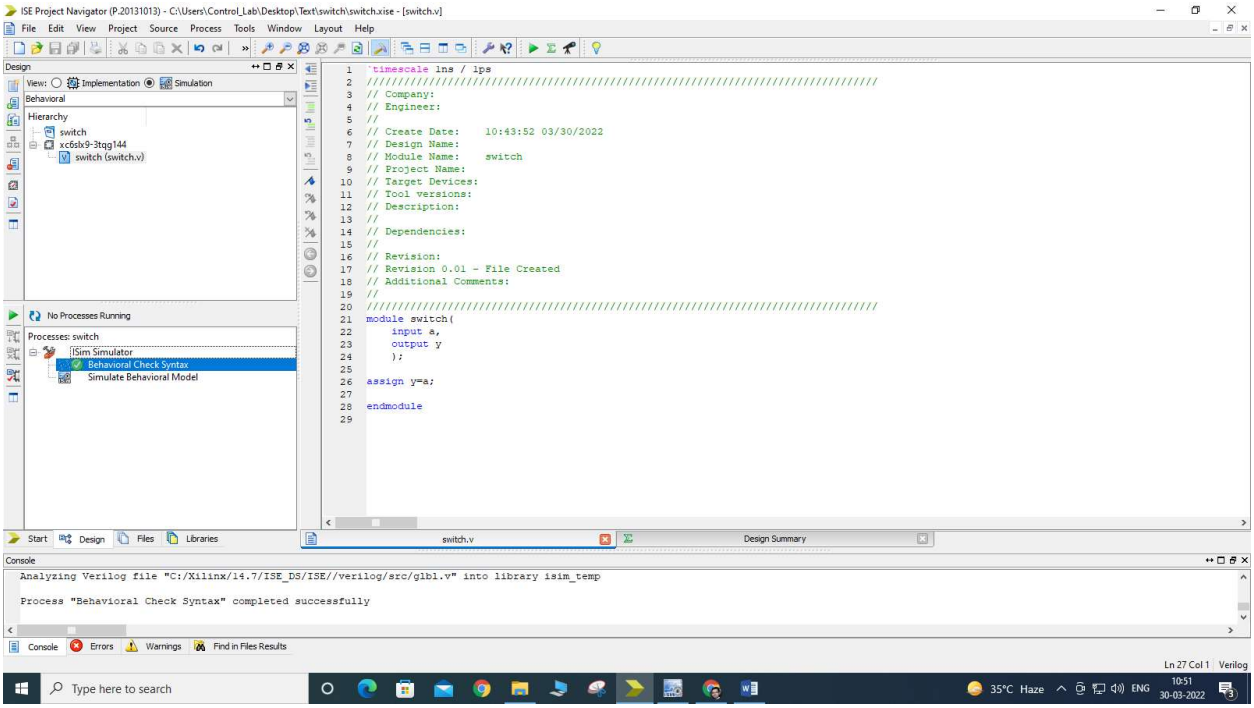


**Fig 1.13:** Simulation window

15. Now in order to check the **functionality** of the design, we have to apply test vectors and simulate the circuit. In order to apply test vectors, a **test bench** file is written. Through test bench, all the possible inputs are applied to the module and the output is verified. For "switch" the test bench can be created by one of the following methods. Either right click on the design (switch.v) and select **New Source** or, click on **New Source** from the **Project** menu. A **New Source Wizard** window will be opened. In the New Source Wizard window select **Verilog Test Fixture** as shown in the Fig 1.14.

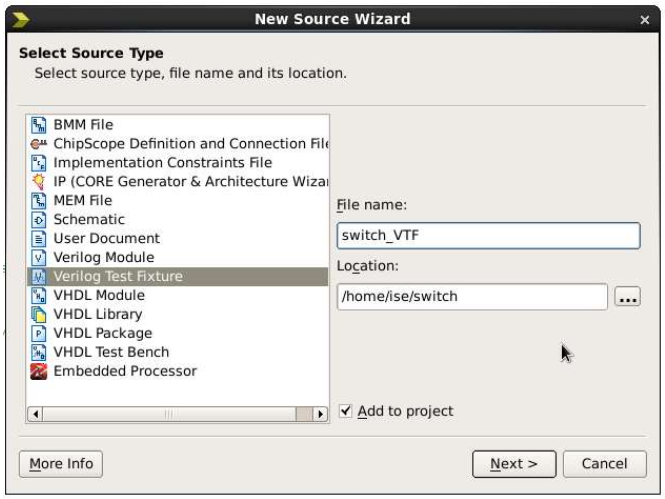


**Fig 1.14:** Verilog Test bench

16. Write the test bench name in the File name field. Here the test bench name is "switch_VTF". Click **Next**.

17. In the **Associate Source** window, select the source file with which you want to associate the test bench (Fig 1.15). Click **Next.**


**Fig 1.15:** Associate Source

18. Now in the Summary window verify the test bench summary and click Finish (Fig 1.16).

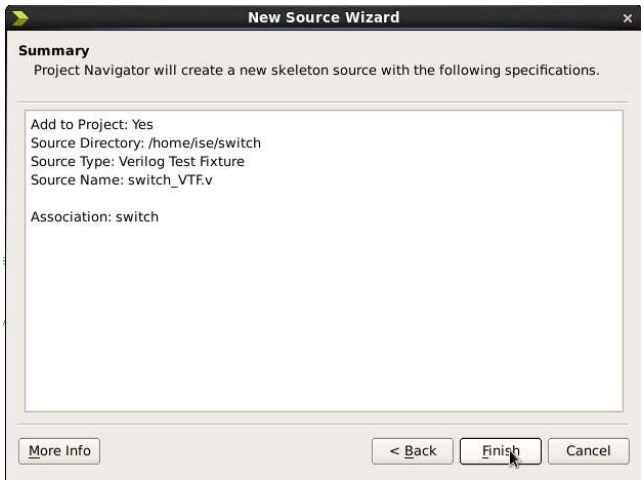


**Fig 1.16:** Summary window

19. The ISE project navigator will generate template for the test bench as shown in the Fig 1.17.

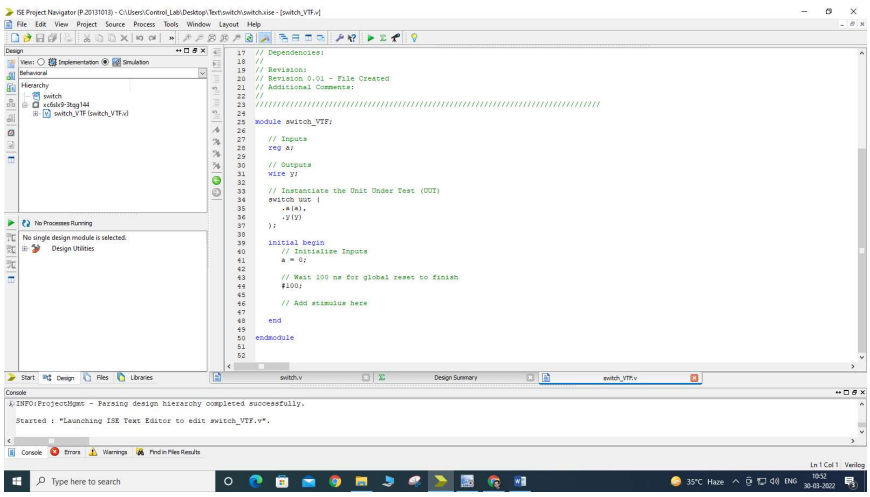

**Fig 1.17:** Test bench template

The ISE design tool detects the inputs and outputs in the source file and creates a template instantiating the original source module and provides initial values.

20. Give different input values for ‘a’ after 100 units of time delay. After writing the input combinations for ‘a’ save the code (Fig 1.18).

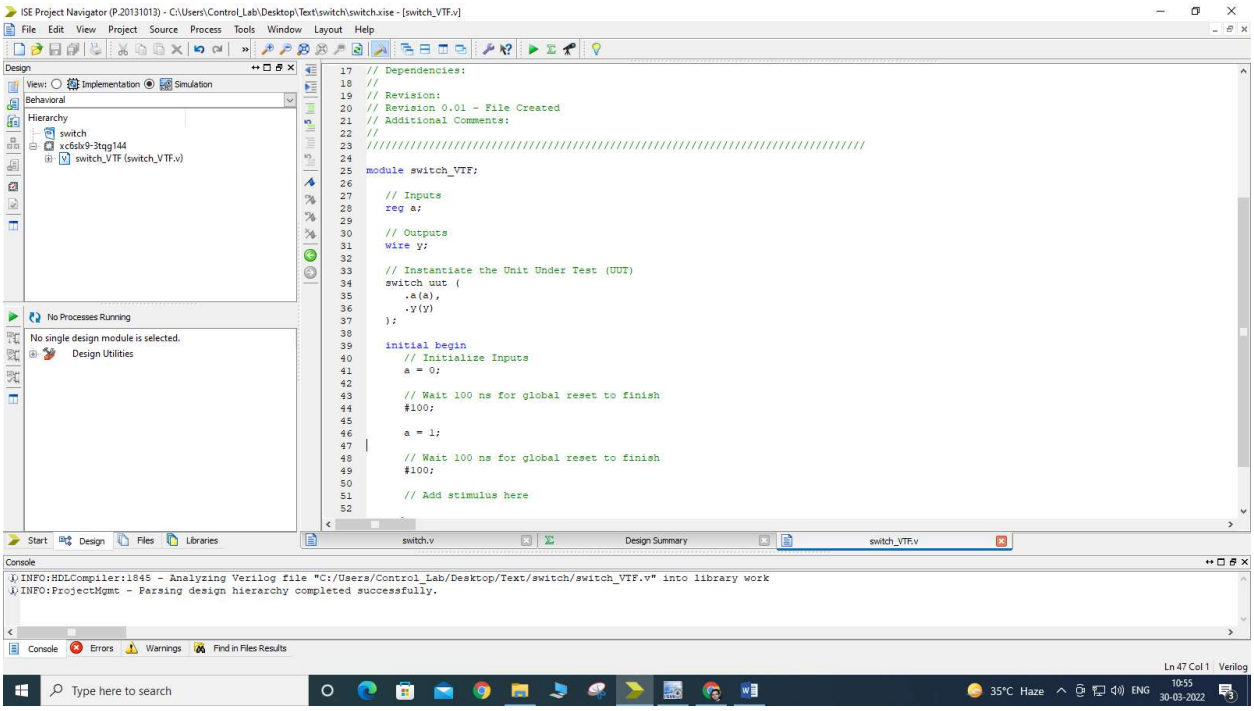


**Fig 1.18:** Complete Test bench

21. After editing the test bench, click **Behavioral Check Syntax** in the **Processes window** to check for syntax errors in the test bench. If there is no error double click **Simulate Behavioral Model** (Fig 1.19)

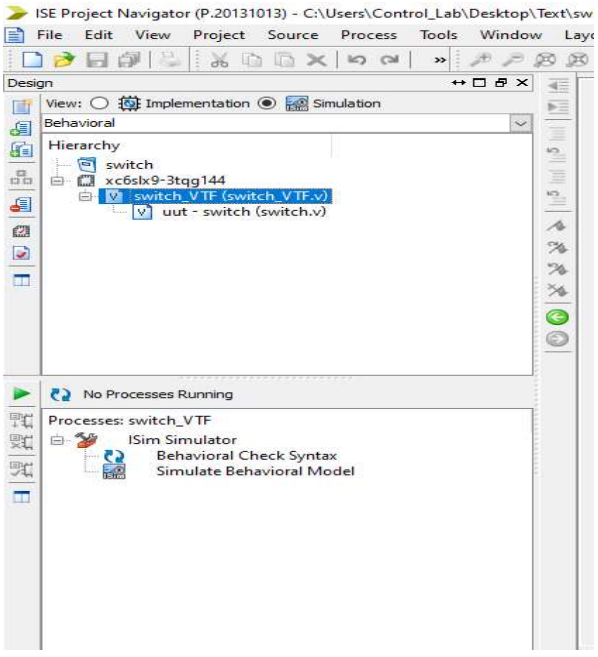

**Fig 1.19:** Process Window

22. A new window showing the waveforms for the inputs and outputs of the design will open. The simulated waveform for the “switch” is shown in the Fig 1.20. The simulation is successful here as the output is verified for switch.

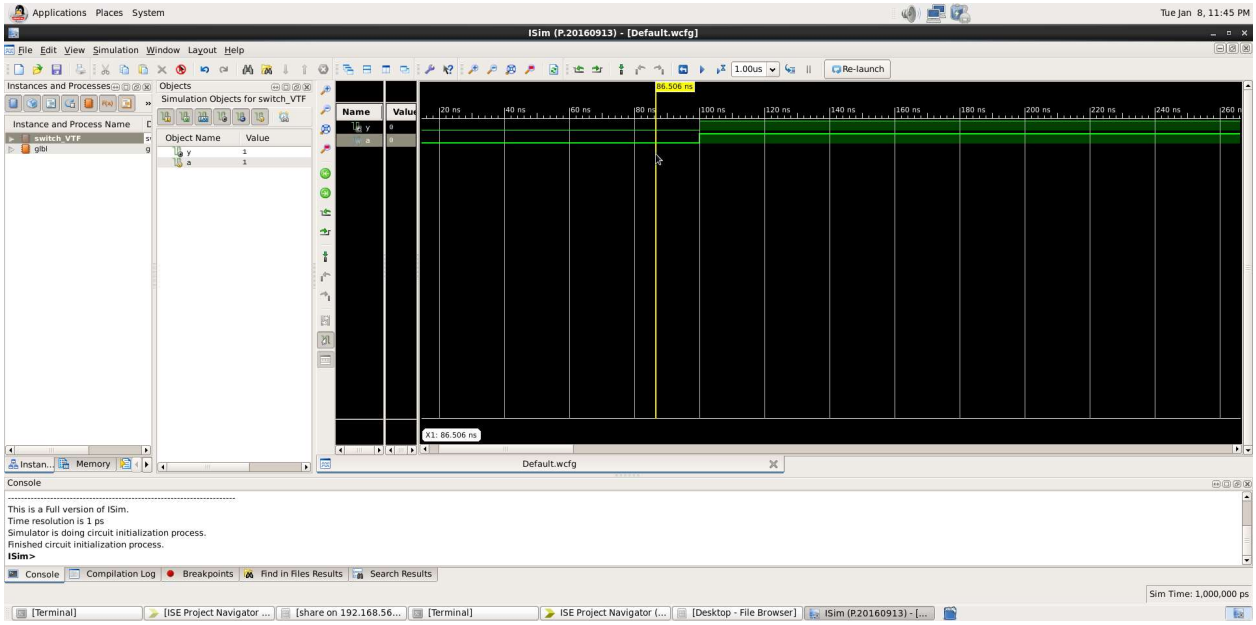


**Fig 1.20:** Simulation Result

23. Next we have to select **Implementation** button in the **Hierarchy window** on the left pan (Fig 1.21) for the hardware implementation of the design.


**Fig 1.21:** Hierarchy Window

24. Now we have to synthesize the design. The **synthesis tool** takes the HDL code and generates a netlist for the Xilinx implementation tools. The synthesis tool performs the following steps to create the netlist:

- **Check Syntax:** Checks the syntax of the source code.
- **Compile:** Translates and optimizes the HDL code into a set of components that the synthesis tool can recognize.
- **Map:** Translates the components from the compile stage into the target technology’s primitive components.

Right click on the **Synthesize-XST** and then select **Run**. If the design is successfully synthesized then green check will appear near the Synthesize-XST as shown in Fig 1.23 and ”**Process "Synthesize - XST" completed successfully**" message is displayed in the console window.

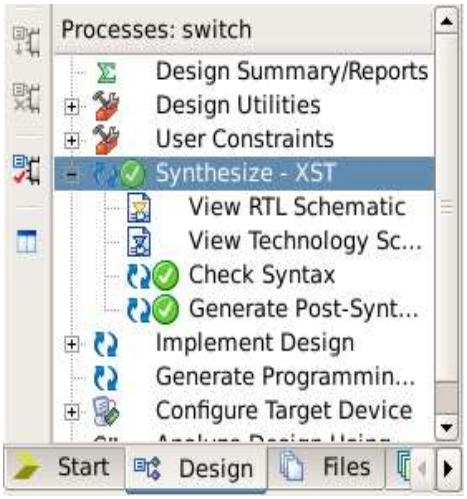


**Fig 1.23:** Synthesis successful

25. Synthesize tool can generate a schematic representation of the HDL code that you have entered. A **schematic view** of the code gives the graphical connection between the various components. Following are the two forms of schematic representation:

    • **RTL View:** Pre-optimization of the HDL code.
    • **Technology View:** Post-synthesis view of the HDL design mapped to the target technology.

    In order to view a schematic representation of the HDL code, in the **Processes** pane expand **Synthesize** (click the **+** sign), and double-click **View RTL Schematic** or **View Technology Schematic**. **"Select RTL/Tech Viewer Startup Mode"** window pops-up. It provides two options for the startup mode, select the second option: **"Start with a schematic of the top-level block".** Click **Ok**.

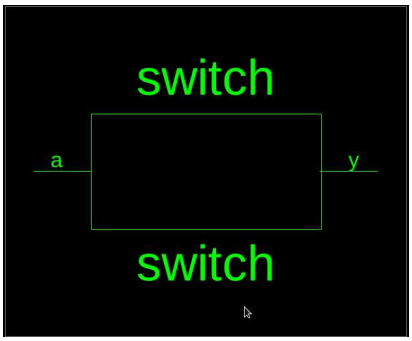


**Fig 1.23:** RTL Schematic


**Fig 1.24:** Internal View of RTL schematic

Fig 1.23 represents the **RTL schematic view** of "switch.v" and to see a detailed view of the RTL schematic double click on the Fig 1.23 you will get the internal gate level diagram as shown in the Fig 1.24. Similarly, you can get the view for the Technology Schematic.

26. Next step is the design implementation. But before design implementation we need to assign the pin usage for the chip. For this we have to write the **User Constraints File (UCF)**.

    The user constraints file (or implementation constraints file) is used to specify the actual pins on the FPGA to which the ports in the top most entity are mapped. In practice, FPGAs in all applications would have been mounted on a board with peripheral devices and thus specifying pins as either inputs or outputs based on peripheral connections .Refer to Appendix B for complete UCF of the FPGA board.

    For UCF File either right click on the design (switch.v) from Hierarchy section and select **New Source** or click **New Source** from the **Project menu**. From the New Source Wizard window choose **Implementation Constraints File.**

    27. In **Implementation Constraints File** enter the name of the UCF to be generated (Fig 1.25).

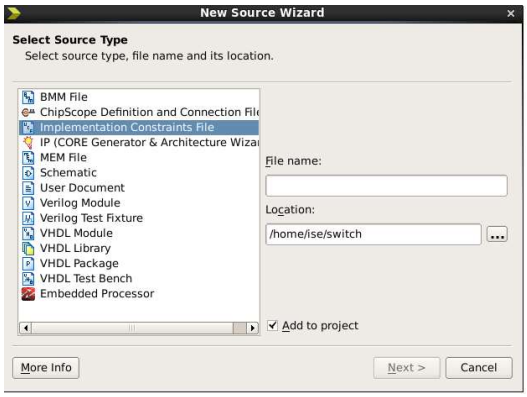


**Fig 1.25:** Implementation constraints file

Make sure that **Add to Project** option is selected to add the constraints file to the project folder. Click **Next**.

28. **Summary Window** will be displayed (Fig 1.26).Select **Finish**.

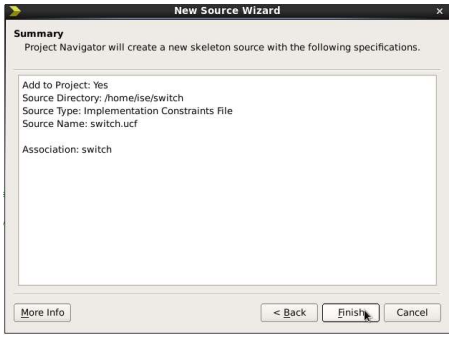


**Fig 1.26:** Summary Window

29. A blank window will appear in a new tab having the design name with .ucf extension. Here the tab “switch.ucf” is generated as shown in the Fig 1.27.

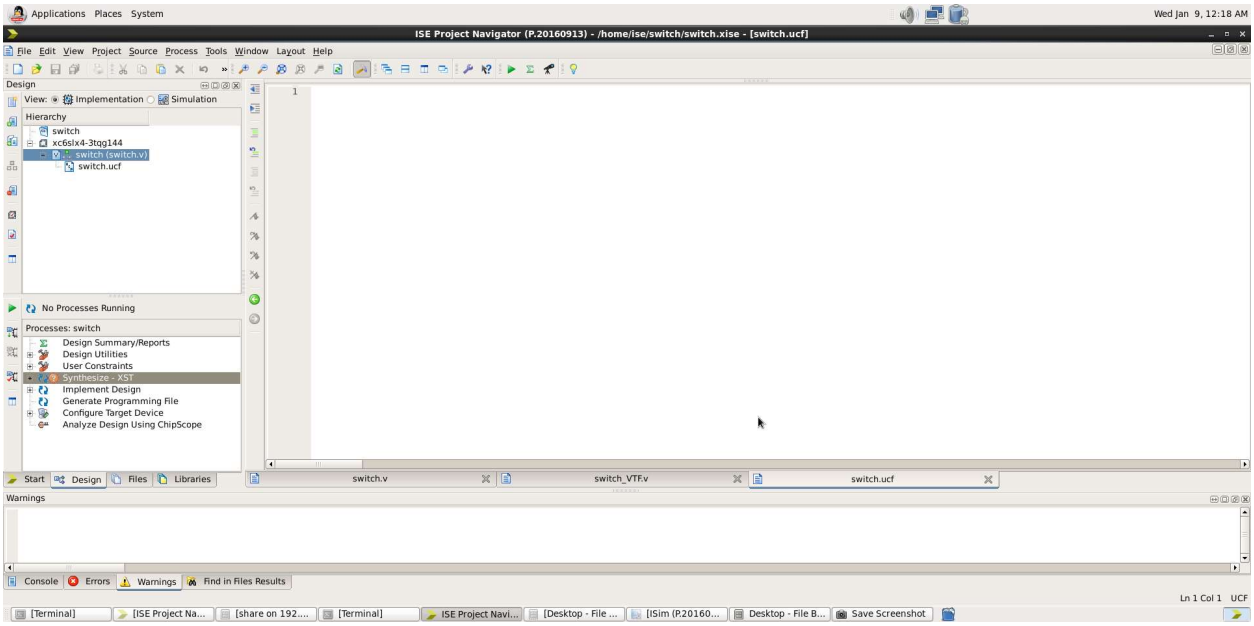


**Fig 1.27:** Blank UCF

30. In the **UCF** , enter the constraints for each input and output.
The syntax to add a constraint manually is :
**NET** "input/output/clock" **LOC** = pin number**;**

The UCF written for the “switch” module is as follows:

**NET "a" LOC = p21;**
**NET "y" LOC = p32;**

After adding the constraints save the UCF.

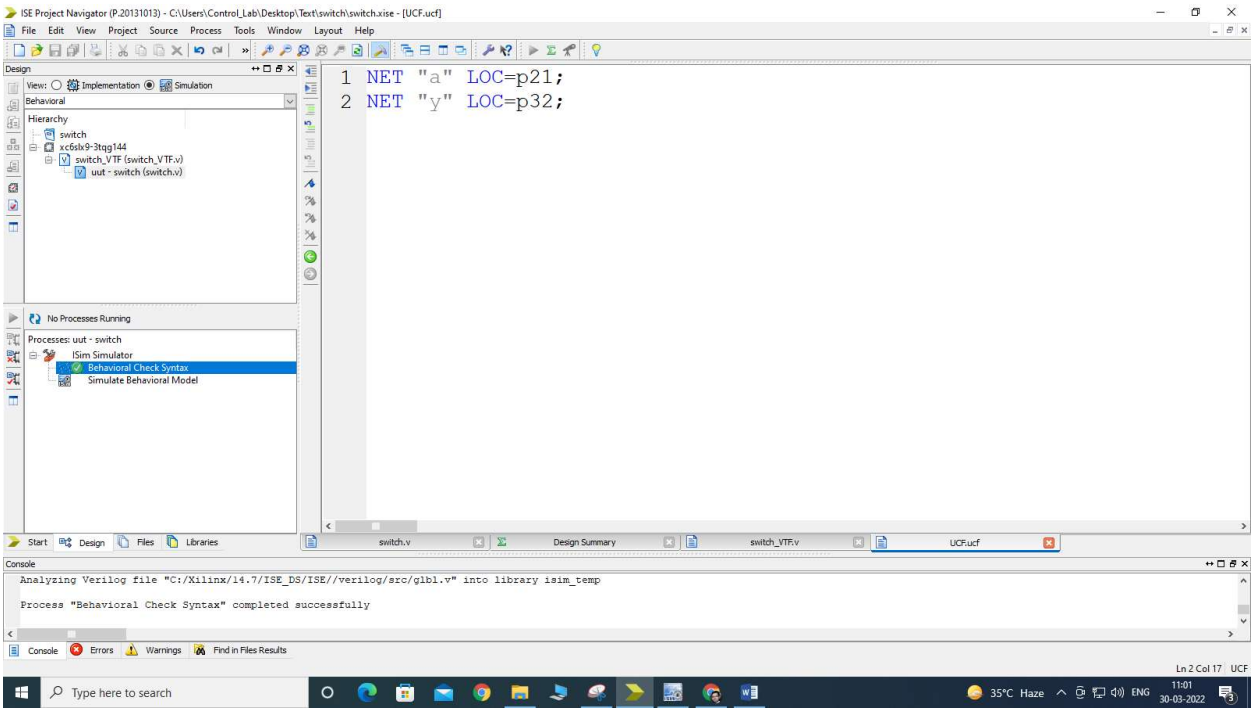


**Fig 1.28:** UCF for switch

31. After writing the UCF, we will proceed with the **Implement Design** step. Design implementation is the process of translating, mapping, placing, routing, and generating a bit file for the design. For this right click on **Implement Design** and select **Run**. If the design is implemented successfully then green check will appear near the Translate, Map and Place & Route process as shown in Fig1.29

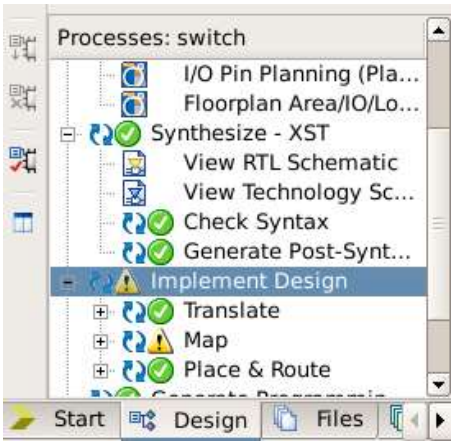


**Fig 1.29:** Implementation successful

32. Now right click on the **Generate Programming file** and select **Run** . This will produce the bit stream file. The bit stream file is the configuration file to be downloaded to the chip.If the Generate Programming File process is successfully completed the "**Process Generate Programming File completed successfully**" message is displayed in the console window.

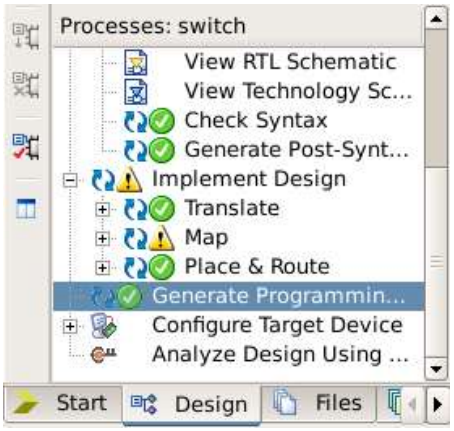


**Fig 1.30:** Generate Programming File

33. The **Design Summary** tab lists the summary of the design, including any errors or warnings. This can be accessed from **Project** -> **Design Summary/Reports** in the dropdown menu.

34. Now for hardware implementation or programming the FPGA board, connect the board to the computer through the USB cable provided and switch on the power supply of the board.

35. Double click on the **Configure target device**, a popup windows will open and then click on **OK** (Fig 1.31).

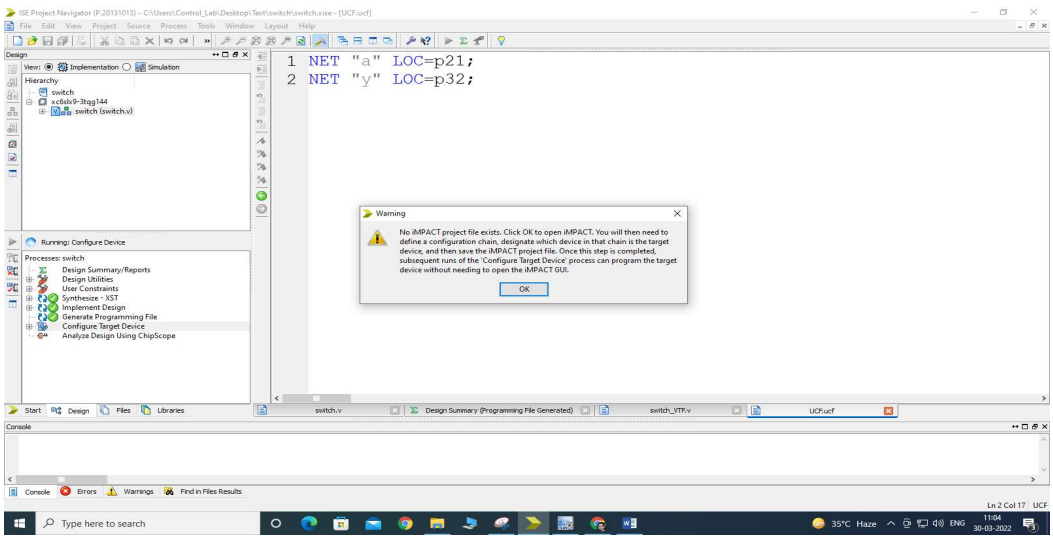


Fig 1.31

36. A iSE iMPACT window will open, double click on boundary scan (Fig 1.32).

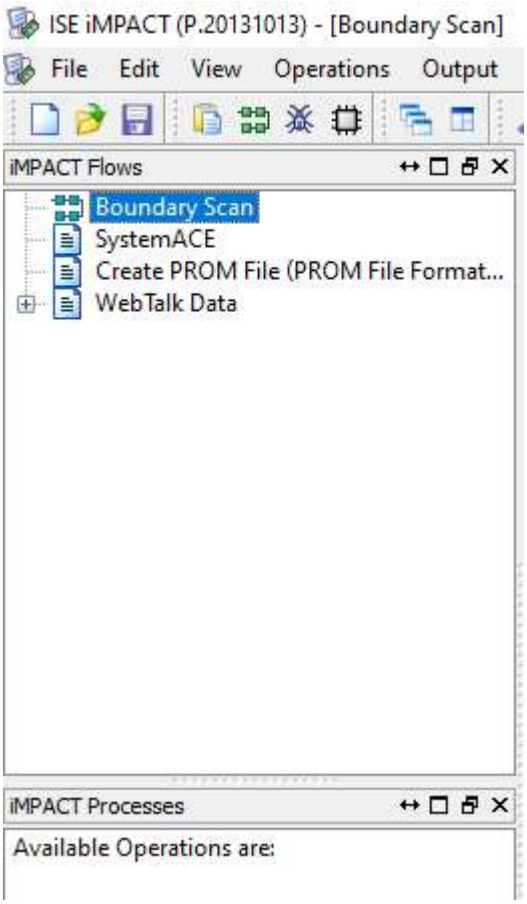


Fig 1.32

37. Right click to add device or initialize JTAG chain.(Fig 1.33)

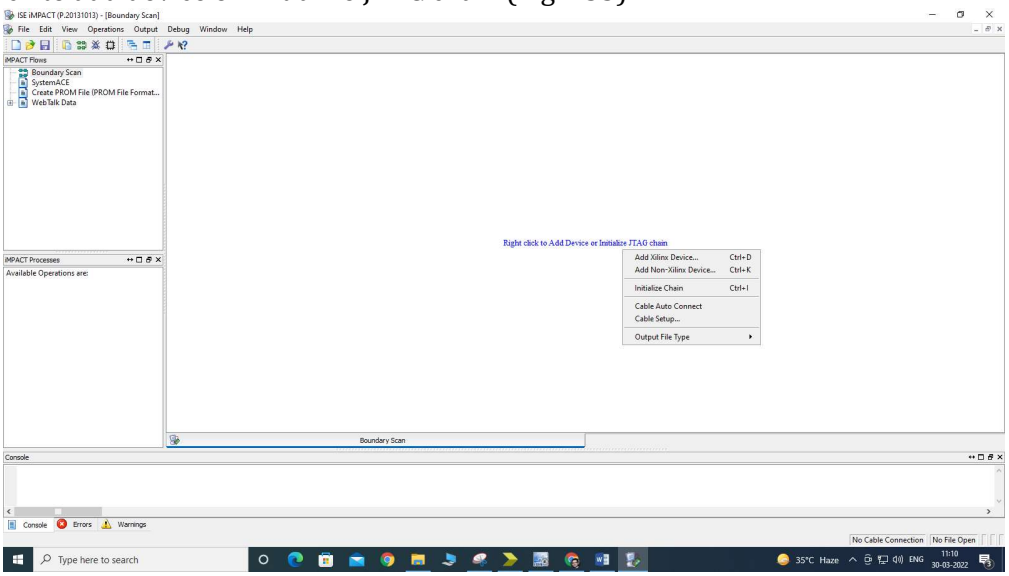


Fig 1.33

38. Click on add Xilinx device then go to file location **select switch.bit** file and click open.(Fig 1.34)

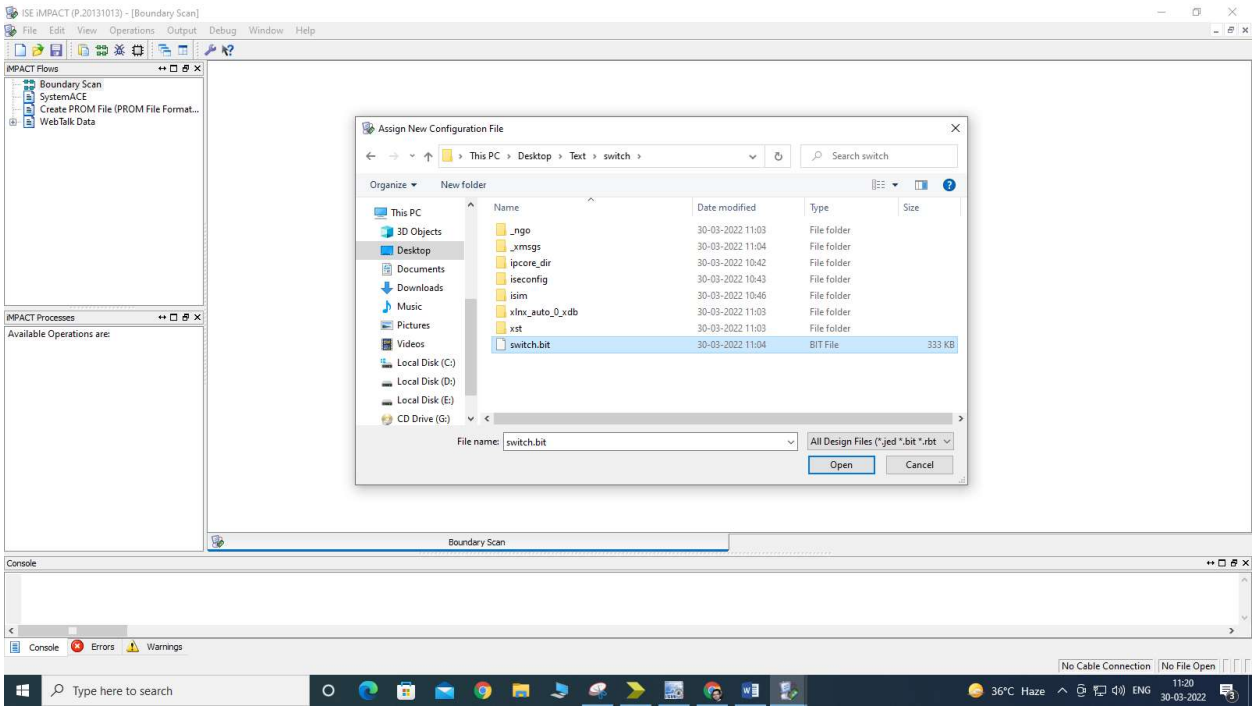


Fig 1.34

39. Click on Xilinx IC symbol (Fig1.35)

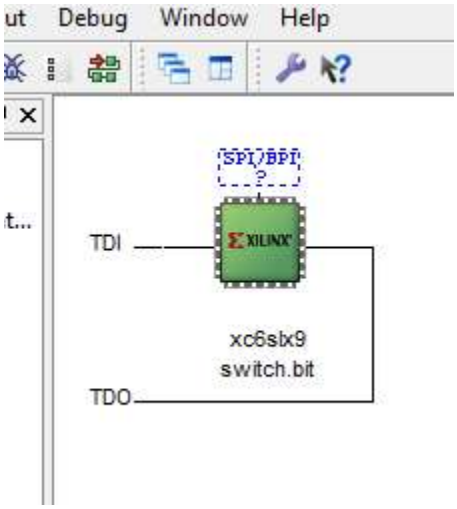


Fig 1.35

40. Double click on program then a popup window (device programming property window) will open. (Fig 1.36)

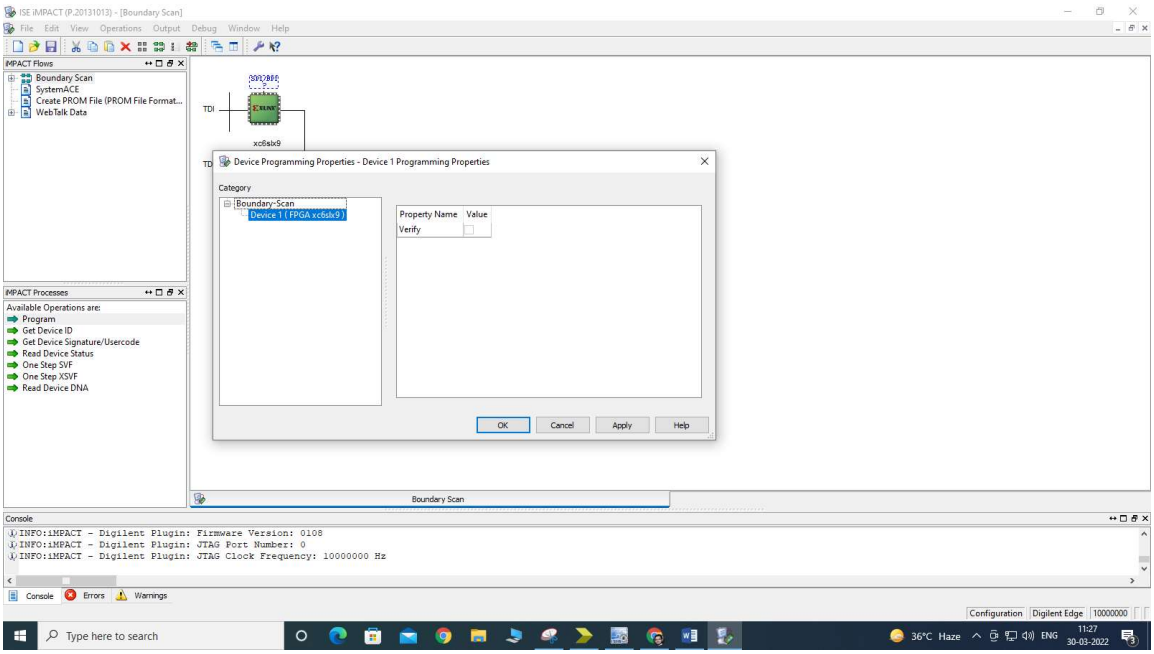


Fig 1.36

41. Click ok on Device Programming Property window (Fig 1.36)

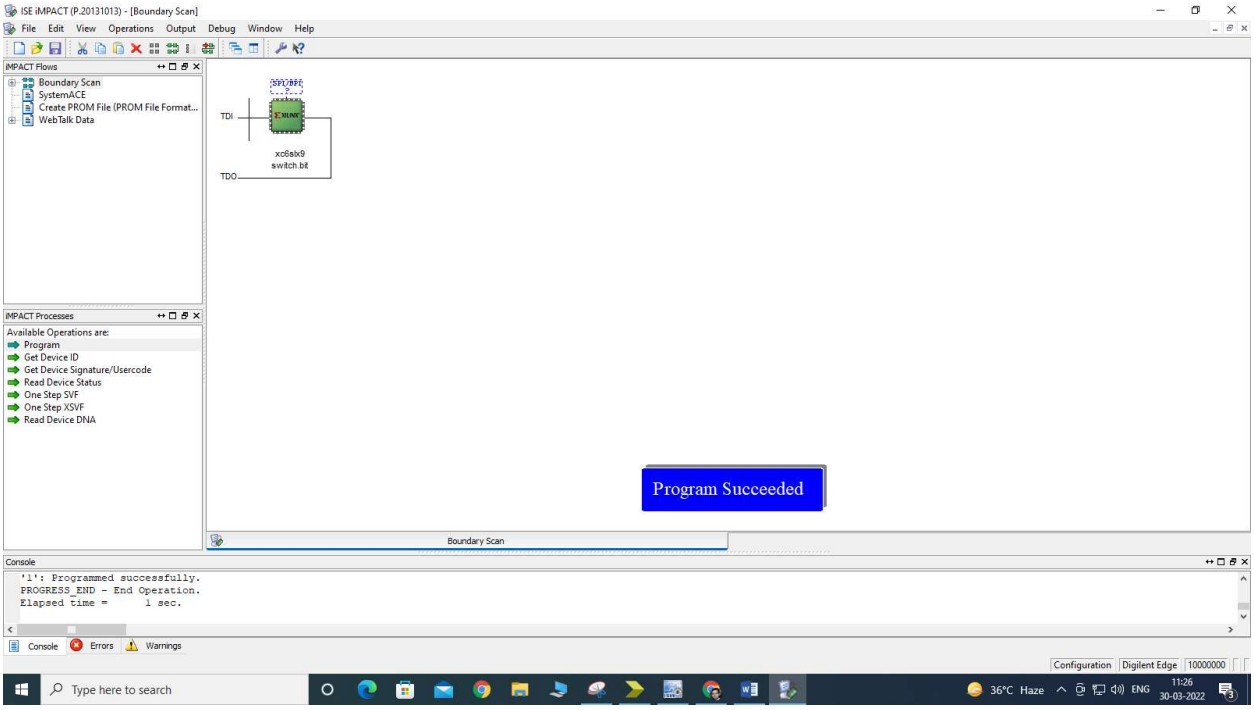


Fig 1.37

39. When the FPGA is programmed successfully then a message "**Program Succeeded**" will be displayed in the **Status** console at the bottom of the window (Fig 1.37). Also a red LED on the board will glow to indicate that the FPGA has been programmed successfully. (Fig 1.38)

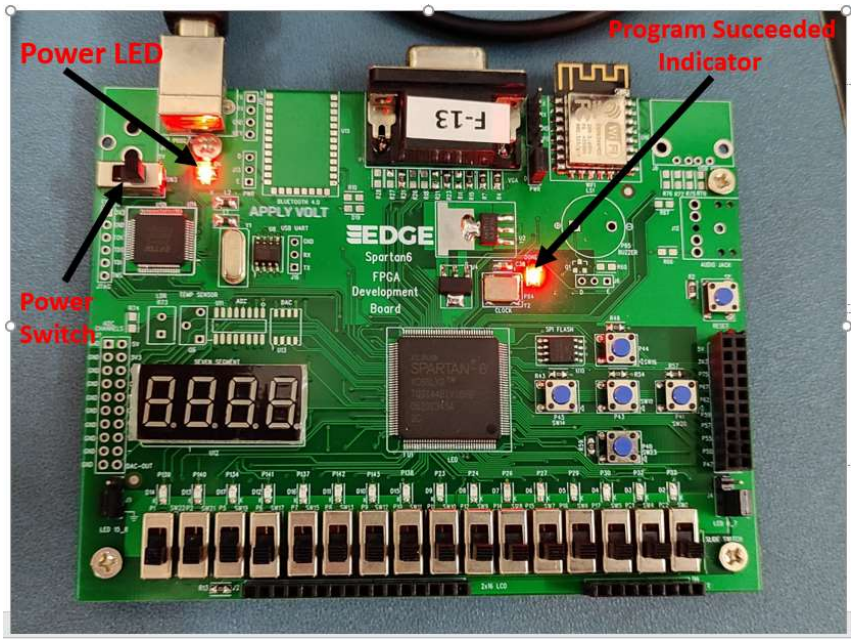


Fig1.38

40. With the board having been programmed you can now verify your design. Remember, from the User/Implementation constraints file, that the one switch is used as input (SW1 for "a") and a LED (led1) is used for "y".

**RESULT:**

1)When the switch is not pressed, LED remains OFF (Fig 1.39)

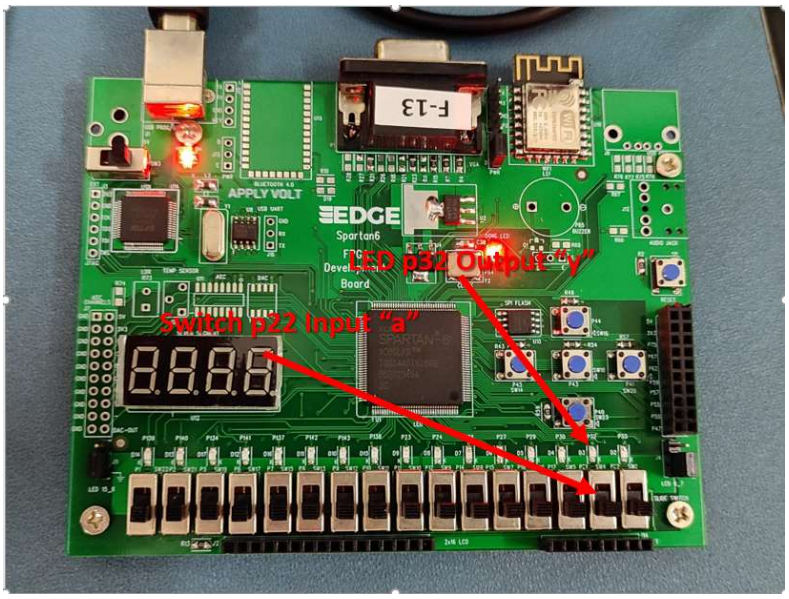


Fig 1.39

1)When the switch is pressed, LED turns ON (Fig 1.40)

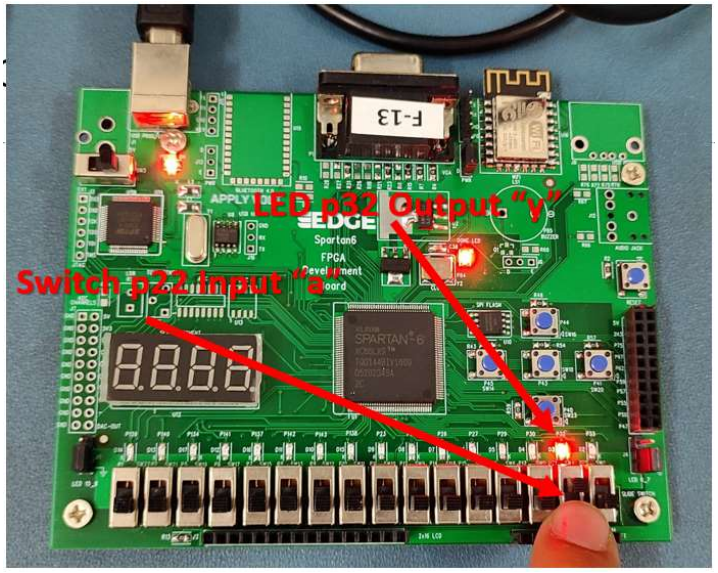


Fig 1.40

# Constraint File of FPGA Board

```
NET "*" IOSTANDARD=LVCMOS33;  CONFIG VCCAUX = "3.3" ;

#50 MHz oscillator
NET "CLK50M"       LOC = "p84";
NET "CLK50M"       TNM_NET = "osc";
TIMESPEC "TS_osc" = PERIOD "osc" 20.000ns;

#Slide Switches
NET "sw<0>" LOC = p22; #LSB
NET "sw<1>" LOC = p21;
NET "sw<2>" LOC = p17;
NET "sw<3>" LOC = p16;
NET "sw<4>" LOC = p15;
NET "sw<5>" LOC = p14;
NET "sw<6>" LOC = p12;
NET "sw<7>" LOC = p11;
NET "sw<8>" LOC = p10;
NET "sw<9>" LOC = p9;
NET "sw<10>" LOC = p8;
NET "sw<11>" LOC = p7;
NET "sw<12>" LOC = p6;
NET "sw<13>" LOC = p5;
NET "sw<14>" LOC = p2;
NET "sw<15>" LOC = p1; #MSB

#Push Buttons
NET "PB[4]" LOC = P45 | PULLDOWN;
NET "PB[3]" LOC = P44 | PULLDOWN;
NET "PB[2]" LOC = P43 | PULLDOWN;
NET "PB[1]" LOC = P41 | PULLDOWN;
NET "PB[0]" LOC = P40 | PULLDOWN;

#LEDs
NET "led<0>" LOC = p33; #LSB
NET "led<1>" LOC = p32;
NET "led<2>" LOC = p30;
NET "led<3>" LOC = p29;
NET "led<4>" LOC = p27;
NET "led<5>" LOC = p26;
NET "led<6>" LOC = p24;
NET "led<7>" LOC = p23;
NET "led<8>" LOC = p138;
NET "led<9>" LOC = p143;
NET "led<10>" LOC = p142;
NET "led<11>" LOC = p137;
NET "led<12>" LOC = p141;
NET "led<13>" LOC = p134;
NET "led<14>" LOC = p140;
NET "led<15>" LOC = p139; #MSB

#Seven Segment Display sharing I/O with LED
NET "digit[0]" LOC = P127;
```

```
NET "digit[1]" LOC = P131;
NET "digit[2]" LOC = P132;
NET "digit[3]" LOC = P133;
NET "Seven_Segment[0]" LOC = P134; #DP
NET "Seven_Segment[1]" LOC = P137; #G
NET "Seven_Segment[2]" LOC = P138; #F
NET "Seven_Segment[3]" LOC = P139; #E
NET "Seven_Segment[4]" LOC = P140; #D
NET "Seven_Segment[5]" LOC = P141; #C
NET "Seven_Segment[6]" LOC = P142; #B
NET "Seven_Segment[7]" LOC = P143; #A

#flash chip
NET FLASH_CS       LOC = "p38" ;
NET FLASH_SCK      LOC = "p70" ;
NET FLASH_MOSI     LOC = "p64" ;
NET FLASH_MISO     LOC = "p35" ;

#serial interface
NET "TXD"        LOC = "P88"; # transmit data
NET "RXD"   LOC = "P87"; # receive data

#wifi interface
NET "WIFI_TXD"          LOC = "P117"; # transmit data (output to FTDI chip)
NET "WIFI_RXD"  LOC = "P116"; # receive data (input from FTDI chip)

# VGA port
NET "vga_R<0>"  LOC = "p95";
NET "vga_G<0>"          LOC = "p100";
NET "vga_B<0>"  LOC = "p105";
NET "vga_HSync" LOC = "p111";
NET "vga_Vsync" LOC = "p112";

# 2X16 LCD
NET "lcd_data(0)"  LOC = "p23"  ;
NET "lcd_data(1)"  LOC = "p24"  ;
NET "lcd_data(2)"  LOC = "p26"  ;
NET "lcd_data(3)"  LOC = "p27"  ;
NET "lcd_data(4)"  LOC = "p29"  ;
NET "lcd_data(5)"  LOC = "p30"  ;
NET "lcd_data(6)"  LOC = "p32"  ;
NET "lcd_data(7)"  LOC = "p33"  ;
NET "lcd_e"   LOC = "p35"  ;
NET "lcd_rs"  LOC = "p34"  ;

# general-purpose I/O port (J5) / CMOS Camera Interface
NET "GPIO[0]"      LOC = "p75";
NET "GPIO[1]"      LOC = "p74";
NET "GPIO[2]"      LOC = "p67";
NET "GPIO[3]"      LOC = "p66";
NET "GPIO[4]"      LOC = "p62";
NET "GPIO[5]"      LOC = "p61";
NET "GPIO[6]"      LOC = "p59";
NET "GPIO[7]"      LOC = "p58";
NET "GPIO[8]"      LOC = "p57";
```

```
NET "GPIO[9]"      LOC = "p56";
NET "GPIO[10]"     LOC = "p55";
NET "GPIO[11]"     LOC = "p51";
NET "GPIO[12]"     LOC = "p50";
NET "GPIO[13]"     LOC = "p48";
NET "GPIO[14]"     LOC = "p47";
NET "GPIO[15]"     LOC = "p46";

# TFT Connector (J14) sharing I/O with LED and LCD.
NET "tft_cs"    LOC = P33;
NET "tft_dc"    LOC = P30;
NET "tft_reset" LOC = P32;
NET "tft_sck"   LOC = P27;
NET "tft_sdi"   LOC = P29;
```