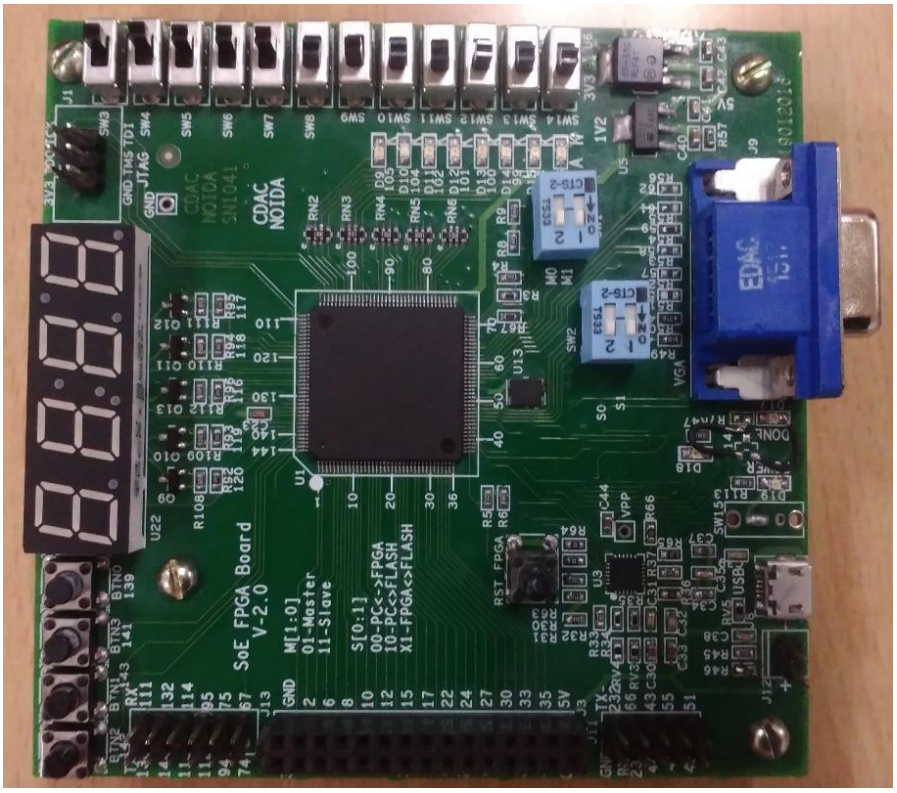
FPGA BOARD

LAB MANUAL



Under the project

"Enhancing the Outreach of Electronic System Design and Training through E-learning"

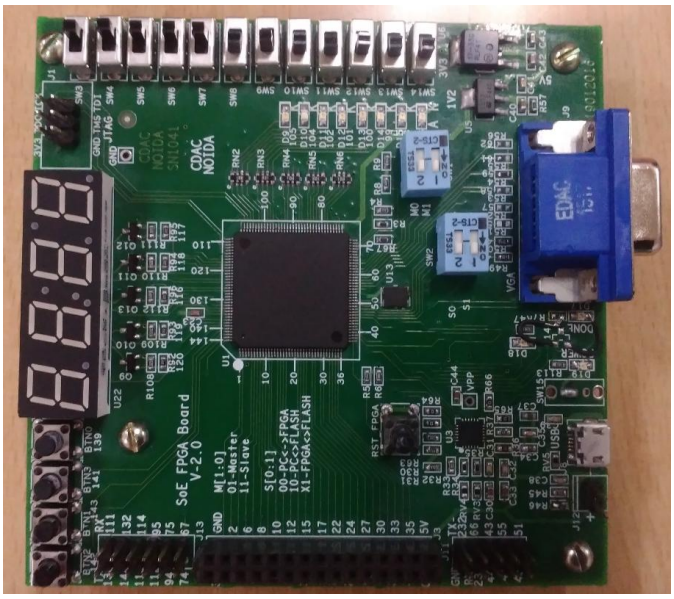
(*A Sponsored project from MeitY Govt. of India*)

By:

Center for Development of Advanced Computing (CDAC)
B-30 , Institutional Area, Sec - 62, NOIDA (U.P) 201307
Phone: +91-120-3063371-73,
Fax.: +91-120-3063374
Website: www.cdac.in

FPGA Board

Lab Manual



Under the project

"Enhancing the Outreach of Electronic System Design and Training through E-learning"

(*A Sponsored project from MeitY Govt. of India*)

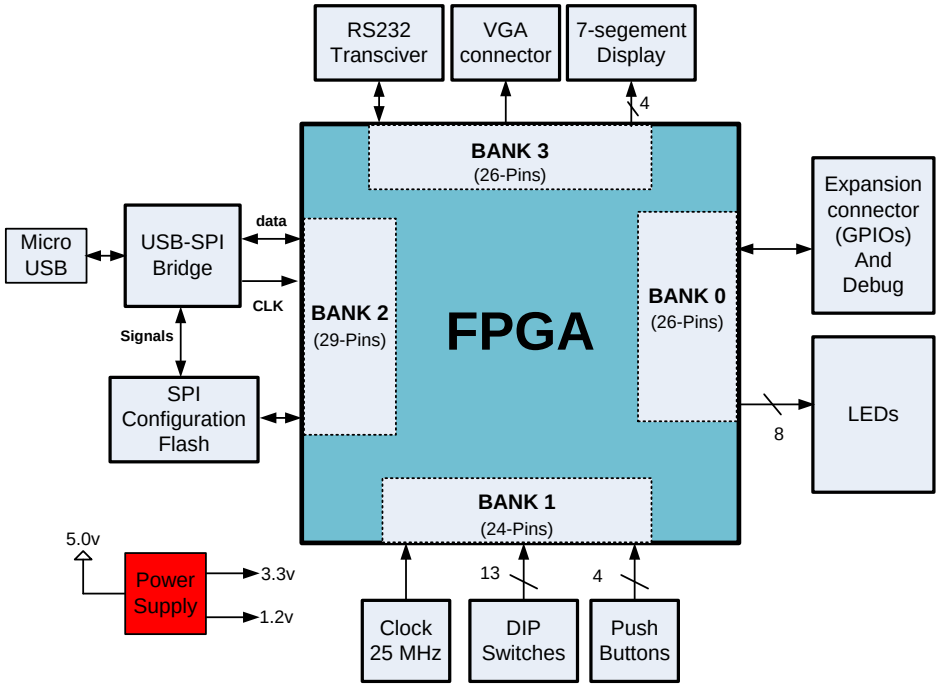
By:
Center for Development of Advanced Computing (CDAC)
B-30 , Institutional Area, Sec - 62, NOIDA (U.P) 201307
Phone: +91-120-3063371-73,
Fax.: +91-120-3063374
Website: www.cdac.in

Contents

ABOUT THE FPGA BOARD

The developed platform is an easy to use Low cost FPGA Development board featuring Xilinx Spartan-6 FPGA. It is specially designed for experimenting and learning system design with FPGAs. The board offers a rich set of features that make it suitable for use in a laboratory environment for university and college graduate courses, for a variety of design projects, as well as for the development of sophisticated digital systems. By default the board is powered from a +5V supply from USB cable.

Built around a Xilinx Spartan-6 FPGA the board provides complete, ready-to-use hardware suitable for hosting circuits ranging from basic logic devices to complex controllers. A large collection of on-board I/O devices/interfaces and all required FPGA support circuits are included, so countless designs can be created without the need for any other components. The design can be controlled and configured through PC using high speed USB interface provided on the board. The board is designed to work with the free ISE Web Pack software from Xilinx



The diagram depicts the complete interface from FPGA banks to on board I/O devices and expansion connectors. The board includes oscillator that produces a reference frequency of 25MHz available for FPGA design. Multiple high frequency and low frequency clocks can be generated as per the design requirement.

The On board peripheral provided on the board are listed as follows:

1. 16 digital inputs via switches and Push buttons.
2. 8 digital outputs via LEDs.
3. 4 digit seven segment display.
4. VGA interface for image/video output

5. UART interface for serial communication
6. High density expansion connectors for external digital I/O interface.

Board Power

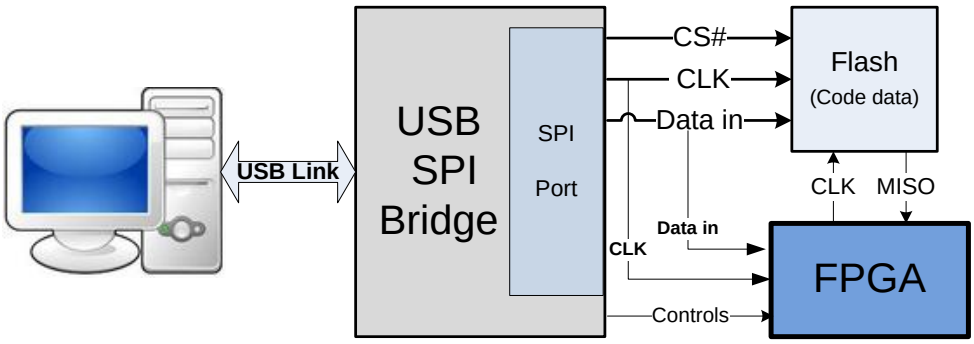
The FPGA board is typically powered from a USB cable, but a battery connector/dc power supply connector is also provided so that external supplies can be used. To use USB power, simply attach the USB cable. Voltages higher than 5V on either power connector may cause permanent damage.

Input power is routed through the power switch (SW15) to the 30-pin expansion connectors, to 800mA low dropout positive voltage regulator LD1117/12 voltage regulator and 1.0A Low dropout positive voltage regulator NCP1117/33. The NCP1117/33 produces the main 3.3V supply for the board. The LD1117/12 produces 1.2V supply voltage required by the FPGA. Total board current is dependent on FPGA configuration, clock frequency, and external connections. Required current will increase if larger circuits are configured in the FPGA, or if peripheral boards are attached.

Configuration

The FPGA board has an on-board USB to SPI controller which facilitates easy reprogramming of on-board SPI flash and FPGA through USB interface. The controller receives bit stream from the host application PC and program it in to the SPI Flash and lets the FPGA boot from the flash.

The USB 2.0 interface provides fast and easy configuration download to the on-board SPI flash. Here we don't need a programmer or special downloader cable to download the bit stream to the board. The FPGA can be configured in master or slave select modes. The device can also be configured and debugged using JTAG programmer. JTAG connector provides access to FPGA's JTAG pins. A XILINX platform cable can be used for JTAG programming. Figure shown below depicts the configuration interface connections.



After power-on, the FPGA on the development board must be configured before it can perform any useful functions. During configuration, a "bit" file is transferred into memory cells within the FPGA to define the logical functions and circuit interconnects. The free ISE/WebPack CAD software from Xilinx can be used to create bit files from VHDL, Verilog, or schematic-based source files.

CDAC's PC-based program called CLR13.0 can be used to configure the FPGA with any suitable bit file stored on the computer. CLR13.0 uses the USB cable to transfer a selected bit file from the PC to the FPGA. CLR13.0 can also program a bit file into an on-board non-volatile ROM called "SPI Flash". Once programmed, the SPI Flash can automatically transfer a stored bit file to the FPGA at a subsequent power-on or reset event. The FPGA will remain configured until it is reset by a power-cycle event. The SPI Flash ROM will retain a bit file until it is reprogrammed, regardless of power-cycle events.

The following are the switch configuration for the FPGA board for configuring the FPGA chip:

SW1: 11- FPGA programming through PC
01- FPGA configuration through Flash

SW2: 00- FPGA programming through PC
10- FPGA configuration through Flash

To program the development board, attach the USB cable to the board. Start the **FPGA** programmer and use the load file function to associate the desired bitstream file with the FPGA, or with the SPI Flash. Select the "Program" function for configuring the FPGA directly from the PC. Tick "Flash" function check box if you want to store data to the SPI Flash. The configuration file will be sent to the FPGA or SPI Flash, and the software will indicate whether programming was successful. The "Status LED" LED (D17) will glow after the FPGA has been successfully configured.

Oscillator

The development board includes a primary *Silicon Labs 501JAA25M0000BAG CMEMS* oscillator that produces 25Mhz.

User I/O

Four pushbuttons and twelve slide switches are provided for circuit inputs. Pushbutton inputs are normally low and driven high only when the pushbutton is pressed. Slide switches generate constant high or low inputs depending on position. Pushbuttons and slide switches all have series resistors for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).

Eight LEDs and a four-digit seven-segment LED display are provided for circuit outputs. LED anodes are driven from the FPGA via current-limiting resistors, so they will illuminate when logic '1' is written to the corresponding FPGA pin.

The development board uses 10 FPGA signals to create a VGA port with 8-bit color and the two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). A video controller circuit must be created in the FPGA to drive the sync and color signals with the correct timing in order to produce a working display system.

The FPGA board has two serial communication pins – the receiver, RX for receiving the data and the transmitter, TX for sending the data. The RX pin of the FPGA board should be connected to the TX of

the other device with which one wants to communicate. Similarly, the TX pin of the FPGA board will be connected to the RX of the other device for the serial communication.

The development board provides 30-pin peripheral module ports. It provides 5V, 3.3V and two signals. Several module boards/devices can be interfaced to the board through these pins. For example A/D converters, speaker amplifiers, microphones, sensors etc.

EXPERIMENT: 1

Objective: To design and implement two input OR gate on the FPGA Board.

Software: Xilinx ISE Design Suite14.5

Target Hardware: FPGA Board

Theory: The OR gate is a digital logic gate. It behaves according to the truth table given below:

Table 1.1: Truth Table OR Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Equation for the OR Gate is **Y=A+B.**

A HIGH output (1) results if one or both the inputs to the gate are HIGH (1). If neither input is high, a LOW output (0) results.

Procedure:

1. To start the **Xilinx ISE design Suite 14.5**, double-click on the **Project Navigator** icon on your desktop, or select Start => All Programs => Xilinx ISE Design Suite => Xilinx ISE Design Suite 14.5 => ISE =>Design Tools => Project Navigator.

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

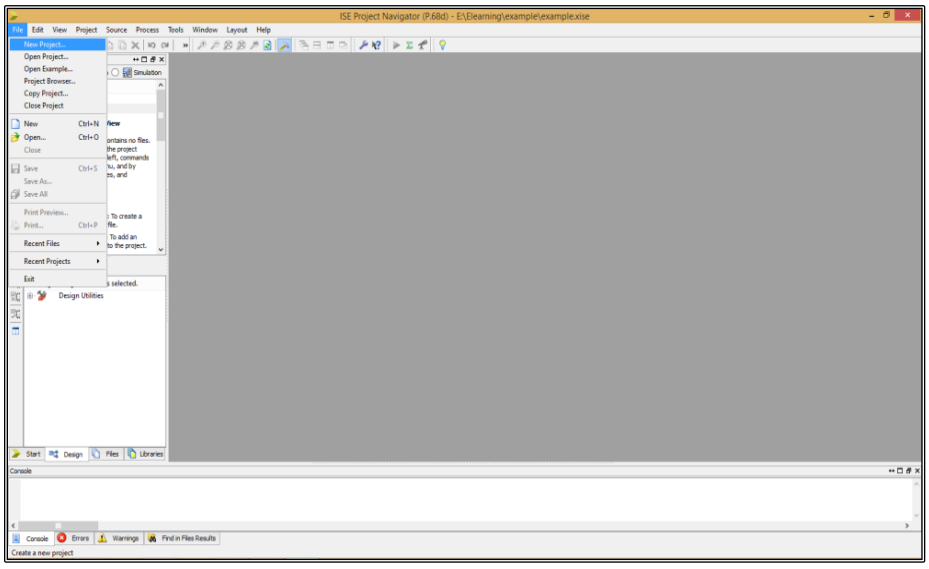


Fig 1.1: ISE Project navigator interface

3. The **New Project Wizard** window will be displayed. In the **Name** field type OR_Logic (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 C:\Project\OR_Logic (Fig. 1.2)

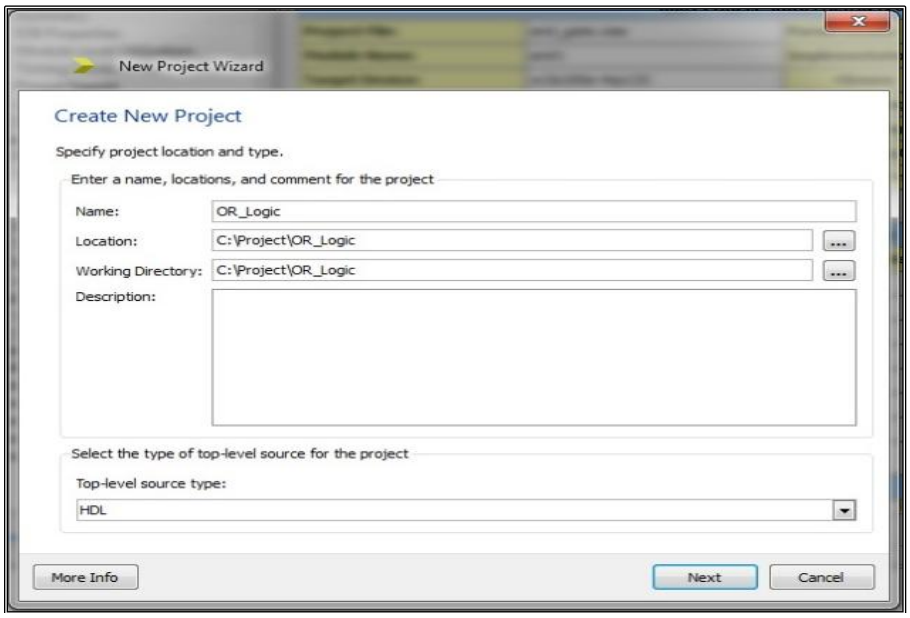


Fig 1.2: New Project Wizard window

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 displayed.

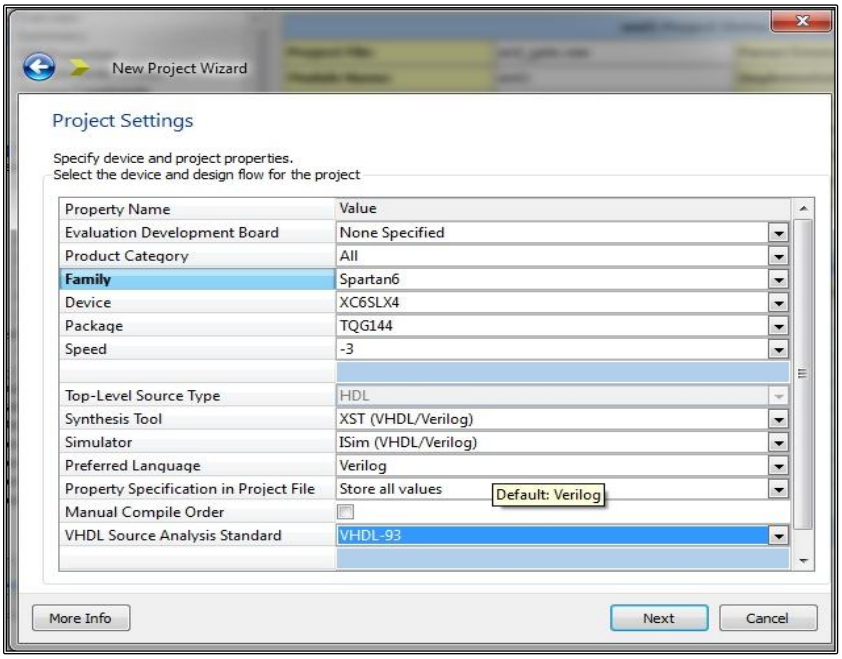


Fig 1.3: Project setting page

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

- **Product Category:** All
- **Family:** Spartan6
- **Device:** XC6SLX4
- **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.

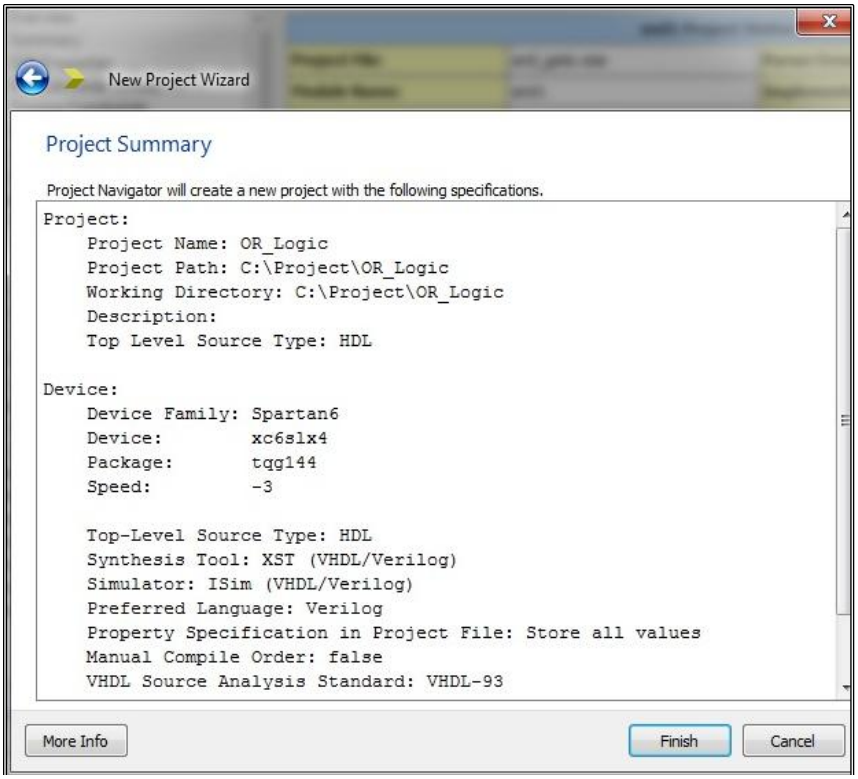


Fig 1.4: Project Summary page

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

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

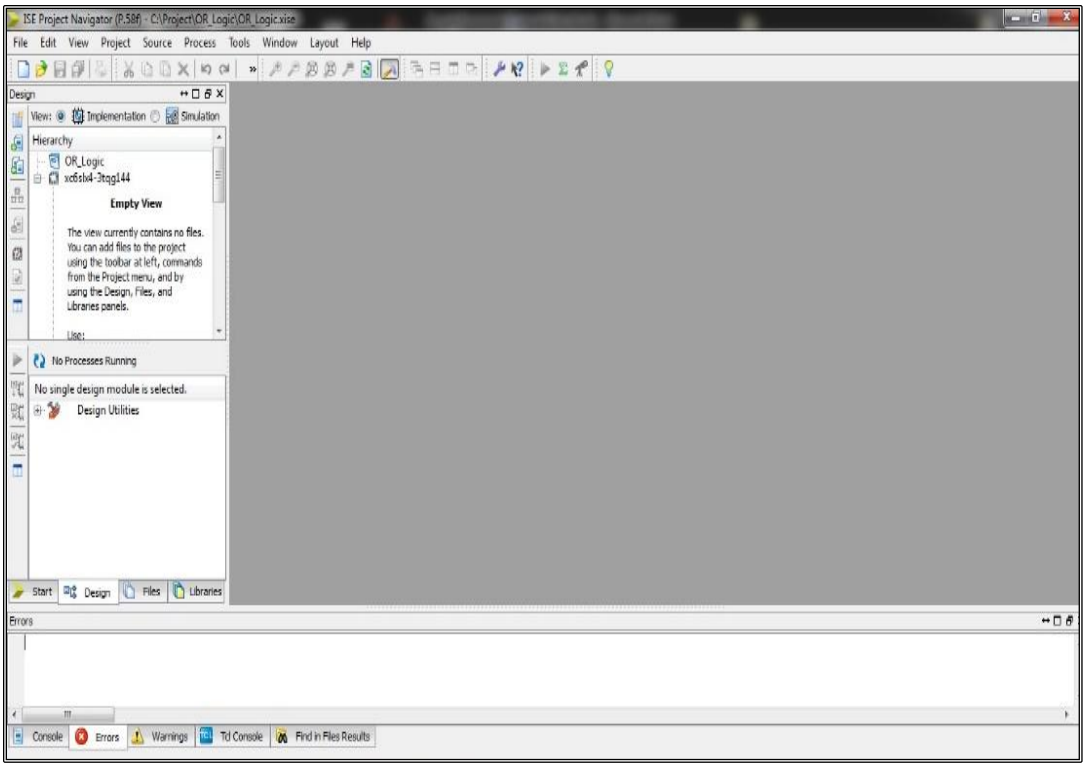


Fig 1.5: ISE Project Navigator

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 "OR_gate" for the new source in the **File Name** field. Click **Next.**

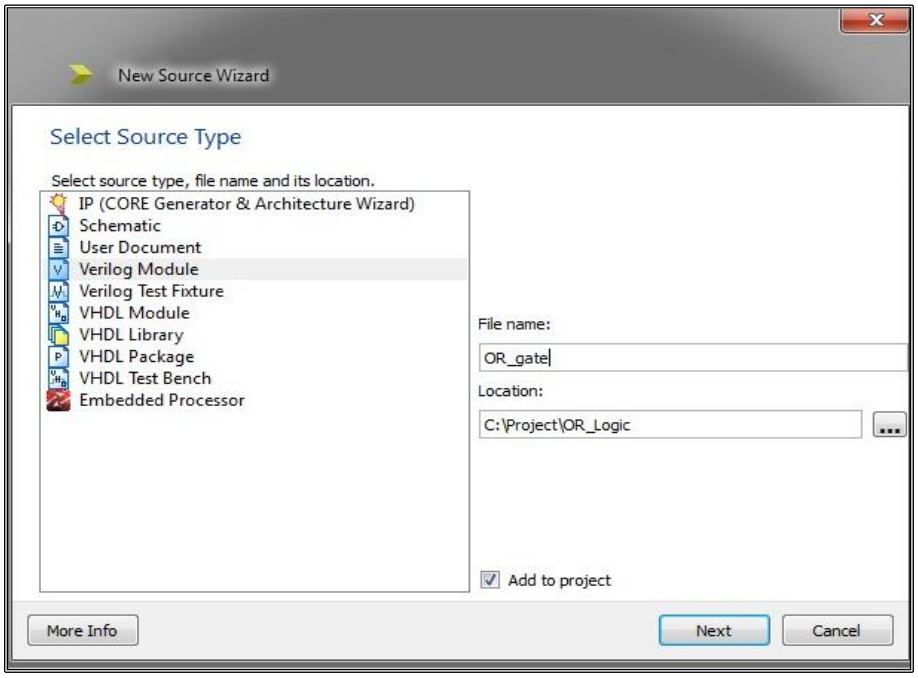


Fig 1.6: Select Source Type

8. In the **Define Module** page, enter the port information for the "OR_gate" as follows (Fig 1.7):
   a. In the first three **Port Name** field, enter 'y', 'a' and 'b'.
   b. Set the **Direction** field to output for 'y' and to input for 'a' and 'b'.
   Leave the **Bus** check boxes as it is.

Click **Next**.

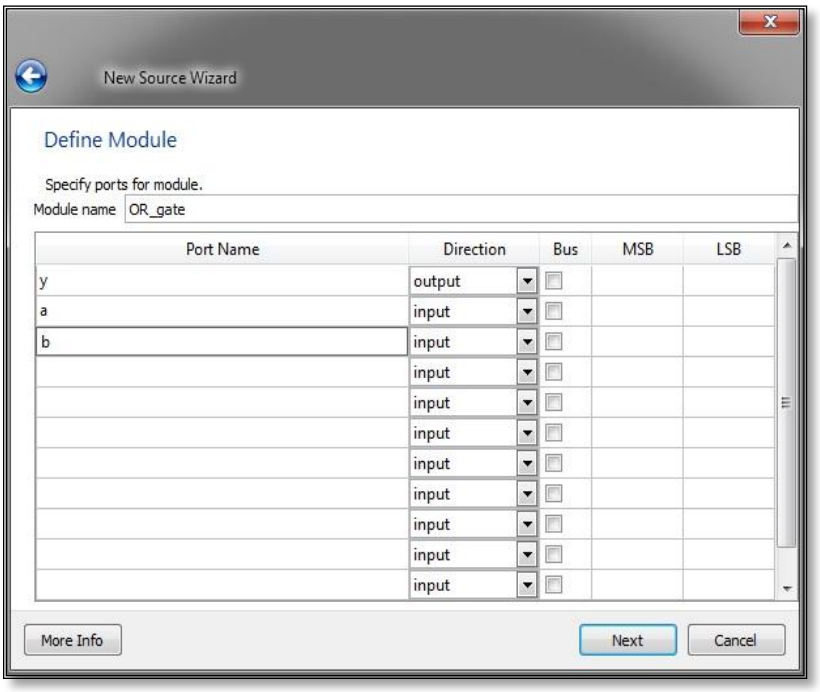


Fig 1.7: Define Module window

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

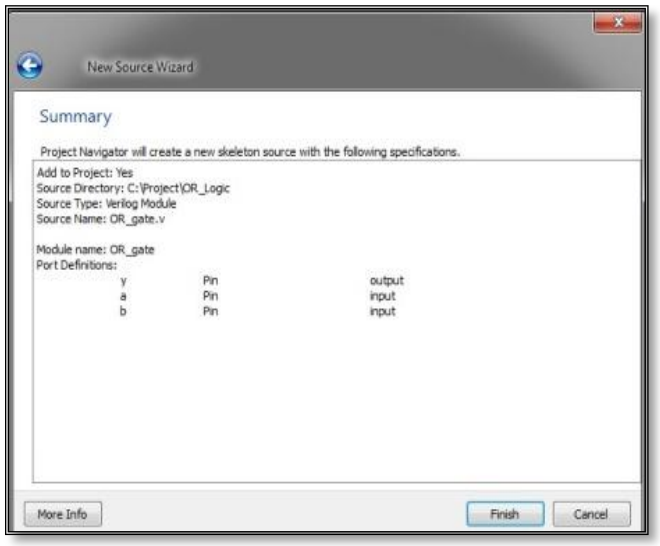


Fig 1.8: Summary Window

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

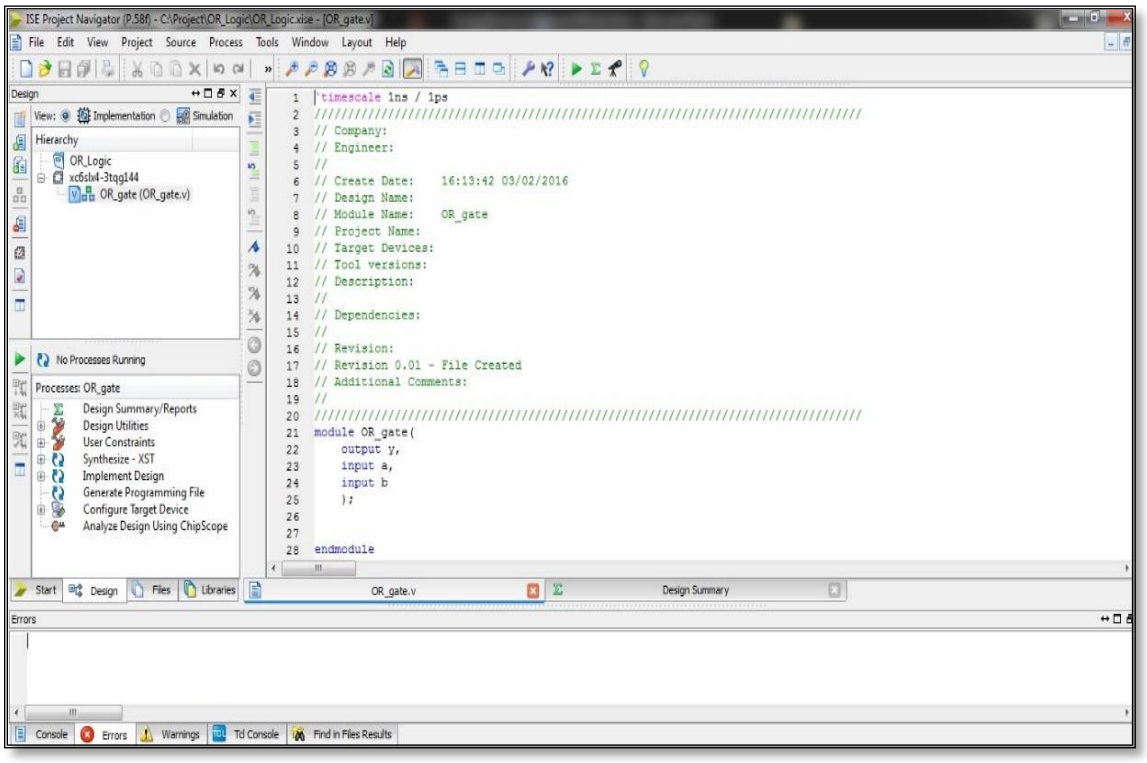


Fig 1.9: OR gate Structure in the workspace

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 OR logic gate. 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:    16:13:42 03/02/2016
7  // Design Name:
8  // Module Name:    OR_gate
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 OR_gate(
22     output y,
23     input a,
24     input b
25     );
26
27 assign y= a|b;
28 endmodule
```

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 "OR_gate" displays in the Source tab (Fig 1.11).

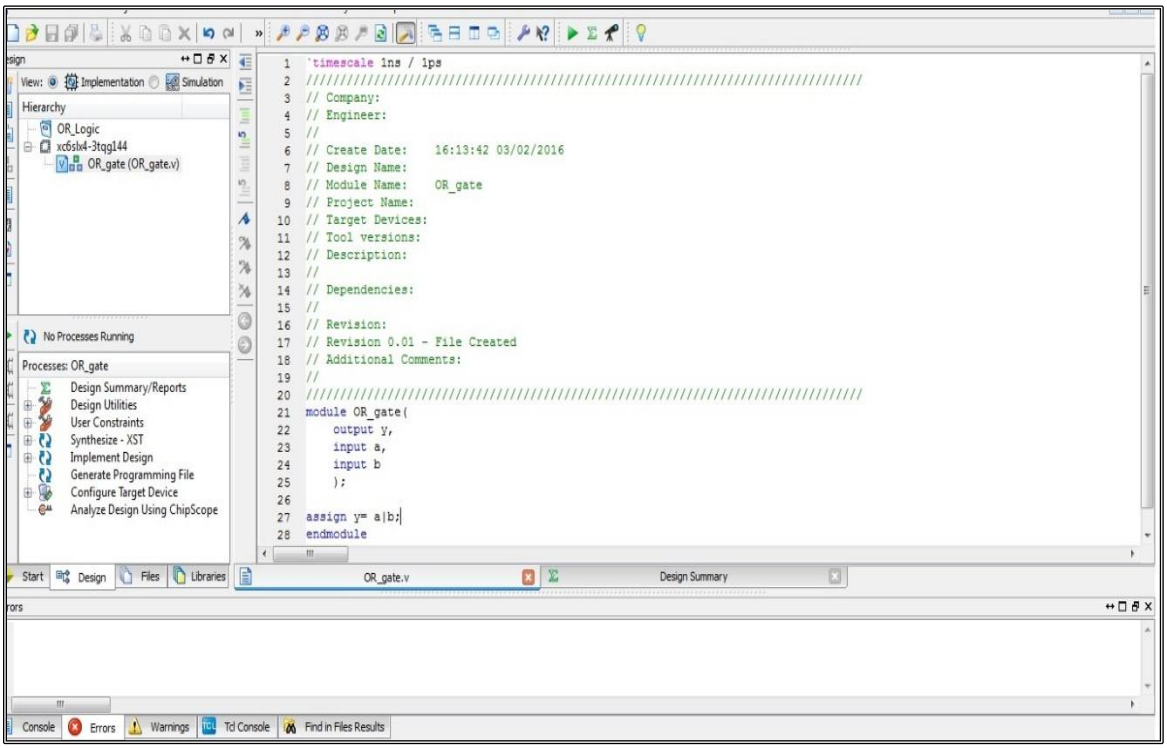


Fig 1.11: Source Tab with complete Verilog code

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.

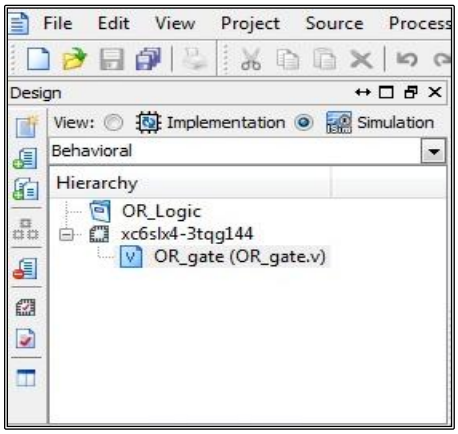


Fig 1.12: Hierarchy window

14. Now in **Simulation window** select the source file "OR_gate.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.

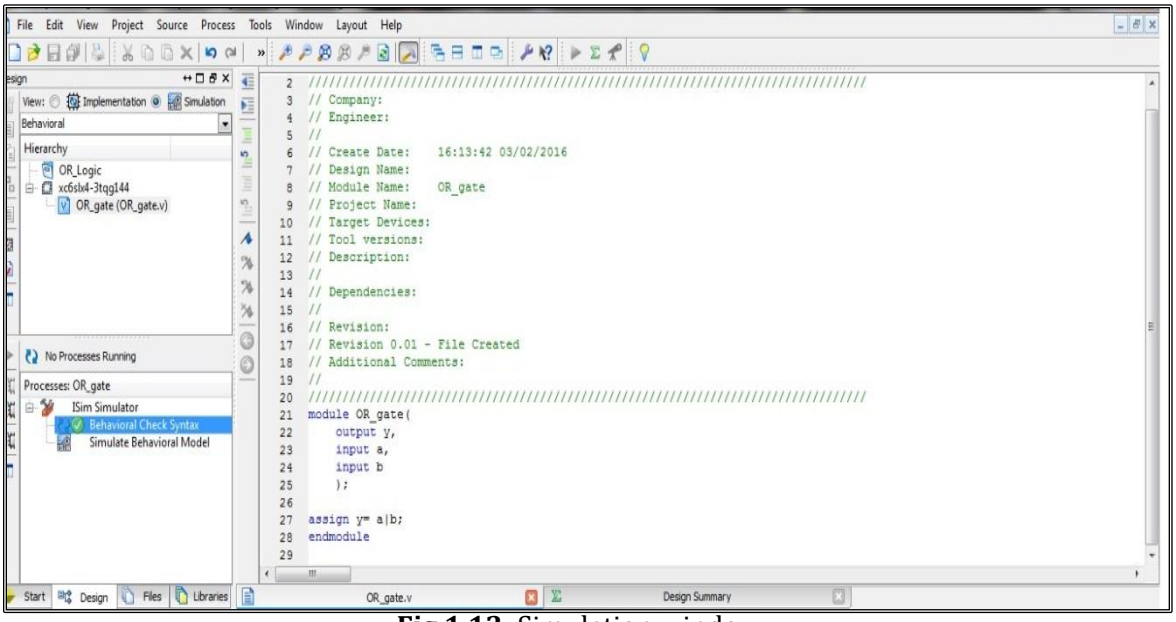


Fig 1.13: Simulation window

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 the two inputs "OR_gate" the test bench can be created by one of the following methods. Either right click on the design (OR_gate.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.

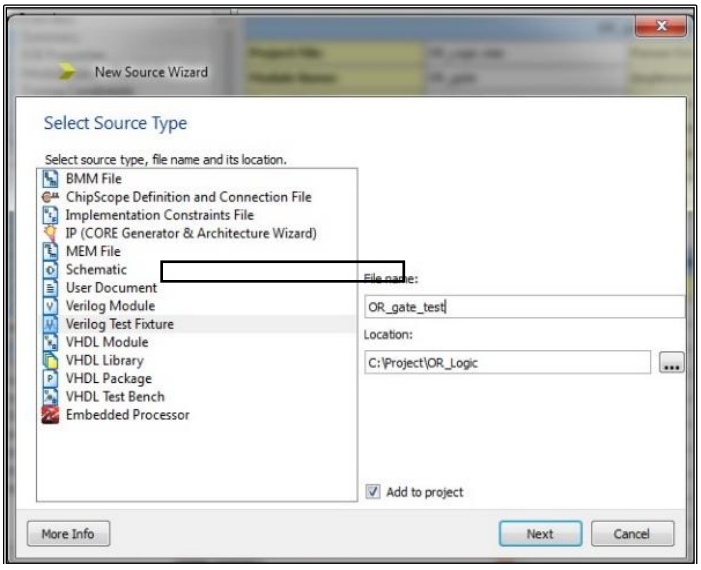


Fig 1.14: Verilog Test bench

16. Write the test bench name in the File name field. Here the test bench name is "OR_gate_test". 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.**

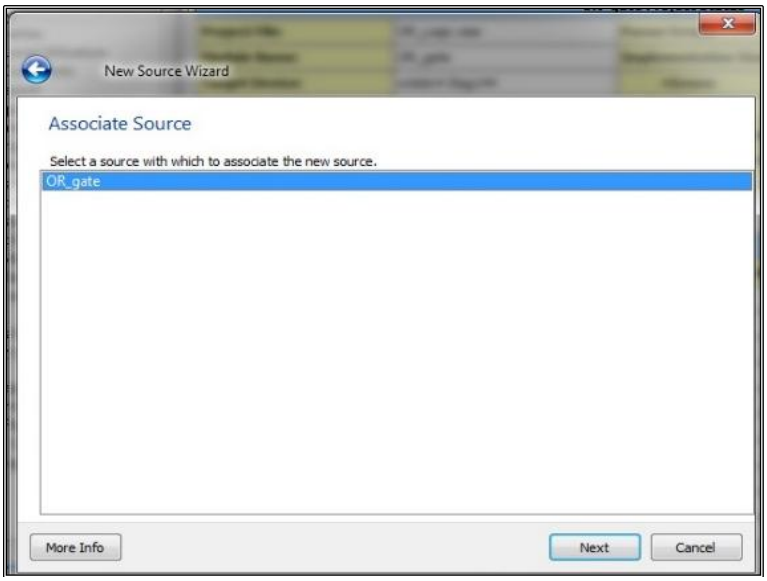


Fig 1.15: Associate Source

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

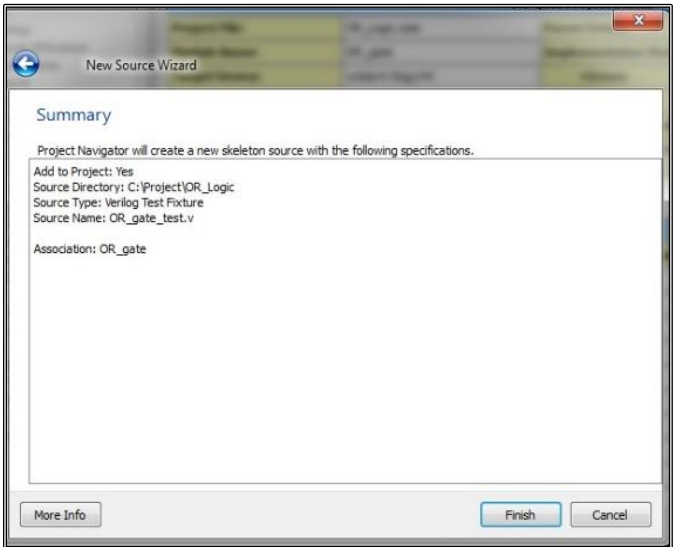


Fig 1.16: Summary window

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

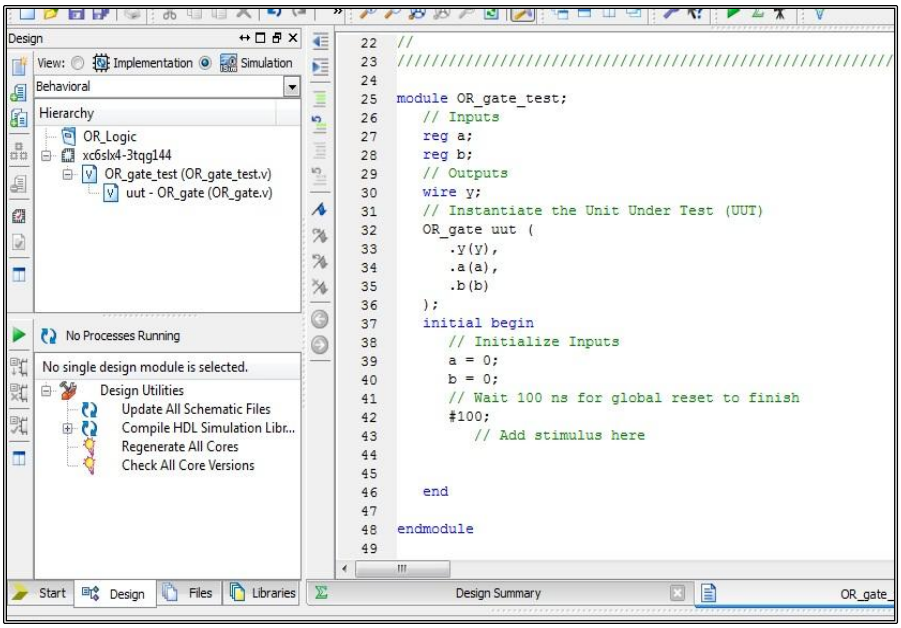


Fig 1.17: Test bench template

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' and 'b' after 100 units of time delay. After writing all the input combinations for 'a' and 'b' save the code (Fig 1.18).

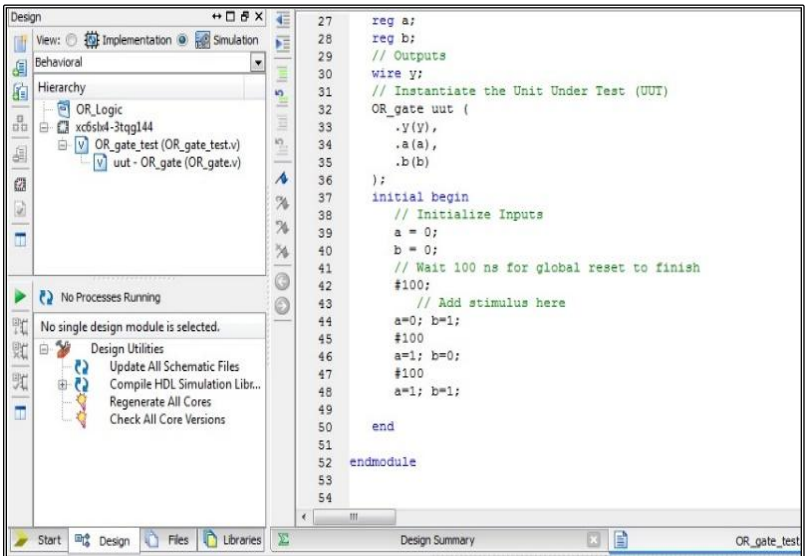


Fig 1.18: Complete Testbench

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)

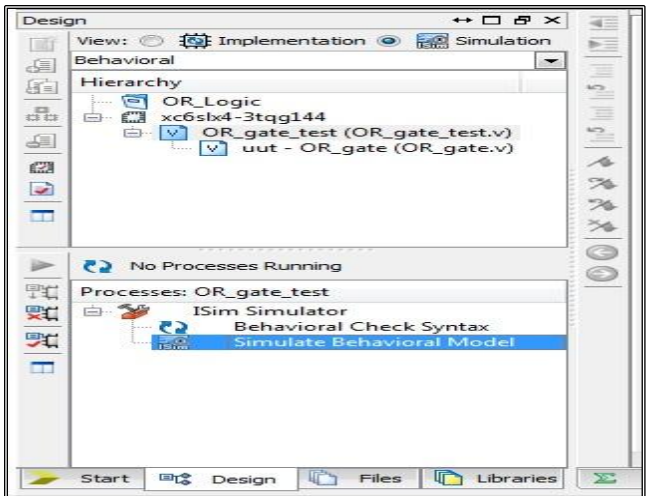


Fig 1.19: Process Window

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

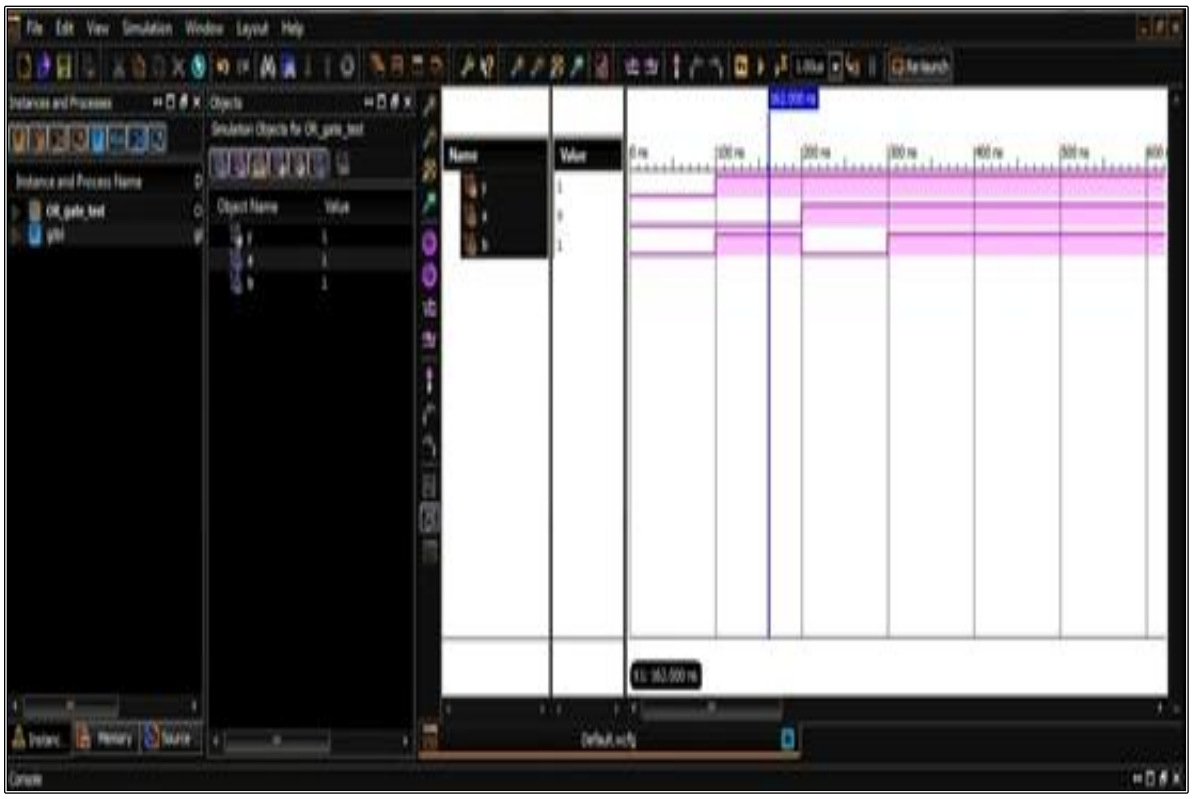


Fig 1.20: Simulation Result

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.

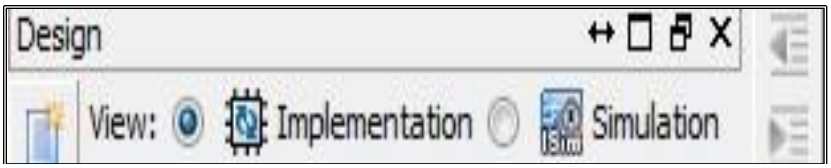


Fig 1.21: Hierarchy Window

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** as shown in Fig 1.22. 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.

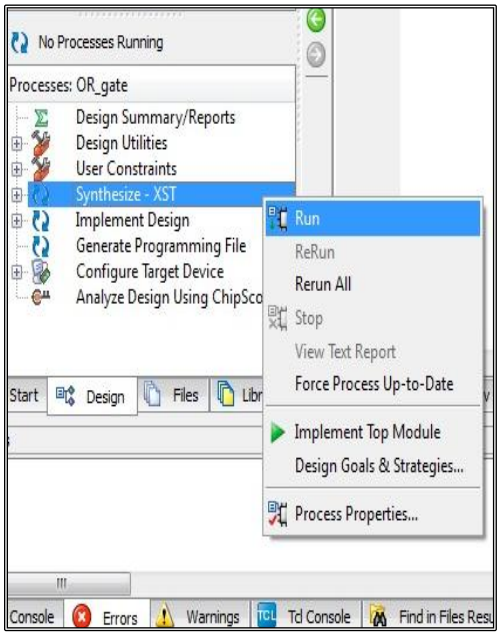


Fig 1.22: Processes Window-Synthesis

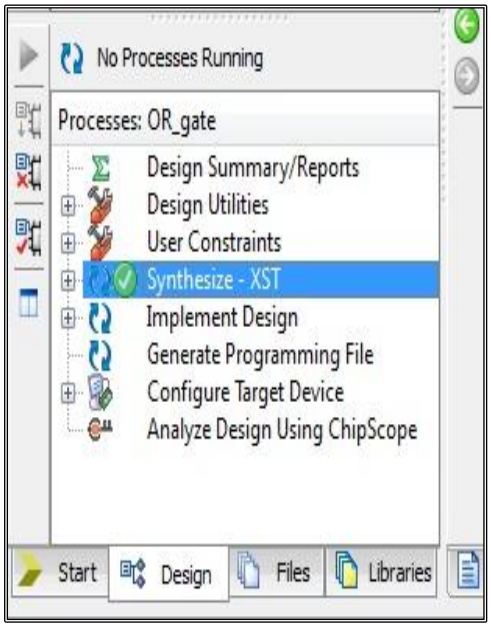


Fig 1.23: Synthesis successful

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**.

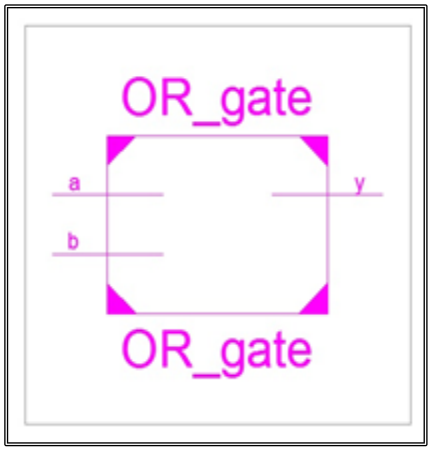


Fig 1.24: RTL Schematic

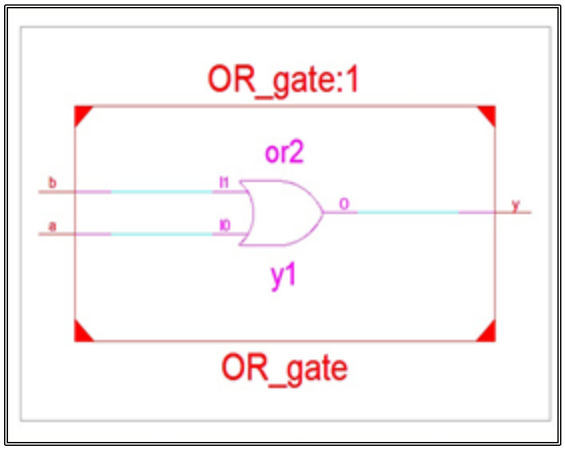


Fig 1.25: Internal View of RTL schematic

Fig 1.24 represents the **RTL schematic view** of "OR_gate.v" and to see a detailed view of the RTL schematic double click on the Fig 1.24 you will get the internal gate level diagram as shown in the Fig 1.25. 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 (OR_gate.v) from Hierarchy section and select **New Source** (Fig 1.26) or click **New Source** from the **Project menu** (Fig 1.27). From the New Source Wizard window choose **Implementation Constraints File.**

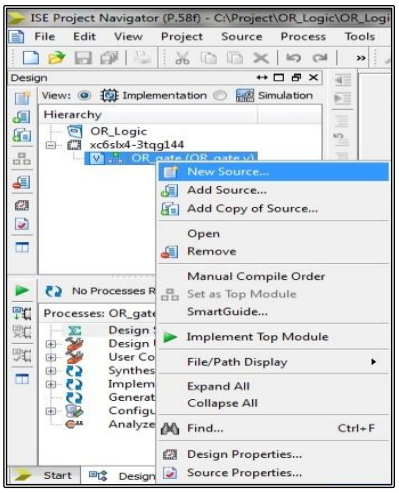


Fig 1.26: New Source (Method 1)

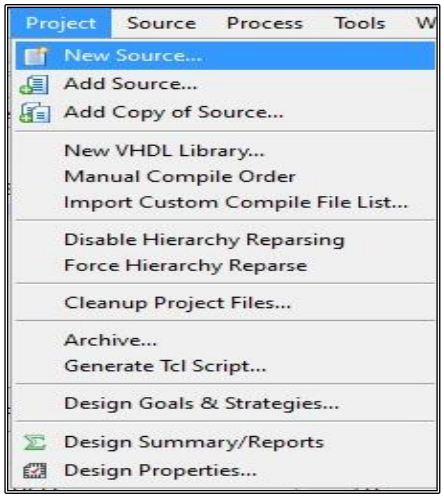


Fig 1.27: New Source (Method 2)

27. In **Implementation Constraints File** enter the name of the UCF to be generated (Fig 1.28).

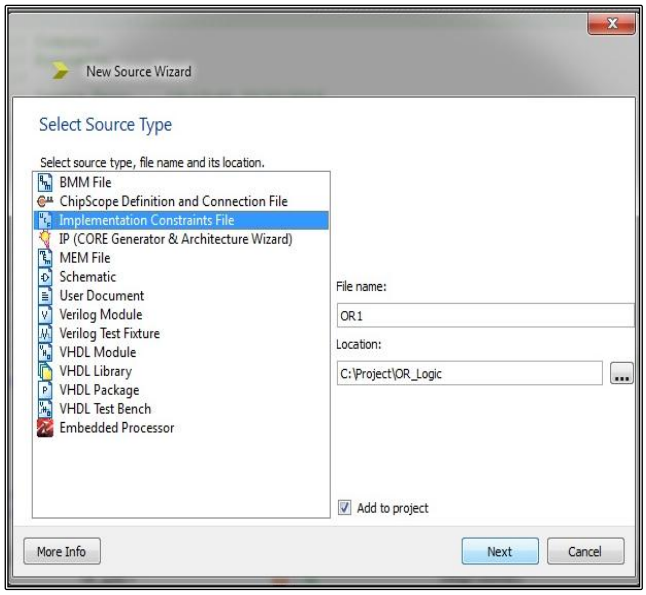


Fig 1.28: Implementation constraints file

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 1.29).Select **Finish**.

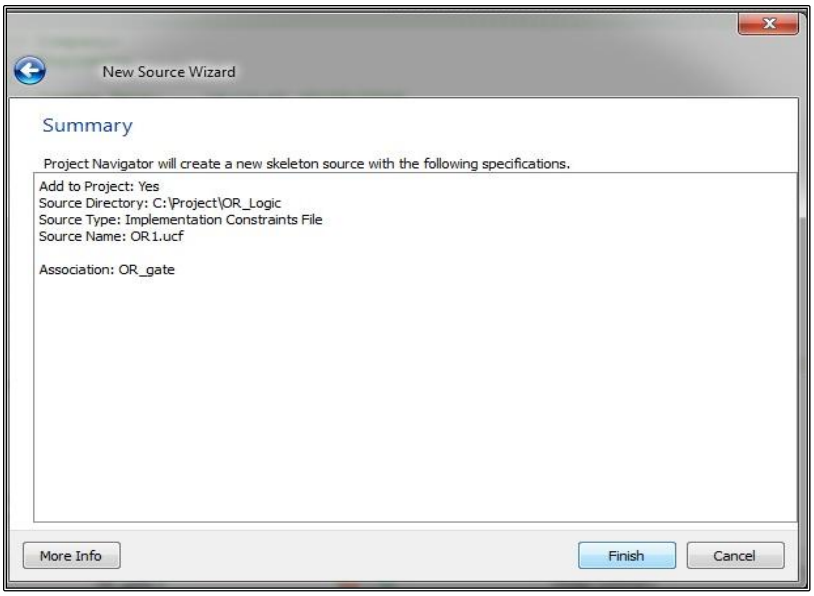


Fig 1.29: Summary Window

29. A blank window will appear in a new tab having the design name with .ucf extension. Here the tab "OR1.ucf" is generated as shown in the Fig 1.30.

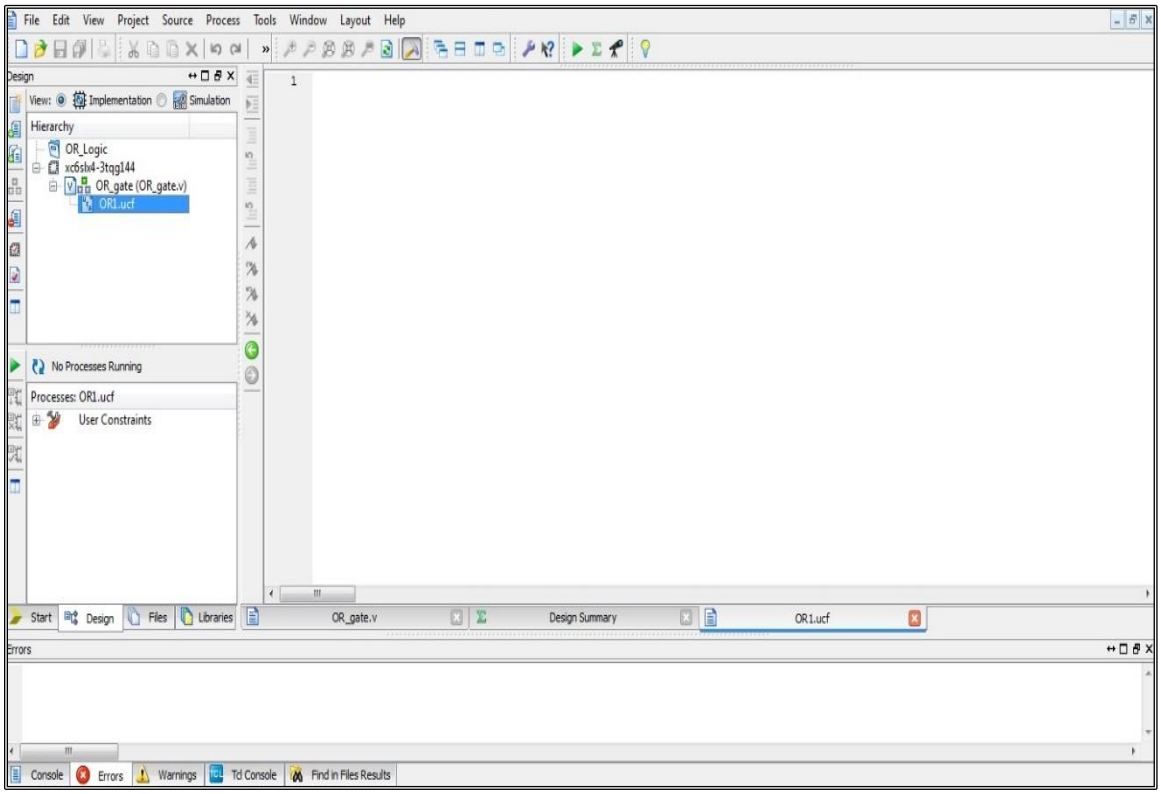


Fig 1.30: Blank UCF

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 "OR_gate" module is as follows:

NET "a" LOC = P93;
NET "b" LOC = P92;
NET "out" LOC = P105;

After adding the constraints save the UCF.

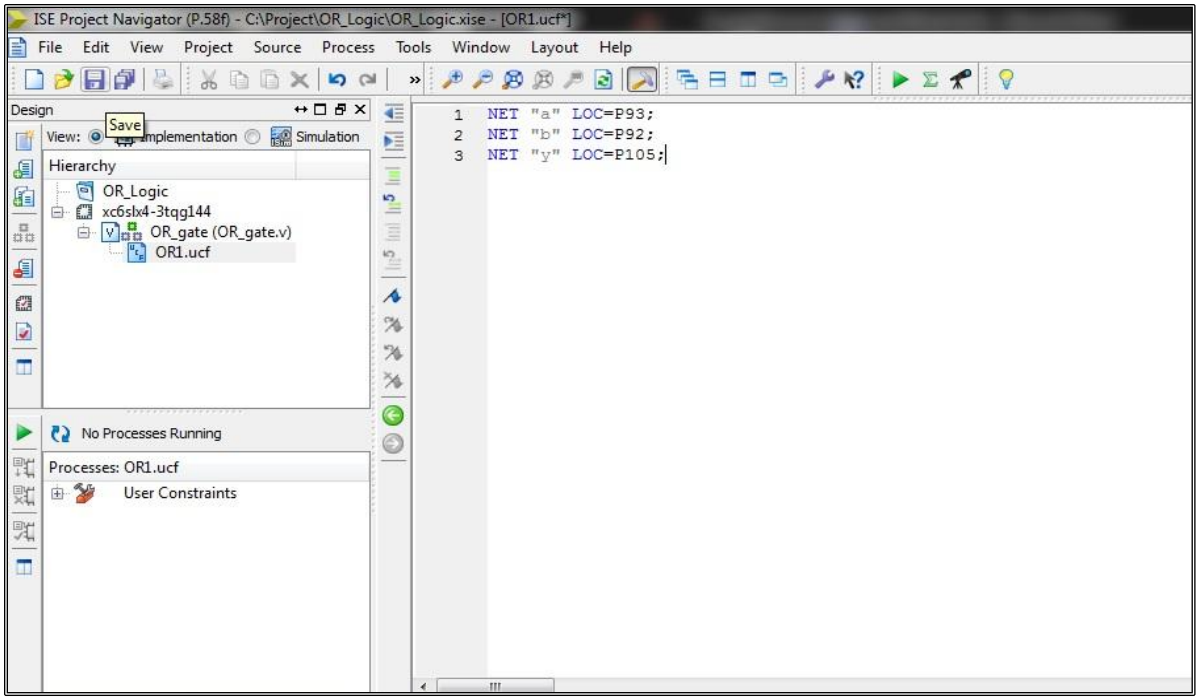


Fig 1.31: UCF for OR gate

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** as shown in Fig 1.32. If the design is implemented successfully then green check will appear near the Translate, Map and Place & Route process as shown in Fig1.33.

32. Now right click on the **Generate Programming file** and select **Run** as shown in Fig 1.34. This will produce the bitstream file. The bitstream 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.

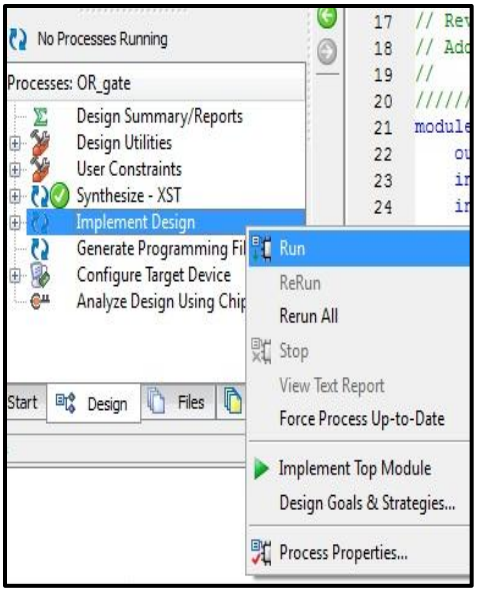


Fig 1.32: Implement Design

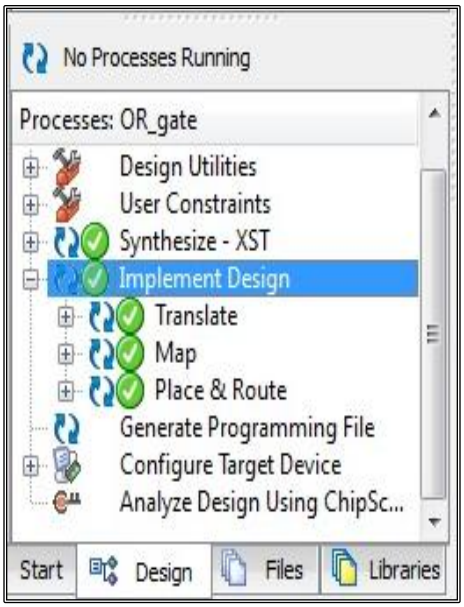


Fig 1.33: Implementation successful

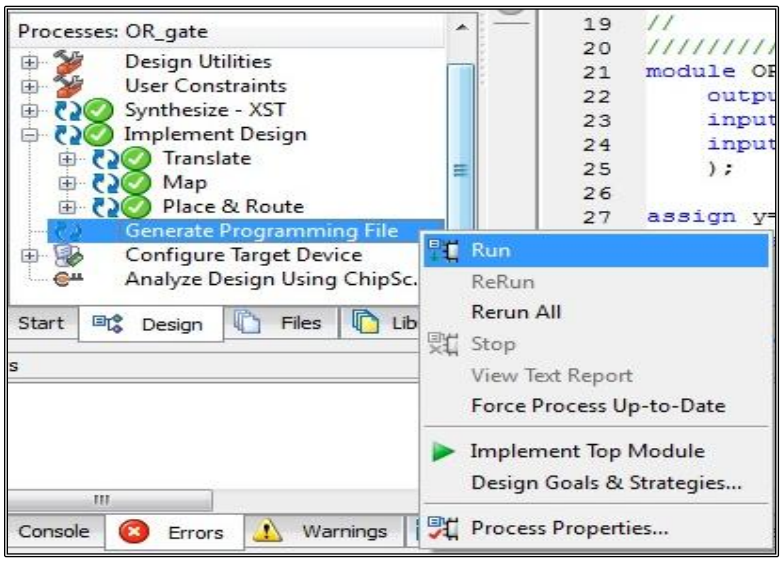


Fig 1.34: Generate Programming File

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 programming the FPGA, connect the board to the computer through the USB cable provided and switch on the power supply of the board.
35. Open **CDAC FPGA Board** programmer (Fig 1.35).

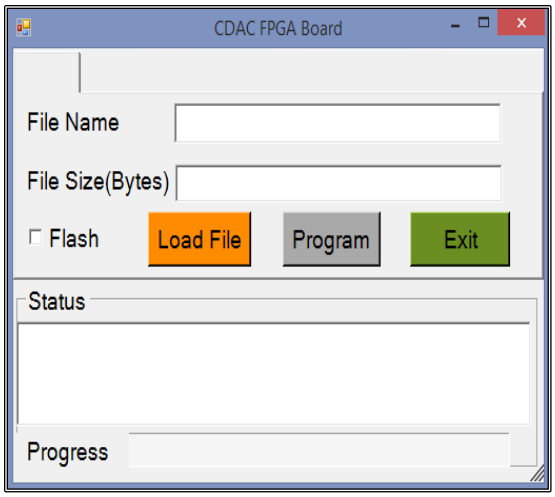


Fig 1.35: CDAC FPGA Programmer

36. Click **Load File** tab and navigate to the directory containing the desired folder in which your project is stored. Select the 'bit' or 'bin' file.

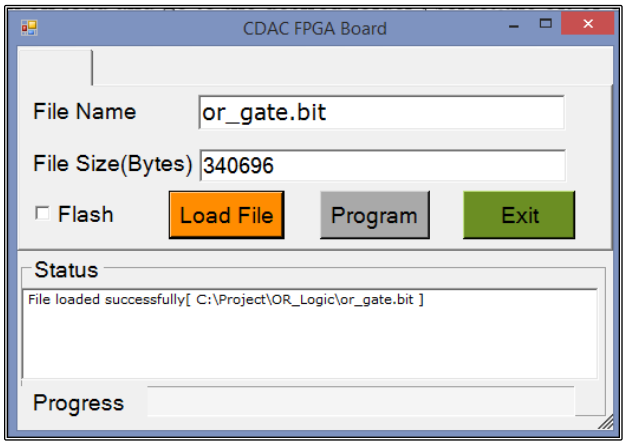


Fig 1.36: or_gate.bit

37. The size of the bit file will be displayed in "**File Size**" field. If file is loaded properly then in status window message "**File loaded successfully**" will be displayed.

38. Now after loading the file, you have to program the board. Click "**Program**" to program the FPGA.

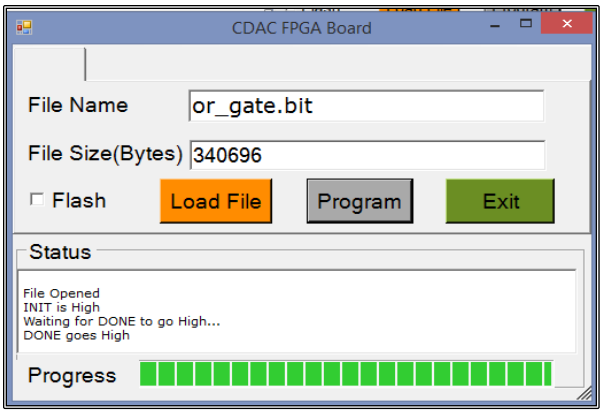


Fig 1.37: Programming the FPGA

39. When the FPGA is programmed successfully then a message "**DONE goes High**" 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.

40. With the board having been programmed you can now verify your design. Remember, from the User/Implementation constraints file, that the two switches are used as inputs (SW3 for "a" and SW4 for "b") and a LED (D9) is used for "y". Provide various input combinations using the switches and verify that the outputs are correct.

Verilog code for the OR gate

```
module and_gate(
  output y,
  input a,
  input b
  );
assign y=a|b;
endmodule
```

User Constraints File (UCF) For OR gate

```
NET "a" LOC = "P93";
NET "b" LOC = "P92";
NET "y" LOC = "P105";
```

EXPERIMENT: 2

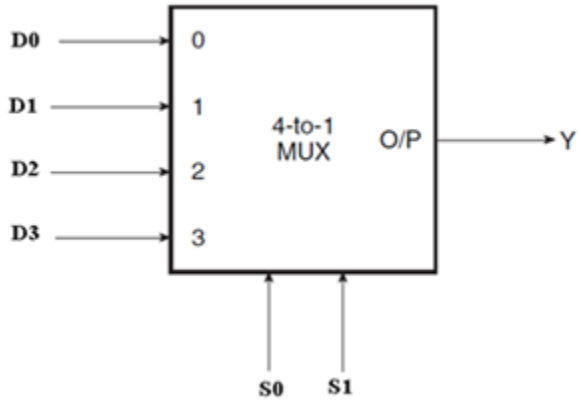
Objective: To Design and implement 4:1 Multiplexer on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: A multiplexer is a device that selects one of several digital input signals and forwards the selected input into a single line. A multiplexer of 2n inputs has n select lines, which are used to select which input line will be passed to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. A multiplexer is also called as a data selector. Multiplexers can also be used to implement Boolean functions of multiple variables.

The block diagram and the truth table for 4:1 Multiplexer is shown in Fig 2.1.



S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Fig 2.1: 2:1 MUX and its truth table

Procedure:

1. To start the **Xilinx ISE design Suite 14.5**, double-click on the **Project Navigator** icon on your desktop, or select Start => All Programs => Xilinx ISE Design Suite => Xilinx ISE Design Suite 14.5 => ISE =>Design Tools => Project Navigator.

2. The **ISE Project navigator** interface will be opened. In Project Navigator, select **File => New Project.**

3. The **New Project Wizard** window will be displayed. In the **Name** field, type "Multiplexer" and in the **Location** field, browse to C:\Project or choose the directory in which you want to store the project. Select **HDL** as the Top-Level Source Type and click **Next** (Fig 2.2).

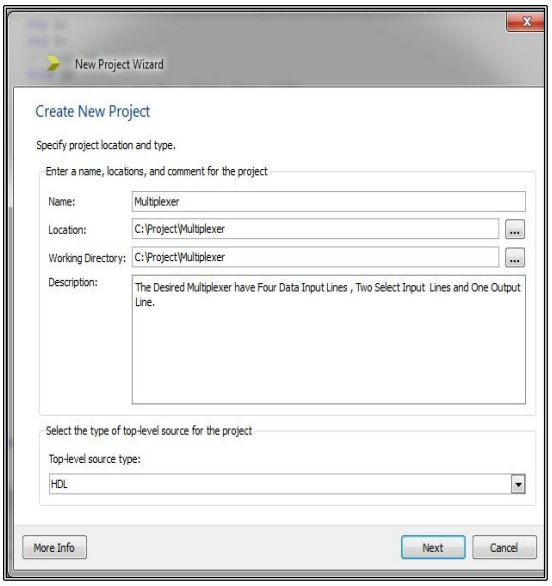


Fig 2.2: New Project

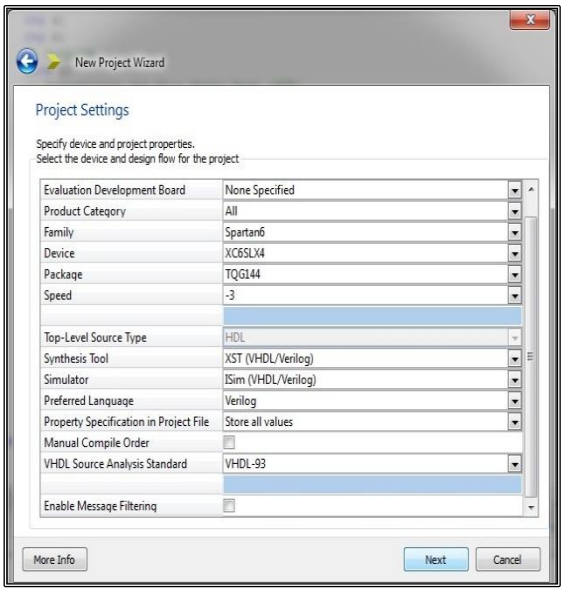


Fig 2.3: Project Properties

4. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties (Fig 2.3):
   - **Product Category**: All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator**: ISim (VHDL/Verilog)
   - **Preferred Language:** VHDL or Verilog depending on the preference. For our project we will 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. Verify the **Project Summary** and click **Finish** (Fig 2.4).

6. Now in the ISE Project Navigator Environment, select **Project -> New Source**. Select **Verilog Module** as the **Source Type** and enter a name "Mux" for the new source in the **File Name** field (Fig 2.5). Click **Next**.

7. In the **Define Module** page, enter the port information for the "Mux" as follows (Fig 2.6):
   a. In the first three **Port Name** fields, enter 'I', 'Sel' and 'Y'.
   b. Set the **Direction** field to input for 'I', 'Sel' and output for 'Y'.
   c. Since the input 'I' is a 4-bit vector, tick the **Bus** checkbox and set its Bus range from 3(MSB) to 0(LSB). Similarly for the input 'Sel' set the Bus range from 1(MSB) to 0(LSB) as it is a 2-bit vector.

Click **Next**.

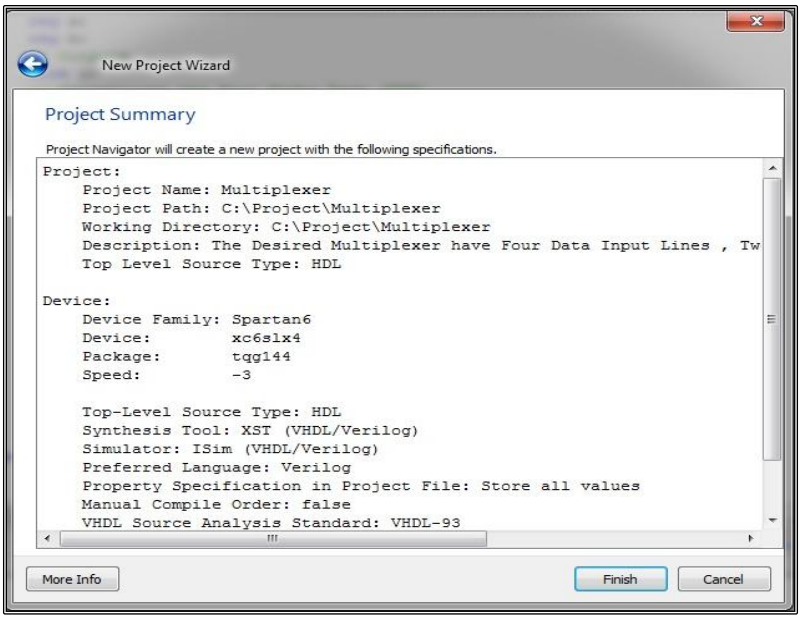


Fig 2.4: Project Summary

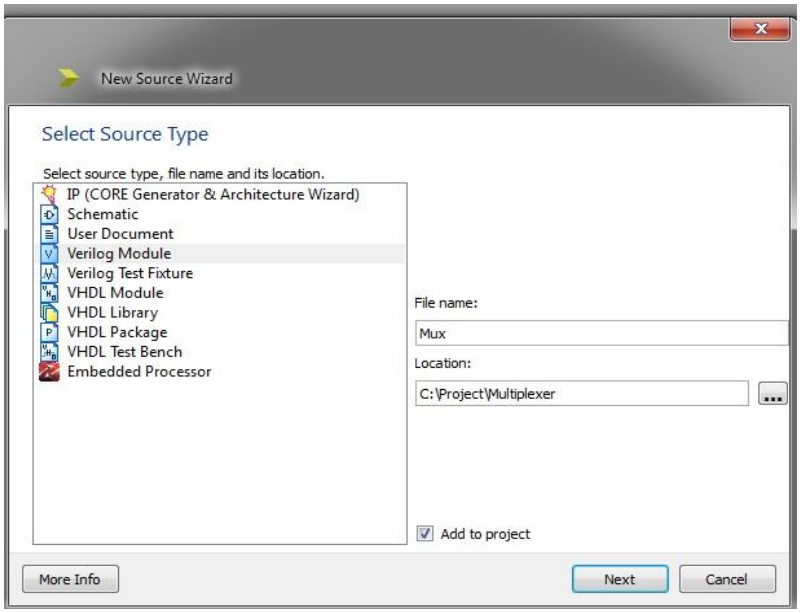


Fig 2.5: New Source Wizard

7.

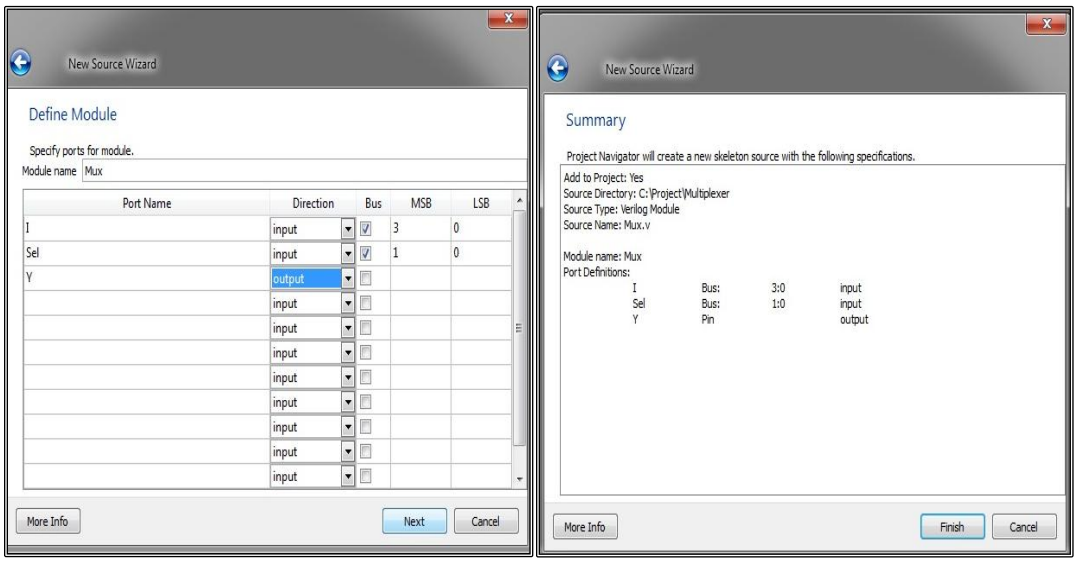


Fig 2.6: Module Definition **Fig 2.7:** Module Summary

8. Next the summary of the module will be displayed (Fig 2.7). Check the source summary and click **Finish**.

9. In the **Hierarchy window** you can see that the new source file "Mux.v" has been added to the project "Multiplexer".

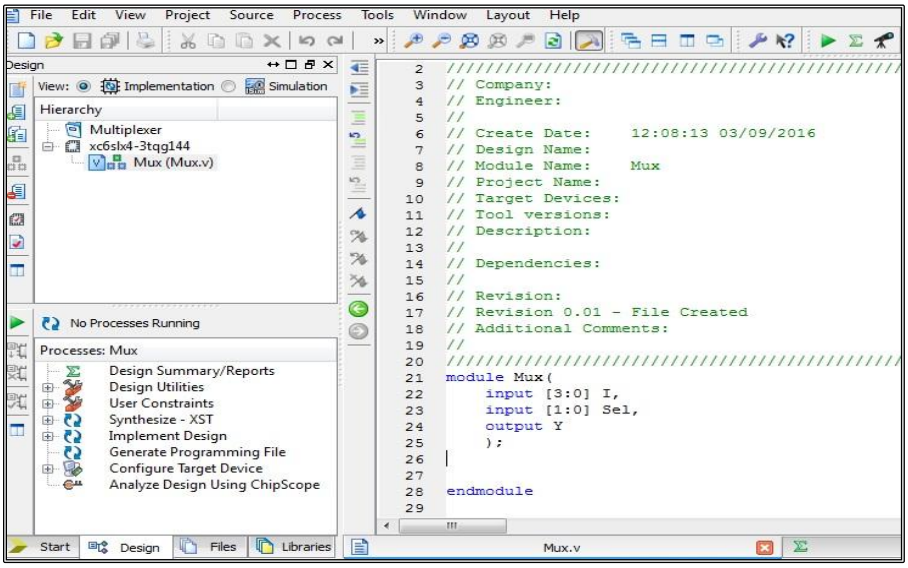


Fig 2.8: Mux.v added to the project

10. 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 (Fig 2.9). Now you need to complete the source code for the multiplexer. For this example we have used behavioral modeling (Fig 2.10).

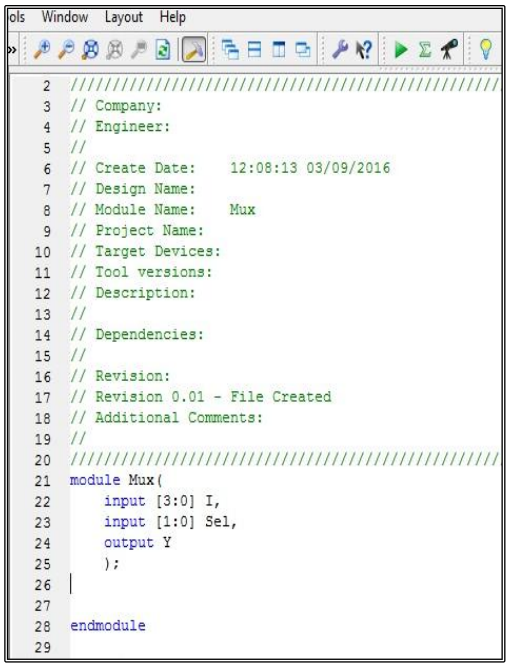


Fig 2.9: Incomplete Mux module

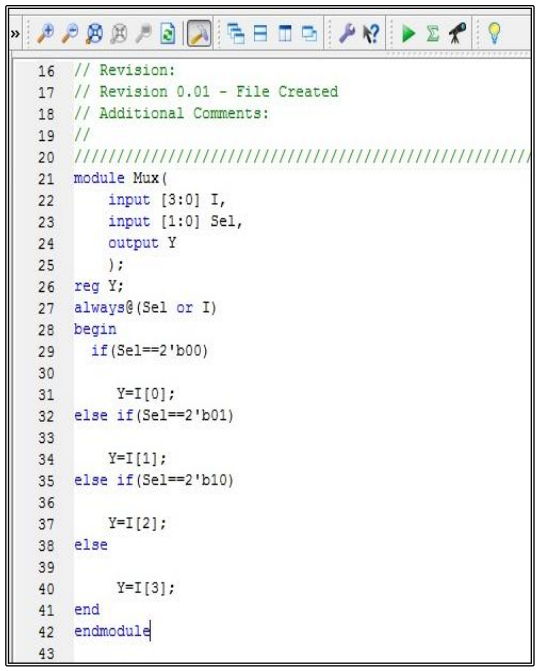


Fig 2.10: Complete Mux module

11. Save the module. After writing the code next step is to check the syntax and functionality of the design. For this we have to select **Simulation** button in the **Hierarchy window** on the left pane (Fig 2.11).

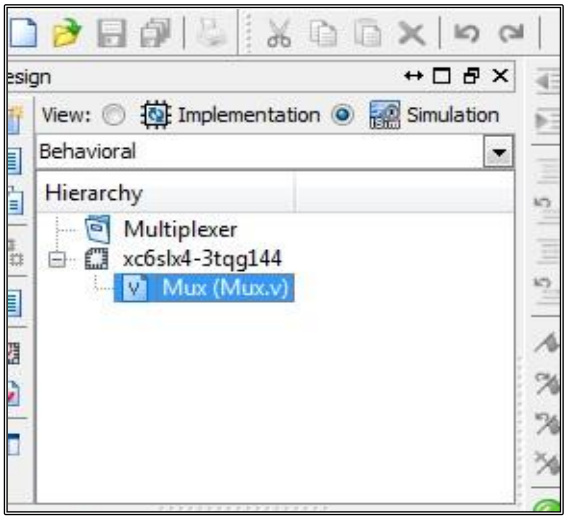


Fig 2.11: Simulation

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

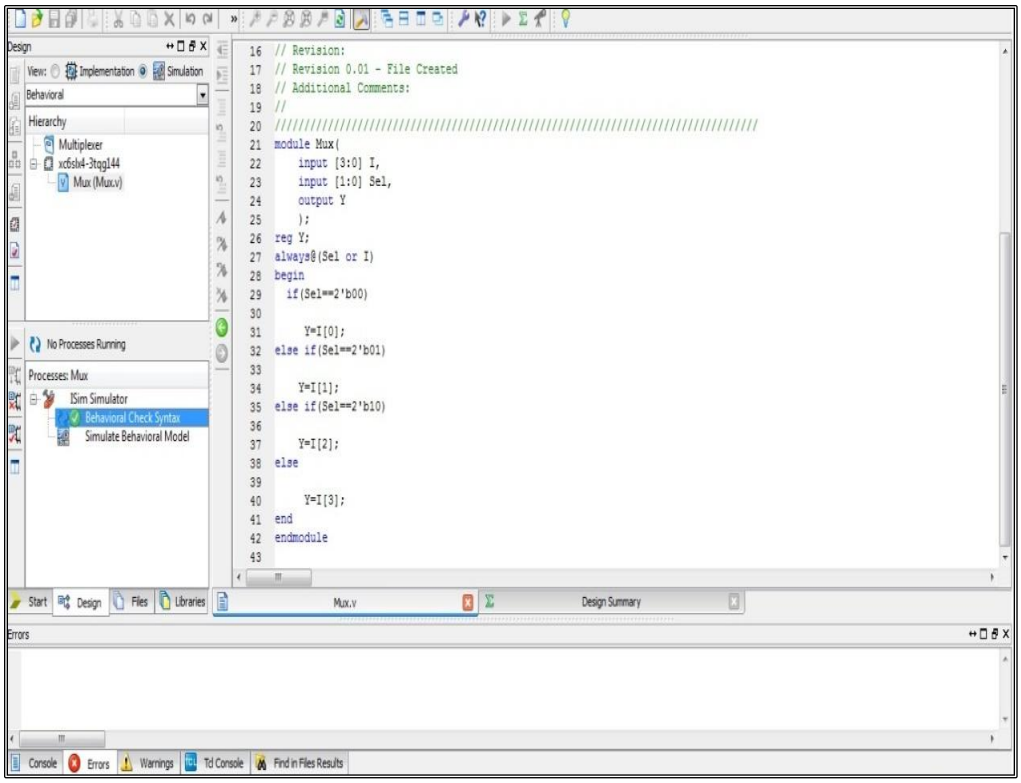


Fig 2.12: Behavioral Check Syntax

13. Now you have to write the test bench for checking the functionality of the design. Right click on "Mux.v" and select **New Source**. In the New Source Wizard window select **Verilog Test Fixture** (Fig 2.13). Write "Mux_test" in the **File name** field for the test bench. Click **Next**.

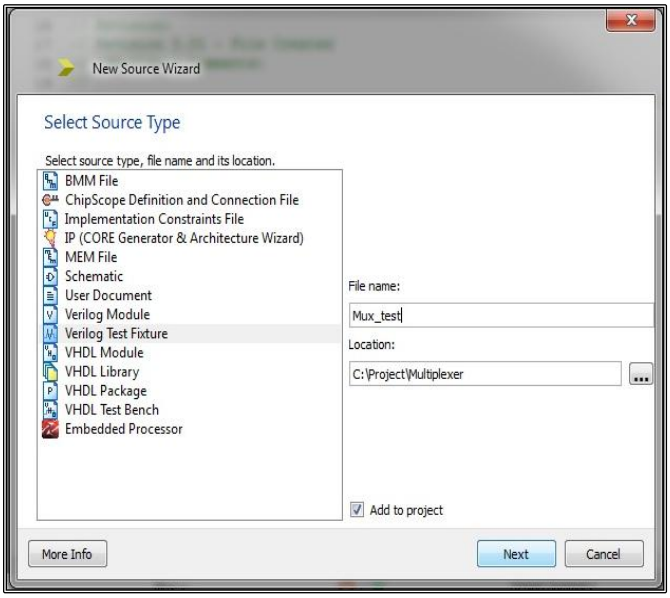


Fig 2.13: Testbench

14. In the next window, select the source file to which you want to associate the test bench (Fig 2.14). Click **next**, check the summary and click **Finish** (Fig 2.15).

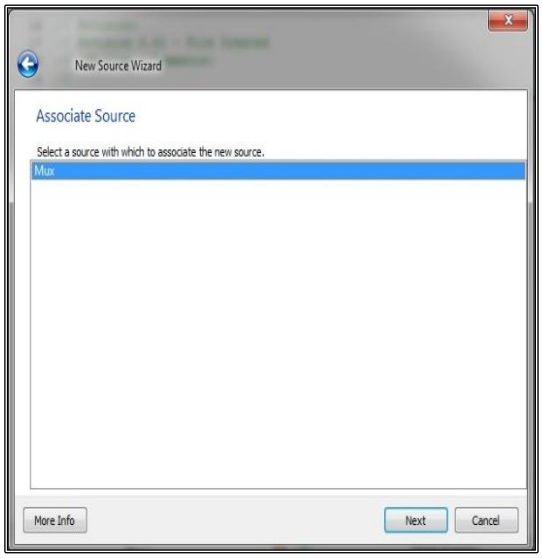


Fig 2.14: Associate Source

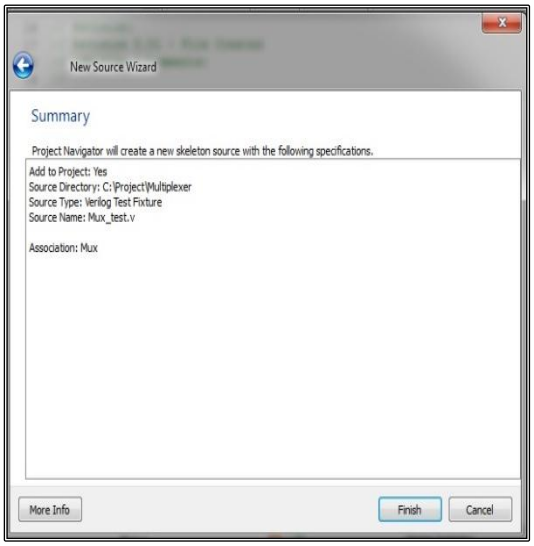


Fig 2.15: Testbench Summary

15. Template for the test bench is generated (Fig 2.16) with initial values instantiated.

```
24
25  module Mux_test;
26
27      // Inputs
28      reg [3:0] I;
29      reg [1:0] Sel;
30
31      // Outputs
32      wire Y;
33
34      // Instantiate the Unit Under Test (UUT)
35      Mux uut (
36          .I(I),
37          .Sel(Sel),
38          .Y(Y)
39      );
40
41      initial begin
42          // Initialize Inputs
43          I = 0;
44          Sel = 0;
45
46          // Wait 100 ns for global reset to finish
47          #100;
48
49          // Add stimulus here
50
51      end
```

Fig 2.16: Testbench for "Mux.v"

16. Initialize the input 'I' and give different combinations of select (Sel) lines. After entering the combinations save the code.

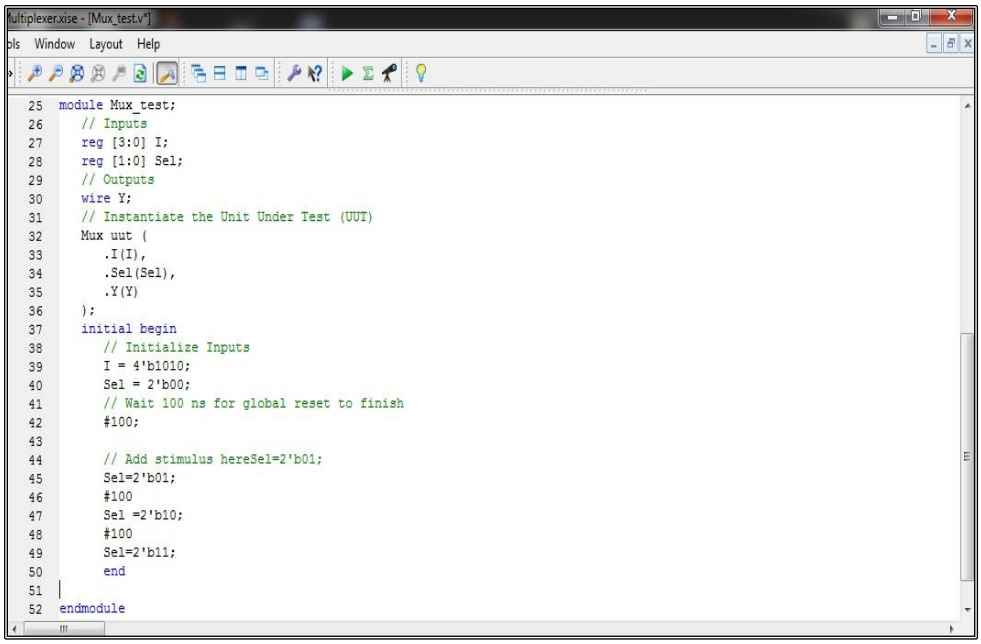


Fig 2.17: Complete Testbench

17. After editing the testbench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the testbench.If no errors are present double click on **Simulate Behavioral Model** (Fig 2.18).

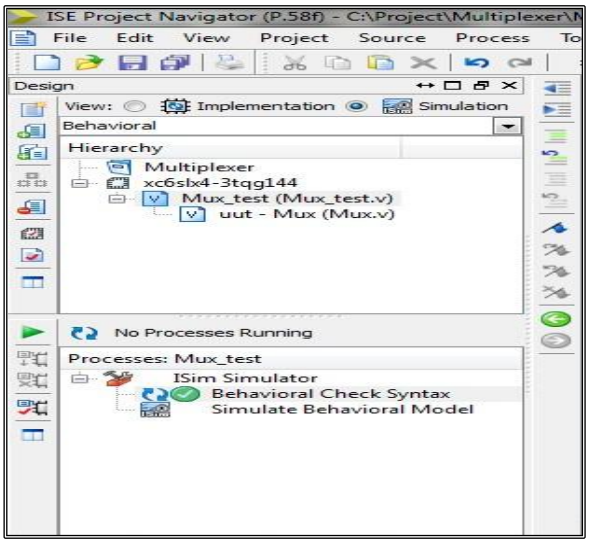


Fig 2.18: Behavioral Simulation

18. The simulated waveform for the "Mux" is shown in the Fig 2.19. The simulation is successful here as the output is verified for the multiplexer.

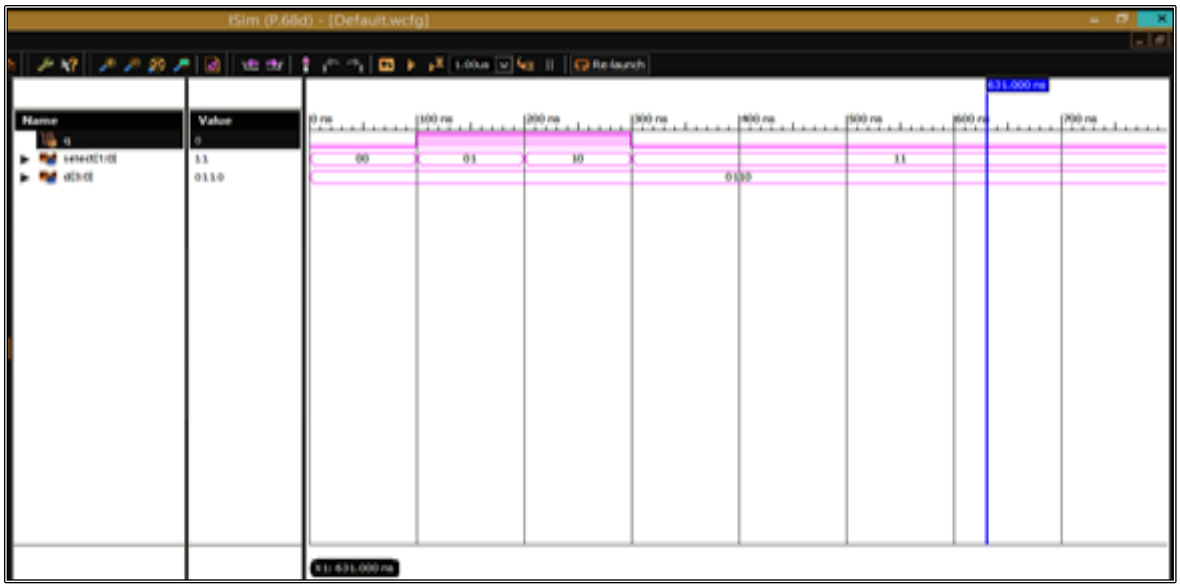


Fig 2.19: Simulation Waveform

19. For synthesis and implementation, select **Implementation Button** in the Hierarchy window on the left pan (Fig 2.20).

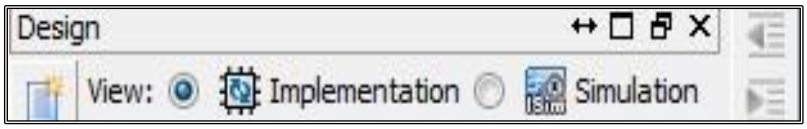


Fig 2.20: Implementation

20. For UCF File right click on the design (Mux.v) and select **New Source**. New Source Wizard window will open.

21. Select source type as **Implementation Constraints File** and enter the file name as "Mux_UCF" (Fig 2.21).

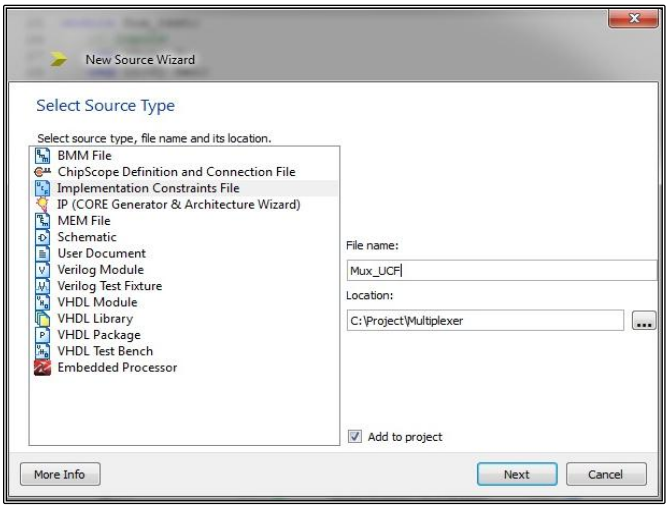


Fig 2.21: UCF creation

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

22. Check the summary. Click **Finish** (Fig 2.22).

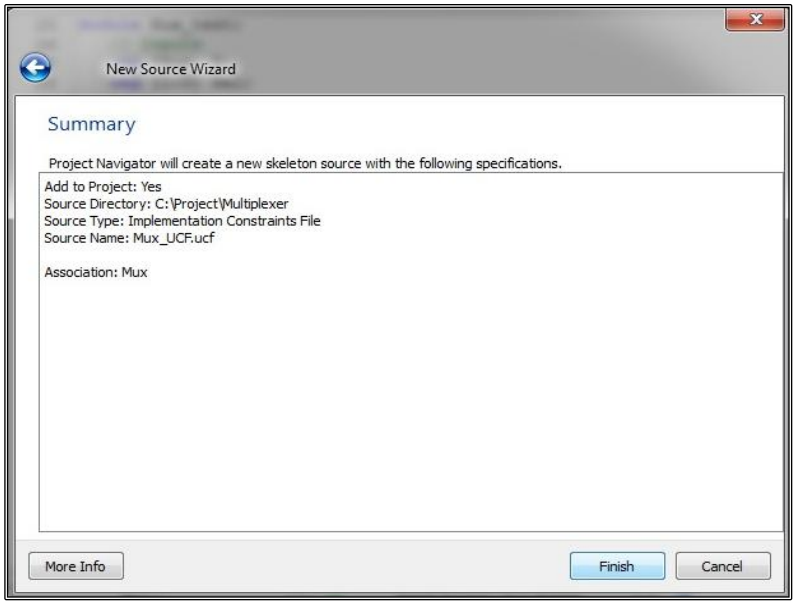


Fig 2.22: UCF summary

23. A blank window will appear in a new tab having the design's name as "Mux_UCF.ucf". Enter the constraints for each input and output port (Fig2.23).

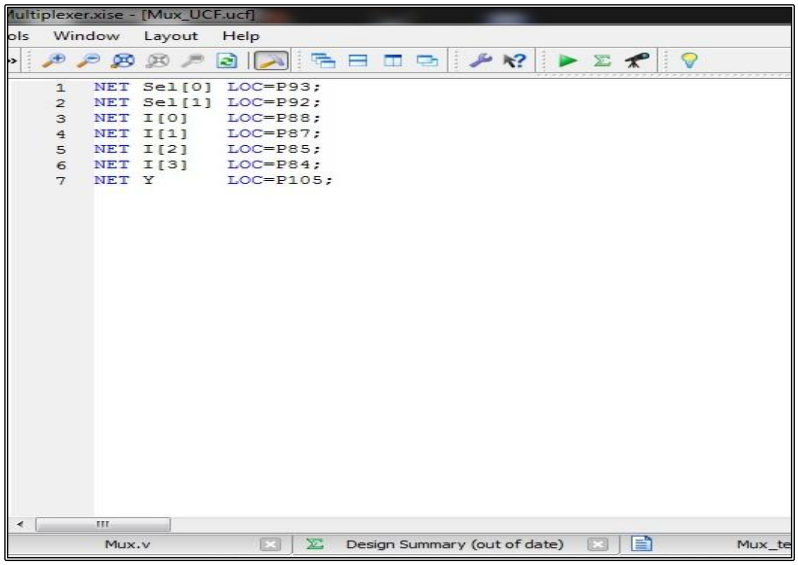


Fig 2.23: UCF Declaration

24. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST**(Fig 2.24) and select Run. If the design is successfully syntheszied then green check will appear near the Synthesize-XST (Fig 2.25).

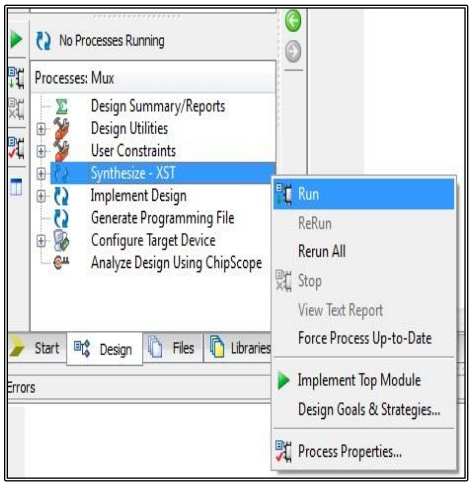


Fig 2.24: Synthesize

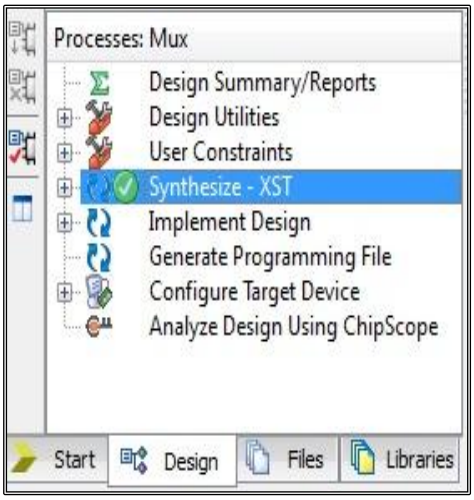


Fig 2.25: Synthesis Successful

25. Synthesize tool can generate two forms of schematic representation in the form of **RTL View** and **Technology View** of the multiplexer HDL code. For schematic and technology view double click on the RTL schematic and Technology Schematic.

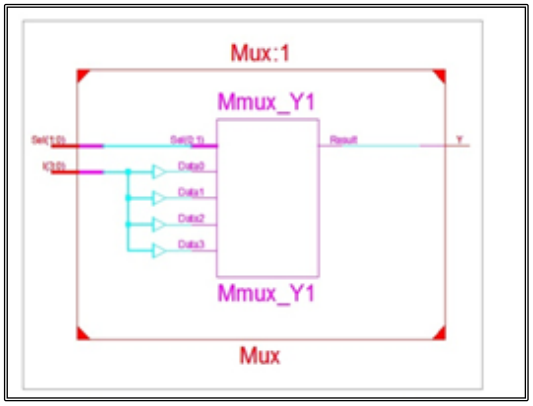


Fig 2.26: RTL Schematic

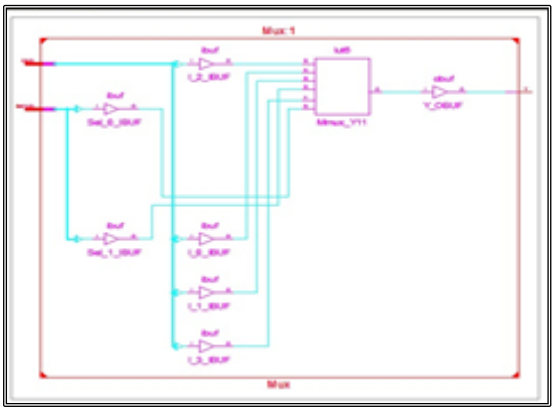


Fig 2.27: RTL schematic internal view

26. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route (Fig 2.28).

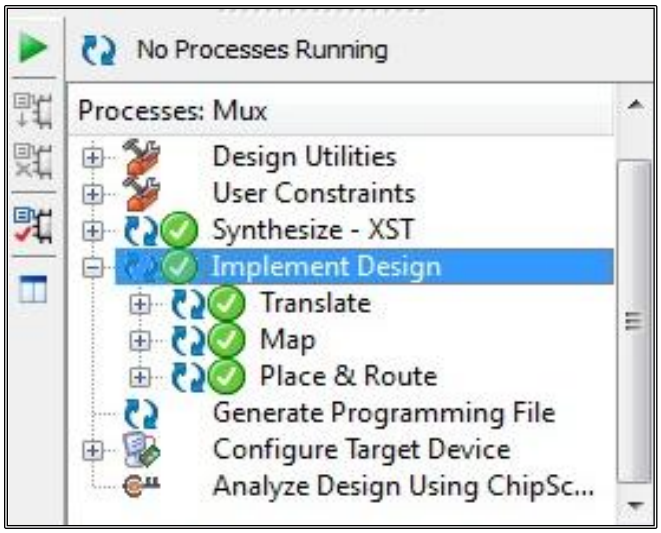


Fig 2.28: Implement Design

27. Then we have to generate programming file. For that right click on the **Generate Programming file** and click **Run**. This will generate the bit file which will 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.

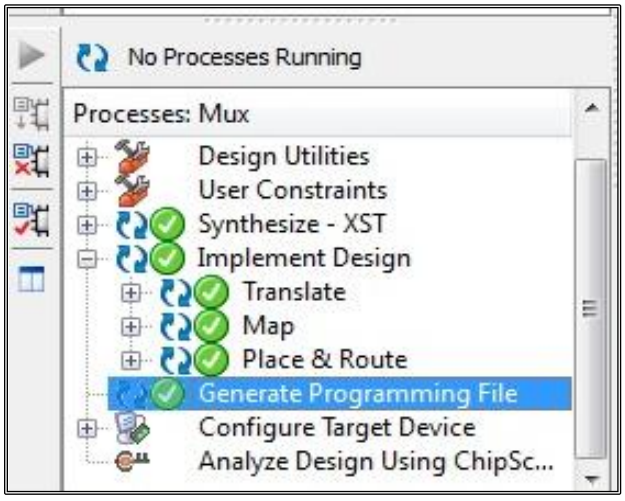


Fig 2.29: Bitstream generation

28. Now for programming the FPGA, connect the board to the computer through the USB cable and switch on the power supply of the board. A green LED near the power switch on the board will turn on when the board is powered on.

29. Start CDAC FPGA Board programmer (Fig2.30).

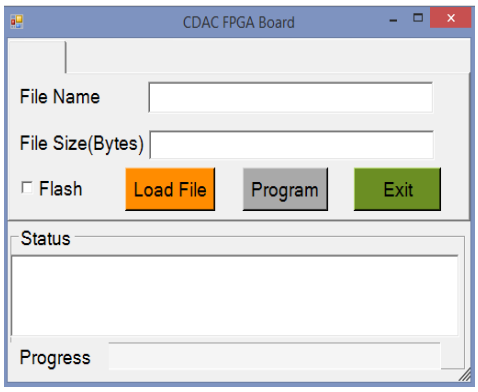


Fig 2.30: FPGA programmer interface

30. Click on the **load file** tab and go to the desired folder, select the "mux.bit" or "mux.bin" file.

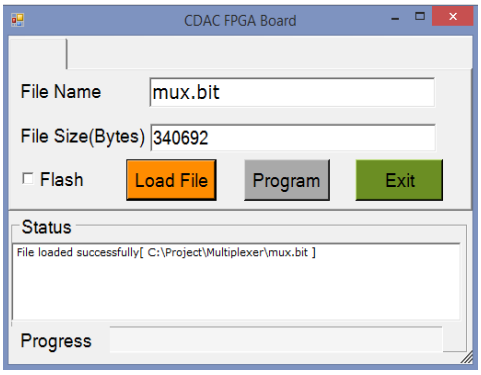


Fig 2.31: Loading the bit file

31. After loading the bitstream file, program the board. For this, click **Program**. This will download the bitstream file to the FPGA (Fig 2.32).

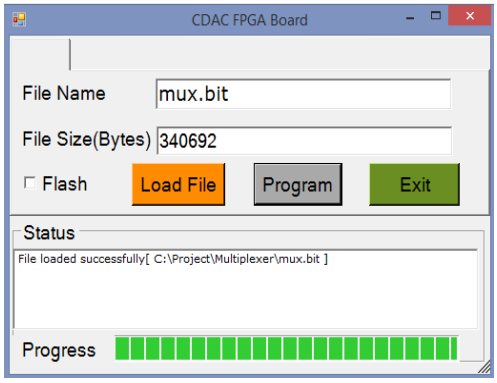


Fig 2.32: Program the FPGA

32. When the FPGA is programmed successfully then a message "**DONE goes High**" will be displayed in the **Status** console on the bottom window. Also a red LED on the board glows to indicate that the FPGA has been programmed successfully. Now the FPGA is configured as 4:1 Multiplexer.

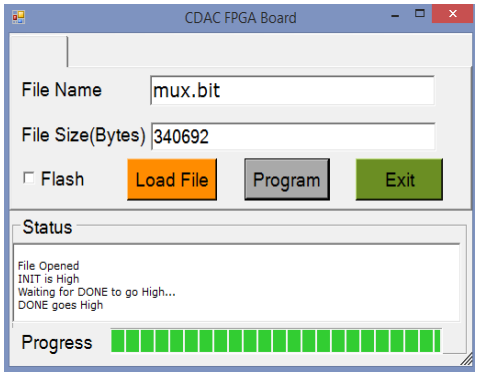


Fig 2.33: Programming successful

33. With the board having been programmed we can check the output on the board.

VERILOG CODE FOR THE 4:1 MULTIPLEXER

```
module Mux( input [3:0] I, input [1:0] Sel,output reg Y);
always@(Sel or I)
begin
if(Sel==2'b00)
Y=I[0];
else if(Sel==2'b01)
Y=I[1];
else if(Sel==2'b10)
Y=I[2];
 else
Y=I[3];
end
endmodule
```

User Constraints File (UCF) For 4:1 MULTIPLEXER

```
NET "Sel[0]" LOC="P93";
NET "Sel[1]" LOC="P92";
NET "I[0]"  LOC="P88";
NET "I[1]"  LOC="P87";
NET "I[2]"  LOC="P85";
NET "I[3]"  LOC="P84";
NET "Y"     LOC="P105";
```

EXPERIMENT: 3

Objective: To design and implement full adder circuit on the FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: - FPGA Board

Theory: In electronics, an adder is a digital circuit that performs addition of numbers. They can be constructed for most of the numerical representations like Binary, Gray code, Excess – 3, Binary Coded Decimal (BCD) etc. Binary addition is the most frequently performed task by most common adders.

A one-bit half adder adds two 1-bit numbers and generates 2-bit output, referred to as carry and sum. But it does not have the scope to add the carry bit from the previous position. This becomes the limitation of half adder, especially in the real time applications where we need to work on multiple bits.

A 1-bit full adder adds three 1-bit numbers, where two can be referred to as operands and one can be referred to as bit carried in. Therefore, full adder overcomes the limitation of half adder by taking in account the carry from the previous stage.

The truth table for the 1-bit full adder is shown in Table 3.1. A, B, and Cin are the three inputs where, A and B are the operands, and Cin is a bit carried in from the previous less significant stage.

Table 3.1: Truth table Full Adder

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.
2. From the toolbar, select **File** => **New Project**. In the **New Project Wizard** window type "Fulladder" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next** (Fig 3.1).

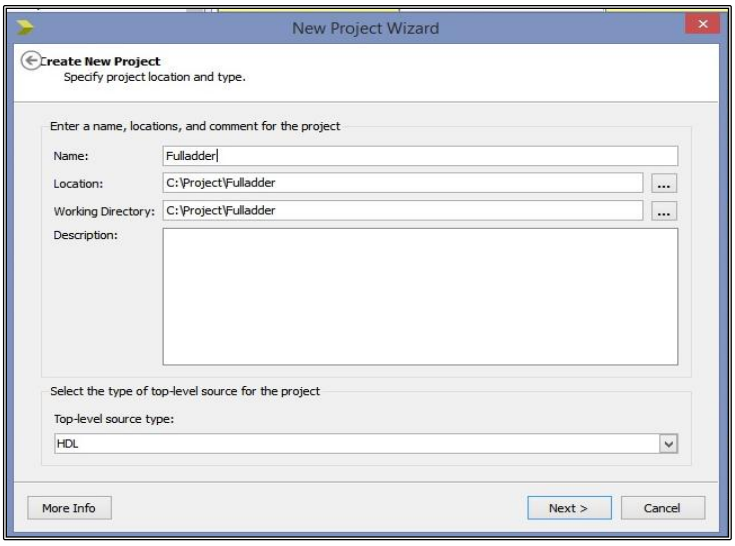


Fig 3.1: New Project

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties (Fig 3.2):
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

   Other properties can be left at their default values.

   Click **Next**.

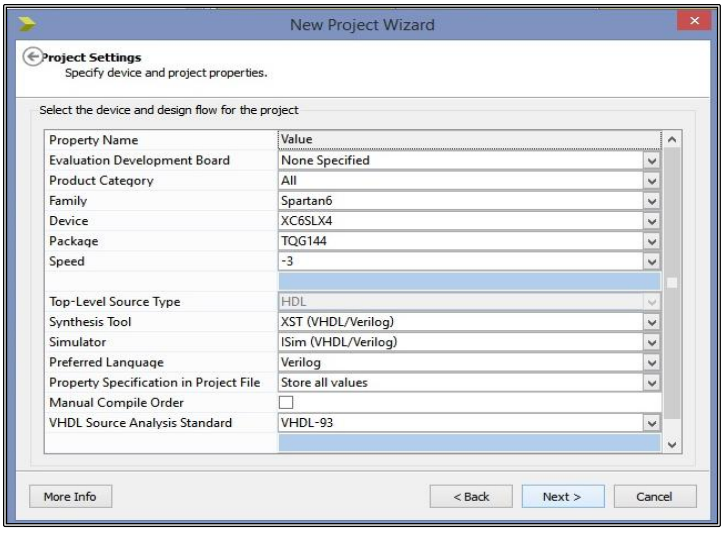


Fig 3.2: Project Properties

4. The **New Project Wizard—Project Summary** page appears as shown in Fig 3.3.

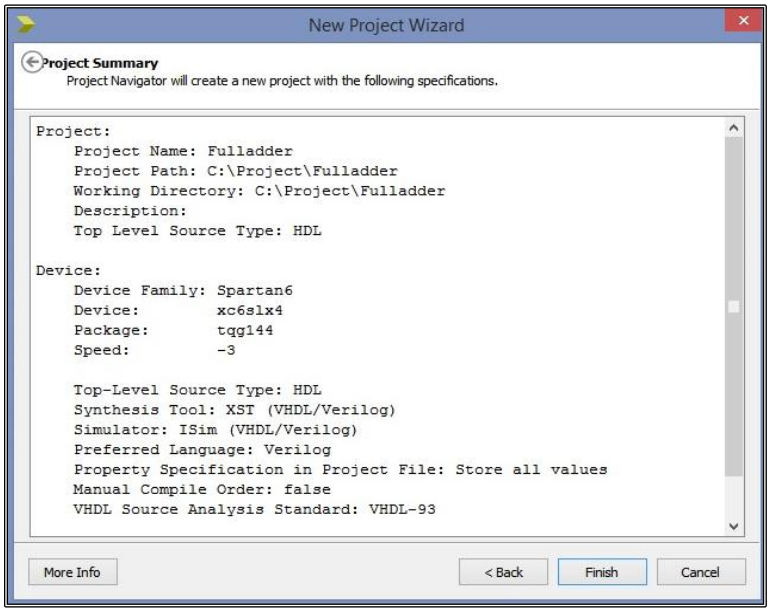


Fig 3.3: Project Summary

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

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "Fulladder" for the new source in the **File Name** field (Fig 3.4). Click **Next**.

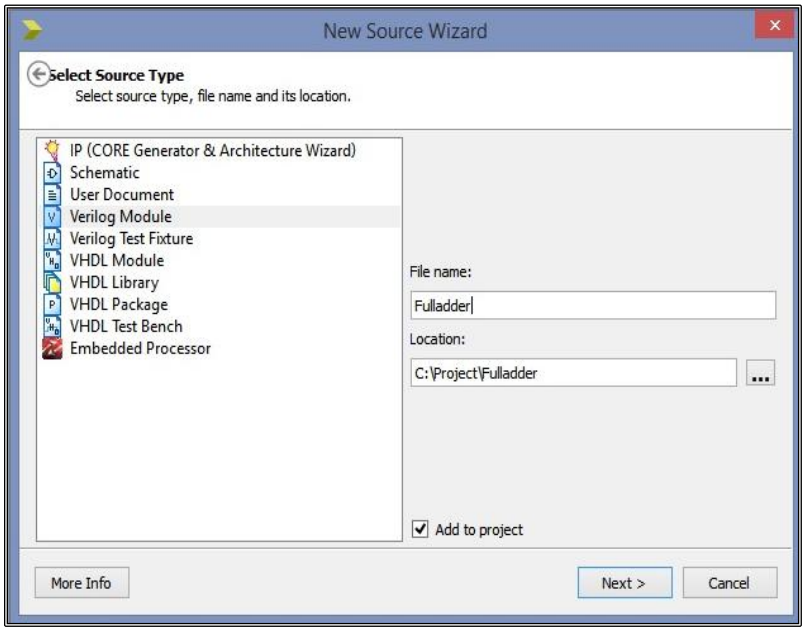


Fig 3.4: New Source

6. In the **Define Module** page, enter the port information for the "Fulladder" as follows (Fig 3.5):

   a. In the first five **Port Name** fields, enter 'a', 'b', 'c', 'Sum' and 'Carry'.

b. Set the **Direction field** to 'output' for 'Sum', 'Carry' and to 'input' for 'a', 'b' and 'c'.
c. Leave the **Bus** designation boxes unchecked.

Fig 3.5: Module Definition

7. Next the summary description of the module will be displayed (Fig 3.6). Check the source **summary** and click **Finish**.

Fig 3.6: Source Summary

8. In the Xilinx ISE interface you can see that the new source file "Fulladder.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the Fulladder HDL file, and some of the basic file structure is already in place (Fig 3.7).

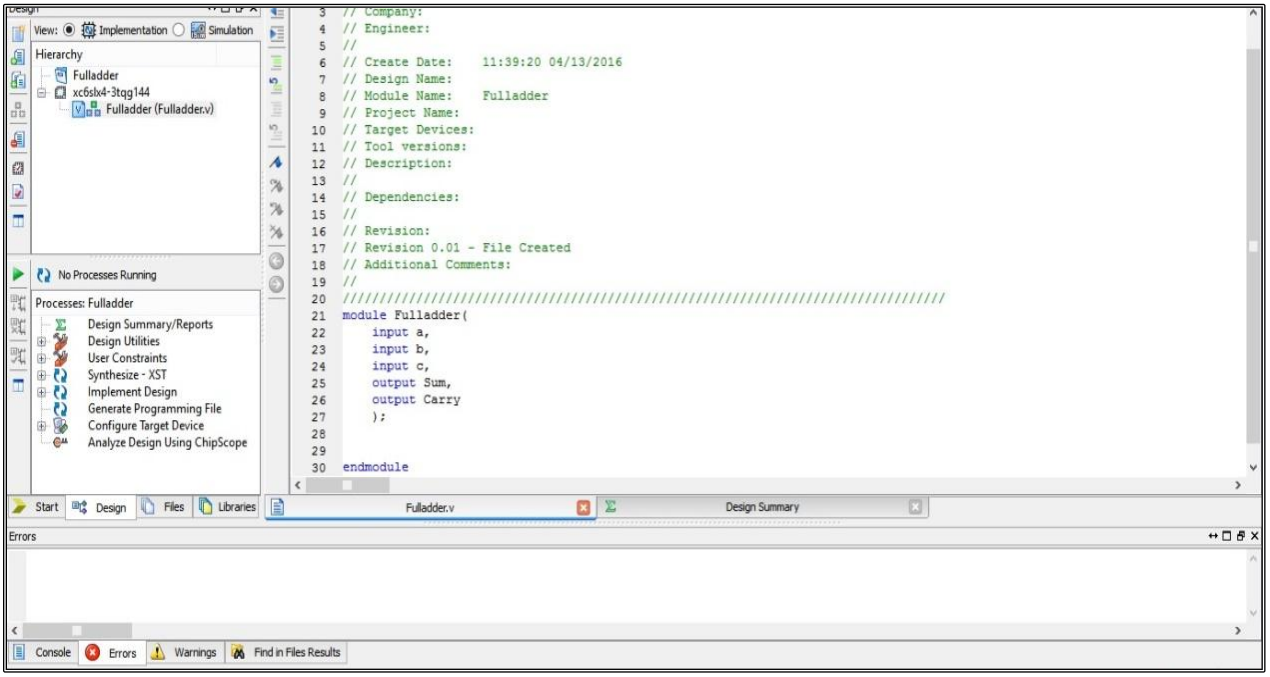


Fig 3.7: Fulladder module structure

9. Next, you have to write the desired full adder logic. For this example dataflow modeling is used (Fig 3.8).Save the file.

```
 5  //
 6  // Create Date:     11:39:20 04/13/2016
 7  // Design Name:
 8  // Module Name:     Fulladder
 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 Fulladder(
22       input a,
23       input b,
24       input c,
25       output Sum,
26       output Carry
27       );
28
29   assign Sum=a^b^c;
30   assign Carry=ab|bc|ca ;
31  endmodule
32
```

Fig 3.8: Complete Fulladder Module

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window on the left pane (Fig 3.9).

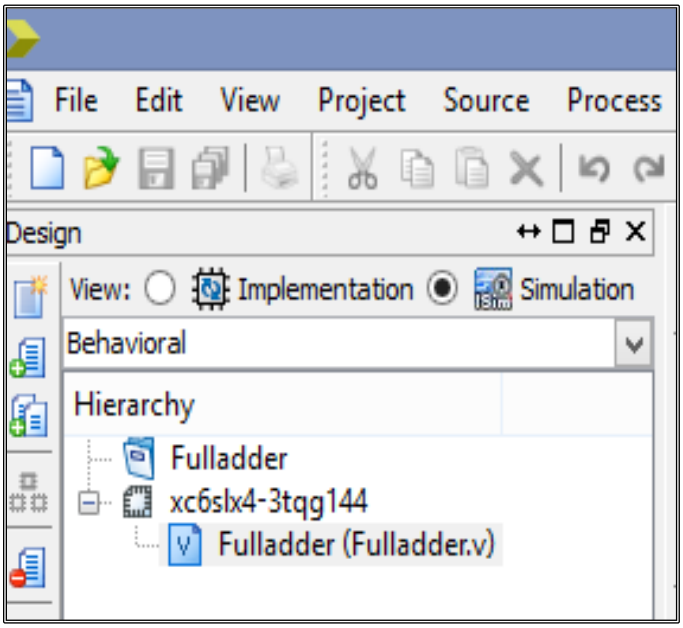


Fig 3.9: Simulation

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

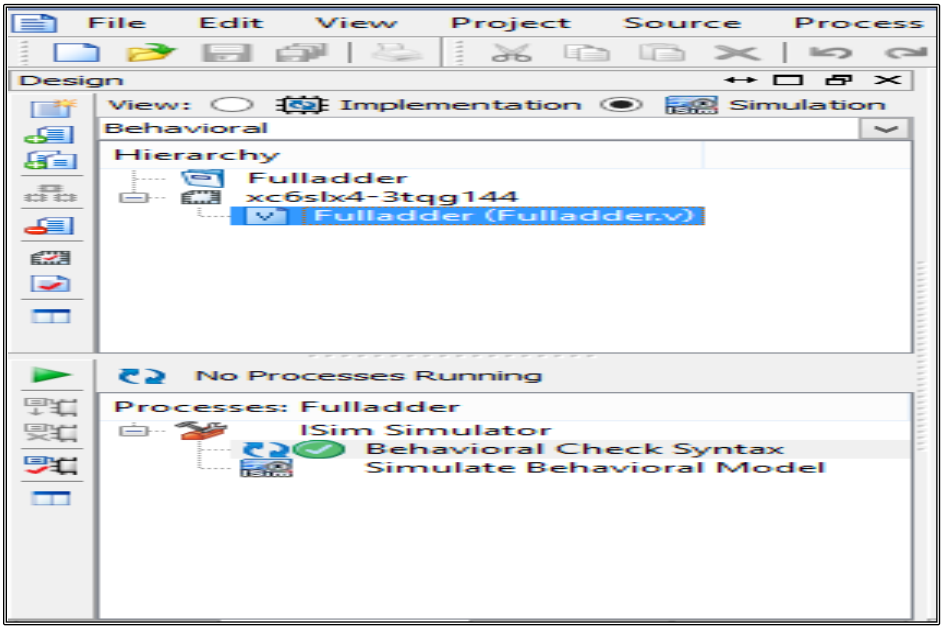


Fig 3.10: Behavioral Check Syntax

12. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (Fulladder.v) and select **New Source**. In the New Source

Wizard window, select **Verilog Test Fixture** (Fig 3.11). Write the "Fulladder_test" in the **File name** field for the test bench. Click **Next**.

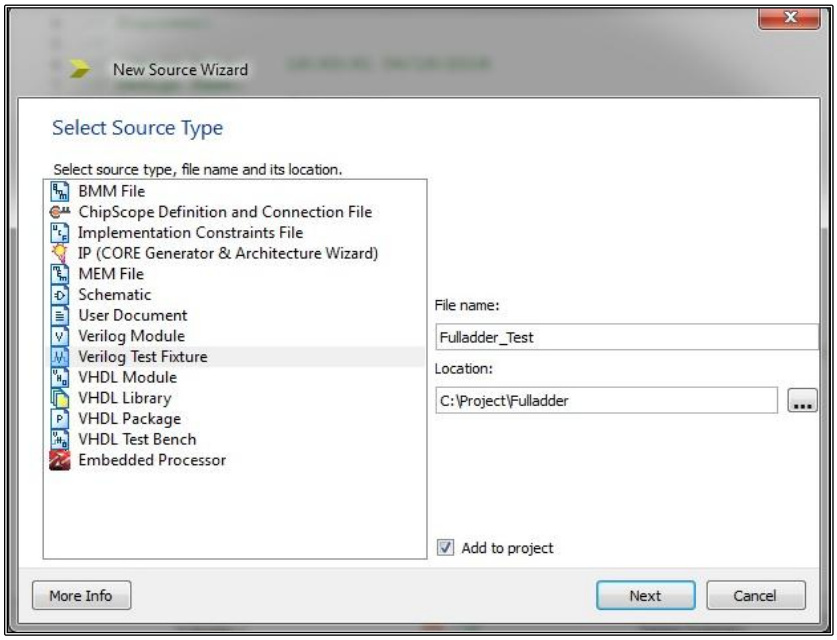


Fig 3.11: Testbench for Fulladder

13. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

14. The ISE project navigator will generate template for testbench as shown in Fig3.12.

```
24
25  module Fulladder_Test;
26
27      // Inputs
28      reg a;
29      reg b;
30      reg c;
31
32      // Outputs
33      wire Sum;
34      wire Carry;
35
36      // Instantiate the Unit Under Test (UUT)
37      Fulladder uut (
38          .a(a),
39          .b(b),
40          .c(c),
41          .Sum(Sum),
42          .Carry(Carry)
43      );
44
45      initial begin
46          // Initialize Inputs
47          a = 0;
48          b = 0;
49          c = 0;
50
```

Fig 3.12: Testbench template

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

15. Give different values for ‘a’, ‘b’ and ‘c’ after 100 units of time delay (Fig 3.13). After specifying all possible combinations save the code. Click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench. If no errors are present double click on **Simulate Behavioral Model**.

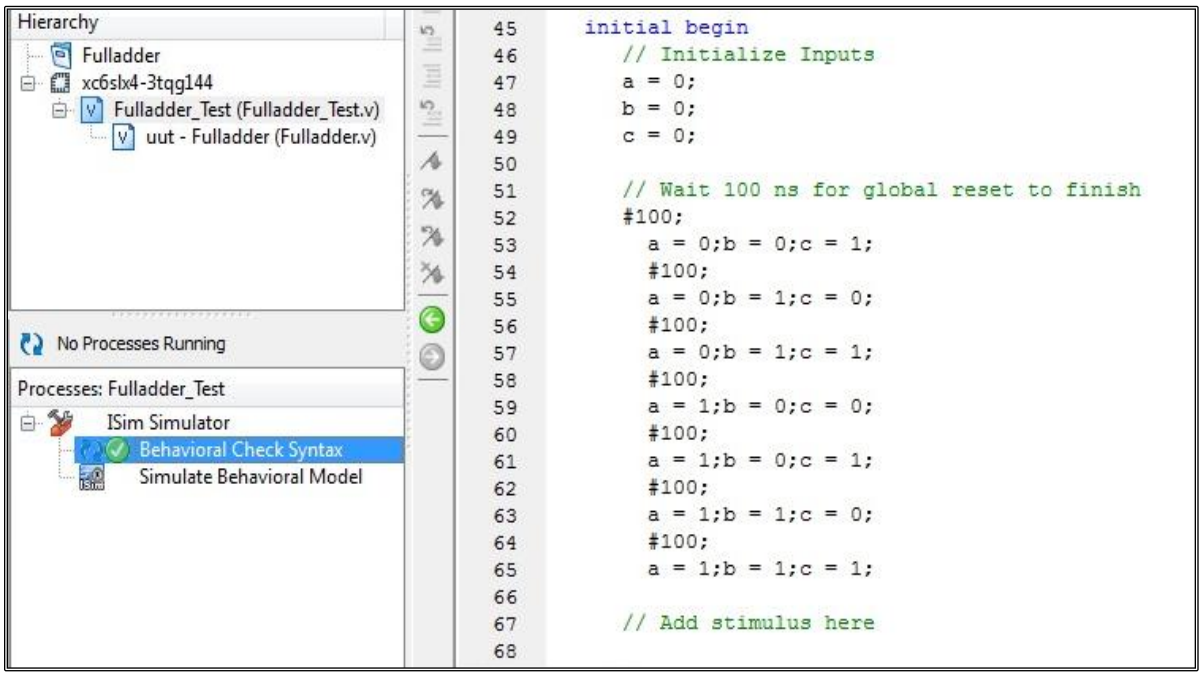


Fig 3.13: Fulladder testbench

16. A new window showing the waveforms for the inputs and outputs of the design will be displayed.The simulated waveform for the full adder is shown in Fig 3.14.

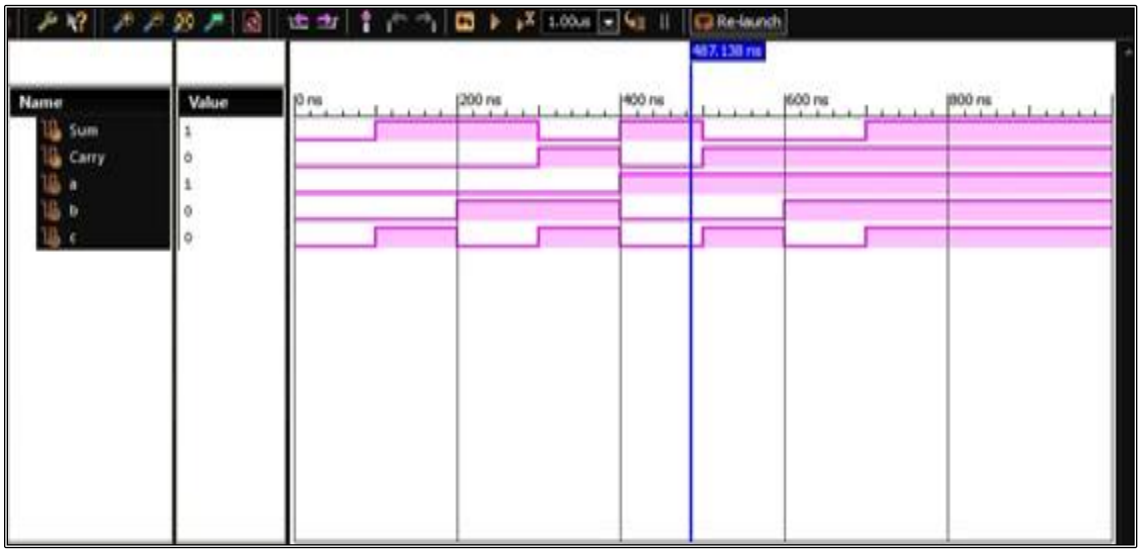


Fig 3.14: Simulated waveforms

17. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan (Fig 3.19).

Fig 3.15: Design Implementation

18. For UCF File, right click on the design (Fulladder.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "FA" (Fig 3.16).

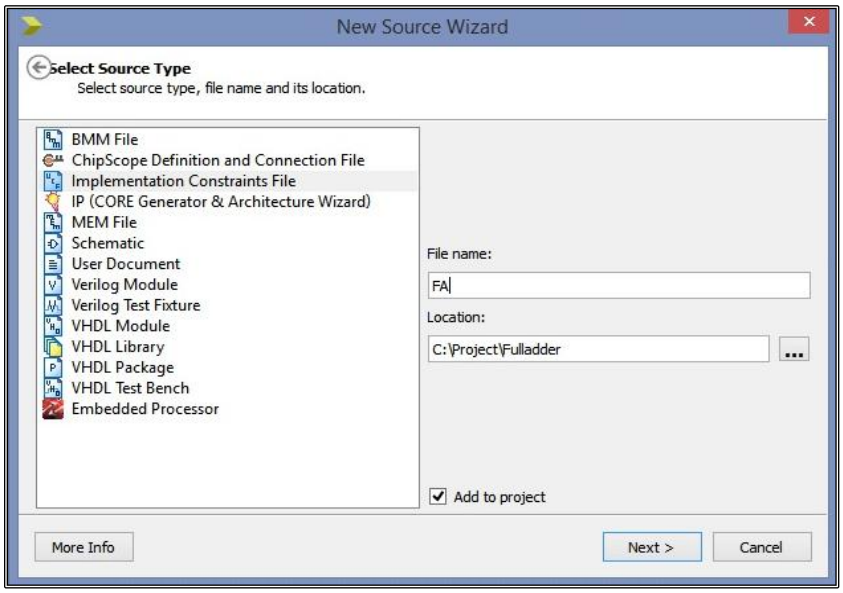


Fig 3.16: UCF creation

Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

19. A blank window will appear in a new tab having the design's name as "FA.ucf" . Add the constraints for each input port and output port (Fig 3.22).

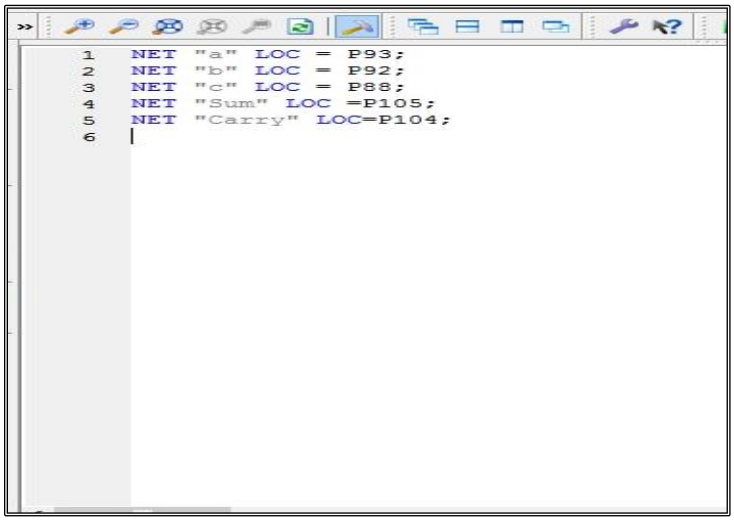


Fig 3.17: UCF declaration

20. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST as shown in Fig 3.18 and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

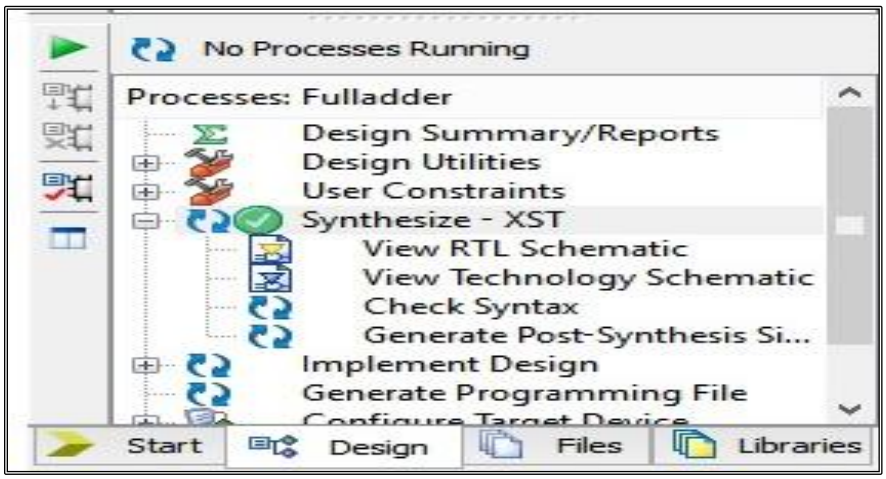


Fig 3.18: Synthesis

21. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** (Fig 3.19)and **Technology View** (Fig 3.20) for the "Fulladder.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

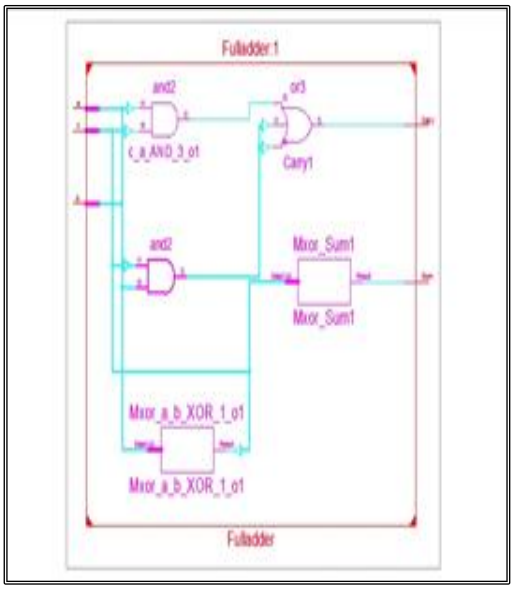


Fig 3.19: RTL View

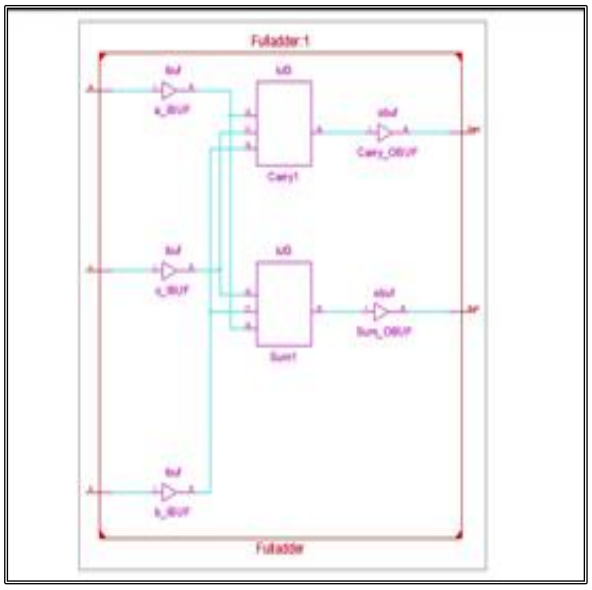


Fig 3.20: Technology View

22. Next step is the implementation of the design.Right click **Implement Design** and click **Run**(Fig 3.21).After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route(Fig 3.22).

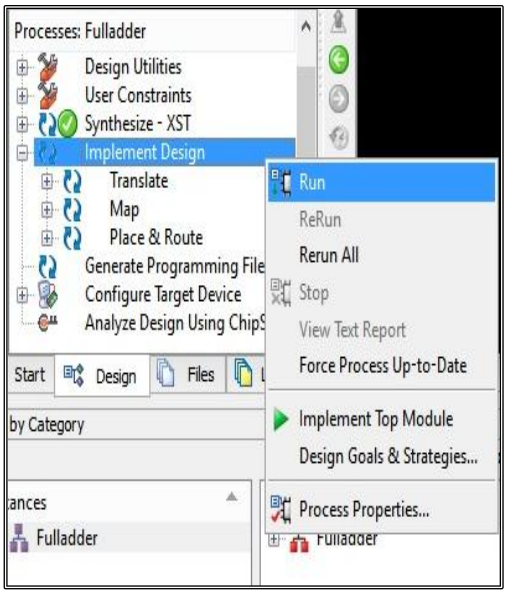


Fig 3.21: Design Implementation

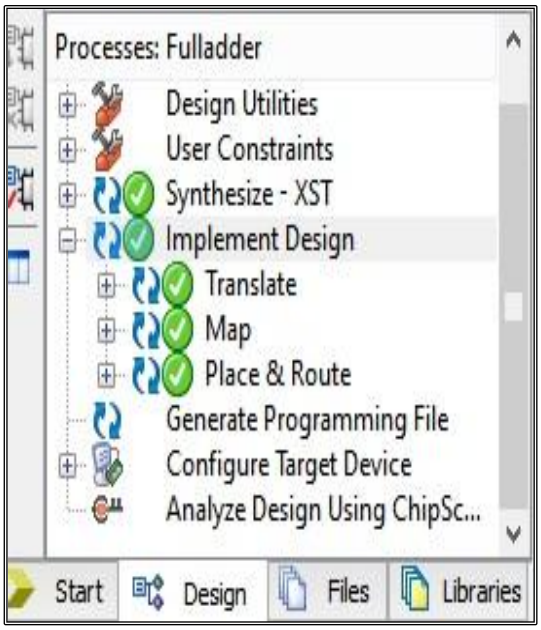


Fig 3.22: Successful implementation

23. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip (Fig 3.23).

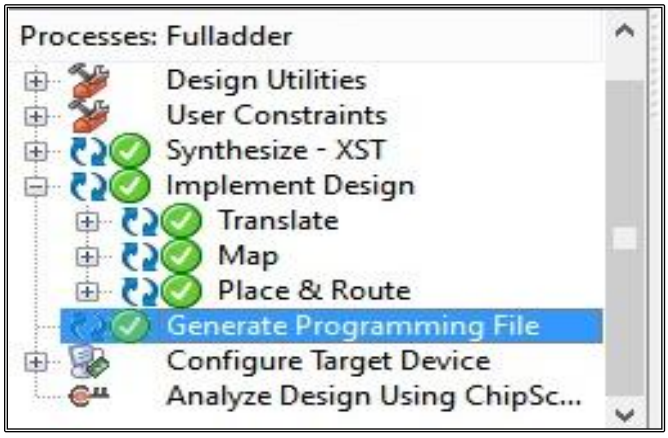


Fig 3.23: Bitstream generation

24. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the full adder circuit.

VERILOG CODE FOR THE FULL ADDER

```
module Fulladder(
   input a,
   input b,
   input c,
   output Sum,
   output Carry
   );
assign Sum= a^b^c;
assign Carry=(a&b)|(b&c)|(c&a) ;

endmodule
```

User Constraints File (UCF) For FULL ADDER

```
NET "a" LOC="P93";
NET "b" LOC="P92";
NET "c" LOC="P88";
NET "Sum" LOC="P105";
NET "Carry" LOC="P104";
```

EXPERIMENT: 4

Objective: To design and implement full subtractor circuit on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: A logic circuit which is used for subtracting three 1- bit binary data is known as full subtractor. The truth table of full subtractor is shown in Table 4.1.

Table 4.1: Truth table Full Subtractor

INPUT			OUTPUT	
a	**b**	**c**	**Difference**	**Borrow**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.

2. From the toolbar, select **File** => **New Project**. In the **New Project Wizard** window type "FullSubtractor" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next** (Fig 4.1).

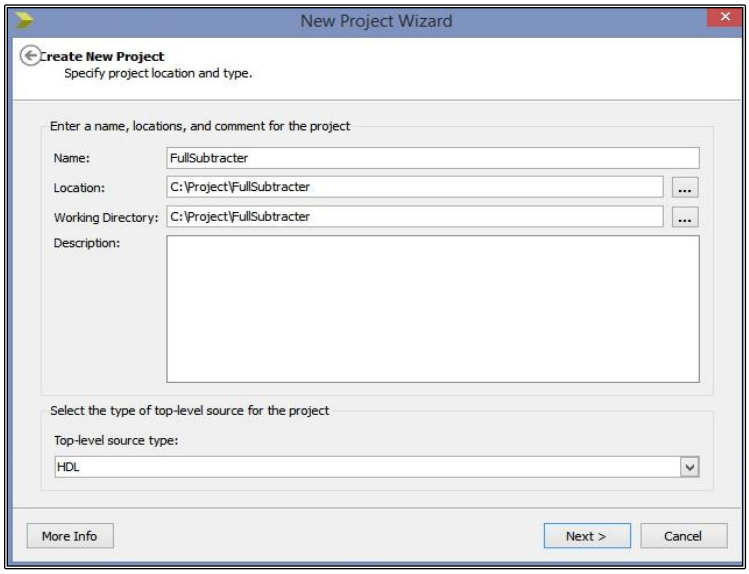


Fig 4.1: New Project

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The **New Project Wizard—Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "FullSubtractor" for the new source in the **File Name** field (Fig 4.2). Click **Next**.

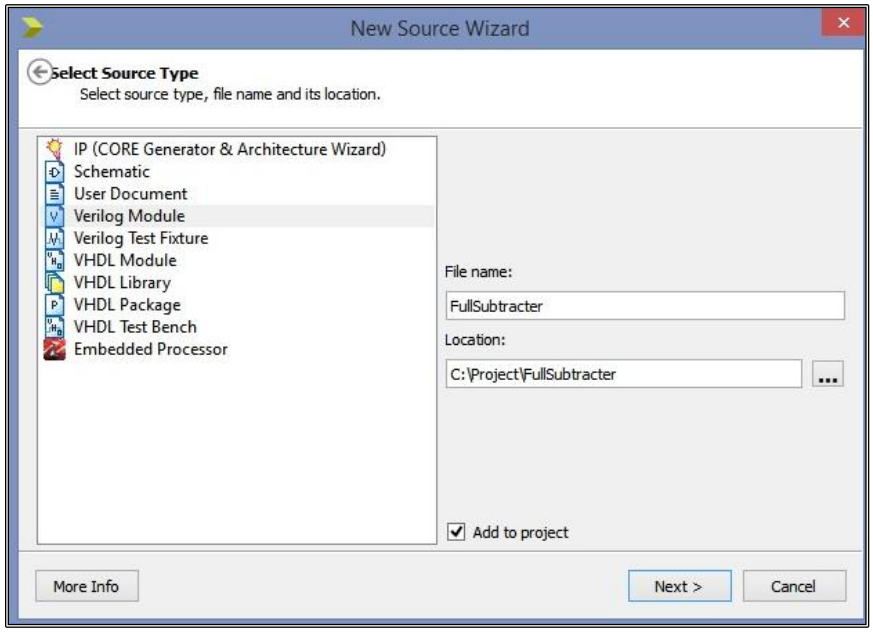


Fig 4.2: New Source

6. In the **Define Module** page, enter the port information for the "FullSubtractor" as follows (Fig 4.3):

   a. In the first five **Port Name** fields, enter 'a', 'b', 'c', 'Difference' and 'Borrow'.
   b. Set the **Direction field** to 'output' for 'Difference', 'Borrow' and to 'input' for 'a', 'b' and 'c'.
   c. Leave the **Bus** designation boxes unchecked.
   Click **Next.**

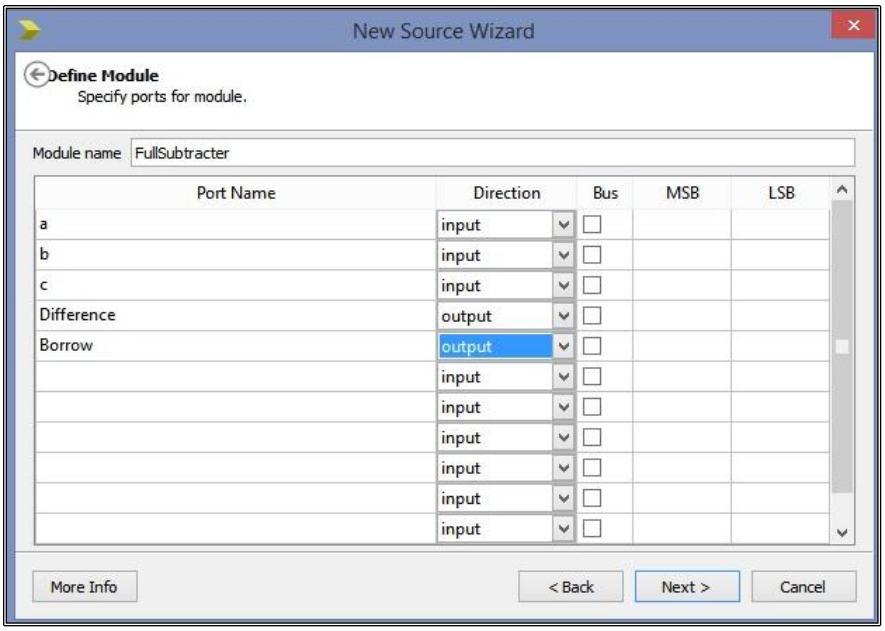


Fig 4.3: Module Definition

7. Next the summary description of the module will be displayed (Fig 4.4). Check the source **summary** and click **Finish**.

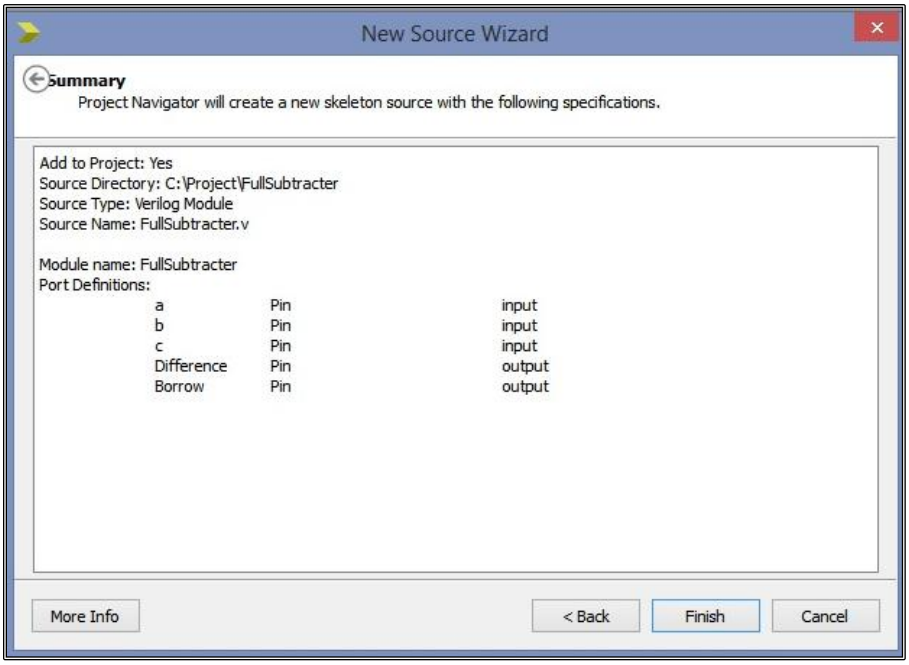


Fig 4.4: Source Summary

8. In the Xilinx ISE interface you can see that the new source file "FullSubtractor.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the FullSubtractor HDL file, and some of the basic file structure is already in place (Fig 4.5).

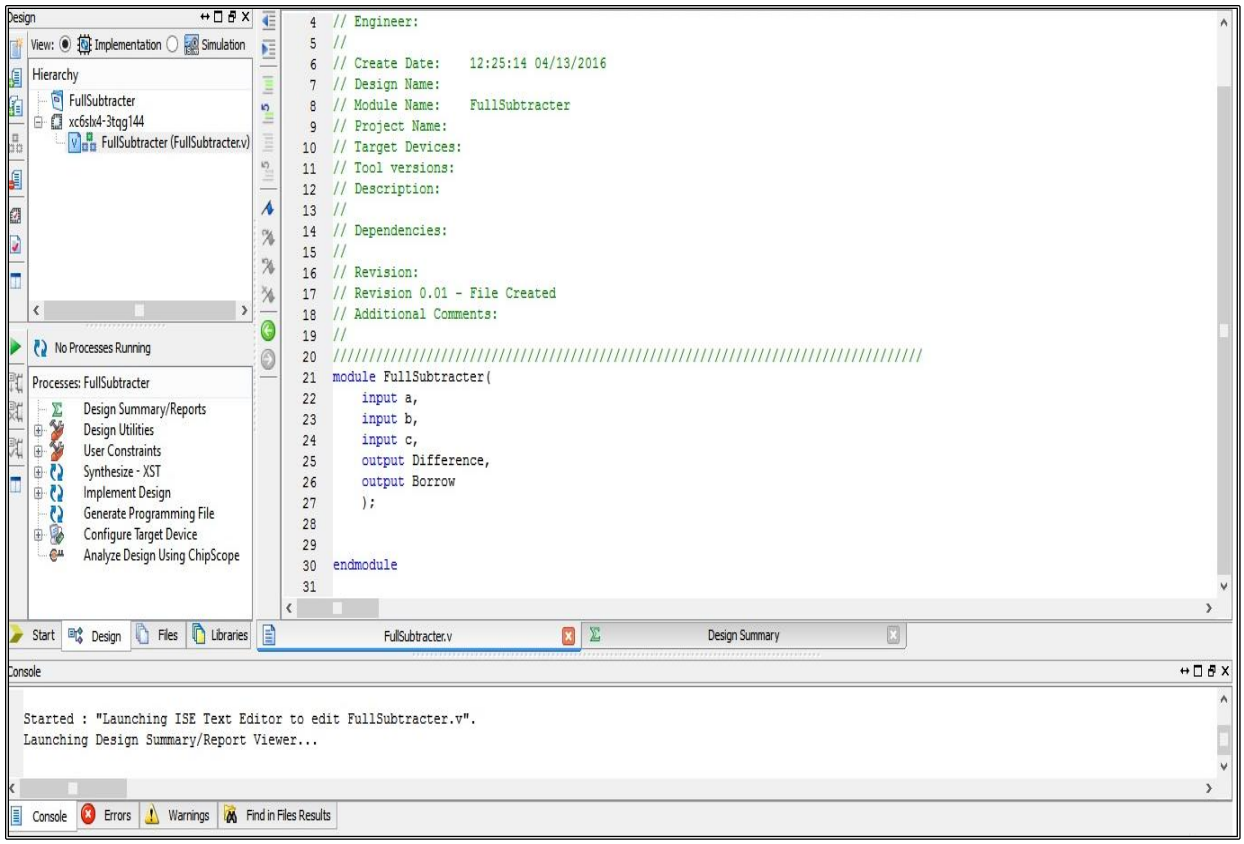


Fig 4.5: Fullsubtractor Module structure

9. Next, you have to write the desired full subtractor logic. For this example dataflow modeling is used (Fig 4.7).**Save** the file.

```
20  //////////////////////////////////////////////////////////////////////////////////
21  module FullSubtracter(
22      input a,
23      input b,
24      input c,
25      output Difference,
26      output Borrow
27      );
28
29  assign Difference =a^b^c;
30  assign Borrow=((~a)&b)|(b&c)|(c&(~a));
31  endmodule
32
```

Fig 4.6: Complete Full Subtractor Module

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window on the left pane (Fig 4.8).

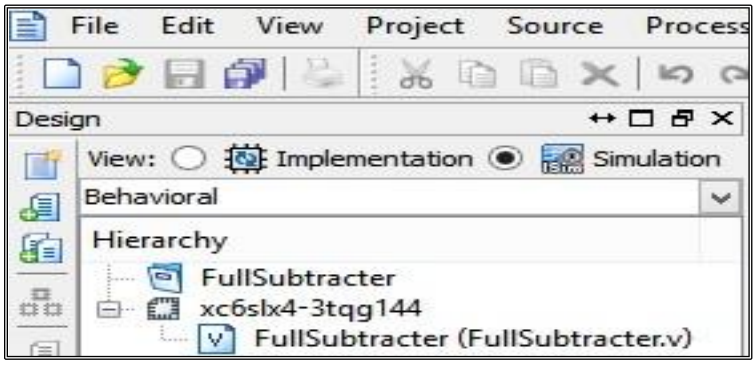


Fig 4.7: Simulation

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

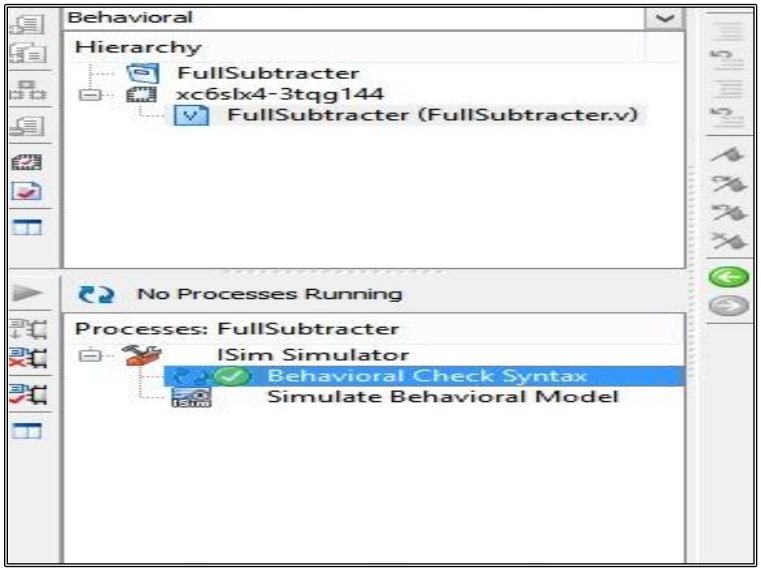


Fig 4.8: Behavioral Check Syntax

12. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (FullSubtractor.v) and select **New Source**. In the New Source Wizard window, select **Verilog Test Fixture** .Write the "Fullsubtractor_test" in the **File name** field for the test bench (Fig 4.9). Click **Next**.

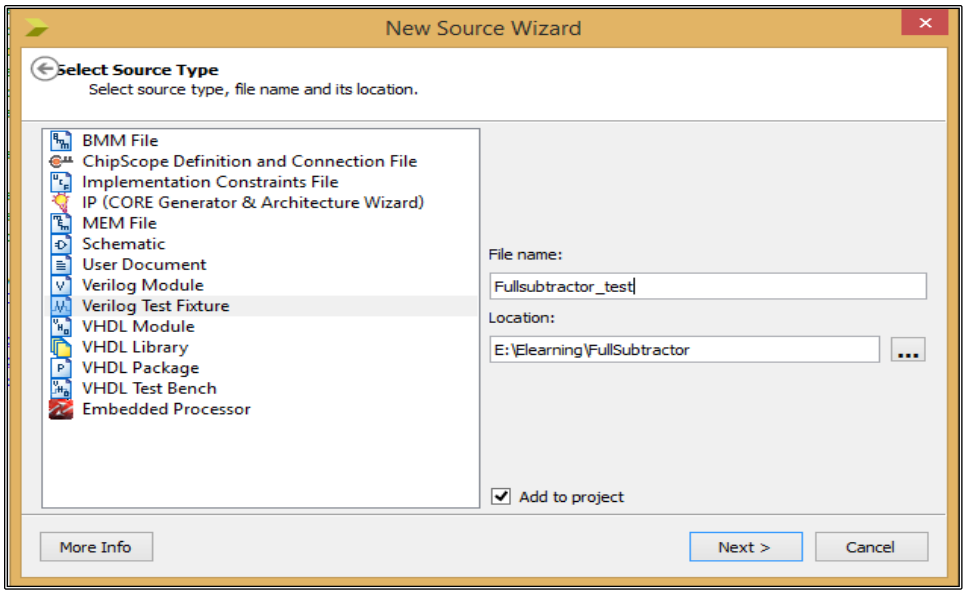


Fig 4.9: FullSubtractor testbench

13. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

14. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values (Fig 4.10). After editing the test bench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench. If no errors are present double click on Simulate Behavioral Model. A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs (Fig 4.11).

```
25  module Fullsubtractor_test;
26
27      // Inputs
28      reg a;
29      reg b;
30      reg c;
31
32      // Outputs
33      wire Difference;
34      wire Borrow;
35
36      // Instantiate the Unit Under Test (UUT)
37      FullSubtracter uut (
38          .a(a),
39          .b(b),
40          .c(c),
41          .Difference(Difference),
42          .Borrow(Borrow)
43      );
44
45      initial begin
46          // Initialize Inputs
47          a = 0;
48          b = 0;
49          c = 0;
50          #100;
51          a = 0;
52          b = 0;
53          c = 1;
54          #100;
55          a = 1;
56          b = 0;
57          c = 1;
58          #100;
59          a = 1;
60          b = 1;
61          c = 0;
62          #100;
63
```

Fig 4.10: Complete testbench

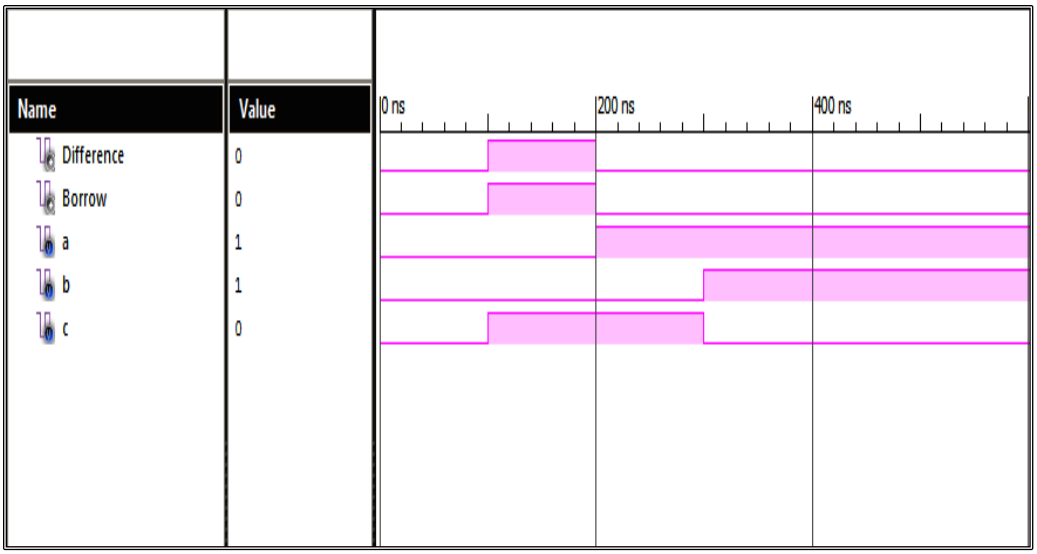


Fig 4.11: Simulated waveform

15. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan (Fig 4.12).

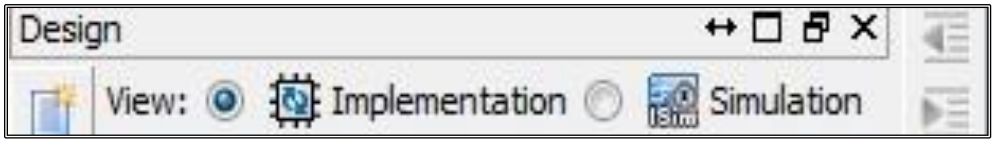


Fig 4.12: Design Implementation

16. For UCF File, right click on the design (Fullsubtractor.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "FS" (Fig 4.13).

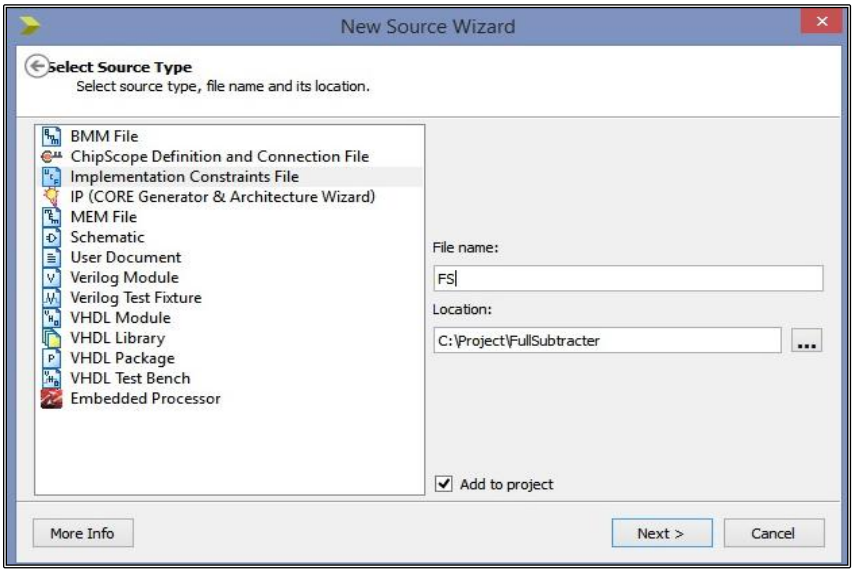


Fig 4.13: New Source UCF

Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

17. A blank window will appear in a new tab having the design's name as "FS.ucf" . Add the constraints for each input port and output port (Fig 4.14).

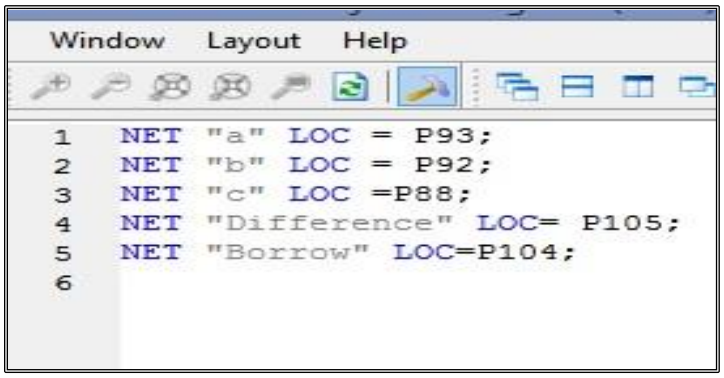


Fig 4.14: UCF declaration

18. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST as shown in Fig 4.15 and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

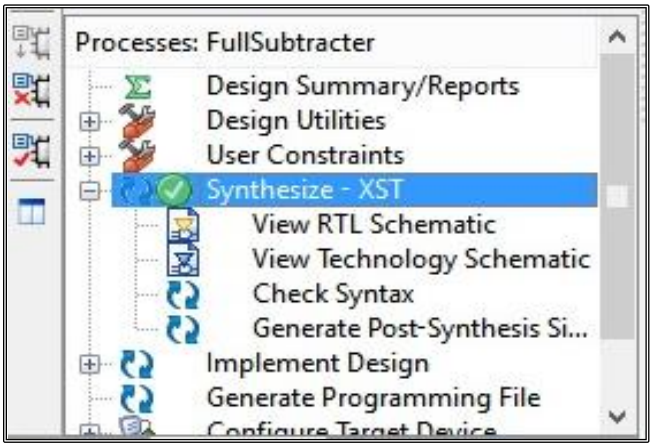


Fig 4.15: Synthesis

19. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** (Fig 4.16)and **Technology View** (Fig 4.17) for the "Fulladder.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

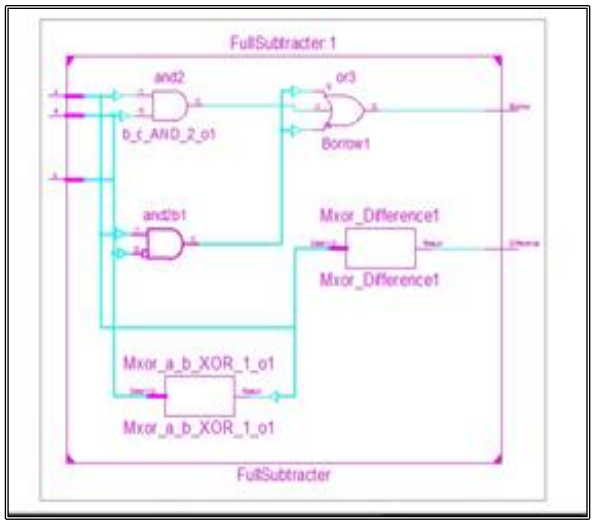


Fig 4.16: RTL View

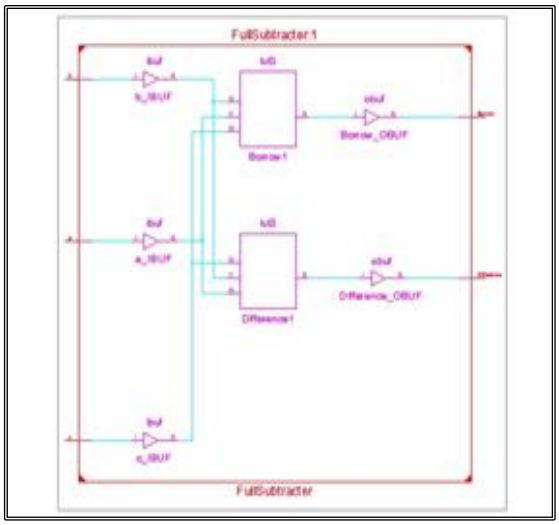


Fig 4.17: Technology View

20. Next step is the implementation of the design.Right click **Implement Design** and click **Run**(Fig 4.18).After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route(Fig 4.19).

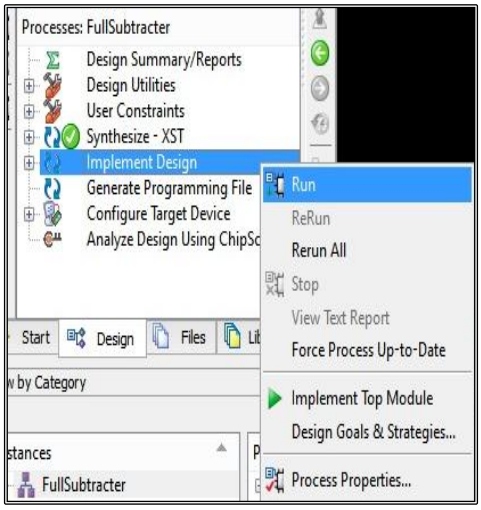


Fig 4.18: Design Implementation

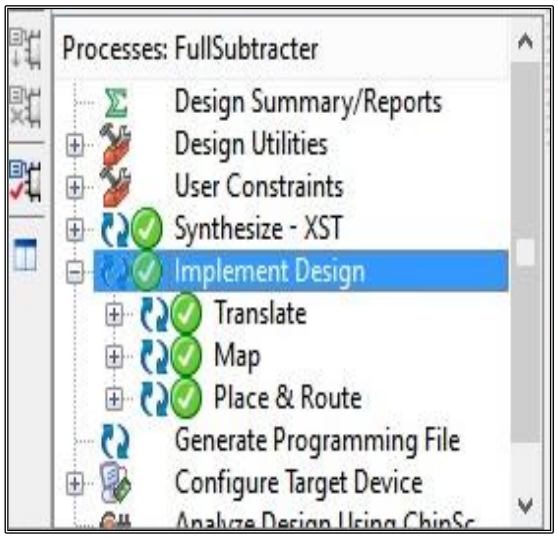


Fig 4.19: Successful implementation

21. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip (Fig 4.20).

22. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the full subtractor circuit.

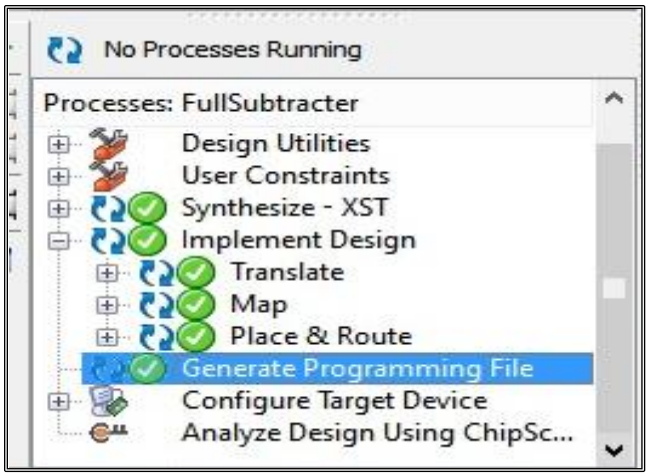


Fig 4.20: Bitstream generation

VERILOG CODE FOR THE FULL SUBTRACTER

module FullSubtracter(**input** a, **input** b, **input** c, **output** Difference, **output** Borrow);

assign Difference =a**^**b**^**c;

assign Borrow= ((**~**a)**&**b)|(b**&**c)|(c**&**(**~**a));

endmodule

User Constraints File (UCF) For FULL SUBTRACTER

NET "a" LOC="P93";

NET "b" LOC="P92";

NET "c" LOC="P88";

NET "Difference" LOC="P105";

NET "Borrow" LOC="P104";

EXPERIMENT: 5

Objective: To Design and implement 3:8 decoder circuit on FPGA Board.

Software: Xilinx ISE Design Suite14.5

Target Hardware: FPGA Board

Theory: - In digital electronics, a binary decoder is a combinational logic circuit that converts a binary integer value to an associated pattern of output bits. They are used in a wide variety of applications, including data demultiplexing, seven segment displays, and memory address decoding. There are several types of binary decoders, but in all cases a decoder is an electronic circuit with multiple data inputs and multiple outputs that converts every unique combination of data input states into a specific combination of output states. In addition to its data inputs, some decoders also have one or more "enable" inputs. Depending on its function, a binary decoder will convert binary information from n input signals to as many as 2n unique output signals. Some decoders have less than 2n output lines; in such cases, at least one output pattern will be repeated for different input values.

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.
2. From the toolbar, select **File** => **New Project**. In the **New Project Wizard** window type "Decoder" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

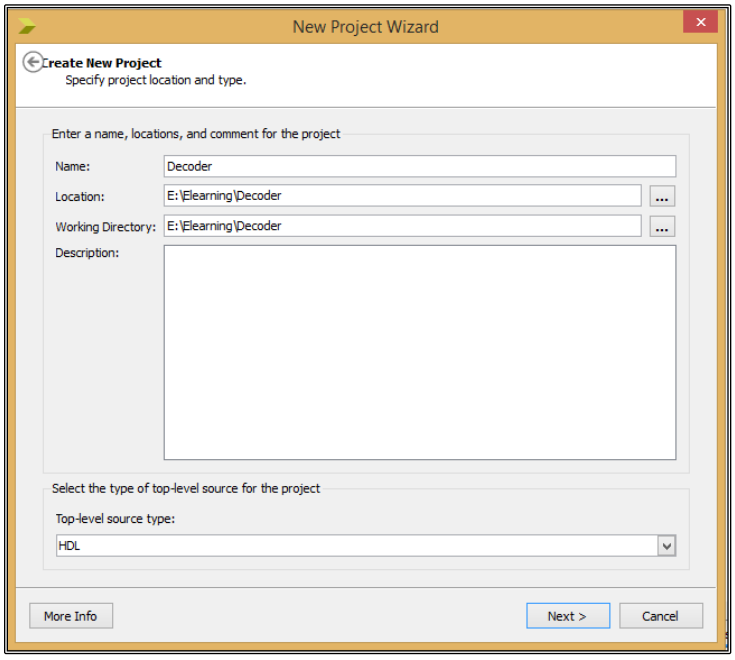


Fig 5.1: New Project Decoder

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "Decoder" for the new source in the **File Name** field. Click **Next**.

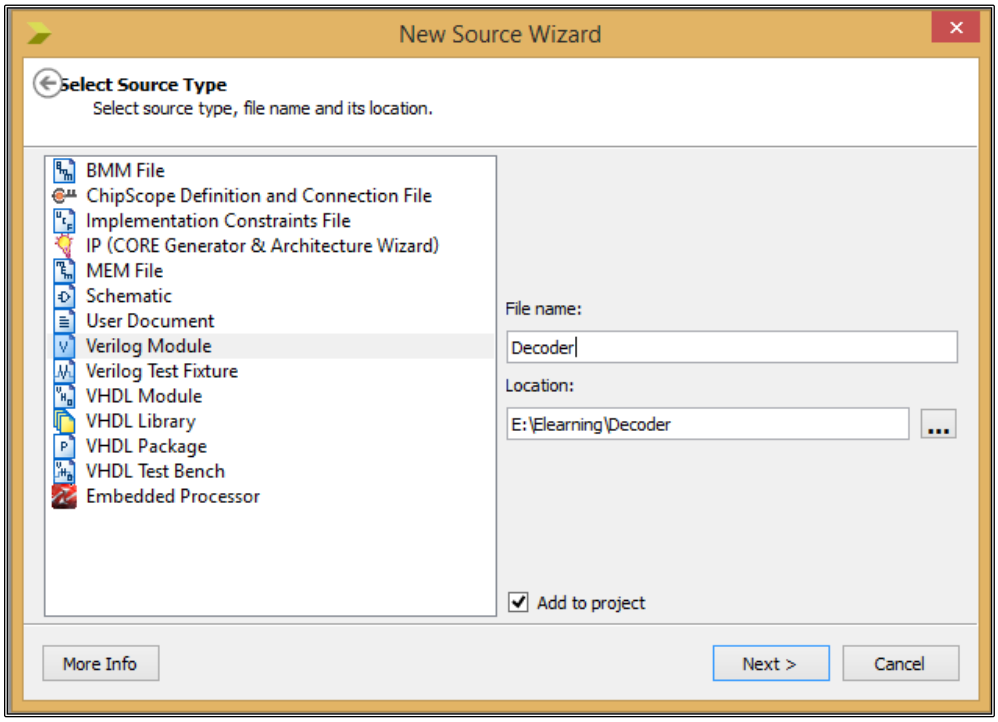


Fig 5.2: New Source Decoder

6. In the **Define Module** page, enter the port information for the "Decoder" as follows:

   a. In the first two **Port Name** fields, enter 'I' and 'Y'.
   b. Set the **Direction field** to 'output' for 'Y' and to 'input' for 'I'.
   c. Set the MSB bus range for input 'I' as 2 as 'I' is a 3-bit vector and MSB range for 'Y' as 7 as 'Y' is an 8-bit vector.

   Click **Next.**

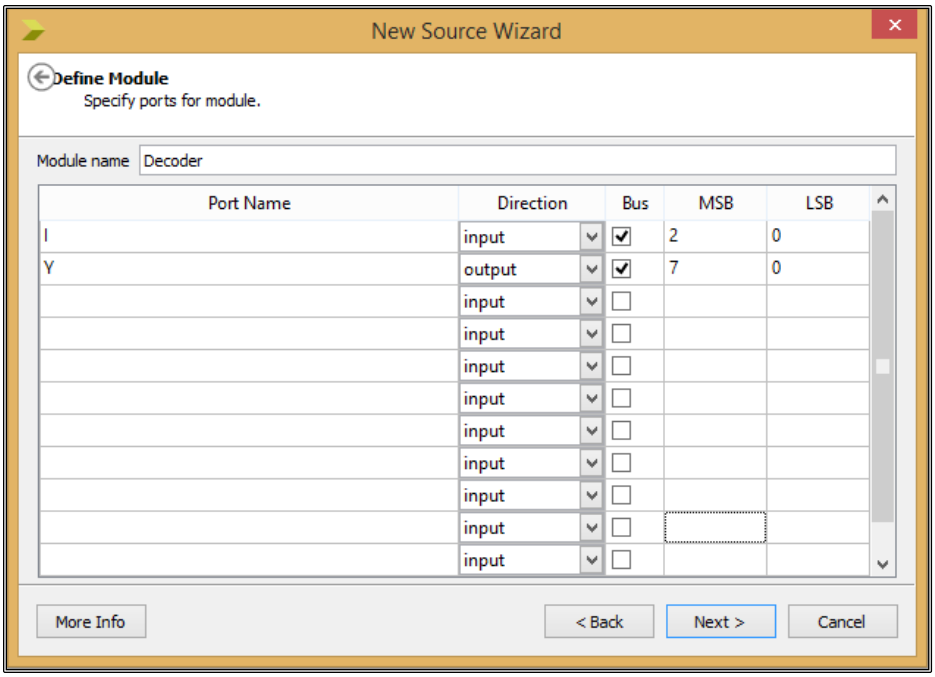


Fig 5.3: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the ISE Text Editor, the ports are already declared in the Decoder HDL file, and some of the basic file structure is already in place. Now you have to write the desired 3:8 decoder logic. For this example behavioral modeling is used (Fig 5.4). **Save** the file.

```
20  //////////////////////////////////////////////////////////////////
21  module Decoder(
22       input [2:0] I,
23       output [7:0] Y
24       );
25
26  always@(I)
27  begin
28  case(I)
29
30  3'b000: Y=8'b00000001;
31  3'b001: Y=8'b00000010;
32  3'b010: Y=8'b00000100;
33  3'b011: Y=8'b00001000;
34  3'b100: Y=8'b00010000;
35  3'b101: Y=8'b00100000;
36  3'b110: Y=8'b01000000;
37  3'b111: Y=8'b10000000;
38  endcase
39  end
40  endmodule
41
42
```

Fig 5.4: Complete Decoder Module

9. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window on the left pane.

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

11. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (Decoder.v) and select **New Source**. In the New Source Wizard window, select **Verilog Test Fixture** .Write the "DecoderTest" in the **File name** field for the test bench. Click **Next**.

    In the next window, select the source file you want to associate with the test bench. Check the **summary** and click **Finish**.

12. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values (Fig 5.5). After editing the test bench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench. If no errors are present double click on Simulate Behavioral Model.

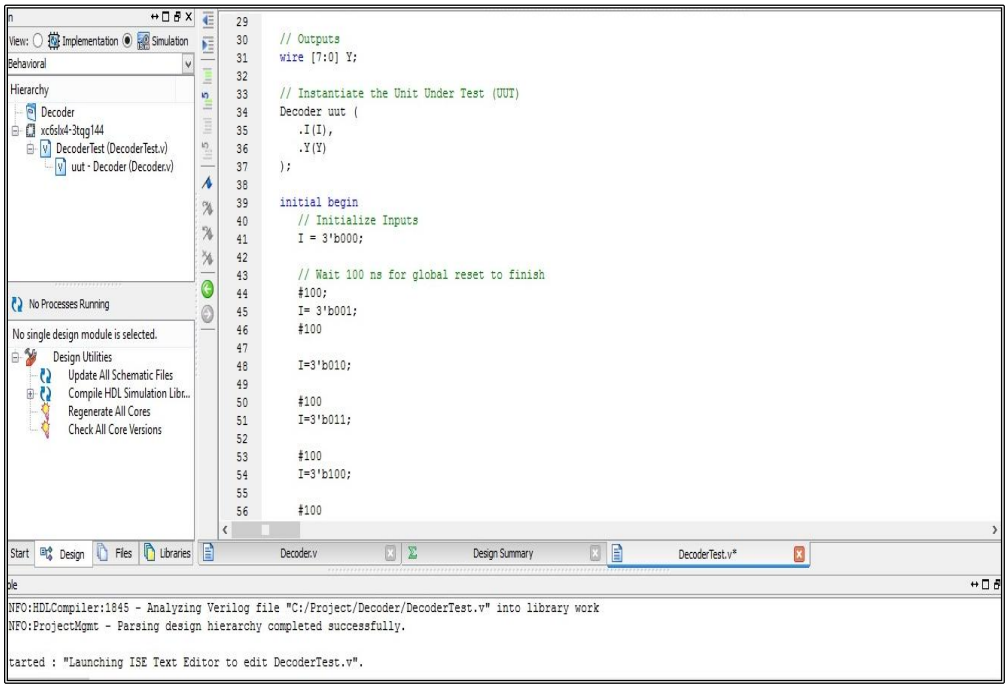


Fig 5.5: 3:8 Decoder testbench

13. A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs (Fig 5.6).

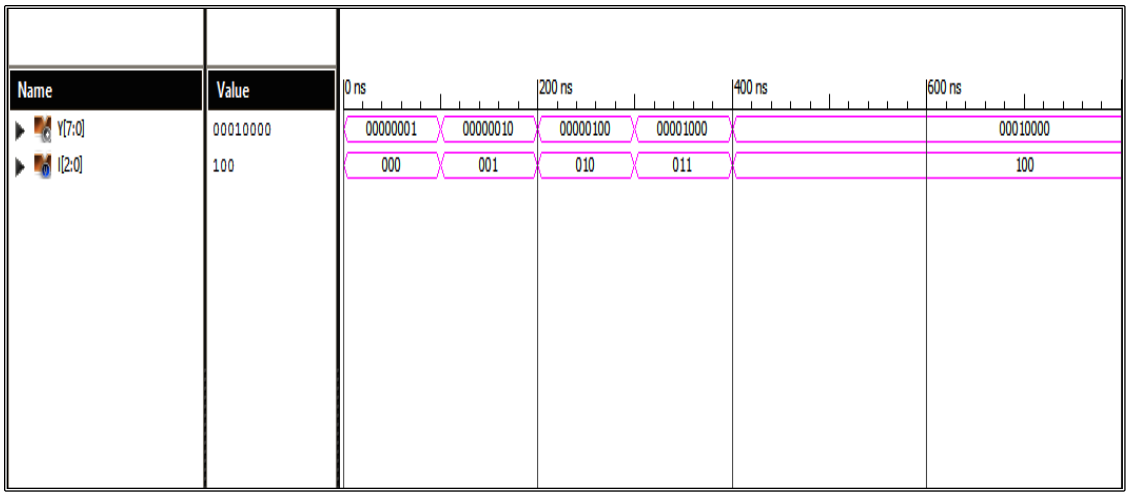


Fig 5.6: Simulated Waveform for Decoder

14. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan (Fig 5.7).

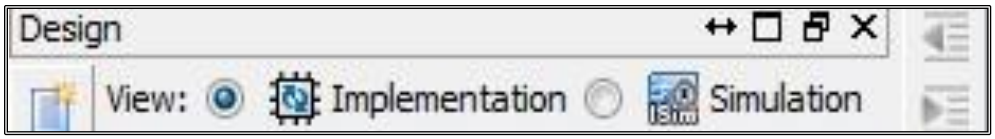


Fig 5.7: Design Implementation

15. For UCF File, right click on the design (Decoder.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "decoder_1" (Fig 5.8).

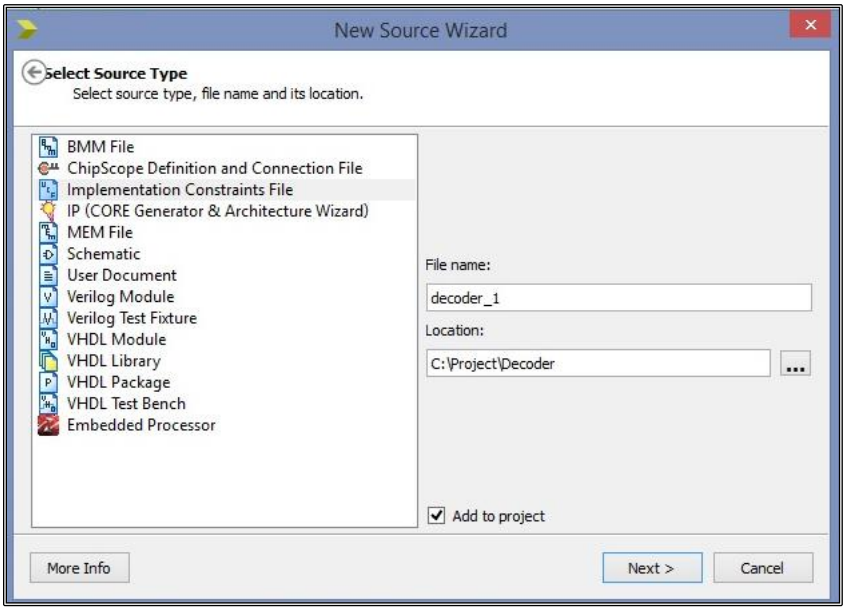


Fig 5.8: UCF creation

Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

16. A blank window will appear in a new tab having the design's name as "decoder_1.ucf" . Add the constraints for each input port and output port (Fig 5.9).

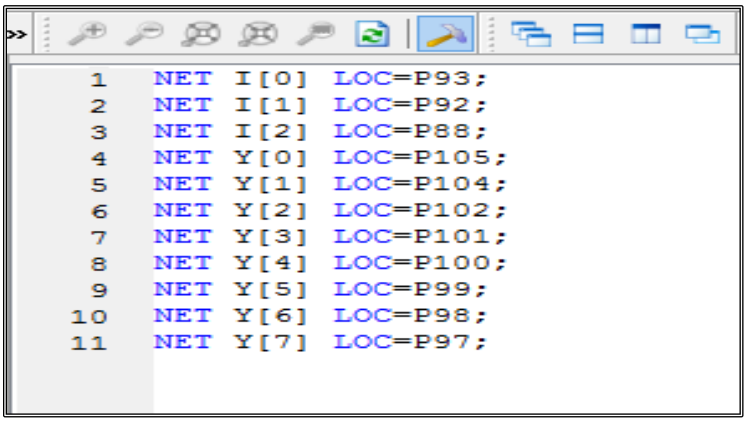


Fig 5.9: UCF declaration

17. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

18. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View**)and **Technology View** for the "Decoder.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

19. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route.

20. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip (Fig 5.10).

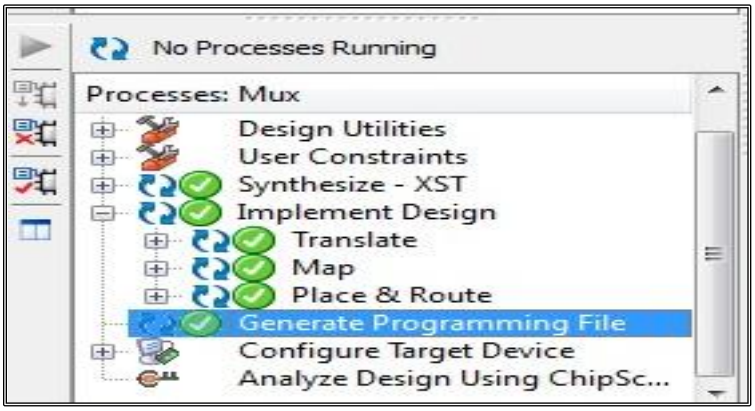


Fig 5.10: Bitstream generation

21. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the decoder circuit.

Verilog Code of the 3:8 Decoder

```
module Decoder(input [2:0] I, output reg [7:0] Y);
always@(I)
begin
case(I)

3'b000: Y=8'b00000001;

3'b001: Y=8'b00000010;

3'b010: Y=8'b00000100;

3'b011: Y=8'b00001000;

3'b100: Y=8'b00010000;

3'b101: Y=8'b00100000;

3'b110: Y=8'b01000000;

3'b111: Y=8'b10000000;

endcase
end
endmodule
```

User Constraints File (UCF) for the Decoder

```
NET "I[0]" LOC="P93";

NET "I[1]" LOC="P92";

NET "I[2]" LOC="P88";

NET "Y[0]" LOC="P105";

NET "Y[1]" LOC="P104";

NET "Y[2]" LOC="P102";

NET "Y[3]" LOC="P101";

NET "Y[4]" LOC="P100";

NET "Y[5]" LOC="P99";

NET "Y[6]" LOC="P98";

NET "Y[7]" LOC="P97";
```

EXPERIMENT: 6

Objective: To design and implement 8:3 Encoder circuit on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: An encoder is an electronic device used to convert an analogue signal to a digital signal such as a BCD code. It has a number of input lines, but only one of the inputs is activated at a given time and produces an N-bit output code that depends on the activated input. The encoder allows 2^N inputs and generates N-number of outputs. For example, in 4:2 encoder, if we give 4 inputs it produces only 2 outputs.

The truth table of 4:2 Encoder is shown in Table 6.1:

Table 6.1: Truth table 4:2 Encoder

INPUT				OUTPUT	
I3	I2	I1	I0	Y1	Y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.

2. From the toolbar, select **File** => **New Project**. In the **New Project Wizard** window type "Encoder" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

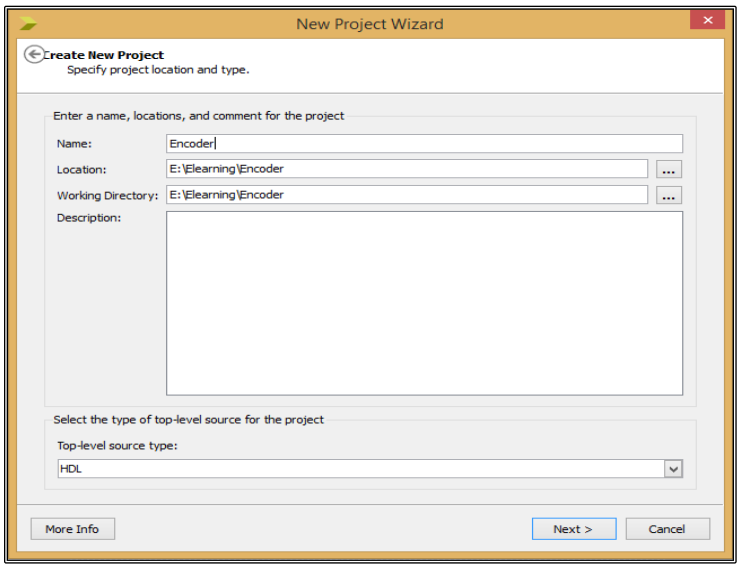


Fig 6.1: New Project Encoder

3. Next in the **New Project Wizard Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "Encoder" for the new source in the **File Name** field. Click **Next**.

6. In the **Define Module** page, enter the port information for the "Encoder" as follows (Fig 6.2):

   a. In the first two **Port Name** fields, enter 'I' and 'Y'.
   b. Set the **Direction field** to 'output' for 'Y' and to 'input' for 'I'.
   c. Set the MSB bus range for input 'I' from 7 to 0 as 'I' is 8-bit vector and MSB range for 'Y' from 2 to 0 as 'Y' is a 3-bit vector.

   Click **Next.**

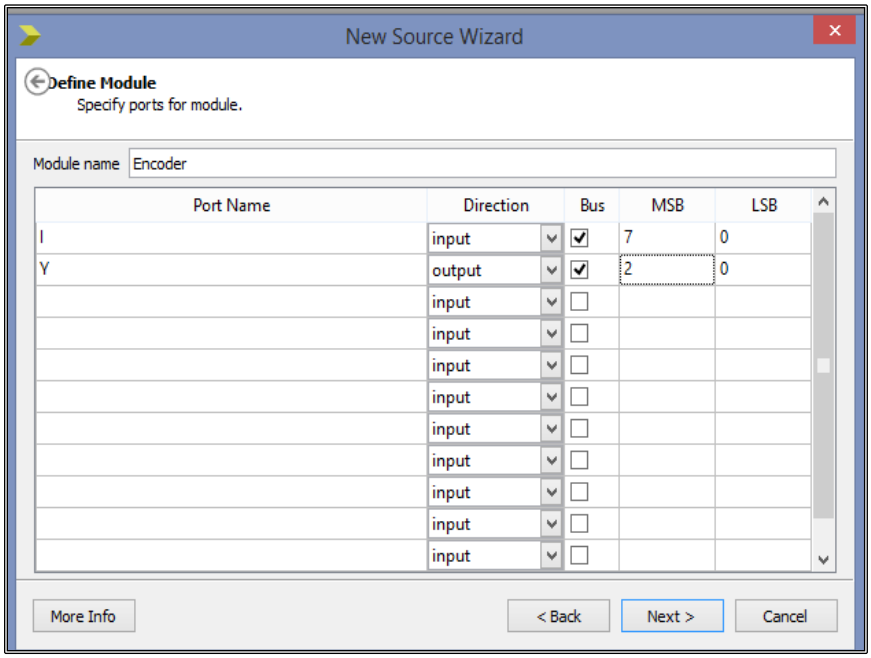


Fig 6.2: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the Xilinx ISE interface you can see that the new source file "Encoder.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the Encoder HDL file, and some of the basic file structure is already in place (Fig 6.3).

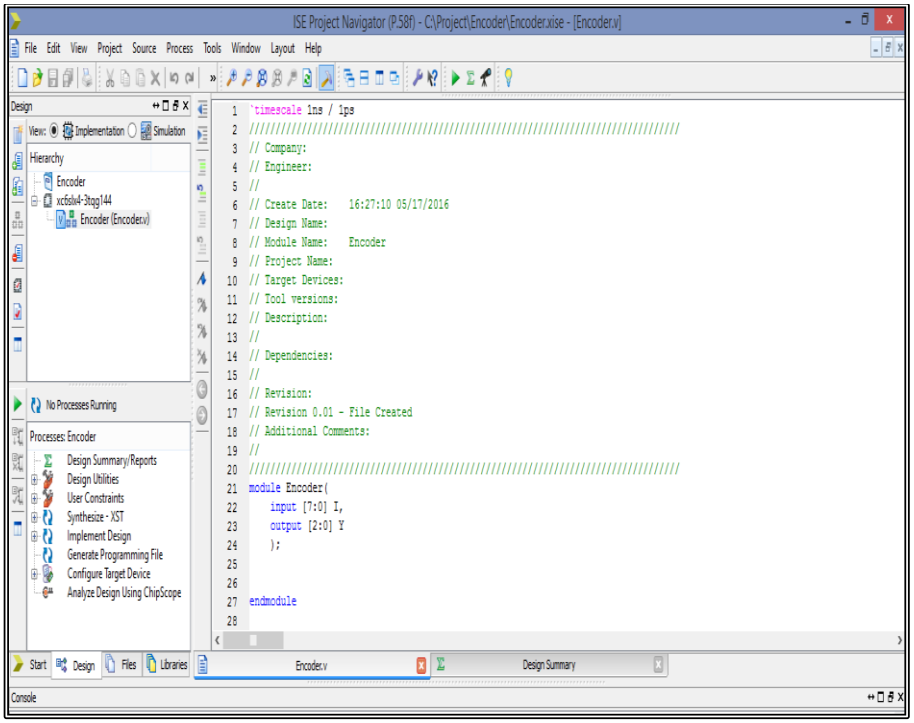


Fig 6.3: Encoder module structure

9. Next, you have to write the desired encoder logic. For this example behavioral modeling is used. **Save** the file.

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window.

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

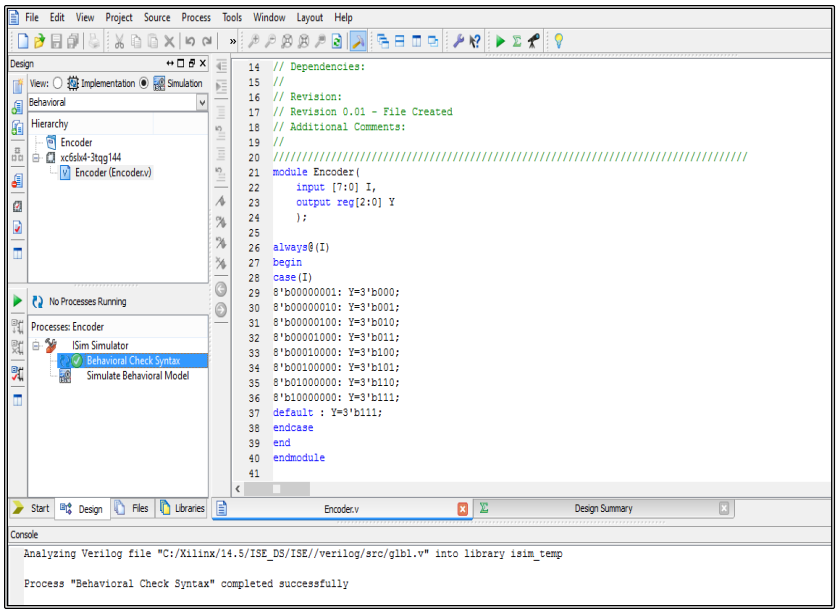


Fig 6.4: Behavioral Check Syntax

12. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (Encoder.v) and select **New Source**. In the New Source Wizard window, select **Verilog Test Fixture** .Write the "Encoder_Test" in the **File name** field for the test bench. Click **Next**.

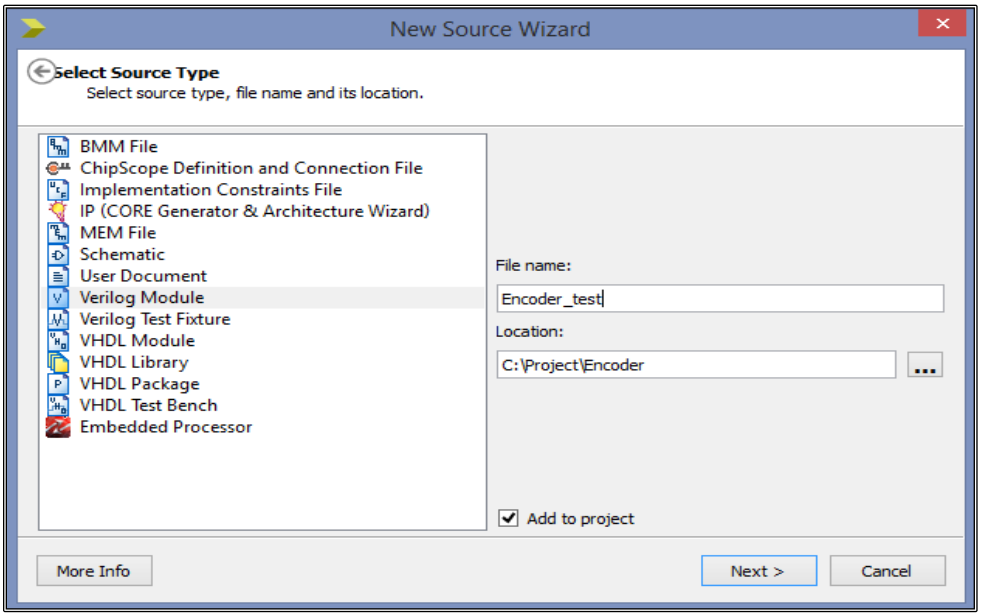


Fig 6.5: Testbench Creation

13. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

14. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values. You have to provide different combinations of data line I (Fig 6.6). Save the test bench.

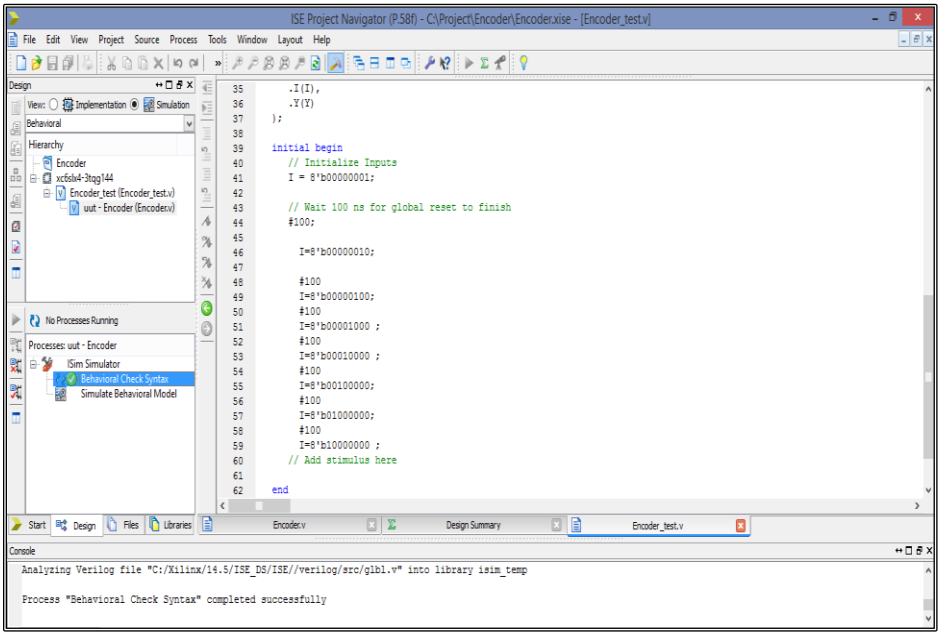


Fig 6.6: Encoder Testbench

15. After editing the test bench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench. If no errors are present double click on **Simulate Behavioral Model**. A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs.

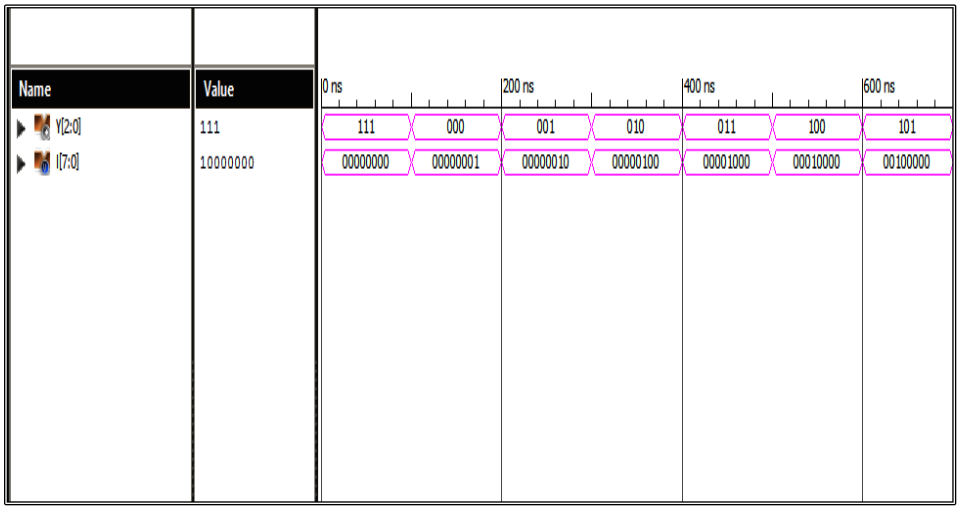


Fig 6.7: Simulated waveform

16. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan (Fig 6.8).

Fig 6.8: Design Implementation

17. For UCF File, right click on the design (Encoder.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "Encoder1".

    Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

18. A blank window will appear in a new tab having the design's name as "Encoder1.ucf" . Add the constraints for each input port and output port (Fig 6.9).Save.

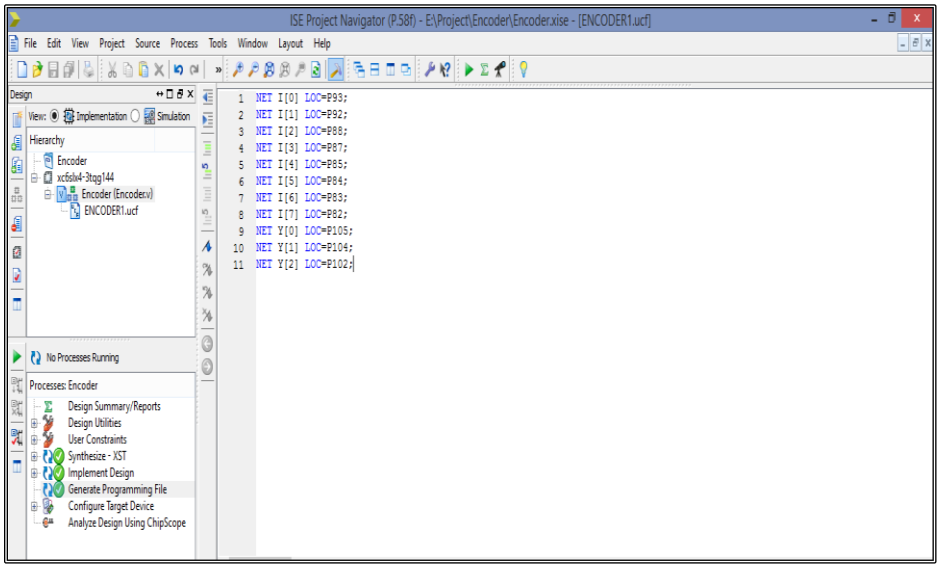


Fig 6.9: UCF declaration

19. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

20. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** and **Technology View** for the "Encoder.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

21. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route.

22. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip.

23. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the encoder circuit.

Verilog Code for the 8:3 Encoder

```
module Encoder( input [7:0] I, output reg [2:0] Y  );

always@(I)
begin
case(I)
8'b00000001: Y=3'b000;

8'b00000010: Y=3'b001;

8'b00000100: Y=3'b010;

8'b00001000: Y=3'b011;

8'b00010000: Y=3'b100;

8'b00100000: Y=3'b101;

8'b01000000: Y=3'b110;

8'b10000000: Y=3'b111;

default : Y=3'b111;

endcase
end
endmodule
```

User Constraints File (UCF) for the 8:3 Encoder

```
NET I[0] LOC=P93;

NET I[1] LOC=P92;

NET I[2] LOC=P88;

NET I[3] LOC=P87;

NET I[4] LOC=P85;

NET I[5] LOC=P84;

NET I[6] LOC=P83;

NET I[7] LOC=P82;

NET Y[0] LOC=P105;

NET Y[1] LOC=P104;

NET Y[2] LOC=P102;
```

EXPERIMENT: 7

Objective: To design and implement 4 Bit Comparator circuit on the FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: Binary comparators, also called digital comparators or logic comparators are combinational logic circuits that are used for testing whether the value represented by one binary word is greater than, less than, or equal to the value represented by another binary word. The following two basic types of comparator can be used.

- Equality comparators.
- Magnitude comparators.

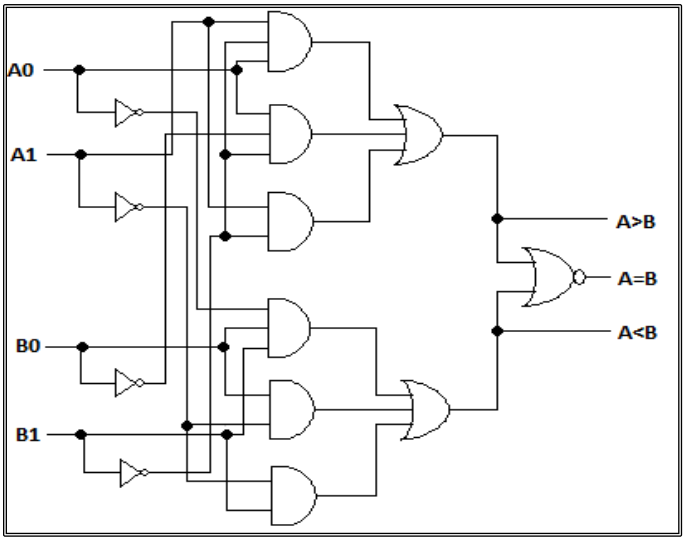


Fig 7.1: Logic Diagram

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.
2. From the menu bar, select **File** => **New Project**. In the **New Project Wizard** window type "Comparator" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

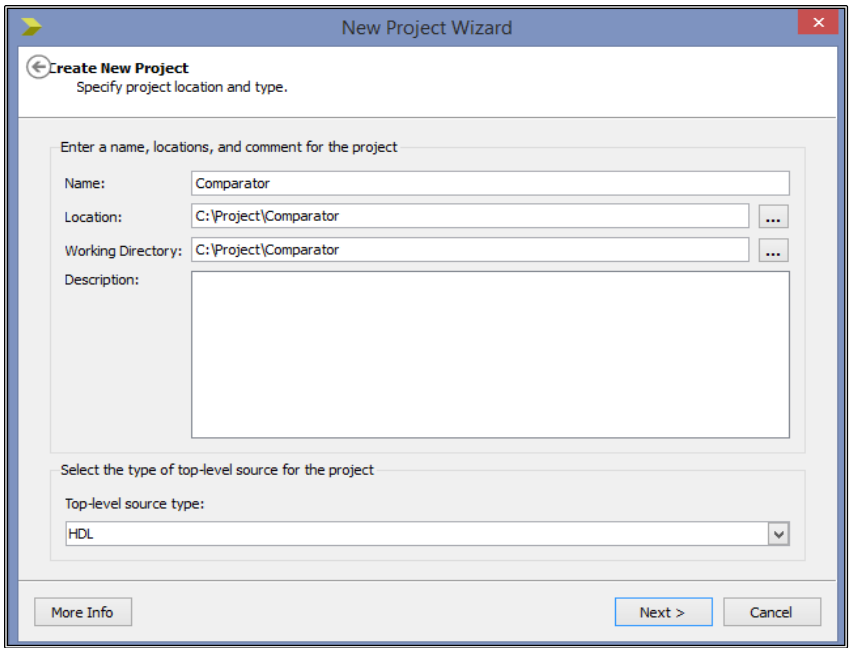


Fig 7.2: New Project

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "Comparator" for the new source in the **File Name** field. Click **Next**.

6. In the **Define Module** page, enter the port information for the "Comparator" as follows:

   a. In the first three **Port Name** fields, enter 'y', 'a' and 'b'.
   b. Set the **Direction field** to 'output' for 'y' and to 'input' for 'a', 'b'.
   c. Set the MSB bus range for input 'a', 'b' from 3 to 0 and MSB range for 'y' from 2 to 0.
   Click **Next.**

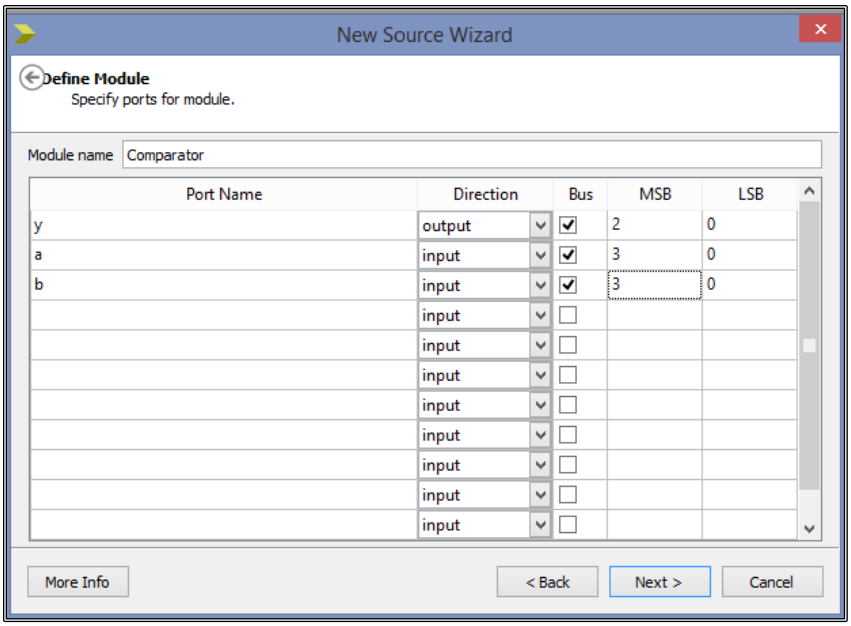


Fig 7.3: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the Xilinx ISE interface you can see that the new source file "Comparator.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the Comparator HDL file, and some of the basic file structure is already in place (Fig 7.4).

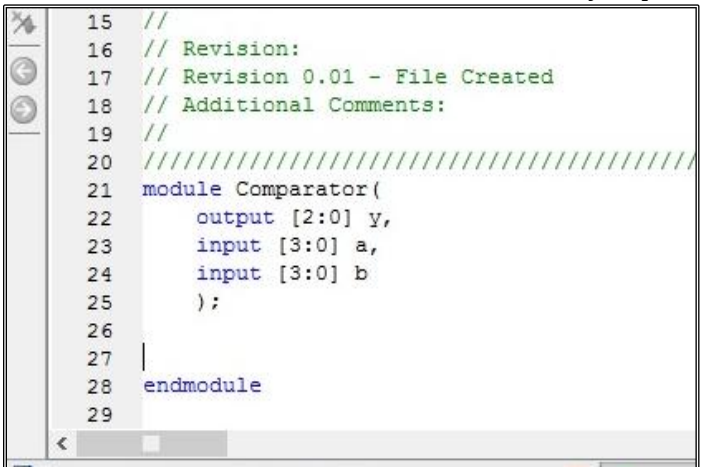


Fig 7.4: Comparator module structure

9. Next, you have to write the desired comparator logic. For this example dataflow modeling is used. **Save** the file.

```
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////
21  module Comparator(
22      output [2:0] y,
23      input [3:0] a,
24      input [3:0] b
25      );
26
27  assign y[0] =a>b?1:0;
28  assign y[1] = a<b?1:0;
29  assign y[2] = a==b?1:0;
30
31  endmodule
```

Fig 7.5: Complete Comparator Module

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window.

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

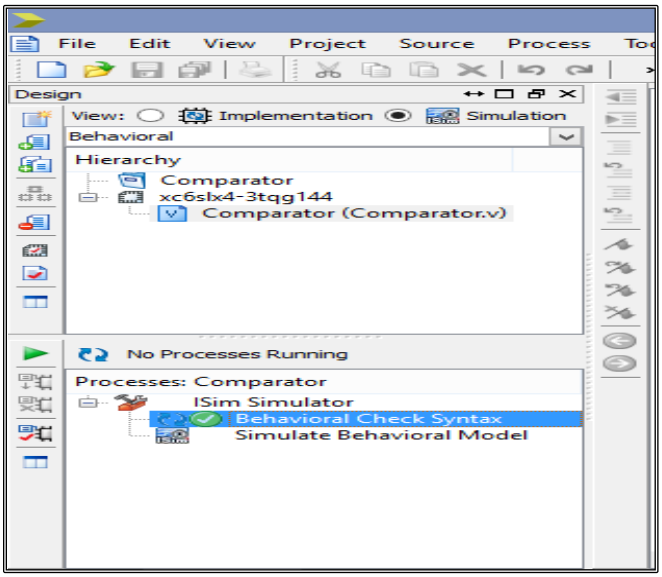


Fig 7.6: Behavioral Check Syntax

12. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (Encoder.v) and select **New Source**. In the New Source Wizard window, select **Verilog Test Fixture** .Write the "Comparator_Test" in the **File name** field for the test bench (Fig 7.6). Click **Next**.

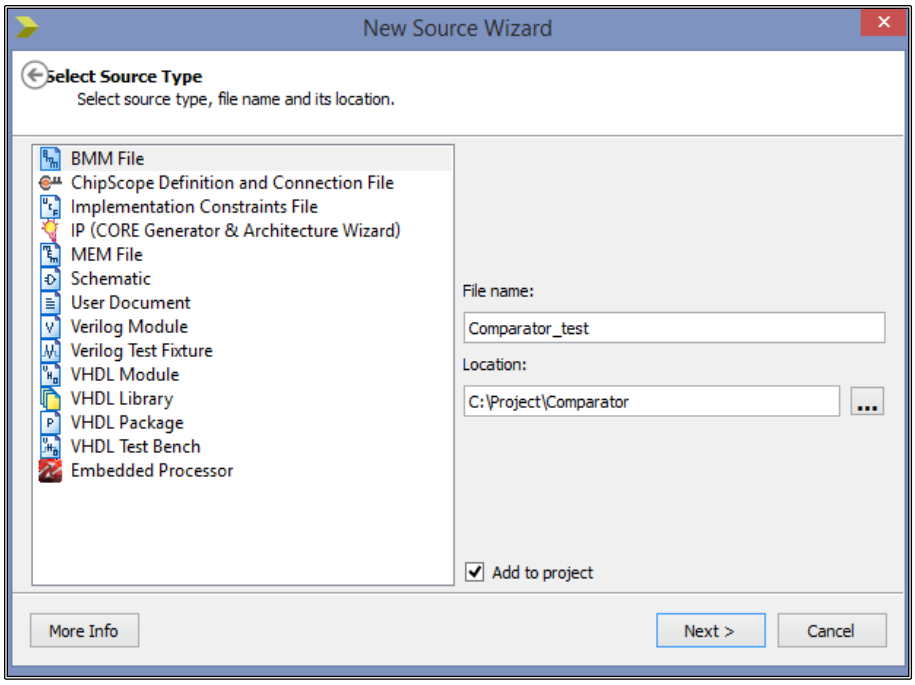


Fig 7.7: Testbench Creation

13. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

14. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values. After editing the test bench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench. If no errors are present double click on **Simulate Behavioral Model.**

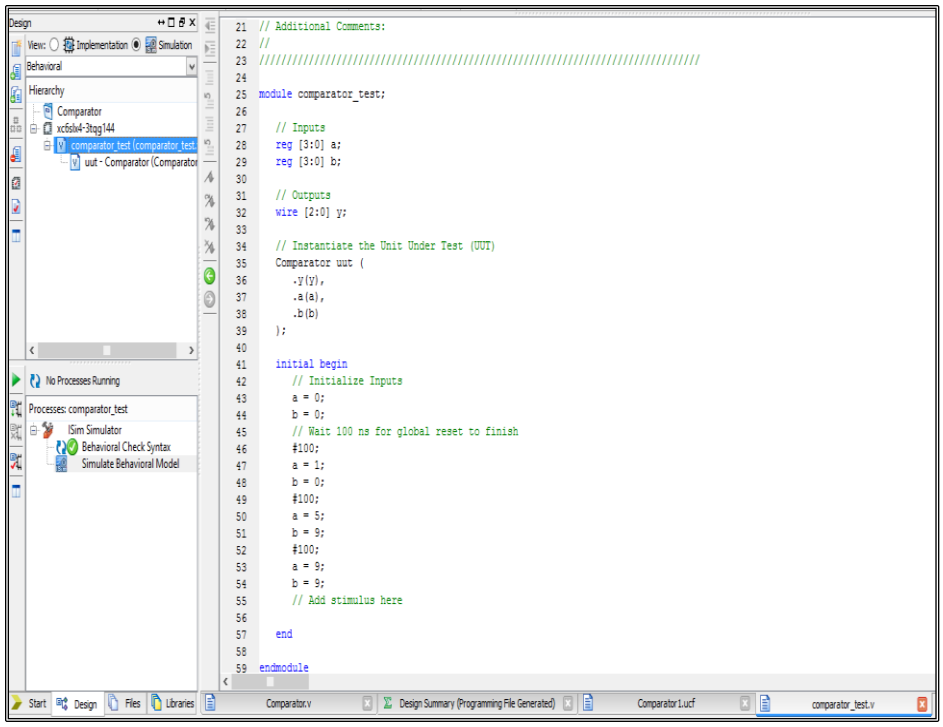


Fig 7.8: Complete testbench

15. A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs.

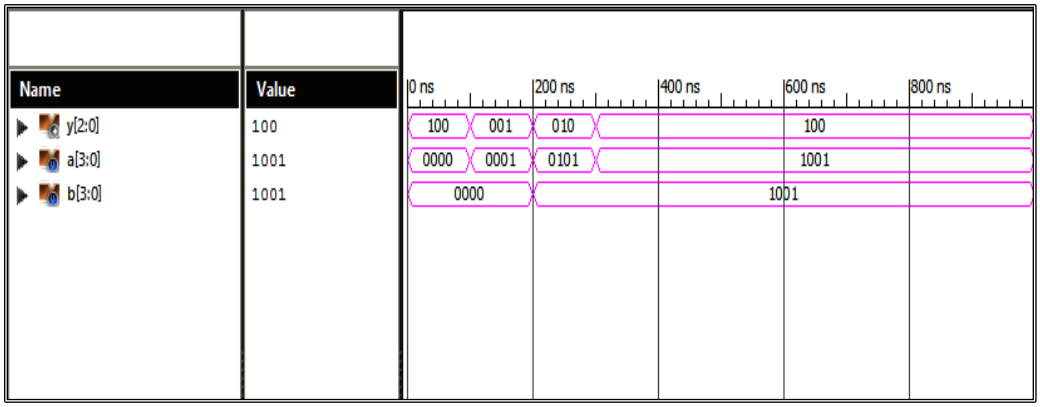


Fig 7.9: Simulated Waveform

16. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan (Fig 4.9).

Fig 7.10: Design Implementation

17. For UCF File, right click on the design (Comparator.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "comparator1".

    Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

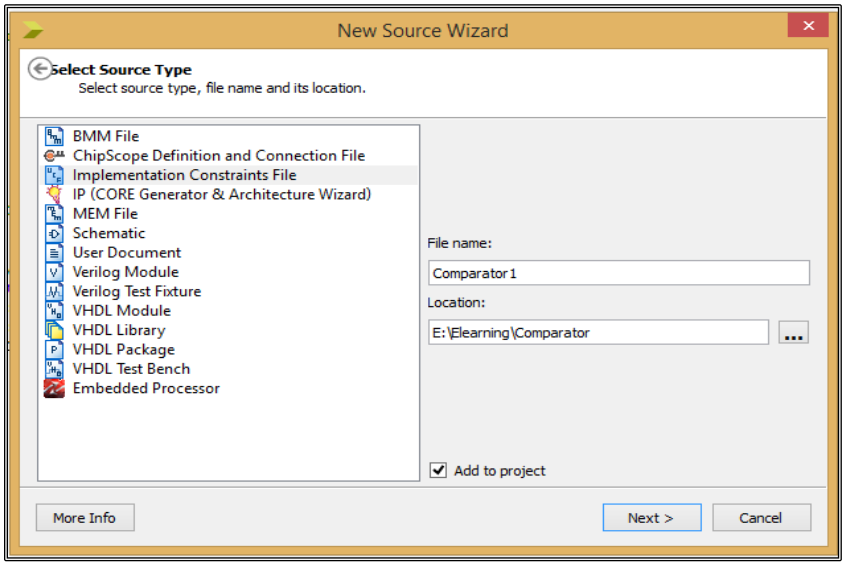


Fig 7.11: UCF creation

18. A blank window will appear in a new tab having the design's name as "Comparator1.ucf" . Add the constraints for each input port and output port (Fig 7.12).

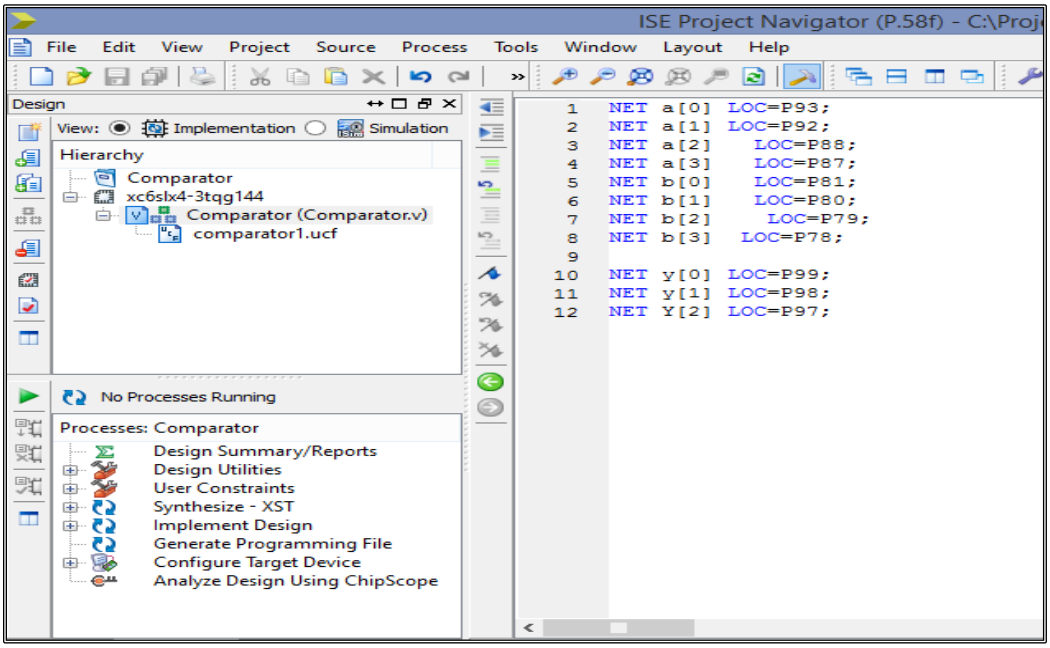


Fig 7.12: UCF declaration

19. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

20. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** and **Technology View** for the "Fulladder.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

21. Next step is the implementation of the design.Right click **Implement Design** and click **Run**. After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route.

22. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip.

23. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the full subtractor circuit.

VERILOG CODE FOR THE 4 BIT COMPARATOR

```
module Comparator( output [2:0] y, input [3:0] a, input [3:0] b);

assign y[0] = (a>b)?1:0;

assign y[1] = (a<b)?1:0;

assign y[2] = (a==b)?1:0;

endmodule
```

User Constraints File (UCF) for the 4 BIT COMPARATOR

```
NET a[0] LOC=P93;

NET a[1] LOC=P92;

NET a[2] LOC=P88;

NET a[3] LOC=P87;

NET b[0] LOC=P81;

NET b[1] LOC=P80;

NET b[2] LOC=P79;

NET b[3] LOC=P78;

NET y[0] LOC=P99;

NET y[1] LOC=P98;

NET Y[2] LOC=P97;
```

EXPERIMENT: 8

Objective: To design and implement BCD to Seven Segment Decoder on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: A decoder IC is a device which converts one digital format into another and the most commonly used device for doing this is the Binary Coded Decimal (BCD) to 7-Segment Display Decoder. BCD to Seven Segment Decoder is used to display the number from 0 to 9 or Hex Character A to F on Seven segment display. Seven-segment displays are used to display the digits in digital watches, calculators, clocks, measuring instruments and digital counters, etc. Normally LCD and LED segments provide the display output of numerical numbers and characters. Seven-segment displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters and they consist of 7 individual LED's (the segments), within one single display package.

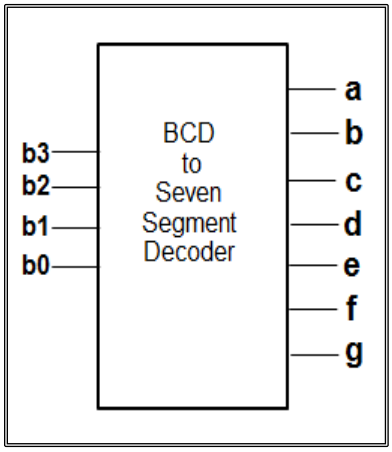


Fig 8.1: Block Diagram

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.
2. From the menu bar, select **File** => **New Project**. In the **New Project Wizard** window type "BCDtoSevensegment" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

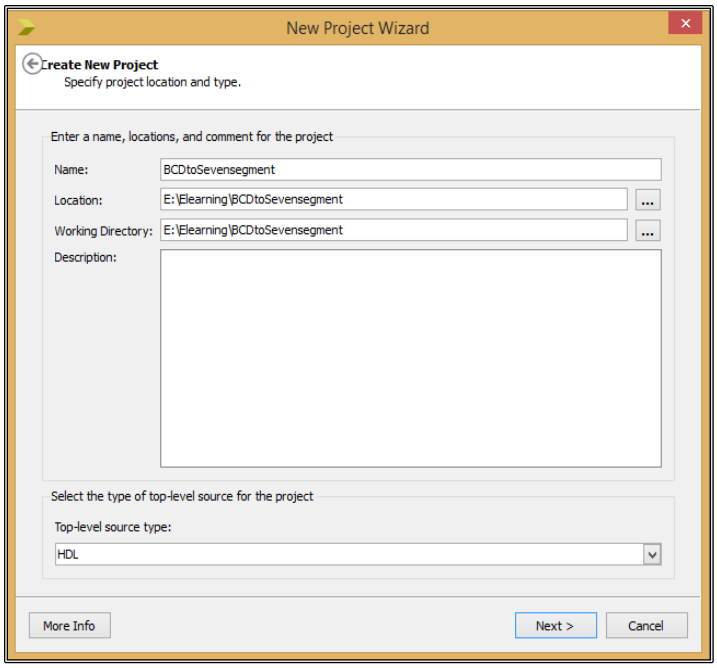


Fig 8.2: New Project BCDtoSevensegment

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "seven_seg" for the new source in the **File Name** field. Click **Next**.

6. In the **Define Module** page, enter the port information for the "seven_seg" as follows:

   a. In the first two **Port Name** fields, enter 'bcd' and 'seg'.
   b. Set the **Direction field** to 'input' for 'bcd' and to 'output' for 'seg'.
   c. Set the MSB bus range for input 'bcd' from 9 to 0 and MSB range for 'seg' from 6 to 0.
   Click **Next.**

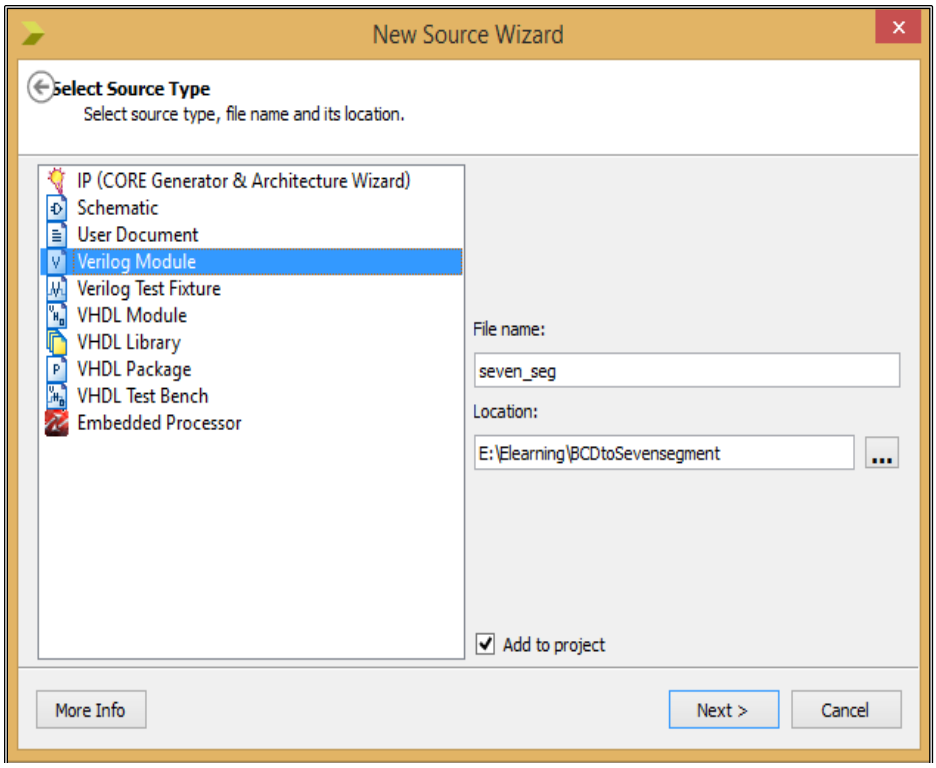


Fig 8.3: Module seven_seg

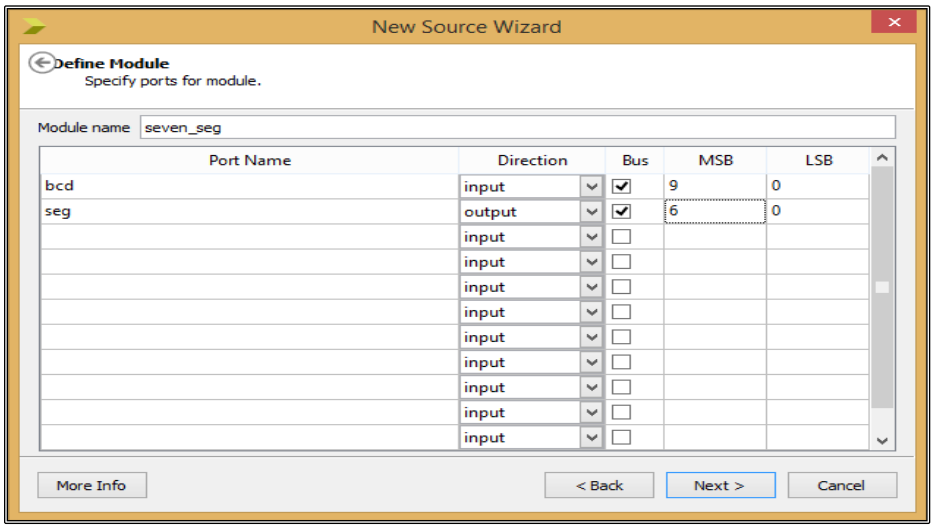


Fig 8.4: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the Xilinx ISE interface you can see that the new source file "seven_seg.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the seven_seg HDL file, and some of the basic file structure is already in place.

9. Next, you have to write the desired BCD to seven segment logic. For this example behavioral modeling is used. **Save** the file.

```
21  module seven_seg(
22      input [9:0] bcd,
23      output reg [6:0] seg
24      );
25
26  always@(bcd)
27  begin
28  case(bcd)
29
30  4'b0000 : seg = 7'b0000001;
31  4'b0001 : seg = 7'b1001111;
32  4'b0010 : seg = 7'b0010010;
33  4'b0011 : seg = 7'b0000110;
34  4'b0100 : seg = 7'b1001100;
35  4'b0101 : seg = 7'b0100100;
36  4'b0110 : seg = 7'b0100000;
37  4'b0111 : seg = 7'b0001111;
38  4'b1000 : seg = 7'b0000000;
39  4'b1001 : seg = 7'b0000100;
40  default :seg =7'b0000001;
41  endcase
42  end
43
44  endmodule
45
```

Fig 8.5: Complete module seven_seg

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window.

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

12. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

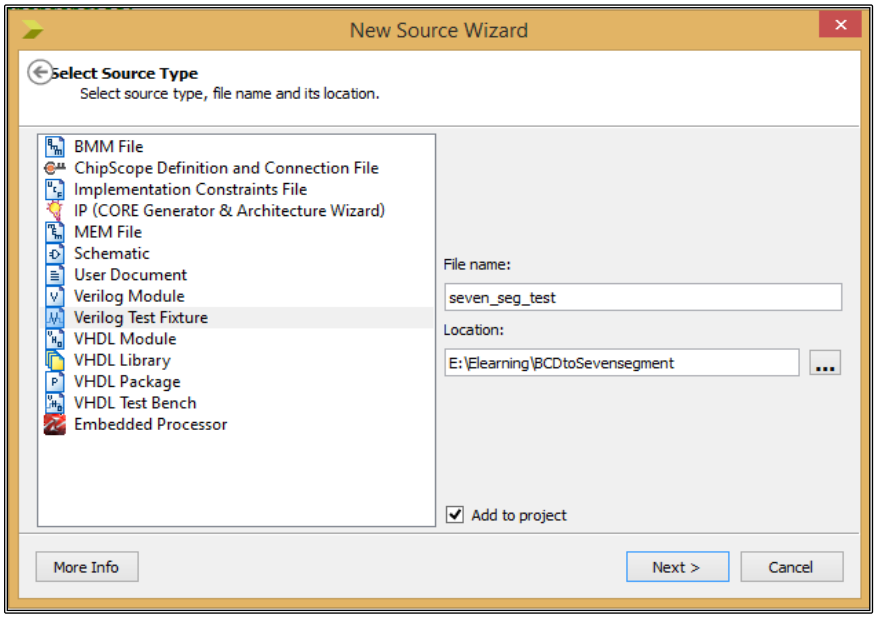


Fig 8.6: Testbench Creation

13. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values. After editing the test bench, click on **Behavioral Check**

Syntax in the Processes window to check for syntax errors in the test bench. If no errors are present double click on **Simulate Behavioral Model.**

```
25  module seven_seg_test;
26
27     // Inputs
28     reg [9:0] bcd;
29
30     // Outputs
31     wire [6:0] seg;
32
33     // Instantiate the Unit Under Test (UUT)
34     seven_seg uut (
35        .bcd(bcd),
36        .seg(seg)
37     );
38
39     initial begin
40        // Initialize Inputs
41        bcd = 0;
42        #100;
43        bcd = 7;
44        #100;
45        bcd = 4;
46        #100;
47        bcd = 9;
48        #100;
49     end
50
51  endmodule
```

Fig 8.7: Complete testbench for seven_seg

14. A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs.

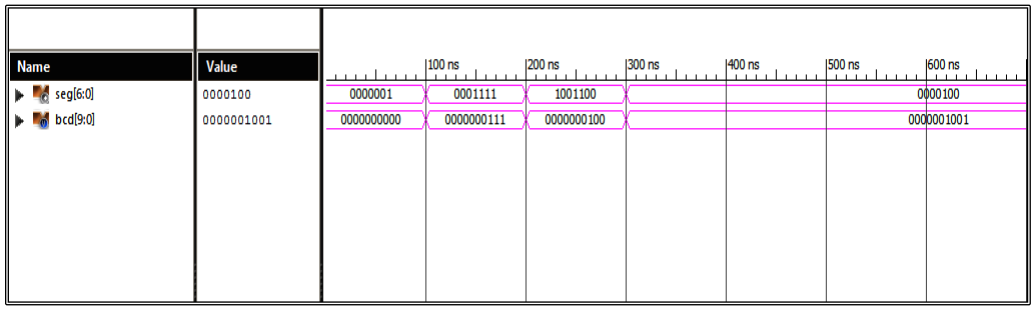


Fig 8.8: Simulated waveform

15. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan.

16. For UCF File, right click on the design (seven_seg.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "seven1".

    Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

17. A blank window will appear in a new tab having the design's name as "seven1.ucf". Add the constraints for each input port and output port.

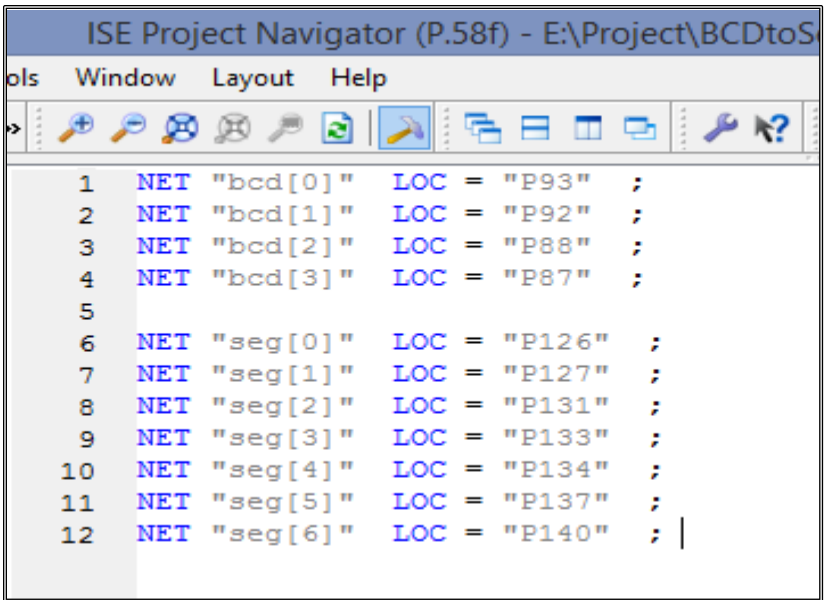
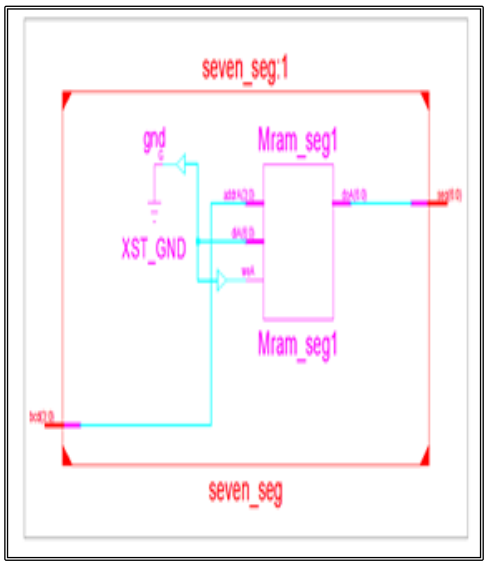


Fig 8.9: UCF declaration

18. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

19. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** and **Technology View** for the "seven_seg.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.



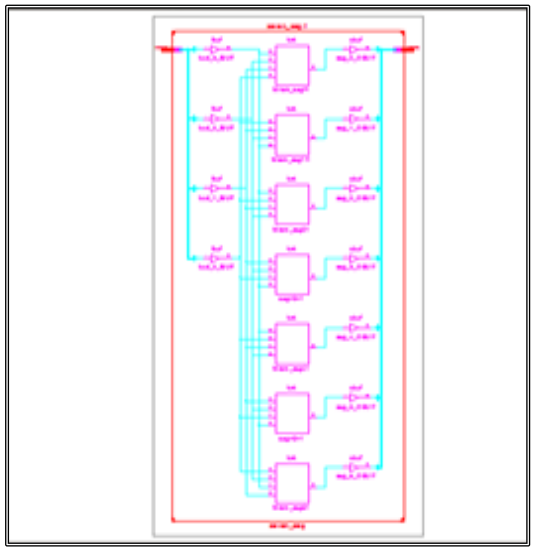


Fig 8.10: RTL Schematic

Fig 8.11: Technology Schematic

20. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route.

21. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip.

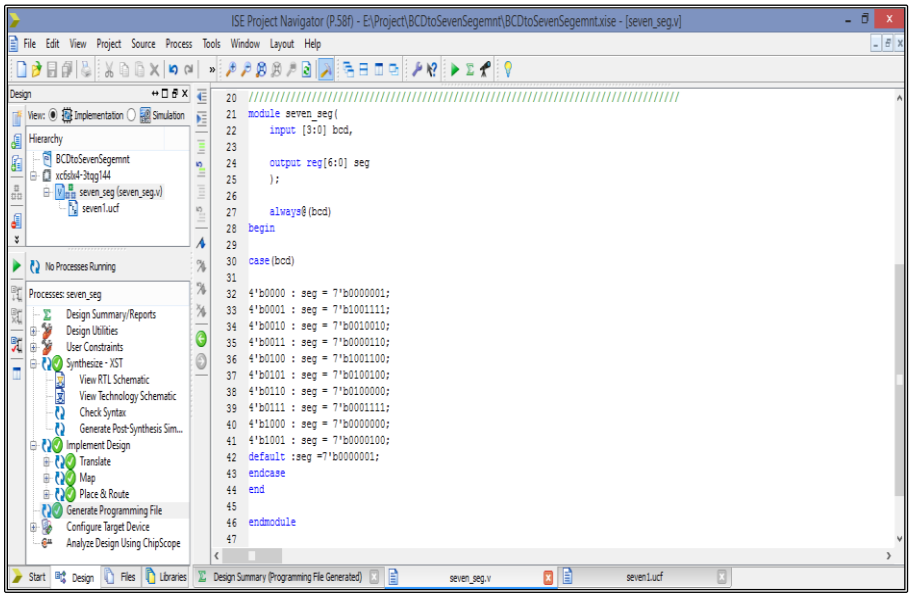


Fig 8.12: Bitstream Generation

22. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can check the output on seven segment display by providing input through the switches.

Verilog Code for BCD To Seven Segment Decoder

```
module seven_seg( input [3:0] bcd, output reg[6:0] seg);

always@(bcd)

begin
case(bcd)

4'b0000 : seg = 7'b0000001;
4'b0001 : seg = 7'b1001111;
4'b0010 : seg = 7'b0010010;
4'b0011 : seg = 7'b0000110;
4'b0100 : seg = 7'b1001100;
4'b0101 : seg = 7'b0100100;
4'b0110 : seg = 7'b0100000;
4'b0111 : seg = 7'b0001111;
4'b1000 : seg = 7'b0000000;
4'b1001 : seg = 7'b0000100;
default :seg =7'b0000001;
endcase
end

endmodule
```

User Constraints File (UCF) For BCD To Seven Segment Decoder

```
NET "bcd[0]"  LOC = "P93";
NET "bcd[1]"  LOC = "P92";
NET "bcd[2]"  LOC = "P88";
NET "bcd[3]"  LOC = "P87";

NET "seg[0]"  LOC = "P126";
NET "seg[1]"  LOC = "P127";
NET "seg[2]"  LOC = "P131";
NET "seg[3]"  LOC = "P133";
NET "seg[4]"  LOC = "P134";
NET "seg[5]"  LOC = "P137";
NET "seg[6]"  LOC = "P140";
```

EXPERIMENT: 9

Objective: To design and implement D flip flop circuit on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: The D flip-flop is the Most widely used flip flop. It is also known as a "data" or "delay" flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line.

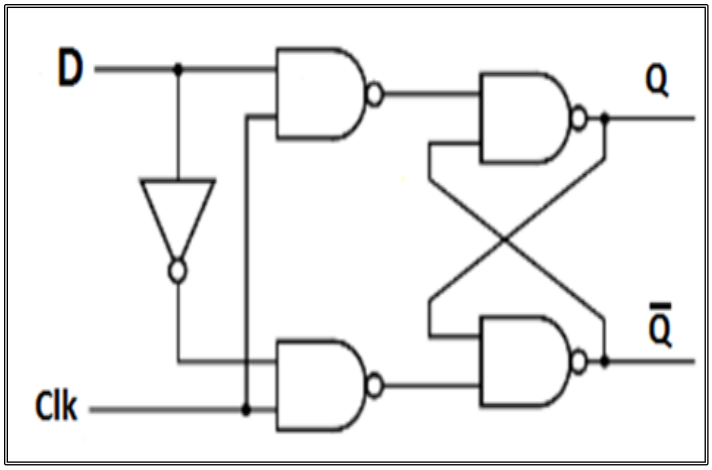


Fig 9.1: Logic diagram D Flip Flop

Table 9.1: Truth Table

Clock	D	Q
↑	0	0
↑	1	1
↓	X	Q

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.
2. From the menu bar, select **File** => **New Project**. In the **New Project Wizard** window type "DFlipFlop" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

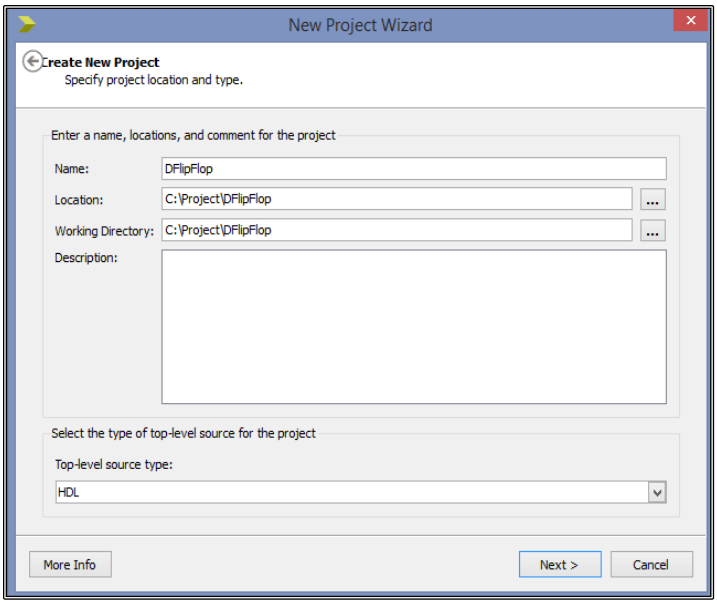


Fig 9.2: New Project

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

4. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

5. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "DFF" for the new source in the **File Name** field. Click **Next**.

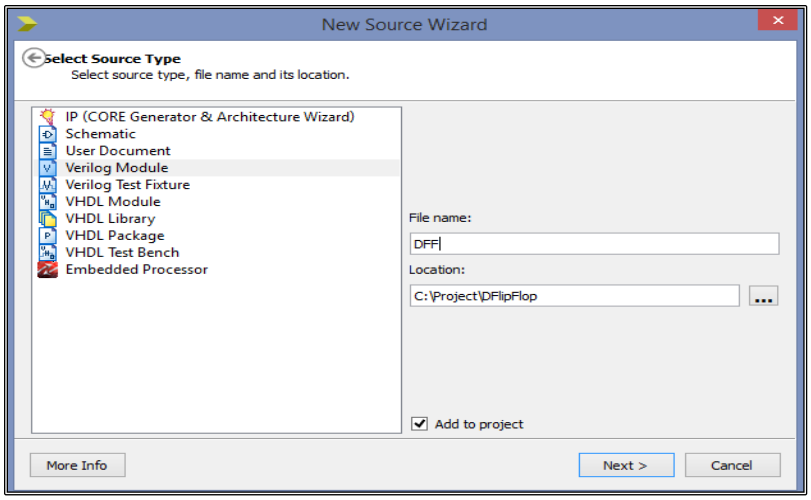


Fig 9.3: New Source

6. In the **Define Module** page, enter the port information for the "DFF" as follows:

   a. In the first six **Port Name** fields, enter 'q', 'q1', 'test', 'd', 'clk' and 'rst'.
   b. Set the **Direction field** to 'output' for 'q', 'q1' and 'test'. and to 'output' for 'seg'.
   c. Set the **Direction field** to 'input' for'd', 'clk' and 'rst'.
   Click **Next.**

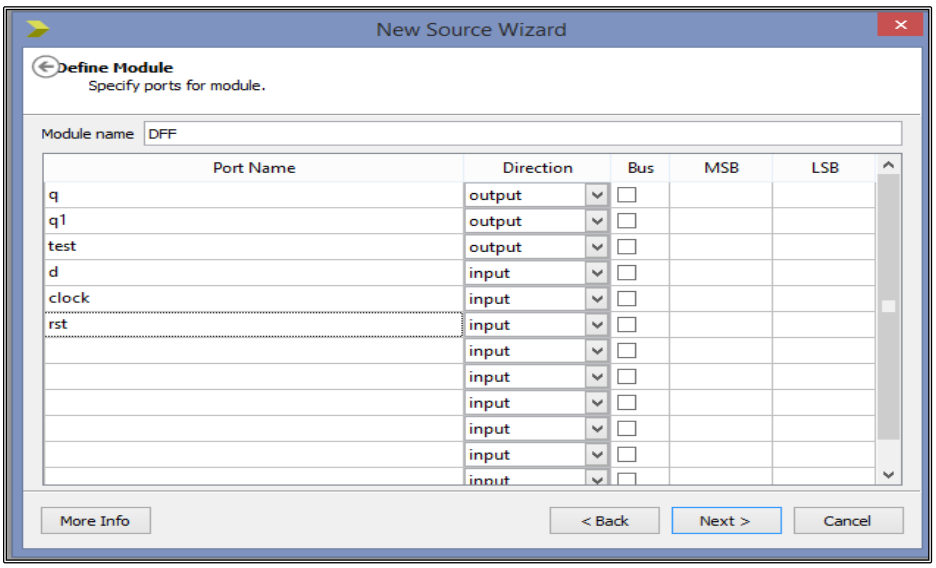


Fig 9.4: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the Xilinx ISE interface you can see that the new source file "DFF.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the DFF HDL file, and some of the basic file structure is already in place.

```
19  //
20  //////////////////////////////////////
21  module DFF(
22      output q,
23      output q1,
24      output test,
25      input d,
26      input clock,
27      input rst
28      );
29
30
31  endmodule
32
```

Fig 9.5: D Flip Flop module structure

9. Next, you have to write the desired D Flip Flop logic. For this example behavioral modeling is used. **Save** the file.

```
21  module DFF(
22      output q,
23      output q1,
24      output test,
25      input d,
26      input clock,
27      input rst
28      );
29
30  reg q=1'b0;
31  reg q1=1'b1;
32  reg [26:0]count;
33
34  always @(posedge clock or posedge rst)
35  begin
36  if(rst)
37  count<=0;
38  else
39  count<=count+1;
40  end
41  assign test = count[26];
42
43  always @(posedge test)
44  begin
45  q<=d;
46  q1<=~d;
47  end
48  endmodule
```

Fig 9.6: Complete D Flip Flop Module

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window.

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

12. For checking the functionality of the design, you have to apply test vectors and simulate the circuit. Right click on the design (DFF.v) and select **New Source**. In the New Source Wizard

window, select **Verilog Test Fixture** .Write the "DFFTest" in the **File name** field for the test bench. Click **Next**.

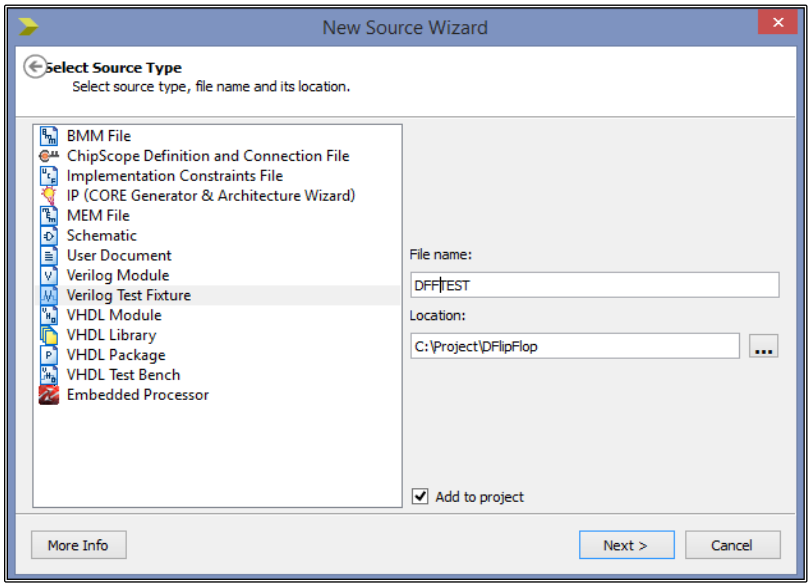


Fig 9.7: D Flip Flop testbench

13. Next, select the source file you want to associate with the testbench. Click **Next**. Check the summary and click **Finish**.

14. The ISE project navigator will generate template for testbench. The ISE tool detects the inputs/outputs from the source file and creates a template instantiating the original source module and provides initial values. After editing the test bench, click on **Behavioral Check Syntax** in the Processes window to check for syntax errors in the test bench.

15. If no errors are present double click on **Simulate Behavioral Model.** A new window showing the waveforms for the inputs and outputs of the design will be displayed.You can verify the outputs.

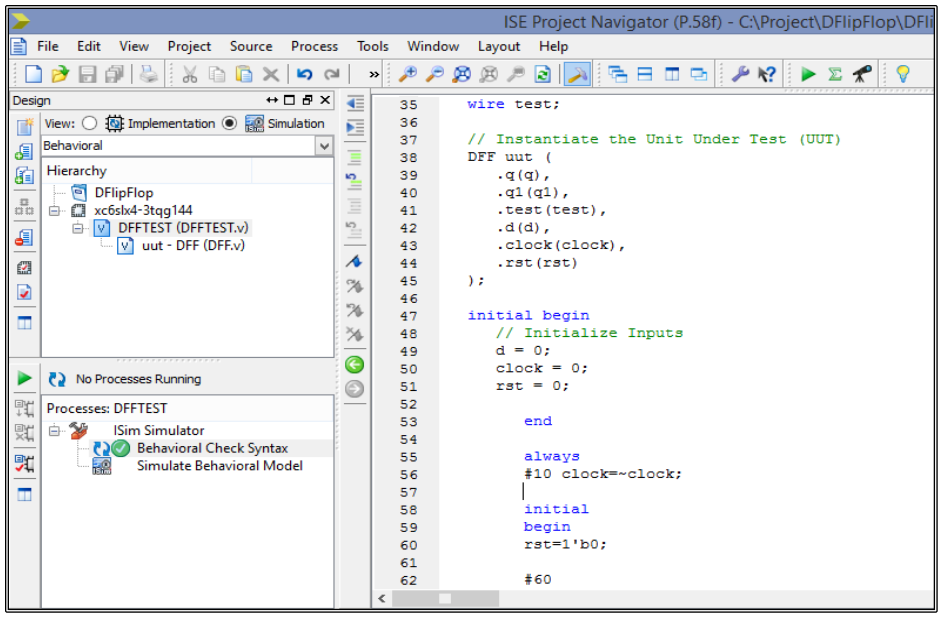


Fig 9.8: Complete Testbench D Flip Flop

16. For the implementation of the design, select **Implementation** Button in the Hierarchy window on the left pan.

17. For UCF File, right click on the design (DFF.v) and select **New Source**. New Source Wizard will open. Select source type as **Implementation Constraints File** and enter the file name as "DFFTEST".

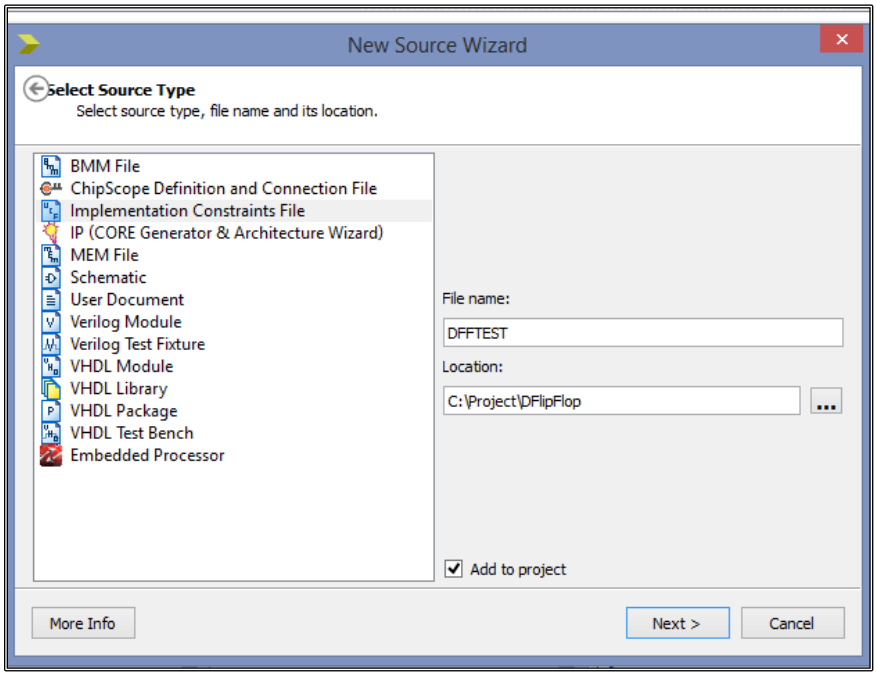


Fig 9.9: UCF creation

Make sure that Add to Project box is checked to add the constraints file to the project folder. Click **Next**. Check **Summary**. Click **Finish**.

18. A blank window will appear in a new tab having the design's name as "DFFTEST.ucf". Add the constraints for each input port and output port.

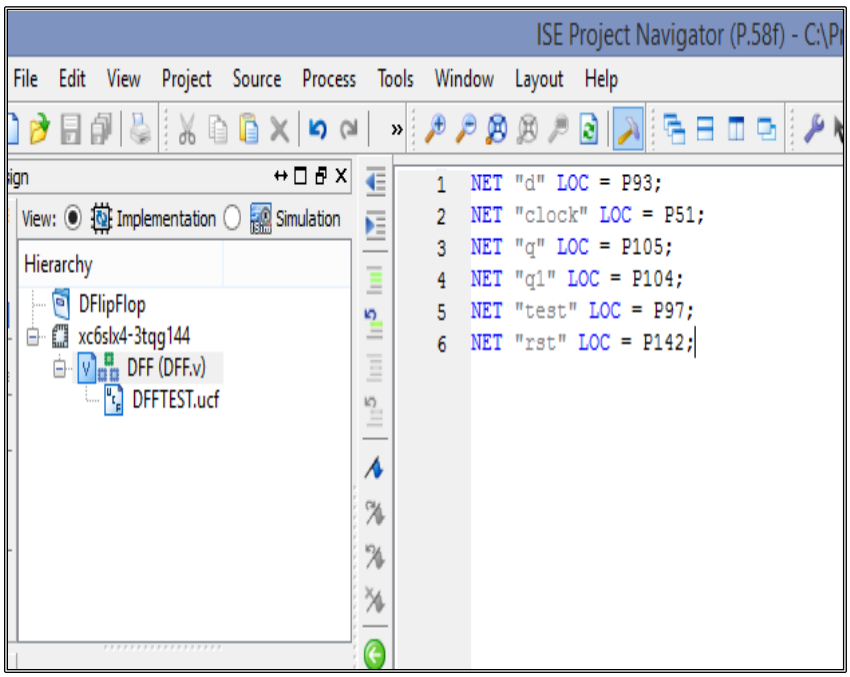


Fig 9.10: UCF declaration

19. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select **Run**. If the design is successfully syntheszied then a green check will appear near the Synthesize-XST and "**Process "Synthesize - XST" completed successfully**" is displayed in the console window.

20. Synthesize tool can generate two forms of schematic representation, in the form of **RTL View** and **Technology View** for the "DFF.v". Double click on the RTL schematic and Technology schematic for viewing the respective schematics.

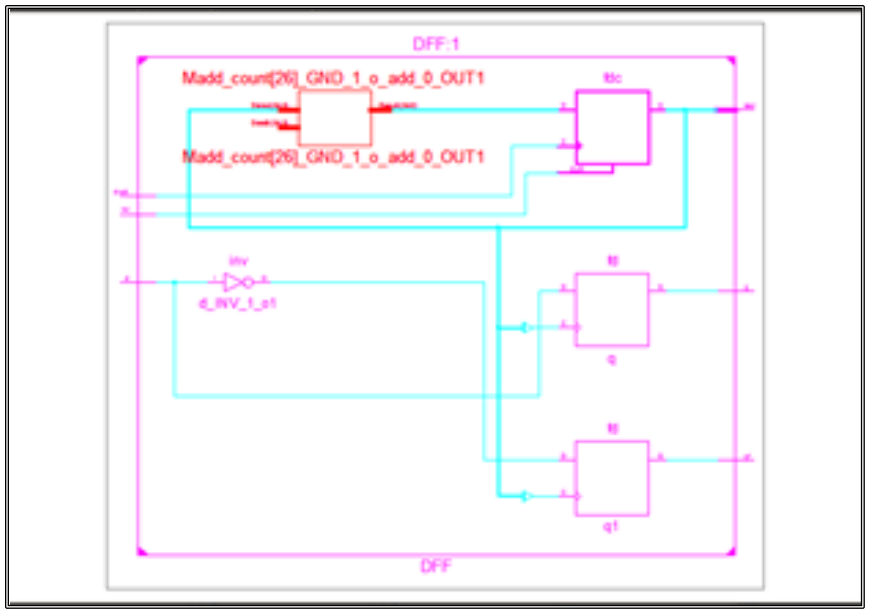


Fig 9.11: RTL View

21. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route.

22. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip.

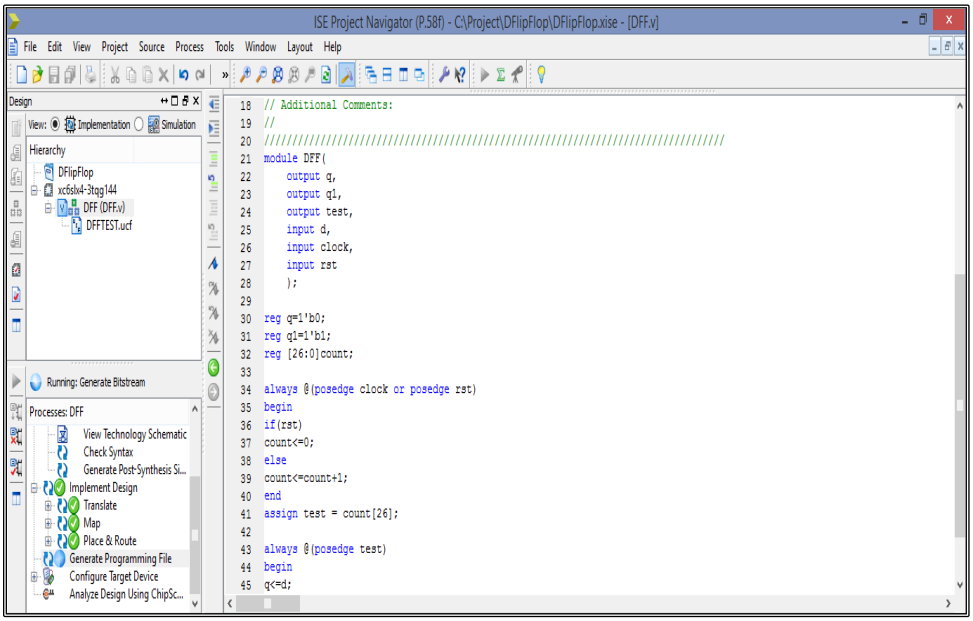


Fig 9.12: Bitstream generation

23. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the D Flip Flop circuit.

Verilog Code D Flip Flop

```
module DFF(  output q,output q1,output test,input d,input clock,input rst);

reg q=1'b0;
reg q1=1'b1;
reg [26:0]count;

always @(posedge clock or posedge rst)
begin
if(rst)
count<=0;
else
count<=count+1;
end
assign test = count[26];

always @(posedge test)
begin
q<=d;
q1<=~d;
end
endmodule
```

User Constraints File (UCF)

```
NET "d" LOC = “P93”;
NET "clock" LOC = “P51”;
NET "q" LOC = “P105”;
NET "q1" LOC = “P104”;
NET "test" LOC = “P97”;
NET "rst" LOC = “P142”;
```

EXPERIMENT: 10

Objective: To design and implement counter which counts from 0000 to FFFF on FPGA Board.

Software: Xilinx ISE Design Suite 14.5

Target Hardware: FPGA Board

Theory: Counter is a sequential circuit. A digital circuit which is used for counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied.

Counters are of two types.

1. Synchronous counters.
2. Asynchronous or ripple counters.

Procedure:

1. Start the **ISE 14.5 design suite**; double-click the Project Navigator icon on your desktop.

2. From the menu bar, select **File** => **New Project**. In the **New Project Wizard** window type "counter" in the **Name** field and in the **Location** field, browse C:\Project or choose the directory in which you want to store the project. Select **HDL** as the **Top-Level Source** Type and click **Next**.

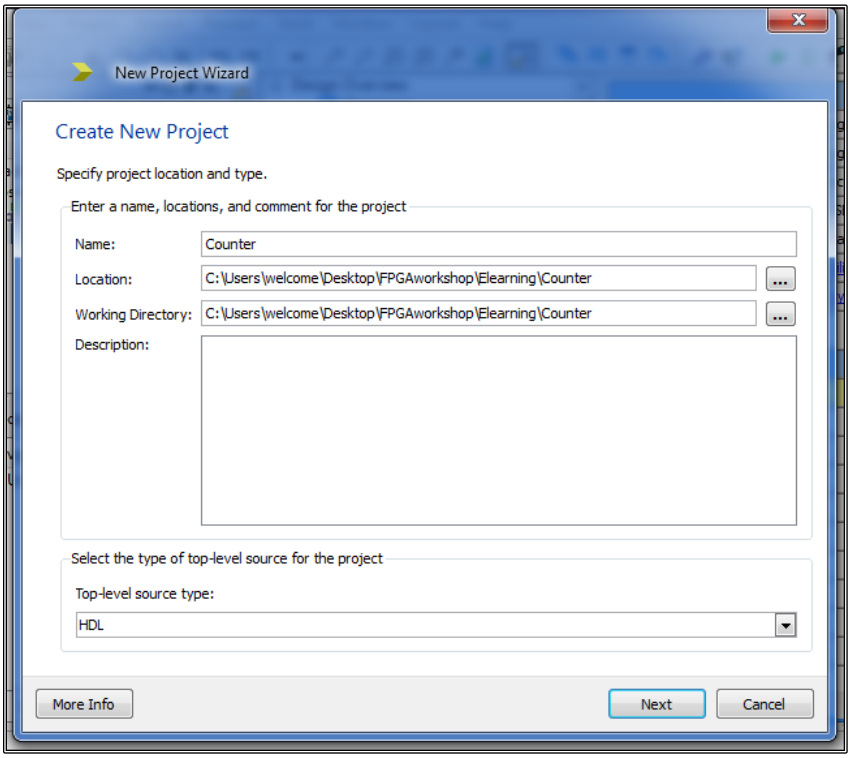


Fig 10.1: New project Counter

3. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **Package:** TQG144
   - **Speed:** -3
   - **Synthesis Tool**: XST (VHDL/Verilog)
   - **Simulator:** ISim (VHDL/Verilog)
   - **Preferred Language:** Verilog

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

3. The New Project Wizard—**Project Summary** page appears. Check the Project Summary and click **Finish**.

4. Now in the ISE Project Navigator Environment, select Project => **New Source**. Select **Verilog Module** as the **Source Type** and enter a name "COUNTER_SEVEN" for the new source in the **File Name** field. Click **Next**.

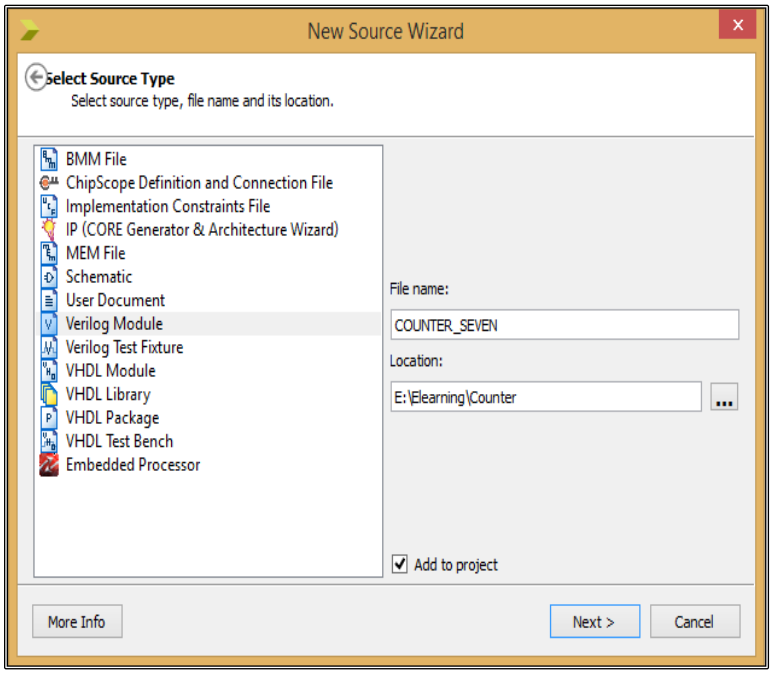


Fig 10.2: New Source COUNTER_SEVEN

6. In the **Define Module** page, enter the port information for the "COUNTER_SEVEN" as follows:

   a. In the first eleven **Port Name** fields, enter 'clock', 'reset', 'a', 'b', 'c', 'd', 'e', 'f', 'g' , 'dp' and 'an'.
   b. Set the **Direction field** to 'output' for 'a', 'b', 'c', 'd', 'e', 'f', 'g' , 'dp', 'an' and to 'input' for 'clock', 'reset'.
   c. Set the MSB bus range for output 'an' as 3 as 'an' is a 4-bit vector.
   Click **Next.**

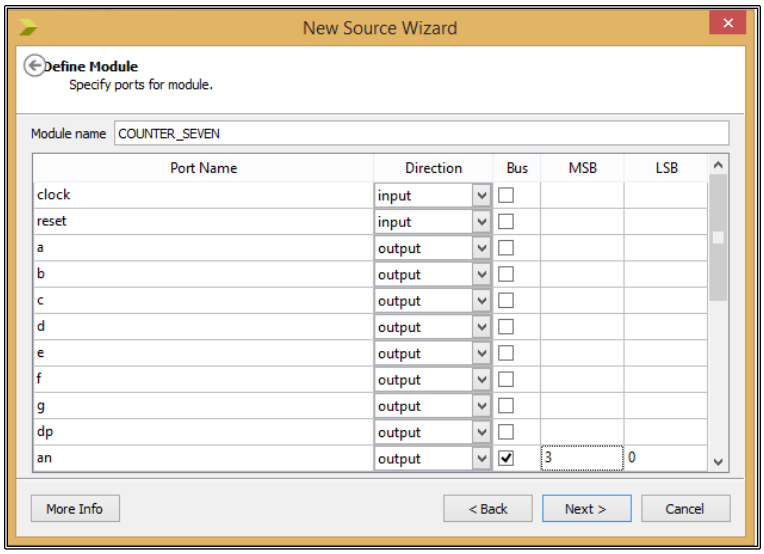


Fig 10.3: Module Definition

7. Next the summary description of the module will be displayed. Check the source **summary** and click **Finish**.

8. In the Xilinx ISE interface you can see that the new source file "COUNTER_SEVEN.v" has been added to the project (Hierarchy window). In the ISE Text Editor, the ports are already declared in the COUNTER_ SEVEN HDL file, and some of the basic file structure is already in place.

```
19  //
20  //////////////////////////////////////
21  module COUNTER_SEVEN(
22      input clock,
23      input reset,
24      output a,
25      output b,
26      output c,
27      output d,
28      output e,
29      output f,
30      output g,
31      output dp,
32      output [3:0] an
33      );
34
35
36  endmodule
37
```

Fig 10.4: Module structure

9. Next, you have to write the desired counter logic. For this example behavioral modeling is used. **Save** the file.

10. After writing the code next step is to check the syntax and functionality of the design. For this, select **Simulation** Button in the Hierarchy window.

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

12. For functional verification the test bench is created and simulation is performed as explained in the previous experiments.

13. For synthesis and implementation, select **Implementation Button** in the Hierarchy window on the left pan.

14. For UCF File right click on the design (COUNTER_SEVEN.v) and select **New Source**. New Source Wizard window will open. Select source type as **Implementation Constraints File** and enter the file name as "Counter1".

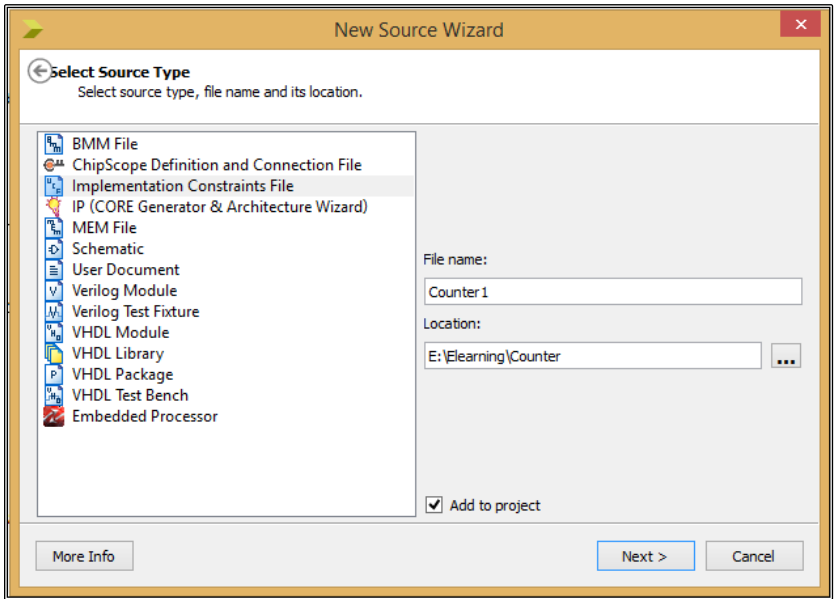


Fig 10.5: UCF creation

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

Check the summary. Click **Finish.**

15. A blank window will appear in a new tab having the design's name as "Counter1.ucf". Enter the constraints for each input and output port.

```
1   NET "an[0]" LOC = "P117";
2    NET "an[1]" LOC = "P118";
3    NET "an[2]" LOC = "P119";
4    NET "an[3]" LOC = "P120";
5    NET "a" LOC = "P140";
6    NET "b" LOC = "P137";
7    NET "c" LOC = "P134";
8    NET "clock" LOC = "P51";
9    NET "d" LOC = "P133";
10   NET "dp" LOC = "P121";
11   NET "e" LOC = "P131";
12   NET "f" LOC = "P127";
13   NET "g" LOC = "P126";
14   NET "reset" LOC = "P142";
```

Fig 10.6: UCF Declaration

16. Next we have to synthesize the design. For synthesis right click on the **Synthesize-XST** and select Run. If the design is successfully syntheszied then green check will appear near the Synthesize-XST.

17. Synthesize tool can generate two forms of schematic representation in the form of **RTL View** and **Technology View** of the counter HDL code. For schematic and technology view double click on the RTL schematic and Technology Schematic.

18. Next step is the implementation of the design.Right click **Implement Design** and click **Run**.After successfull implementataion a green check will appear on all the three steps i.e. translate, map and place & route (Fig 10.7).

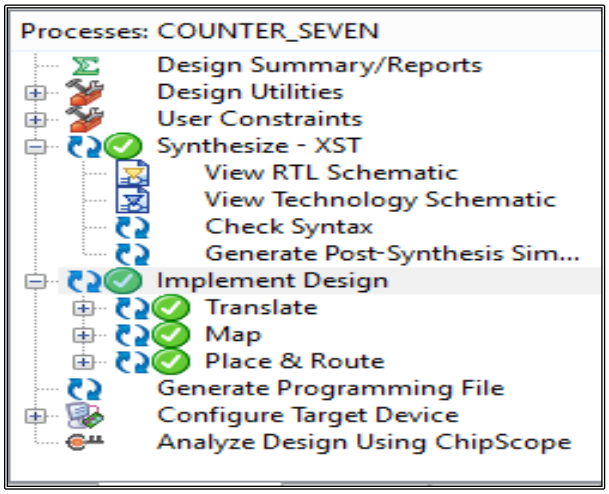


Fig 10.7: Design Implementation

19. Then we have to generate the programming file. For that, right click **Generate Programming file** and click **Run.** This will generate the bitstream file which will be downloaded to the chip.

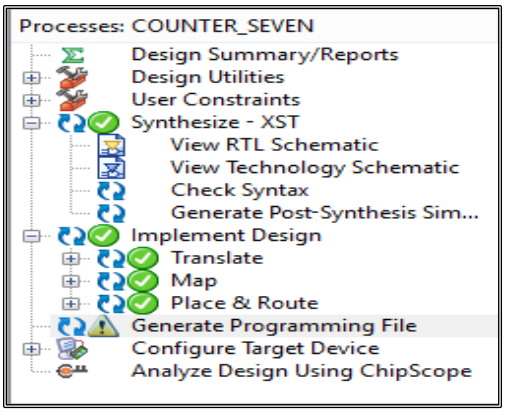


Fig 10.8: Bitstream Generation

20. Now for downloading the bitstream file on the FPGA board, follow the steps explained in the previous experiments. After programming the board, you can verify the counter circuit.

Verilog Code for counter

```
module COUNTER_SEVEN( input clock, input reset, output a, output b, output c, output d,
output e, output f, output g, output dp, output [3:0] an);

reg [3:0]first;
reg [3:0]second;
reg [3:0]third;
reg [3:0]fourth;

reg [26:0] delay;
wire test;
always @ (posedge clock or posedge reset)
 begin
 if (reset)
 delay <= 0;
 else
 delay <= delay + 1;
 end
 assign test = &delay;

always @ (posedge test or posedge reset)
begin
  if (reset) begin
          first <= 0;
          second <= 0;
          third<=0;
          fourth<=0;
 end
```

```
        else
        if ({first==4'd15 && second==4'd15 && third ==4'd15 && fourth==4'd15})
begin
first<=4'd15;
second<=4'd15;
third<=4'd15;
fourth<=4'd15;
end
else
begin
if (first==4'd15) begin
   first <= 0;
    if (second == 4'd15)

                    begin
                    second <= 0;
     if(third==4'd15)
                    begin
                    third<=0;
                    if(fourth==4'd15)
                    fourth<=0;
                    else
                    fourth<= fourth+1;
                    end
                    else
                    third<=third+1;
                    end
                    else
      second <= second + 1;

   end
   else
    first <= first + 1;
  end
  end
   localparam N = 16;
reg [N-1:0]count;
 always @ (posedge clock or posedge reset)
 begin
  if (reset)
   count <= 0;
  else
   count <= count + 1;
 end
 reg [6:0]sseg;
reg [3:0]an_temp;
```

```
always @ (*)
 begin
  case(count[N-1:N-2])
   2'b00 :
   begin
    sseg = first;
    an_temp = 4'b1110;
   end
   2'b01:
   begin
    sseg = second;
    an_temp = 4'b1101;
   end
    2'b10:
   begin
    sseg = third;
    an_temp = 4'b1011;
   end

  2'b11:
   begin
    sseg = fourth;
    an_temp = 4'b0111;
   end
  endcase
 end
assign an = an_temp;
 reg [6:0] sseg_temp;
always @ (*)
 begin
  case(sseg)
   4'd0 : sseg_temp = 7'b1000000; //0
   4'd1 : sseg_temp = 7'b1111001; //1
   4'd2 : sseg_temp = 7'b0100100; //2
   4'd3 : sseg_temp = 7'b0110000; //3
   4'd4 : sseg_temp = 7'b0011001; //4
   4'd5 : sseg_temp = 7'b0010010; //5
   4'd6 : sseg_temp = 7'b0000010; //6
   4'd7 : sseg_temp = 7'b1111000; //7
   4'd8 : sseg_temp = 7'b0000000; //8
   4'd9 : sseg_temp = 7'b0010000; //9
   4'd10 : sseg_temp = 7'b0001000; //9
   4'd11 : sseg_temp = 7'b0000000; //9
   4'd12 : sseg_temp = 7'b1000110; //9
           4'd13 : sseg_temp = 7'b1000000; //9
           4'd14: sseg_temp = 7'b0000110; //9
```

```
        4'd15: sseg_temp = 7'b0001110; //9
  default : sseg_temp = 7'b0111111; //dash
 endcase
end
assign {g, f, e, d, c, b, a} = sseg_temp;
assign dp = 1'b1;
endmodule
```

User Constraints File (UCF)

```
NET "an[0]" LOC = “P117”;
NET "an[1]" LOC = “P118”;
NET "an[2]" LOC = “P119”;
NET "an[3]" LOC = “P120”;
NET "a" LOC = “P140”;
NET "b" LOC = “P137”;
NET "c" LOC = “P134”;
NET "clock" LOC = “P51”;
NET "d" LOC = “P133”;
NET "dp" LOC = “P121”;
NET "e" LOC = “P131”;
NET "f" LOC = “P127”;
NET "g" LOC = “P126”;
NET "reset" LOC = “P142”;
```

Appendix A

Objective: Installation steps for FPGA programmer.

Step 1: Right click on the application setup.

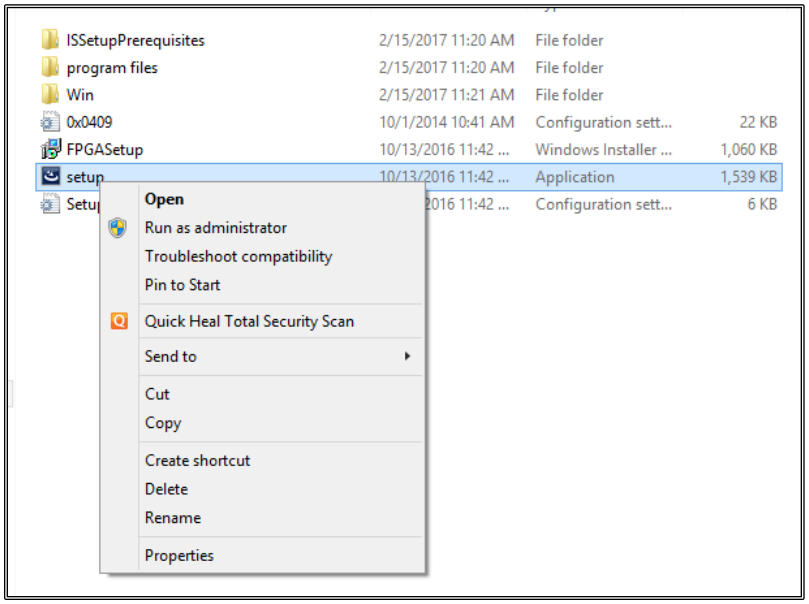


Fig A.1

Step 2: Click Open.

Step 3: It might ask you to install Microsoft Visual C++, if it is not installed on your system. Select **Install**.

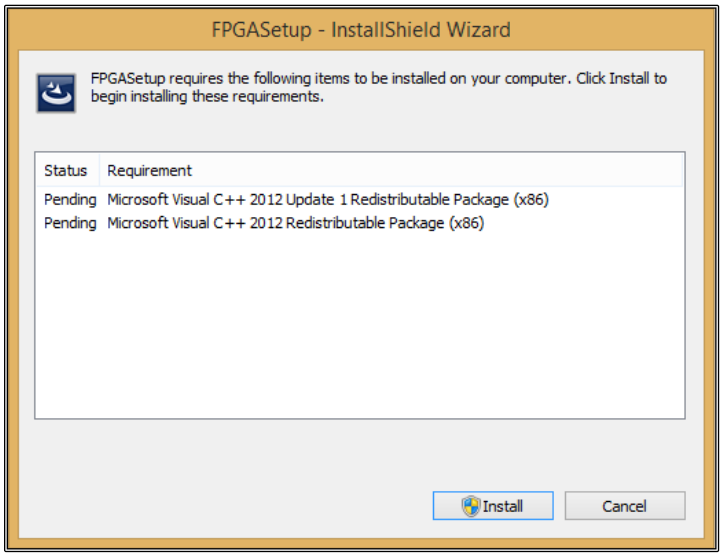


Fig A.2

Step 4: Click **Next** (Fig A.3).

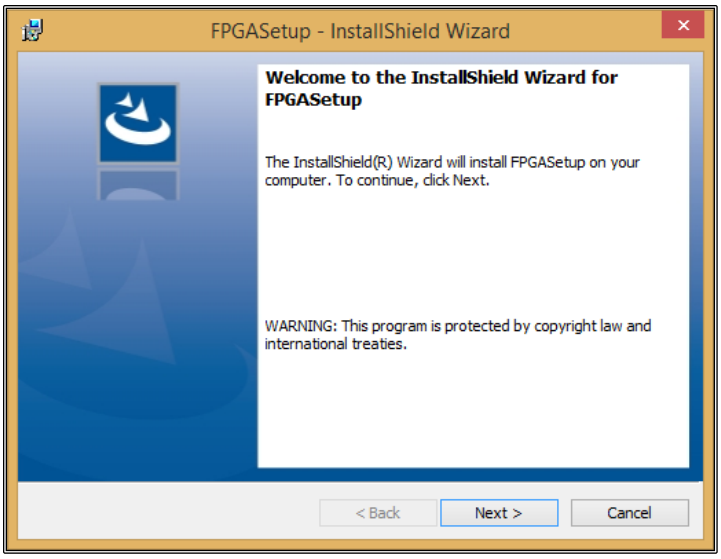


Fig A.3

Step 5: Accept the license agreement (Fig A.4). Click **Next**.

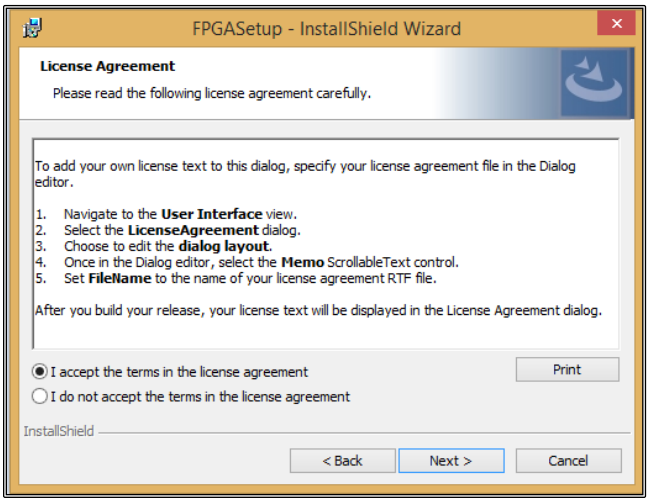


Fig A.4

Step 6: Fill you information i.e. Name and Organization Name (Fig A.5). Click **Next**.

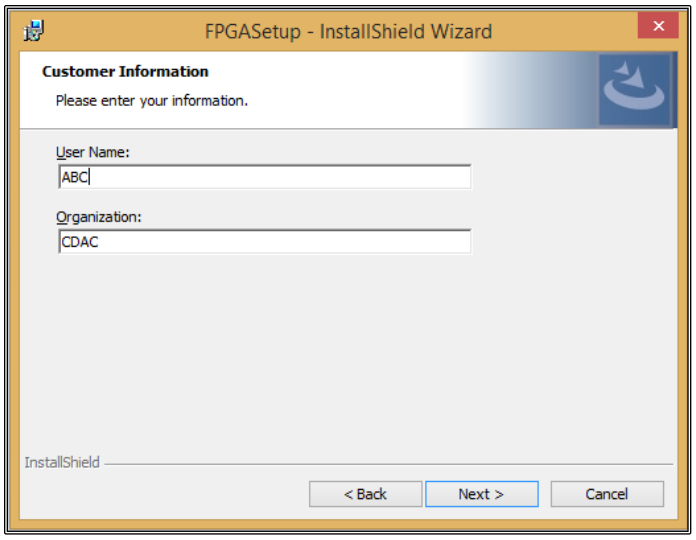


Fig A.5

Step 7: If you want to change any settings click **Back** otherwise click **Install**.

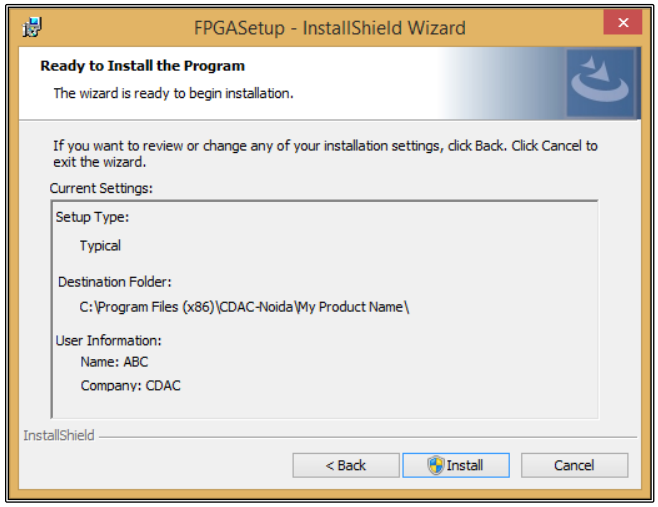


Fig A.6

Step 8: You might require to install the USB driver. **Silicon Labs USB driver installer** will install the driver. Click **Next**.

Fig A.7

Step 9: Accept the **license agreement** and click **Next**.

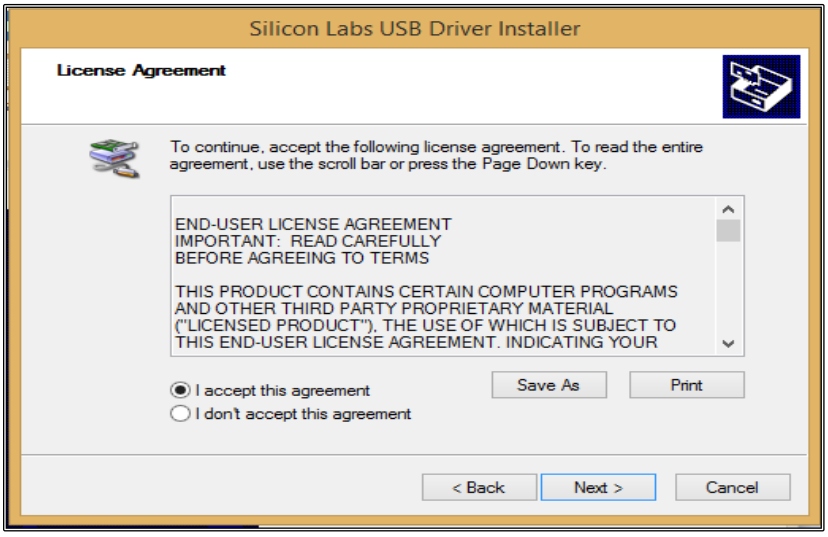


Fig A.9

Step 10: Click **Finish.**

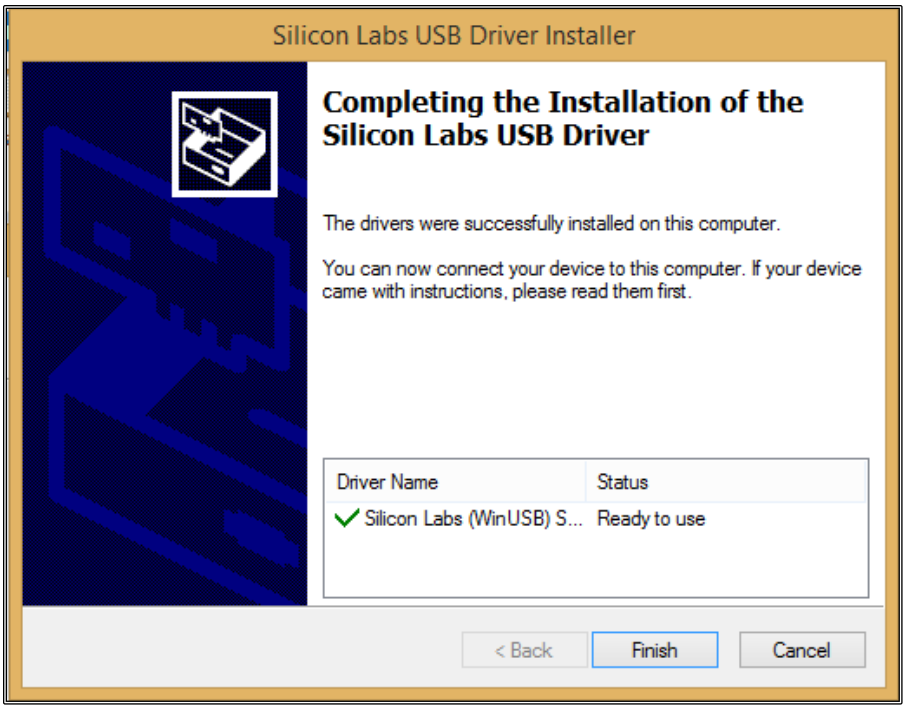


Fig A.10

Step 11: A window showing **InstallSheild Wizard Completed** will appear. Click **Finish**.

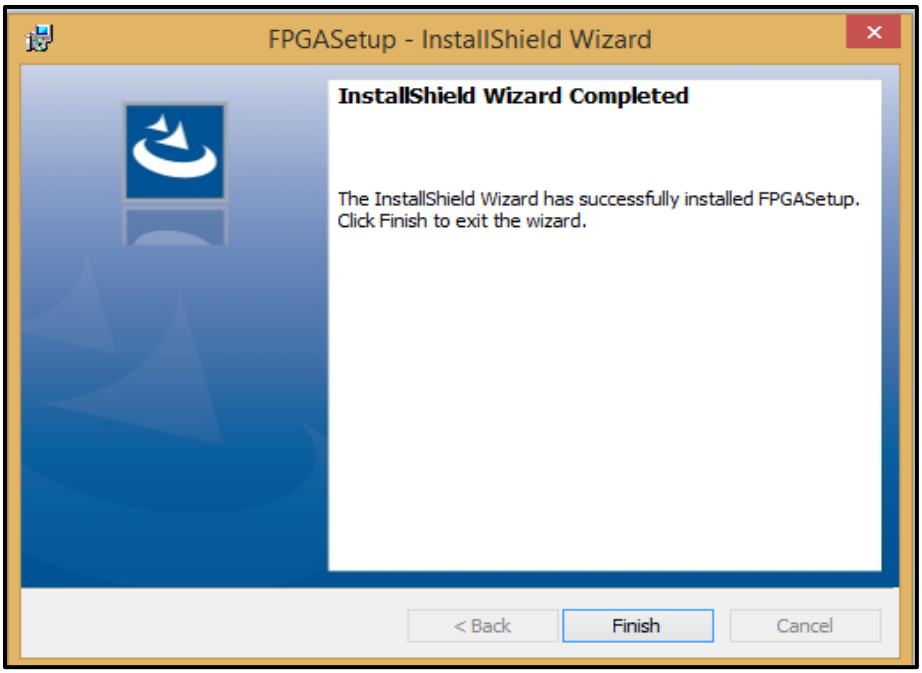


Fig A.11

Step 12: Icon as shown in Fig A.12 will appear on your desktop after complete installation of the programmer.

Fig A.12

Appendix B

Complete CDAC FPGA Board UCF

 # clock pins

```
NET "clk" LOC = "P51"; # Bank = 0, Signal name = FPGA_clk_25
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

 # Pin assignment for seven segment

```
NET "seg<0>" LOC = "P140"; # Bank = 1, Signal name = CA
NET "seg<1>" LOC = "P137"; # Bank = 1, Signal name = CB
NET "seg<2>" LOC = "P134"; # Bank = 1, Signal name = CC
NET "seg<3>" LOC = "P133"; # Bank = 2, Signal name = CD
NET "seg<4>" LOC = "P131"; # Bank = 2, Signal name = CE
NET "seg<5>" LOC = "P127"; # Bank = 1, Signal name = CF
NET "seg<6>" LOC = "P126"; # Bank = 1, Signal name = CG
NET "dp" LOC = "P121";          # Bank = 1, Signal name = DP
NET "anode<0>" LOC = "P120"; # Bank = 1, Signal name = AN0
NET "anode<1>" LOC = "P119"; # Bank = 1, Signal name = AN1
NET "anode<2>" LOC = "P118"; # Bank = 1, Signal name = AN2
NET "anode<3>" LOC = "P117"; # Bank = 1, Signal name = AN3
```

 # Pin assignment for LEDs

```
NET "led<7>" LOC = "P97" ; # Bank = 3, Signal name = LD7
NET "led<6>" LOC = "P98" ; # Bank = 2, Signal name = LD6
NET "led<5>" LOC = "P99" ;  # Bank = 2, Signal name = LD5
NET "led<4>" LOC = "P100" ;  # Bank = 2, Signal name = LD4
NET "led<3>" LOC = "P101" ; # Bank = 2, Signal name = LD3
NET "led<2>" LOC = "P102" ; # Bank = 3, Signal name = LD2
NET "led<1>" LOC = "P104" ; # Bank = 2, Signal name = LD1
NET "led<0>" LOC = "P105" ;  # Bank = 2, Signal name = LD0
```

 # Pin assignment for SWs

```
NET "sw<14>" LOC = "P78";  # Bank = 2, Signal name = SW14
NET "sw<13>" LOC = "P79";  # Bank = 2, Signal name = SW13
NET "sw<12>" LOC = "P80";  # Bank = 2, Signal name = SW12
NET "sw<11>" LOC = "P81";  # Bank = 2, Signal name = SW11
NET "sw<10>" LOC = "P82";  # Bank = 2, Signal name = SW10
NET "sw<9>" LOC = "P83";  # Bank = 3, Signal name = SW9
NET "sw<8>" LOC = "P84";  # Bank = 3, Signal name = SW8
NET "sw<7>" LOC = "P85";  # Bank = 3, Signal name = SW7
NET "sw<6>" LOC = "P87";  # Bank = 3, Signal name = SW6
NET "sw<5>" LOC = "P88";  # Bank = 3, Signal name = SW5
NET "sw<4>" LOC = "P92";  # Bank = 3, Signal name = SW4
NET "sw<3>" LOC = "P93";  # Bank = 2, Signal name = SW3
```

 # Pin assignment for push buttons

```
NET "btn<0>" LOC = "P139";  # Bank = 1, Signal name = BTN0
NET "btn<1>" LOC = "P143";  # Bank = 0, Signal name = BTN1
NET "btn<2>" LOC = "P142"; # Bank = 2, Signal name = BTN2
NET "btn<3>" LOC = "P141"; # Bank = 0, Signal name = BTN3
```

 # Pin assignment for VGA

```
NET "hsync" LOC = "P47"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = HSYNC
NET "vsync" LOC = "P46"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = VSYNC

NET "OutRed<2>"  LOC = "P59"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED2
NET "OutRed<1>"  LOC = "P61"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED1
NET "OutRed<0>"  LOC = "P62"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED0
NET "OutGreen<2>"  LOC = "P56"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN2
NET "OutGreen<1>"  LOC = "P57"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN1
NET "OutGreen<0>"  LOC = "P58"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN0
NET "OutBlue<2>"  LOC = "P48"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU1
NET "OutBlue<1>"  LOC = "P50"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU0
```

Appendix C

Creating new project in ISE Design Suite

Objective: Get familiar with creating a new project in ISE design suite

1. To start the **Xilinx ISE design Suite 14.5**, double-click on the **Project Navigator** icon on your desktop, or select Start => All Programs => Xilinx ISE Design Suite => Xilinx ISE Design Suite 14.5 => ISE =>Design Tools => Project Navigator.

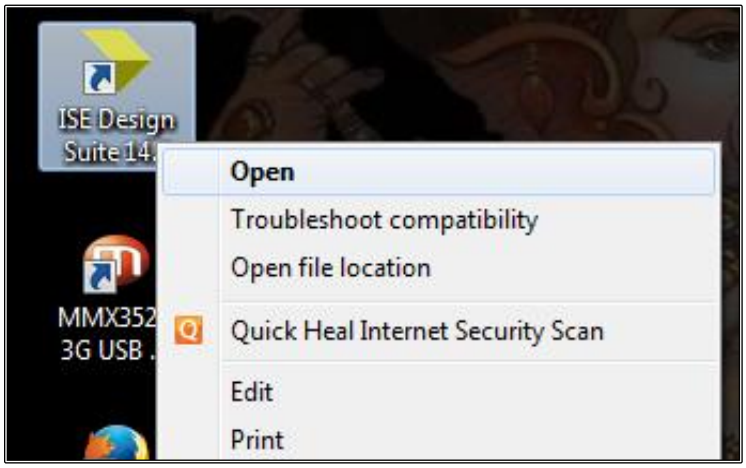


Fig C.1

2. The **ISE Project navigator** interface will be opened (Fig C.2). From the menu bar, select **File => New Project.**

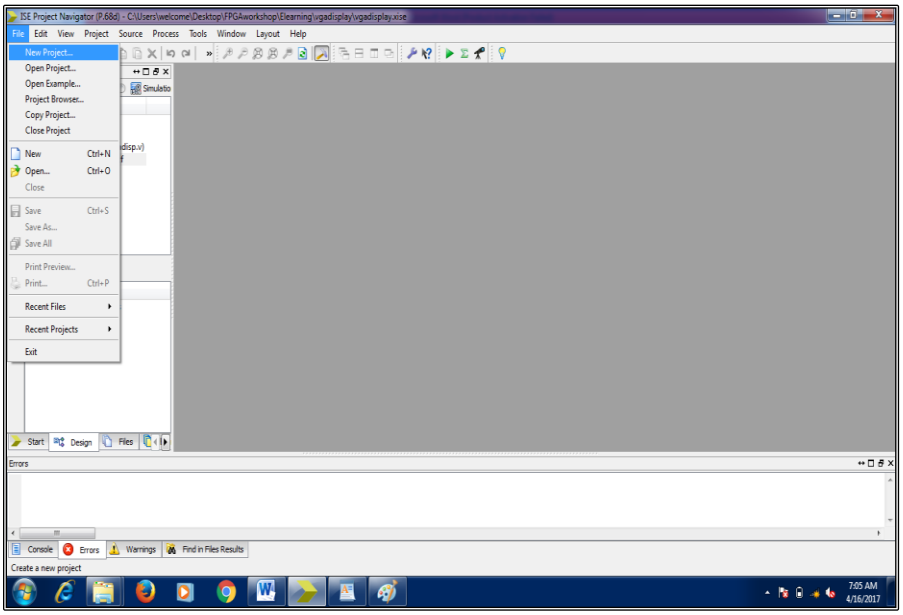


Fig C.2

3. The **New Project Wizard** window will be displayed. In the **Name** field type “example” (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 C:\Project\OR_Logic (Fig.C.3)

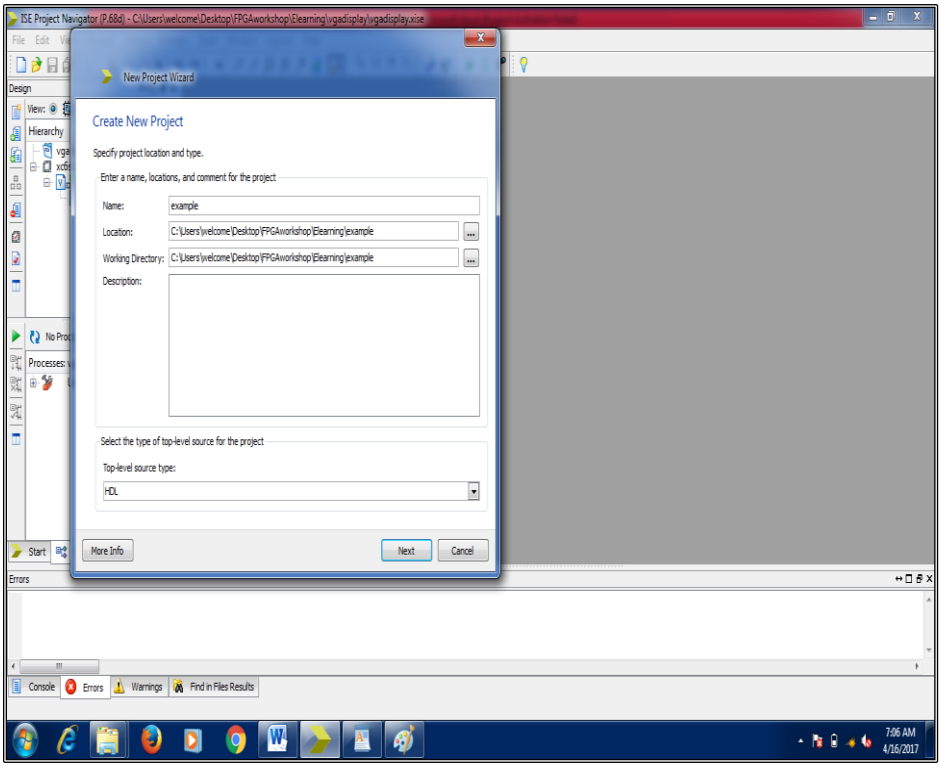


Fig C.3

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

4. Next in the **New Project Wizard - Project Settings** page, select the following to specify project and device properties:
   - **Product Category:** All
   - **Family:** Spartan6
   - **Device:** XC6SLX4
   - **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**.

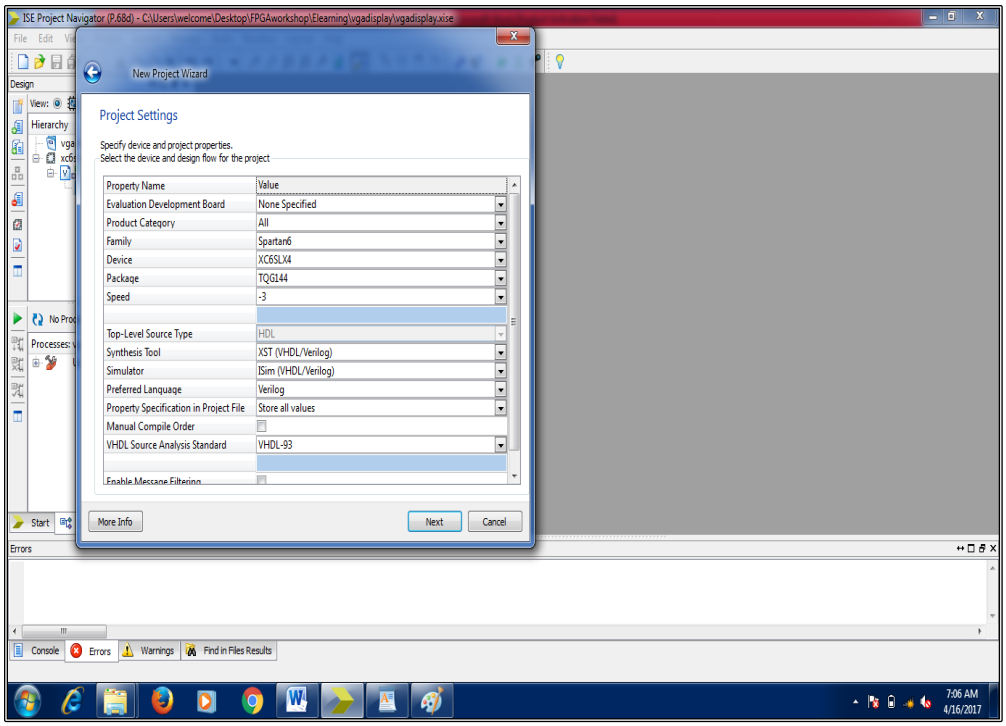


Fig C.4

5. The **New Project Wizard—Project Summary** page appears. Check the Project Summary and click **Finish**.

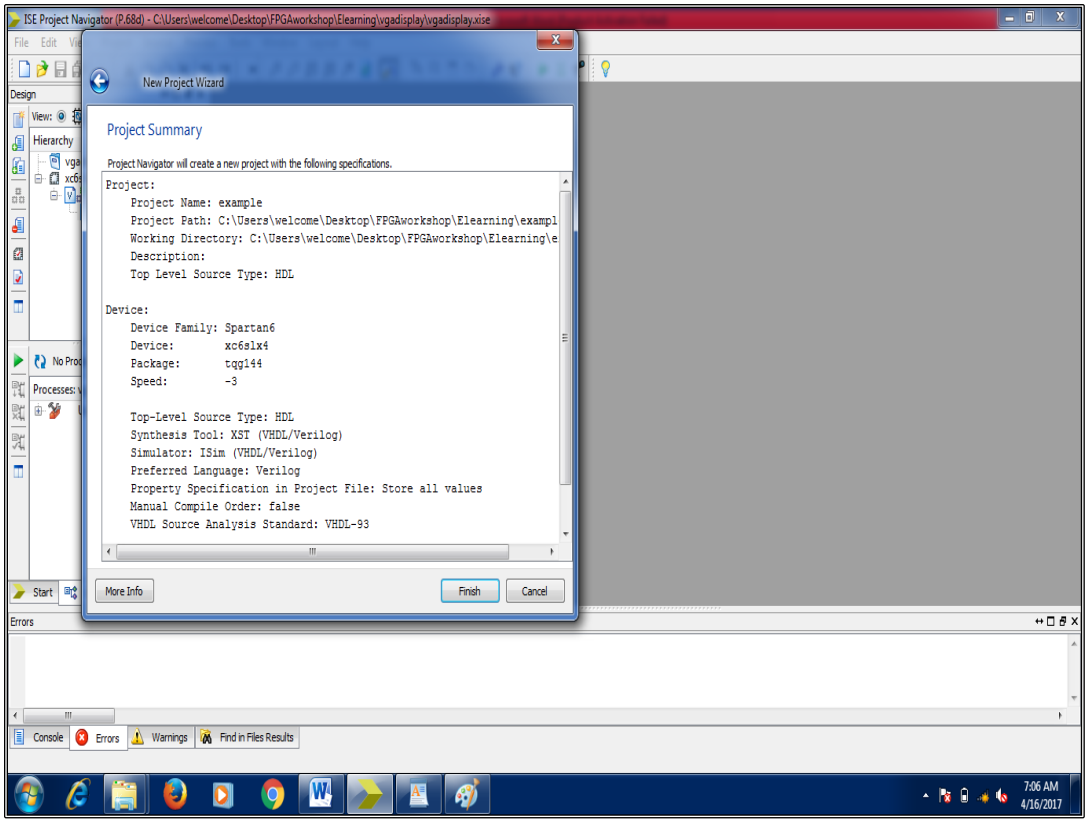


Fig C.5

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

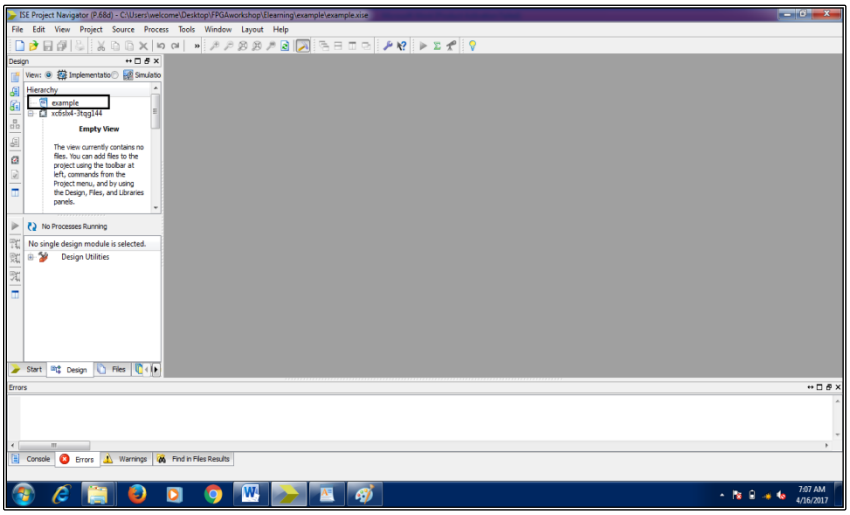


Fig C.6

Appendix D

Objective: Adding New Source to the project

1. To create new source file, select **Project => New Source**.

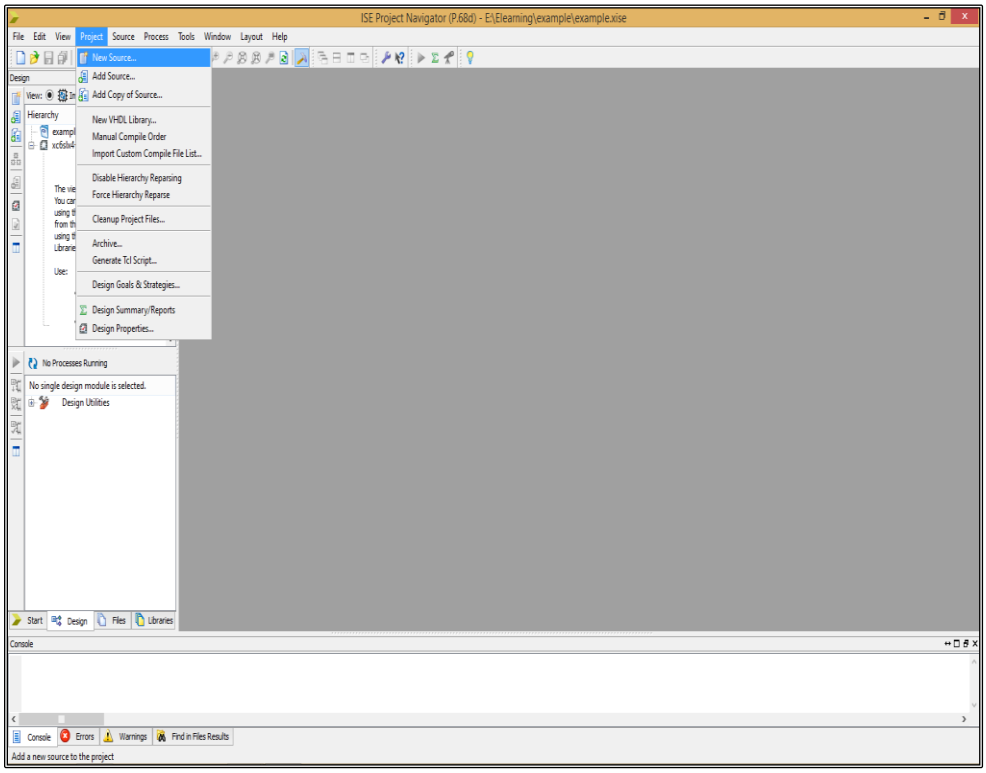


Fig D.1

2. The **New Source Wizard** window will open in which you have to specify the type of source you want to create.

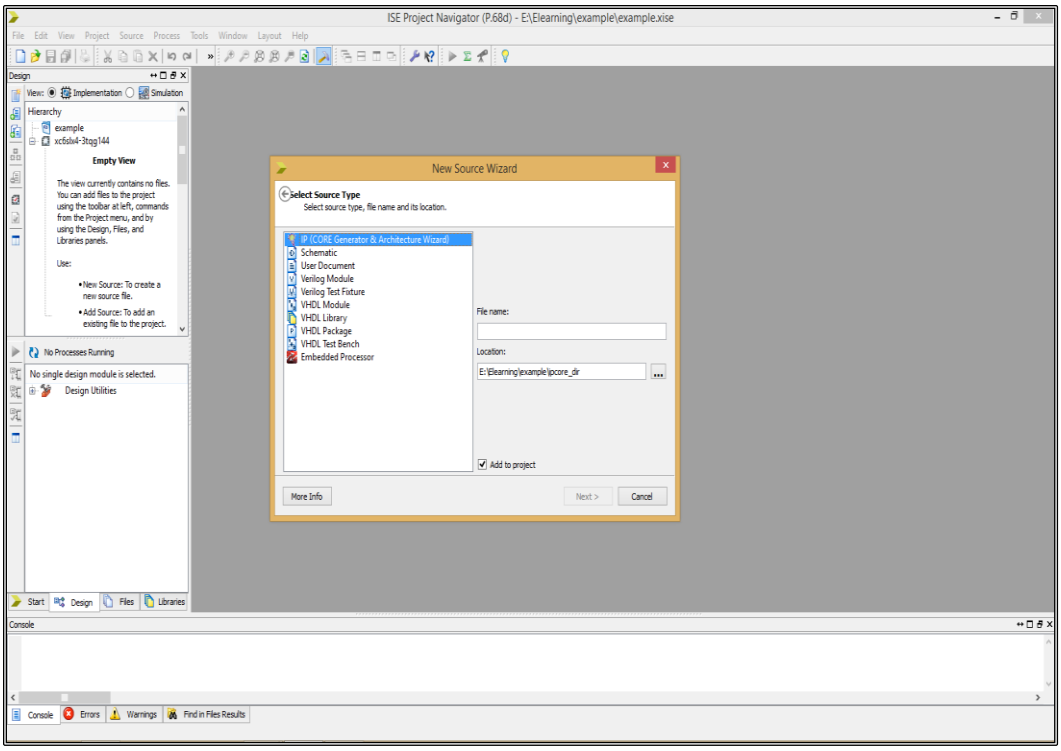


Fig D.2

For this, select the sources which you require are Verilog Module and Verilog Test Fixture.

3. You have to specify the name of the particular source file and the location where you want to save the source file.