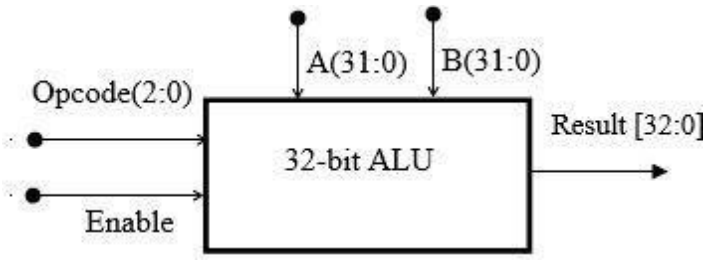
BASICS OF –VHDL LAB		Semester	IV
Course Code	**BEE456A**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	03
Examination nature (SEE)	Practical/Viva-Voce		

Course objectives:

1. Along with prescribed hours of teaching –learning process, provide opportunity to perform the experiments/programmes at their own time, at their own pace, at any place as per their convenience and repeat any number of times to understand the concept.
2. Provide unhindered access to perform whenever the students wish.
3. Vary different parameters to study the behaviour of the circuit without the risk of damaging equipment/device or injuring themselves.

Sl.NO	Experiments
	PART B **Note:** Programming can be done using any compiler. Download the programs on a FPGA/CPLD board and performance testing may be done using 32 channel pattern generator and logic analyser, apart from verification by simulation with tools such as Altera/Modelsim or equivalent
1	Write Verilog program for the following combinational design along with test bench to verify the design: a) 2 to 4 decoder realization using NAND gates only (structural model) b) 8 to 3 encoder with priority encoder and without priority encoder (behavioral model) c) 8 to 1 Multiplexer using case statement and if statement d) 4 bit binary to gray code converter using 1 bit gray to binary converter 1 bit adder and subtractor.
2	Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modelled behavior.
3	Verilog 32 bit ALU shown in figure below and verify the functionality of ALU by selecting appropriate test patterns. The functionality of the ALU is shown in Table-1. a) Write test bench to verify the functionality of the ALU considering all possible input patterns b) The enable signal will set the output to required functions if enabled, if disabled all the outputs are set to tri-state. c) The acknowledge signal is set high after every operation is complete. 

<table>
<tr><td></td><td>ALU Top Level Diagram
Table -1 ALU functions:
<table>
<tr><th>Opcode (2:0)</th><th>ALU Operation</th><th colspan="2">Remarks</th></tr>
<tr><td>000</td><td>A + B</td><td>Addition of two numbers</td><td rowspan="2">Both A and B are in two's complement format</td></tr>
<tr><td>001</td><td>A – B</td><td>Subtraction of two numbers</td></tr>
<tr><td>010</td><td>A + 1</td><td>Increment Accumulator by 1</td><td rowspan="2">A is in two's complement format</td></tr>
<tr><td>011</td><td>A - 1</td><td>Decrement accumulator by 1</td></tr>
<tr><td>100</td><td>A</td><td>True</td><td rowspan="4">Inputs can be in any format</td></tr>
<tr><td>101</td><td>A Complement</td><td>Complement</td></tr>
<tr><td>110</td><td>A OR B</td><td>Logical OR</td></tr>
<tr><td>111</td><td>A AND B</td><td>Logical AND</td></tr>
</table></td></tr>
<tr><td>4</td><td>Write Verilog code for SR, D and JK and verify the flip flop</td></tr>
<tr><td>5</td><td>Write Verilog code for 4 bit BCD synchronous counter</td></tr>
<tr><td>6</td><td>Write Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16 . Verify the functionality of the code.</td></tr>
<tr><td></td><td>PART B
Note; Interfacing and Debugging: (ED) WinXp, PSpice, MultiSim, Proteus, Circuit Lab, or any other equivalent tool canbe used.</td></tr>
<tr><td></td><td>Demonstration Experiments (For CIE)</td></tr>
<tr><td>7</td><td>Write a Verilog code to design a clock divider circuit that generates ½, 1/3rd, 1/4th, clock from given input clock. Port the design to FPGA and validate the functionality through CRO.</td></tr>
<tr><td>8</td><td>Interface a DC motor to FPGA and write Verilog code to change its speed and direction</td></tr>
<tr><td>9</td><td>Interface a stepper motor to FPGA and write Verilog code to control the stepper motor rotationwhich in turn may control a Robatic arm. External switches to be used for different controls like rotate the stepper motor:
a) + N steps if the switch number 1 of a DIP switch is closed.
b) +N/2 steps if switch number 2 of a DIP switch is closed. c)-
N steps if switch number 3 of a DIP switch is closed etc.</td></tr>
<tr><td>10</td><td>Interface a DAC to FPGA and write Verilog code to generate a sine wave of frequency f KHz, ex f=100 KHz, or 200 KHz etc, .Modify the code to down sample the frequency to f/2 KHz. Display the original and down sampled signals by connecting them to CRO.</td></tr>
<tr><td>11</td><td>Write Verilog code using FSM to simulate elevator operation.</td></tr>
<tr><td>12</td><td>Write Verilog code to convert an analog input signal of a sensor to digital form and to displaythe same on a suitable display like simple set of LEDs , 7 segment display digits or LCD display</td></tr>
</table>

Introduction to HDL

Hardware description language (HDL) is a computer aided design (CAD) tool for the modern design and synthesis of digital systems. The recent, steady advances in semiconductor technology continue to increase the power and complexity of digital systems. Due to their complexity, such systems cannot be realized using discrete integrated circuits. They are usually realized using high density, programmable chips, such as application specific Integrated circuits (ASICs) and Field programmable gate arrays (FPGAs) and require sophisticated CAD tools. HDL is an integral part of such tools. HDL offers the designer a very efficient tool for implementing and synthesizing designs on chips.

The two widely used hardware description languages are VHDL and Verilog. These languages provide support for modeling the system hierarchically and also supports top down and bottom up design methodologies. The system and its subsystems can be described at any level of abstraction ranging from the architecture level to the gate level.

The complex constructs and features of these languages are enough to be able to model designs with high degrees of complexity.

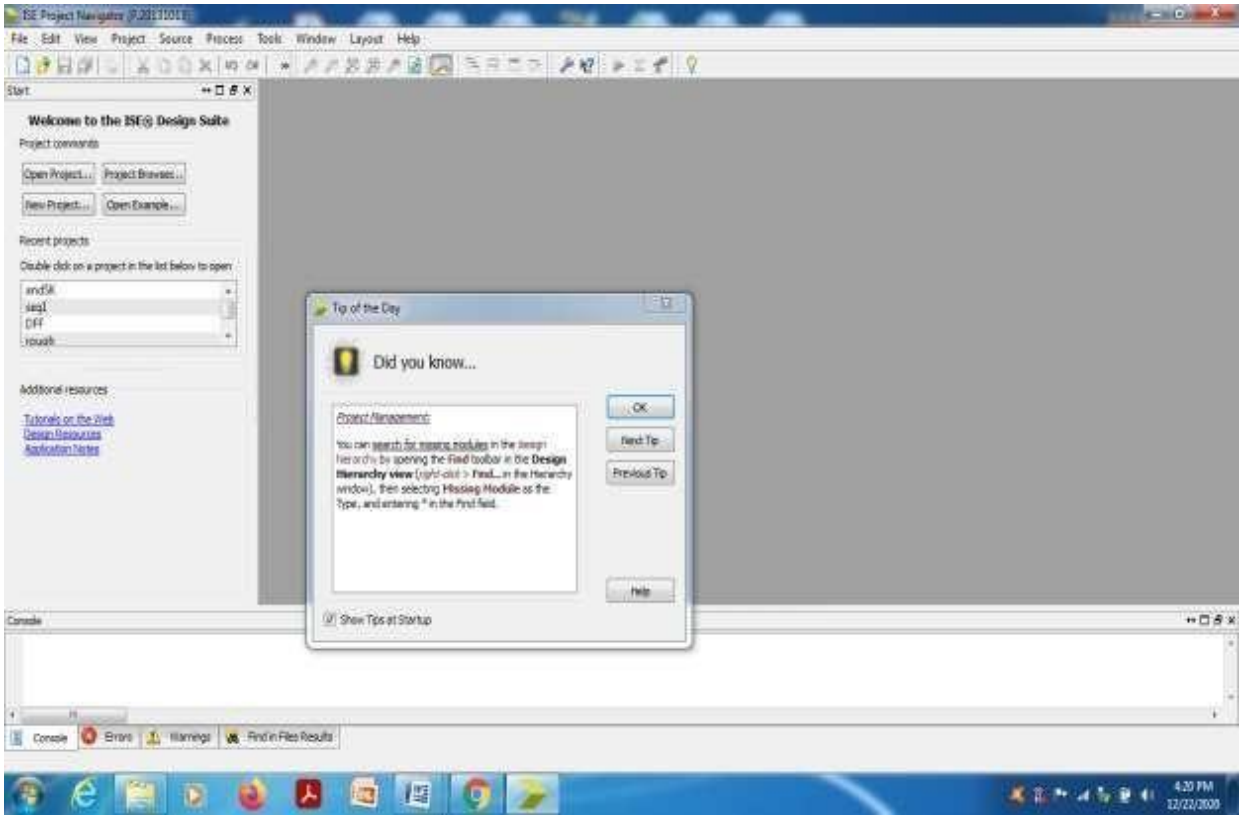
Software Required: Xilinx ISE14.7
Hardware Used: XC9572 CPLD

PROCEDURE

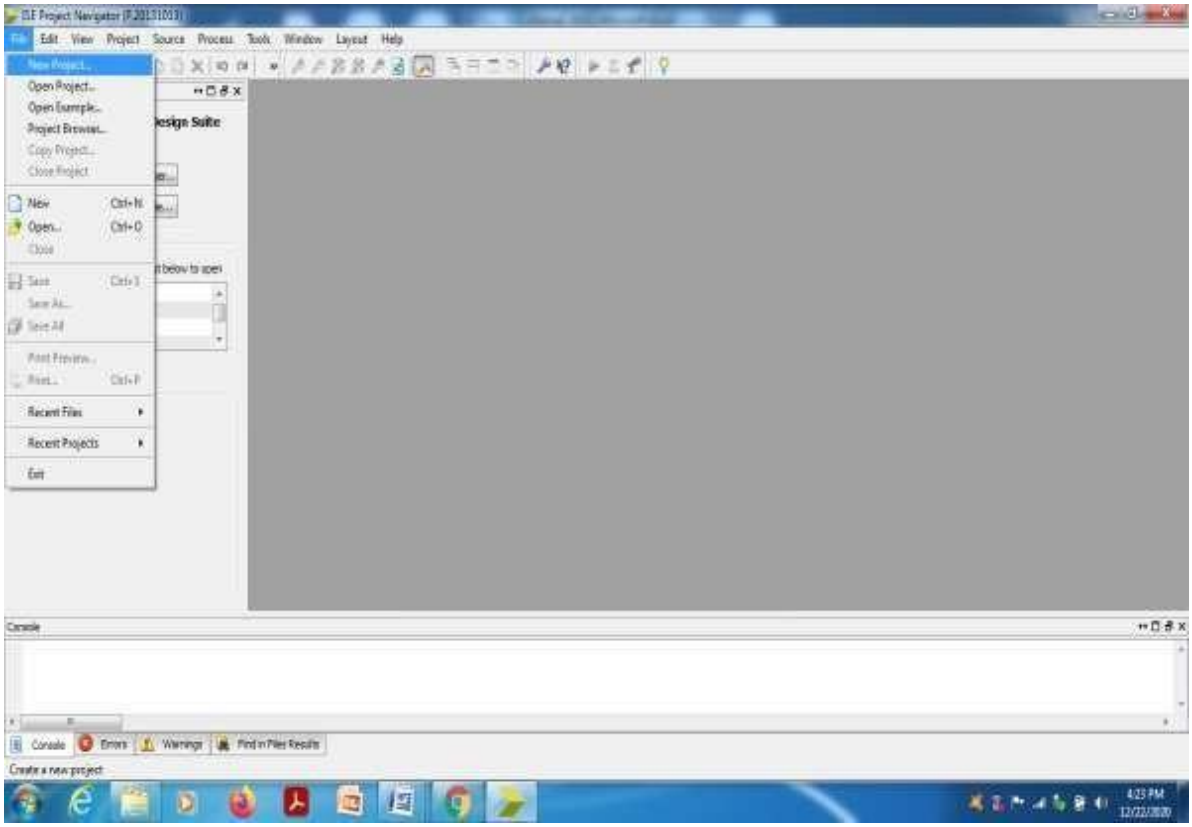
Procedure to work with Xilinx ISE 14.7 software:

1. To Create a Project:
   A project in ISE is a collection of all files necessary to create and download a design to the selected device.

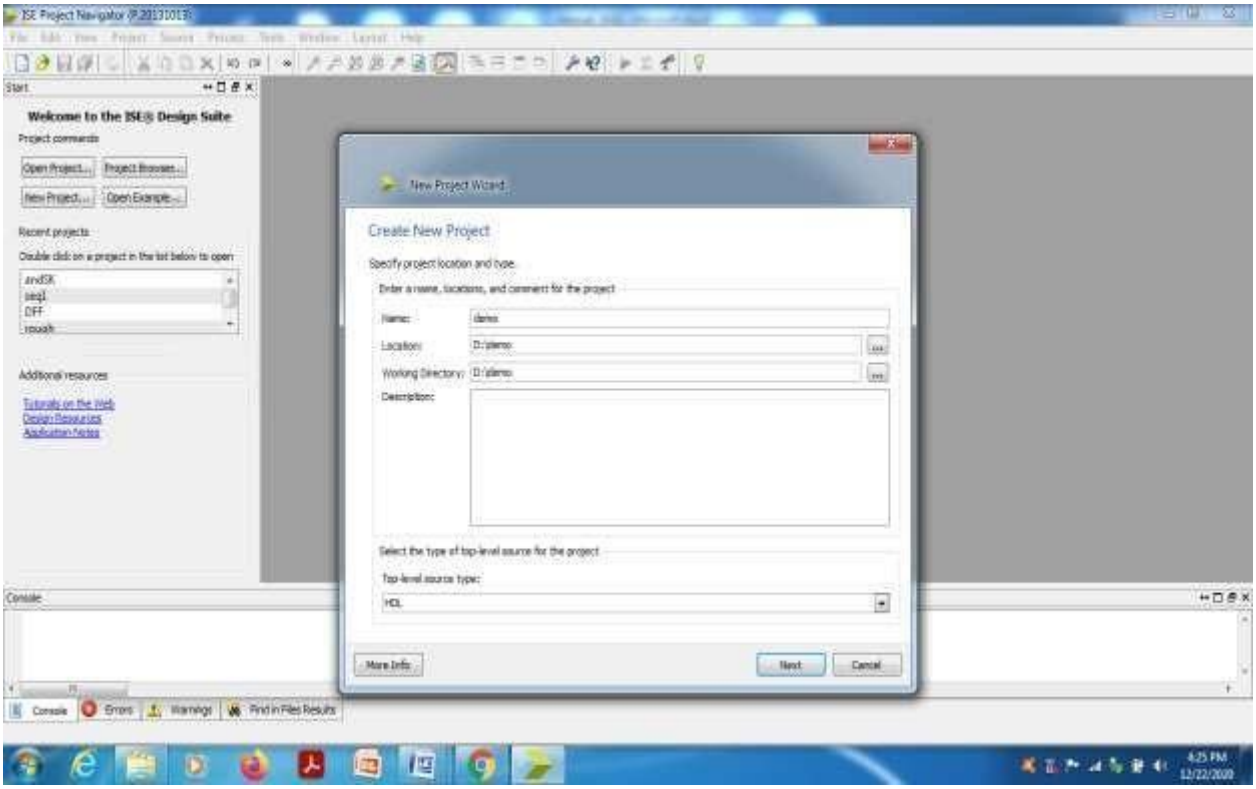
   Open XilinxISE Window.



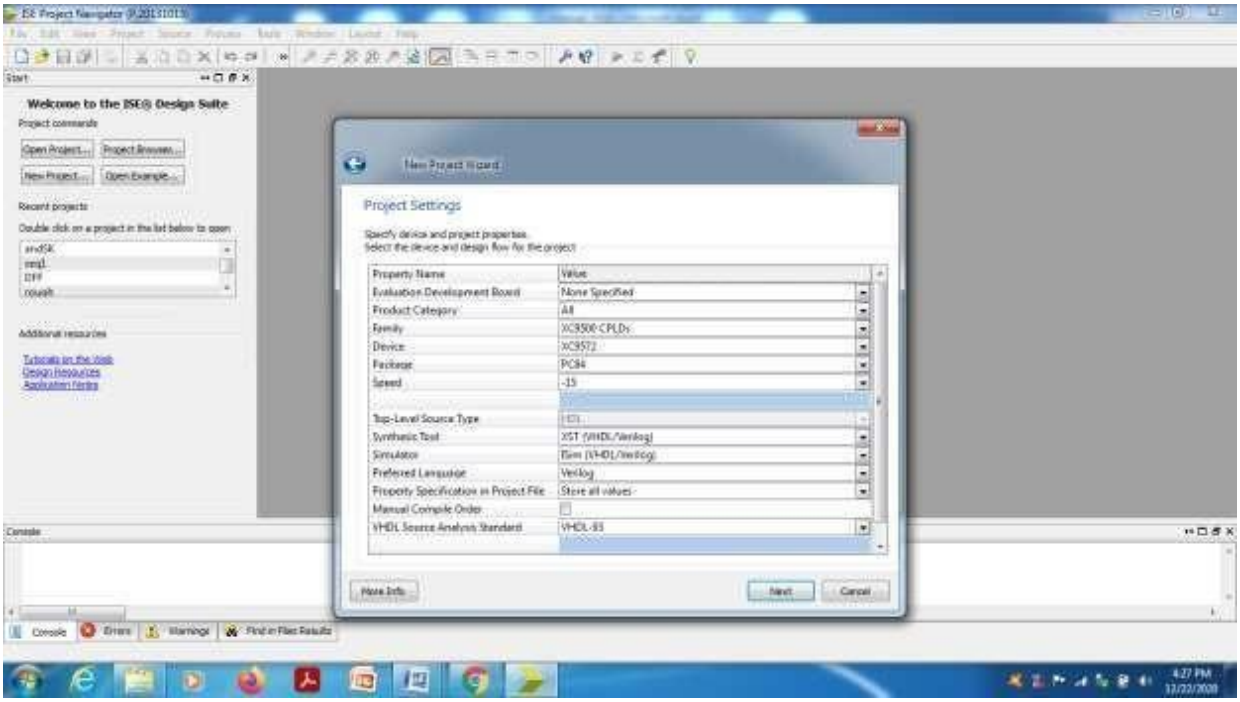
2. Select File-New Project



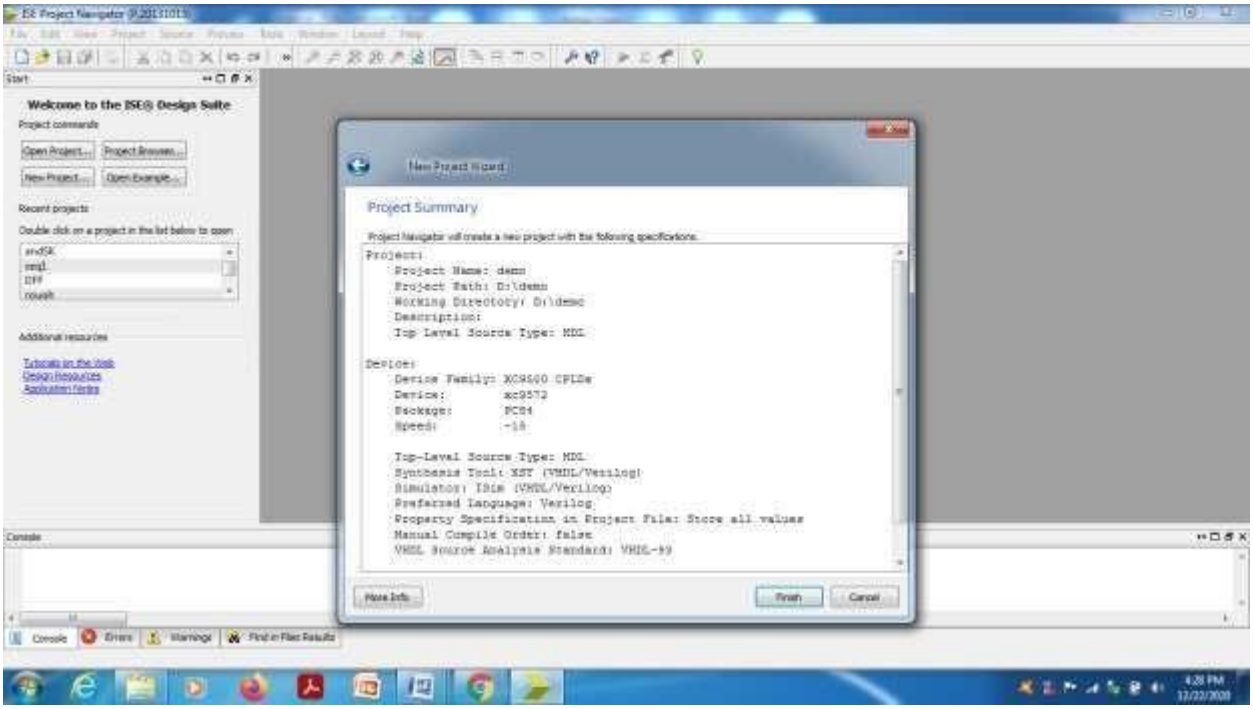
3. Give the name of the project, browse the location for the project and select ‘Next’



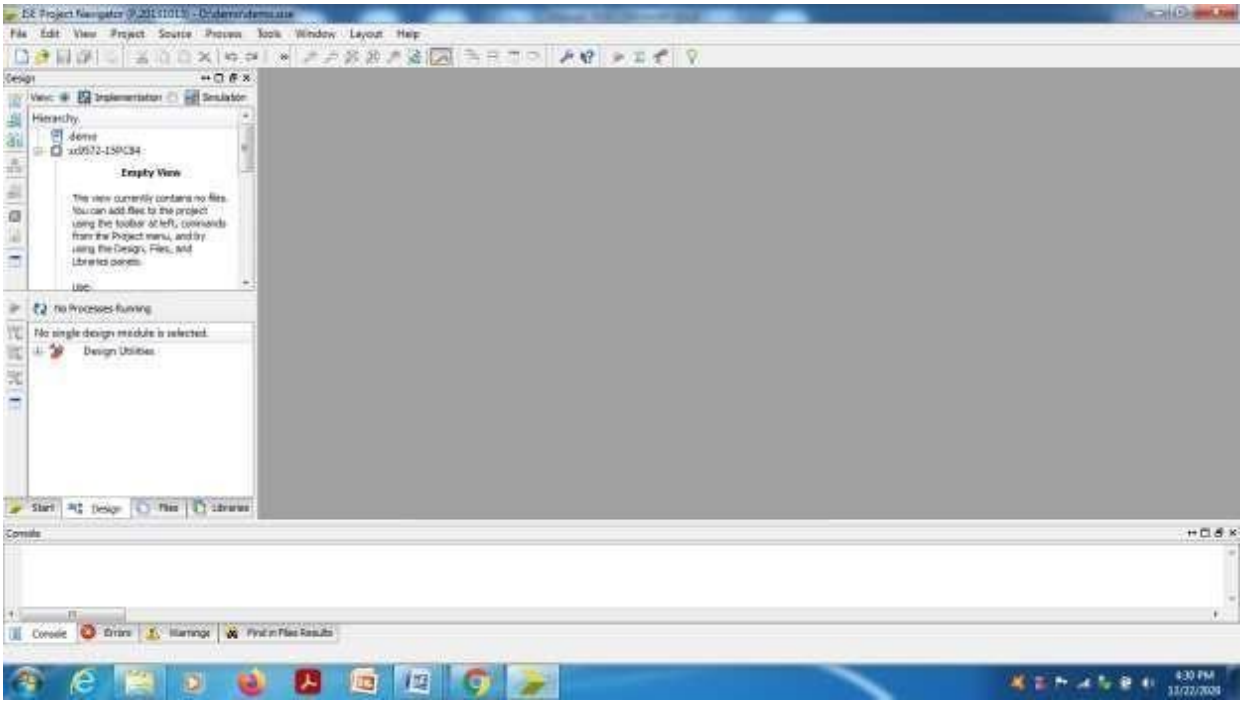
4. Enter the CPLD device details and choose ‘Next’.



5. Verify the previous steps in the summary and choose ‘Finish’.

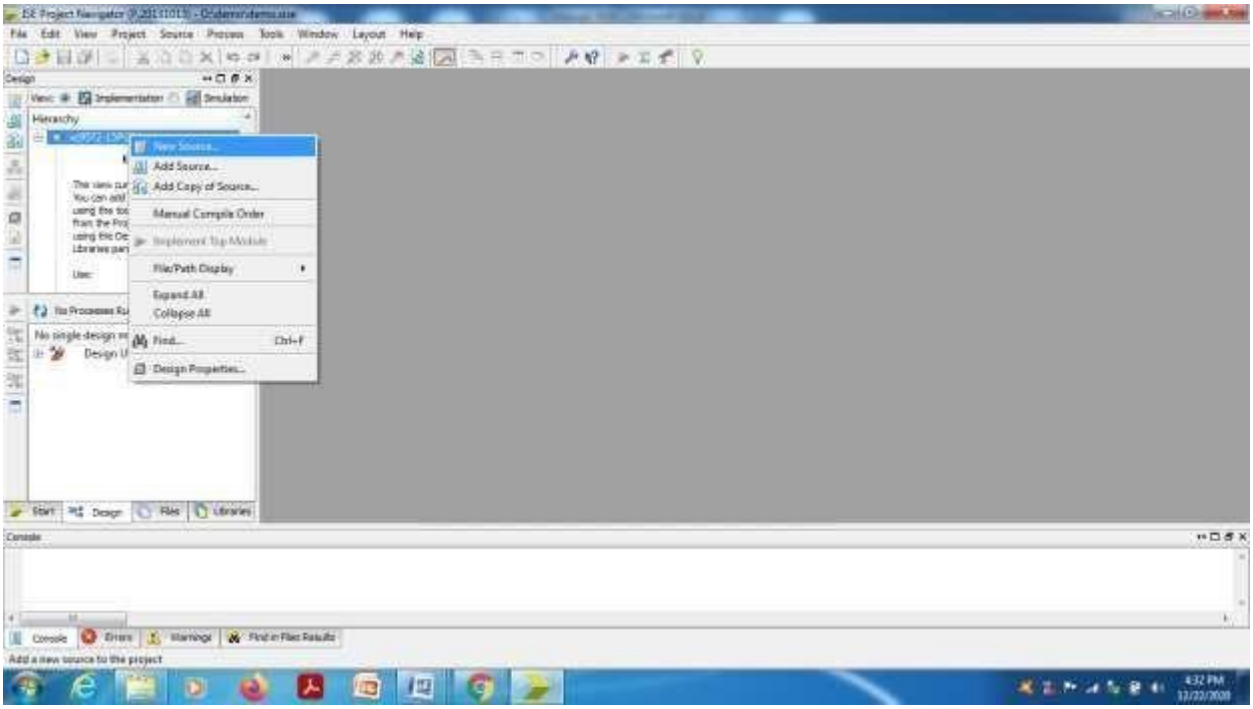


6. Project window will be opened.

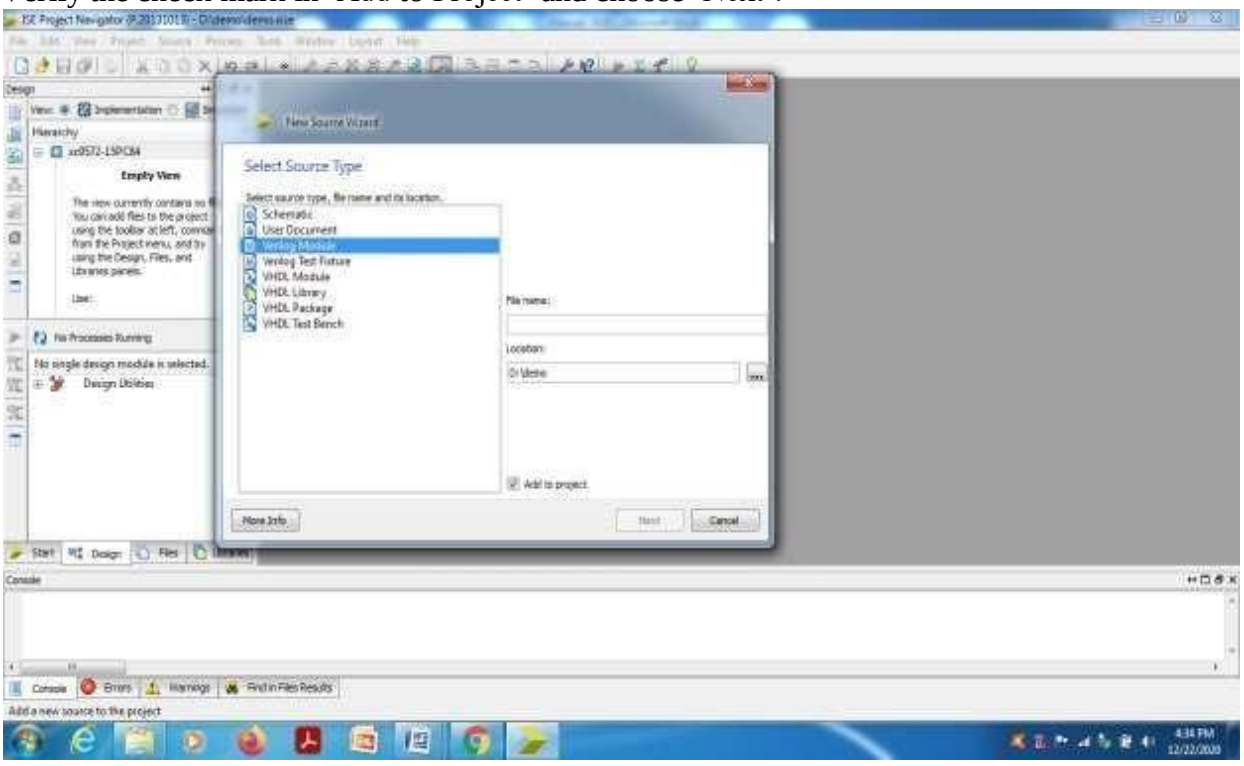


II. To create a Verilog source(program) under the project.

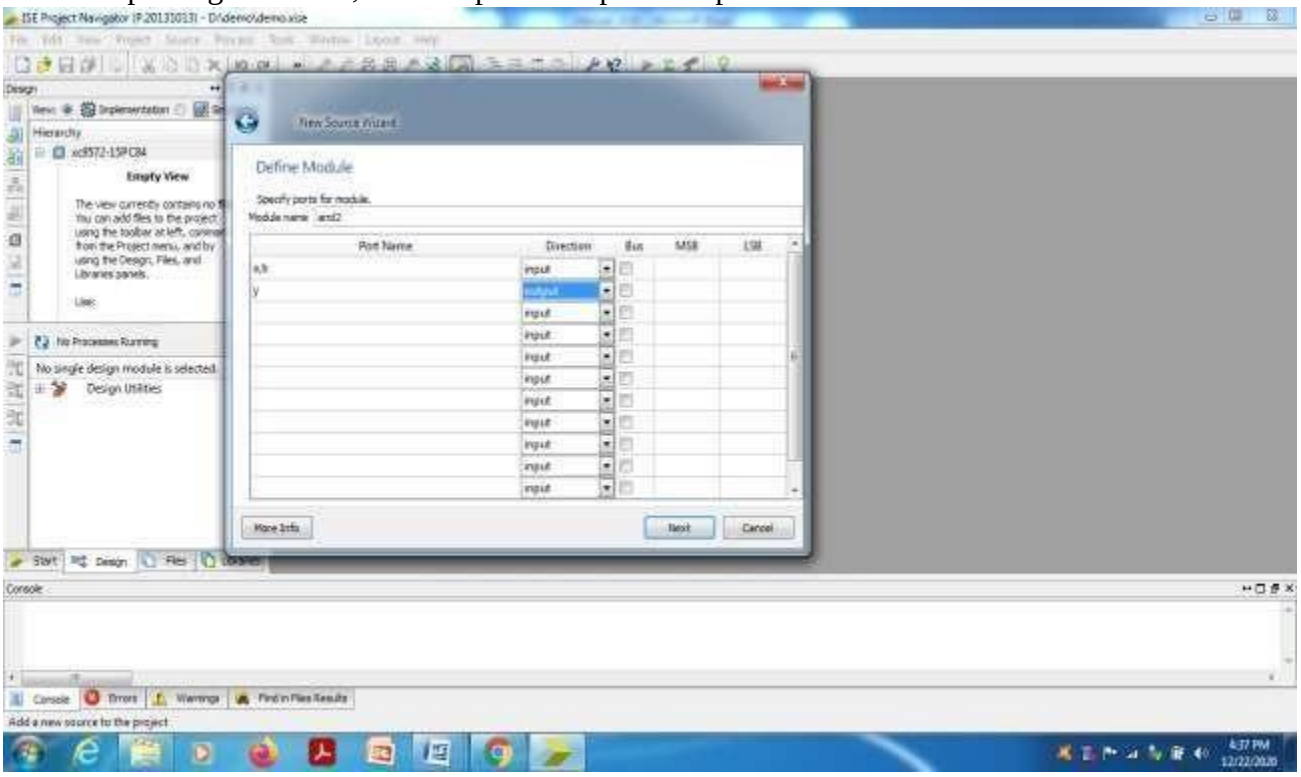
Select the device ‘xc9572-15PC84’, right click and select ‘New Source’.



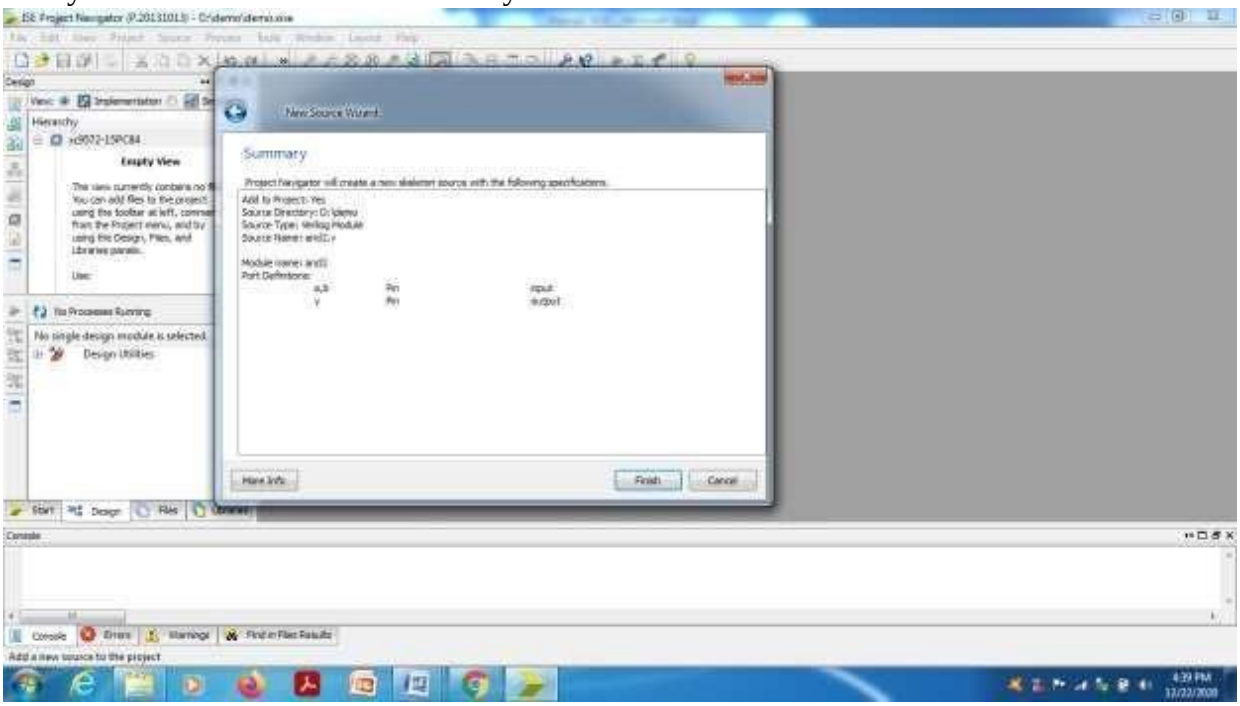
Select ‘Verilog Module’ from source type window and give the file name without any extension.
Verify the check mark in ‘Add to Project’ and choose ‘Next’.



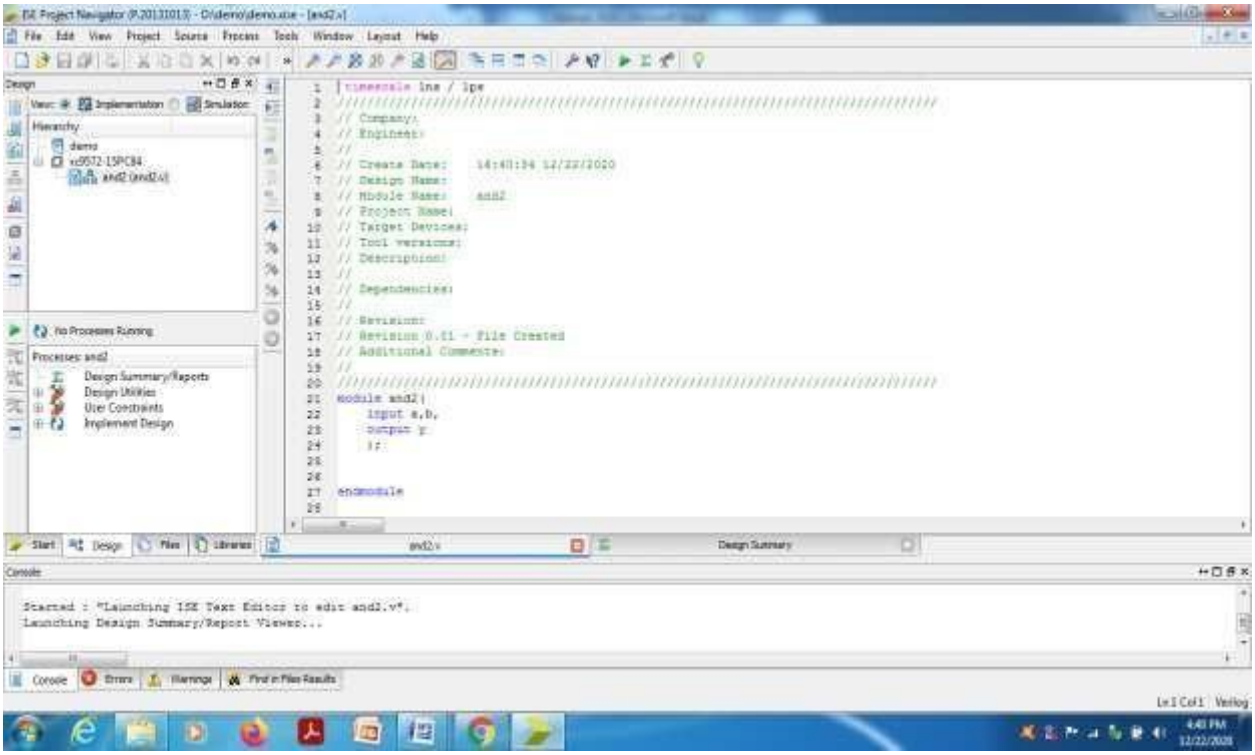
Write the port signal names, select input or output from pull down menu and choose ‘Next’.

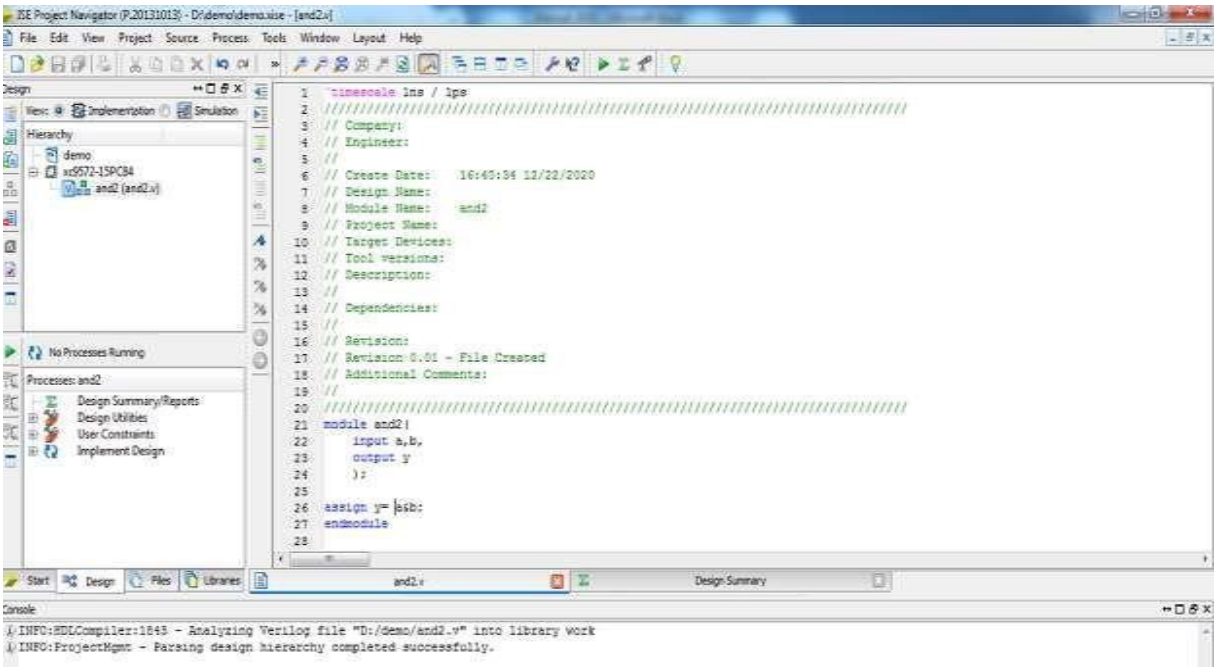


Verify the source details in the summary and choose ‘Finish’.

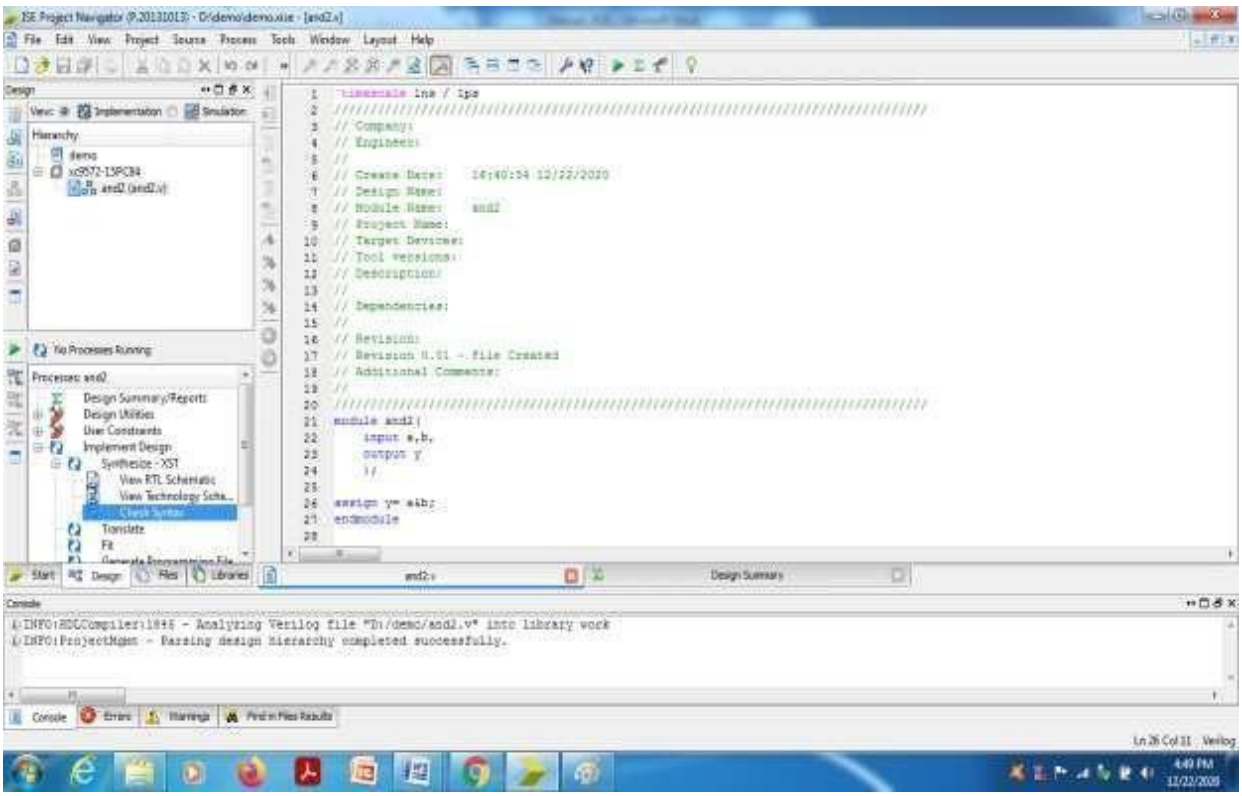


Verilog source file'and2.v' will be opened with skeleton. Edit the program and save.

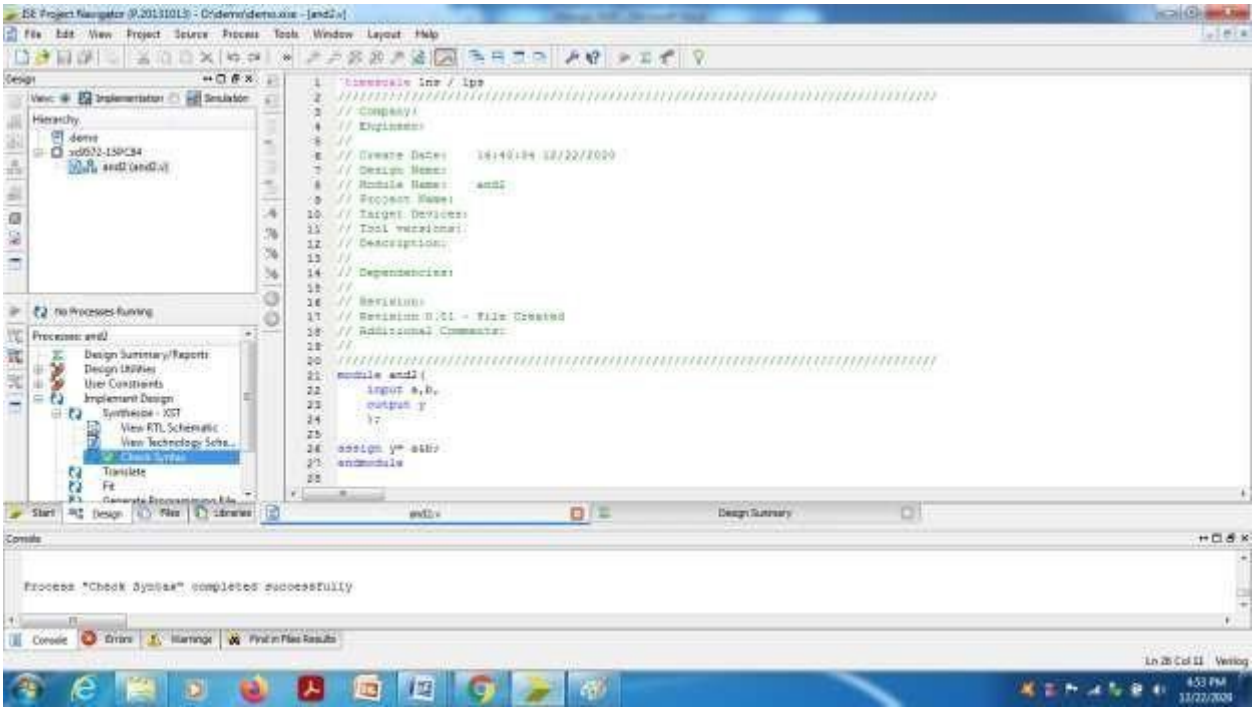




Select the ‘and2.v’ in project window and select
Implementation Design---Synthesize XST ----- Check syntax. (Double click).

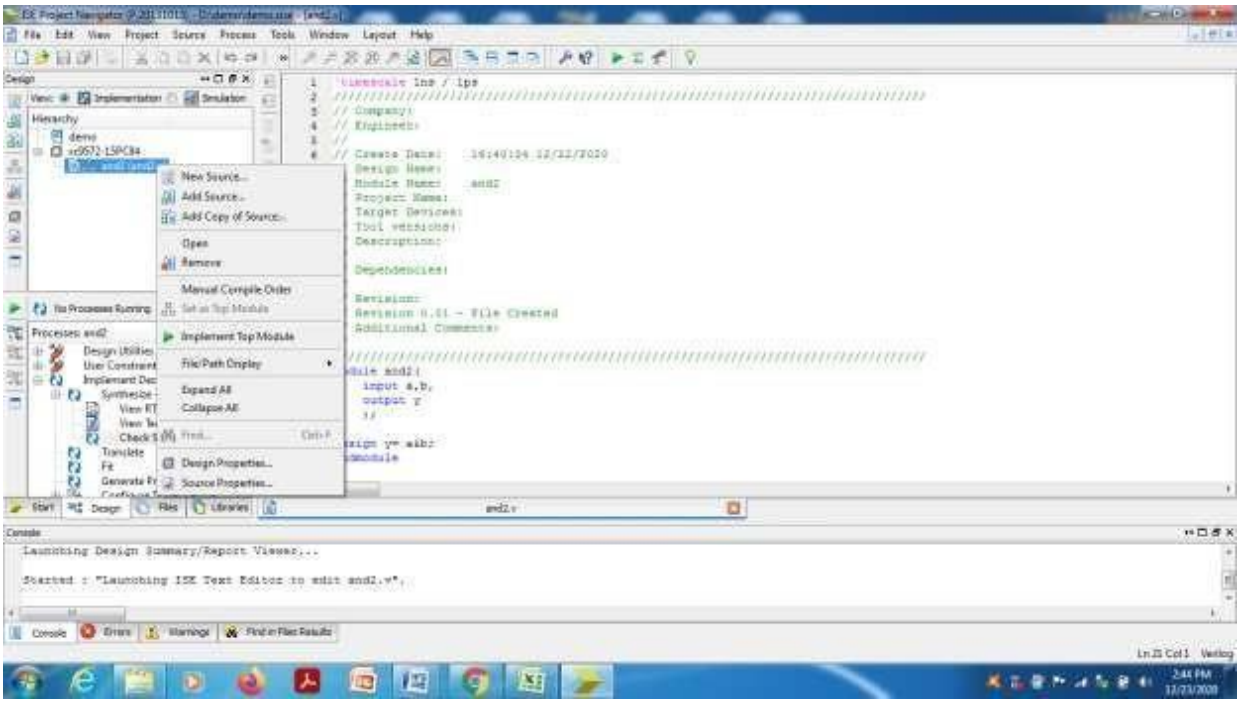


Check for any errors in the console window. If any error is shown, edit the program and do the corrections until ‘process ‘check syntax ‘completed successfully.

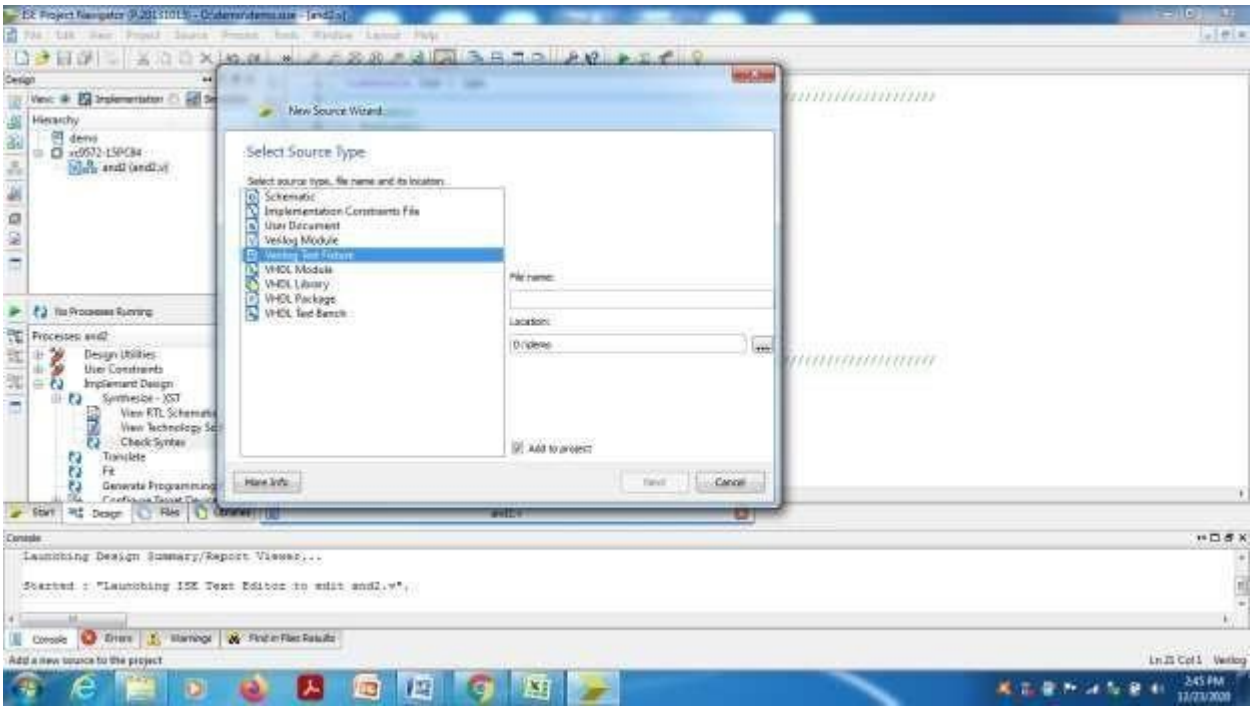


To create a test bench Program and perform the simulation:

Select the design program from Project window, right click, select ‘New source’.

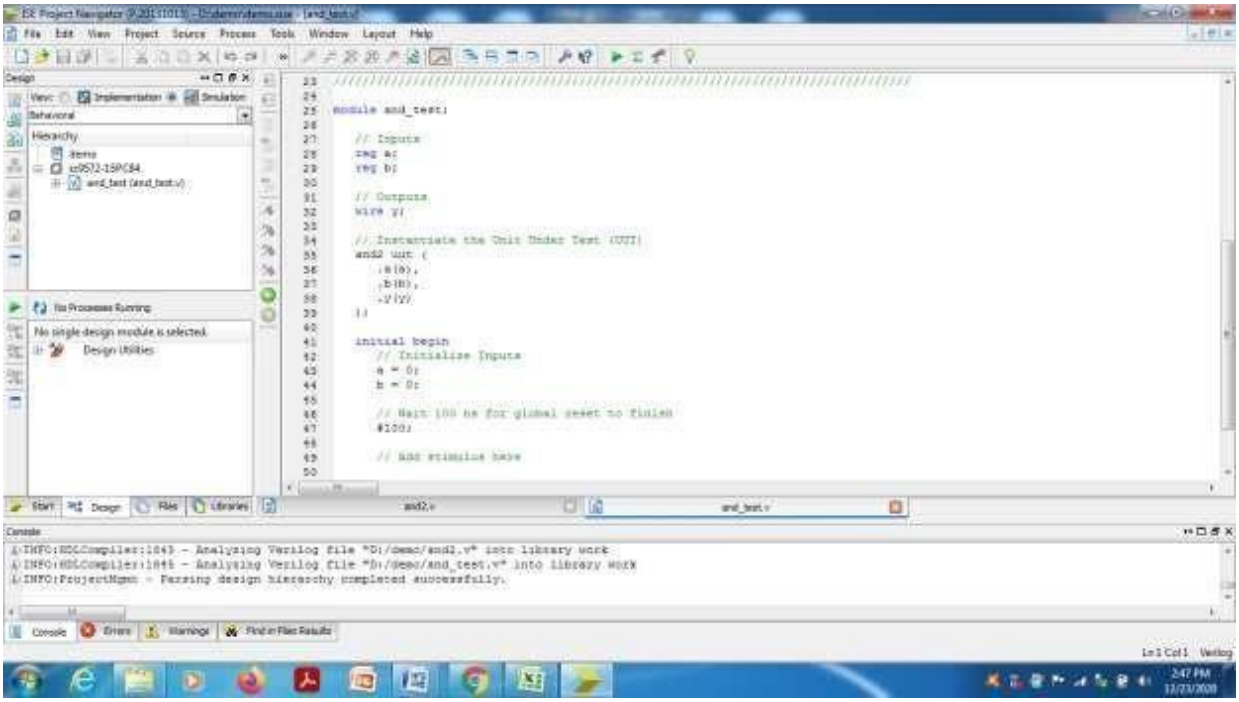


Select source type as ‘Verilog Test Fixture’ and give file name for the test bench program.

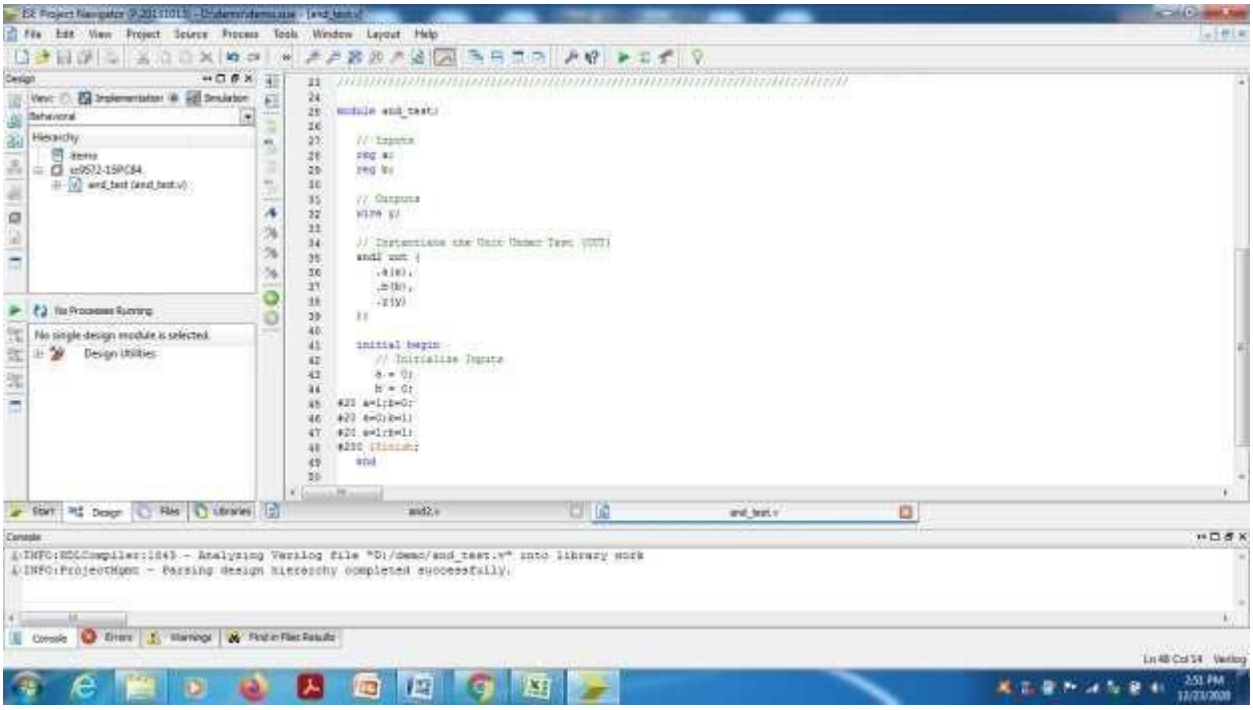


Choose Next---Next --- and Finish.

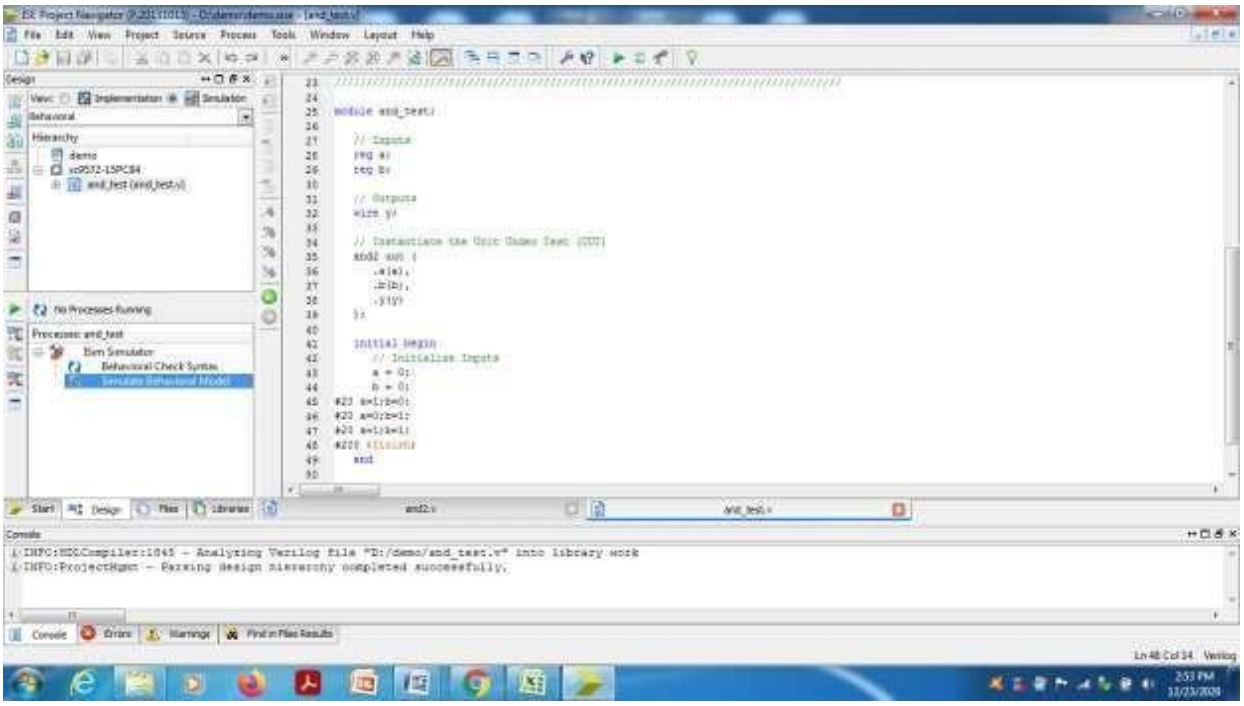
Change the view above Project window from ‘Implementation’ to ‘Simulation’ mode.



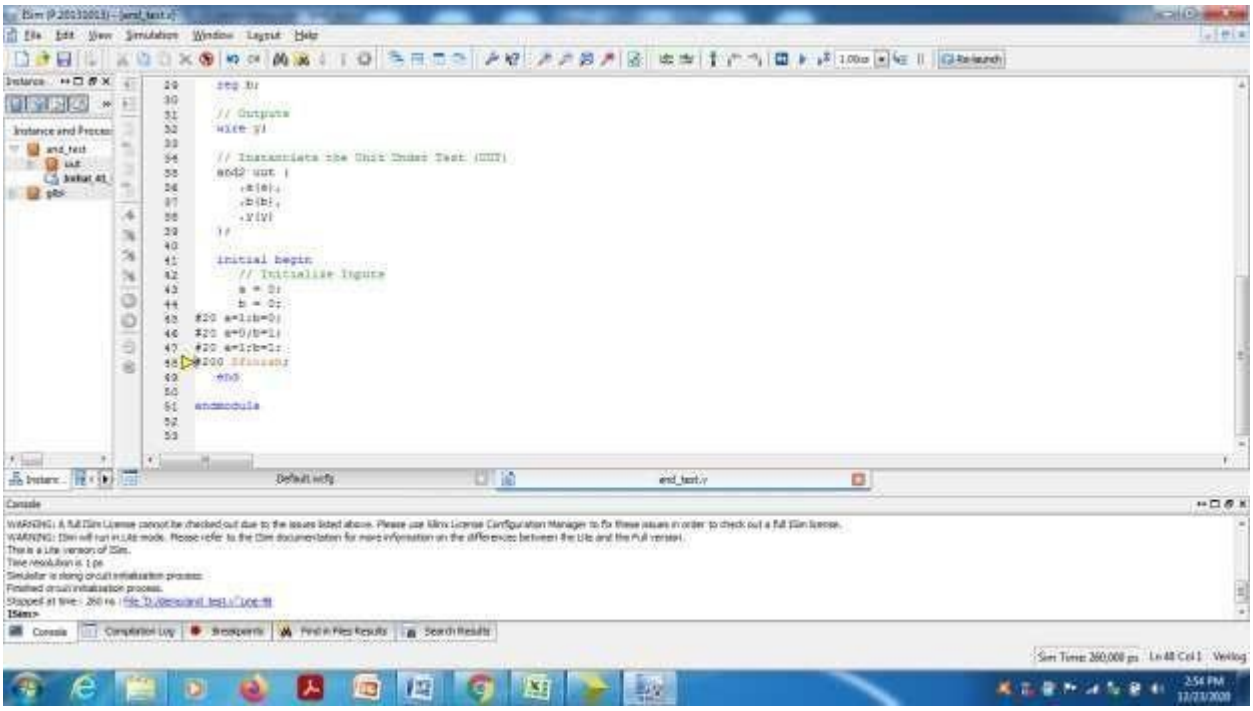
In the skeleton of test bench program, do the changes in the input test vectors inside the initial block. (Or the necessary changes according to the program requirement).



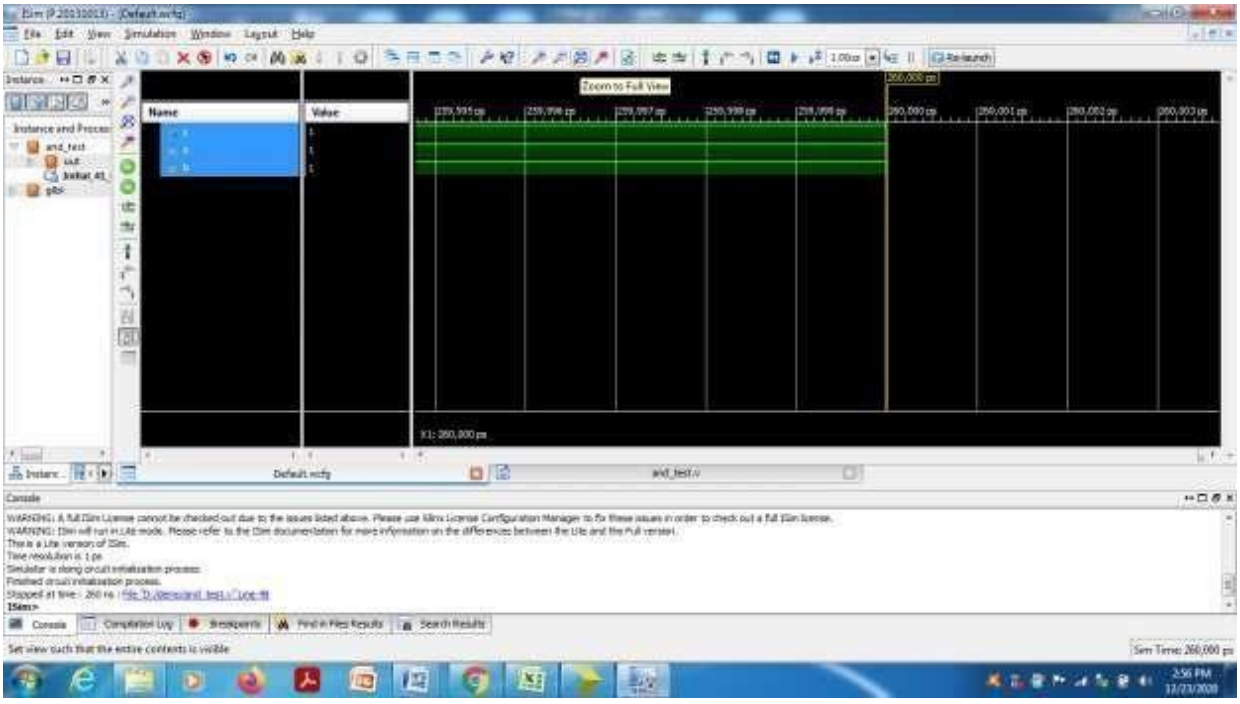
Perform the ‘Behavioral check syntax’ for any errors. If no errors, then choose ’Simulate Behavioral Model’.



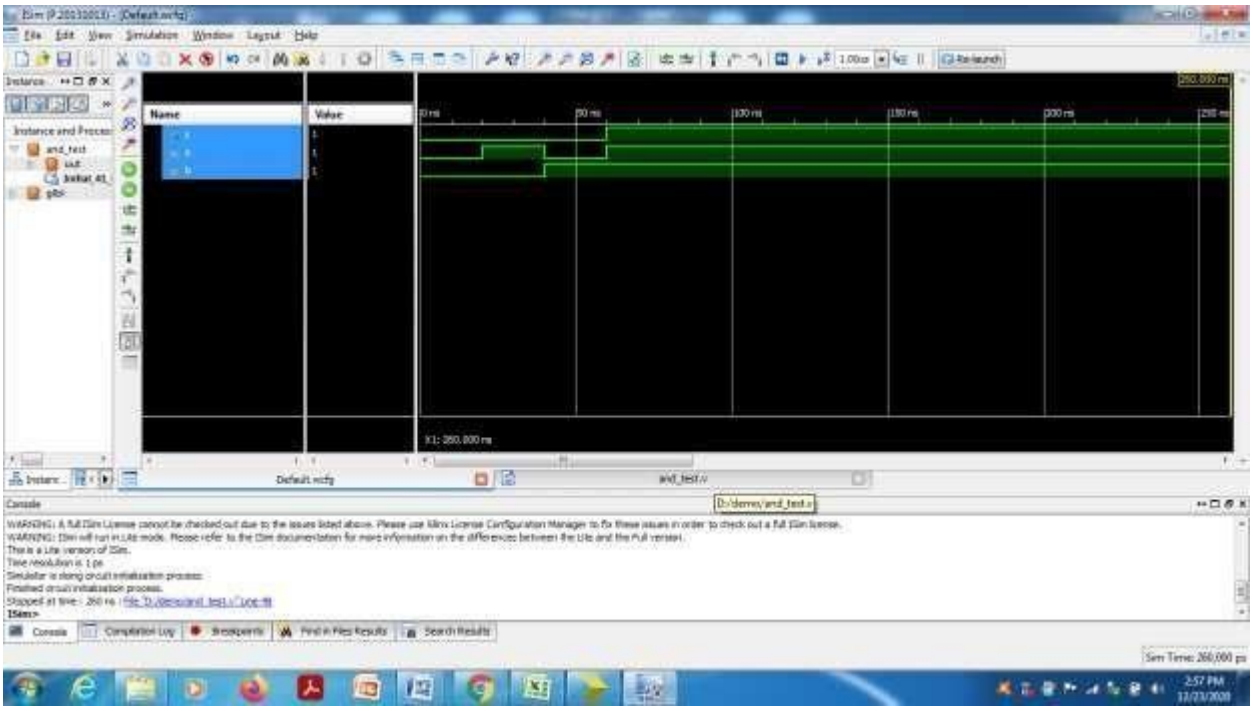
Isim window will be opened and open “default.wfcg” tab to view simulation result.



Select ‘Zoom to full view’.

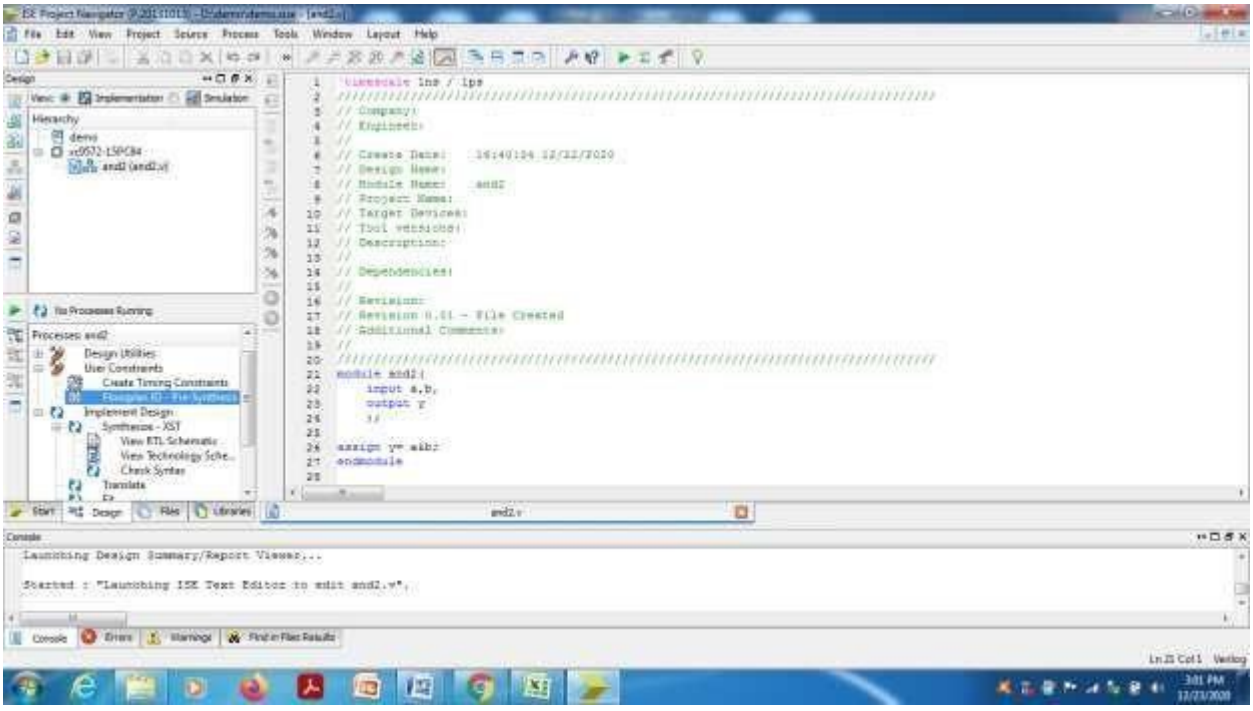


Place the marker at different inputs and verify the output.

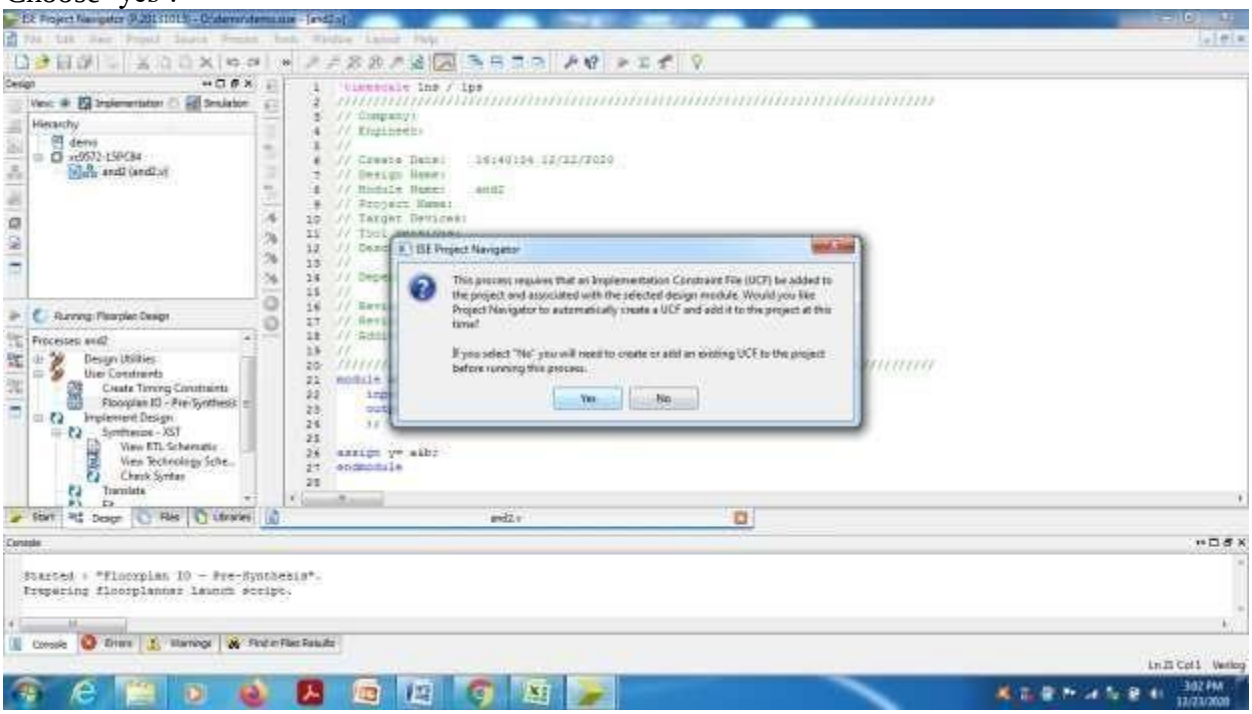


III. To Create .UCF file for pin assignment.

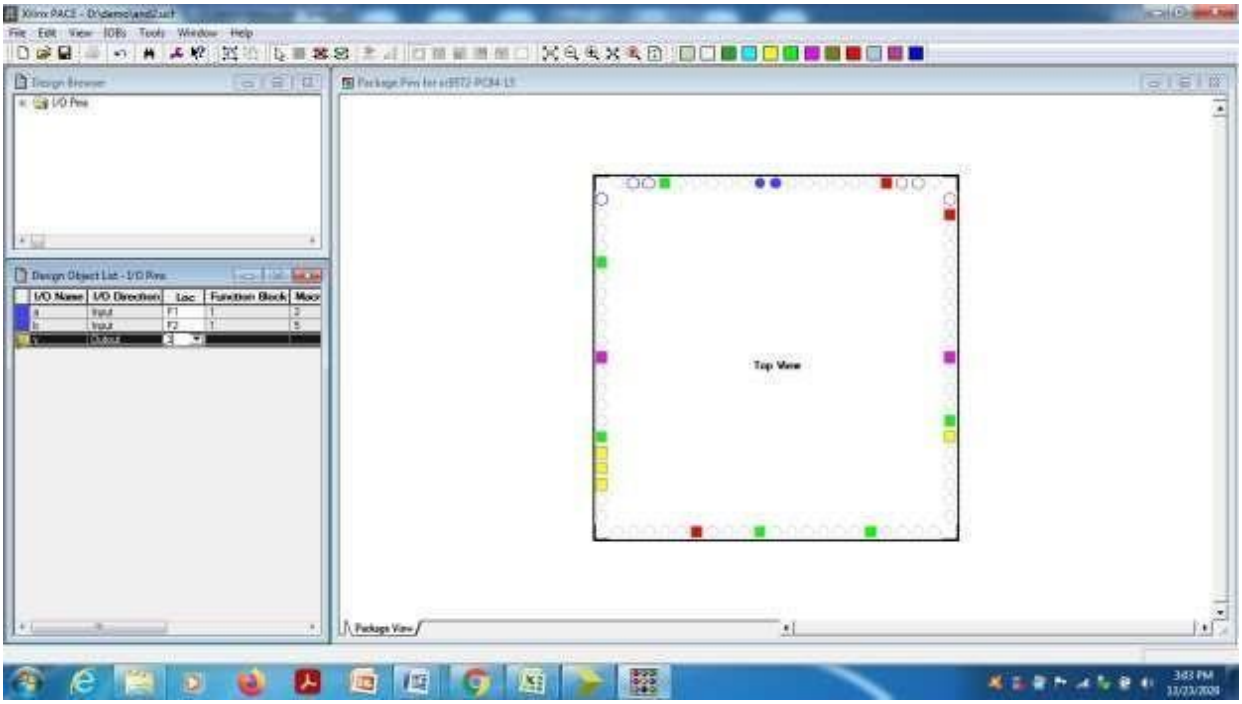
Change the view from ‘Simulation’ to ‘Implementation’ (above the project window).
Select the design program by just one click and choose User constraints----Floor Planning-IO Pre synthesis.



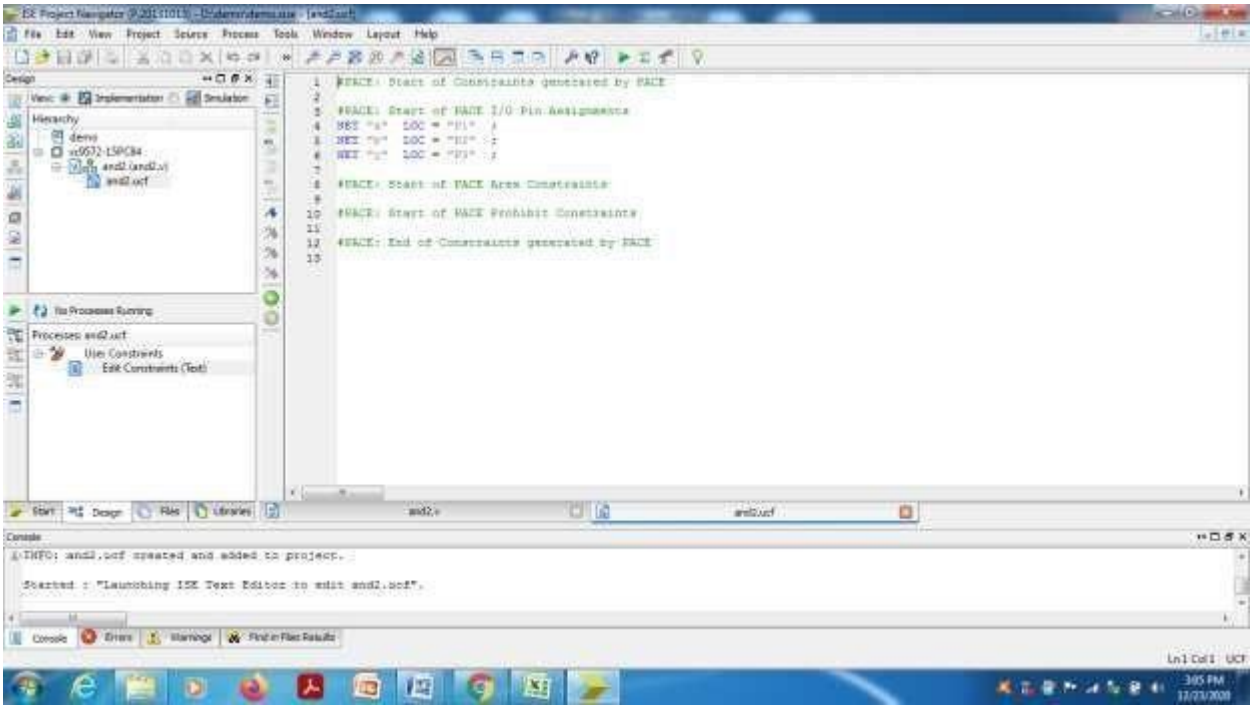
Choose ‘yes’.



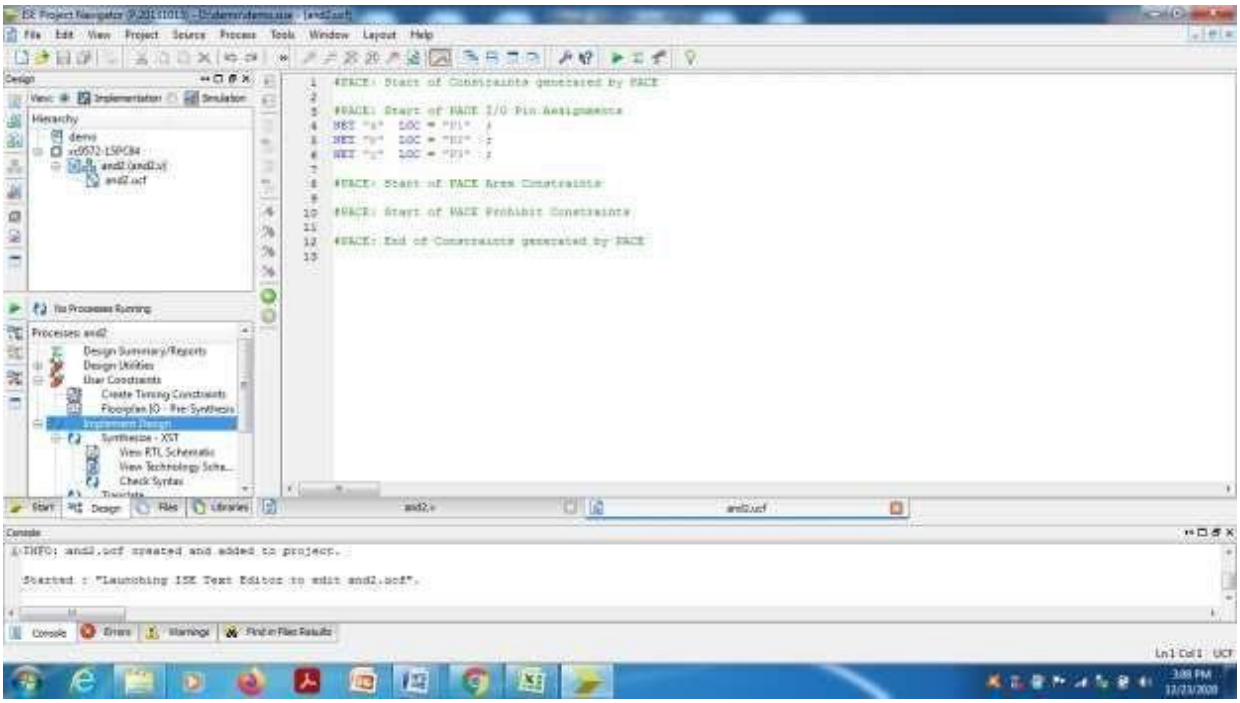
In the Xilinx PACE window, do the pin assignment according to the requirement and save, select ok and close the file. By doing this, and2.ucf file would have created and saved underthe design file.

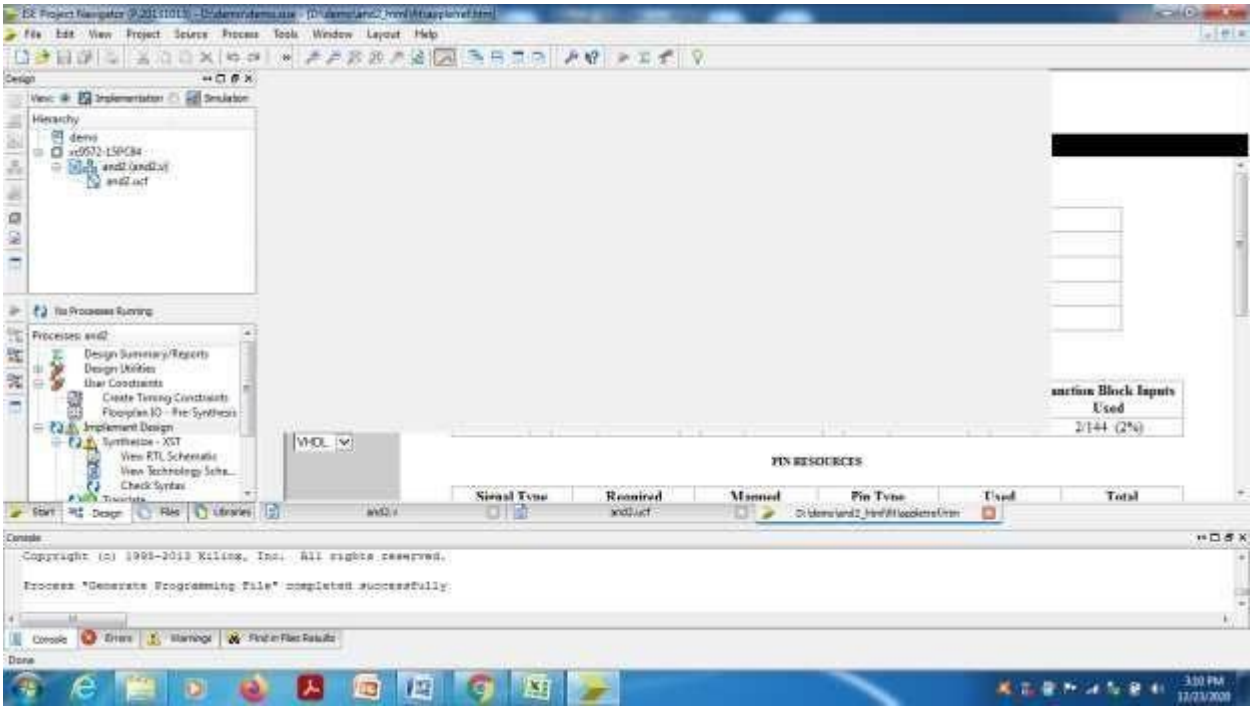


With ‘Edit constraints’ option, we can open and edit the .UCF file.

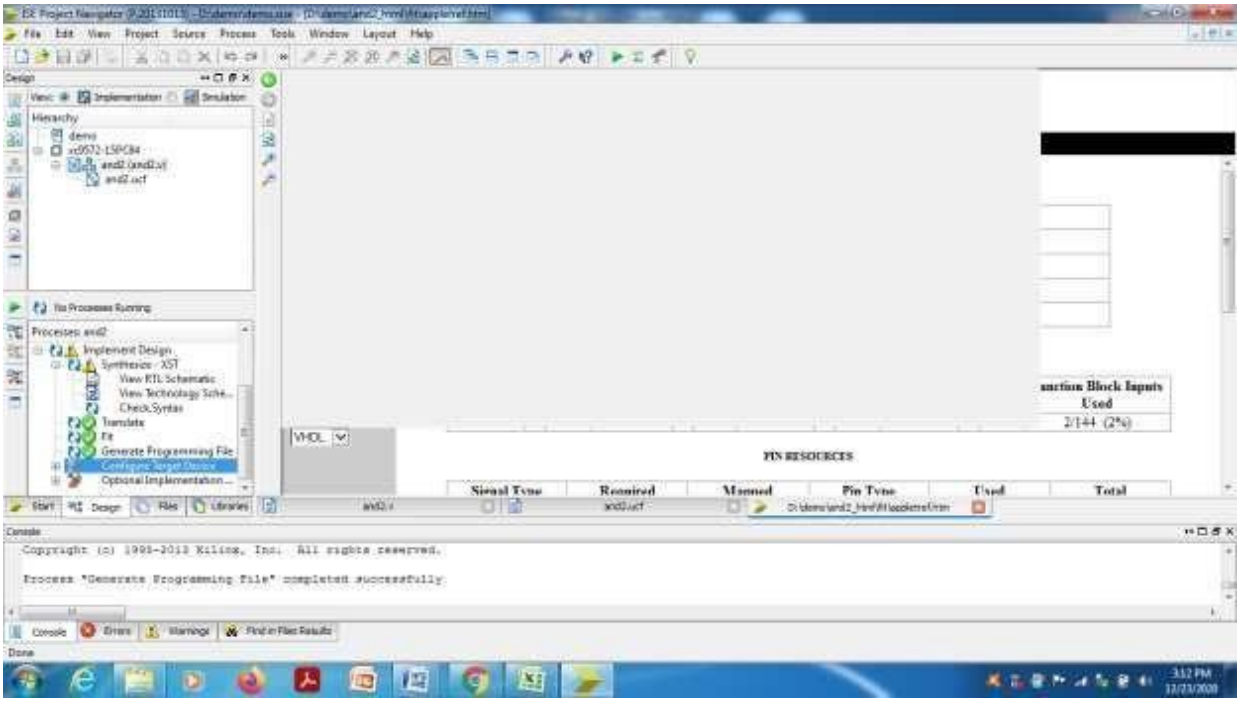


Select the design program in project window and double click on’ Implementation design’ and wait till we get the message as Process “Generate Programming file” completed successfully.

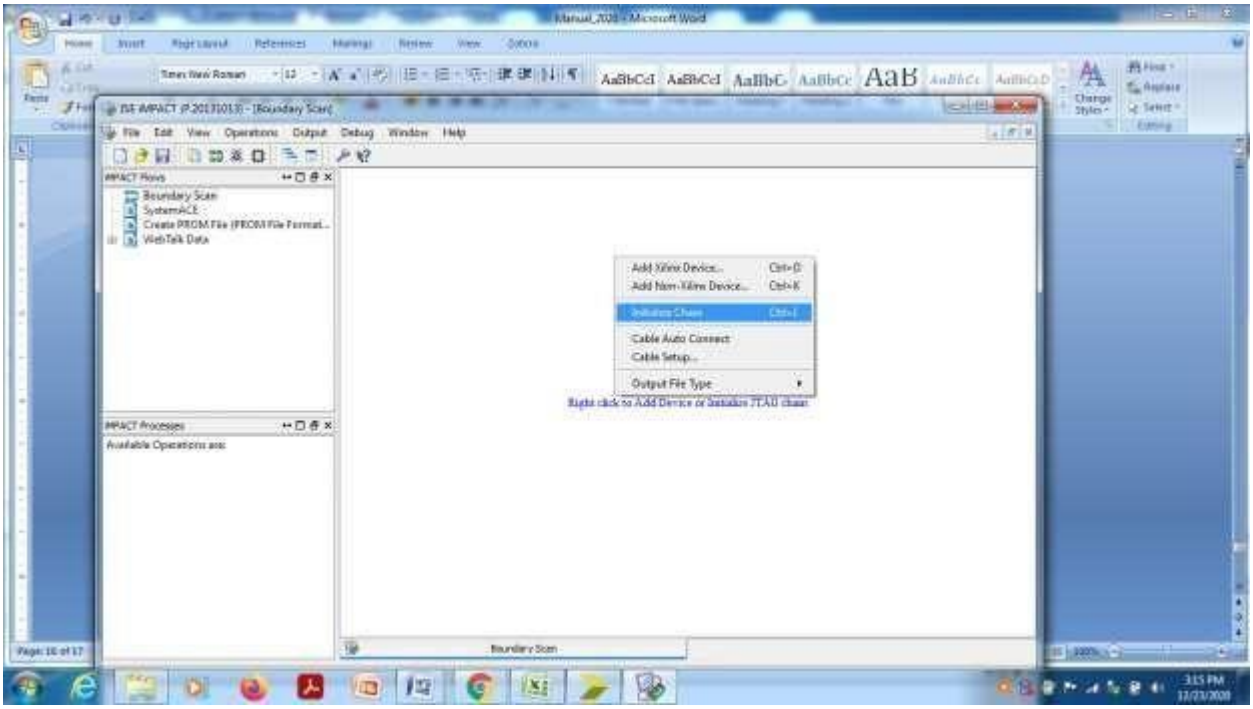




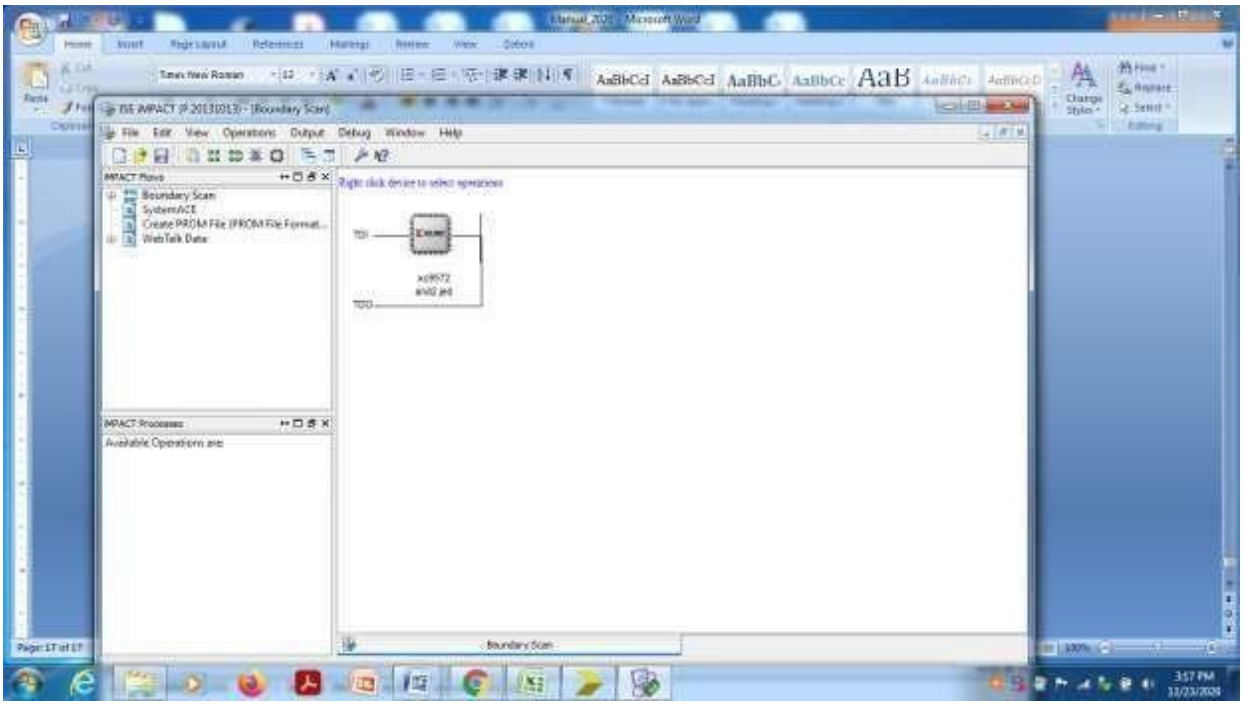
Double click on ‘Configure Target device’ for dumping the code on CPLD chip and choose ‘ok’.



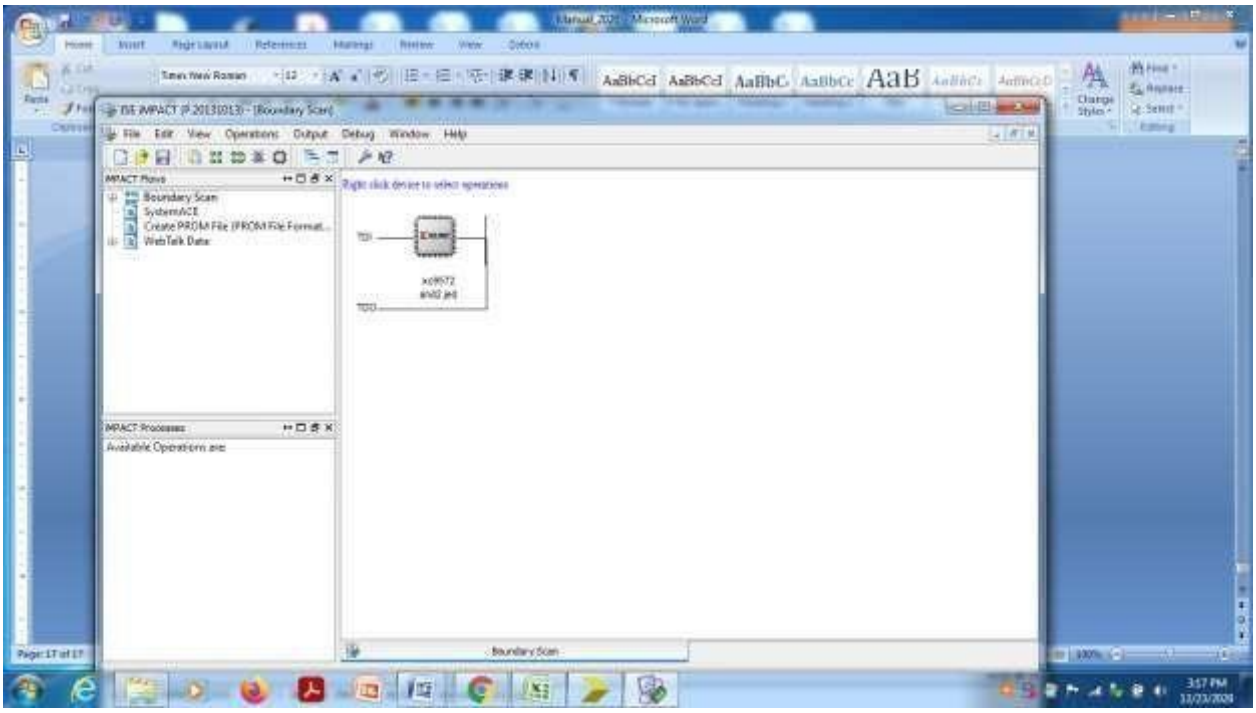
Xilinx ISE IMPACT window will be opened, select ‘Boundary scan’.
Right click on the workspace and select ‘Initialize the chain’.

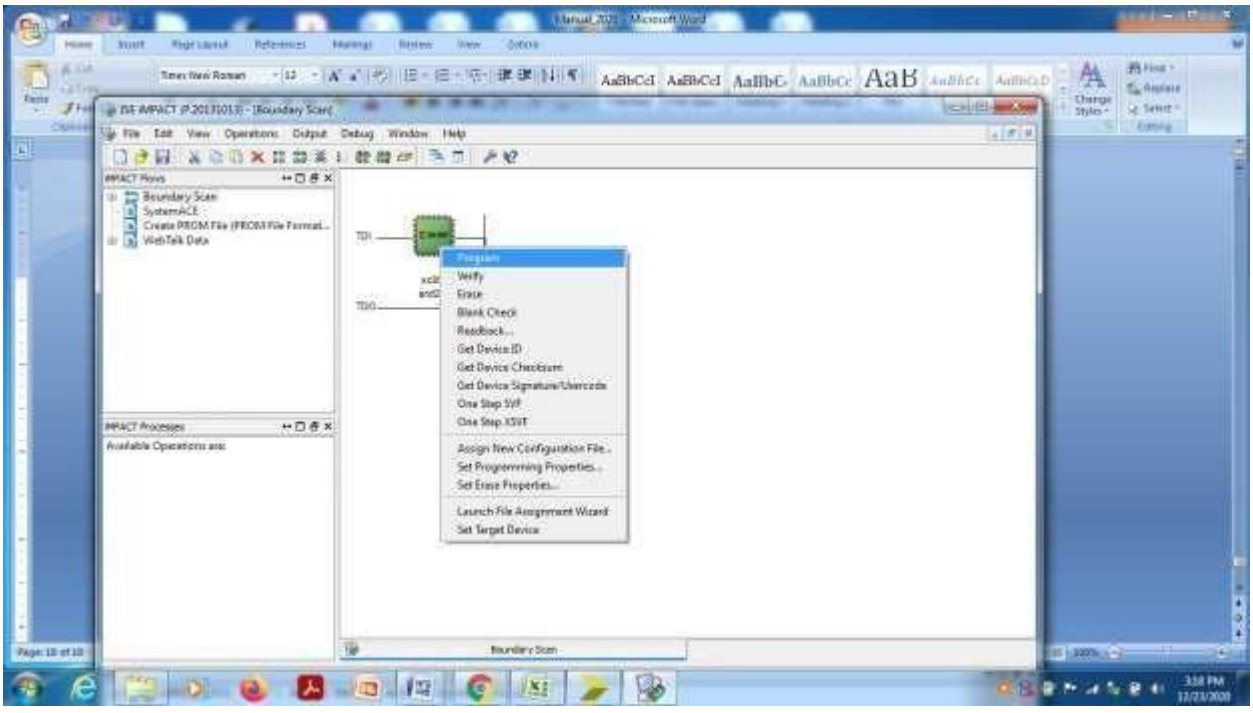


Browse the file name with .jed extension and assign to the IC icon.



Select the icon and right click, choose' Program'.





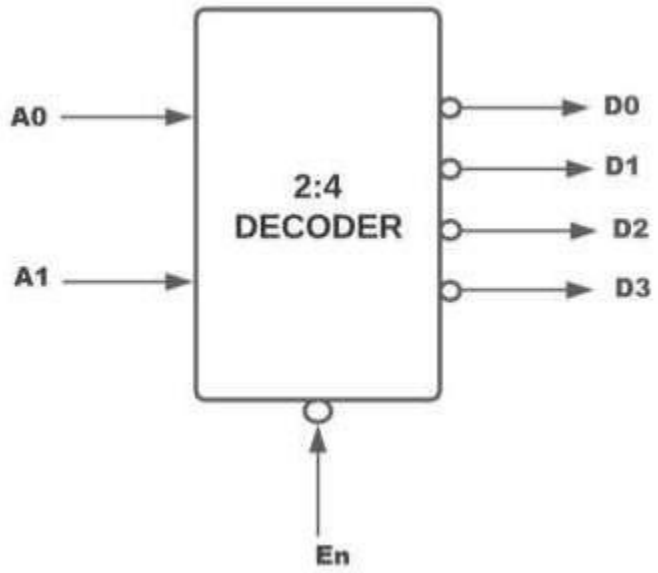
After getting 'Programming succeeded', feed the inputs and check the output in LEDs in hardware domain.

PART-A

Expt No. 1 a)

Aim: Write Verilog program for 2 to 4 decoder realization using NAND gates only (structural model) along with test bench to verify the design

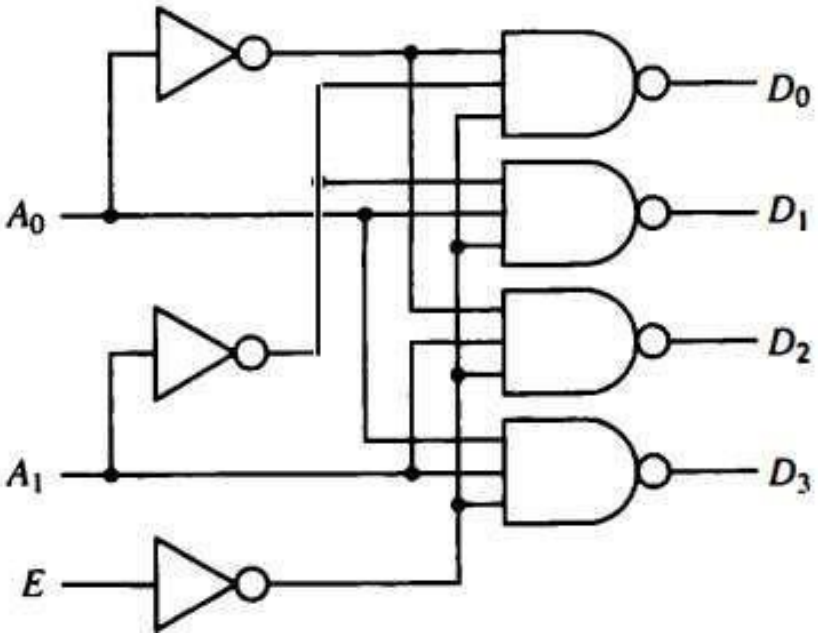
Block Diagram:



Truth Table:

En	INPUT		OUTPUT			
	A1	A0	D3	D2	D1	D0
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Circuit diagram:



Program:

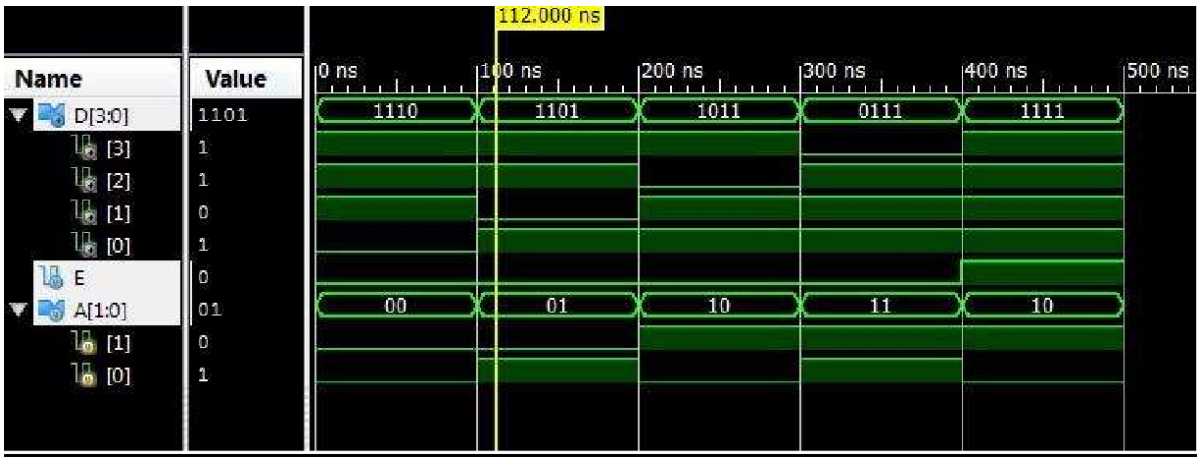
```
module decoder_24(
   input E,
   input [1:0] A,
   output [3:0] D
   );

wire E_bar, A0_bar, A1_bar;

not   not_01(E_bar,      E);
not not_02(A0_bar, A[0]);
not    not_03(A1_bar,  A[1]);

nand   nand_01(D[0],   A0_bar,   A1_bar,   E_bar);
nand   nand_02(D[1],   A[0],   A1_bar,   E_bar);
nand   nand_03(D[2],   A0_bar,   A[1],   E_bar);
nand nand_04(D[3], A[0], A[1],  E_bar);

endmodule
```

Test Bench:

```
module TB_decoder_24;

        // Inputs
        reg E;
        reg [1:0] A;

        // Outputs
        wire [3:0] D;

        // Instantiate the Unit Under Test (UUT)
        decoder_24 uut (
                .E(E),
                .A(A),
                .D(D)
        );

        initial begin
                // Initialize Inputs
                E = 0;
                A = 0; // A= 00 in binary

                // Wait 100 ns for global reset to finish
                #100;
```

```
                A = 1; // A= 01 in binary
                #100;

                A = 2; // A= 10 in binary
                #100;

                A = 3; // A= 11 in binary
                #100;


                E = 1;
                A = 2; // A= 11 in binary
                #100;

$finish;
                end

endmodule
```

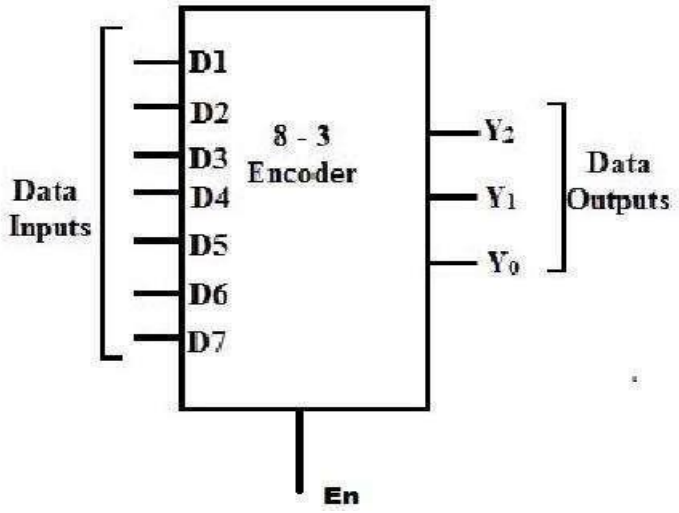
Simulation Result:



Expt No. 1 b)

Aim: Write Verilog program for 8 to 3 encoder with priority and without priority (behavioral model) along with test bench to verify the design.

Block Diagram:



Encoder Without Priority:

TruthTable:

INPUT									OUTPUT		
EN	D7	D6	D5	D4	D3	D2	D1	D0	Y2	Y1	Y0
0	X	X	X	X	X	X	X	X	Z	Z	Z
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	X	X	X

Program:

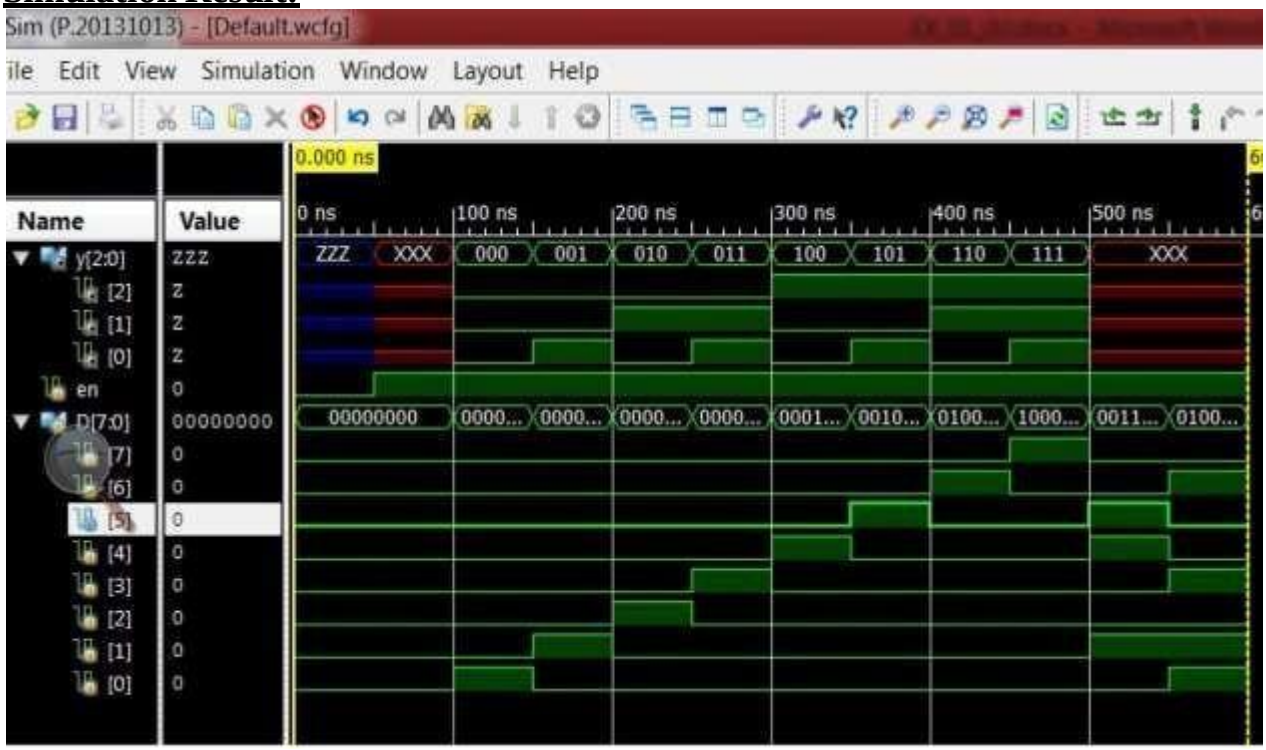
```
module encoder_83(
   input en,
   input [7:0] D,
   output reg [2:0] y
   );

always@(en,D)
        begin
                if(en)
                        case(D)

                        8'd1    : y=3'b000;
                        8'd2    : y=3'b001;
                        8'd4    : y=3'b010;
                        8'd8    : y=3'b011;
                        8'd16   : y=3'b100;
                        8'd32   : y=3'b101;
                        8'd64   : y=3'b110;
                        8'd128: y=3'b111;
                        default: y= 3'bxxx;
                        endcase

                else
                        y=3'bzzz;

                end     //end always
endmodule
```

Test Bench:

```
module encoderWOP_83_TB;

        // Inputs
        reg en;
        reg [7:0] D;

        // Outputs
        wire [2:0] y;

        // Instantiate the Unit Under Test (UUT)
        encoder_83 uut (
                .en(en),
                .D(D),
                .y(y)
        );

        initial begin
                // Initialize Inputs
```

```
        en = 0;
        D = 0;
        #50;

        en = 1;
        D = 0;
        #50;

        D = 1;
        #50;

        D = 2;
        #50;

        D = 4;
        #50;

        D = 8;
        #50;

        D =16;
        #50;

        D = 32;
        #50;

        D = 64;
        #50;

        D = 128;
        #50;

        D = 50;
        #50;

        D = 75;
        #50;

$finish;
        end
endmodule
```

Simulation Result:



Encoder With Priority: Truth Table:

INPUT									OUTPUT		
EN	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Y2**	**Y1**	**Y0**
0	X	X	X	X	X	X	X	X	Z	Z	Z
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	X	0	0	1
1	0	0	0	0	0	1	X	X	0	1	0
1	0	0	0	0	1	X	X	X	0	1	1
1	0	0	0	1	X	X	X	X	1	0	0
1	0	0	1	X	X	X	X	X	1	0	1
1	0	1	X	X	X	X	X	X	1	1	0
1	1	X	X	X	X	X	X	X	1	1	1
1	0	0	0	0	0	0	0	0	X	X	X

Program:

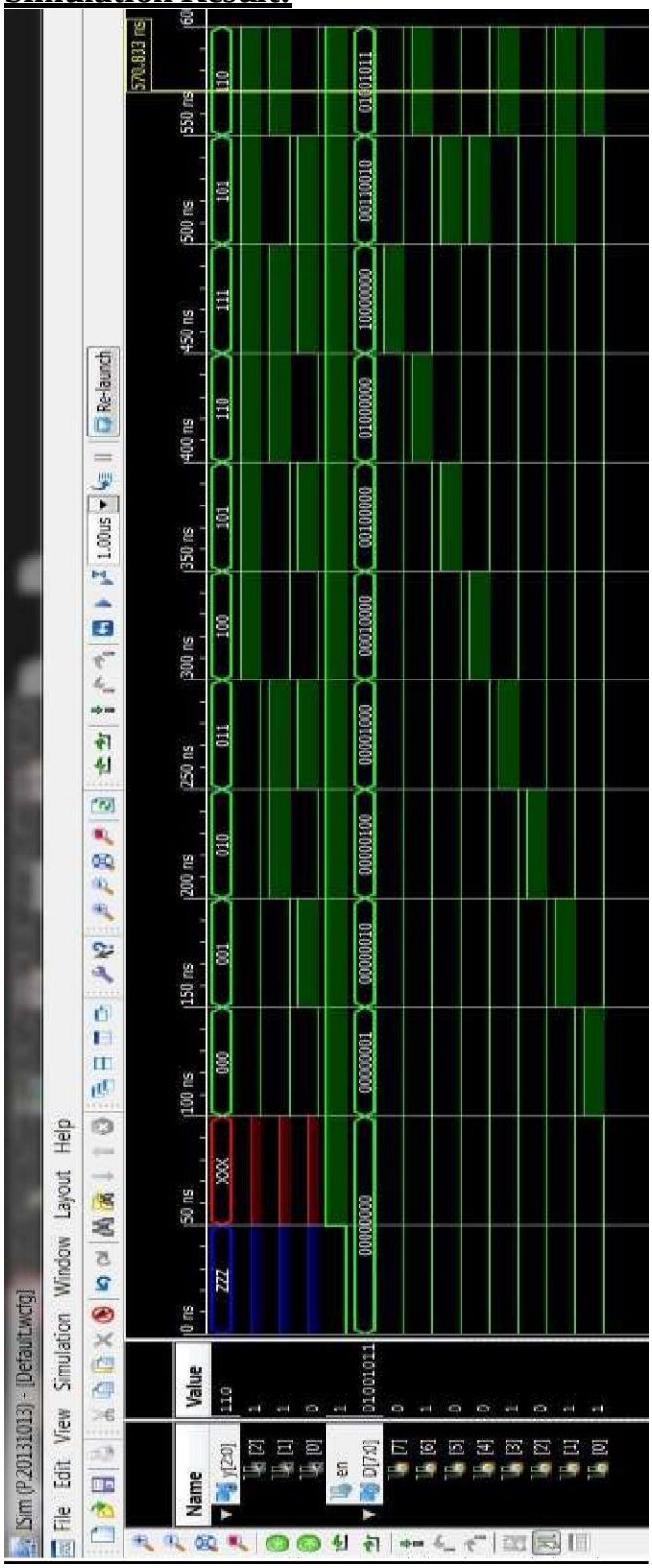
```
module encoder83_WP(
   input en,
   input [7:0] D,
   output reg [2:0] y
   );

always@(en,D)
        begin

                if(en)

                        casex(D)11111111

                        8'b0000_0001: y=3'b000;
                        8'b0000_001x: y=3'b001;
                        8'b0000_01xx: y=3'b010;
                        8'b0000_1xxx:  y=3'b011;
                        8'b0001_xxxx:  y=3'b100;
                        8'b001x_xxxx:  y=3'b101;
                        8'b01xx_xxxx:  y=3'b110;
                        8'b1xxx_xxxx: y=3'b111;
                        default: y=3'bxxx;
                        endcase
                else
                        y=3'bZZZ;
                end

endmodule
```

Test Bench:

```
module TB_encoder83_WP;

        // Inputs
        reg en;
        reg [7:0] D;

        // Outputs
        wire [2:0] y;

        // Instantiate the Unit Under Test (UUT)
        encoder83_WP uut (
                .en(en),
                .D(D),
                .y(y)
        );
```

```
    initial

    begin
            en = 0;
            D = 0;
            #50;            // Add stimulus here

            en = 1;
            D = 0;
            #50;

            D = 1;
            #50;

            D = 2;
            #50;

            D = 4;
            #50;

            D = 8;
            #50;

            D =16;
            #50;

            D = 32;
            #50;

            D = 64;
            #50;

            D = 128;
            #50;

            D = 50;
            #50;

            D = 75;
            #50;

            $finish;
    end
endmodule
```

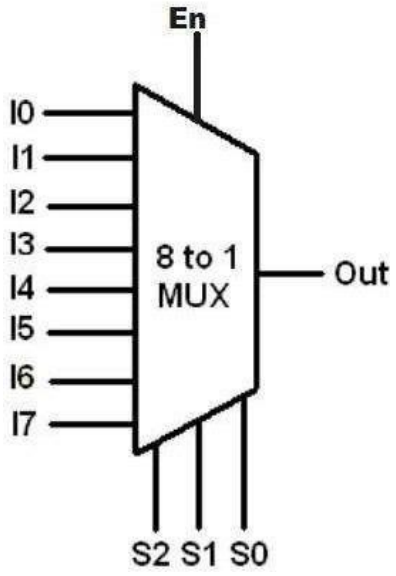
Simulation Result:



Expt No. 1 c)

Aim: Write Verilog program for 8 to 1 multiplexer using case statement and if statements along with test bench to verify the design

Block Diagram:



Truth Table:

E	Selection			Y
	S2	**S1**	**S0**	
0	X	X	X	z
1	0	0	0	I0
1	0	0	1	I1
1	0	1	0	I2
1	0	1	1	I3
1	1	0	0	I4
1	1	0	1	I5
1	1	1	0	I6
1	1	1	1	I7

Program using *if* statement.

```
module mux81_if_st(
        input en,
        input [7:0] i,
        input [2:0] sel,
        output reg y
        );

    always@(en,i,sel)
    begin

            if(en)
            begin

                if(sel==3'b000)
                        y=i[0];

                else if(sel==3'b001)
                        y=i[1];

                else if(sel==3'b010)
                        y=i[2];

                else if(sel==3'b011)
                        y=i[3];

                else if(sel==3'b100)
                        y=i[4];

                else if(sel==3'b101)
                        y=i[5];

                else if(sel==3'b110)
                        y=i[6];

                else if(sel==3'b111)
                        y=i[7];
                else
                    y=1'bx;
                 end                    //end if statement
                 else
                    y=1'bz;     // if en=0, then o/p should be high impedance state
            end                                 //end always statement
    endmodule
```

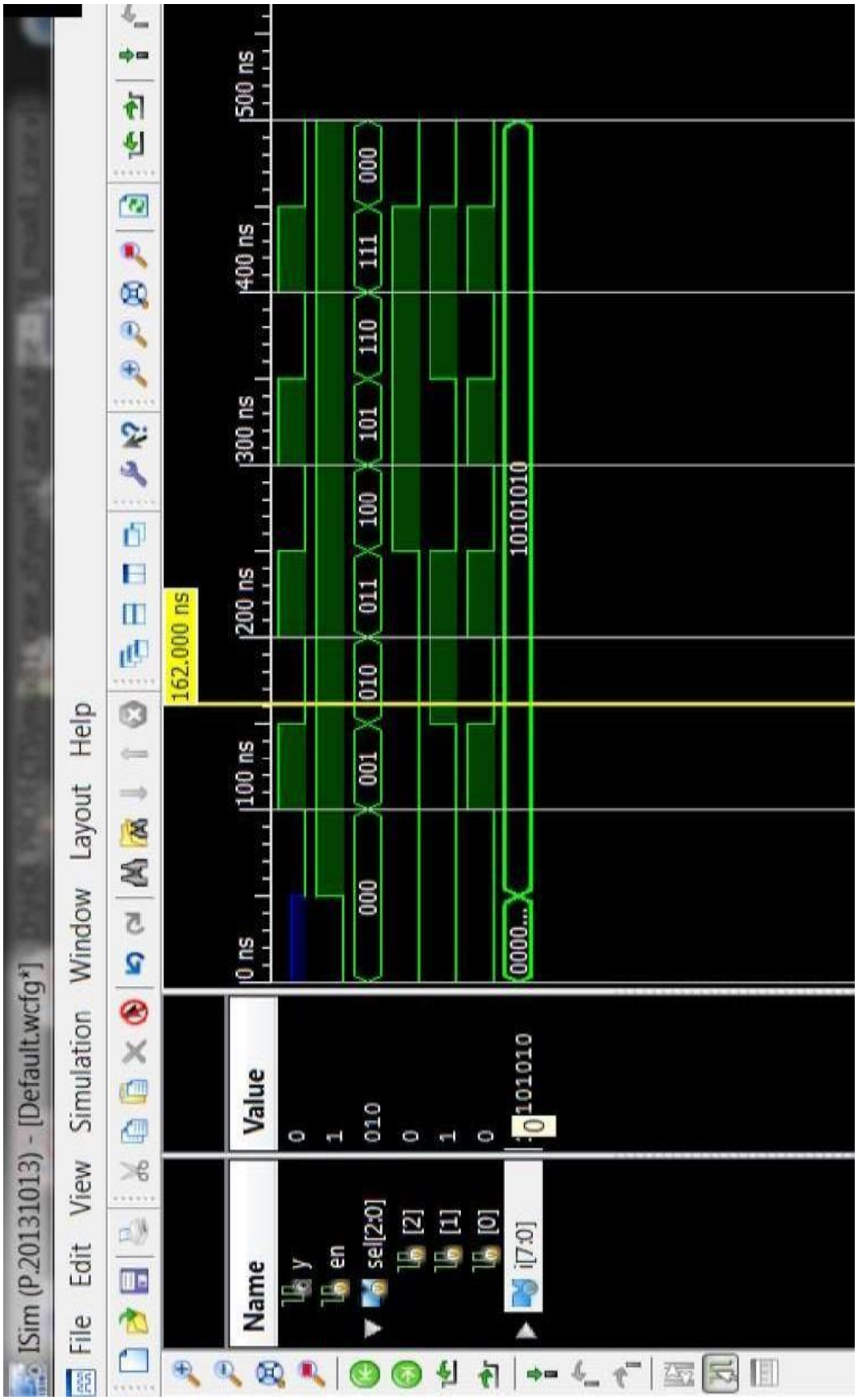
Program using ***case*** statement.

```
module mux81_case(
   input en,
   input [7:0] i,
   input [2:0] sel,
   output reg y
    );

always@(en,i,sel)
        begin
        if(en)
                case(sel)
                3'b000: y=i[0];
                3'b001: y=i[1];
                3'b010: y=i[2];
                3'b011: y=i[3];
                3'b100: y=i[4];
                3'b101: y=i[5];
                3'b110: y=i[6];
                3'b111: y=i[7];
                default: y=1'bx;
                endcase

        else
                y=1'bz;
         end

endmodule
```

Test Bench:

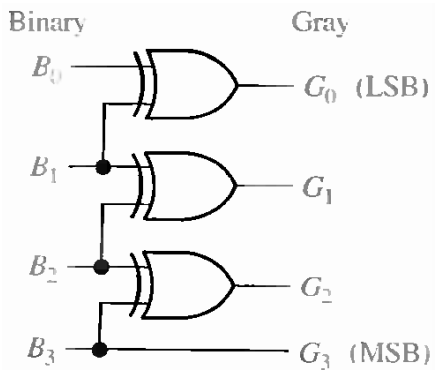
```
module TB_mux81_case;

	// Inputs
	reg en;
	reg [7:0] i;
	reg [2:0] sel;

	// Outputs
	wire y;

	// Instantiate the Unit Under Test (UUT)
	mux81_case uut (
		.en(en),
		.i(i),
		.sel(sel),
		.y(y)
	);

	initial
	begin
		// Initialize Inputs
		en = 0;
		i = 0;
		sel = 0;

		// Wait 50 ns for global reset to finish
		#50;

		en = 1;
		i = 8'b10101010;
		sel = 3'b000;


		#50 sel = 1;
		#50 sel = 2;
		#50 sel = 3;
		#50  sel = 4;
		#50; sel = 5;
		#50; sel = 6;
		#50; sel = 7;
		#50; sel = 8;
		#50;

		$finish;

		end
endmodule
```

Simulation Result:



Expt No. 1 d)

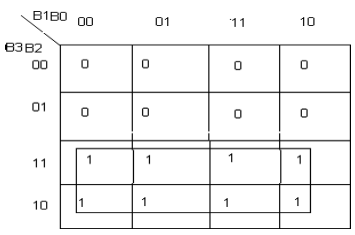
Aim: Write Verilog program for 8 to 1 multiplexer using case statement and if statementsalong with test bench to verify the design.



Truth Table:

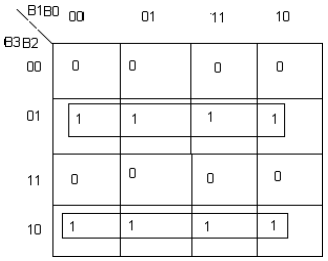
BinaryInputs(b_in)				**GrayOutputs(g_op)**			
b_in[3]	b_in[2]	b_in[1]	b_in[0]	g_op[3]	g_op[2]	g_op[1]	g_op[0]
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	1
0	1	1	1	0	1	0	1
0	1	1	0	0	1	0	0
1	0	0	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

K-MAP FOR G3:



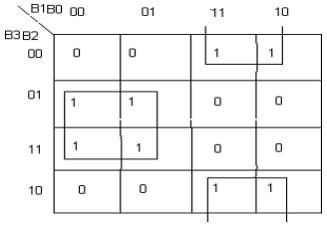
G3=B3

K-MAP FOR G2:



G2=B3B2+B3B2

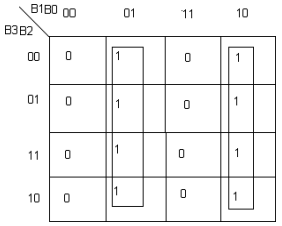
K-MAP FOR G1:



G1=B1B2+B1B2

G1= B1 XOR B2

K-MAP FOR G0:



G0=B1B0+B1B0

G0= B1 XOR B0

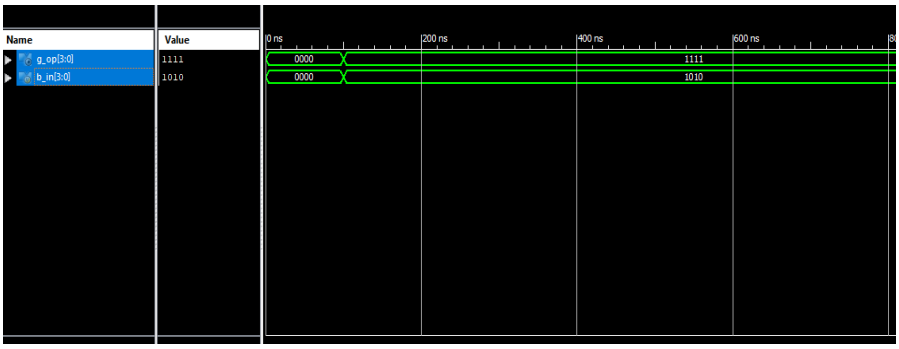
Verilog Code:

```
modulebinary_gray(b_in,g_op);

input[3:0]b_in;

output[3:0]g_op;

assign g_op[3]=b_in[3];

assign g_op[2] = b_in[3] ^ b_in[2];

assign g_op[1] = b_in[2] ^ b_in[1];

assign g_op[0]= b_in[1] ^ b_in[0];

endmodule
```

Test bench:

```
module b_g;
   reg[3:0]b_in;
  wire[3:0]g_op;
binary_gray1uut(
          .b_in(b_in),
          .g_op(g_op)
   );
   initialbegin
          b_in=0000;#100;
          b_in=0010;#100;
     end
    endmodule
```

Simulation Result:



Result: The Simulation has carried out and verified with respect to truth table.

Reflections:

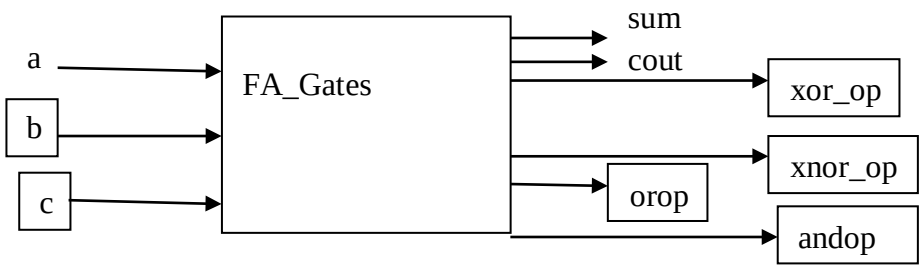
Viva Questions:

1. Write a gate level Verilog description to realize 2:4 decoder using NAND logic.
2. What is the logic function of a decoder?
3. What is the use case of multiplexer?
4. What is the purpose of the case statement in Verilog?
5. What is the difference between encoder with priority and without priority?
6. What are the applications of encoder?
7. How do you convert GREY code to binary?
8. What is the use of decoder in VHDL?

Exp.No:2. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modeled behaviour.

Aim: To model the full adder circuit along with and, or, xor and xnor functions.

Logic Symbol:



Program:

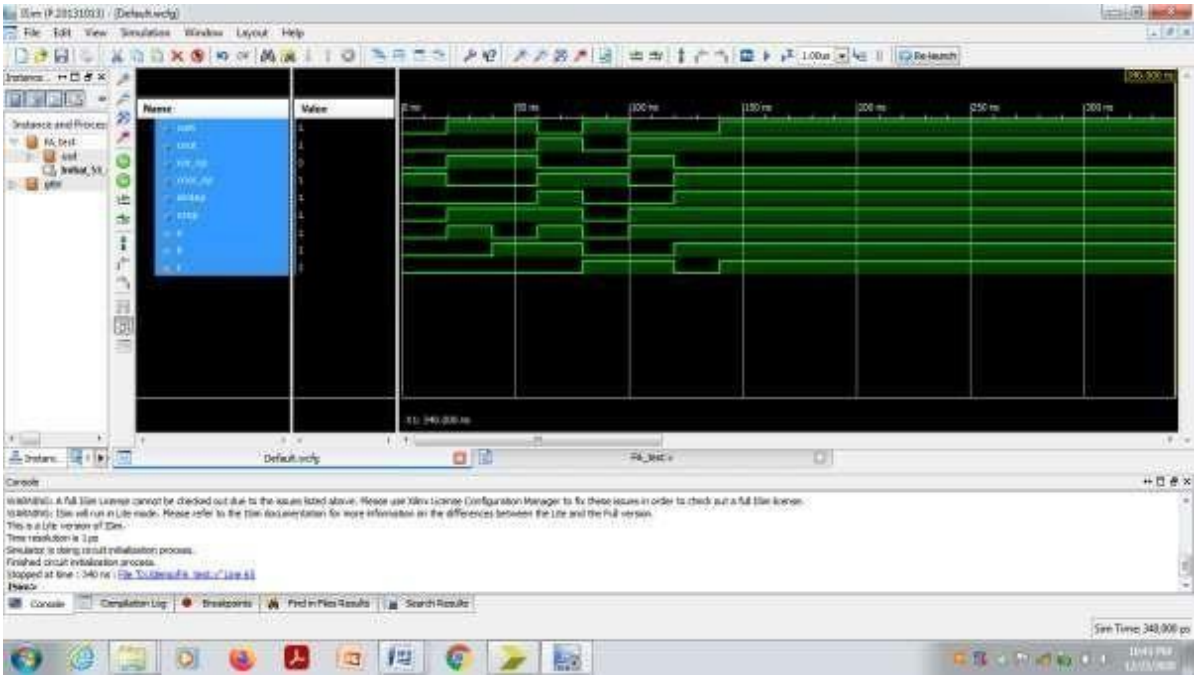
```
module FA_Gates(
   input a,b,c,
   output sum,cout,
   output xor_op, xnor_op, andop, orop
   );
assign sum= xor_op^c;
assign cout=(andop)|(b&c)|(c&a);
assign xor_op=a^b;
assign xnor_op=~xor_op;
assign andop=a&b;
assign orop=a|b;
endmodule
```

Test Bench Program:

```
‘timescale 1ns/1ps
module FA_test;

        // Inputs
        reg a;
        reg b;
        reg c;

        // Outputs
        wire sum;
        wire cout;
        wire xor_op;
        wire xnor_op;
        wire andop;
        wire orop;

        // Instantiate the Unit Under Test (UUT)
```

```
        FA_Gates uut (.a(a),.b(b), .c(c), .sum(sum), .cout(cout), .xor_op(xor_op),
                .xnor_op(xnor_op), .andop(andop),    .orop(orop));

        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;
                c = 0;
#20 a=1;b=0;c=0;
#20 a=0;b=1;c=0;
#20 a=1;b=1;c=0;
#20 a=0;b=0;c=1;
#20 a=1;b=0;c=1;
#20 a=1;b=1;c=0;
#20 a=1;b=1;c=1;
#200 $finish;
end
endmodule
```

Simulation Result:



Reflections:

Viva Questions:

1. What is the full adder model in Verilog?
2. How to implement a full adder in VHDL?
3. What is the VHDL logic for full adder?
4. How do you make a full adder using half adder in Verilog?
5. How many full and half adders are required to add 4-bit numbers?
6. What is the difference between an adder and a subtractor?

Exp.No:3:

Aim: Write a verilog program for 32-bit ALU shown in figure below and verify the functionality of ALU by selecting appropriate test patterns. The functionality of the ALU is presented in Table 1.

a. Write test bench to verify the functionality of the ALU considering all possible input patterns
b. The enable signal will set the output to required functions if enabled, if disabled all the outputs are set to tri-state
c. The acknowledge signal is set high after every operation is completed

Block Diagram:

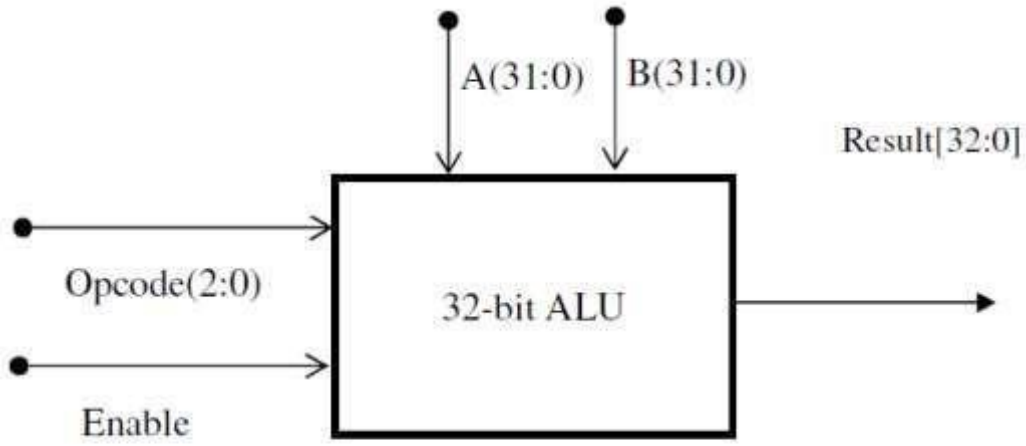


Figure 1 ALU top level block diagram

Table 1 ALU Functions

Logic Table:

Opcode(2:0)	ALU Operation	
000	A+B	Addition of two numbers
001	A-B	Subtraction of two numbers
010	A+1	Increment Accumulator by 1
011	A-1	Decrement accumulator by 1
100	A	True
101	A Complement	Complement
110	A OR B	Logic OR
111	A AND B	Logic AND

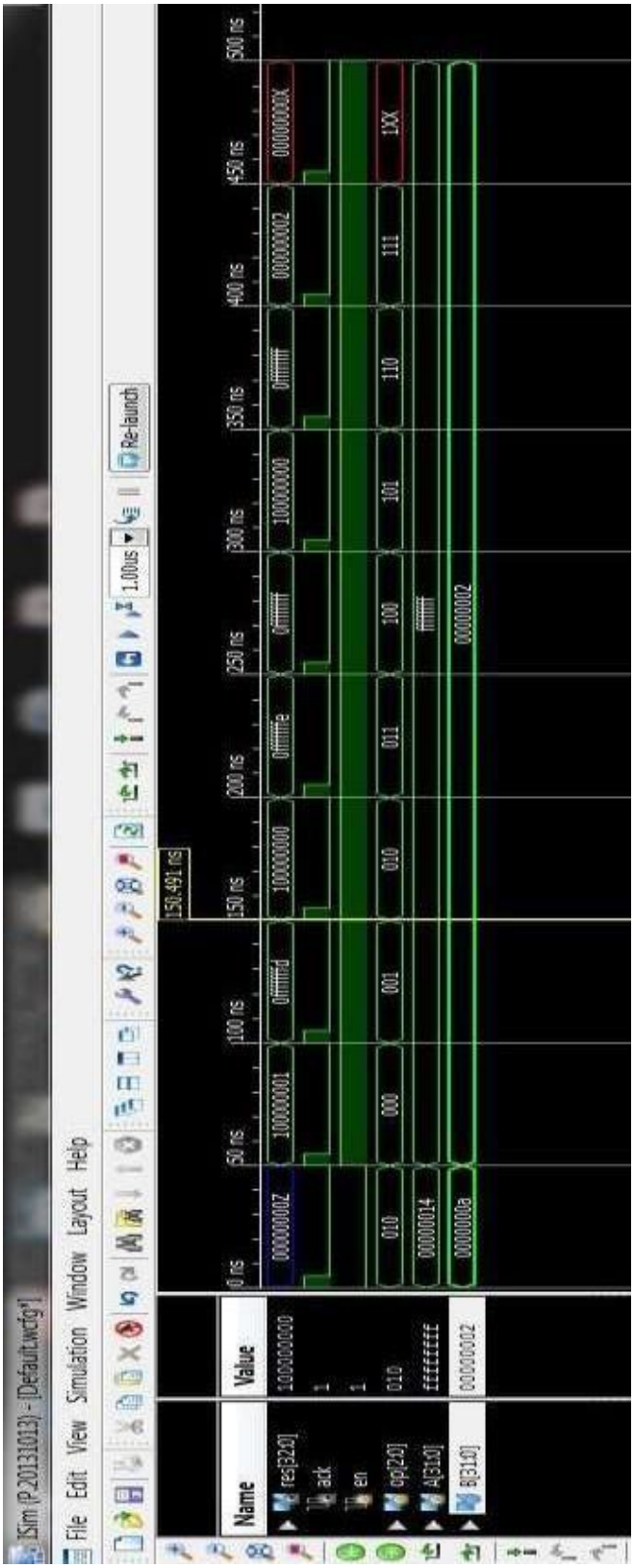
Program:

```
module ALU32(
   input en,
   input [0:2] op ,
   input [31:0] A,
   input [31:0] B,
   output reg ack,
   output reg [32:0] res
   );

always@(en,op,A,B)
begin

      if(en)

            case(op)

            3'b000: res= A+B;
            3'b001: res= A-B;
            3'b010: res= A+1;
            3'b011: res= A-1;
            3'b100: res= A;
            3'b101: res= ~A;
            3'b110: res= A|B;
            3'b111: res= A&B;
            default: res=33'hxxxxxxxx;
            endcase
      else
            res=33'hzzzzzzzz;

      ack = 1'b1;
      #5 ack =1'b0;
end
endmodule
```

Test Bench:

```
module TB_ALU32;

	// Inputs
	reg en;
	reg [2:0] op;
	reg [31:0] A;
	reg [31:0] B;

	// Outputs
	wire [32:0] res;
	wire ack;

	// Instantiate the Unit Under Test (UUT)
	ALU32 uut (
		.en(en),
		.op(op),
		.A(A),
		.B(B),
		.ack(ack),
		.res(res)
	);

	initial begin
		// Initialize Inputs
		en = 0;
		op = 02;
		A = 20;
		B = 10;

		// Wait 50 ns for global reset to finish
		#50;

		// Add stimulus here

		en = 1;
		A = 32'hffff_ffff;
		B = 32'h2;
		op = 0;
		#50;	op = 1;
		#50;	op = 2;
		#50;	op = 3;
		#50;	op = 4;
		#50;	op = 5;
		#50;	op = 6;
		#50;	op = 7;
		#50;	op = 3'b1xx;
		#50;

	 $finish;
	end
endmodule
```

Simulation Result:



Reflections:

Viva Questions:

1. What is the function of 32-bit ALU?
2. What is the basic function of ALU?
3. What are the 7 functions of ALU?
4. What are the functions of memory unit?
5. How many bits does an ALU have?

Exp.No:4: **i). Write the verilog code for SR-Flip Flop and verify**

Aim:

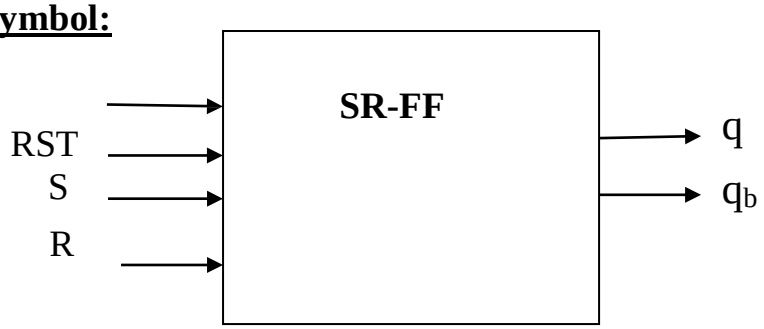
To model SR Flip Flop using verilog behavioral description and verify its function using ISE simulator as well as on the CPLD XC9572.

Tools Required:

i). Software : Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit, JTAG, Power Adapter, Flying Leads.

Logic Symbol:



Truth Table:

CLK	RST	S	R	q_{prev}	q	q_b
↑	1	X	X	X	0	1
	0	0	0	0	0	1
↑	0	0	0	1	1	0
↑	0	0	1	X	0	1
↑	0	1	0	X	1	0
↑	0	1	1	X	Indeterminate state	

Behavioral Description:

```
`timescale 1ns / 1ps
module srff(     input clk, input rst, input s, input r, output reg q, output regqb );
 reg[1:0] sr;
always@(posedge clk)
begin
if(rst==1'b1)
    q=1'b0;
 else
   begin
        sr={s,r};
```

```
        case (sr) 2'b00:
        q=q; 2'b01:
        q=1'b0;
        2'b10: q=1'b1;
        2'b11: q=1'bZ;
        default: q=1'bx;
        endcase
    end
  qb=~q;
end
endmodule
```

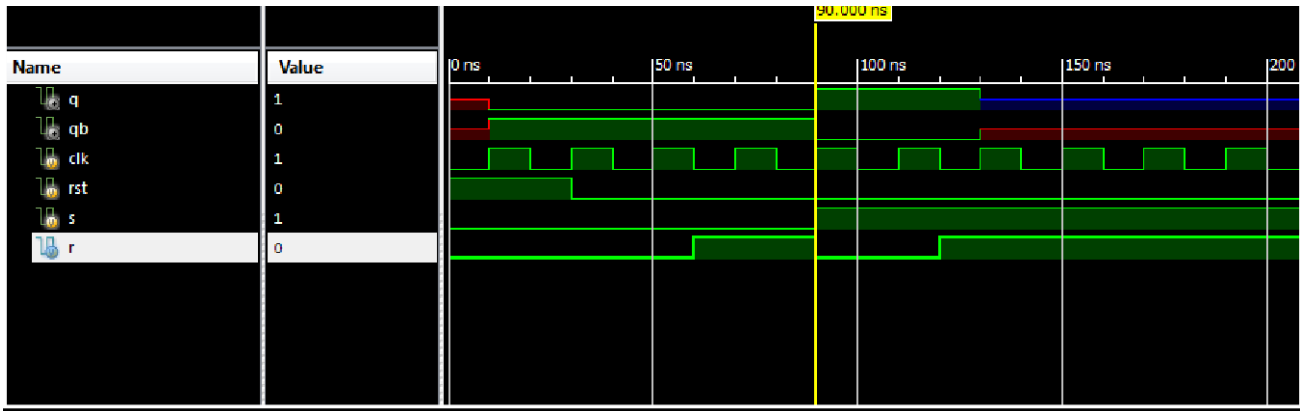
Test Bench Program:

```
`timescale 1ns / 1ps
module srff_test;
reg clk;
reg rst;
reg s;
reg r;
wire q;
wire qb;
srff uut (.clk(clk), .rst(rst), .s(s), .r(r), .q(q), .qb(qb));initial
 begin
                clk = 0;
                rst = 1;
                s = 0;
                r = 0;
                #30 s=0 ;r=0;rst=0;
                #30 s=0;r=1;
                #30 s=1;r=0;
                #30 s=1;r=1;
                #200 $finish;
   end
   always
 #10 clk=~clk;
endmodule
```

User Constraint File:

```
#PINLOCK_BEGIN
NET "clk"        LOC = "S:PIN1";
NET "rst"        LOC = "S:PIN2";
NET "s "         LOC = "S:PIN3";
NET "r "         LOC =  "S:PIN4";
NET "q"          LOC = "S:PIN5";
NET "qb"         LOC = "S:PIN6";
#PINLOCK_END
```

Simulation Result:



Exp.No:4.ii). Write the verilog code for JK-Flip Flop and verify

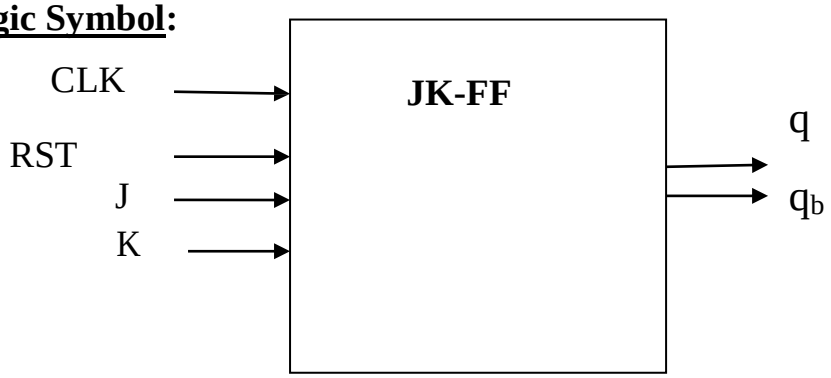
Aim:

To model JK Flip Flop using verilog behavioral description and verify its function using ISE simulator as well as on the CPLD XC9572.

Tools Required:

i). Software : Xilinx ISE14.7
ii).Hardware: XC9572 based CPLD Kit, JTAG, Power Adapter, Flying Leads.

Logic Symbol:



Truth Table:

CLK	RST	J	K	q_{prev}	q	q_b
↑	1	X	X	X	0	1
↑	0	0	0	0	0	1
↑	0	0	0	1	1	0
↑	0	0	1	X	0	1
↑	0	1	0	X	1	0
↑	0	1	1	0	1	0
↑	0	1	1	1	0	1

Behavioral Description:

```
`timescale 1ns / 1ps
module jkff(    input clk, input rst, input j, input k, output reg q, output reg qb );
 reg[1:0] jk;
 always@(posedge clk)
```

```
begin
if(rst==1'b1)
    q=1'b0;
 else
   begin
        jk={j,k};
        case (jk)
        2'b00: q=q;
        2'b01: q=1'b0;
        2'b10: q=1'b1;
        2'b11: q=~q;
        default: q=1'bx;
        endcase
   end
qb=~q;
end
endmodule
```

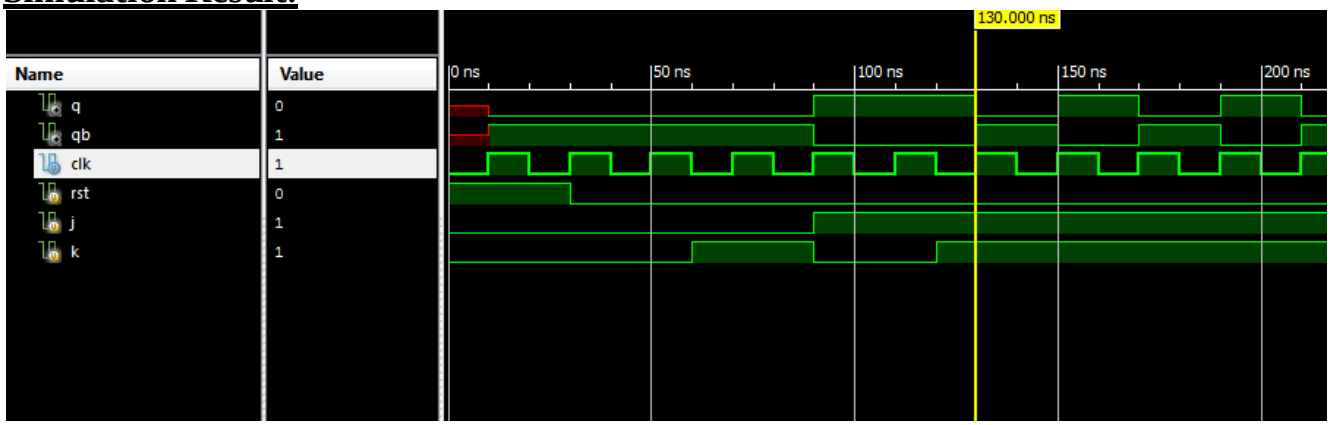
Test Bench Program:

```
`timescale 1ns / 1ps
module jkff_test;
reg clk;
reg rst;
reg j;
reg k;
wire q;
wire qb;
jkff uut (.clk(clk), .rst(rst), .j(j), .k(k), .q(q), .qb(qb));
initial
 begin
                    clk = 0;
                    rst = 1;
                    j = 0;
                    k = 0;
                    #30 j=0 ;k=0;rst=0;
                    #30j=0;k=1;
                    #30 j=1;k=0;
                    #30 j=1;k=1;
                    #200 $finish;
  end
  always
 #10 clk=~clk;

endmodule
```

User Constraint File:

```
#PINLOCK_BEGIN
NET "clk"          LOC = "S:PIN1";
NET "rst"           LOC = "S:PIN2";
NET "j "          LOC = "S:PIN3";
NET "k "          LOC = "S:PIN4";
NET "q"           LOC = "S:PIN5";
NET "qb"          LOC = "S:PIN6";
#PINLOCK_END
```

Simulation Result:



Exp.No.4.iii). Write the verilog code for D-Flip Flop and verify

Aim:

To model D_Flip Flop using verilog behavioral description and verify its function using ISE simulator as well as on the CPLD XC9572.

Tools Required:

i). Software : Xilinx ISE14.7
ii).Hardware: XC9572 based CPLD Kit, JTAG, Power Adapter, Flying Leads.

Logic Symbol:

Truth Table:

CLK	RST	D	q_{prev}	q	q_b
↑	1	x	X	0	1
↑	0	0	X	0	1
↑	0	1	X	1	0

Behavioral Description:

```
`timescale 1ns / 1ps
module dff(clk,rst,d, q,qb);
input clk,rst,d;
output q,qb;
reg q,qb;
 always@(posedge clk)
 begin
 if(rst==1'b1)
 q=1'b0;
 else
```

```
 q=d;
 qb=~q;
 end
 endmodule
```

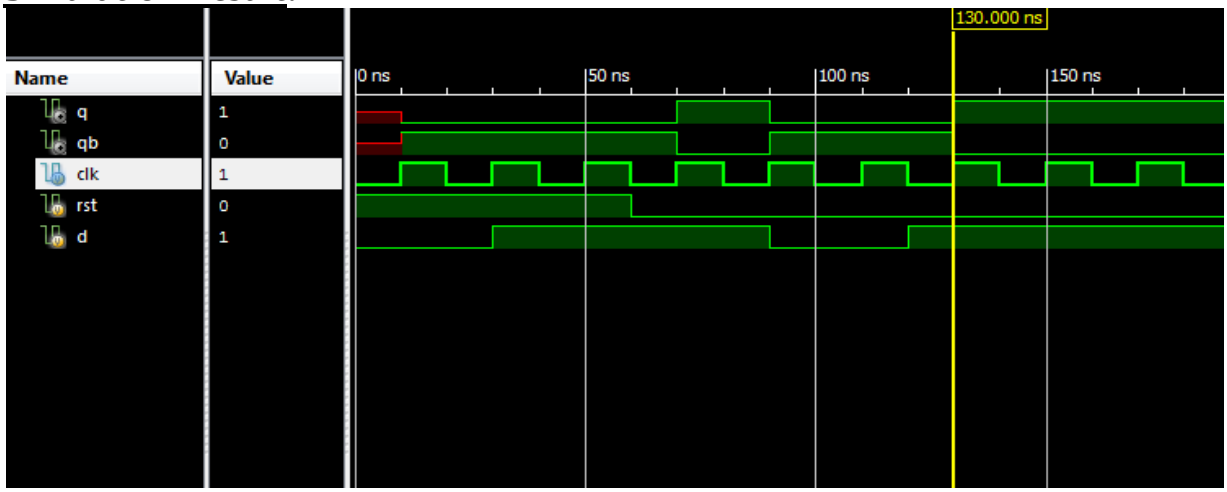
Test Bench Program:

```
`timescale 1ns / 1ps
module dFF_test;
reg clk;
reg rst;
reg d;
wire q;
wire qb;
dFF uut (.clk(clk),.rst(rst),.d(d),.q(q),.qb(qb));
initial
begin
clk = 0;
rst = 1;
d = 0;
#30 d=1;
#30 rst=0;
#30 d=0;
#30 d=1;
#100 $finish;
end
always
#10 clk=~clk;
endmodule
```

User Constraint File:

```
#PINLOCK_BEGIN
NET "clk"          LOC = "S:PIN1";
NET "rst"           LOC = "S:PIN2";
NET "d "          LOC = "S:PIN3";
NET "q"           LOC = "S:PIN4";
NET "qb"           LOC = "S:PIN5";
#PINLOCK_END
```

Simulation Result:



Reflections:

Viva Questions:

1. What is the basic principle of flip-flop?
2. What are the properties of flip-flop?
3. What is the principle of SR and JK flip flop?
4. What is the objective of D flip-flop?
5. What is JK flip flop and its types?
6. What is the logic circuit of SR, JK, D flip-flop?
7. Why is SR flip-flop unstable?
8. What is the logic circuit of SR flip-flop?

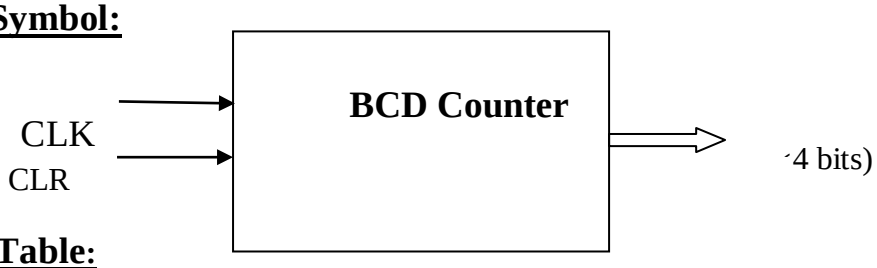
Expt.No:5. Write a Verilog code for 4-bit BCD synchronous counter andverify.

Aim: To model 4 bit Synchronous BCD counter using verilog and verify its function using ISEsimulator as well as on the CPLD XC9572.

Tools Required:

i). Software : Xilinx ISE14.7
ii).Hardware: XC9572 based CPLD Kit,JTAG,Power Adapter,Flying Leads.

Logic Symbol:



Truth Table:

CLK	CLR	Q
↑	1	0000
↑	0	0001
↑	0	0010
↑	0	0011
↑	0	0100
↑	0	0101
↑	0	0110
↑	0	0111
↑	0	1000
↑	0	1001

Behavioral Description:

```
`timescale 1ns / 1ps
module cntr( input clk, input clr, output reg[3:0] q );initial
q=4'd0;
always@(posedge clk)
begin
if(clr==1)
                        q=4'd0;
else
                        if(q==4'b1001) q=4'd0;
                        else q=q+1;


end
endmodule
```

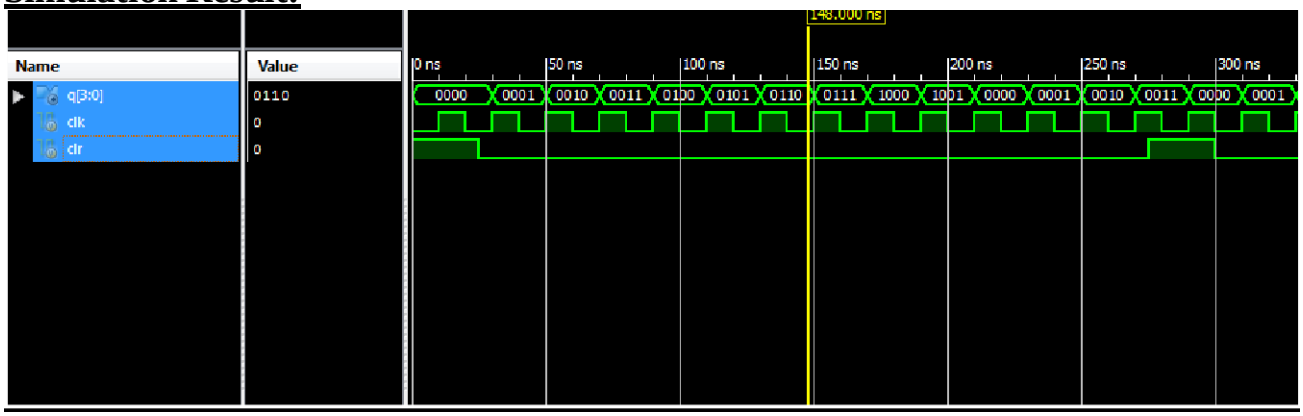
Test Bench Program:

```
`timescale 1ns / 1ps
module cntr_test; reg
clk;
reg clr;
wire [3:0] q;
cntr uut ( .clk(clk), .clr(clr), .q(q));initial
begin
          clk = 0;
          clr = 1;
           #25 clr=0;
           #250 clr=1;
           #25 clr=0;
          #450 $finish;
 end
always
         #10 clk=~clk;
endmodule
```

User Constraint File:

```
NET "clk" LOC = "P1" ;
NET "clr" LOC = "P2" ;
NET "q<0>" LOC = "P3" ;
NET "q<1>" LOC = "P4" ;
NET "q<2>" LOC = "P5" ;
 NET "q<3>" LOC = "P6" ;
```

Simulation Result:



Reflections:

Viva Questions:

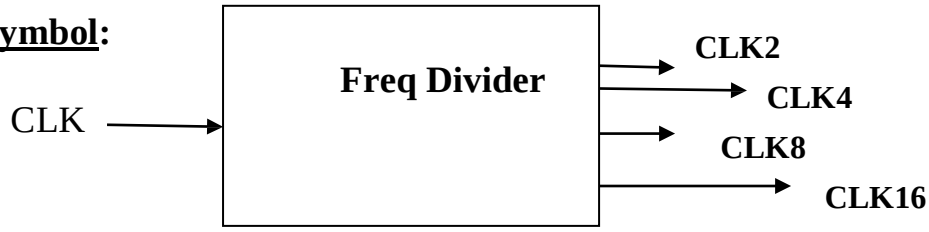
1. What Does A 4-Bit Synchronous Counter Use?
2. What Is BCD Synchronous Counter?
3. What Is A 4-Bit Counter Also Known As?
4. Is BCD Counter Synchronous Or Asynchronous?
5. What Are The Two Types Of BCD?
6. What Is BCD Function?
7. What Is BCD With Example?

Exp.No: 6. Write a Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

Aim: To model a frequency divider (frequency division by 2,4,8, and 16) using verilog and verify the same using ISE simulator as well as on the CPLD XC9572.

Tools Required: i). Software : Xilinx ISE14.7 **ii).Hardware:** XC9572 basedCPLD Kit,JTAG,Power Adapter,Flying Leads.

Logic Symbol:



Concept: On every rising edge of the clock, temp variable is incremented by 1.

clk	Temp[3]	Temp[2]	Temp[1]	Temp[0]
	0	0	0	0
↑	0	0	0	1
↑	0	0	1	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	0	1
↑	0	1	1	0
↑	0	1	1	1
↑	1	0	0	0
↑	1	0	0	1
	1	0	1	0
↑	1	0	1	1
↑	1	1	0	0
↑	1	1	0	1
↑	1	1	1	0
↑	1	1	1	1

clk2=temp[0]
Clk4=temp[1]
Clk8=temp[2]
Clk16=temp[3]

Behavioral Description:

```
`timescale 1ns / 1ps
module clkdiv( input clk,output reg clk2, output reg clk4, output reg clk8,output reg clk16 );
reg[3:0] temp=4'd0;
 always@(posedge clk)
 begin
 temp=temp+1;
 clk2=temp[0];
 clk4=temp[1];
 clk8=temp[2];
 clk16=temp[3];
 end
 endmodule
```

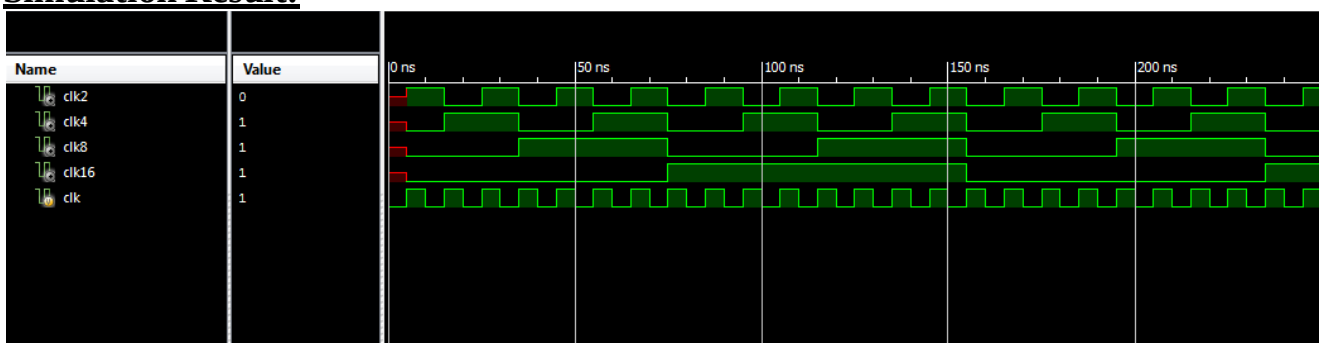
Test Bench Program:

```
`timescale  1ns  /  1ps
module  clkdiv_test; reg
clk;
wire       clk2;
wire       clk4;
wire       clk8;
wire clk16;
clkdiv uut (.clk(clk), .clk2(clk2), .clk4(clk4), .clk8(clk8), .clk16(clk16));
initial
begin
 clk = 0;
#300 $finish;
end
always
#5 clk=~clk;
endmodule
```

User Constraint File:

```
NET "clk" LOC = "P1" ;
NET "clk2" LOC = "P2" ;
NET "clk4" LOC = "P3" ;
NET "clk8" LOC = "P4" ;
NET "clk16" LOC = "P5"  ;
```

Simulation Result:



Reflections:

Viva Questions:

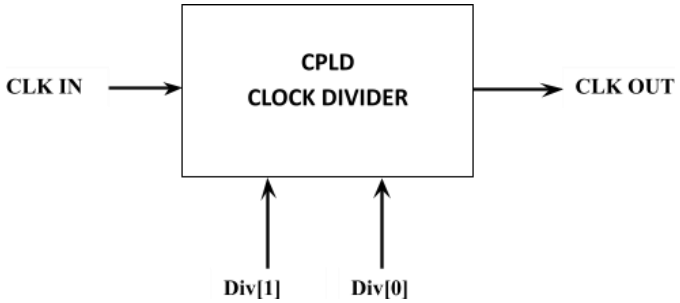
1. What is a clock divider in Verilog?
2. How do you implement a clock divider?
3. How do you make a binary counter?
4. What is another name for binary counter?
5. Why do we use binary counter?
6. How do you stop a binary counter?

Part –B

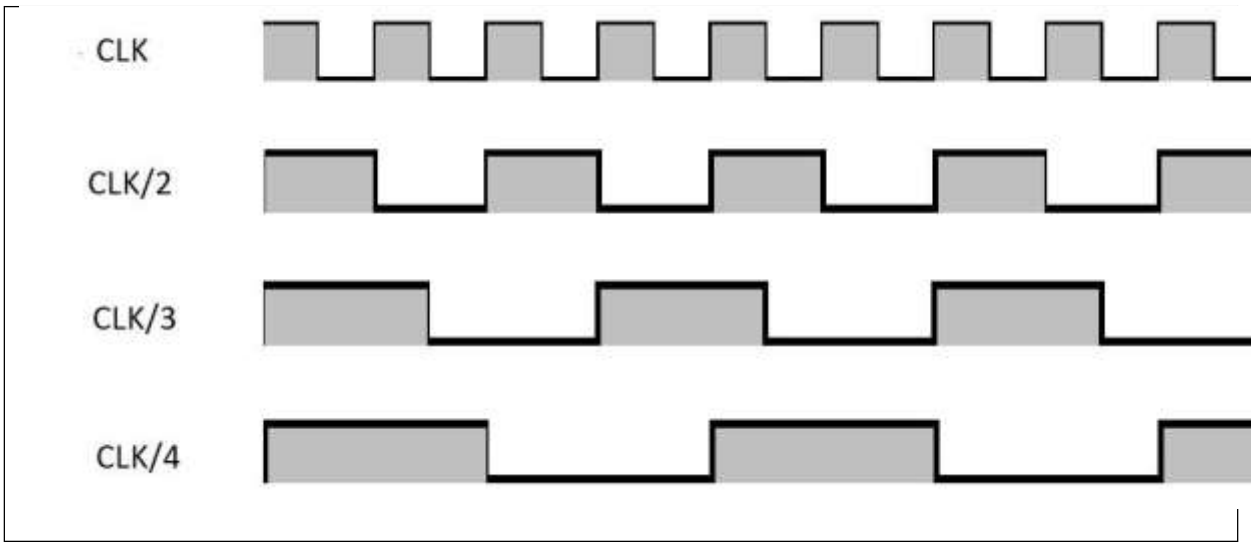
Expt No. 7

Aim: To write a Verilog code to realize a clock divider circuit that generates 1/2, 1/3rd and 1/4th clock from a given input clock using FPGA/CPLD and validate the functionality through oscilloscope.

Block Diagram



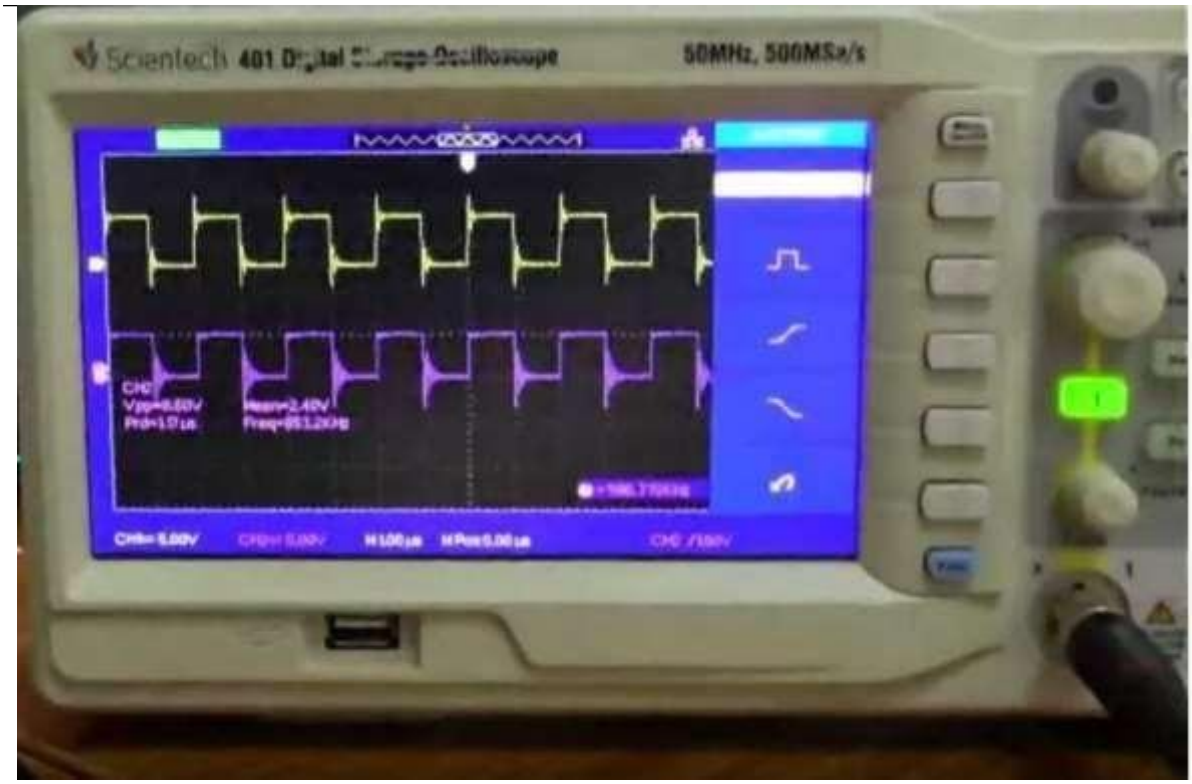
Waveform:



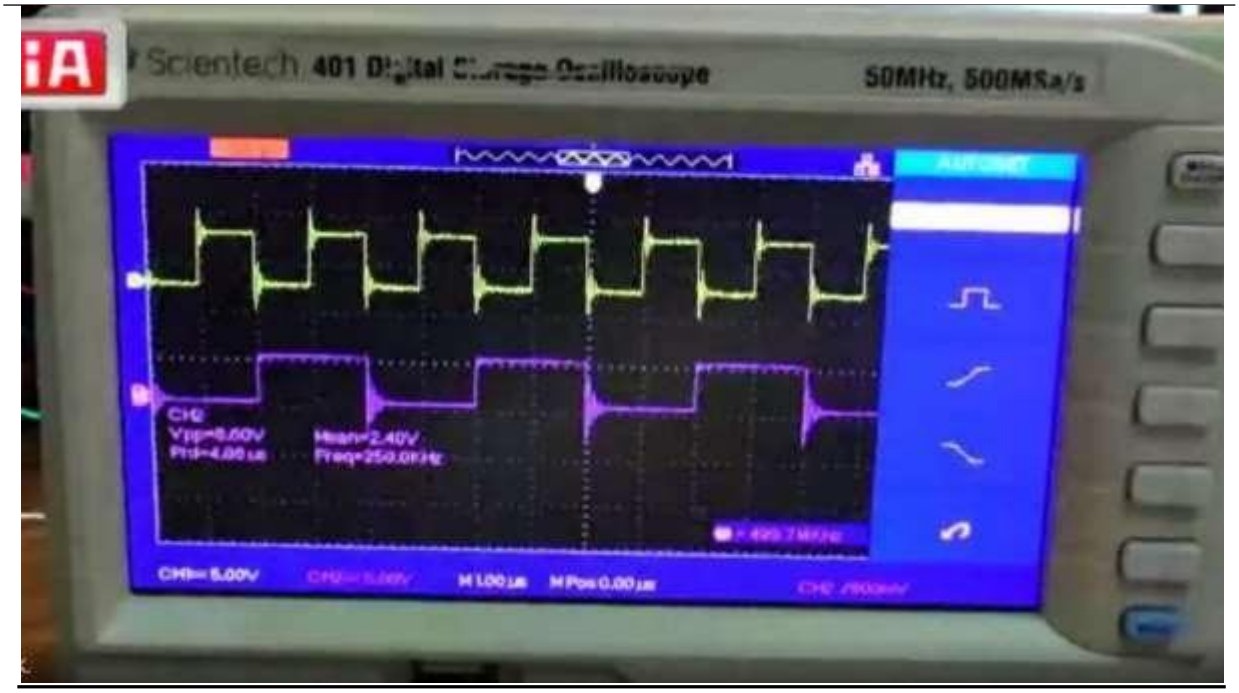
Program:

```
module CLK_DIVIDER(
   input CLK_IN,
   output reg CLK_OUT = 0,
   input [1:0] DIVISOR
   );
reg [1:0] counter=0;

always@(CLK_IN)
begin
        if(counter != DIVISOR)
                counter = counter + 1;
        else
                begin
                CLK_OUT = ~CLK_OUT;
          counter = 0;
                end
        end
endmodule
```

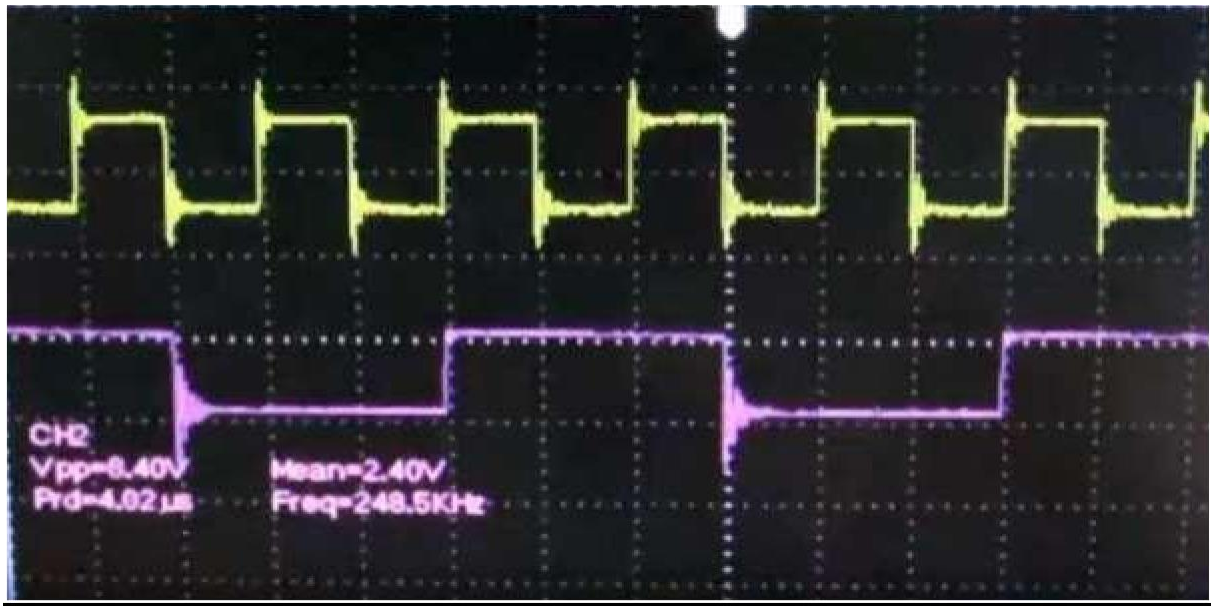
OUTPUT RESULTS:



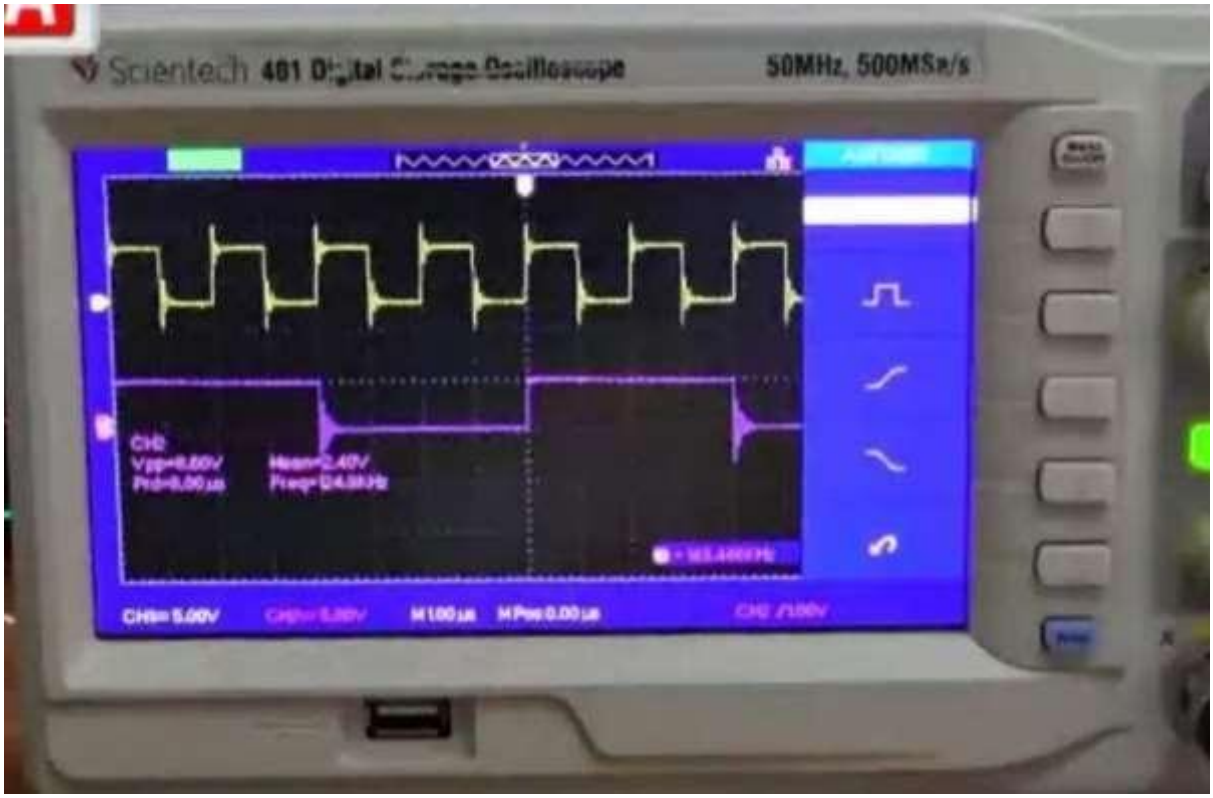
Output on Oscilloscope for DIVISOR=00, (input clock divide by 1)



Output on Oscilloscope for DIVISOR=01, (input clock divide by 2)



Output on Oscilloscope for DIVISOR=10, (input clock divide by 3)



Output on Oscilloscope for DIVISOR=11, (input clock divide by 4)

Reflections:

Viva Questions:

1. What is a clock divider FPGA?
2. How to divide a clock by 2 in VHDL?
3. How to generate a clock signal in VHDL?
4. What is the output of a clock divider?
5. How many clocks does a FPGA have?

Expt No. 8

Interface a DC motor to FPGA and write Verilog code to change its speed and direction.

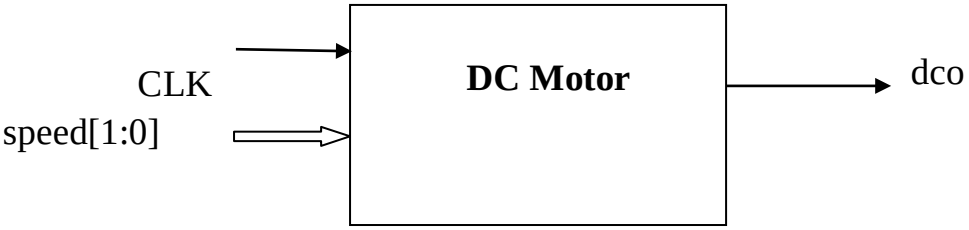
Aim: To Interface a DC motor to FPGA/CPLD and write Verilog code tochange its speed and direction.

Tools Required:

i). Software : Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit, JTAG, Power Adapter, Flying Leads, dc motorinterfacing board with motor.

Logic Symbol:



Program:

```
`timescale 1ns / 1ps
module dco(    input clk, input [1:0] speed, output reg dco );
reg[11:0] clkdiv=12'd0;

always@(posedge clk)
begin
clkdiv=clkdiv+1;
if(clkdiv==12'd3000)
clkdiv=12'd0;
end

always@(speed)
begin
if(clkdiv==12'd0)
dco=1'b1;
else if(speed==2'd0 && clkdiv==12'd700)
dco=1'b0;
else if (speed==2'd1 && clkdiv==12'd1400)
dco=1'b0;
else if (speed==2'd2 && clkdiv==12'd2100)
dco=1'b0;
else if(speed==2'd3 && clkdiv==12'd2800)
dco=1'b0;
end
endmodule
```

User Constraint File:

```
NET "clk" LOC = "P1" ;
NET "dco" LOC = "P4" ;
NET "speed<0>" LOC = "P2" ;
NET "speed<1>" LOC = "P3"  ;
```

Expected Result:

1. By varying the speed signal from 00 to 11, the speed of rotation can bechanged.
2. The dc output can be given to in1 and in2 connection points for changingthe direction of rotation.

Reflection:

Viva Questions:

1. What is the difference between VHDL and FPGA?
2. What is the difference between CPLD and FPGA in Verilog?
3. What are the features of CPLD?
4. Is Verilog used in FPGA?

Expt No.9

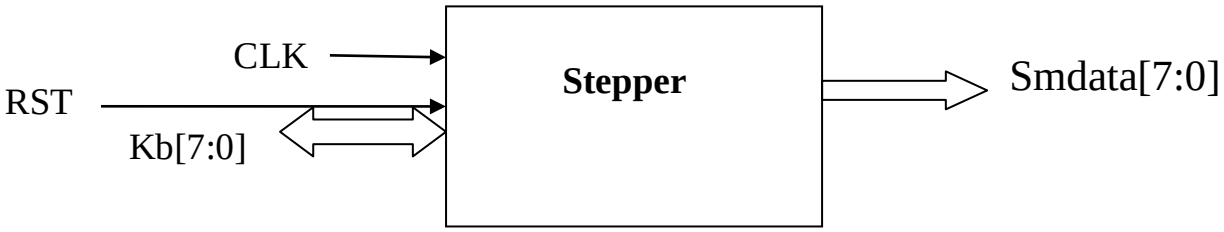
Aim: To Interface a Stepper motor to FPGA and write Verilog code to control the Stepper motor rotation which in turn may control a Robotic Arm. External switches to be used for different controls like rotate the Stepper motor (i) +N steps if Switch no.1 of a Dip switch is closed (ii) +N/2 steps if Switch no. 2 of a Dip switch is closed (iii) –N steps if Switch no. 3 of a Dip switch is closed etc.

Tools Required:

i). Software : Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit,JTAG, Power Adapter, Flying Leads ,stepper motor interfacing board with motor.

Logic Symbol:



Algorithm:

1. Declare the external ports and internal objects.
2. Perform frequency division to control the speed of rotation
3. Check for the reset status. If reset is high, resets all the step control variables tozero else go to step 4.
4. Read the status of switch1 .If it is pressed, send the data to stepper motor coilin such a way that it will be moving in forward direction for N steps else go tostep 5.
5. Read the status of switch2 .If it is pressed, send the data to stepper motor coilin such a way that it will be moving in reverse direction for N steps else go to step 6.
6. Read the status of switch3.If it is pressed; send the data to the stepper motor coil in such a way that it will be moving in forward direction for (N/2) steps elsestop.

[Note: To energize the coils of the stepper motor, send a 4 bit data 0001,0010, 0100 and 1000 in this sequence for a forward direction and send these data in a reverse order for anticlockwise direction rotation].

Program:

```
`timescale 1ns / 1ps
module stepper(     input clk,rst, inout [7:0] kb, output reg[7:0] smdata);reg
 tclk;
 reg[15:0] clkdiv=16'd0;
 reg[1:0]sts=2'd0;
 reg[3:0]coil=4'b00001;
 reg[7:0] i1=8'd0;
 reg[7:0]  i2=8'd0;
reg[7:0] i3=8'd0;
```

```
reg[3:0] N=4'd15;
reg[2:0] stdir;

assign kb[7:3]=5'b00001;

always@(posedge clk)
begin
clkdiv=clkdiv+1;
tclk=clkdiv[15];
end

always@(posedge tclk)
begin
if(rst==1)
begin
        i1=0;
        i2=0;
        i3=0;
end
case (kb[2:0])
3'b110:
        if(i1!=N)
        begin
        sts=sts+1;
        i1=i1+1;
        end
3'b101:
        if(i2!=N)
        begin
        sts=sts-1;
        i2=i2+1;
        end
3'b011:
        if(i3!=(N/2))
        begin
        sts=sts+1;
        i3=i3+1;
        end
endcase
end

always@(sts)
begin
```

```
case(sts)
        2'b00: coil=4'b0001;
        2'b01: coil=4'b0010;
        2'b10: coil=4'b0100;
        2'b11: coil=4'b1000;
        default: coil=4'b0001;
endcase
smdata={4'b0000,coil};
end
endmodule
```

User Constraint File:

```
NET "clk" LOC = "P53" ;
NET "kb<0>" LOC = "P1" ;
NET "kb<1>" LOC = "P2" ;
NET "kb<2>" LOC = "P3" ;
NET "kb<3>" LOC = "P4" ;
NET "kb<4>" LOC = "P5" ;
NET "kb<5>" LOC = "P6" ;
NET "kb<6>" LOC = "P7" ;
NET "kb<7>" LOC = "P9" ;
NET "rst" LOC = "P54" ;
NET "smdata<0>" LOC = "P10" ;
NET "smdata<1>" LOC = "P11" ;
NET "smdata<2>" LOC = "P12" ;
NET "smdata<3>" LOC = "P13" ;
NET "smdata<4>" LOC = "P14" ;
NET "smdata<5>" LOC = "P15" ;
NET "smdata<6>" LOC = "P17" ;
NET "smdata<7>" LOC = "P18" ;
```

Expected Result:

1. Set the number of required steps (N) in the program.
2. By pressing sw1 of 4x4 hexa keypad, stepper motor will rotate inclockwise direction for N steps.
3. By pressing sw2 of 4x4 hexa keypad, stepper motor will rotate in anticlockwise direction for N steps.
4. By pressing sw3 of 4x4 hexa keypad, stepper motor will rotate inclockwise direction for (N/2) steps.

Expt No.10:

Aim: Interface a DAC to FPGA and write Verilog code to generate a sine wave of frequency f KHz, ex f=100 KHz, or 200 KHz etc, .Modify the code to down sample the frequency to f/2 KHz. Display the original and down sampled signals by connecting them to CRO.

Tools Required: i). Software: Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit

Program:

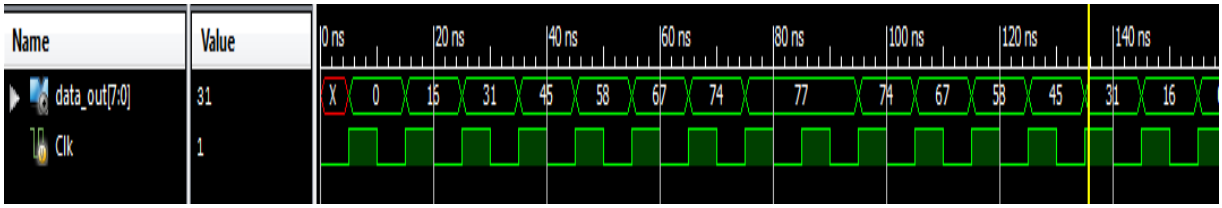
```
module  sine_wave_gen(Clk,data_out);
//declare input and output
   input Clk;
   output [7:0] data_out;
//declare the sine ROM - 30 registers each 8 bit wide.
   reg [7:0] sine [0:29];
//Internal signals
   integer i;
   reg [7:0] data_out;
//Initialize the sine rom with samples.
   initial begin
      i = 0;
      sine[0] = 0;
      sine[1] = 16;
      sine[2] = 31;
      sine[3] = 45;
      sine[4] = 58;
      sine[5] = 67;
      sine[6] = 74;
      sine[7] = 77;
      sine[8] = 77;
      sine[9] = 74;
      sine[10] = 67;
      sine[11] = 58;
      sine[12] = 45;
      sine[13] = 31;
      sine[14] = 16;
      sine[15] = 0;
      sine[16] = -16;
      sine[17] = -31;
      sine[18] = -45;
      sine[19] = -58;
      sine[20] = -67;
      sine[21] = -74;
      sine[22] = -77;
      sine[23] = -77;
      sine[24] = -74;
      sine[25] = -67;
      sine[26] = -58;
      sine[27] = -45;
```

```
        sine[28] = -31;
        sine[29] = -16;
    end

    //At every positive edge of the clock, output a sine wave sample.
    always@ (posedge(Clk))
    begin
        data_out = sine[i];
        i = i+ 1;
        if(i == 29)
            i = 0;
    end

endmodule
```

Testbench:

```
module tb;

    // Inputs
    reg Clk;

    // Outputs
    wire [7:0] data_out;

    // Instantiate the Unit Under Test (UUT)
    sine_wave_gen uut ( .Clk(Clk),  .data_out(data_out)  );

    //Generate a clock with 10 ns clock period.
    initial Clk = 0;
    always #5 Clk = ~Clk;
    endmodule
```

Waveform:



Expt No. 11:

Aim: To write Verilog code using FSM to simulate elevator operation.

Tools Required: i). Software: Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit(with Hexa keypad and 7 segment display), JTAG,Power Adapter, Flying Leads, FRCs**.**

Program:

```
`timescale 1ns / 1ps
module ele(input clk,input[3:0] kycol,
output reg [3:0] kyrow,output reg [3:0] dismux,output reg [7:0] disseg);
reg [3:0] kyflr=4'd0;
reg [3:0] curflr=4'd0;
reg [15:0] clkdiv=16'd0;
reg sclk,flrclk;
reg keyhit;
always @(posedge clk)
begin
clkdiv=clkdiv+1;
sclk=clkdiv[7];
flrclk=clkdiv[15];
end
always@(posedge sclk)
begin
case(kyrow)
4'b1110: kyrow=4'b1101;
4'b1101: kyrow=4'b1011;
4'b1011: kyrow=4'b0111;
4'b0111: kyrow=4'b1110;
default: kyrow=4'b1110;
endcase
end
always@(kycol)
begin
case(kycol)
4'b1110,4'b1101,4'b1011,4'b0111: keyhit=1'b1;
default: keyhit=1'b0;
disseg[7:0]
kyrow[3:0]
dismux[3:0]
endcase
end
always@(keyhit)
begin
if(keyhit==1'b1)
if(kyrow==4'b1110 && kycol==4'b1110)
kyflr=4'd0;
```

```
else if(kyrow==4'b1110 && kycol==4'b1101)
kyflr=4'd1;
else if(kyrow==4'b1110 && kycol==4'b1011)
kyflr=4'd2;
else if(kyrow==4'b1110 && kycol==4'b0111)
kyflr=4'd3;
else if(kyrow==4'b1101 && kycol==4'b1110)
kyflr=4'd4;
else if(kyrow==4'b1101 && kycol==4'b1101)
kyflr=4'd5;
else if(kyrow==4'b1101 && kycol==4'b1011)
kyflr=4'd6;
else if(kyrow==4'b1101 && kycol==4'b0111)
kyflr=4'd7;
else if(kyrow==4'b1011 && kycol==4'b1110)
kyflr=4'd8;
else if(kyrow==4'b1011 && kycol==4'b1101)
kyflr=4'd9;
else if(kyrow==4'b1011 && kycol==4'b1011)
kyflr=4'd10;
else if(kyrow==4'b1011 && kycol==4'b0111)
kyflr=4'd11;
else if(kyrow==4'b0111 && kycol==4'b1110)
kyflr=4'd12;
else if(kyrow==4'b0111 && kycol==4'b1101)
kyflr=4'd13;
else if(kyrow==4'b0111 && kycol==4'b1011)
kyflr=4'd14;
else if(kyrow==4'b0111 && kycol==4'b0111)
kyflr=4'd15;
else
kyflr=kyflr;
```

```
end
always@(posedge flrclk)
begin
if(kyflr>curflr)
curflr=curflr+1;
else if (kyflr<curflr)
curflr=curflr-1;
else
curflr=curflr;
end
always@(posedge flrclk)
begin
dismux=4'b1110;
case (curflr)
```

```
4'd0: disseg=8'd63;
4'd1: disseg=8'd6;
4'd2: disseg=8'd91;
4'd3: disseg=8'd79;
4'd4: disseg=8'd102;
4'd5: disseg=8'd109;
4'd6: disseg=8'd125;
4'd7: disseg=8'd7;
4'd8: disseg=8'd127;
4'd9: disseg=8'd111;
4'd10: disseg=8'd191;
4'd11: disseg=8'd124;
4'd12: disseg=8'd88;
4'd13: disseg=8'd94;
4'd14: disseg=8'd121;
4'd15: disseg=8'd113;
default: disseg=8'd63;
endcase
end
endmodule
```

User Constraint File:

```
NET "clk" LOC = "P55" ;
NET "dismux<0>" LOC = "P14" ;
NET "dismux<1>" LOC = "P15" ;
NET "dismux<2>" LOC = "P17" ;
NET "dismux<3>" LOC = "P18" ;
NET "disseg<0>" LOC = "P31" ;
NET "disseg<1>" LOC = "P32" ;
NET "disseg<2>" LOC = "P33" ;
NET "disseg<3>" LOC = "P34" ;
NET "disseg<4>" LOC = "P35" ;
NET "disseg<5>" LOC = "P36" ;
NET "disseg<6>" LOC = "P37" ;
NET "disseg<7>" LOC = "P39" ;
NET "kycol<0>" LOC = "P1" ;
NET "kycol<1>" LOC = "P2" ;
NET "kycol<2>" LOC = "P3" ;
NET "kycol<3>" LOC = "P4" ;
NET "kyrow<0>" LOC = "P5" ;
NET "kyrow<1>" LOC = "P6" ;
NET "kyrow<2>" LOC = "P7" ;
NET "kyrow<3>" LOC = "P9" ;
```

Expected Result:

1. Hex key pad and 7 segment display are used to simulate the elevator experiment.
2. To specify the floor number to which we would like to go, that number should be pressed in the hex key pad.
3. The transition from current floor to next floor (in steps) should be displayed in the 7 Segment display.

Reflection:

Viva Questions:

1. What is FSM logic in Verilog?
2. Which of the following represents the correct way to write an FSM code in Verilog?
3. What is FSM pattern detector in Verilog?
4. What are the different types of FSM?
5. What is FSM in compiler design?
6. What are limitations of FSM?

Expt No. 12

Aim: Write Verilog code to convert an analog input signal of a sensor to digital form and to display the same on a suitable display like simple set of LEDs, 7 segment display digits or LCD display.

Tools Required: **i) Software:** Xilinx ISE14.7
ii) Hardware: XC9572 based CPLD Kit (with Hexa keypad and 7 segment display), JTAG, Power Adapter, Flying Leads, and FRCs**.**

Program:

```
module seg7(
    input [3:0]data_in,
    output reg [7:0]display_out,
    output [7:0]AN
    );

assign AN = 8'b11111110;

always @(*) begin
    case (data_in)
        4'b0000:
            display_out = 8'b00000011;  //a,b,c,d,e,f,g,dot (zero)
        4'b0001:
            display_out = 8'b10011111;  //one
        4'b0010:
            display_out = 8'b00100101;  //two
        4'b0011:
            display_out = 8'b00001101;  //three
        4'b0100:
            display_out = 8'b10011001;  //four
        4'b0101:
            display_out = 8'b01001001;  //five
        4'b0110:
            display_out = 8'b01000001;  //six
        4'b0111:
            display_out = 8'b00011111;  //seven
        4'b1000:
            display_out = 8'b00000001;  //eight
        4'b1001:
            display_out = 8'b00001001;  //nine
        4'b1010:
            display_out = 8'b00010001;  //A
        4'b1011:
            display_out = 8'b11000001;  //b
        4'b1100:
```

```
            display_out = 8'b01100011;  //C
        4'b1101:
            display_out = 8'b10000101;  //d
        4'b1110:
            display_out = 8'b01100001;  //E
        4'b1111:
            display_out = 8'b01110001;  //F
    endcase
end
endmodule
```

Result: The analog to digital conversion is observed and the values are displayed in segment.

Reflections:

Viva Questions:

1. What Is ADC?
2. How many types of adc?
3. Difference between ADC and DAC?
4. A 4-Bit R/2R Digital-To-Analog (DAC) Converter Has A Reference Of 5 Volts. What is the analog output for the input code 0101?
5. What is the resolution of a Digital-To-Analog Converter (DAC)?
6. The practical use of binary-weighted Digital-To-Analog Converters is limited to.
7. Which is not an Analog-To-Digital (ADC) Conversion Error?