ATRIA INSTITUTE OF TECHNOLOGY

(Affiliated To Visvesvaraya Technological University, Belgaum)

Anandanagar, Bangalore-24

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

HDL LAB MANUAL

5th SEMESTER ELECTRONICS AND COMMUNICATION

SUBJECT CODE: 18ECL58

2020-21

(Affiliated To Visvesvaraya Technological University, Belgaum)
Anandanagar, Bangalore-24

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

HDL LAB MANUAL

The HDL Laboratory Manual pertaining V semester ECE has been prepared as per VTU syllabus and all the experiments are designed, tested and verified according to the experiment list.

This manual typically contains practical/lab sessions related to Verilog HDL and interfacing various hardware devices with CPLD XC9572 which provides a better understanding of the subject. Students are advised to thoroughly go through this manual as it provides them practical insights.

SYLLABUS

Laboratory Code	**18ECL58**	**CIE Marks**	**40**
Number of Lecture Hours/Week	**02Hr Tutorial (Instructions)+ 02 Hours Laboratory**	**SEE Marks**	**60**
RBT Level	**L1, L2, L3**	**Exam Hours**	**03**
CREDITS – 02			

Course Learning Objectives: This course will enable students to:

- Familiarize with the CAD tool to write HDL programs.
- Understand simulation and synthesis of digital design.
- Program FPGAs/CPLDs to synthesize the digital designs.
- Interface hardware to programmable ICs through I/O ports.
- Choose either Verilog or VHDL for a given Abstraction level.

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 analyzer apart from verification by simulation with tools such as Altera/Modelsim or equivalent.

Laboratory Experiments

PART A : Programming

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 and without priority (behavioural model)
   c. 8 to 1 multiplexer using case statement and if statements
   d. 4-bit binary to gray 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 modeled behaviour.

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 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

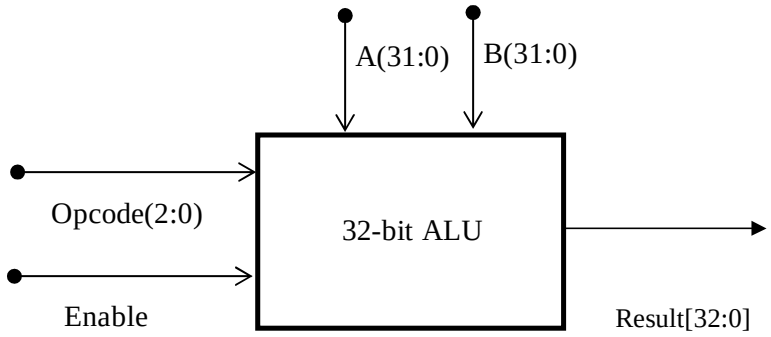


Figure 1 ALU top level block diagram

Opcode(2:0)	ALU Operation	Remarks	
000	A + B	Addition of two numbers	Both A and B are in two's complement format
001	A – B	Subtraction of two numbers	
010	A + 1	Increment Accumulator by 1	A is in two's complement format
011	A - 1	Decrement accumulator by 1	
100	A	True	Inputs can be in any format
101	A Complement	Complement	
110	A OR B	Logical OR	
111	A AND B	Logical AND	

Table 1 ALU Functions

4. Write Verilog code for SR, D and JK and verify the flip flop.

5. Write Verilog code for 4-bit BCD synchronous counter.

6. 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.

PART-B : Interfacing and Debugging (EDWinXP, PSpice, MultiSim, Proteus, CircuitLab or any other equivalent tool can be used)

1. Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to FPGA and validate the functionality through oscilloscope.

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

3. 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.

4. Interface a DAC to FPGA and write Verilog code to generate Sine wave of frequency F KHz (eg. 200 KHz) frequency. Modify the code to down sample the frequency to F/2 KHz. Display the Original and Down sampled signals by connecting them to an oscilloscope.

5. Write Verilog code using FSM to simulate elevator operation.

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

Course Outcomes: At the end of this course, students should be able to:

- Write the Verilog/VHDL programs to simulate Combinational circuits in Dataflow, Behavioral and Gate level Abstractions.
- Describe sequential circuits like flip flops and counters in Behavioral description and obtain simulation waveforms.
- Synthesize Combinational and Sequential circuits on programmable ICs and test the hardware.
- Interface the hardware to the programmable chips and obtain the required output

Conduct of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
- Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero.

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

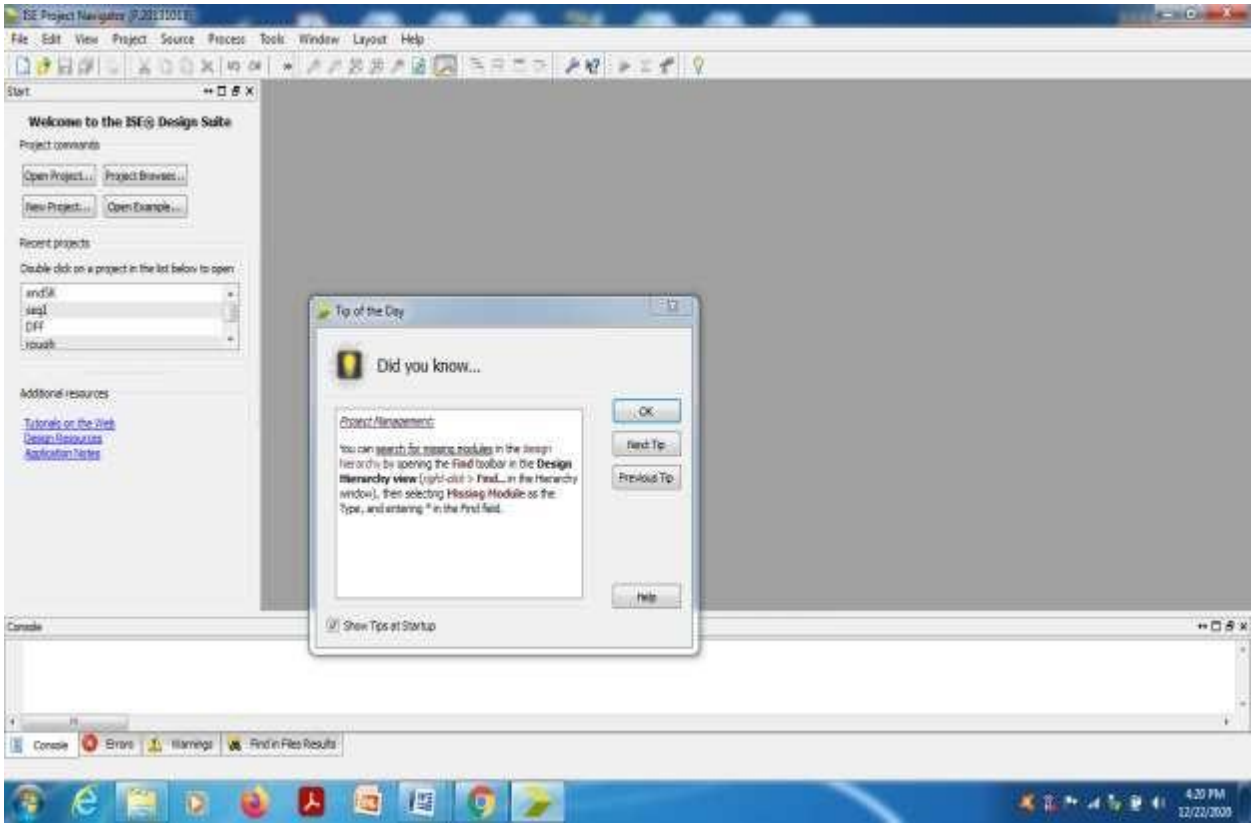
LIST OF EXPERIMENTS

PROCEDURE

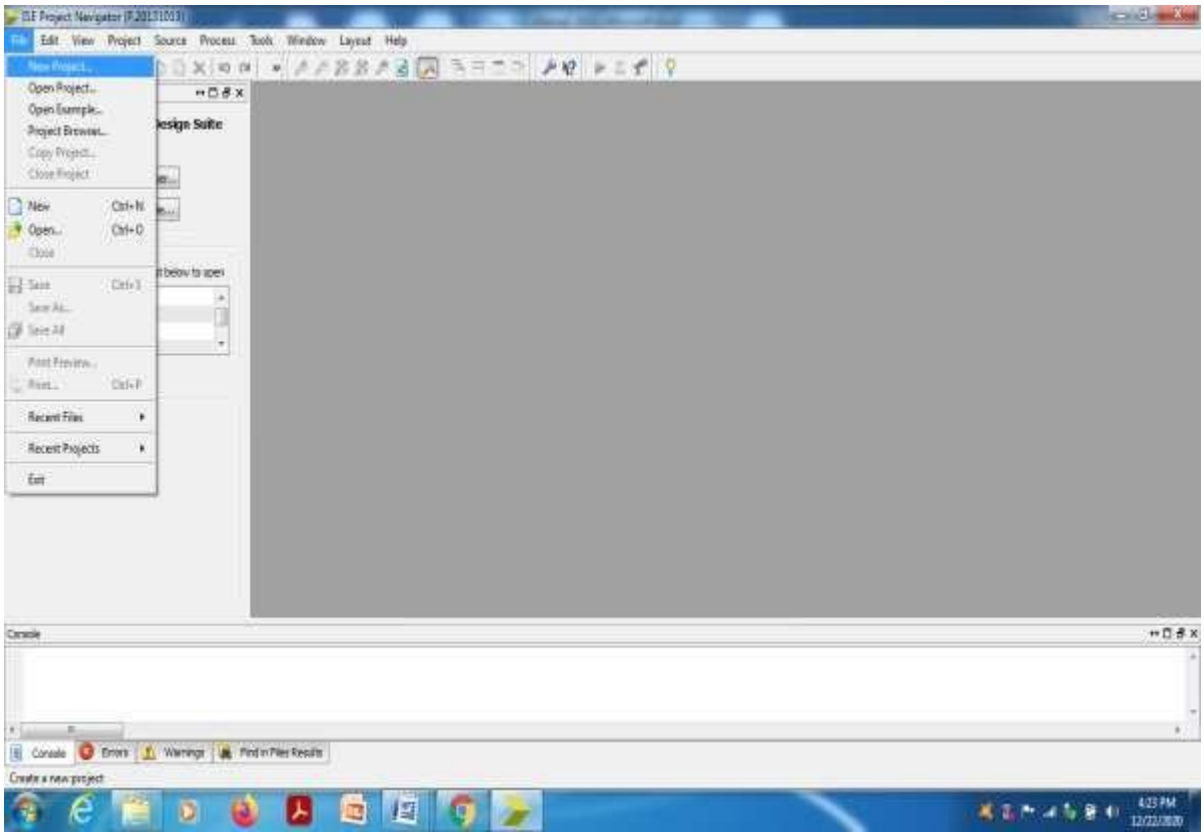
Procedure to work with Xilinx ISE 14.7 software:

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.

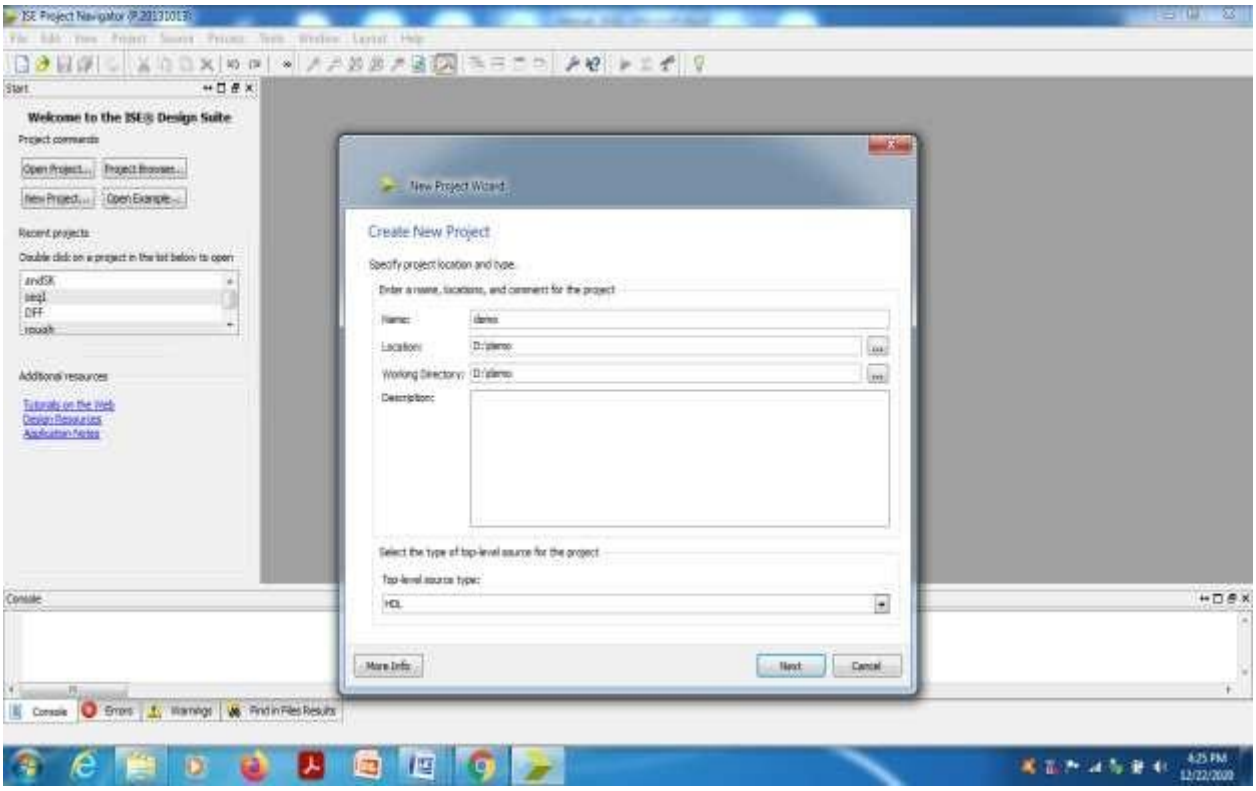
   Open XilinxISE Window.



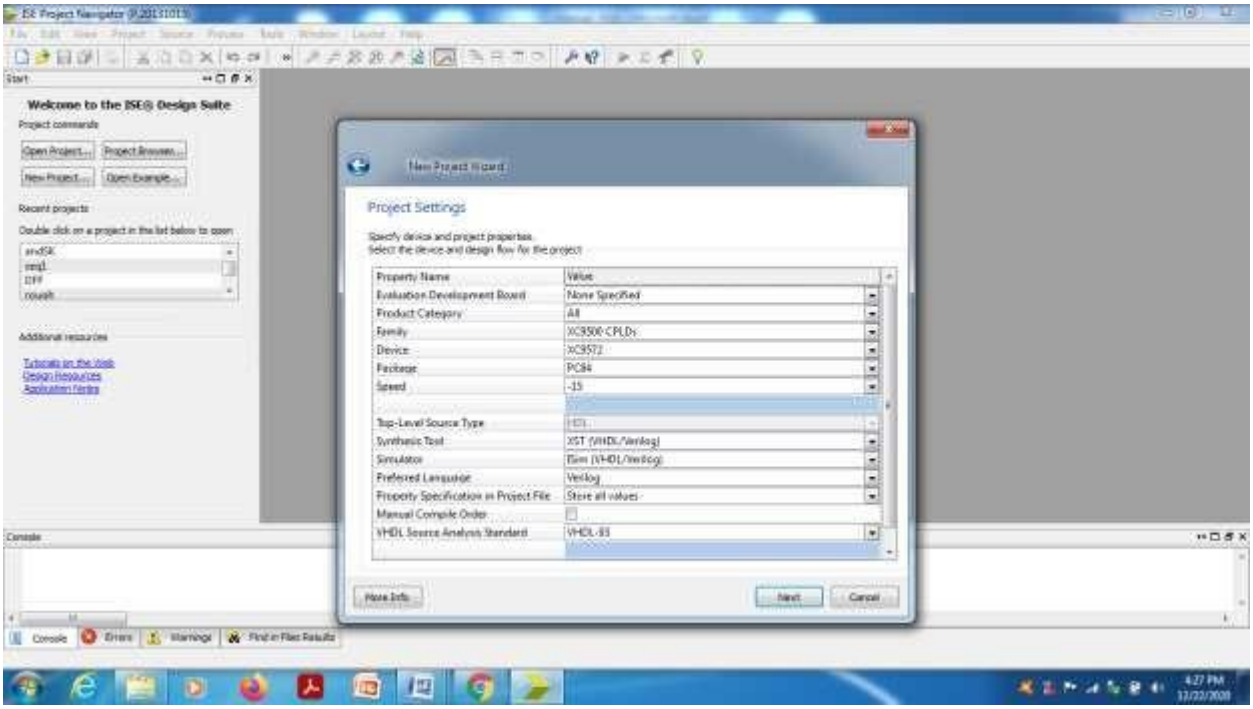
2. Select File-New Project



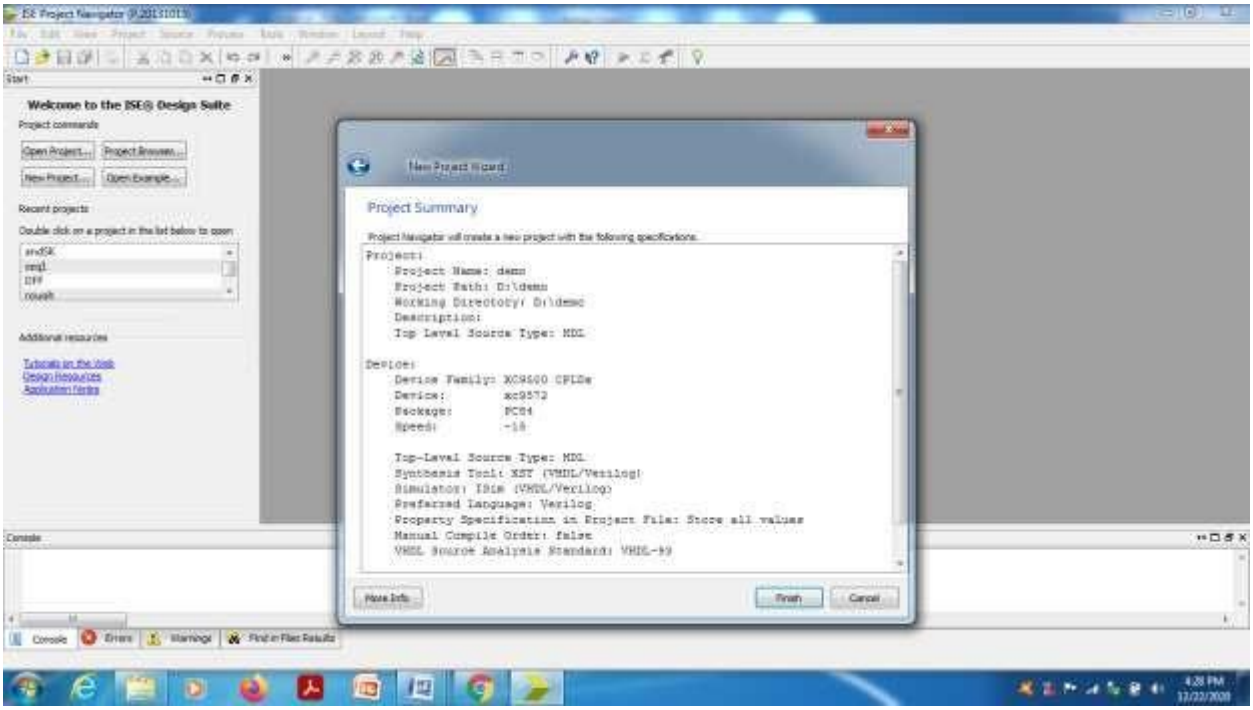
3. Give the name of the project, browse the location for the project and select ‘Next’



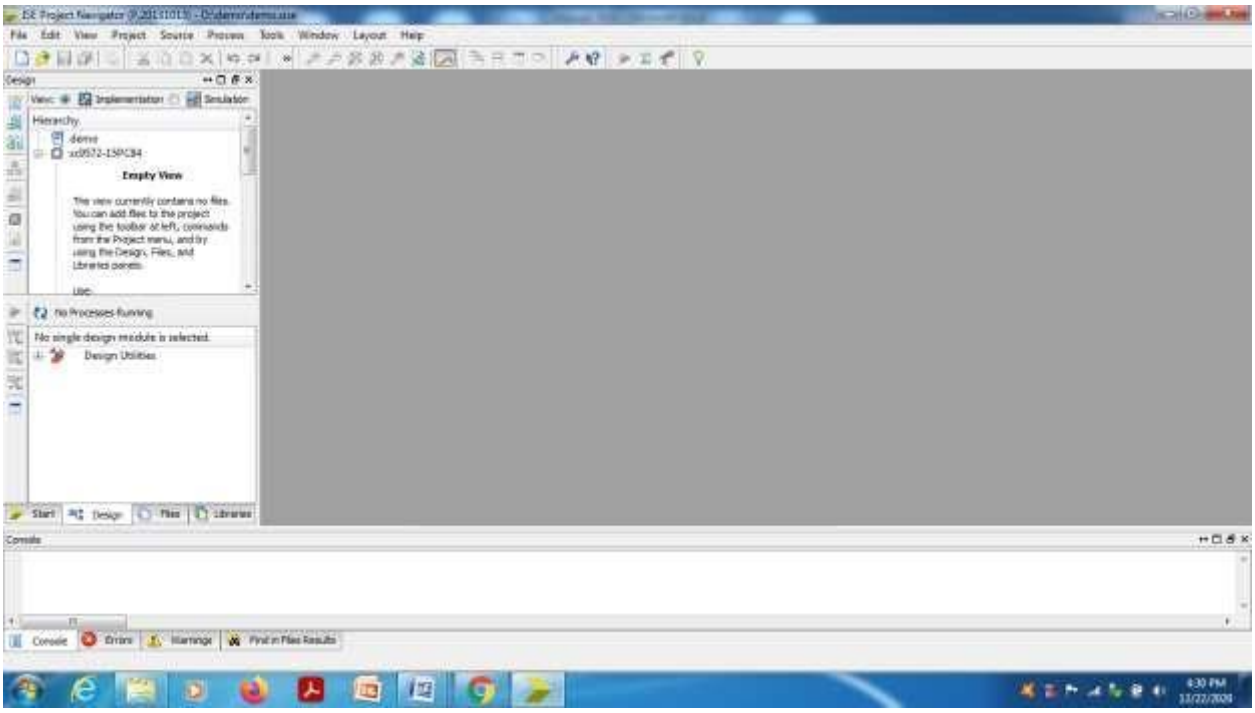
4. Enter the CPLD device details and choose ‘Next’.



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

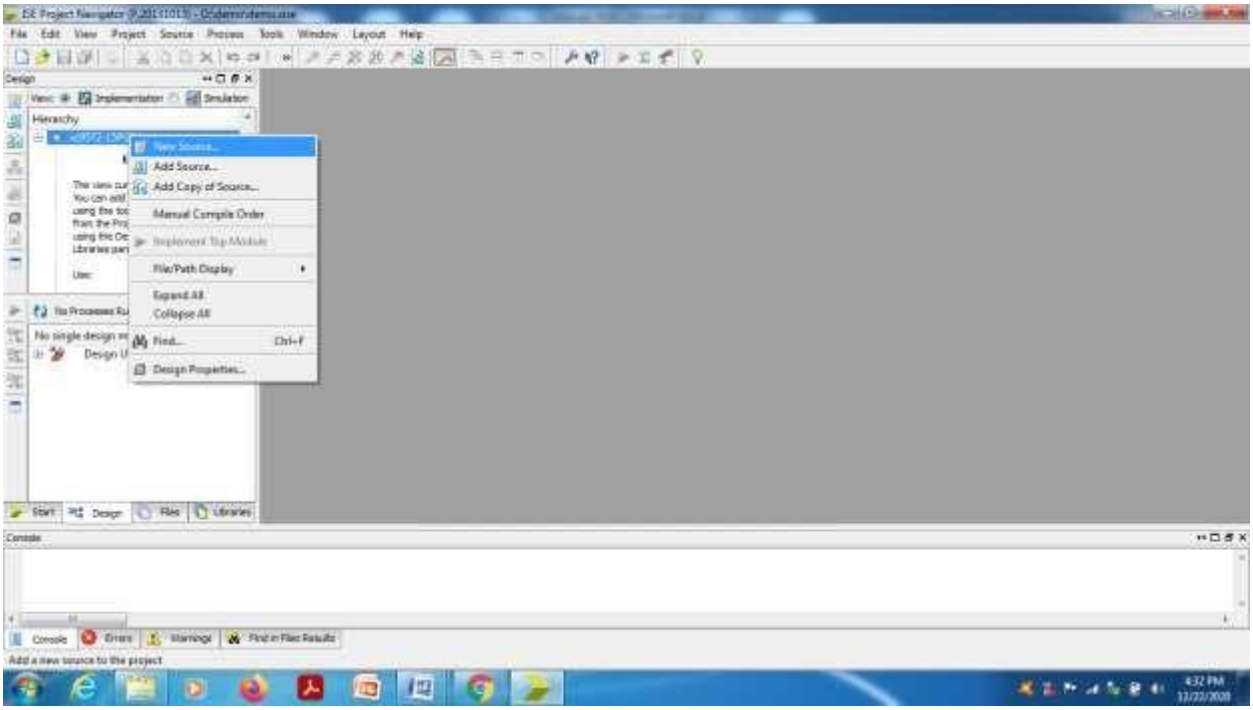


Project window will be opened.

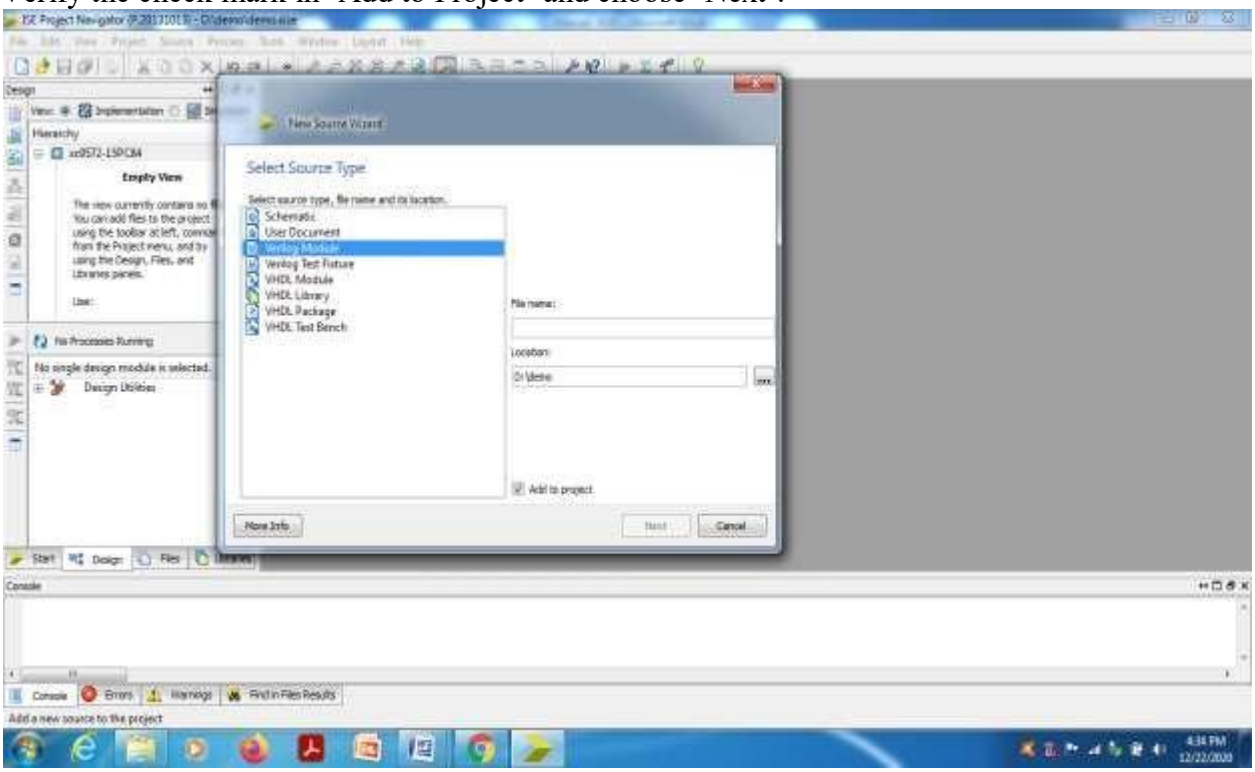


II. To create a Verilog source(program) under the project.

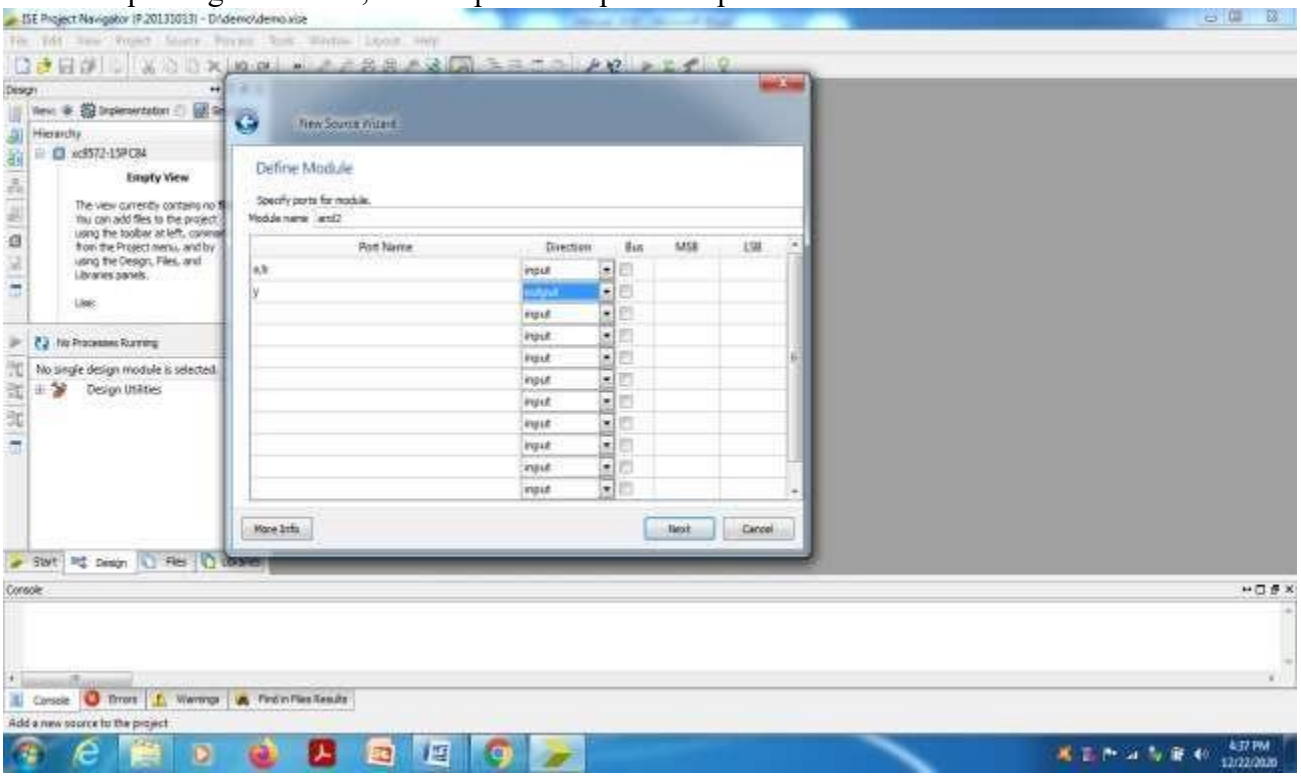
Select the device ‘xc9572-15PC84’ ,right click and select ‘New Source’.



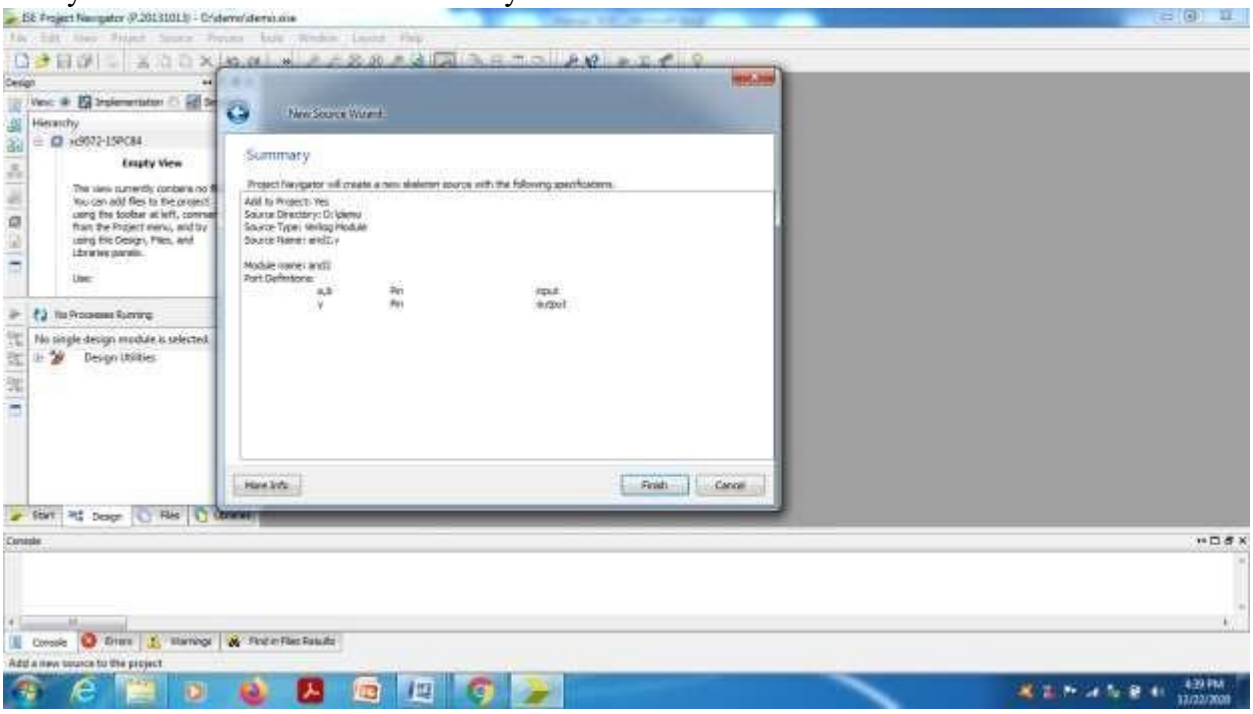
Select ‘Verilog Module’ from source type window and give the file name without any extension.
Verify the check mark in ‘Add to Project’ and choose ‘Next’.



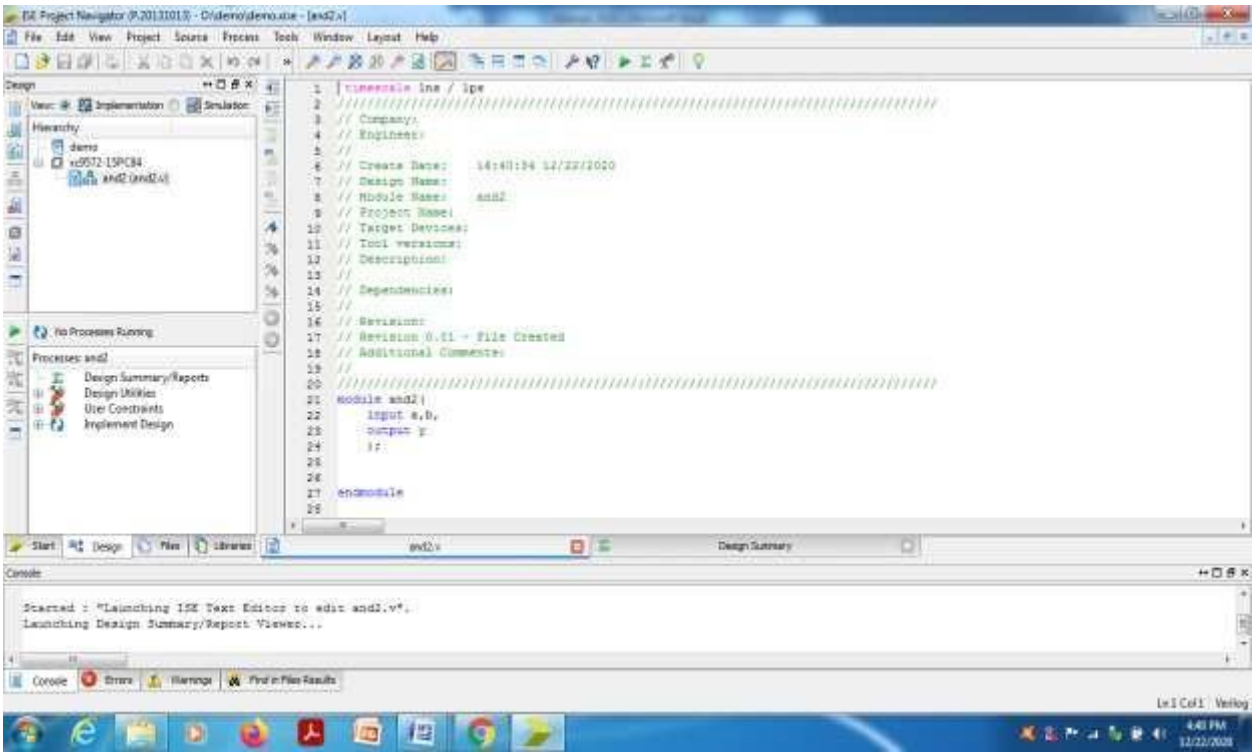
Write the port signal names ,select input or output from pull down menu and choose ‘Next’.

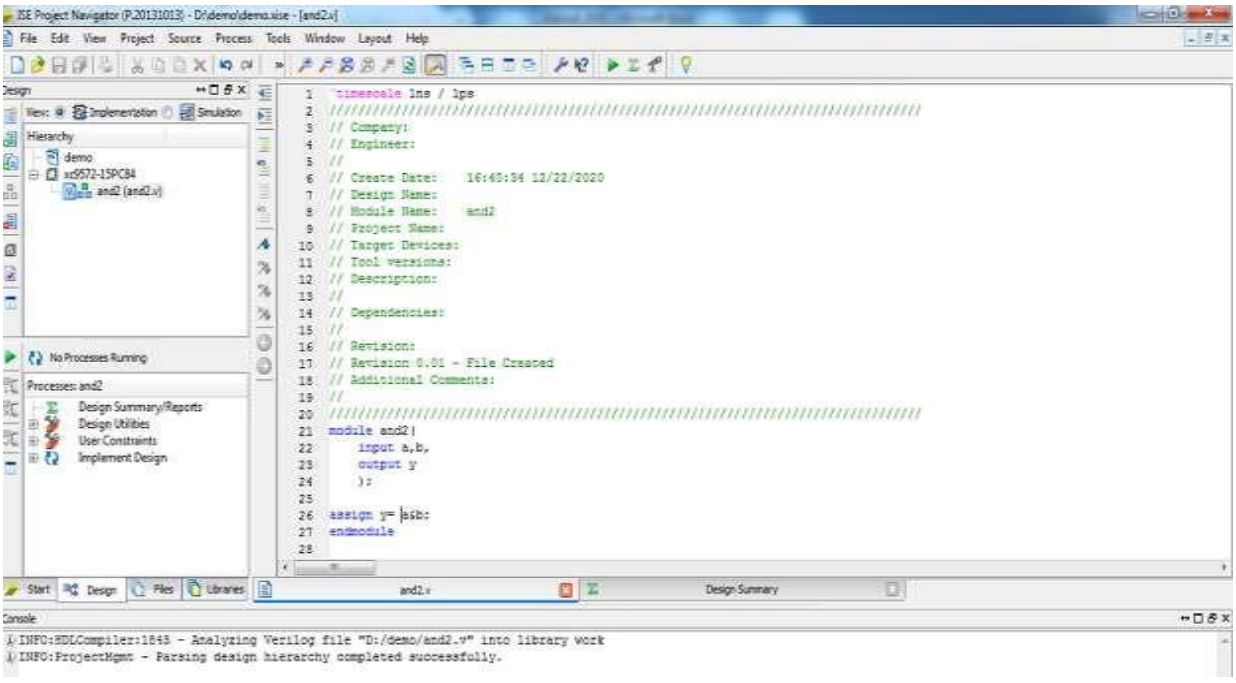


Verify the source details in the summary and choose ‘Finish’.

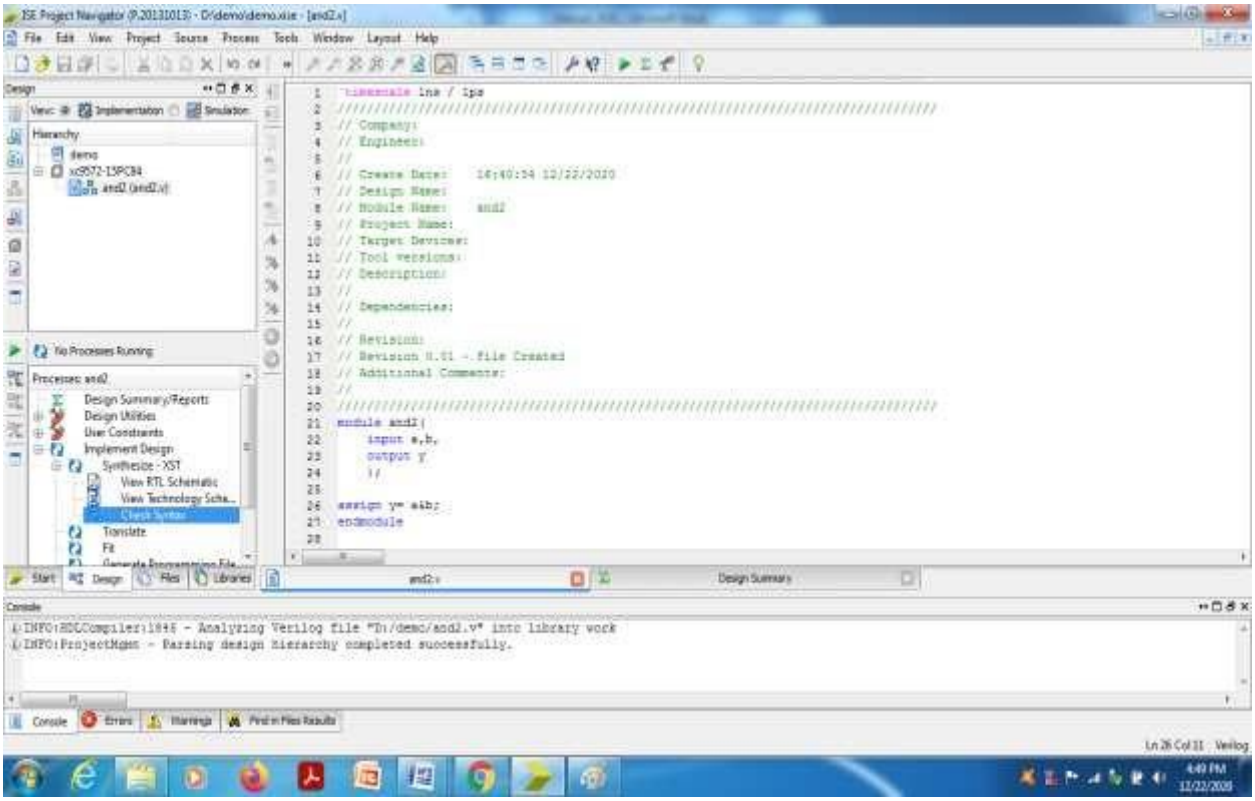


Verilog source file'and2.v' will be opened with skeleton. Edit the program and save.

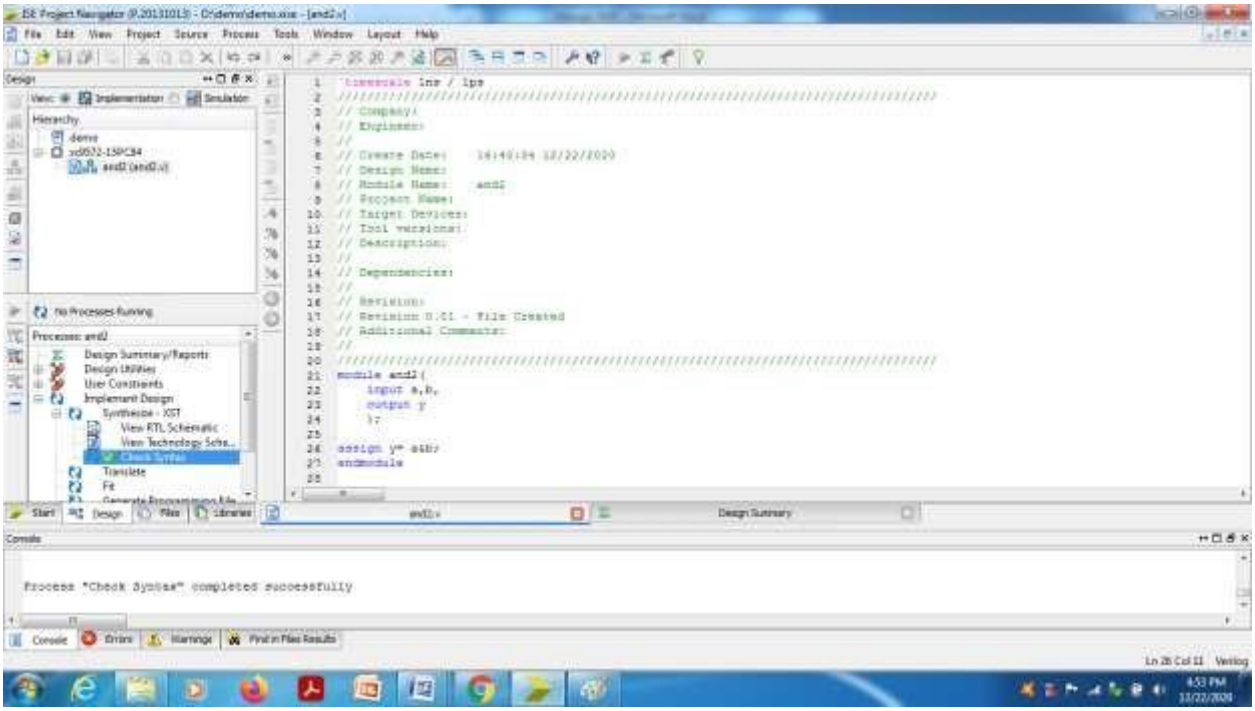




Select the ‘and2.v’ in project window and select
Implementation Design---Synthesize XST -----Check syntax.(Double click).

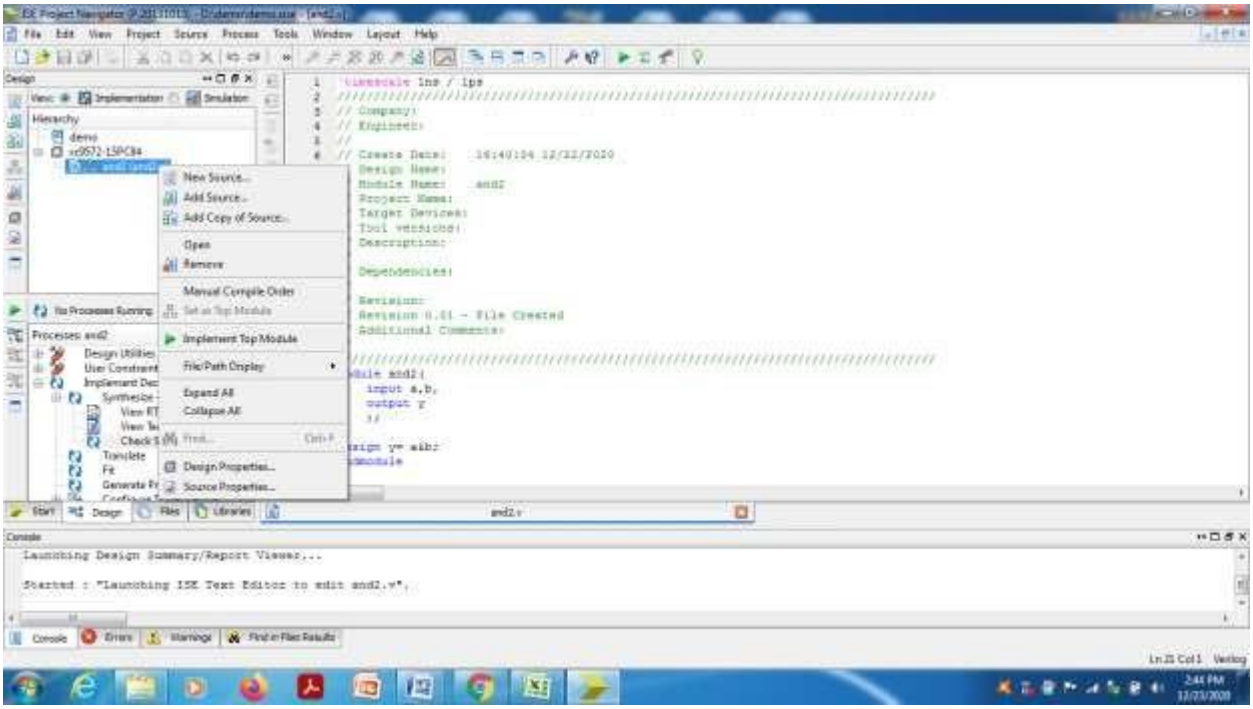


Check for any errors in the console window. If any error is shown ,edit the program and do the corrections until ‘process’check syntax’completed successfully.

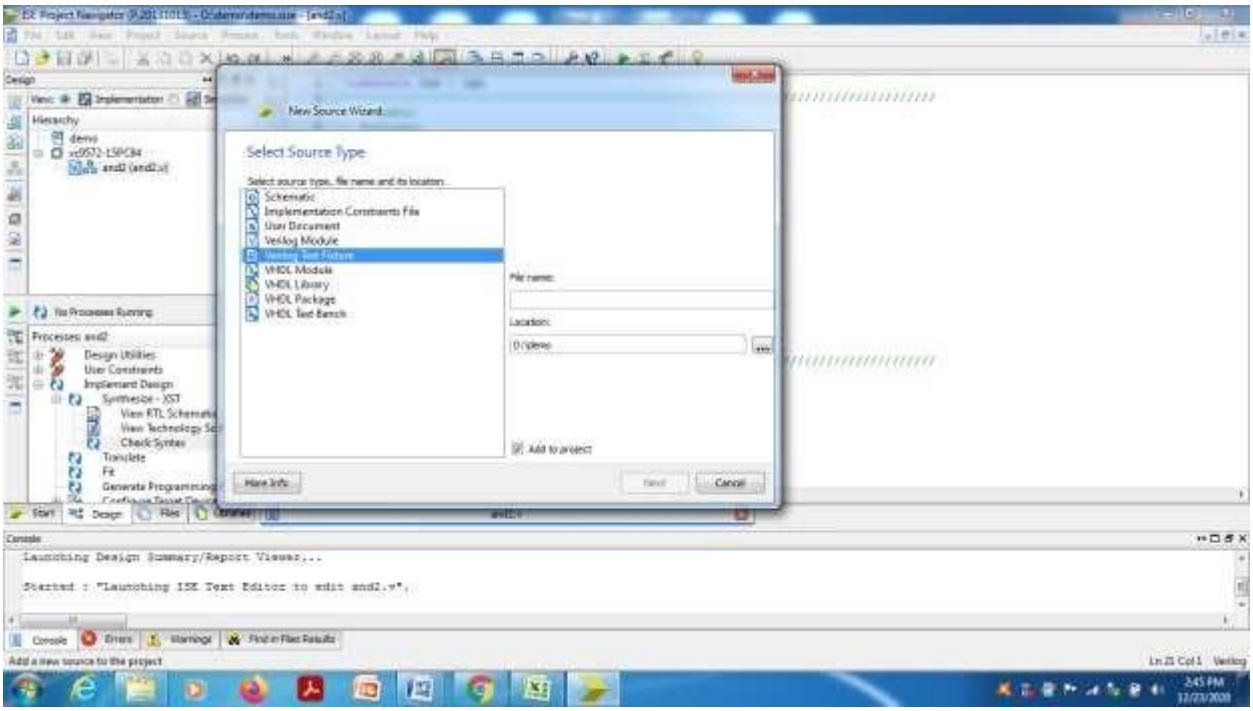


To Create a test bench Program and Perform the simulation:

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

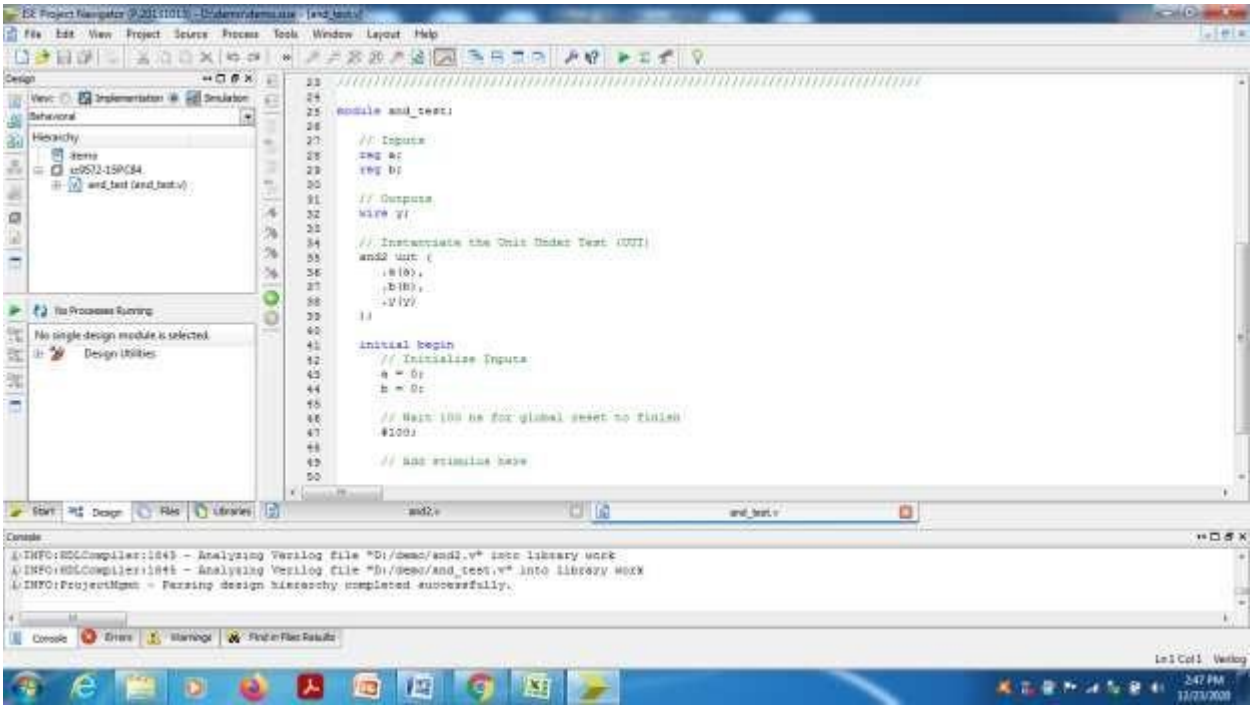


Select source type as ‘Verilog Test Fixture’ and give file name for the test bench program.

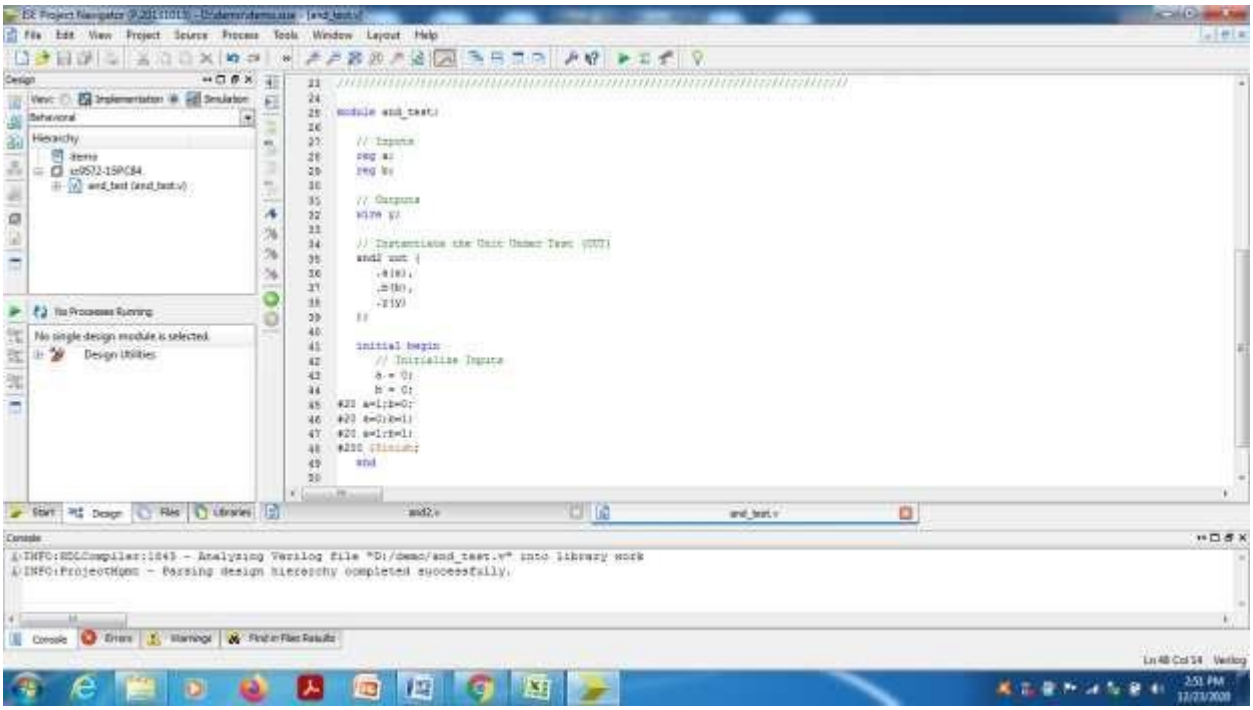


Choose Next---Next --- and Finish.

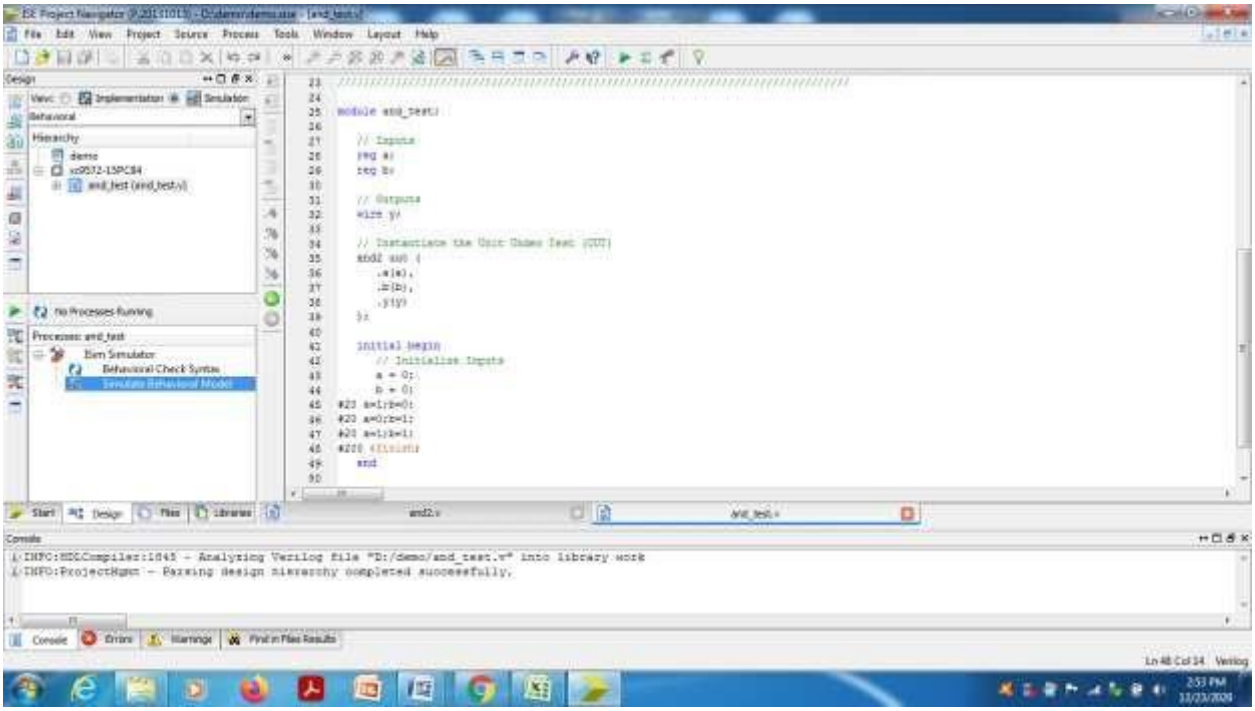
Change the view above Project window from ‘Implementation’ to ‘Simulation’ mode.



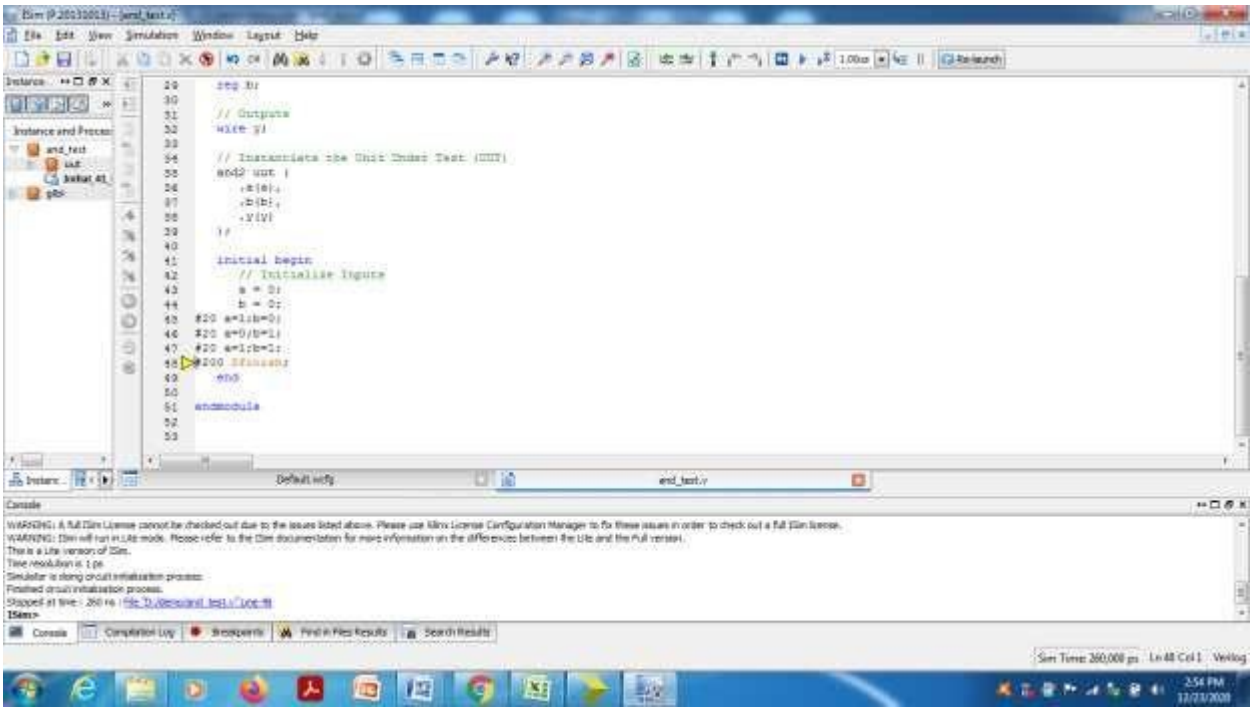
In the skeleton of test bench program, do the changes in the input test vectors inside the initial block.(Or the necessary changes according to the program requirement).



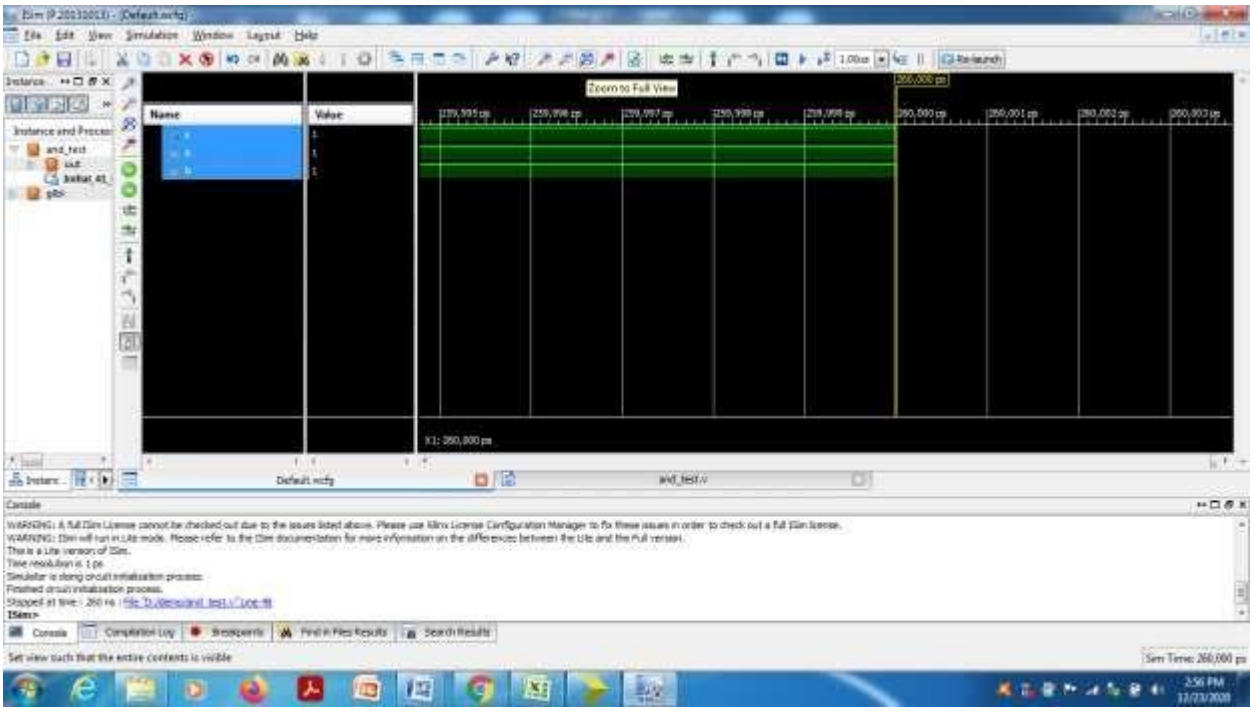
Perform the ‘Behavioral check syntax’ for any errors. If no errors, then choose ’Simulate Behavioral Model’.



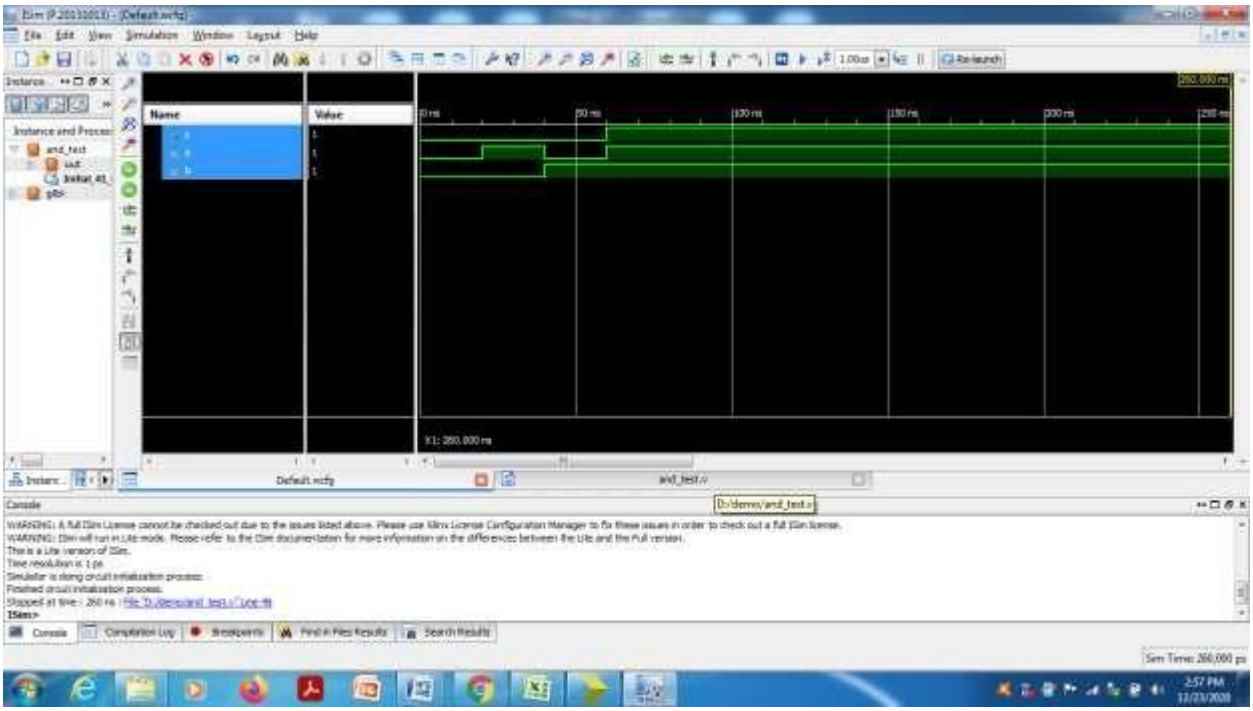
Isim window will be opened and open “default.wfcg” tab to view simulation result.



Select ‘Zoom to full view’.



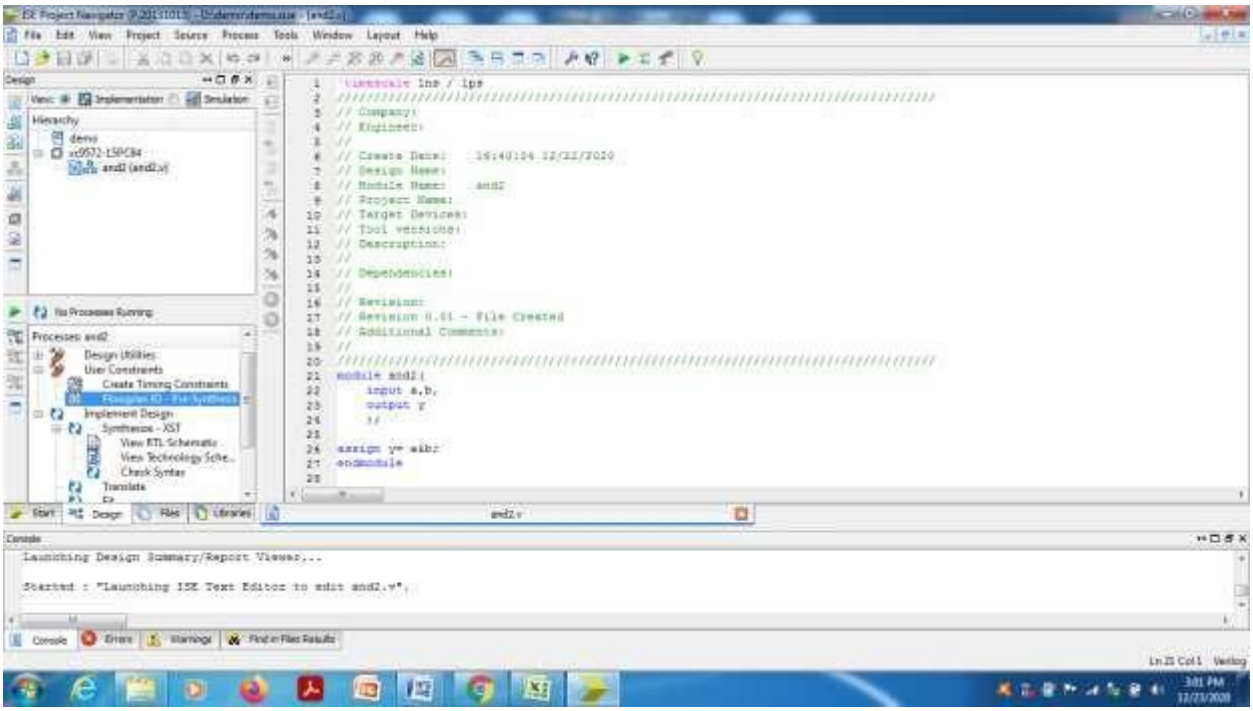
Place the marker at different inputs and verify the output.



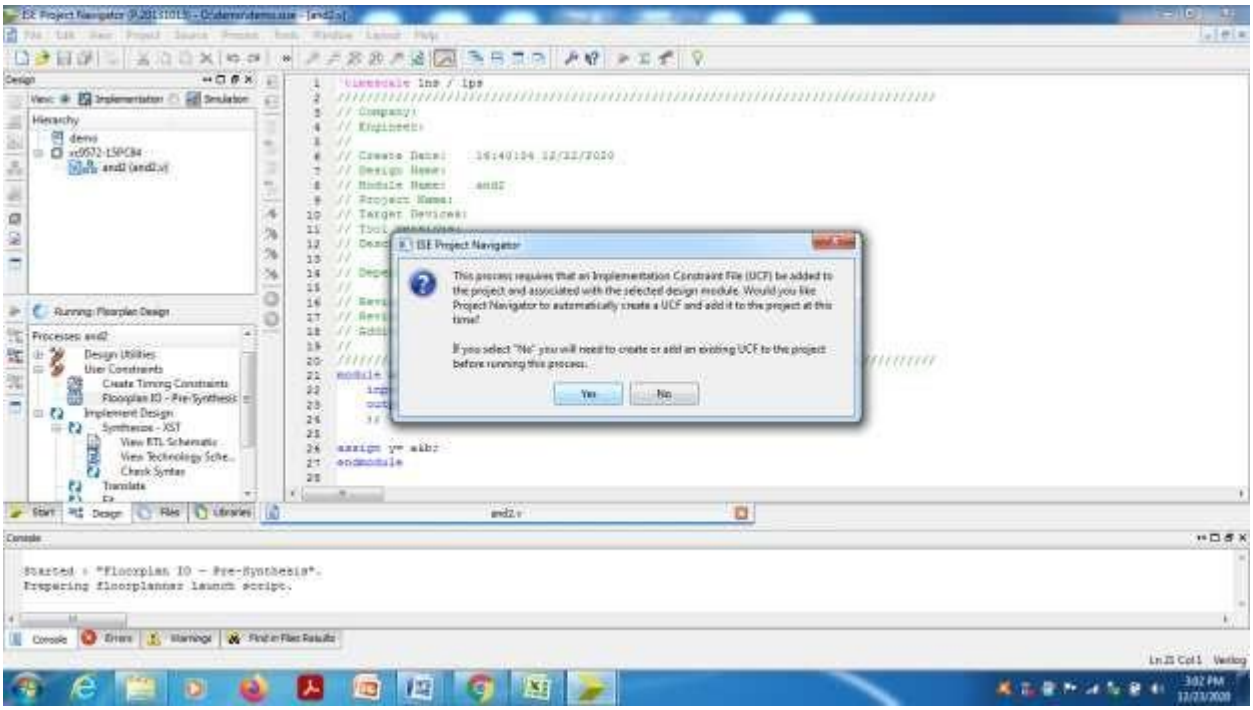
III. To Create .UCF file for pin assignment.

Change the view from 'Simulation' to 'Implementation' (above the project window).

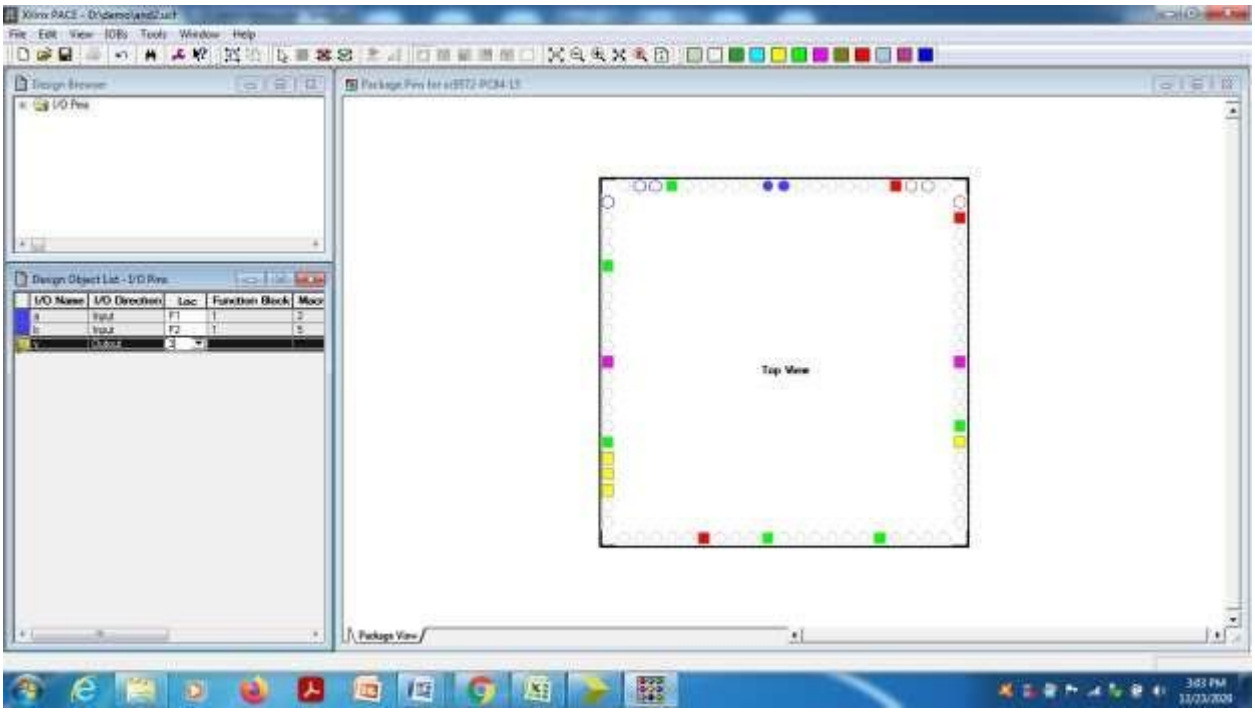
Select the design program by just one click and choose User constraints----Floor Planning-IO Pre synthesis.



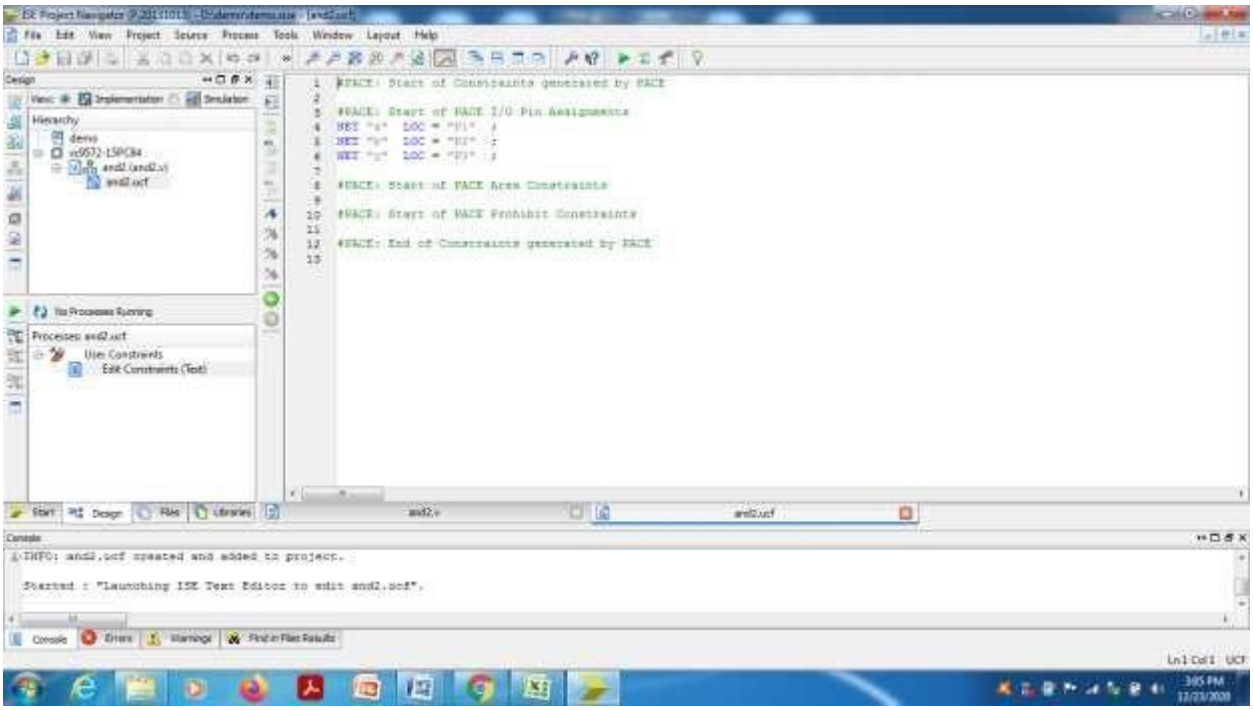
Choose 'yes'.



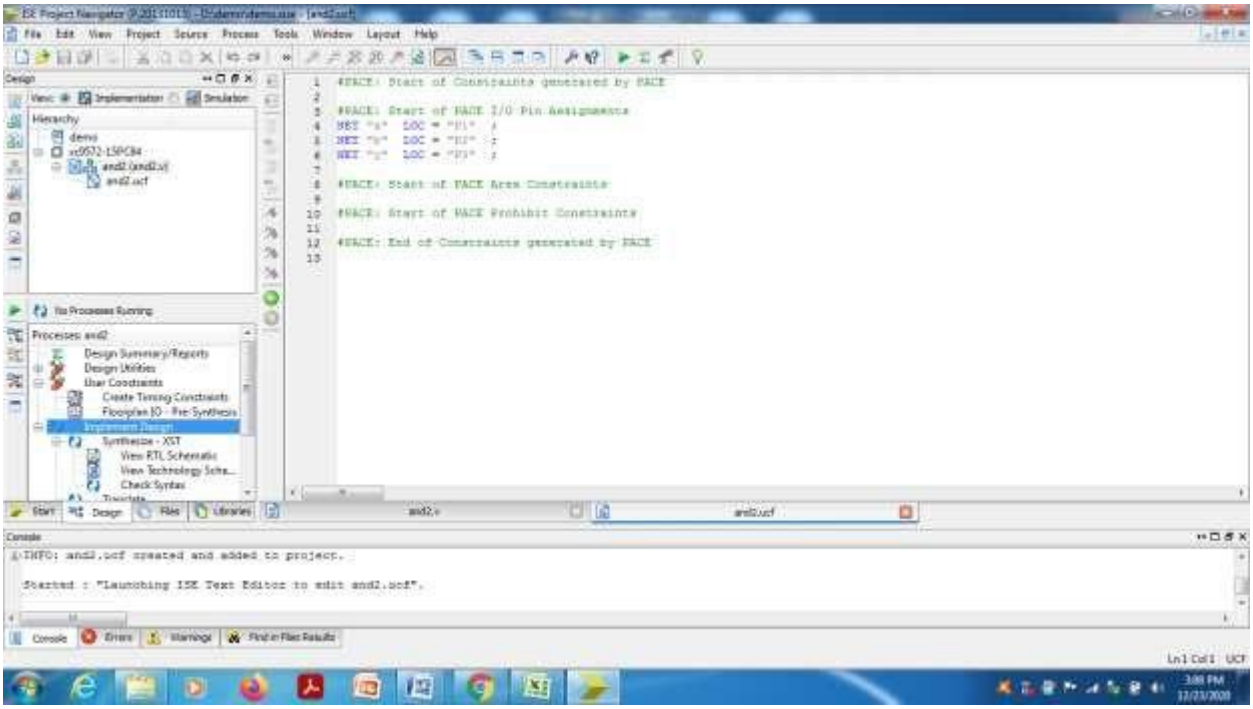
In the Xilinx PACE window,do the pin assignment according to the requirement and save, select ok and close the file. By doing this, and2.ucf file would have created and saved under the design file.

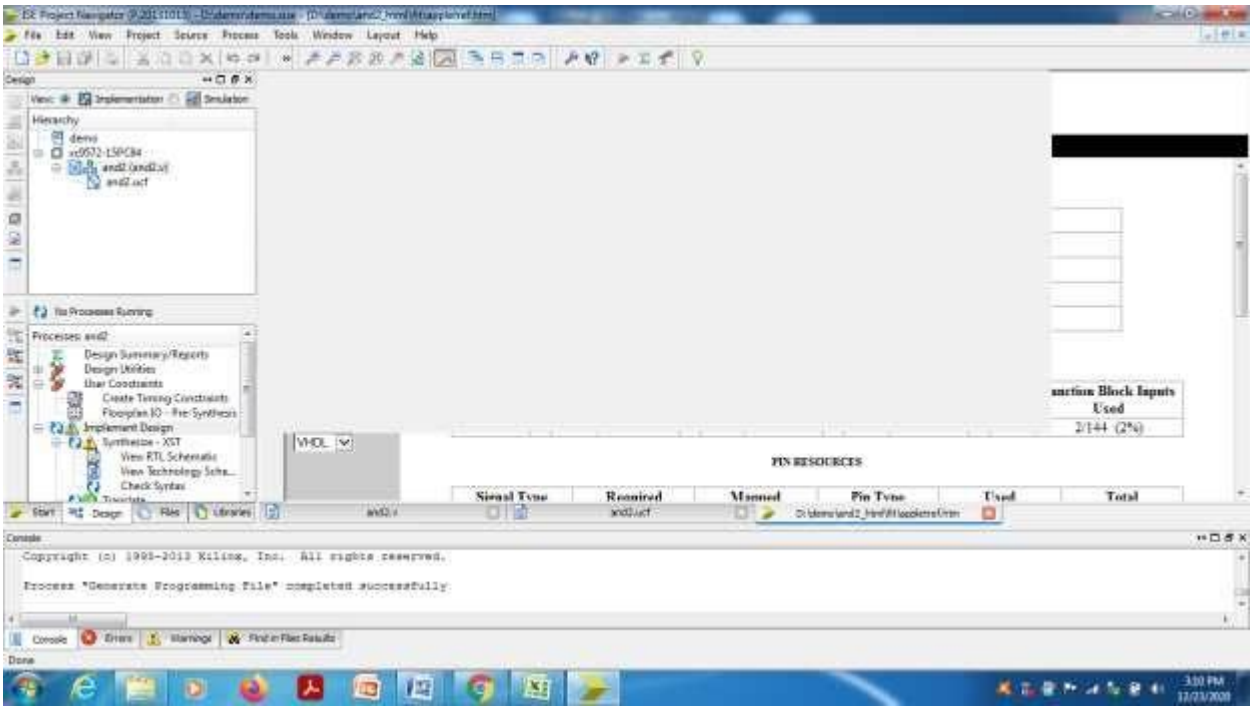


With ‘Edit constraints’ option,we can open and edit the .UCF file.

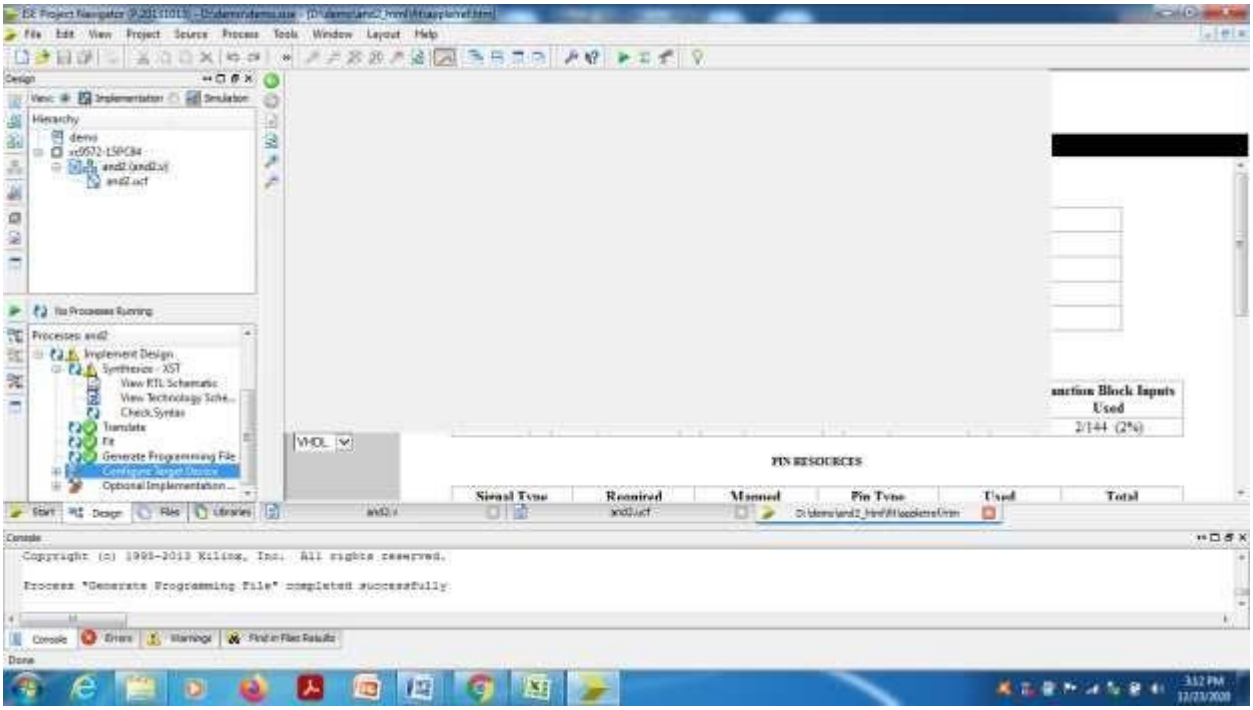


Select the design program in project window and double click on’Inplementation design’ and wait till we get the message as Process “Generate Programming file” completed successfully.

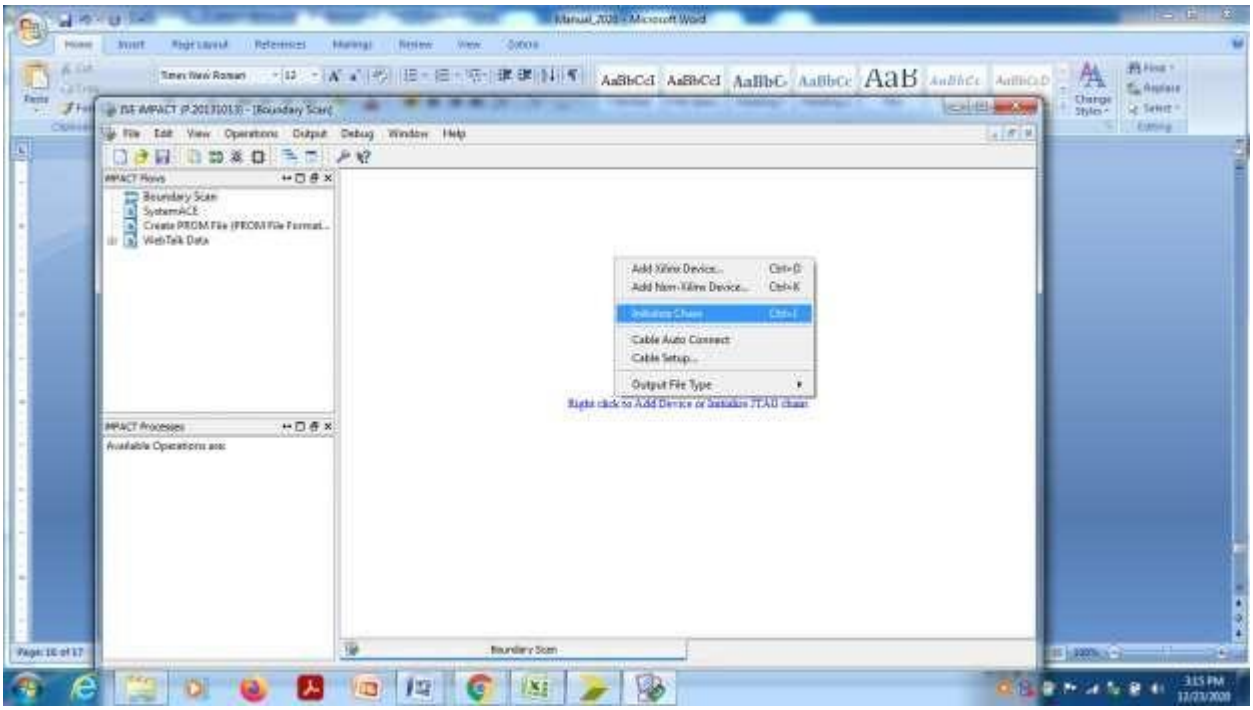




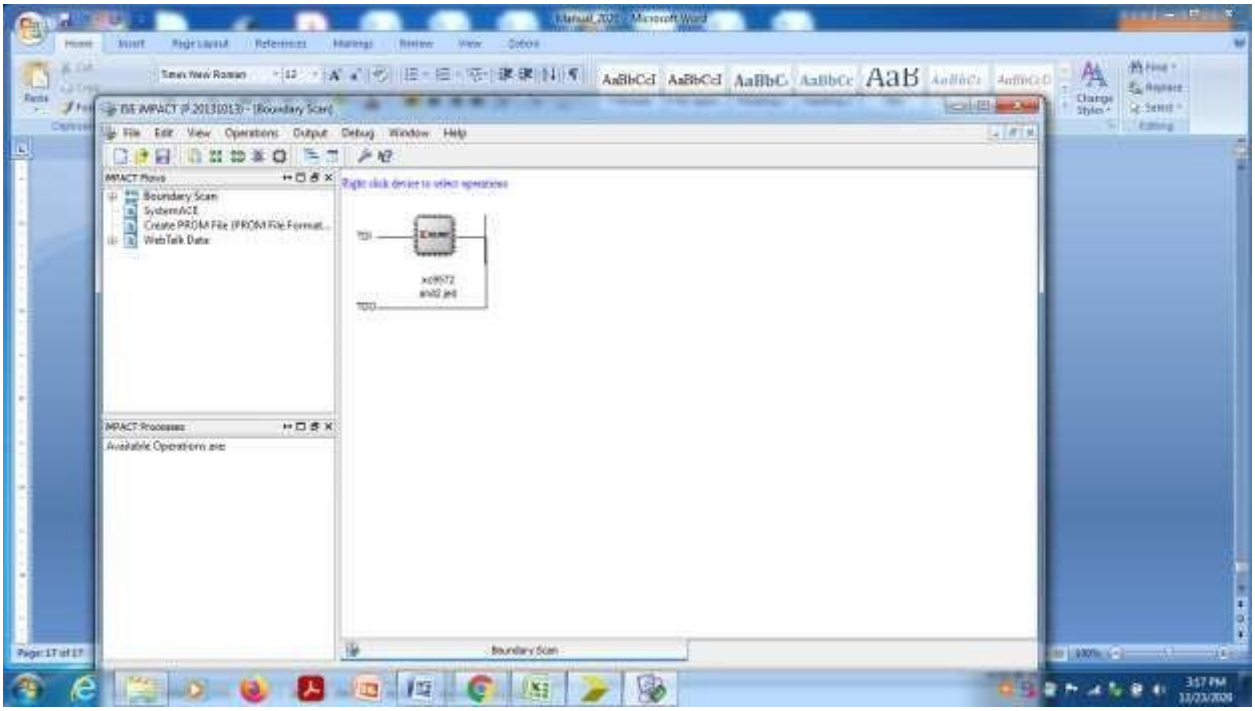
Doule click on 'Configure Target device' for dumping the code on CPLD chip and choose 'ok'.



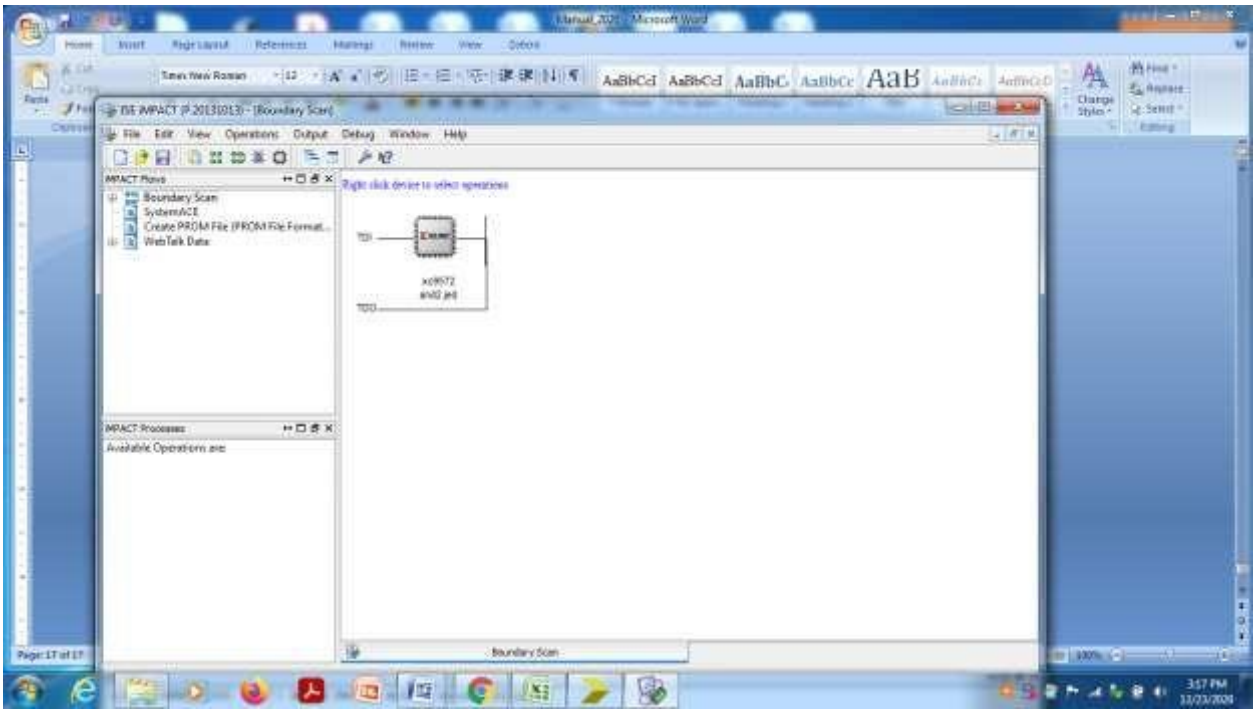
Xilinx ISE IMPACT window will be opened,select ‘Boundary scan’.
Right click on the workspace and select ‘Initialise the chain’.

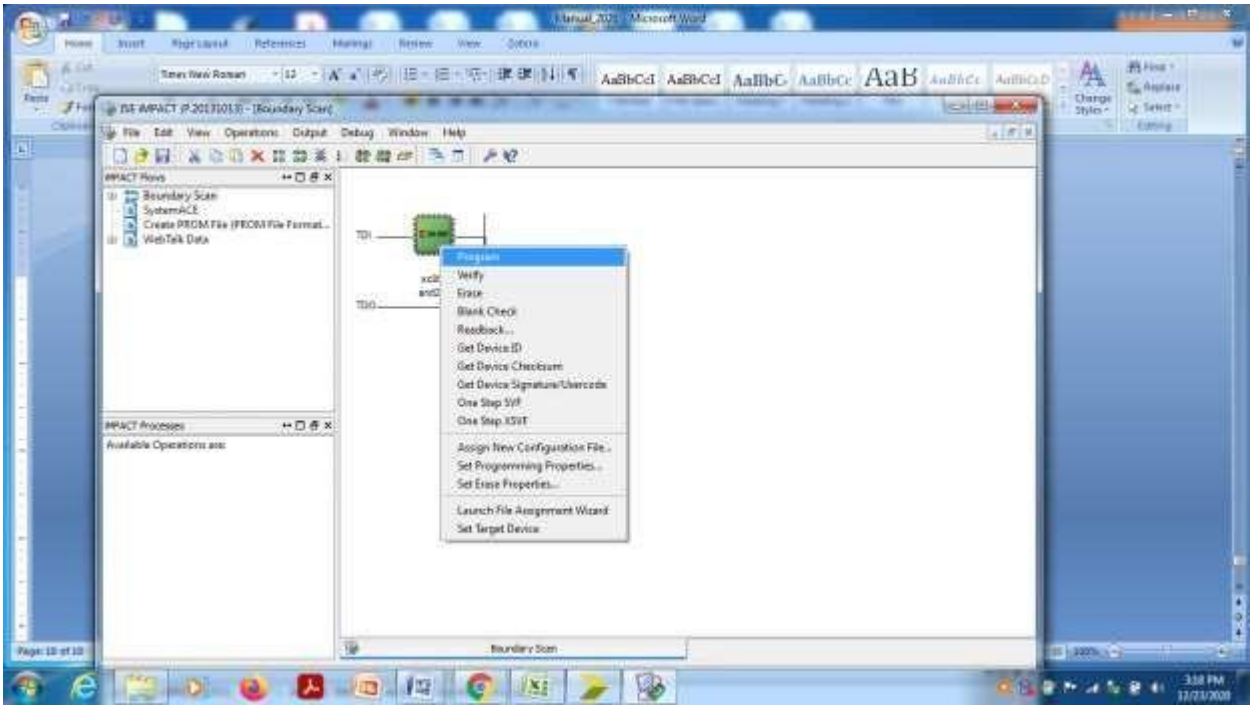


Browse the file name with .jed extension and assign to the IC icon.



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





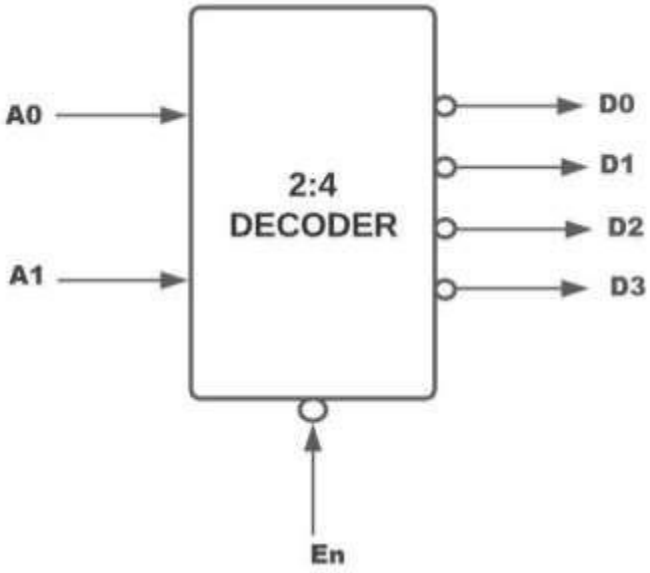
After getting 'Programming succeeded",feed the inputs and check the ouput in LEDs in hardware domain.

PART-A

E.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

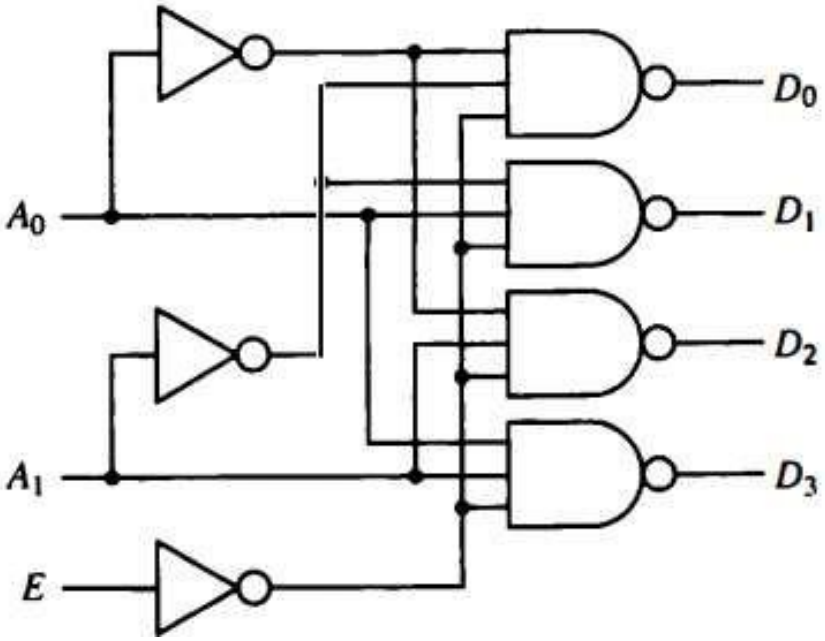
Block Diagram:



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

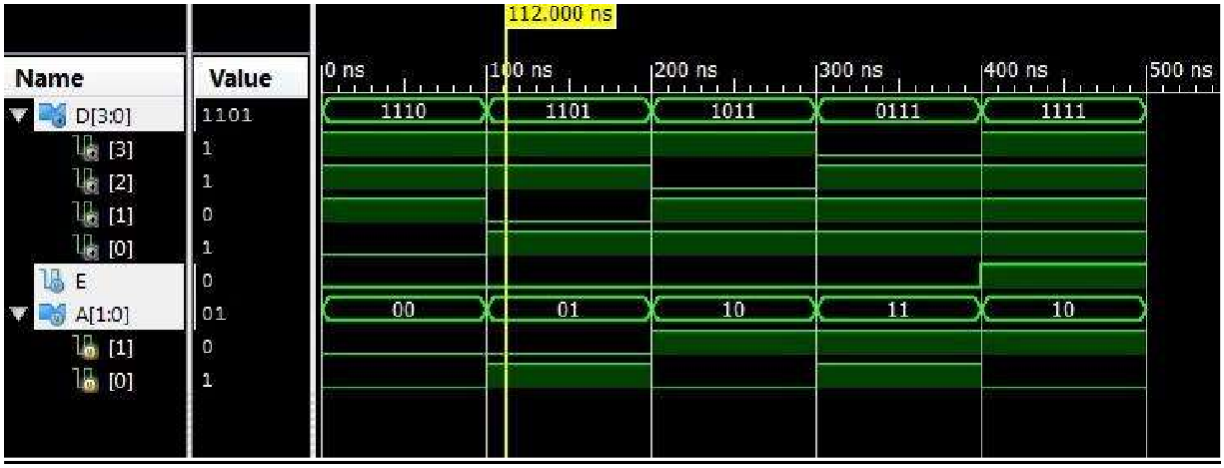
```
module decoder_24(
   input E,
   input [1:0] A,
   output [3:0] D
   );

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
```

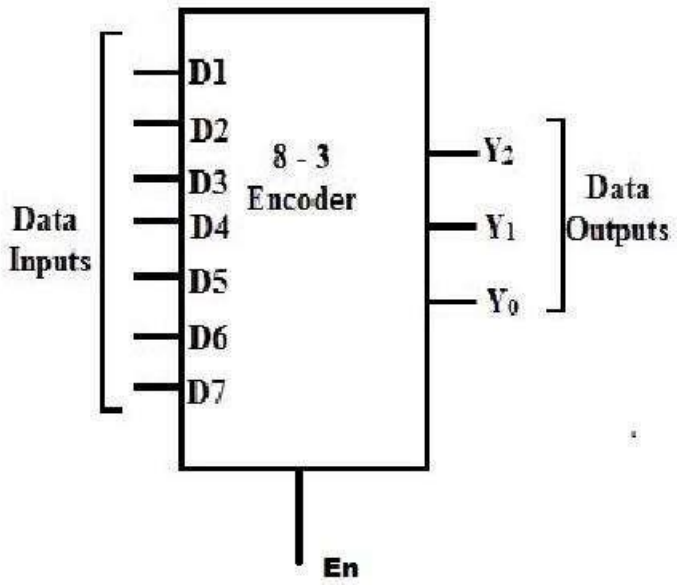
Output waveform



Exp.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:

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	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:

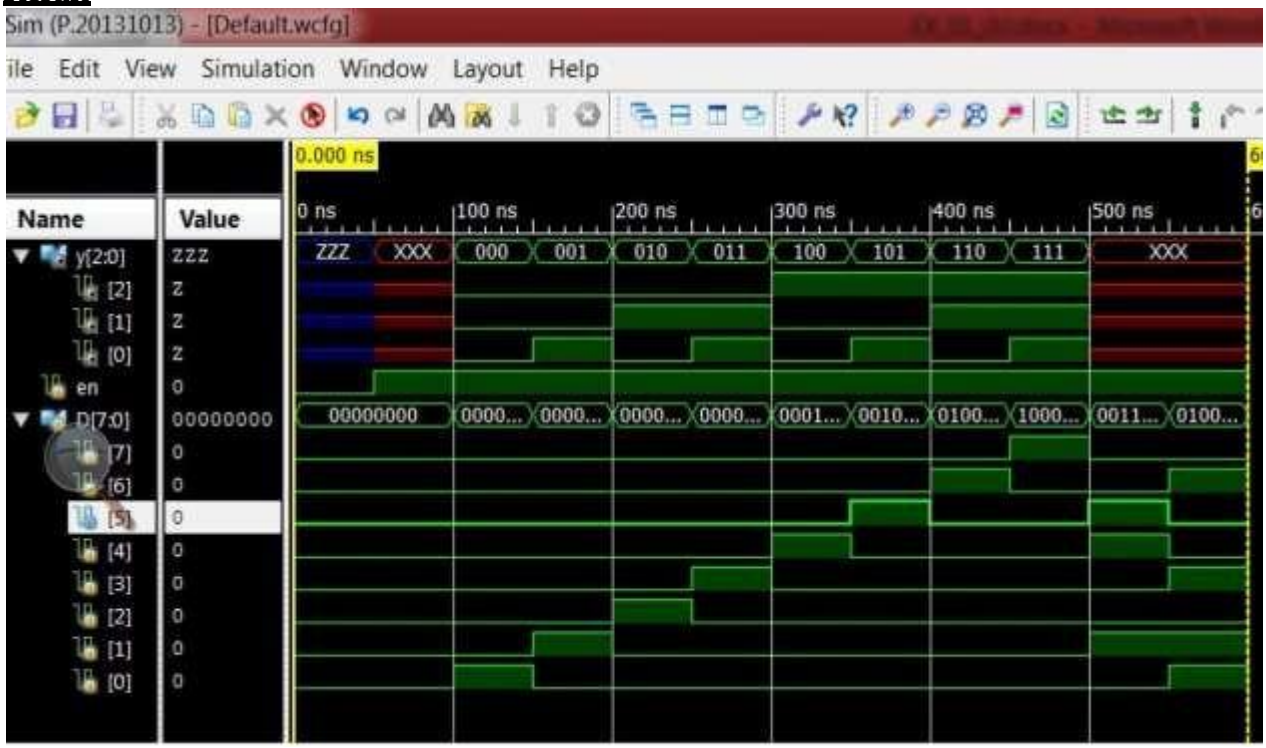
```
module encoder_83(
   input en,
   input [7:0] D,
   output reg [2:0] y
   );

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
```

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:

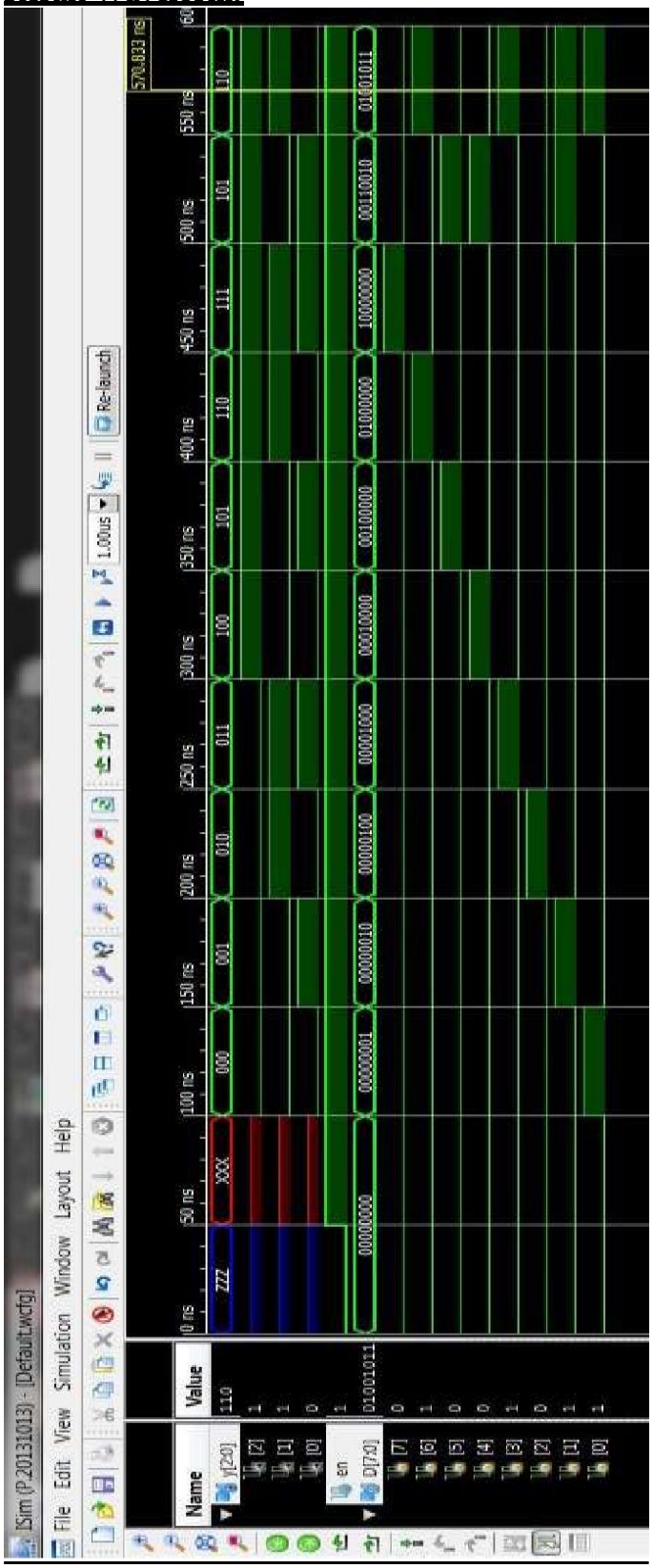
```
module encoder83_WP(
   input en,
   input [7:0] D,
   output reg [2:0] y
   );

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
```

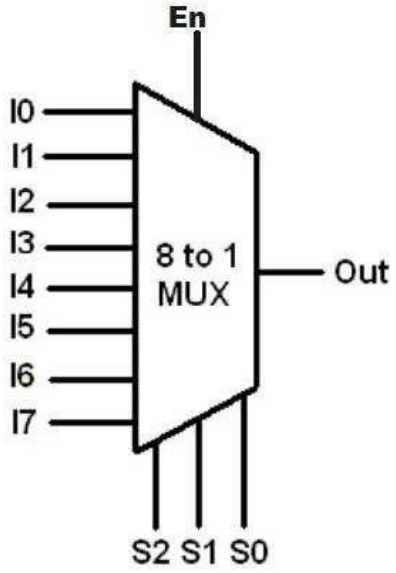
Result Waveform:



Exp.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
```

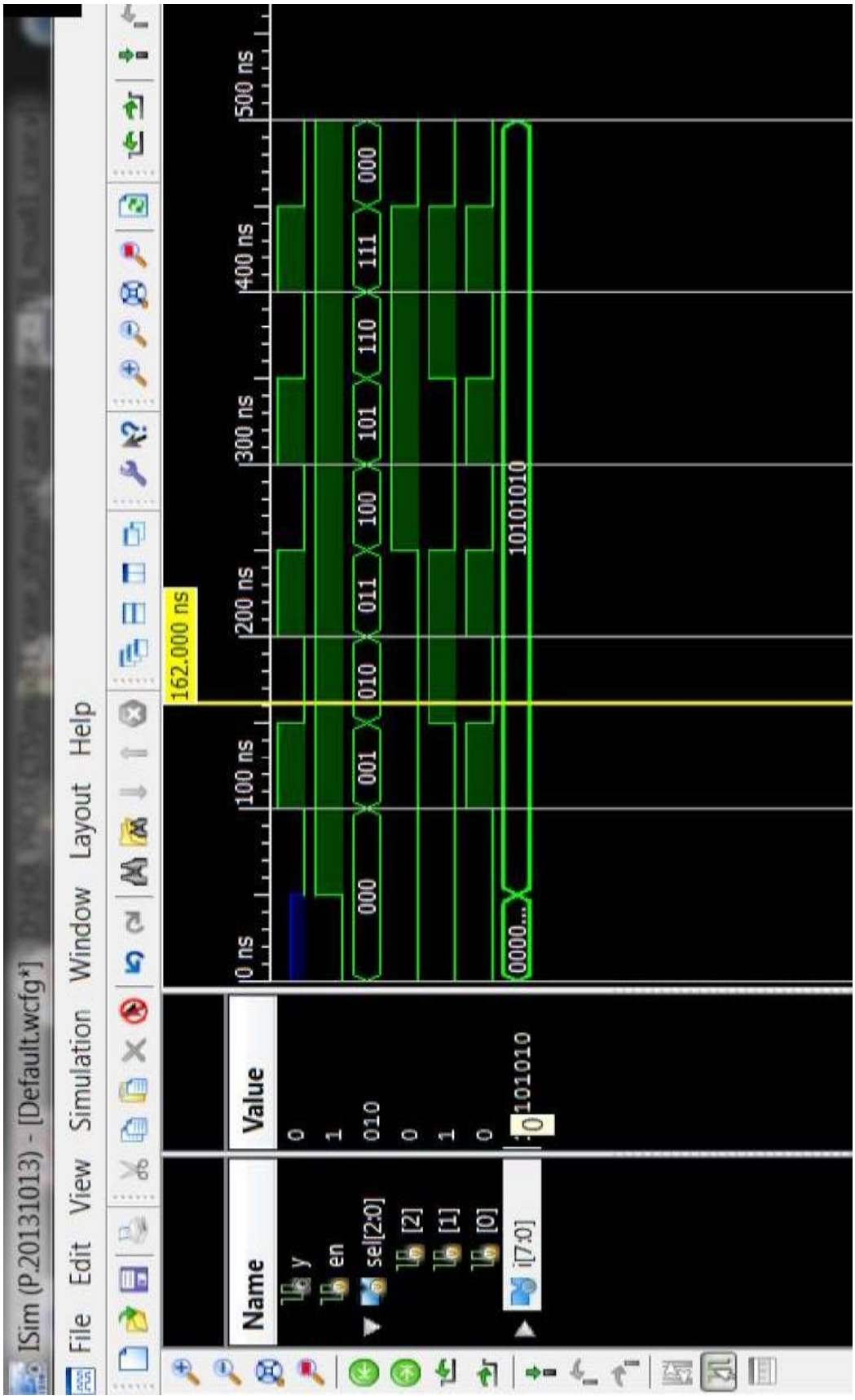
Program using ***case*** statement.

```
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
```

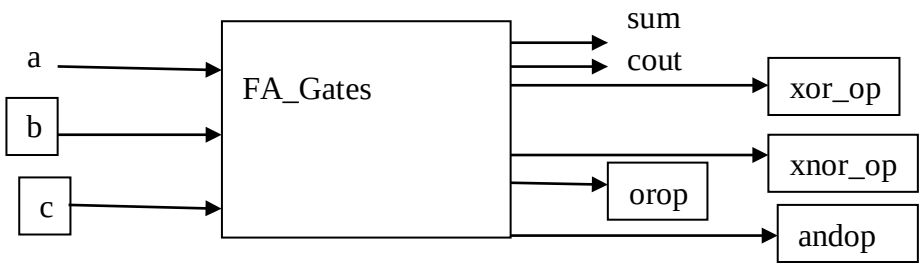
Output waveform



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:



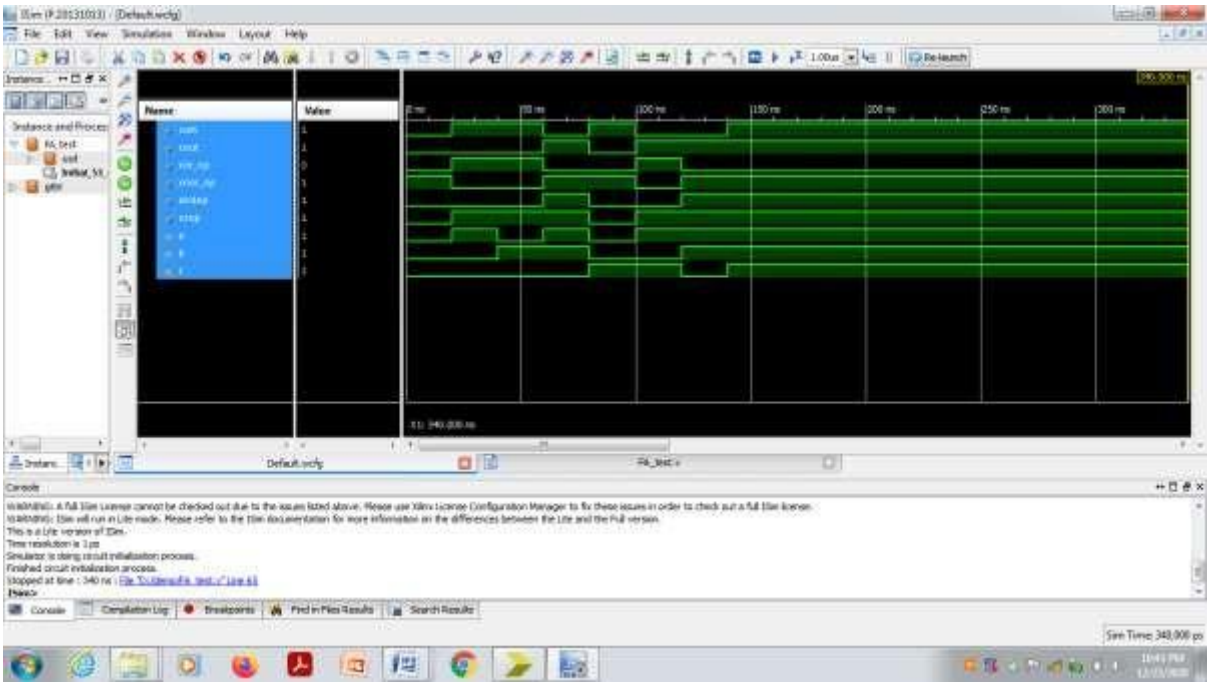
Program:

```
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
```



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:

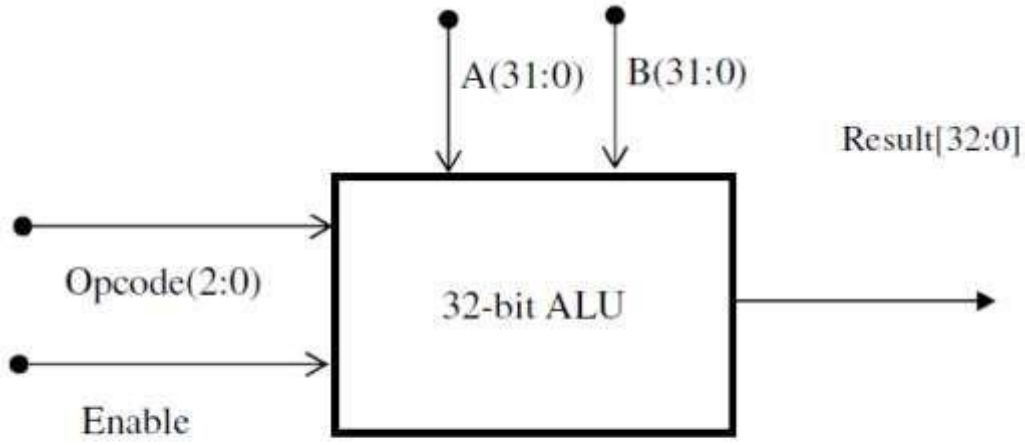


Figure 1 ALU top level block diagram

Table 1 ALU Functions

res

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:

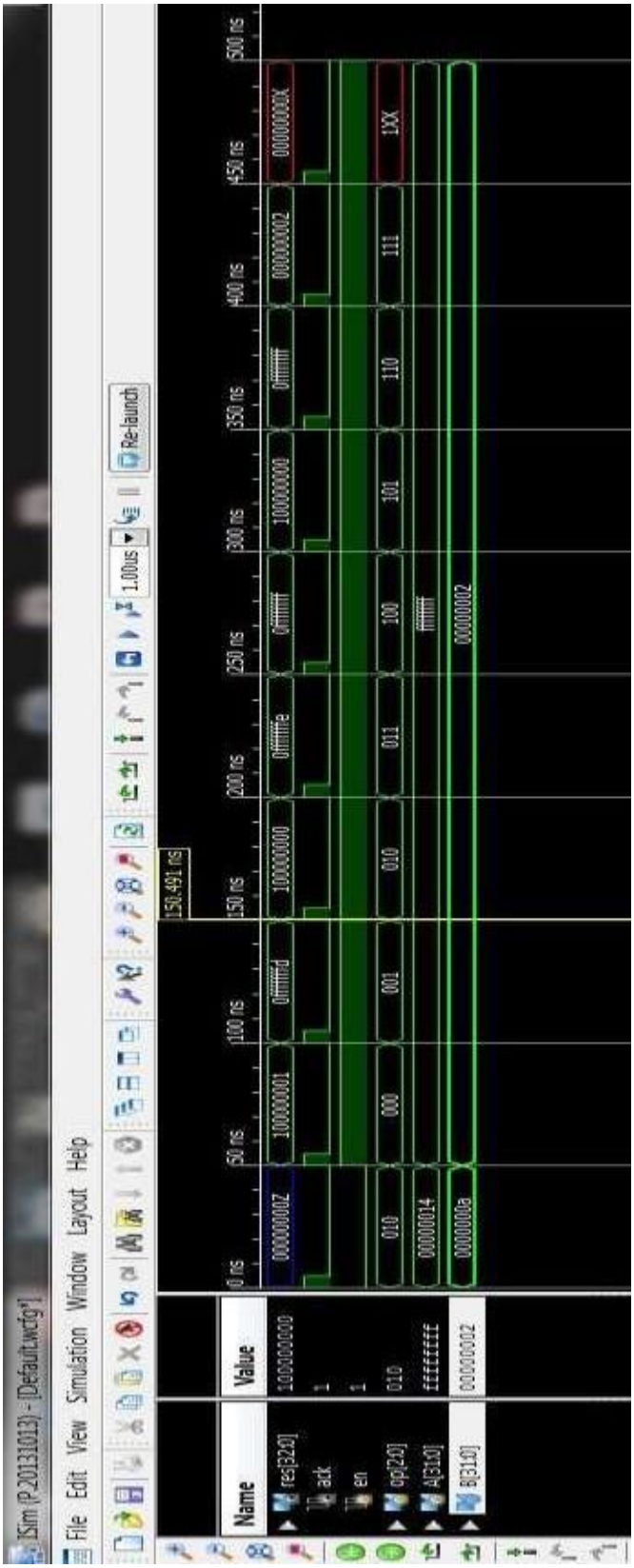
```
module ALU32(
   input en,
   input [0:2] op ,
   input [31:0] A,
   input [31:0] B,
   output reg ack,
   output reg [32:0] res
   );

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
```

Result Waveform



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

AIM:

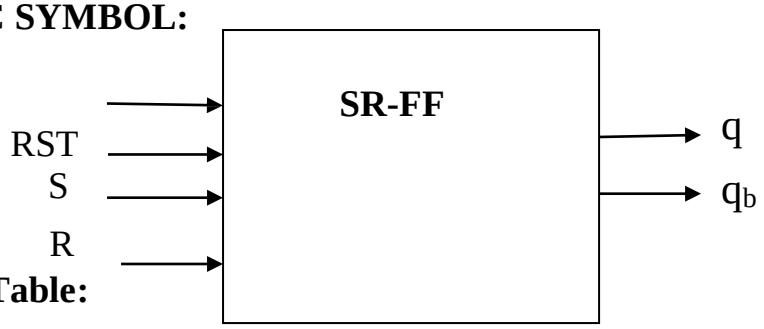
To model SR Flip Flop using verilog behavioral description and verify its function using ISE simulator as well as on the CPLD XC9572.

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 reg
qb );
 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:

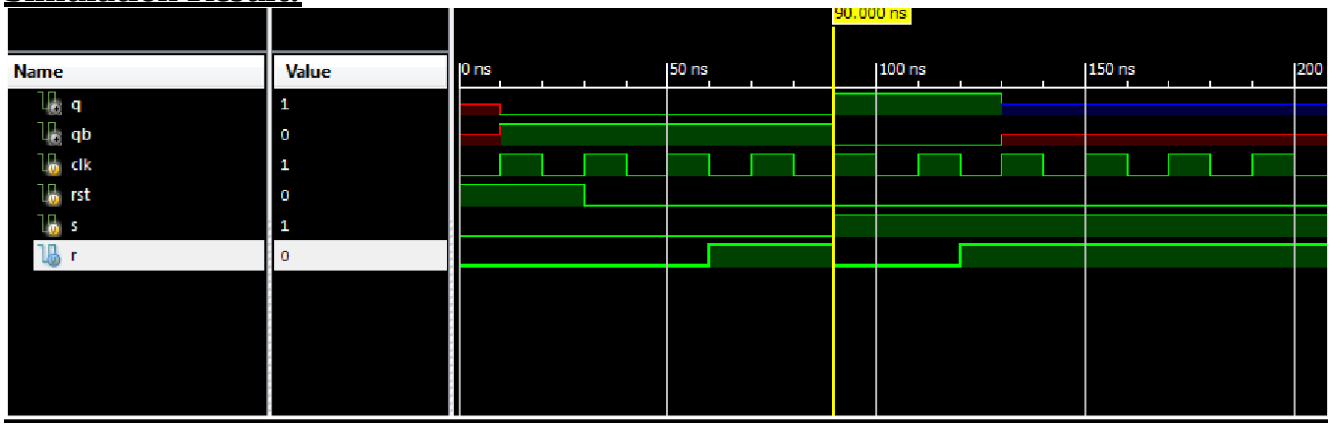
```
`timescale 1ns / 1ps
module srff_test;
reg clk;
reg rst;
reg s;
reg r;
wire q;
wire qb;
srff uut (.clk(clk), .rst(rst), .s(s), .r(r), .q(q), .qb(qb));
initial
 begin
                clk = 0;
                rst = 1;
                s = 0;
                r = 0;
                #30 s=0 ;r=0;rst=0;
                #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**

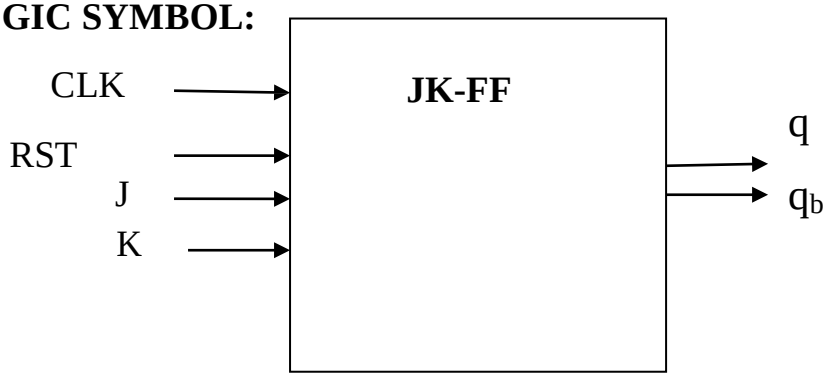
AIM:

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:

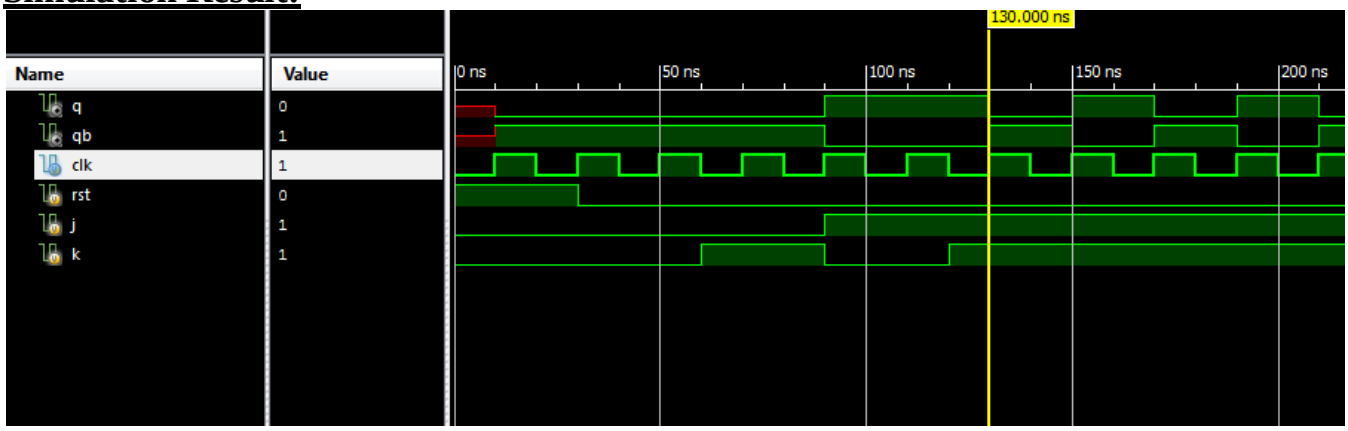
```
`timescale 1ns / 1ps
module jkff_test;
reg clk;
reg rst;
reg j;
reg k;
wire q;
wire qb;
jkff uut (.clk(clk), .rst(rst), .j(j), .k(k), .q(q), .qb(qb));
initial
 begin
                    clk = 0;
                    rst = 1;
                    j = 0;
                    k = 0;
                    #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

```
`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
```

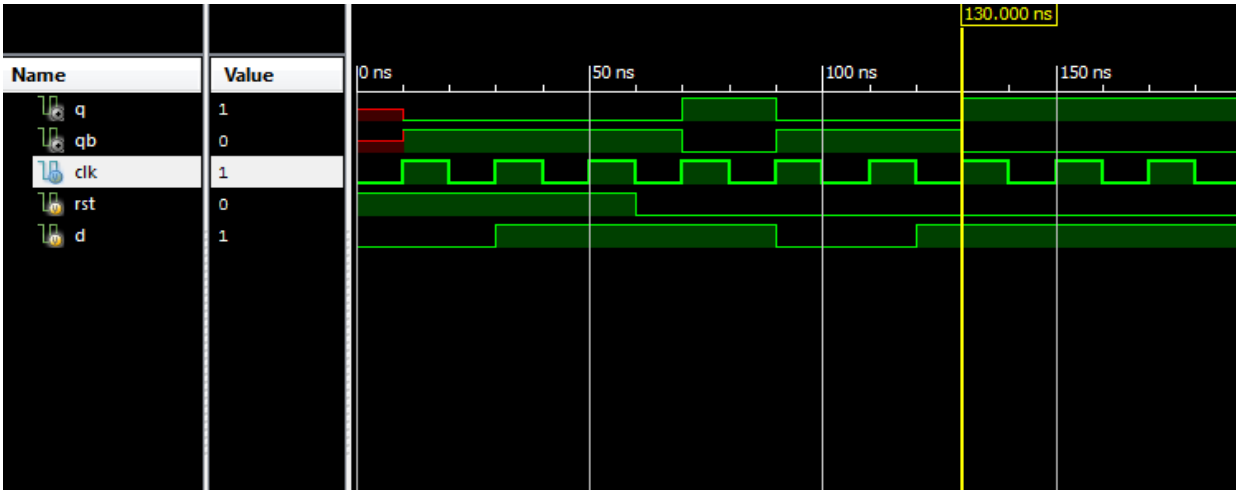
Test Bench Program:

```
`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:

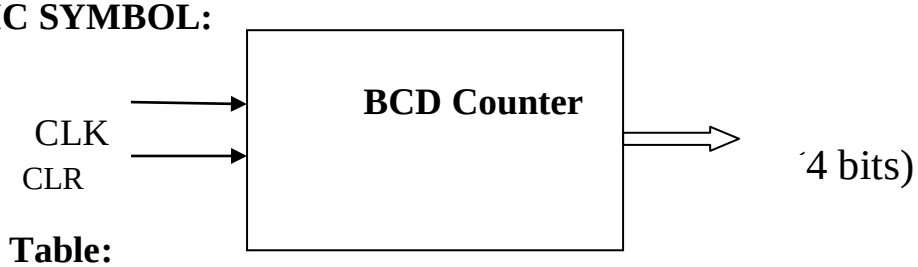


Exp.No:5. Write a Verilog code for 4-bit BCD synchronous counter and verify.

AIM:
To model 4 bit Synchronous BCD counter using verilog 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	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
```

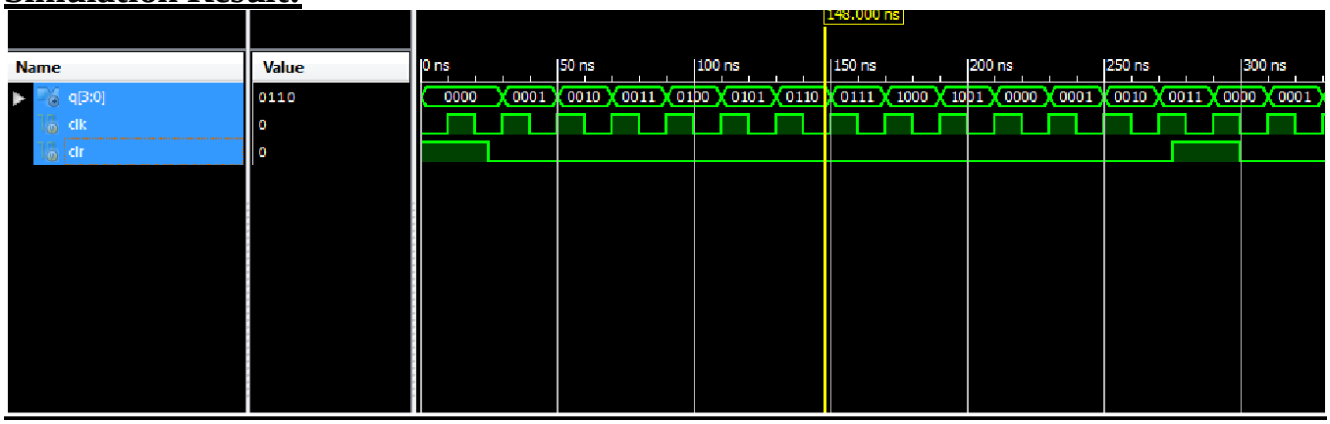
Test Bench Program:

```
`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:

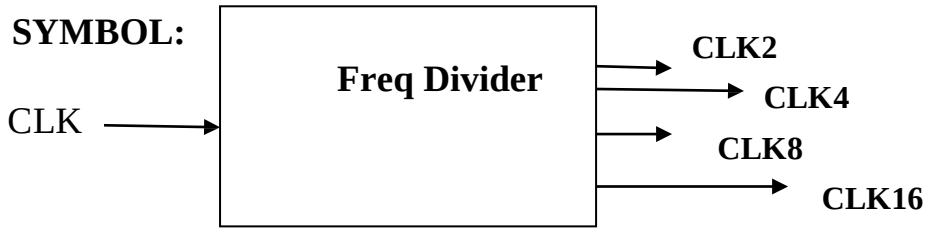


Exp.No:6. Write a Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

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 based CPLD 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
```

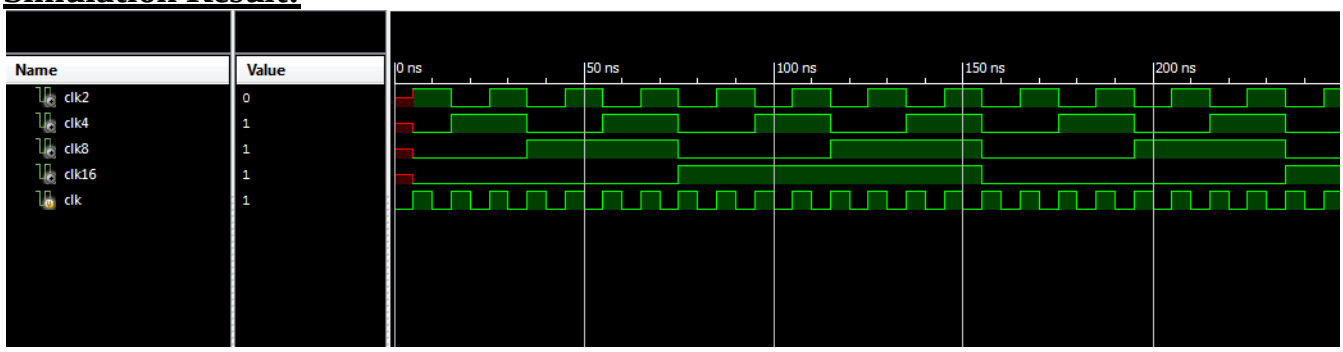
Test Bench Program:

```
`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:



Part –B

Q1. Interface a DC motor to FPGA and write Verilog code to change its speed and direction.

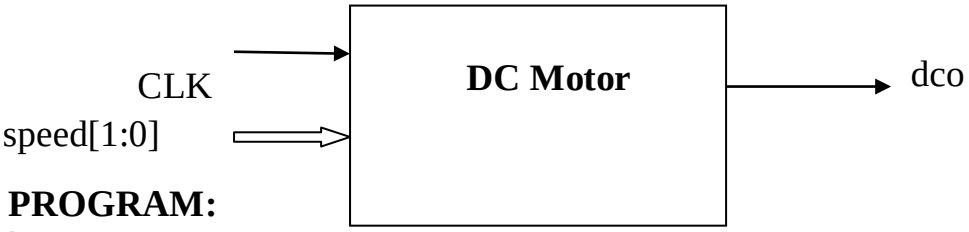
AIM: To Interface a DC motor to FPGA/CPLD and write Verilog code to change its speed and direction.

TOOLS REQUIRED:

i). Software : Xilinx ISE14.7

ii).Hardware: XC9572 based CPLD Kit,JTAG,Power Adapter,Flying Leads,dc motor interfacing 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 be changed.
2. The dco ouput can be given to in1 and in2 connection points for changing the direction of rotation.

Q2.

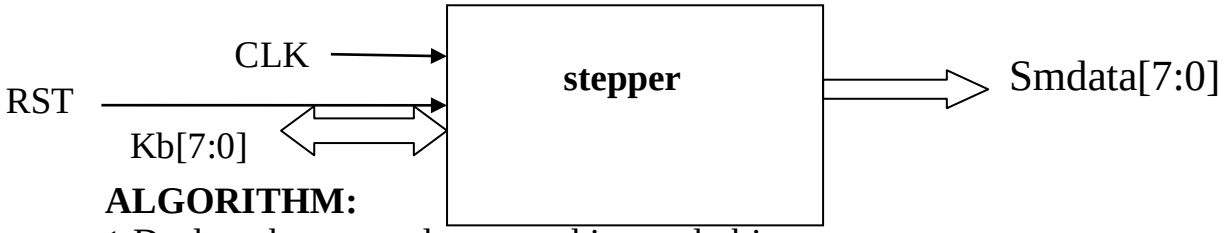
AIM: To Interface a Stepper motor to FPGA and write Verilog code to control the Stepper motor rotation which in turn may control a Robotic Arm. External switches to be used for different controls like rotate the Stepper motor (i) +N steps if Switch no.1 of a Dip switch is closed (ii) +N/2 steps if Switch 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,reset all the step control variables to zero else go to step 4.
4. Read the status of switch1 .If it is pressed,send the data to stepper motor coil in such a way that it will be moving in forward direction for N steps else go to step 5.
5. Read the status of switch2 .If it is pressed,send the data to stepper motor coil in 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 else stop.

[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 in clockwise direction for N steps.
3. By pressing sw2 of 4x4 hexakeypad, stepper motor will rotate in anticlockwise direction for N steps.
4. By pressing sw3 of 4x4 hexakeypad, stepper motor will rotate in clockwise direction for (N/2) steps.

Q3.

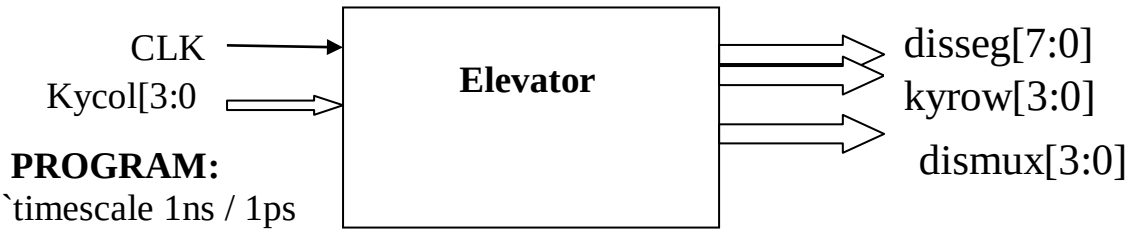
AIM: To write Verilog code using FSM to simulate elevator operation.

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**.**

LOGIC SYMBOL:



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;
```

```
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" ;
```

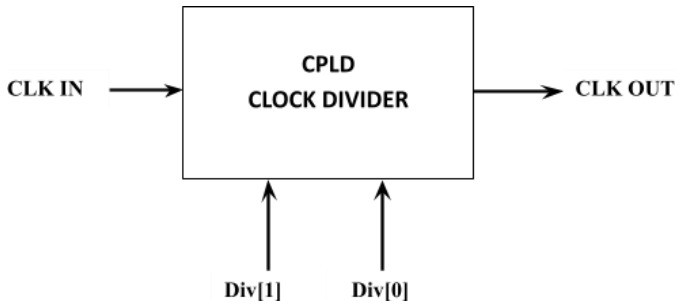
EXPECTED RESULT:

1. Hexkey 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 hexkey pad.
3. The transition from current floor to next floor (in steps) should be displayed in the 7 segment display.

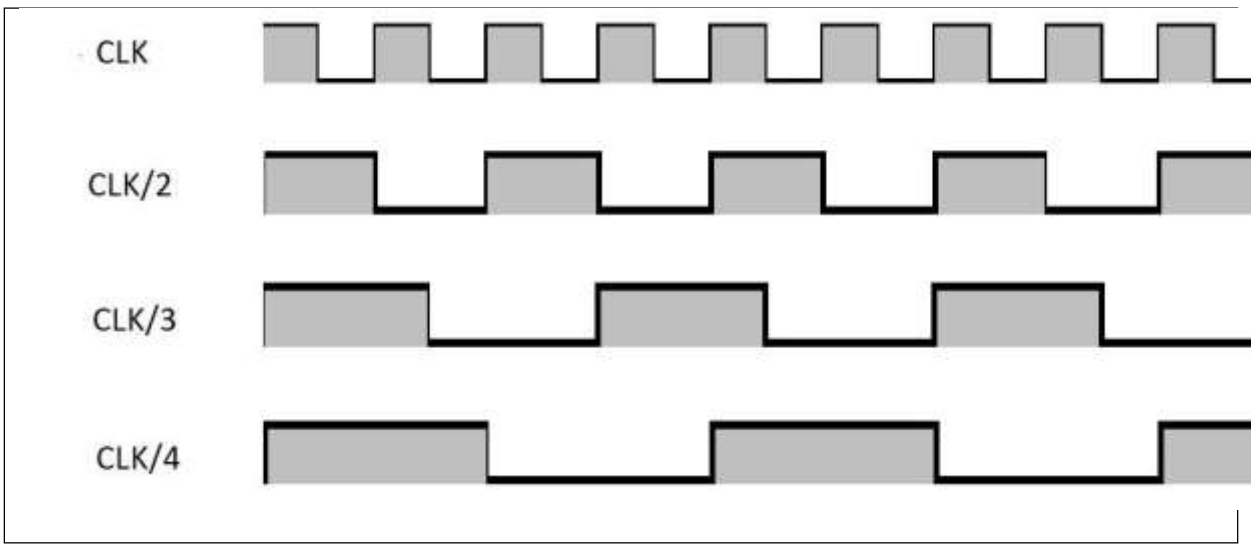
Q4.

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



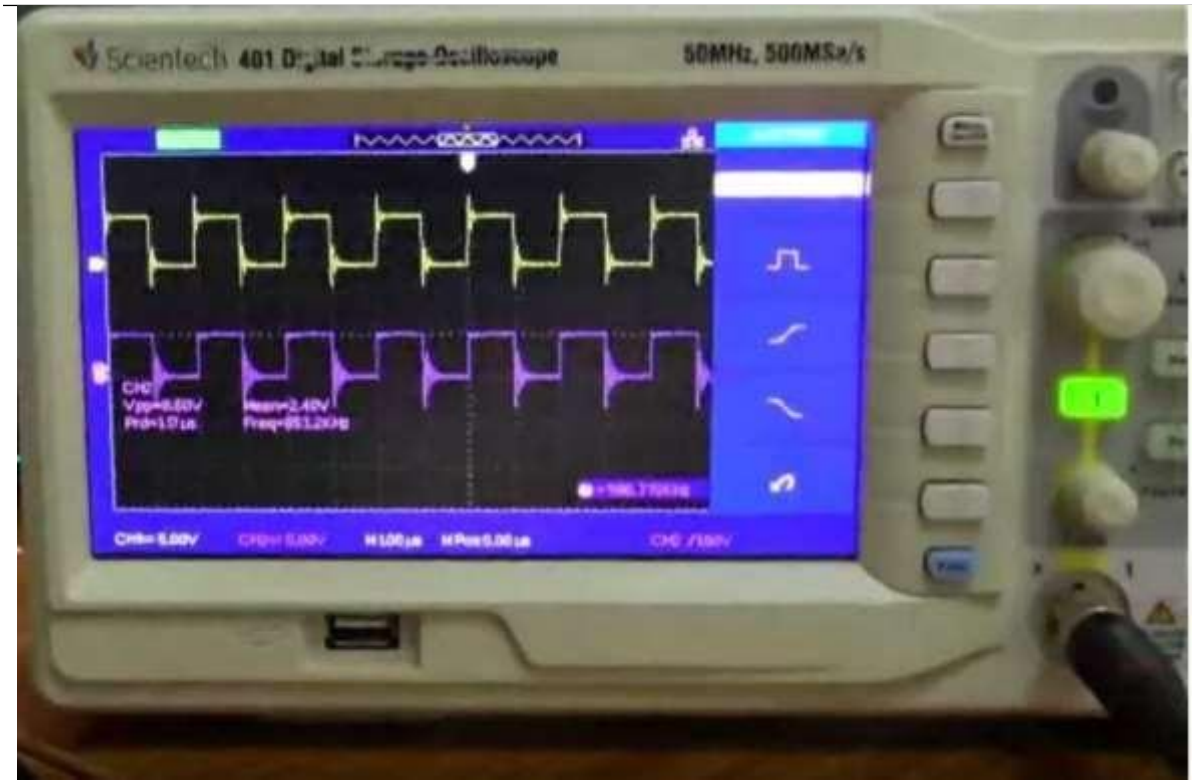
WAVEFORM



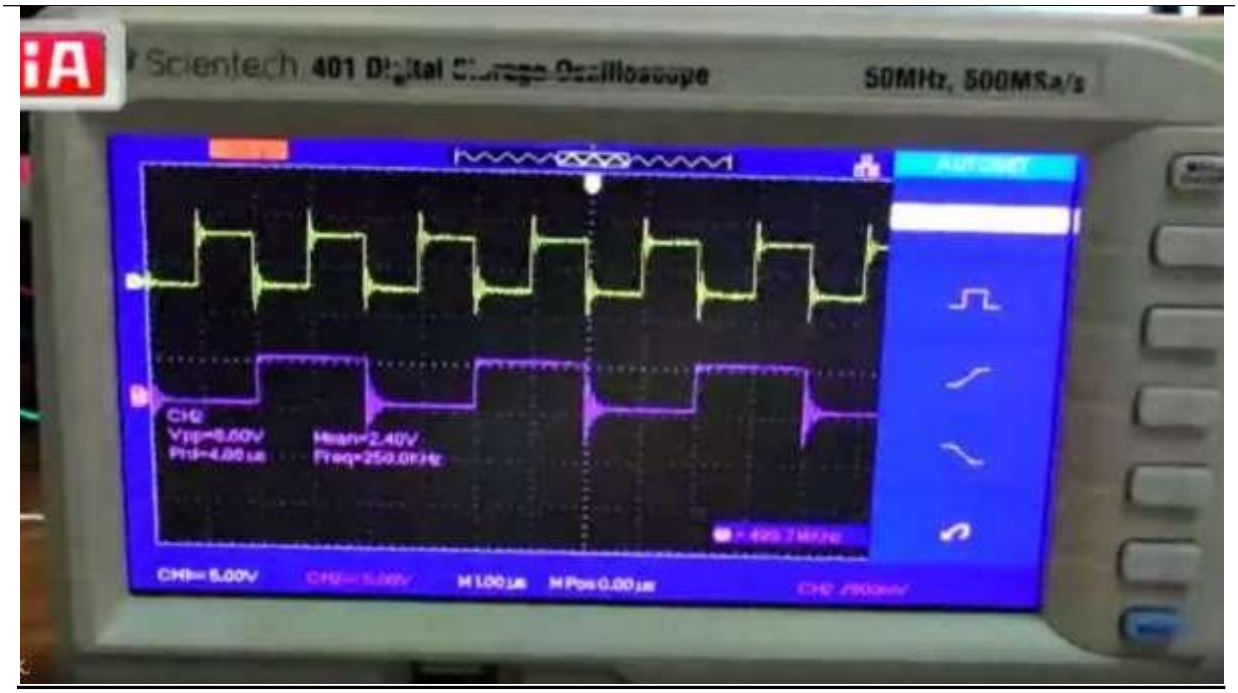
Program:

```
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
```

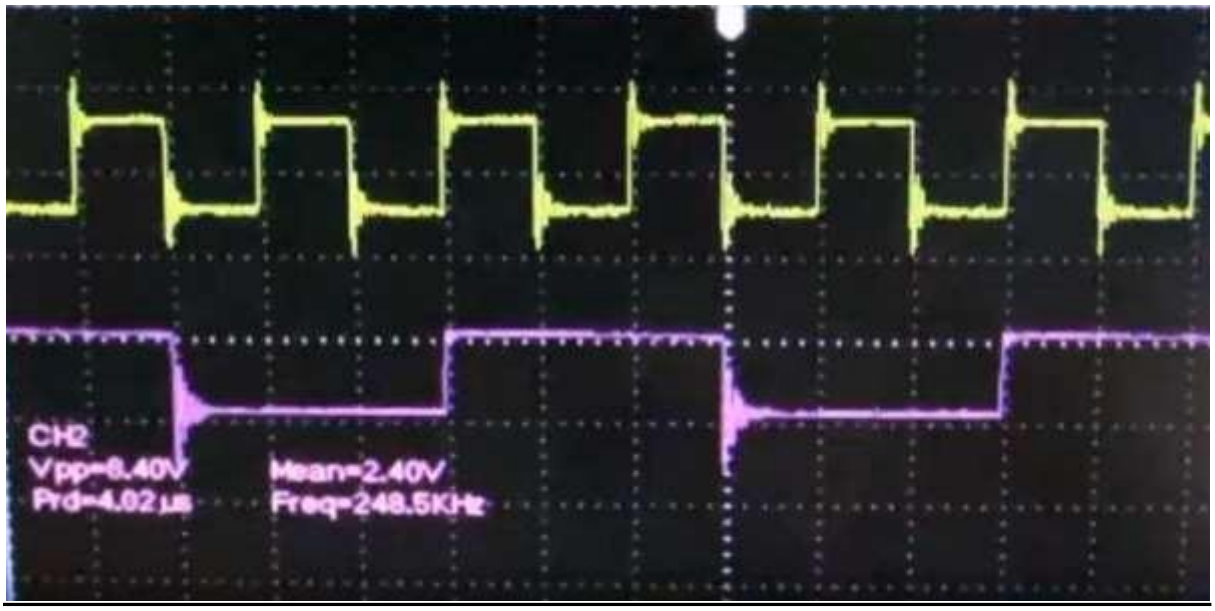
OUTPUT RESULTS:



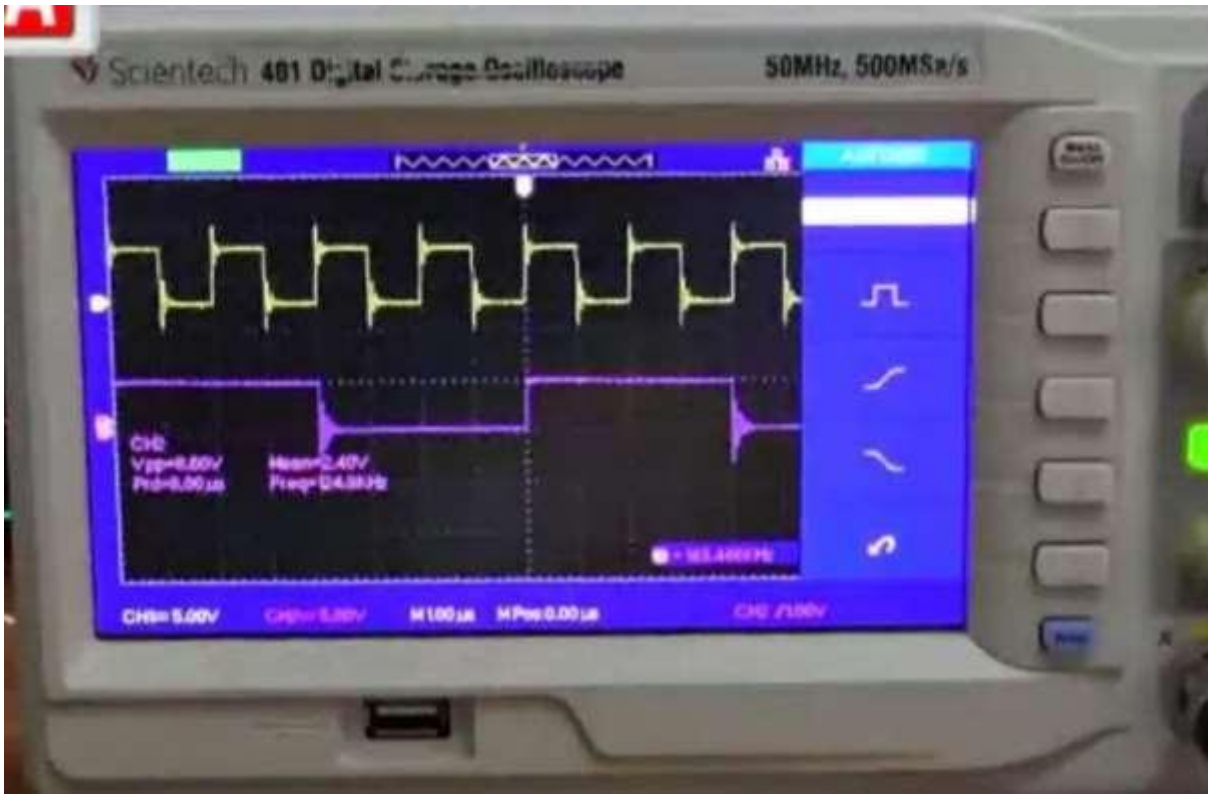
Output on Oscilloscope for DIVISOR=00, (input clock divide by 1)



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)

ATRIA INSTITUTE OF TECHNOLOGY

EXAMINATION QUESTION COMBINATIONS

Class: V-sem (EC-A,B,C)
Sub Code : 18ECL58 **Exam Hours: 03 Hrs**
Hrs/Week: 03 **Exam Marks: 100 Marks**

1. **a).Write a gate level Verilog description to realize 2:4 decoder using NAND logic .**
   b). Write Verilog code to simulate Elevator Operation.

2. **a). Write a behavioral description to realize the 8:3 encoder without priority .**
b).Interface a DC motor to CPLD and write Verilog code to change its speed and direction.

3. **a).Write a behavioral description to realize 8:3 encoder with priority.**
b). Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to CPLD and validate the functionality through oscilloscope.

4. **a). Write a behavioral description to realize 8x1 MUX using IF statement.**
b). Interface a Stepper motor to CPLD 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

5. **a).Write a behavioral description to realize 8X1 MUX using CASE construct.**
b). Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to CPLD and validate the functionality through oscilloscope.

6. **a).Write a behavioral description to model a 32 bit ALU .Simulate and verify the functionality.**
b). Interface a DC motor to CPLD and write Verilog code to change its speed and direction.

7. **a).Write a Verilog program to model FULL ADDER and AND,OR,XOR and XNOR gates and verify .**
b). Write Verilog code using FSM to simulate elevator operation.

8. **a).Write a Verilog code to SR Flip flop and verify.**
b). Interface a Stepper motor to CPLD 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

9. a).Write a Verilog code to JK flip flop and verify.
b). Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 1/4thclock from a given input clock. Port the design to CPLD and validate the functionality through oscilloscope.

10. a).Write a Verilog code to D flip flop and verify.
b). Write Verilog code using FSM to simulate elevator operation.

11. a).Write a verilog program to model 4 bit BCD synchronous counter and verify.
b). Interface a DC motor to CPLD and write Verilog code to change its speed and direction.

12. a). 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.
b). Interface a Stepper motor to CPLD 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

Department of Electronics and Communication Engineering

Vision

"Imparting quality technical education through interdisciplinary research & innovation towards moulding the young talent with professional competence and ethical values for developing inclusive and sustainable technology in the area of Electronics and Communication Engineering".

Mission

Create conducive environment for the holistic development of students and staff members.

- Provide quality technical education to produce industry ready engineers with an entrepreneurial and research outlook.
- Establish centers of excellence in collaboration with industries/universities for exposing the students to latest technologies.
- Nurture the students to actively participate in solving the societal problems and uphold ethics and morality.
- To train the students to meet global challenges in interdisciplinary fields by inculcating a quest for modern technologies in the emerging areas.