



SIDDHARTHA
INSTITUTE OF TECHNOLOGY & SCIENCES
Korremula Road, Narapally, Ghatkesar Mandal, Ranga Reddy District.



LABORATORY MANUAL

For

E-CAD LAB

(III B. Tech ECE- II Semester- R18 .AY:2021 - 22)

Prepared by

- 1. Mr. M. RANJITH REDDY, Asst. Professor**
- 2. Mr. K. RAJESH, Asst. Professor**

Department of
Electronics & Communication Engineering
2022

www.siddhartha.co.in

SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES

CONTENTS

S. No	Description	Page No.
1	Vision & Mission of the Institute	i
2	Vision & Mission of the Department	ii
3	Program Outcomes	iii
4	Rules and Regulations of Lab	iv
5	Introduction to Xilinx	1-11
6	HDL code to realize all logic gates	12-13
7	Design of 2-to-4 encoder	15-16
8	Design of 8-to-3 encoder	17-19
9	Design of 8-to-1 multiplexer and 1x8 demultiplexer	20-21
10	Design of 4 bit binary to gray code converter	22-23
11	Design of 4-bit comparator	24-26
12	Design of full adder using three modelling styles	27-32
13	Design of flip flops (SR,JK,D,T)	33-35
14	Finite state machine design	36-37
15	Design & Implementation of an Inverter	38-40
16	NAND Gate	41-44
17	NOR Gate	45-47
18	X-OR Gate	48-52

ADDITIONAL EXPERIMENTS

SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES

VISION & MISSION OF THE INSTITUTE

VISION:

To be a Centre of Excellence in Technical Education and to become an epic center of Research for creative solutions.

MISSION:

To address the Emerging Needs through Quality Technical Education with an emphasis on practical skills and Advanced Research with social relevance.

OBJECTIVES:

- To translate our vision into action and accomplish our mission, we strive to provide state-of-art infrastructure.
- Recruit, Motivate and develop faculty of high caliber and with multiple specialization.
- Continuously review, innovate and experiment teaching methodologies and learning processes.
- Focus on research, training and consultancy through an Integrated Institute-Industry symbiosis.

SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES

VISION & MISSION OF THE DEPARTMENT

VISION:

To provide innovative teaching and learning methodologies for excelling in a high-value career, higher education and research to the students in the field of Electronics and Communication Engineering to meet the needs of the industry and to be a part of the advancing technological revolution.

MISSION:

- To create engineers of high quality on par with international standards by providing excellent infrastructure and well qualified faculty.
- To establish centers of excellence to enhance collaborative and multidisciplinary activities to develop human and intellectual qualities.
- To provide technical expertise to carry out research and development.

PROGRAM EDUCATIONAL OBJECTIVES (PEOS) :

Graduates shall apply the fundamental, advanced and contemporary knowledge of

1. Electronics, Communication and allied Engineering, to develop efficient solutions and systems, to meet the needs of the industries and society.
2. Graduates will get employed or pursue higher studies or research.
3. Graduates will have team spirit, good communication skills and ethics with lifelong learning attitude.

SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES













PROGRAM OUTCOMES:

Engineering Graduates will be able to:





1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

RULES AND REGULATIONS OF LAB

All students must observe the Dress Code while in the laboratory.

-  All bags must be placed at rack.
-  The lab timetable must be strictly followed.
-  Be PUNCTUAL for your laboratory session.
-  Program/experiment must be executed within the given time.
-  Workspace must be kept clean and tidy at all time.
-  Handle the systems and interfacing kits with care.
-  All students are liable for any damage to the accessories due to their own negligence.
-  All interfacing kits connecting cables must be RETURNED if you taken from the lab supervisor.
-  Students are strictly PROHIBITED from taking out any items from the laboratory.
-  Students are NOT allowed to work alone in the laboratory without the Lab Supervisor
-  USB Ports have been disabled if you want to use USB drive consult lab supervisor.
-  Report immediately to the Lab Supervisor if any malfunction of the accessories, is there

Before leaving the lab

-  **Place the chairs properly.**
-  Turn off the system properly
-  Turn off the monitor.
-  Please check the laboratory notice board regularly for updates.

INTRODUCTION - XILINX

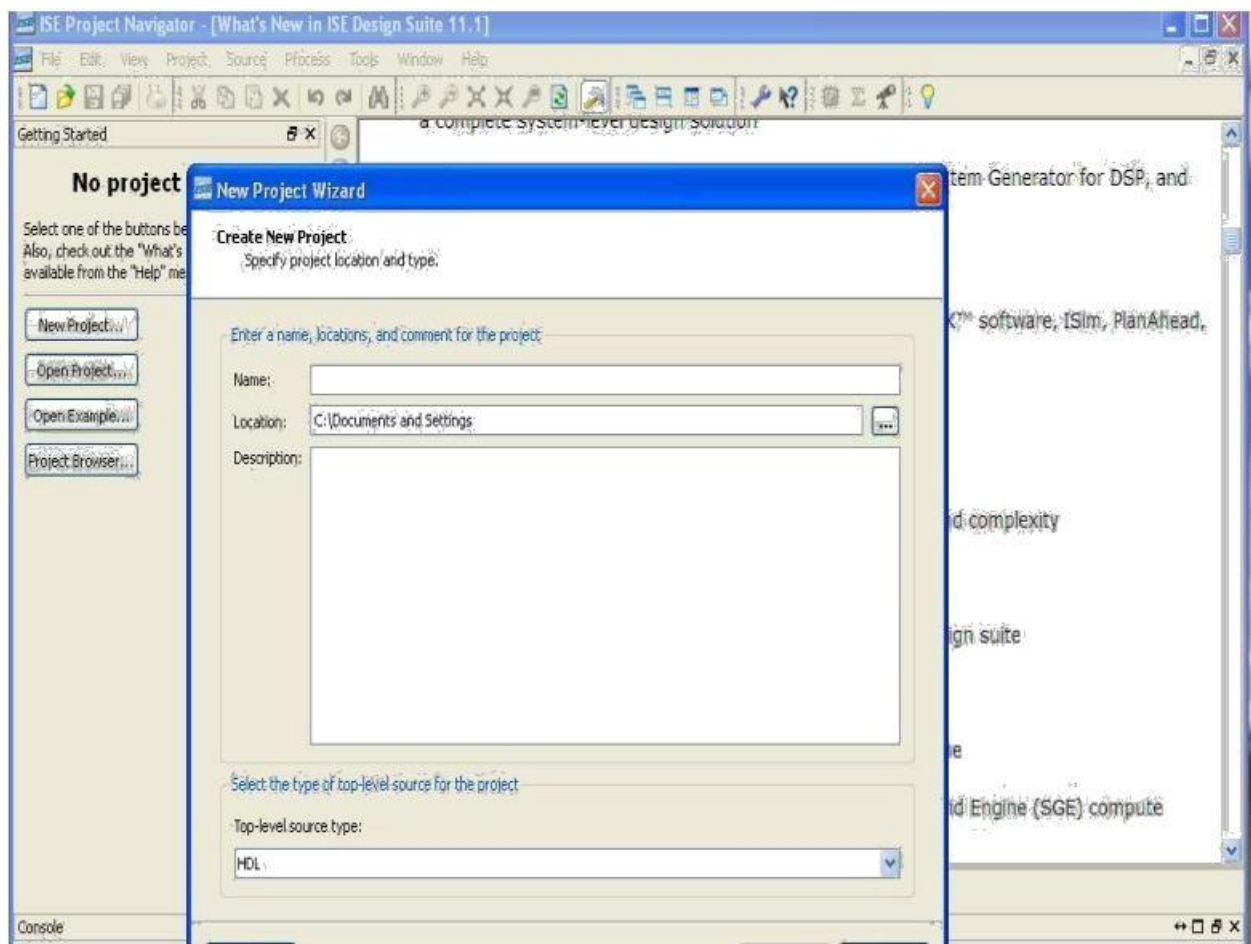
Xilinx ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

In our Lab, the scope is limited to design and analyze the design using test benches & simulation.

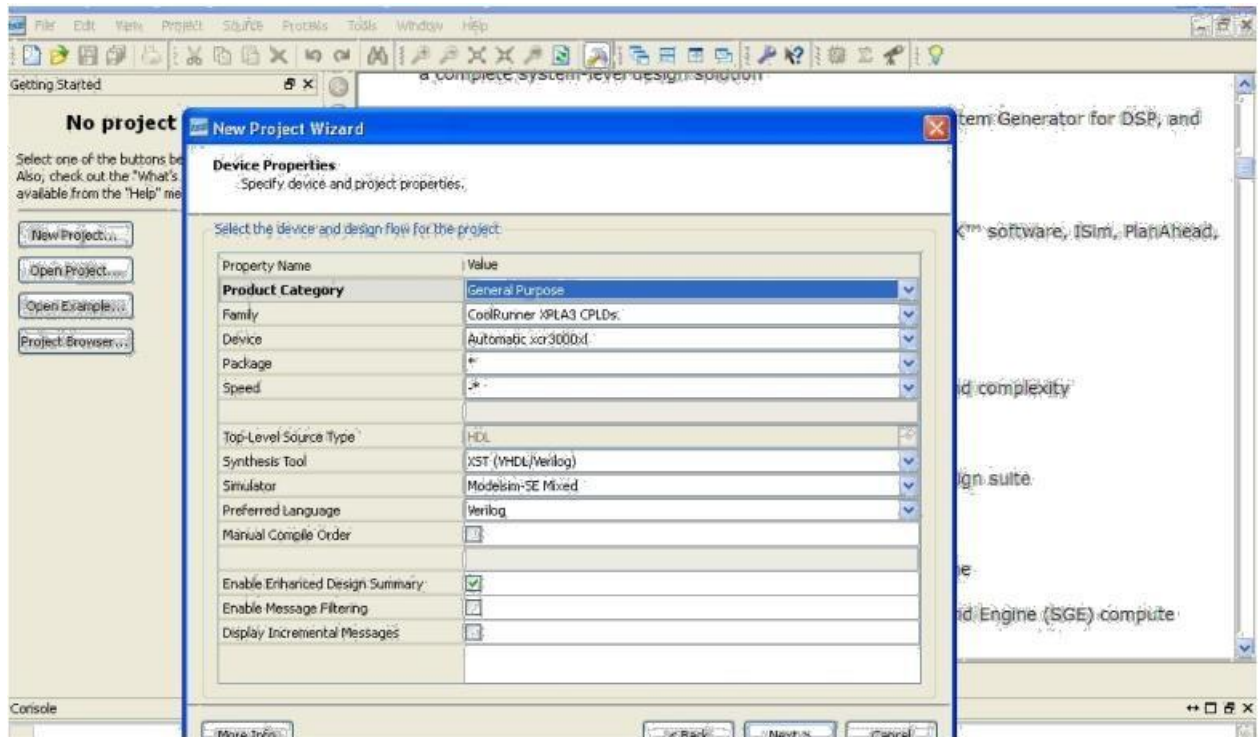
The following is the step by step procedure to design in the Xilinx ISE:

1. New Project Creation

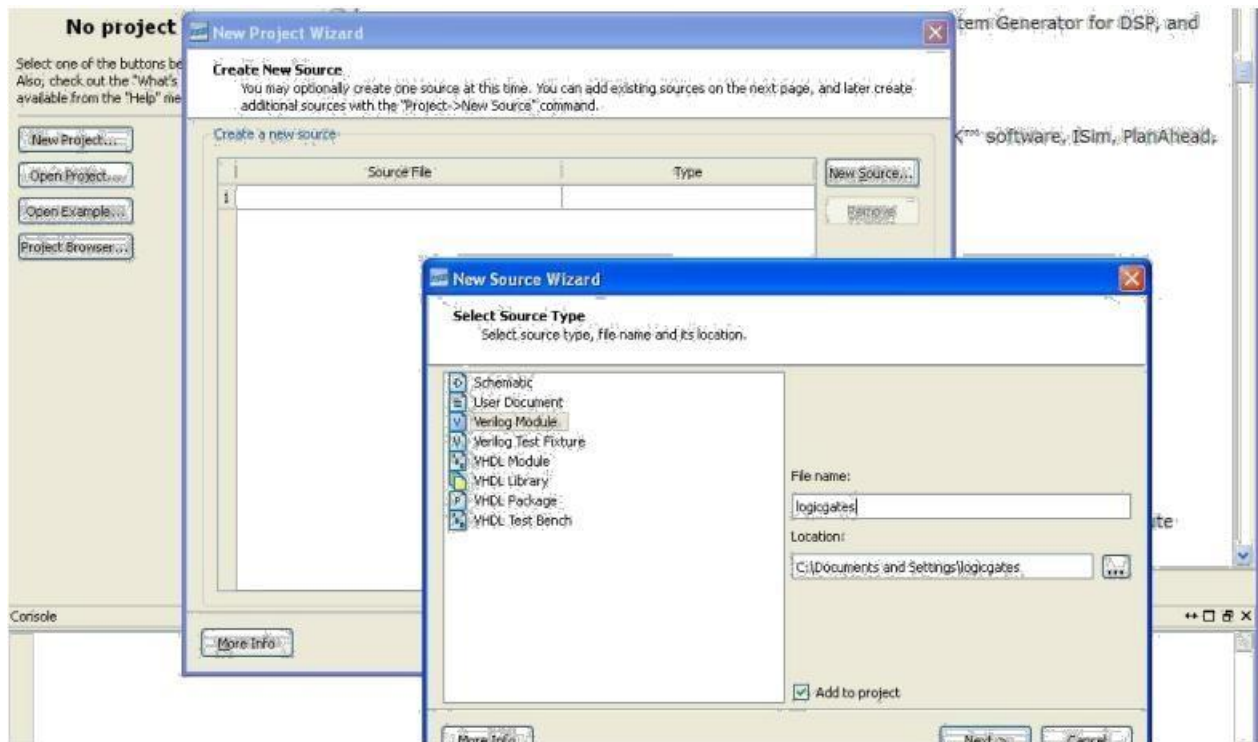
Once the Xilinx ISE Design suite is started, open a new project & enter your design name and the location path. By default 'HDL' is selected as the top-level source type. (If not, please select Top-level source type as 'HDL')



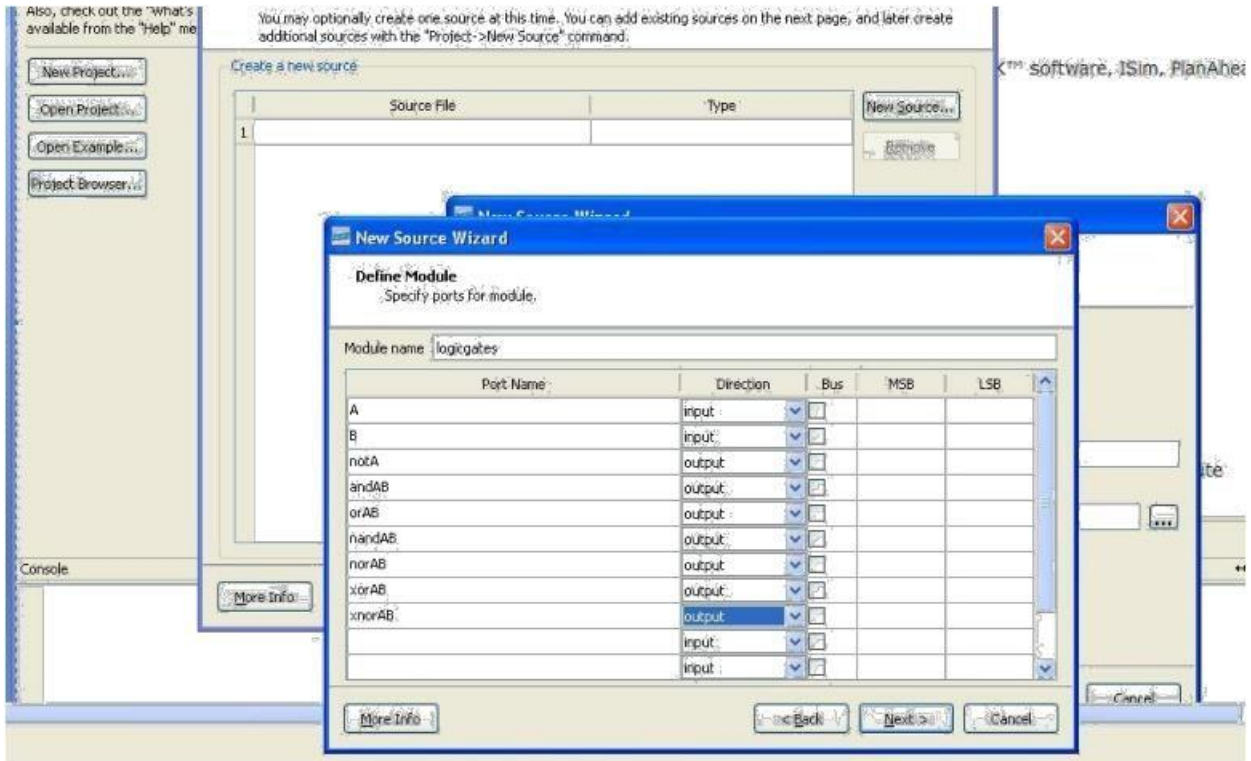
2. Continue to the next window and check if the Preferred Language is selected as 'Verilog'



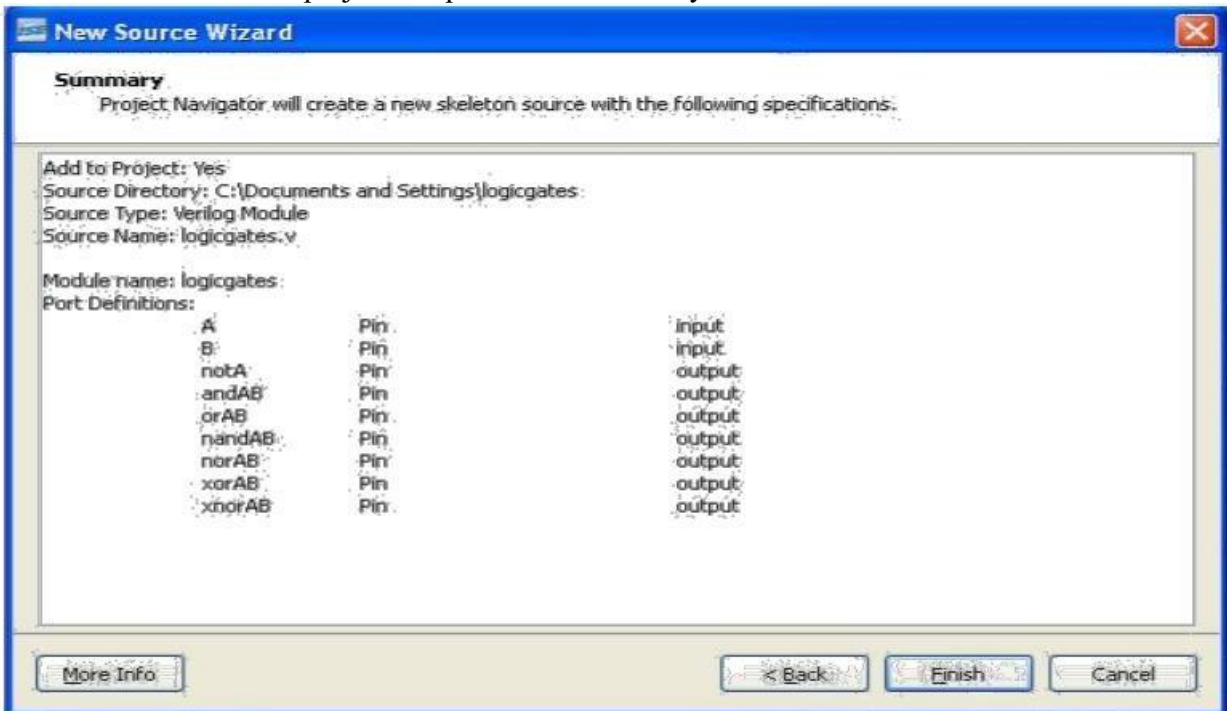
3. Proceed by clicking 'Next' and create a 'New Source' using the 'Create New Source' Window



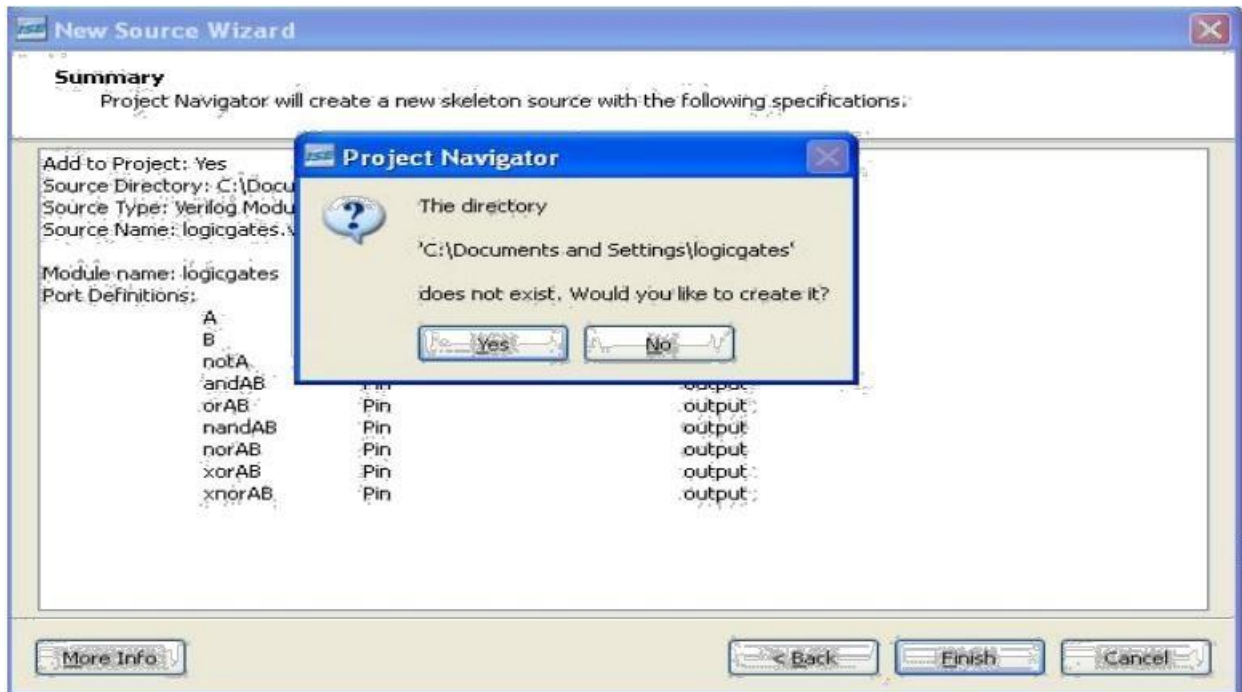
4. Select the source type as 'Verilog Module' and input a filename and proceed to 'Next'. In the next window 'Define Module' enter the ports.



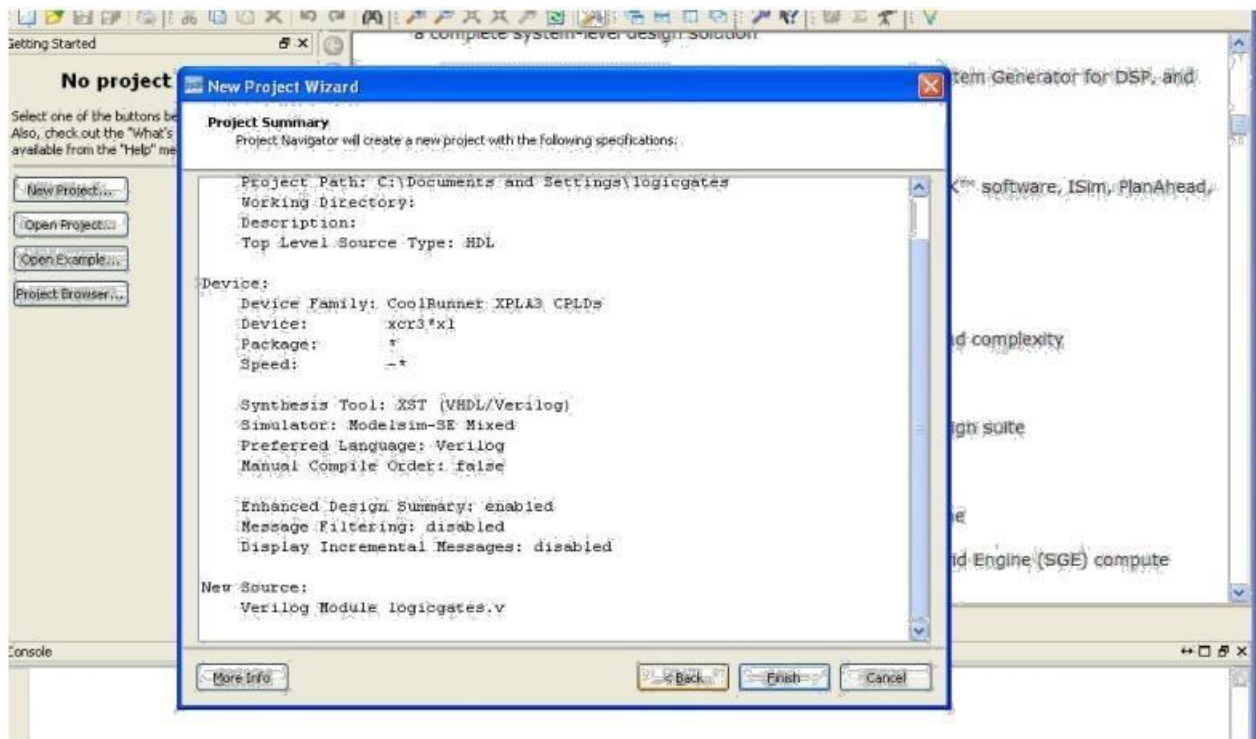
5. Finish with the New project setup with the 'Summary' window.



6. Once 'Finish' is selected a pop-up appears to create the directory. Select 'yes'

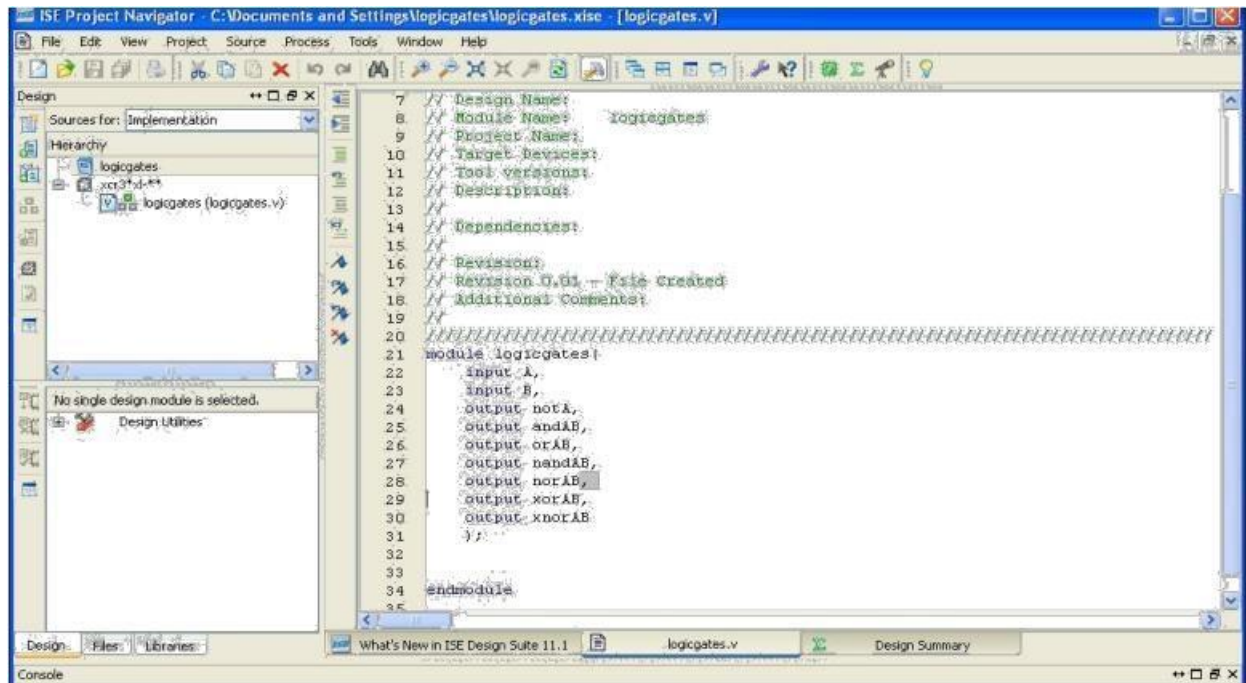


7. Then proceed to 'Next' in the "New Project Wizard" to 'Add Existing Sources'. 'Add source' if an existing source is available, If not proceed to 'Next' and finish with the 'Project Summary' window

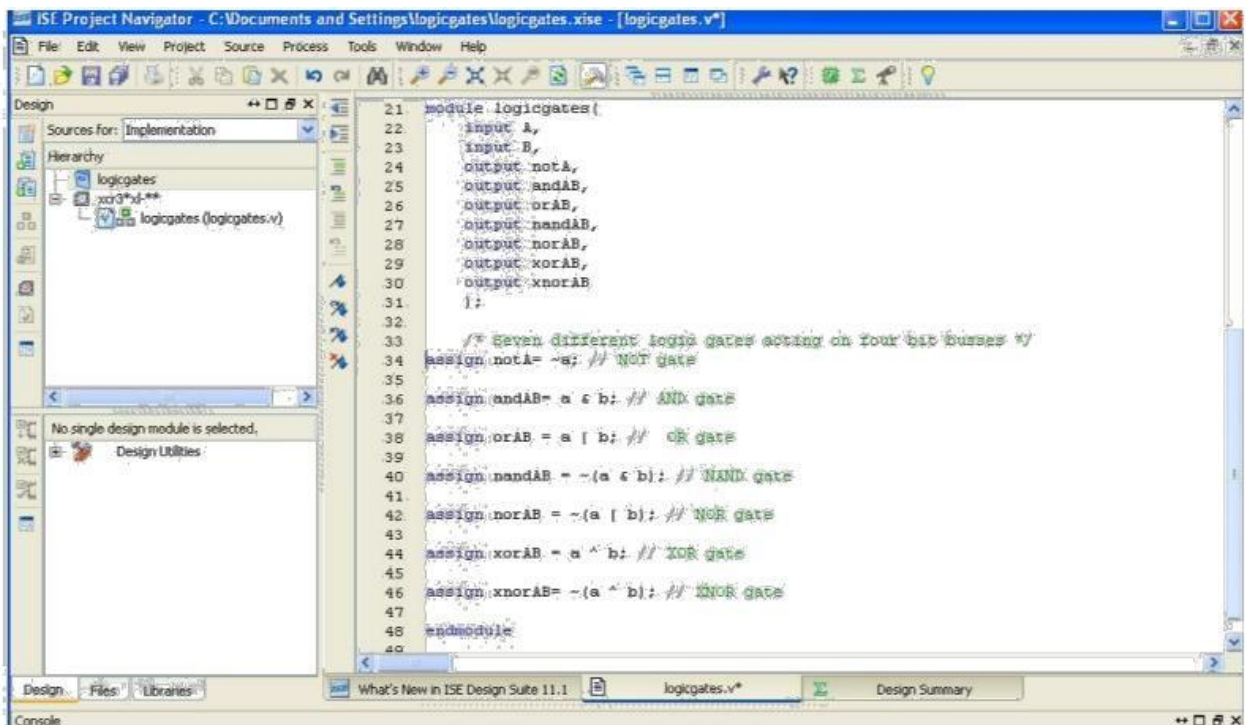


8. Design Entry and Syntax Check

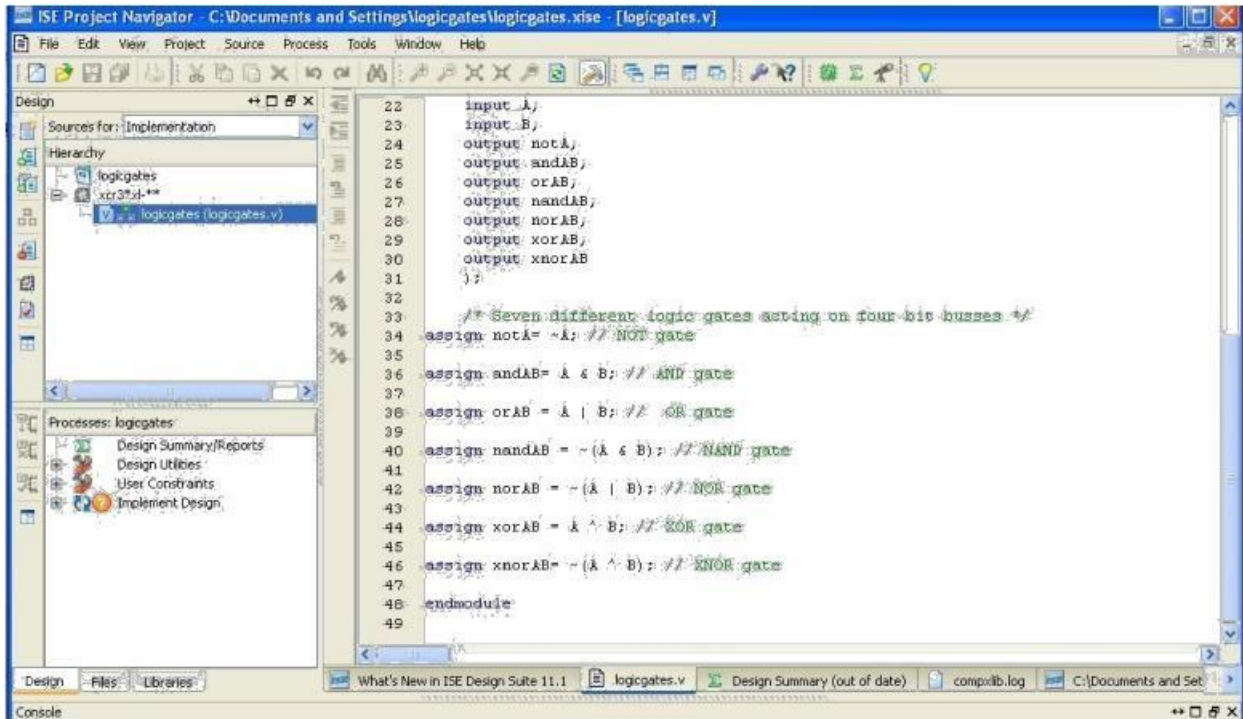
The ports defined during the 'Project Creation' are defined as a module in the 'filename.v' file



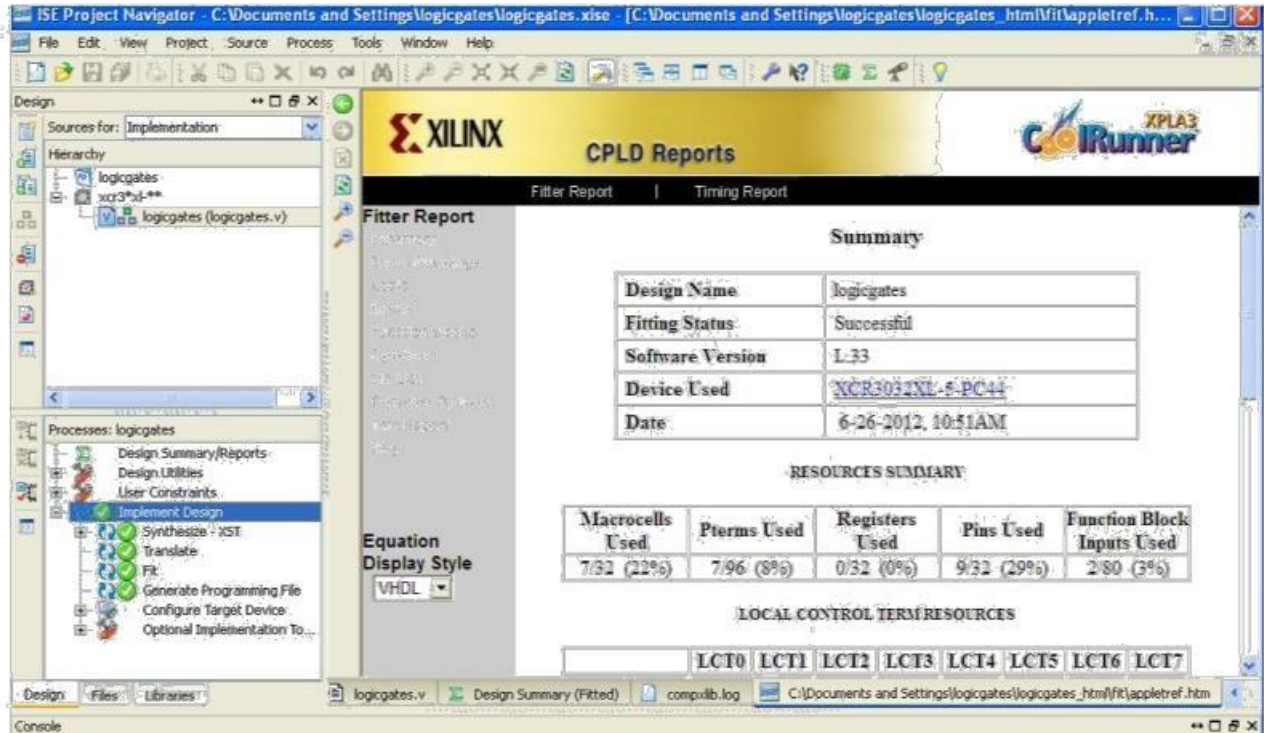
9. Input your design (verilog code) within the module definition



10. Select the design from the 'Hierarchy' window. In the below window of Processes 'Implement Design' would be orange (in color) ready for implementation

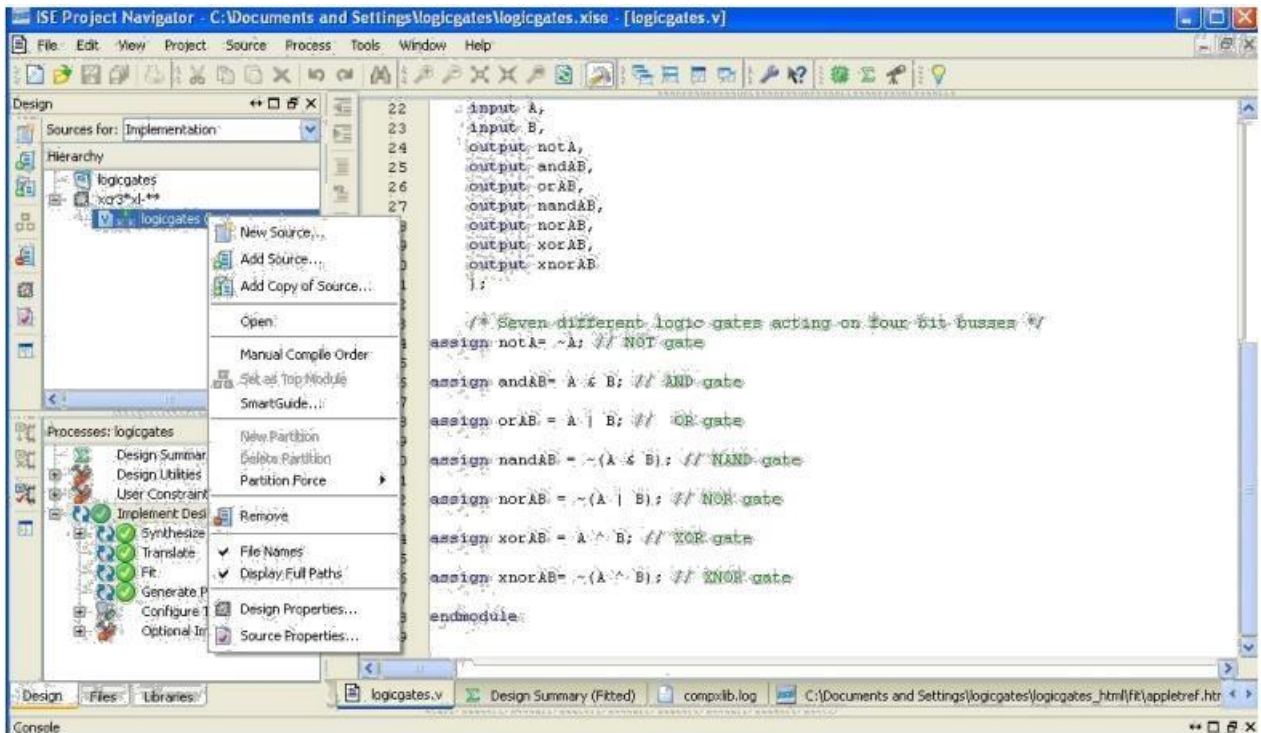


11. Double click on implement design, it turns green (in color) once the design is implemented successfully and the Summary report is displayed.

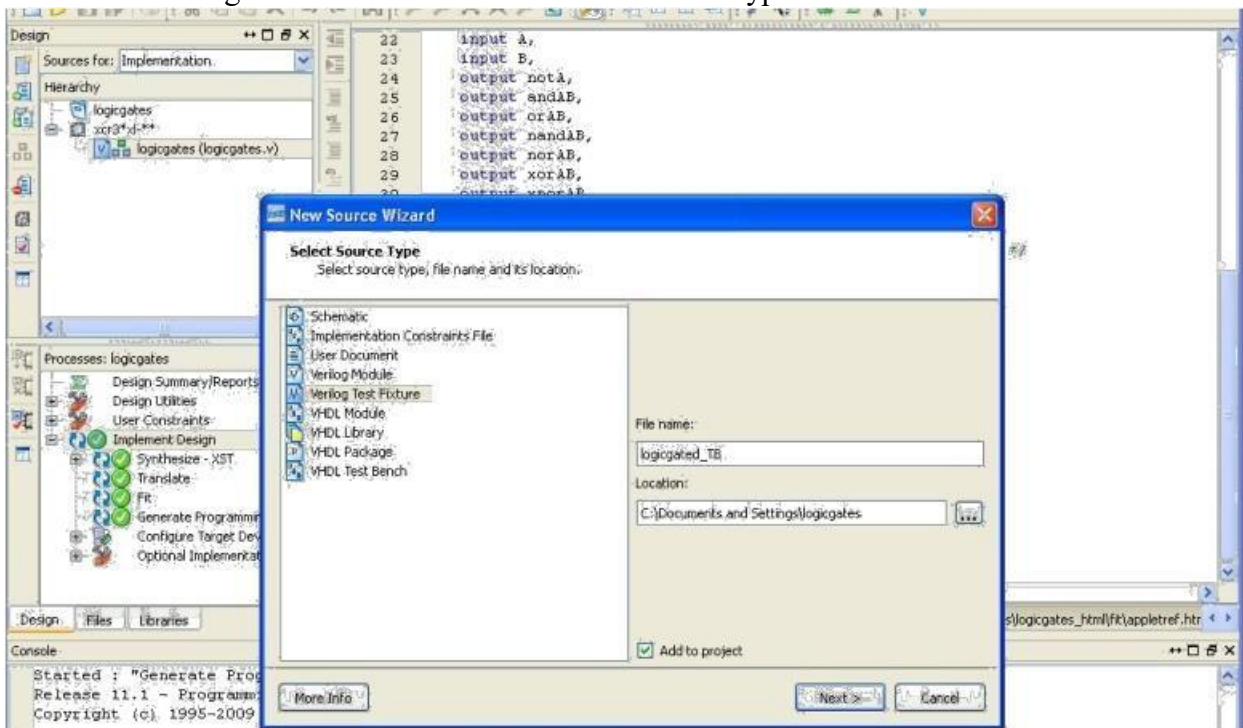


12. Test-Bench creation, Simulation & Verification

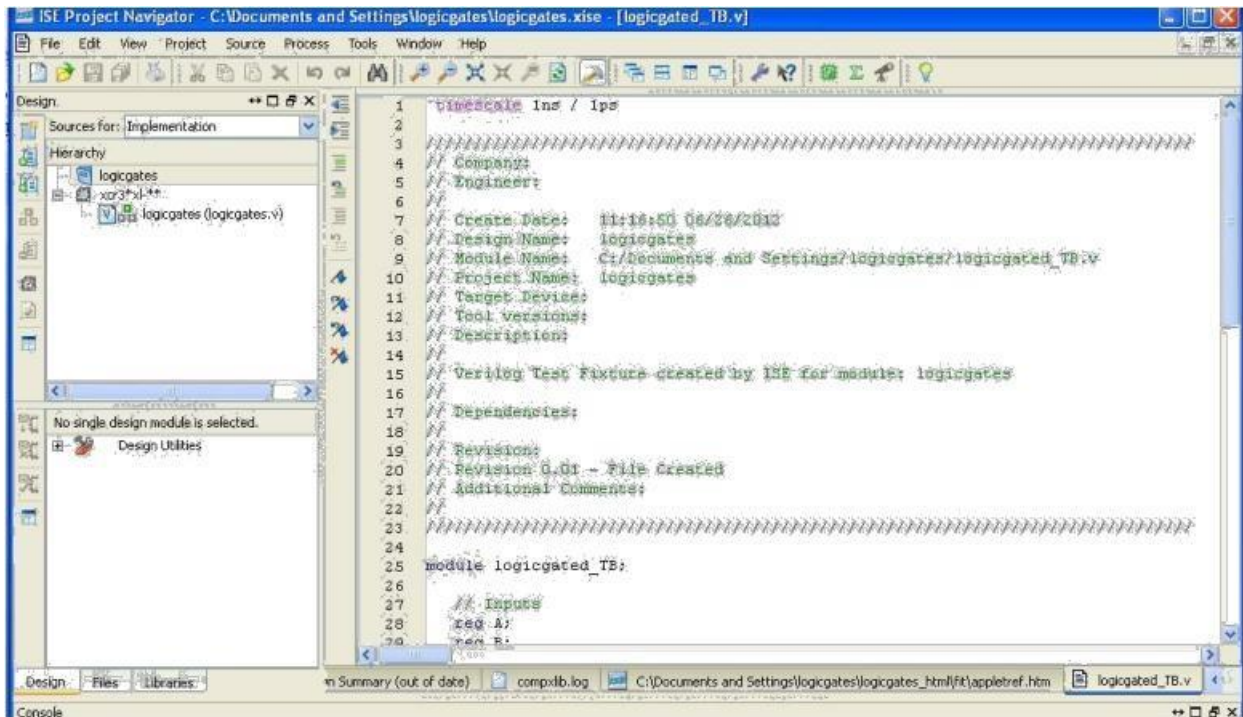
To add a test-bench to the existing design, right click on the '.v' file from the Hierarchy window and select 'New Source'



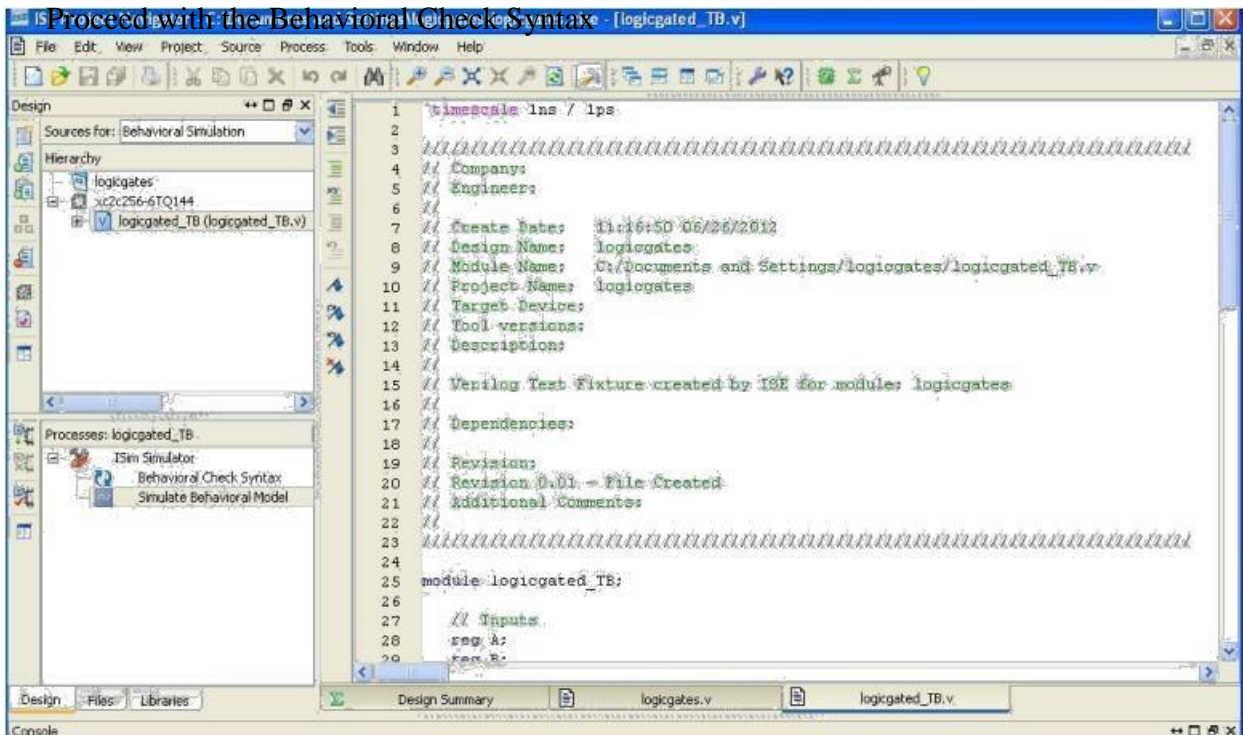
13. Select 'Verilog Text Fixture' from the Select Source Type and name the Test-Bench



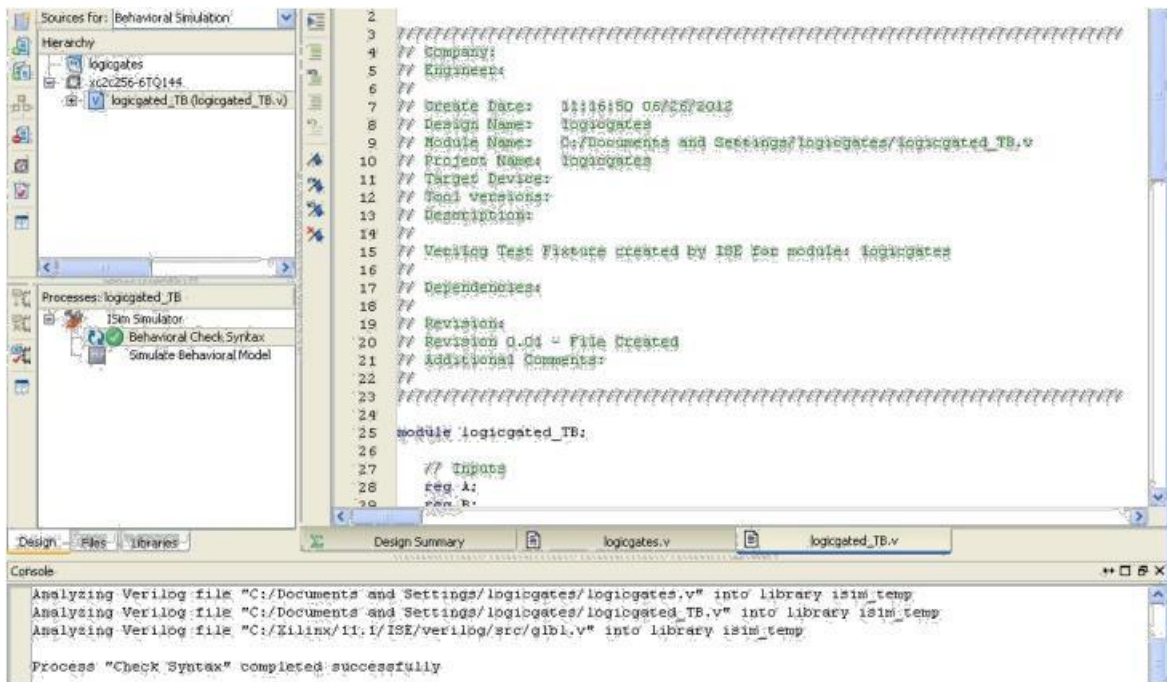
14. Continue to 'Finish' and a test bench is added in the project area



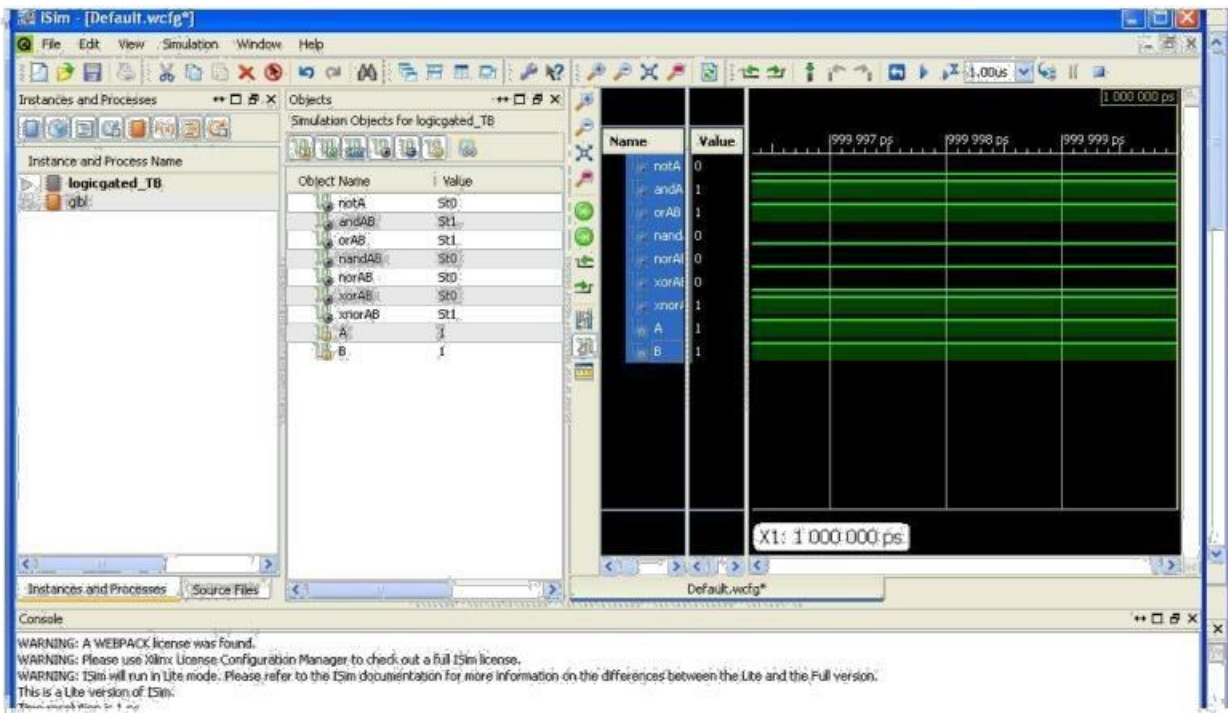
15. Edit the test bench as per your simulation requirements and select 'Behavioral Simulation' in the 'Design Window'. In the Processes window Isim Simulator would be displayed. First Proceed with the Behavioral Check Syntax



16. Double click on 'Behavioral Check Syntax' & check for no errors



17. Then double click on 'Simulate Behavioral Model' and the ISIM simulator window would open. Check for the outputs



EXPERIMENT No: 1

HDL CODE TO REALIZE ALL LOGIC GATES

AIM:

To develop the source code for logic gates by using VERILOG and obtain the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

LOGIC DIAGRAM:

AND GATE:

LOGIC DIAGRAM:

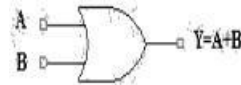


TRUTH TABLE:

A	B	Y=AB
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

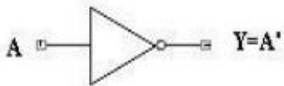
LOGICDIAGRAM TRUTH TABLE:



A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

LOGIC DIAGRAM:



TRUTH TABLE:

A	Y=A'
0	1
1	0

NAND GATE:

LOGICDIAGRAM TRUTH TABLE



A	B	Y=(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:

LOGIC DIAGRAM:



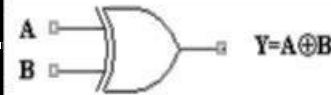
TRUTH TABLE:

A	B	Y=(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

XOR GATE:

LOGICDIAGRAM

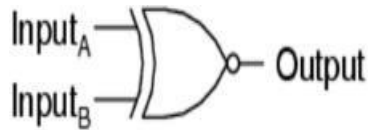
TRUTH TABLE:



A	B	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR GATE:

LOGIC DIAGRAM:



TRUTH TABLE:

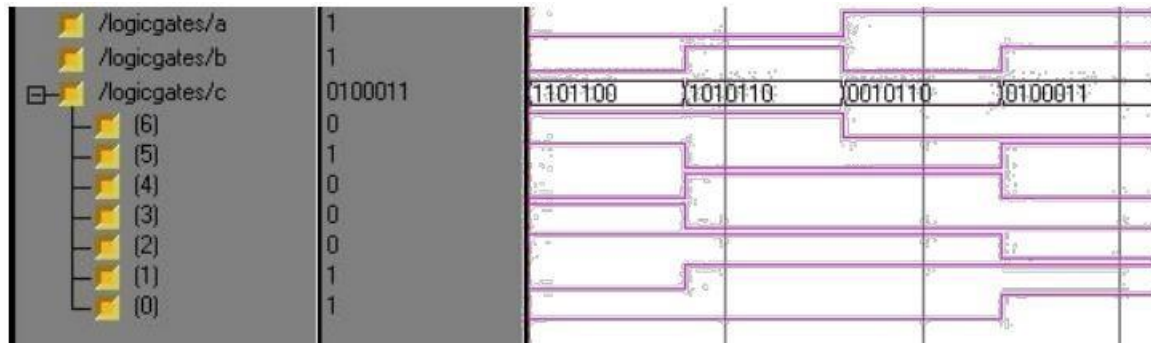
INPUTS		OUTPUT
A	B	$Y = A \oplus B$ $= AB + \overline{A} \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	1

VERILOG SOURCE CODE:

```
module logicgates1(a, b, c);  
    input a;  
    input b;  
    OUTPUT: [6:0] c;  
    assign c[0]= a & b;  
        assign c[1]= a | b;  
        assign c[2]= ~(a & b);  
        assign c[3]= ~(a | b);  
        assign c[4]= a ^ b;  
        assign c[5]= ~(a ^ b);  
        assign c[6]= ~ a;
```

```
endmodule
```

SIMULATION OUTPUT:



RESULT:

Thus the OUTPUT's of all logic gates are verified by simulating the VERILOG code.

EXPERIMENT No: 2

DESIGN OF 2-TO-4 ENCODER

AIM:

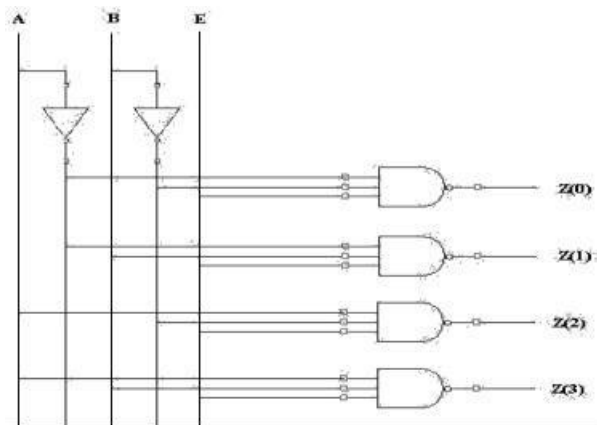
To develop the source code for encoder by using VERILOG and obtain the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

DECODER

LOGIC DIAGRAM:



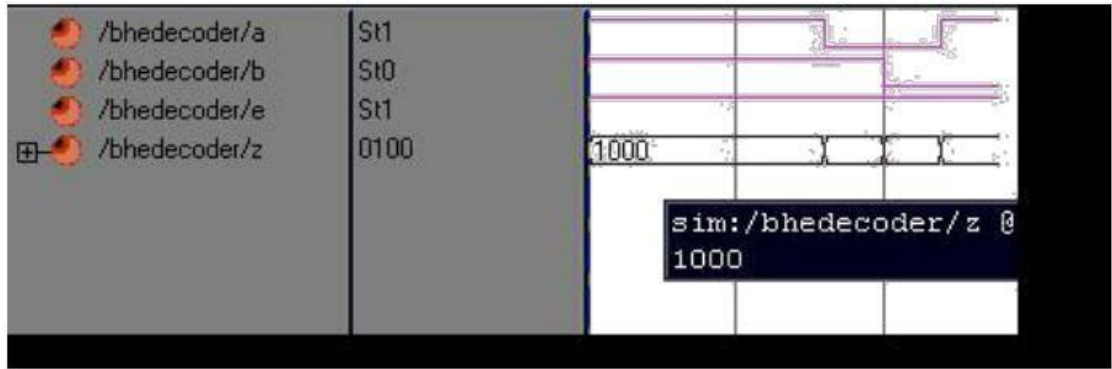
TRUTH TABLE:

A	B	C	Z(0)	Z(1)	Z(2)	Z(3)
0	0	1	0	1	1	1
0	1	1	1	0	1	1
1	0	1	1	1	0	1
1	1	1	1	1	1	0

VERILOG SOURCE CODE:

```
module decoderbehv(a, b, en, z);
    input a;
    input b;
    input en;
    output [3:0] z;
    reg [3:0] z;
    reg abar, bbar;
    always @ (a,b,en) begin
        z[0] = (abar&bbar&en);
        z[1] = (abar&b&en);
        z[2] = (a&bbar&en);
        z[3] = (a&b&en);
    end
endmodule
```

SIMULATION OUTPUT:



RESULT:

Thus the OUTPUT's of encoder are verified by simulating the VERILOG code.

EXPERIMENT No: 3**DESIGN OF 8-TO-3 ENCODER****AIM:**

To develop the source code for encoder by using VERILOG and obtain the simulation.

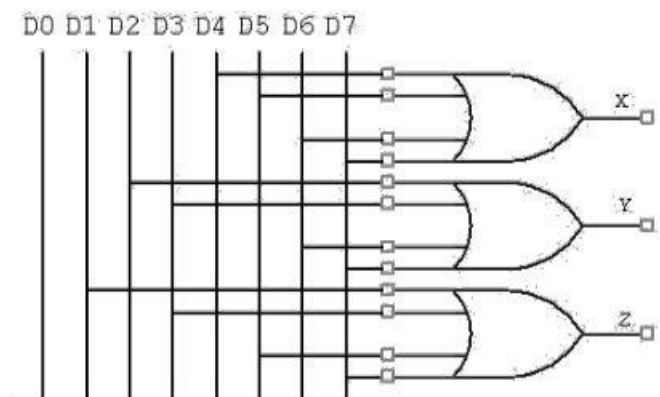
SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

ENCODER:

LOGIC DIAGRAM:

TRUTH
TABLE:



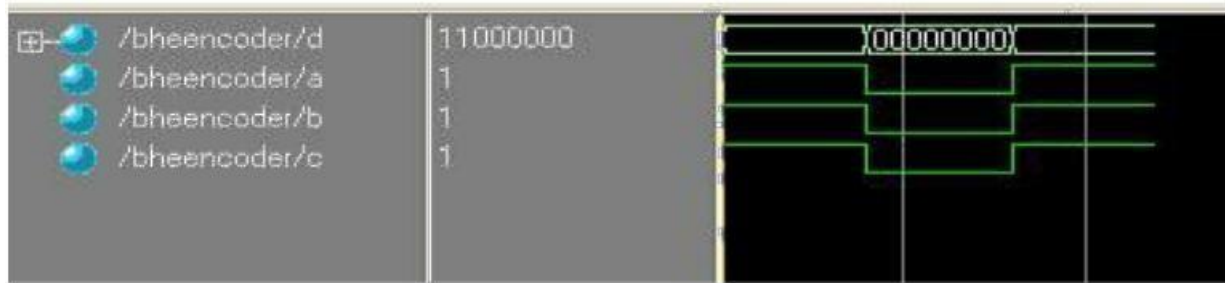
D0	D1	D2	D3	D4	D5	D6	D7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

VERILOG SOURCE CODE:

```

module encoderbehav(d, a,b,c);
  input [7:0] d;
  output x;
  output y;
  output z;
  reg a,b,c;
  always @ (d [7:0]) begin
    a= d[4] | d[5] | d[6] | d[7];
    b= d[2] | d[3] | d[6] | d[7];
    c= d[1] | d[3] | d[5] | d[7];
  end
endmodule

```

SIMULATION OUTPUT:**RESULT:**

Thus the OUTPUT's of Encoded are verified by simulating the VERILOG code.

EXPERIMENT No: 4**DESIGN OF 8-to-1MULTIPLEXER AND 1X8 DEMULTIPLEXER****AIM:**

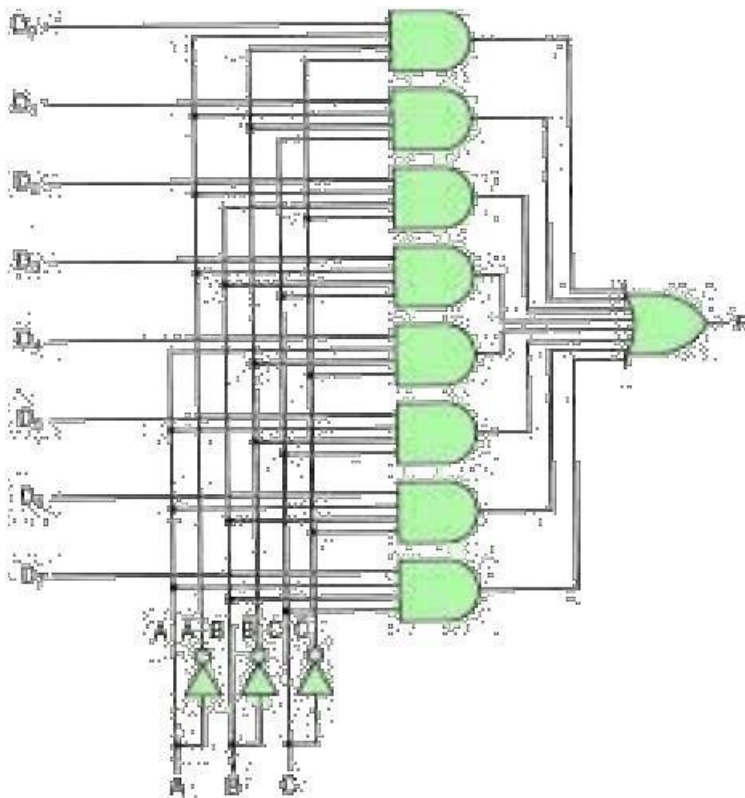
To develop the source code for 8x1 multiplexer and demultiplexer by using VERILOG and obtain the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

MULTIPLEXER:

LOGIC DIAGRAM:



TRUTH TABLE:

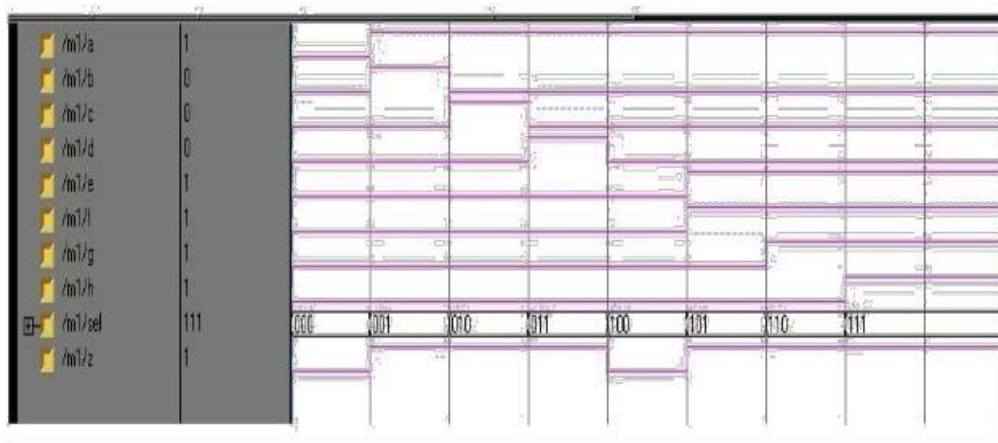
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

VERILOG SOURCE CODE:

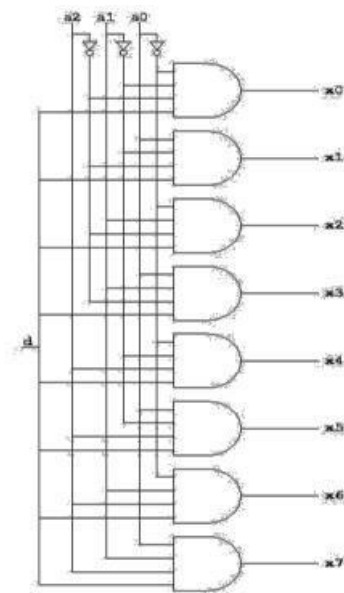
```

module MUX8TO1(sel, A,B,C,D,E,F,G,H, MUX_OUT);
input [2:0] sel;
input A,B,C,D,E,F,G,H;
output reg MUX_OUT;
always@(A,B,C,D,E,F,G,H,sel)
begin
case(sel)
3'd0:MUX_OUT=A;
3'd1:MUX_OUT=B;
3'd2:MUX_OUT=C;
3'd3:MUX_OUT=D;
3'd4:MUX_OUT=E;
3'd5:MUX_OUT=F;
3'd6:MUX_OUT=G;
3'd7:MUX_OUT=H;
default;; // indicates null
endcase
end
endmodule

```

SIMULATION OUTPUT:**DEMULTIPLEXER:**

LOGIC DIAGRAM:

**RESULT:**

Thus the OUTPUT's of Multiplexers and Demultiplexers are verified by simulating the VHDL and VERILOG code.

EXPERIMENT No: 5**DESIGN OF 4-BIT BINARY TO GRAY CONVERTER****AIM:**

To develop the source code for binary to gray converter by using VERILOG and obtained the simulation.

SOFTWARE & HARDWARE:

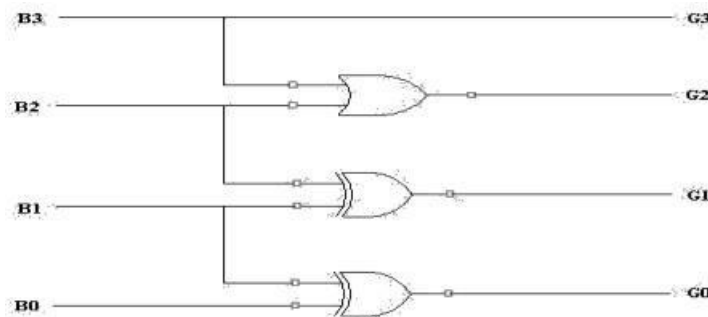
1. XILINX 9.2i
2. FPGA-SPARTAN-3E

CODE CONVERTER (BCD TO GRAY):

TRUTH TABLE:

BCD	GRAY
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101

LOGIC DIAGRAM:

**BEHAVIORAL MODELING:**

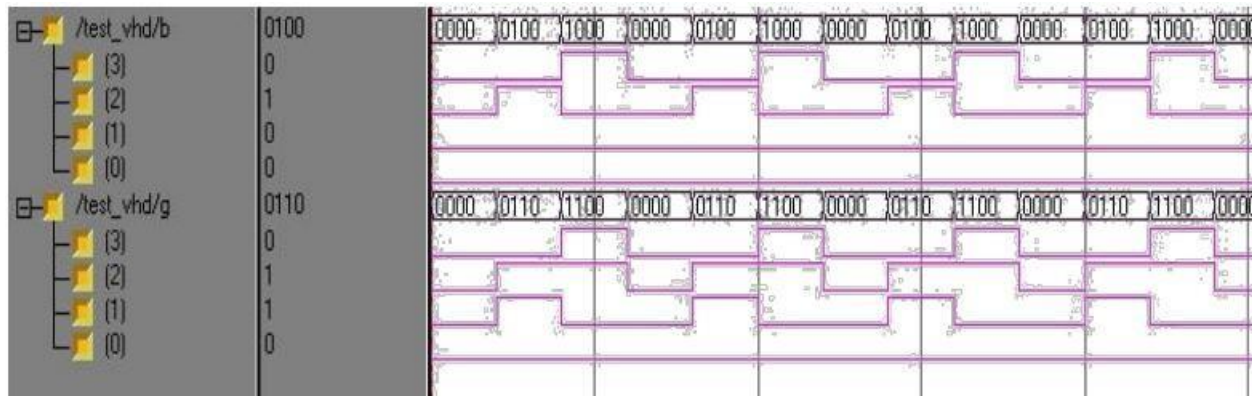
```

module b2g_behv(b, g);
  input [3:0] b;
  output [3:0] g;
  reg [3:0] g;

```

```
always@(b) begin
  g[3]=b[3];
  g[2]=b[3]^b[2];
  g[1]=b[2]^b[1];
  g[0]=b[1]^b[0];
end
endmodule
```

SIMULATION OUTPUT:



RESULT:

Thus the OUTPUT's of binary to gray converter are verified by simulating the VERILOG code.

EXPERIMENT No: 6**4-BIT COMPARATOR****AIM:**

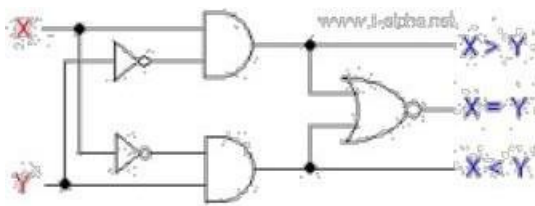
To develop the source code for 4-Bit comparator by using VERILOG and obtained the simulation .

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

4-BIT COMPARATOR:

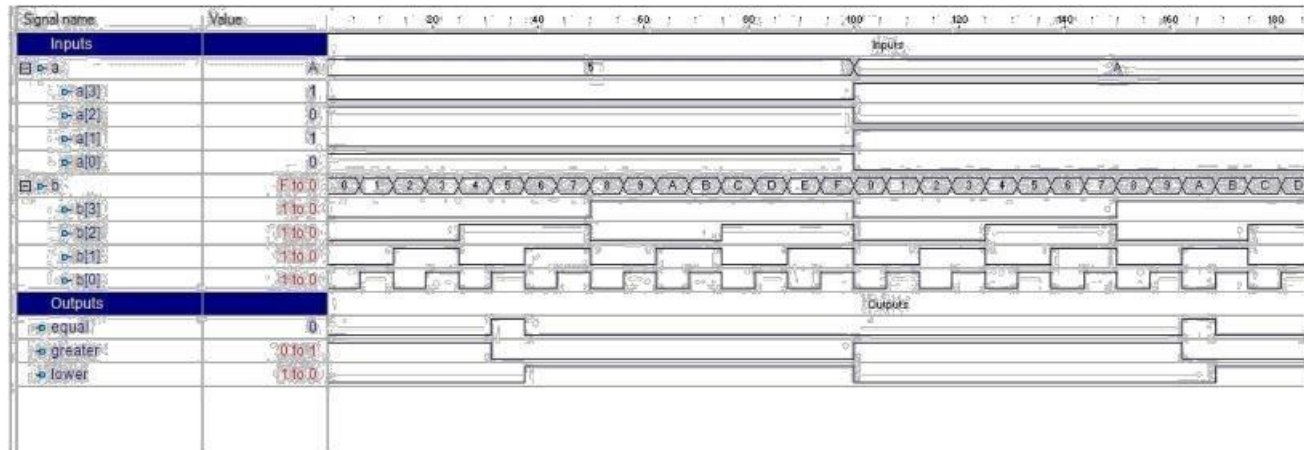
LOGIC DIAGRAM:

**VERILOG SOURCE CODE:**

```

module comparator ( a ,b ,equal ,greater ,lower );
output equal ;
output greater ;
output lower ;
input [3:0] a ;
input [3:0] b ;
always @ (a or b) begin
if (a<b) begin
equal = 0;
lower = 1;
greater = 0;
end else if (a==b) begin
equal = 1;
lower = 0;
greater = 0;
end else begin
equal = 0;
lower = 0;
greater = 1;
end
end
endmodule

```

SIMULATION OUTPUT:**RESULT:**

Thus the OUTPUT's of 4-bit comparator is verified by simulating the VERILOG code.

EXPERIMENT No: 7**DESIGN OF FULL ADDER USING THREE MODELING STYLES****AIM:**

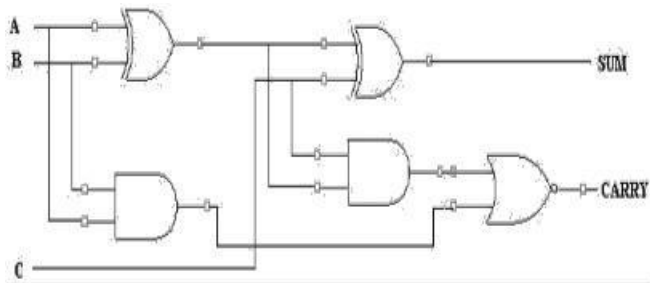
To develop the source code for full adder using three modeling styles by using VERILOG and obtained the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

FULL ADDER:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VERILOG SOURCE CODE:**Dataflow Modeling:**

```

module fulladddataflow(a, b, c, sum, carry);
    input a;
    input b;
    input c;
    output sum;
    output carry;
    assign#2 p=a&b;
    assign#2 q=b&c;
    assign#2 r=c&a;
    assign#4 sum=a^b^c;
    assign#4carry =(p1 | p2) | p3;

endmodule

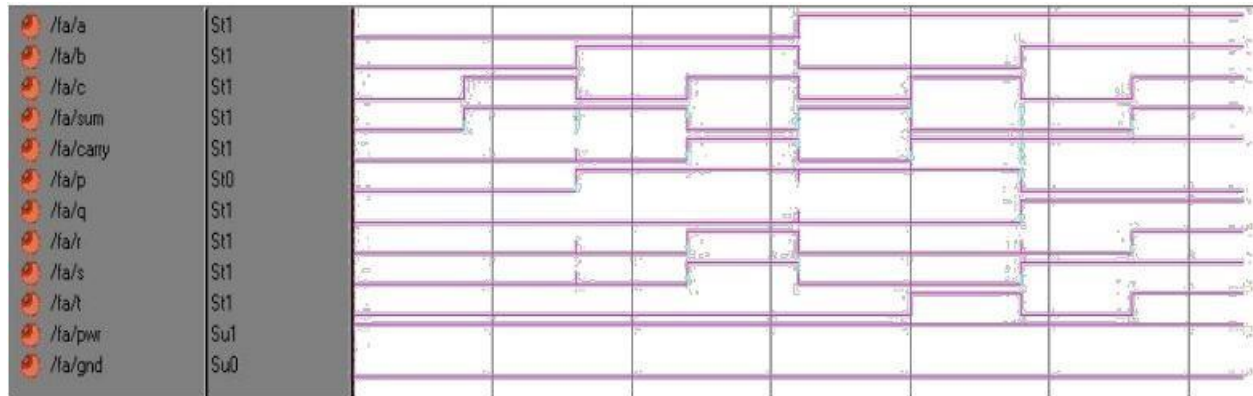
```


Behavioral Modeling:

```
module fuladbehavioral(a, b, c, sum, carry);
    input a;
    input b;
    input c;
    output sum;
    output carry;
    reg sum,carry;
    reg p1,p2,p3;
    always @ (a or b or c) begin
        sum = (a^b)^c;
        p1=a & b;
        p2=b & c;
        p3=a & c;
        carry=(p1 | p2) | p3;
    end
endmodule
```

Structural Modeling:

```
module fa_struct(a, b, c, sum, carry);
    input a;
    input b;
    input c;
    output sum;
    output carry;
    wire t1,t2,t3,s1
    xor
    x1(t1,a,b),
    x2(sum,s1,c);
    and
    a1(t1,a,b),
    a2(t2,b,c),
    a3(t3,a,c);
    or
    o1 (carry, t1, t2, t3);
endmodule
```

SIMULATION OUTPUT:**RESULT:**

Thus the OUTPUT's of full adder using three modeling styles are verified by simulating the VERILOG code.

EXPERIMENT No: 8**DESIGN OF FLIP FLOPS (SR,JK,D,T)****AIM:**

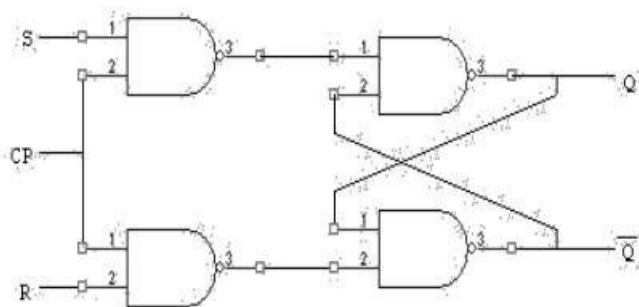
To develop the source code for FLIP FLOPS by using VERILOG and obtained the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

SR FLIPFLOP:

LOGIC DIAGRAM:



TRUTH TABLE:

Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

VERILOG SOURCE CODE:**Behavioral Modeling:**

```

module srflipflop(s, r, clk, rst, q, qbar);
    input s;
    input r;
    input clk;
    input rst;
    output q;
    output qbar;
    reg q,qbar;
    always @ (posedge(clk) or posedge(rst)) begin
        if(rst==1'b1) begin
            q= 1'b0;qbar= 1'b1;
        end
        else if(s==1'b0 &&  r==1'b0)
            begin

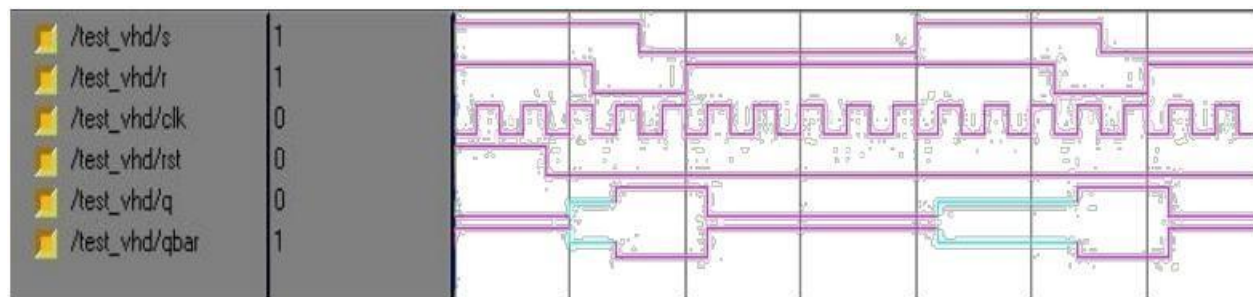
```

```

q=q; qbar=qbar;
end
    else if(s==1'b0 &&    r==1'b1)
        begin
q= 1'b0; qbar= 1'b1;
        end
    else if(s==1'b1 &&    r==1'b0)
        begin
q= 1'b1; qbar= 1'b0;
        end
    else
        begin
q=1'bx;qbar=1'bx;
        end
    end
endmodule

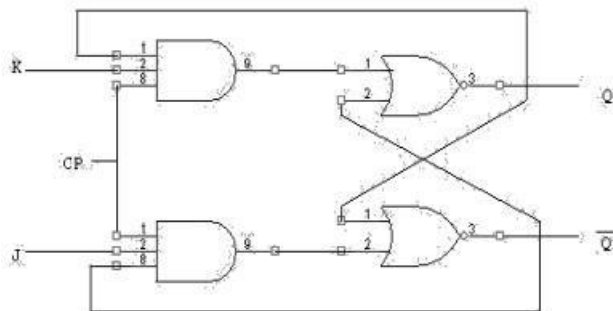
```

SIMULATION OUTPUT:



JK FLIPFLOP:

LOGIC DIAGRAM:

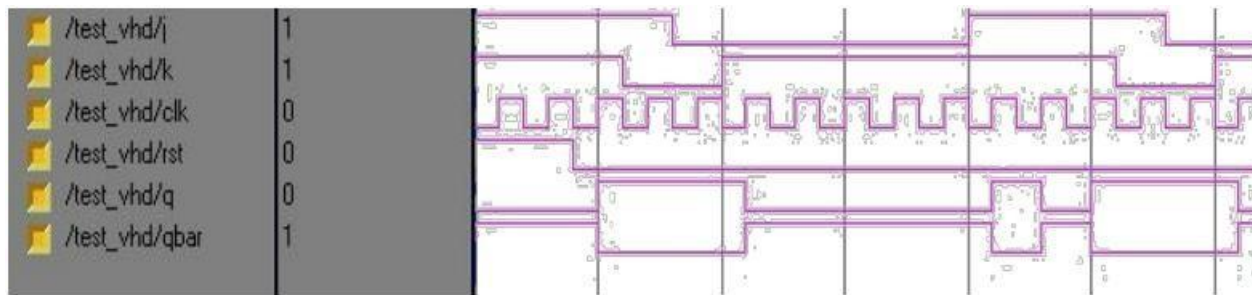


TRUTH TABLE:

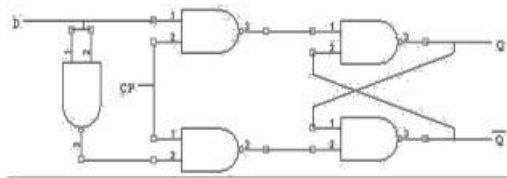
Q(t)	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

VERILOG SOURCE CODE:**Behavioral Modeling:**

```
module jkff(j, k, clk, rst, q, qbar);
  input j;
  input k;
  input clk;
  input rst;
  output q;
  output qbar;
  reg q;
  reg qbar;
  always @ (posedge(clk) or posedge(rst)) begin
    if (rst==1'b1)
      begin
        q=1'b0;
        qbar=1'b1;
      end
    else if (j==1'b0 && k==1'b0)
      begin
        q=q;
        qbar=qbar;
      end
    else if (j==1'b0 && k==1'b1)
      begin
        q=1'b0;
        qbar=1'b1;
      end
    else if (j==1'b1 && k==1'b0)
      begin
        q=1'b1;
        qbar=1'b0;
      end
    else
      begin
        q=~q;
        qbar=~qbar;
      end
  end
endmodule
```

SIMULATION OUTPUT:**D FLIPFLOP:**

LOGIC DIAGRAM:



TRUTH TABLE:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

VERILOG SOURCE CODE:**Behavioral Modeling:**

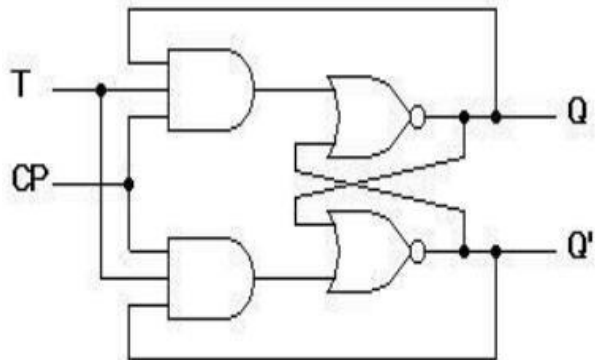
```

module dff(d, clk, rst, q, qbar);
    input d;
    input clk;
    input rst;
    output q;
    output qbar;
    reg q;
    reg qbar;
    always @ (posedge(clk) or posedge(rst)) begin
        if (rst==1'b1)
            begin
                q=1'b0;
                qbar=1'b1;
            end
        else if (d==1'b0)
            begin
                q=1'b0;
                qbar=1'b1;
            end
        else
            begin
                q=1'b1;
                qbar=1'b0;
            end
        end
    end
endmodule

```

SIMULATION OUTPUT:**T-FLIP FLOP**

LOGIC DIAGRAM:

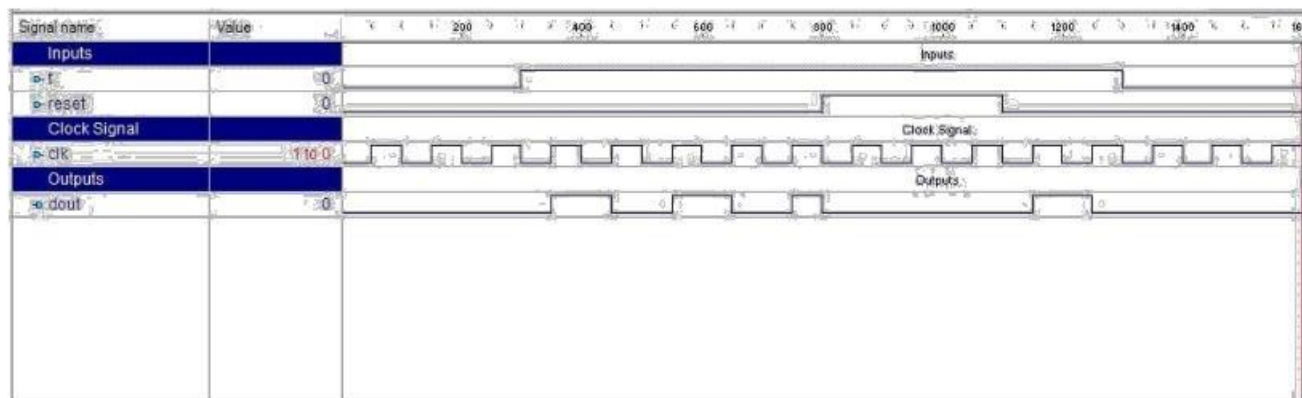


TRUTH TABALE:

clk	D	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	Q	\bar{Q}
1	0	0	1
1	1	1	0

VERILOG SOURCE CODE:

```
module t_flip_flop ( t ,clk ,reset ,dout );  
output dout ;  
input t ;  
input clk ;  
always @ (posedge (clk)) begin  
    if (reset)  
        dout <= 0;  
    else begin  
        if (t)  
            dout <= ~dout;  
    end  
end  
endmodule
```

SIMULATION OUTPUT:**RESULT:**

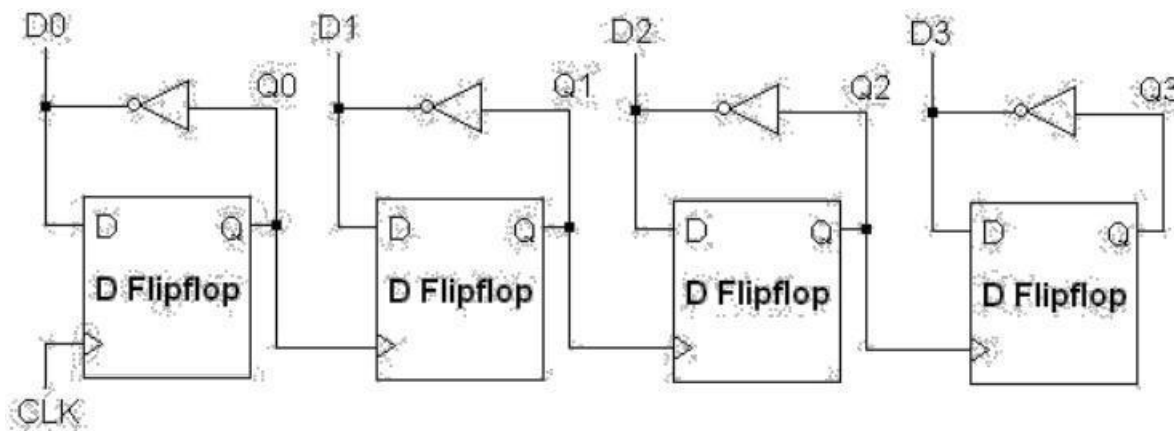
Thus the OUTPUT's of Flip Flops are verified by simulating the VERILOG code.

EXPERIMENT-9**DESIGN OF 4-BIT BINARY COUNTER AND BCD COUNTER****AIM:**

To develop the source code for 4-bit binary counter and BCD counter by using VERILOG and obtained the simulation.

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

LOGIC DIAGRAM:**VERILOG SOURCE CODE:**

```
module Counter_4Bit ( clk ,reset ,dout );
```

```
output [3:0] dout ;
```

```
input clk ;
```

```
input reset ;
```

```
initial dout = 0;
```

```
always @ (posedge (clk)) begin
```

```
if (reset)
```

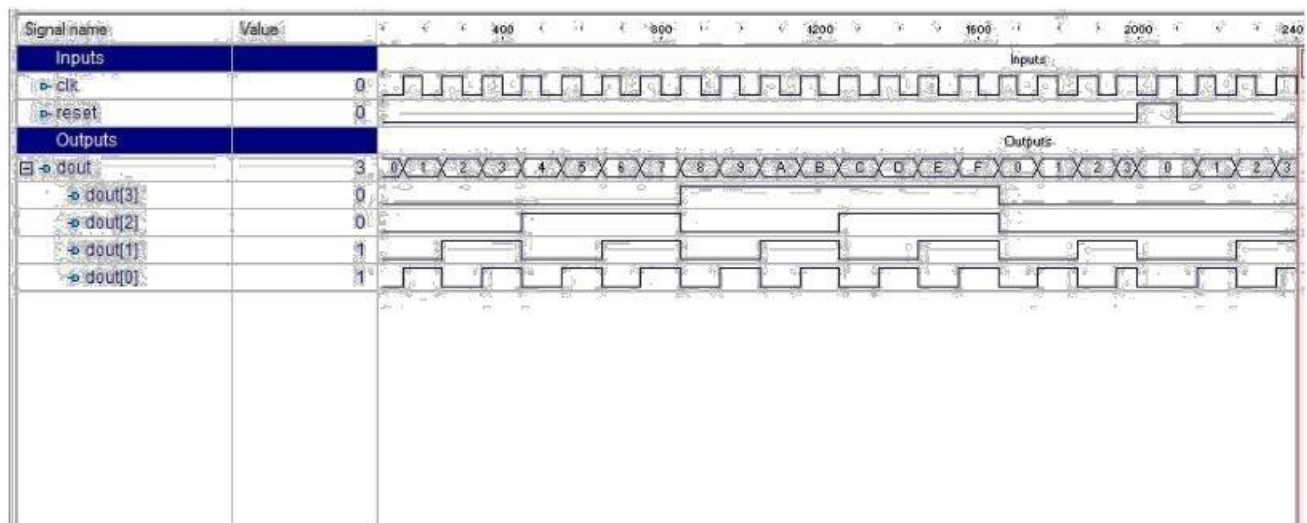
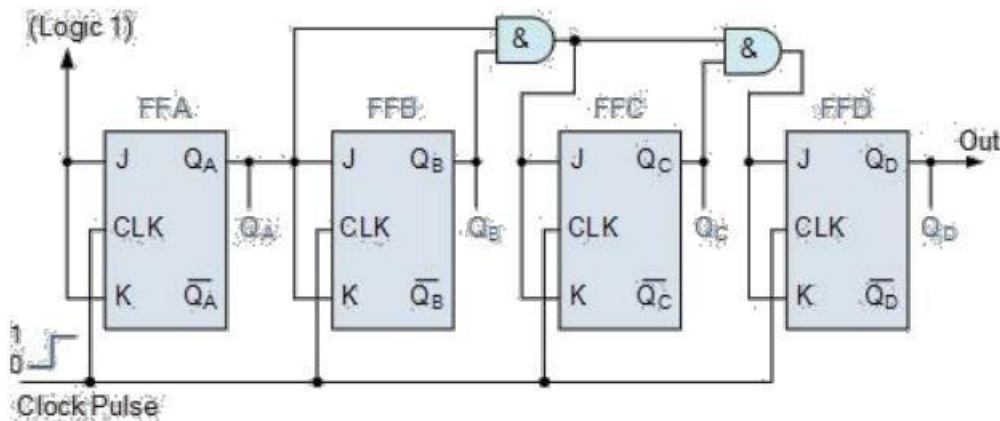
```
    dout <= 0;
```

```
else
```

```
    dout <= dout + 1;
```

```
end
```

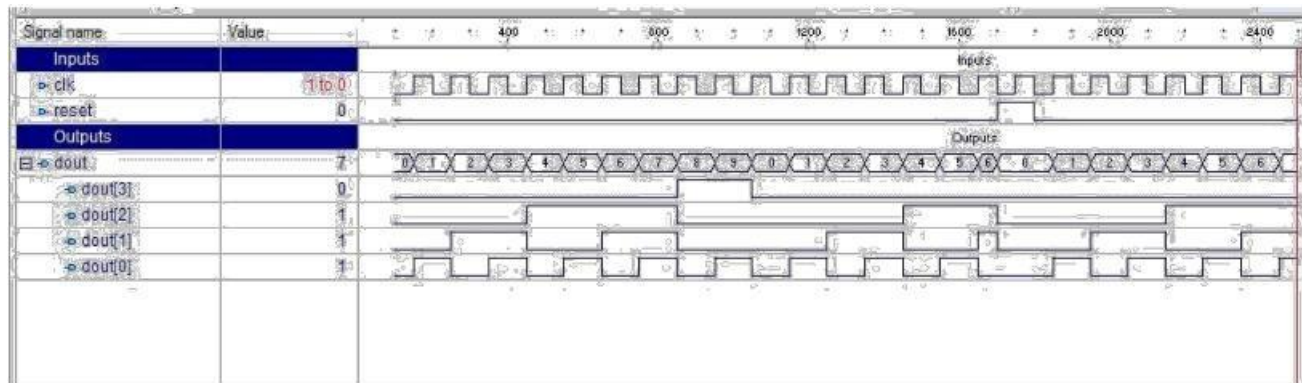
```
endmodule
```

SIMULATION OUTPUT:**BCD COUNTER****LOGIC DIAGRAM****VERILOG SOURCE CODE**

```

module BCD_Counter ( clk ,reset ,dout );
output [3:0] dout ;;
input clk ;
input reset ;
initial dout = 0 ;
always @ (posedge (clk)) begin
if (reset)
dout <= 0;
else if (dout<=9) begin
dout <= dout + 1;
end else if (dout==9) begin
dout <= 0;
end
end
endmodule

```

SIMULATION OUTPUT:**RESULT:**

Thus the OUTPUT's of 4-bit counter and BCD COUNTER using three modeling styles are verified by synthesizing and simulating the VERILOG code

EXPERIMENT: 10**FINITE STATE MACHINE DESIGN****AIM:**

To develop the source code for finite state machine design by using VERILOG and obtained the simulation

SOFTWARE & HARDWARE:

1. XILINX 9.2i
2. FPGA-SPARTAN-3E

FSM DESIGN**VERILOG SOURCE CODE:**

```

module fsm_using_function (
    clock      , // clock
    reset      , // Active high, syn reset
    req_0      , // Request 0
    req_1      , // Request 1
    gnt_0      , // Grant 0
    gnt_1
);
//-----Input Ports-----
input  clock,reset,req_0,req_1;
//-----Output Ports-----
output gnt_0,gnt_1;
//-----Input ports Data Type-----
//-----Output Ports Data Type-----
reg      gnt_0,gnt_1;
//-----Internal Constants-----
parameter SIZE = 3 ;
parameter IDLE = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100 ; //-----
//-----Internal Variables-----
reg [SIZE-1:0] state ;// Seq part of the FSM
//-----Code startes Here-----
assign next_state = fsm_function(state, req_0, req_1);
//-----Function for Combo Logic-----
function [SIZE-1:0] fsm_function;
    input [SIZE-1:0] state ;
    input  req_0 ;
    input  req_1 ;
    case(state)
        IDLE : if (req_0 == 1'b1) begin
                    fsm_function = GNT0;
                end else if (req_1 == 1'b1) begin
                    fsm_function= GNT1;
                end else begin
                    fsm_function = IDLE;
                end
        GNT0 : if (req_0 == 1'b1) begin
                    fsm_function = GNT0;
                end
    endcase
endfunction

```

```

        end else begin
            fsm_function = IDLE;
        end
    GNT1 : if (req_1 == 1'b1) begin
        fsm_function = GNT1;
    end else begin
        fsm_function = IDLE;
    end
    default : fsm_function = IDLE;
endcase
endfunction
//-----Seq Logic-----
always @ (posedge clock)
begin : FSM_SEQ
    if (reset == 1'b1) begin
        state <= #1 IDLE;
    end else begin
        state <= #1 next_state;
    end
end
//-----Output Logic-----
always @ (posedge clock)
begin : OUTPUT_LOGIC
    if (reset == 1'b1) begin
        gnt_0 <= #1 1'b0;
        gnt_1 <= #1 1'b0;
    end
    else begin
        case(state)
            IDLE : begin
                gnt_0 <= # 1'b0;
                gnt_1 <= 1 1'b0;
                #
                1
            end
            GNT0 : begin
                gnt_0 <= #1 1'b1;
                gnt_1 <= #1 1'b0;
            end
            GNT1 : begin
                gnt_0 <= #1 1'b0;
                gnt_1 <= #1 1'b1;
            end
            default : begin
                gnt_0 <= #1 1'b0;
                gnt_1 <= #1 1'b0;
            end
        endcase
    end
end // End Of Block

OUTPUT_LOGIC endmodule // End

of Module arbiter

```

RESULT:

Thus the OUTPUT's of finite state machine design is verified by simulating the VERILOG code.

EXPERIMENT: 11

DESIGN AND IMPLEMENTATION OF AN INVERTER

AIM: To design and Implementation of an Inverter

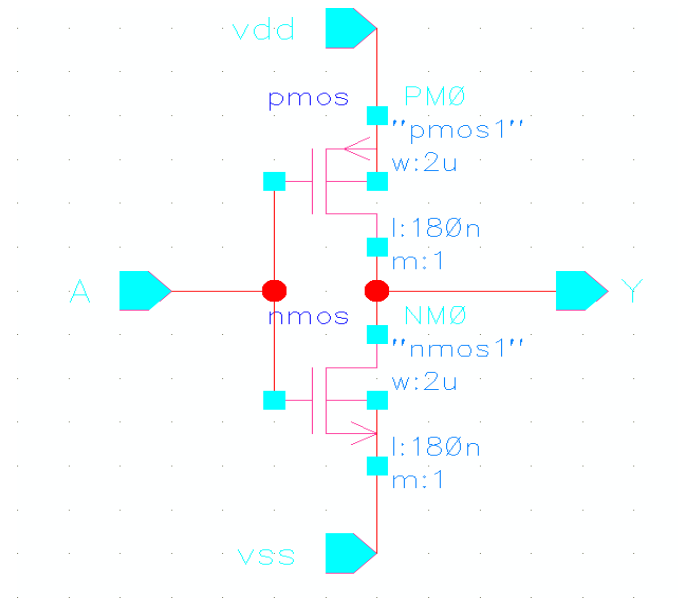
TOOLS: Mentor Graphics: Pyxis Schematic, Pyxis Layout, Eldo, Ezwave, Calibre

THEORY:

The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom. CMOS is also sometimes referred to as **complementary-symmetry metal-oxide-semiconductor**. The words "complementary- symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.

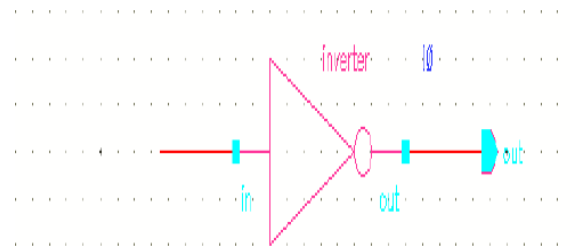
Schematic Capture:

Procedure:

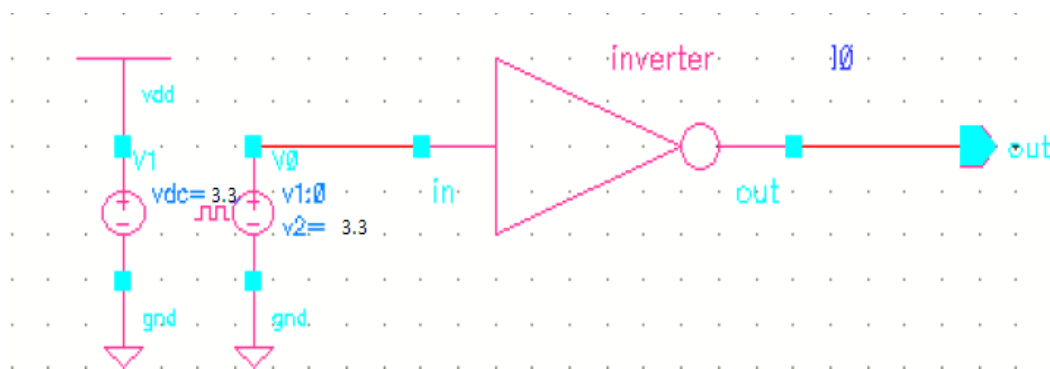


1. Connect the Circuit as shown in the circuit diagram using Pyxis Schematic tool
2. Enter into Simulation mode.
3. Setup the Analysis and library.
4. Setup the required analysis.
5. Probe the required Voltages
6. Run the simulation.
7. Observe the waveforms in EZ wave.
8. Draw the layout using Pysis Layout.
9. Perform Routing using IRoute
10. Perform DRC, LVS, PEX.

Schematic Symbol:

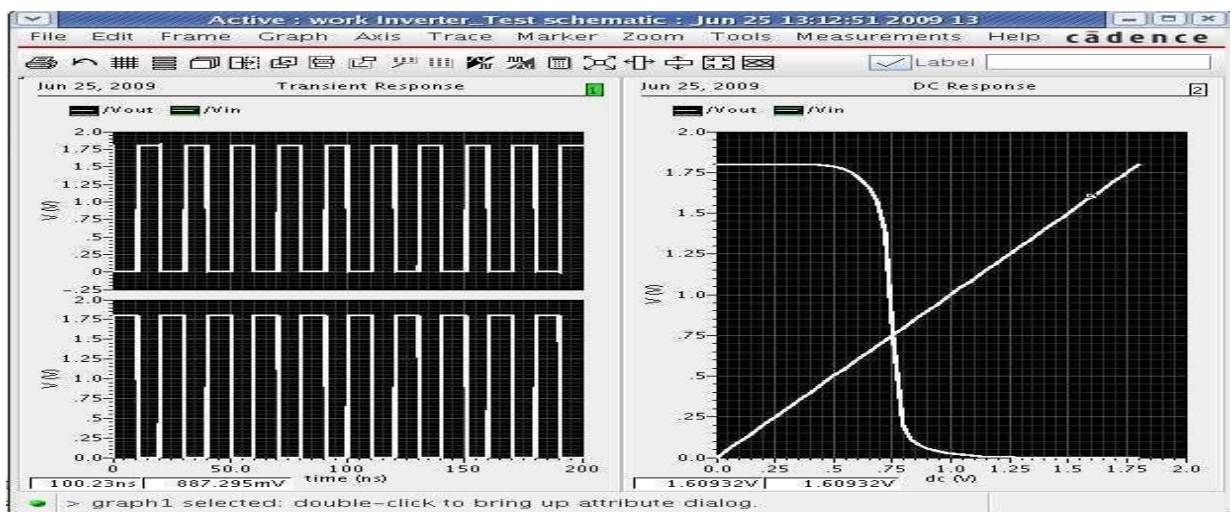


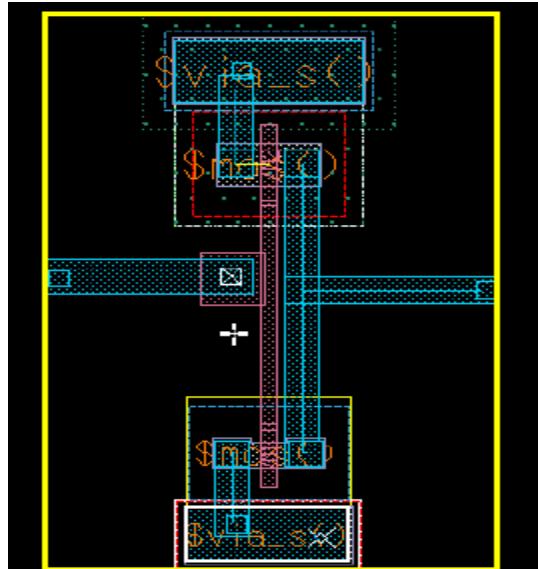
Testing the Schematic:



Simulation Output: Input Vs Output

Transient and DC Characteristics:



Layout of the Inverter:

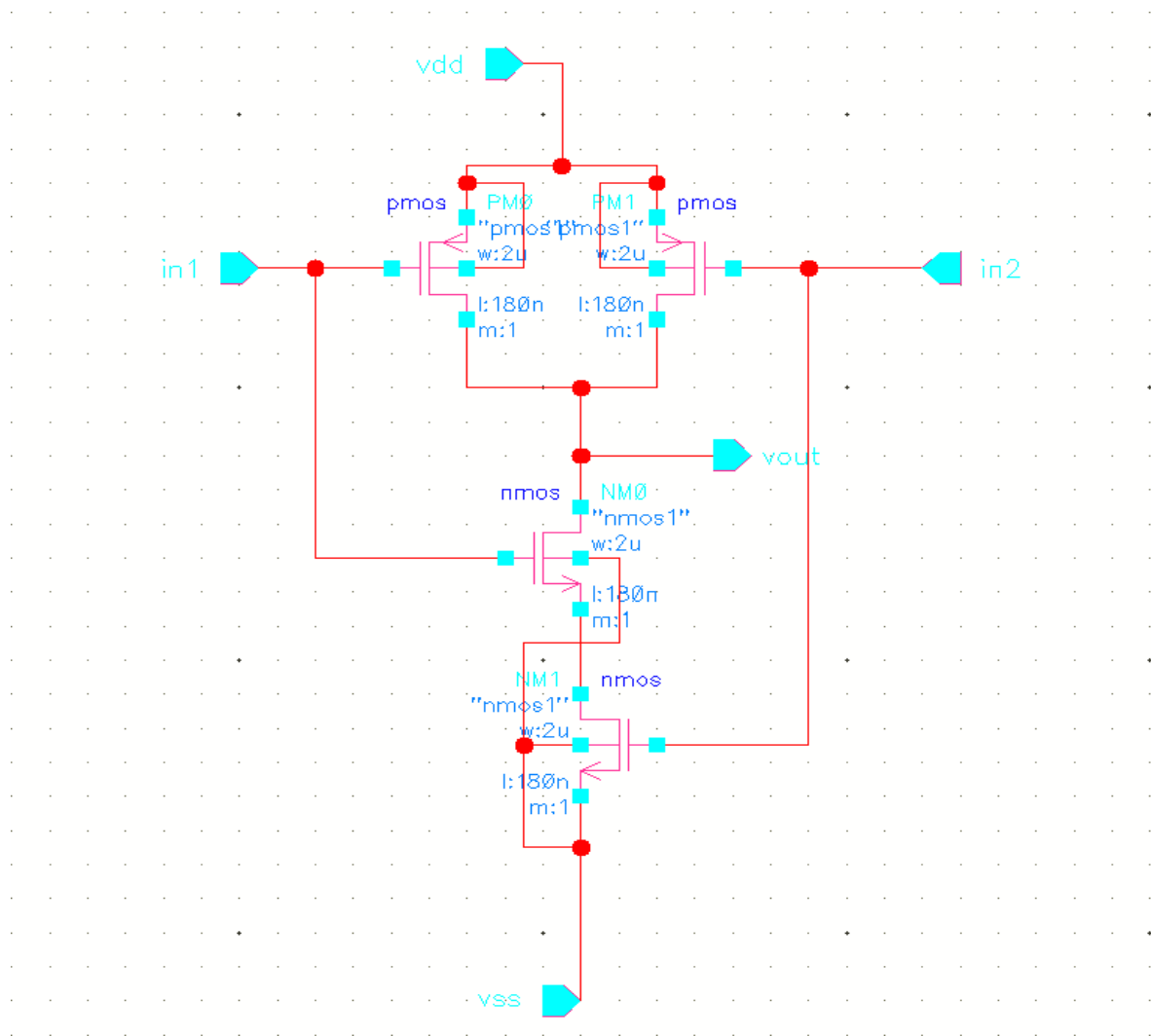
EXPERIMENT: 12

NAND GATE

AIM: To create a library and build a schematic of a NAND GATE, to create a symbol for the Inverter, to build an Inverter Test circuit using your Inverter, To set up and run simulations on the Inverter Test design.

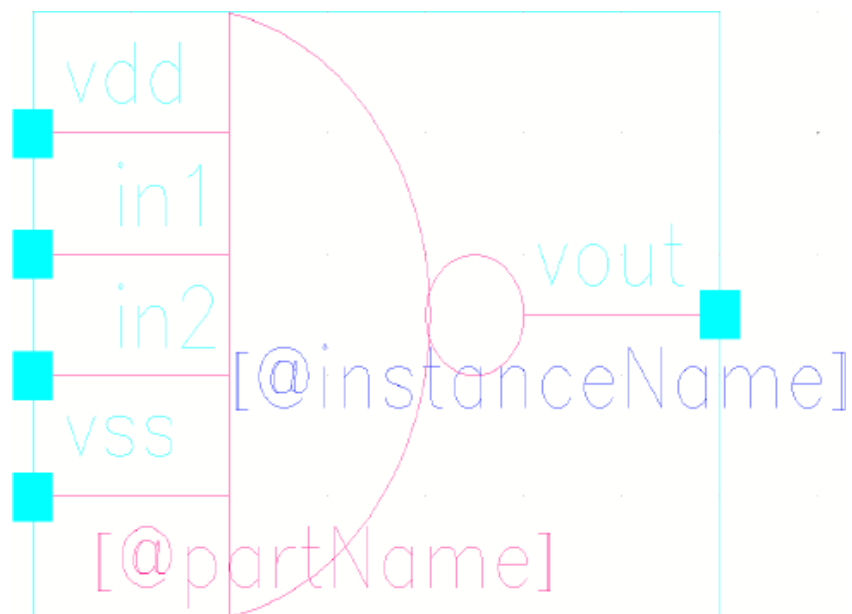
EDA Tool: Mentor Graphics

Schematic Diagram

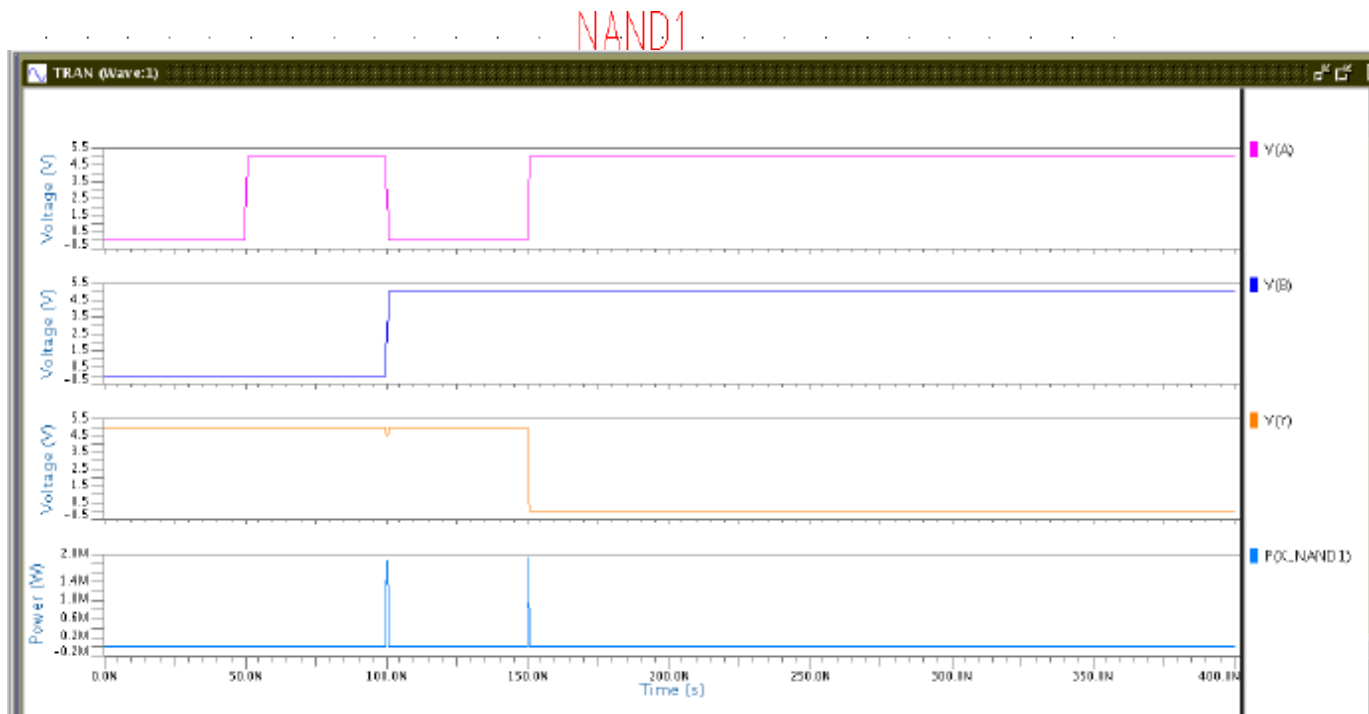


PROCEDURE:

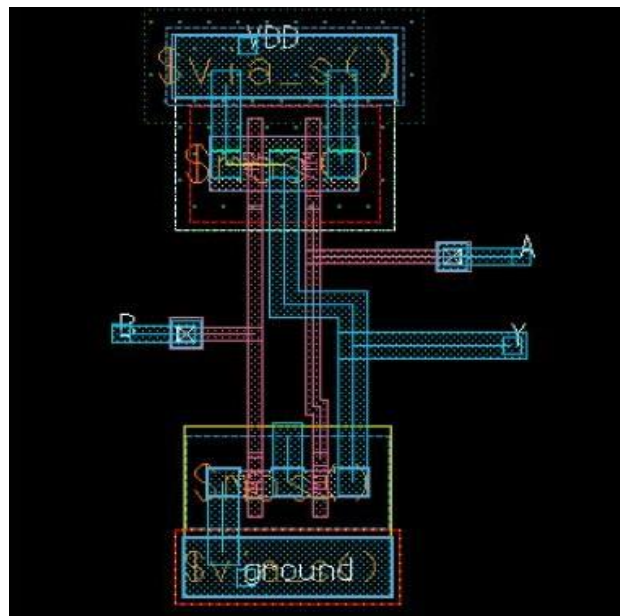
1. Connect the Circuit as shown in the circuit diagram using Pyxis schematic.
2. Create a simulation schematic for simulation.
3. Add necessary nets in outputs to view waveforms.
4. Run the Simulation and observe results in EZwave.
5. Draw the Layout for the circuit using Pyxis Layout.
7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .
8. Run the post layout simulation by adding the .dspf file generated in PEX.
9. Observe the post layout results.

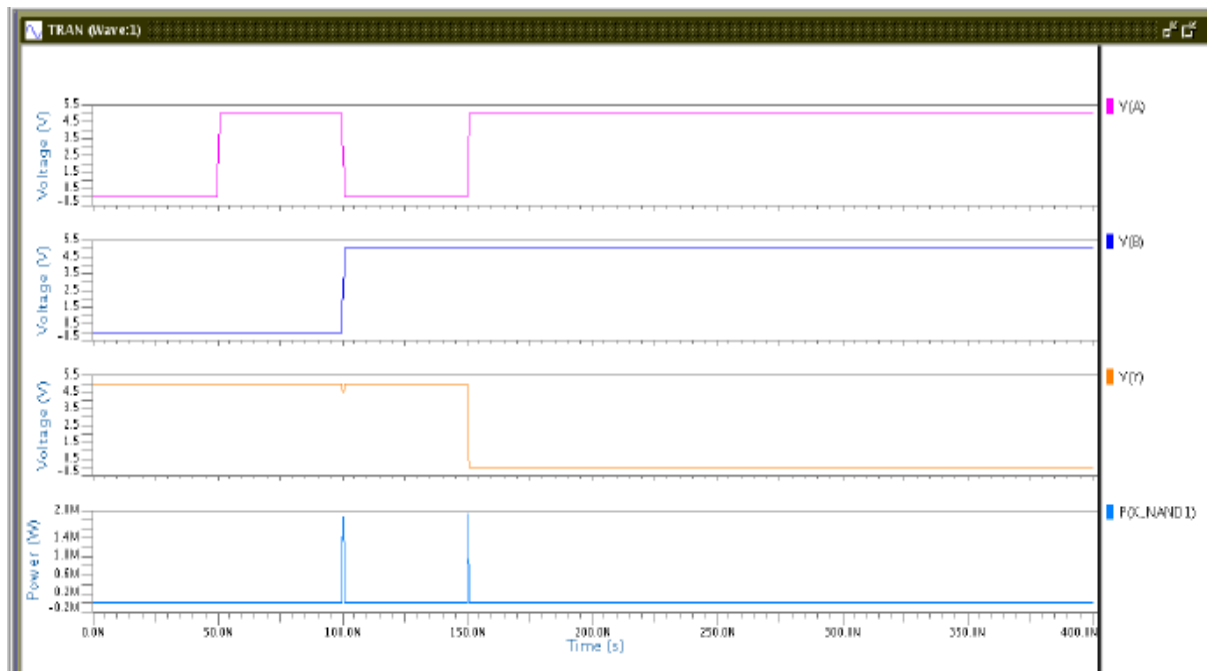
Symbol Creation:

Building the NAND Test Design



Creating a layout view of NAND gate

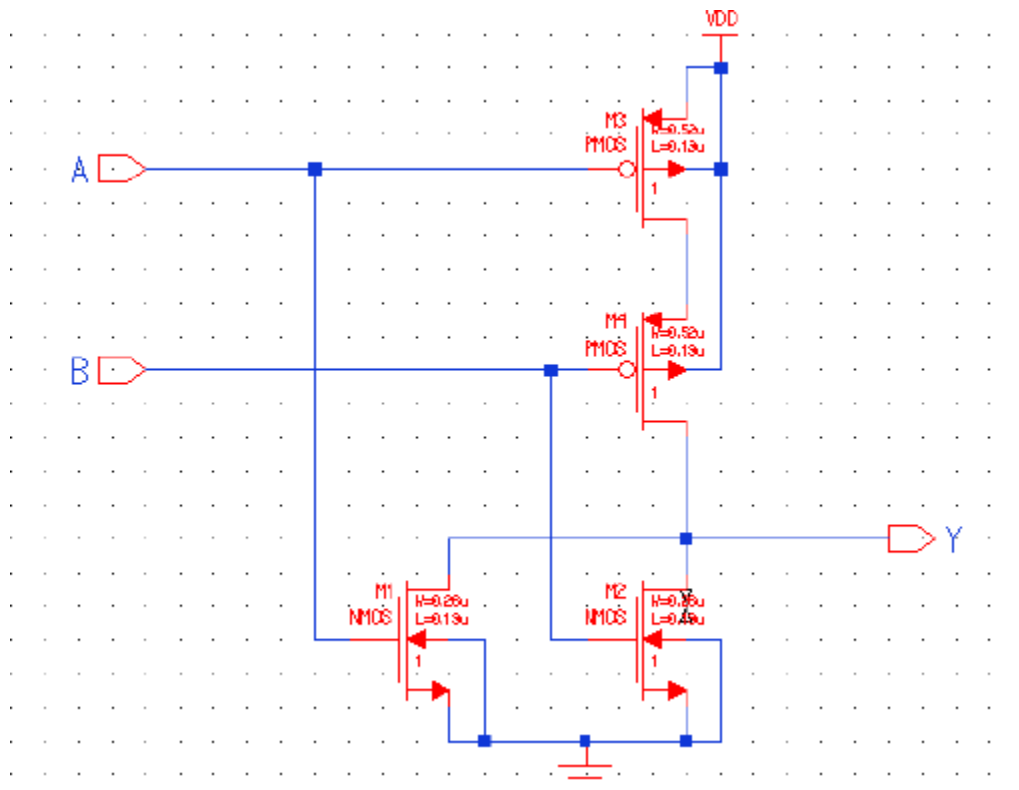


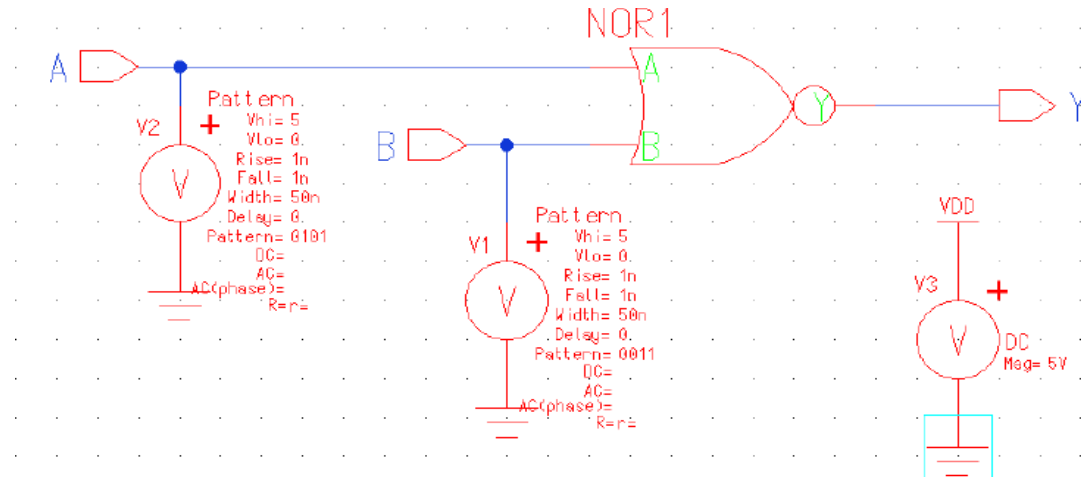
Simulation Output:

EXPERIMENT NO: 13**NOR GATE**

AIM: To design and simulate the CMOS NOR gate

TOOLS: Mentor Graphics: Pyxis Schematic, Pyxis Layout, Eldo, Ezwave, Calibre

CIRCUIT DIAGRAM:

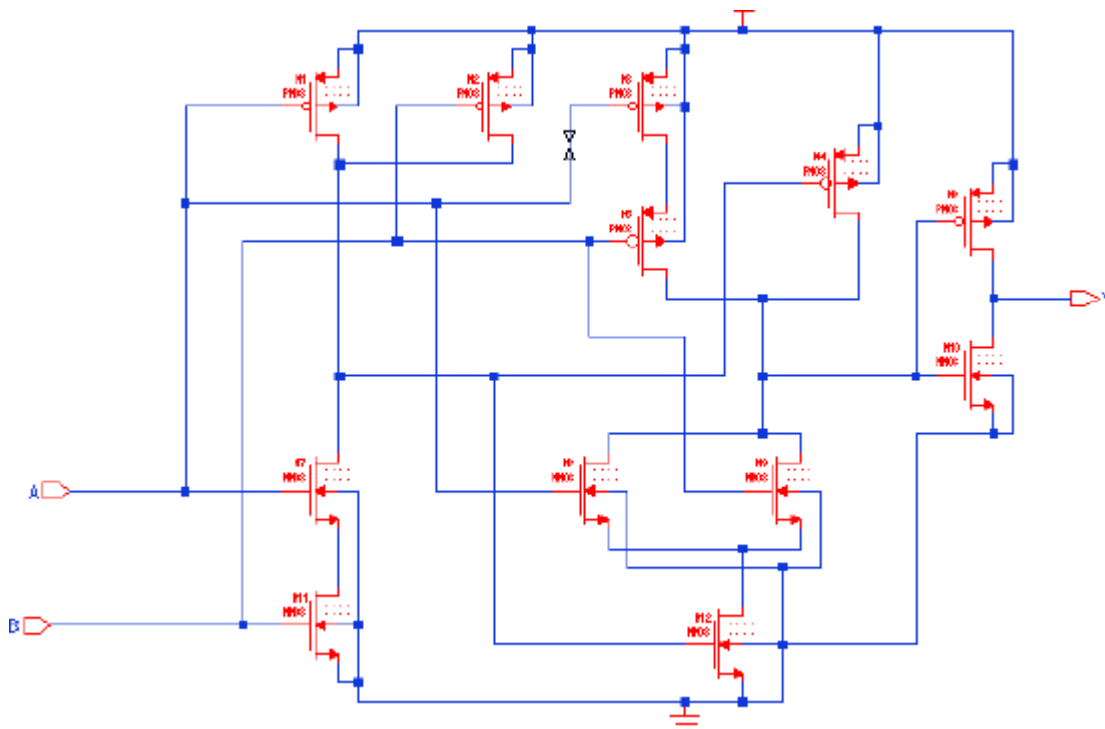
SIMULATION CIRCUIT:**PROCEDURE:**

1. Connect the Circuit as shown in the circuit diagram using Pyxis schematic.
2. Create a simulation schematic for simulation.
3. Add necessary nets in outputs to view waveforms.
4. Run the Simulation and observe results in EZwave.
5. Draw the Layout for the circuit using Pyxis Layout.
7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .
8. Run the post layout simulation by adding the .dspf file generated in PEX.
9. Observe the post layout results.

EXPERIMENT NO: 14**XOR GATE**

AIM: To create a library and build a schematic of an XOR gate, to create a symbol for the XOR, to build an inverter test circuit using your XOR, to set up and run simulations on the XOR_test design.

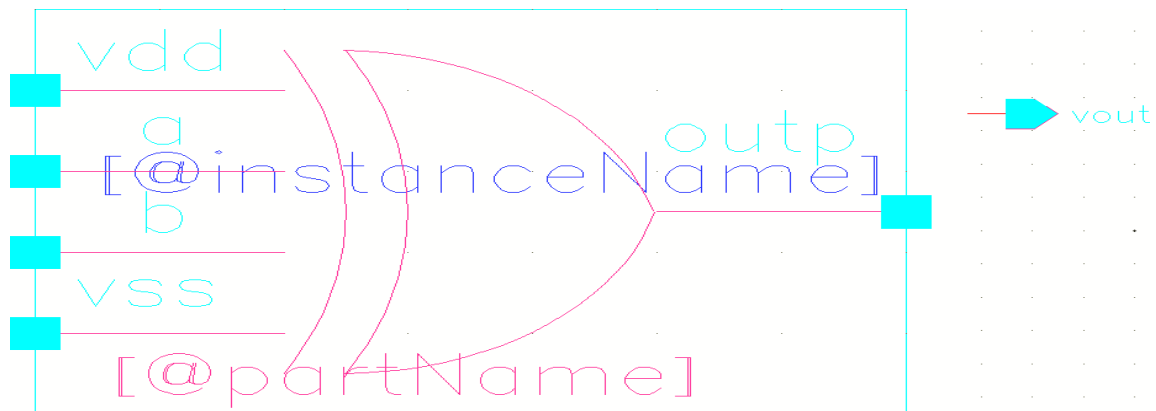
EDA TOOLS: pyxis schematic, pyxis layout, eldo, ezwave, calibre

**PROCEDURE:**

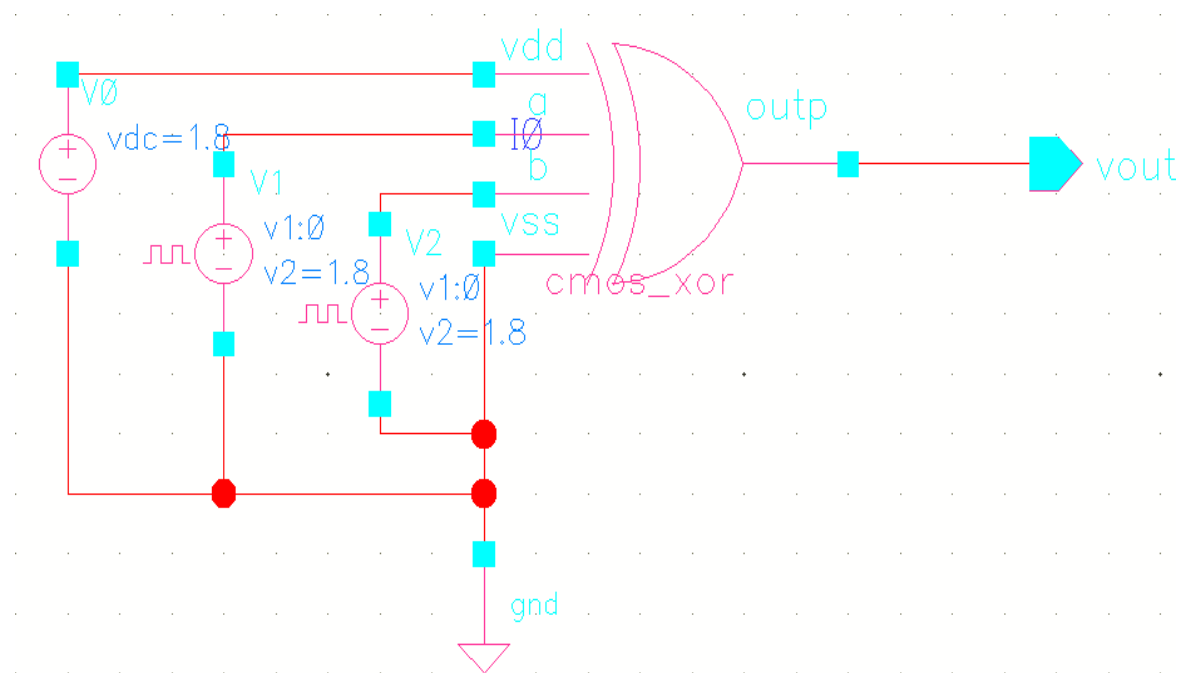
1. Connect the Circuit as shown in the circuit diagram using Pyxis schematic.
2. Create a simulation schematic for simulation.
3. Add necessary nets in outputs to view waveforms.
4. Run the Simulation and observe results in EZwave.
5. Draw the Layout for the circuit using Pyxis Layout.

7. Run the physical verification (DRC, LVS, PEX) using Calibre tool.
8. Run the post layout simulation by adding the .dspf file generated in PEX.
9. Observe the post layout results.

Symbol Creation:

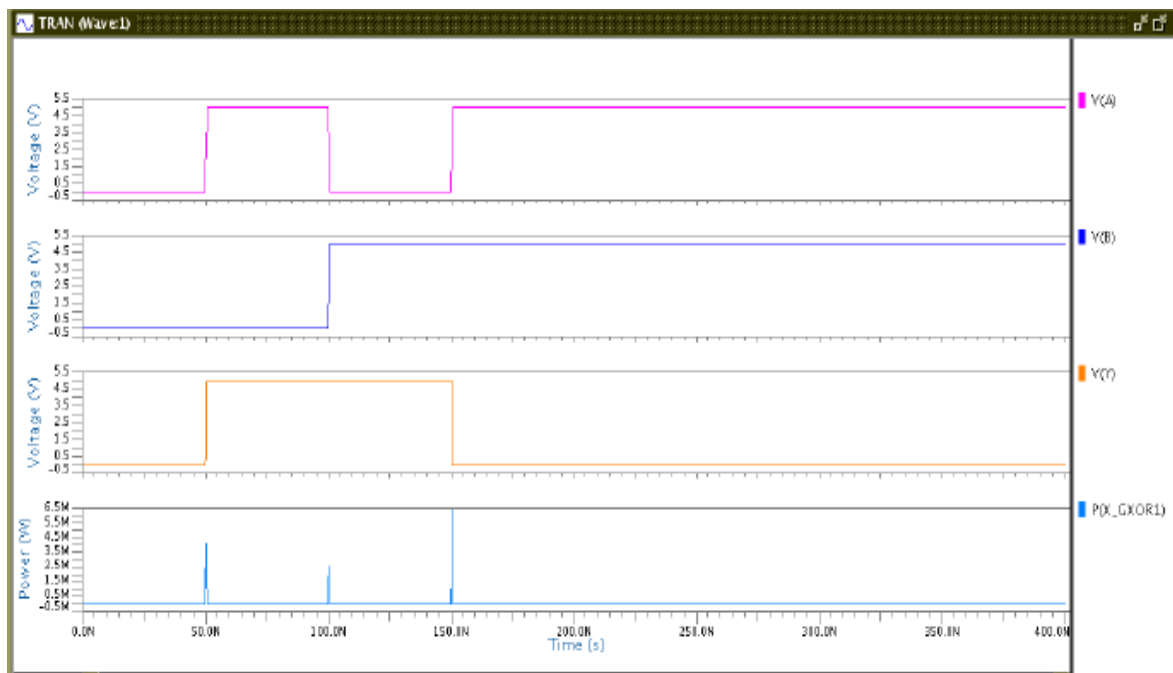


Building the XOR Gate Test Design



PROCEDURE:

1. Connect the Circuit as shown in the circuit diagram using Pyxis schematic.
2. Create a simulation schematic for simulation.
3. Add necessary nets in outputs to view waveforms.
4. Run the Simulation and observe results in EZwave.
5. Draw the Layout for the circuit using Pyxis Layout.
7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .
8. Run the post layout simulation by adding the .dspf file generated in PEX.
9. Observe the post layout results

Simulation output:**Layout:**

