# VLSI DESIGN(R22A0491)

# LABORATORY MANUAL

**B.TECH**
**(III YEAR – II SEM)**
**2024-2025**

**Prepared by:**
**Dr.M.Arun Kumar, Associate Professor**
**Mr.K.Suresh, Associate Professor**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**

**(Autonomous Institution – UGC, Govt. of India)**

Recognized under 2(f) and 12 (B) of UGC ACT 1956
Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified
Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India.

# ELECTRONICS & COMMUNICATION ENGINEERING

## VISION

To evolve into a center of excellence in Engineering Technology through creative and innovative practices in teaching-learning, promoting academic achievement & research excellence to produce internationally accepted competitive and world class professionals.

## MISSION

To provide high quality academic programmes, training activities, research facilities and opportunities supported by continuous industry institute interaction aimed at employability, entrepreneurship, leadership and research aptitude among students.

## QUALITY POLICY

- Impart up-to-date knowledge to the students in Electronics & Communicationarea to make them quality engineers.
- Make the students experience the applications on quality equipment and tools.
- Provide systems, resources and training opportunities to achieve continuousimprovement.
- Maintain global standards in education, training and services.

## PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

**PEO1: PROFESSIONALISM & CITIZENSHIP**

To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.

**PEO2: TECHNICAL ACCOMPLISHMENTS**

To provide knowledge based services to satisfy the needs of society and theindustry by providing hands on experience in various technologies in core field.

**PEO3: INVENTION, INNOVATION AND CREATIVITY**

To make the students to design, experiment, analyze, interpret in the corefield with the help of other multi disciplinary concepts wherever applicable.

**PEO4: PROFESSIONAL DEVELOPMENT**

To educate the students to disseminate research findings with good soft skillsand become a successful entrepreneur.

**PEO5: HUMAN RESOURCE DEVELOPMENT**

To graduate the students in building national capabilities in technology, education and research.

## PROGRAMME SPECIFIC OBJECTIVES (PSOs)

**PSO1**

To develop a student community who acquire knowledge by ethical learning and fulfill the societal and industry needs in various technologies of core field.

**PSO2**

To nurture the students in designing, analyzing and interpreting required in research and development with exposure in multi disciplinary technologies in order to mould them as successful industry ready engineers/entrepreneurs

**PSO3**

To empower students with all round capabilities who will be useful in making nation strong in technology, education and research domains.

## PROGRAM OUTCOMES (POs)

**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 multi disciplinary 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.

# LABORATORY RULES

## General Rules of Conduct in Laboratories:

1. You are expected to arrive on time and not depart before the end of a laboratory.
2. You must not enter a lab unless you have permission from a technician or lecturer.
3. You are expected to comply with instructions, written or oral, that the laboratory Instructor gives you during the laboratory session.
4. You should behave in an orderly fashion always in the lab.
5. You must not stand on the stools or benches in the laboratory.
6. Keep the workbench tidy and do not place coats and bags on the benches.
7. You must ensure that at the end of the laboratory session all equipment used is stored away where you found it.
8. You must put all rubbish such as paper outside in the corridor bins. Broken components should be returned to the lab technician for safe disposal.
9. You must not remove test equipment, test leads or power cables from any lab without permission.
10. Eating, smoking and drinking in the laboratories are forbidden.
11. The use of mobile phones during laboratory sessions is forbidden.
12. The use of email or messaging software for personal communications during laboratory sessions is forbidden.
13. Playing computer games in laboratories is forbidden.

**Specific Safety Rules for Laboratories:**

1. You must not damage or tamper with the equipment or leads.
2. You should inspect laboratory equipment for visible damage before using it. If there is a problem with a piece of equipment, report it to the technician or lecturer. DONOT return equipment to a storage area.
3. You should not work on circuits where the supply voltage exceeds 40 volts without very specific approval from your lab supervisor. If you need to work on such circuits, you should contact your supervisor for approval and instruction on how to do this safely before commencing the work.
4. Always use an appropriate stand for holding your soldering iron.
5. Turn off your soldering iron if it is unlikely to be used for more than 10 minutes.
6. Never leave a hot soldering iron unattended.
7. Never touch a soldering iron element or bit unless the iron has been disconnected from the mains and has had adequate time to cool down.
8. Never strip insulation from a wire with your teeth or a knife, always use an appropriate wire stripping tool.
9. Shield wire with your hands when cutting it with a pliers to prevent bits of wire flying about the bench.

# CYCLE-I


| S.NO. | EXPERIMENT NAME | PAGE NO. |
|---|---|---|
| 1 | HDL code to realize all the logic gates | 1 |
| 2 | Design of Full adder using 3 modeling styles | 8 |
| 3 | Design Of 2-To-4 Decoder | 12 |
| 4 | Design Of 8-To-3 Encoder (without and with parity) | 14 |
| 5 | Design Of 4 Bit Binary To Gray Converter | 15 |
| 6 | Design Of Flip Flops: SR, D, JK | 17 |
| 7 | Design of Multiplexer | 22 |
| 8 | Ripple Counters Realization-(Mod -10 & Mod-12) | 26 |
| 9 | Design of Sequence Detector (Finite State Machine- Mealy and Moore Machines) | 36 |
| 10 | Design of ALU to Perform – ADD, SUB, AND-OR, 1’s and 2’s Compliment, Multiplication, and Division. | |


# CYCLE-II


| | | |
|---|---|---|
| 1 | CMOS INVERTER | 40 |
| 2 | NAND Gate | 44 |
| 3 | NOR Gate | 48 |
| 4 | XOR Gate | 51 |
| 5 | CMOS 1-Bit Full Adder | 55 |
| 6 | Common Source Amplifier | 58 |
| 7 | Differential Amplifier | 62 |

# EXPERIMENT: 1

## HDL CODE TO REALIZE ALL LOGIC GATES

**AIM:** To develop the source code for logic gates by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

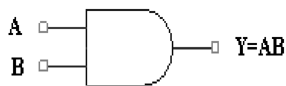
**SOFTWARE & HARDWARE:**

XILINX-VIVADO

**LOGIC DIAGRAM:**

**AND GATE:**

LOGIC DIAGRAM:

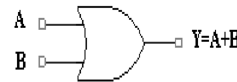



TRUTH TABLE:

| A | B | Y=AB |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**OR GATE:**

LOGICDIAGRAM

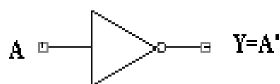



TRUTH TABLE:

| A | B | Y=A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**NOT GATE:**

LOGIC DIAGRAM:

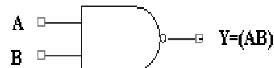



TRUTH TABLE:

| A | Y=A’ |
|---|---|
| 0 | 0 |
| 0 | 1 |

**NAND GATE:**

LOGICDIAGRAM

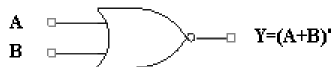



TRUTH TABLE

| A | B | Y=(AB)’ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**NOR GATE:**

LOGIC DIAGRAM:

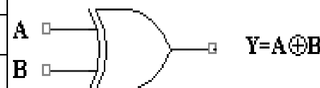



TRUTH TABLE:

| A | B | Y=(A+B)’ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**XOR GATE:**

LOGICDIAGRAM

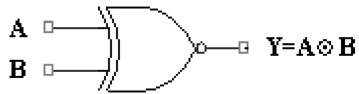



TRUTH TABLE:

| A | B | Y=A⊕B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**XNOR GATE:**

LOGIC DIAGRAM:



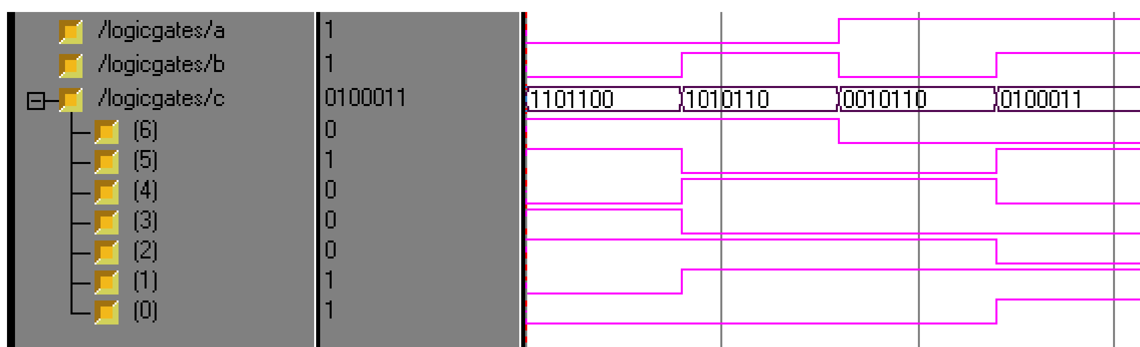

TRUTH TABLE:

| A | B | Y=A⊙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 synthesizing and simulating the VERILOG code.

# EXPERIMENT: 2

# FULL ADDER

**AIM:** To develop the source code for Full adder by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**SOFTWARE REQUIRED:**

XILINX-VIVADO

**THEORY:**

A combinational circuit that performs the addition of three bits is called a half-adder. This circuit needs three binary inputs and produces two binary outputs. One of the input variables designates the augends and other designates the addend. Mostly, the third input represents the carry from the previous lower significant position. The output variables produce the sum and the carry.

The simplified Boolean functions of the two outputs can be obtained

as below: Sum $S = x \oplus y \oplus z$

Carry $C = xy + xz + yz$

Where x, y & z are the two input variables.

**PROGRAM:**

```
//Gate-level description of Full Adder using two Half
Adder //Description of Half Adder
module halfadder(s,co,x,y);
 input x,y;
output s,co;
//Instatiate primitive gates
 xor (s,x,y);
and (co,x,y);
endmodule

//Description of Full Adder
module
fulladder(s,co,x,y,ci);
input x,y,ci;
output s,co;
wire s1,d1,d2;  //Outputs of first XOR and AND gates

//Instantiate Half Adder

 halfadder ha_1(s1,d1,x,y);
```

```
halfadder ha_2(s,d2,s1,ci);
or or_gate(co,d2,d1);
endmodule

//Stimulus for testing Full Adder module
simulation;
reg x,y,ci;
wire s,co;

//Instantiate Full Adder fulladder fa_test(s,co,x,y,ci);
 Initial
begin
x=1'b0; y=1'b0; ci=1'b0;
#100 x=1'b0; y=1'b0; ci=1'b1; #100 x=1'b0; y=1'b1; ci=1'b0; #100 x=1'b0;
y=1'b1; ci=1'b1; #100 x=1'b1; y=1'b0; ci=1'b0; #100 x=1'b1; y=1'b0;
ci=1'b1; #100 x=1'b1; y=1'b1; ci=1'b0; #100 x=1'b1; y=1'b1; ci=1'b1;
end
endmodule
```

**Behavioral Modeling:**

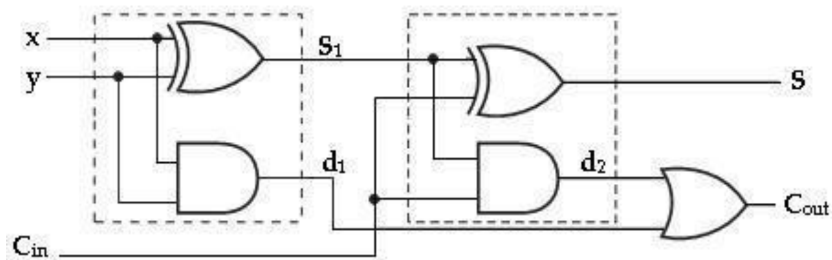
```
module fuladbehavioral(x, y, z, sum, carry);
   input x;
   input  y;
   input z;
   output sum;
   output carry;
          reg sum,carry;
          reg p1,p2,p3;
          always @ (x or y or z) begin
          sum = (x^y)^z;
          p1=x & y;
          p2=y & z;
          p3=x & z;
         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

**Dataflow Modeling:**

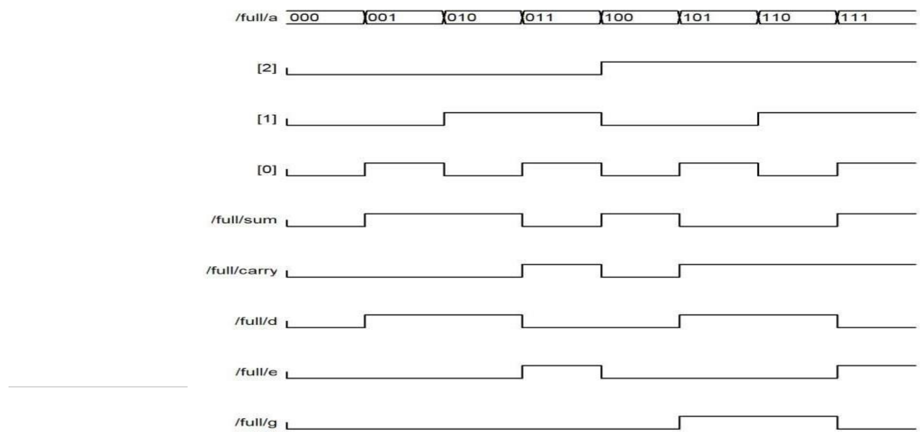
```
module fulladddataflow(x, y, z, sum, carry);
   input x;
   input y;
```

```
   input z;
   output sum;
   output carry;
assign#2 p=x&y;
assign#2 q=y&z;
assign#2 r=z&x;
assign#4 sum=x^y^z;
assign#4carry =(p | q) | r;

 endmodule
```

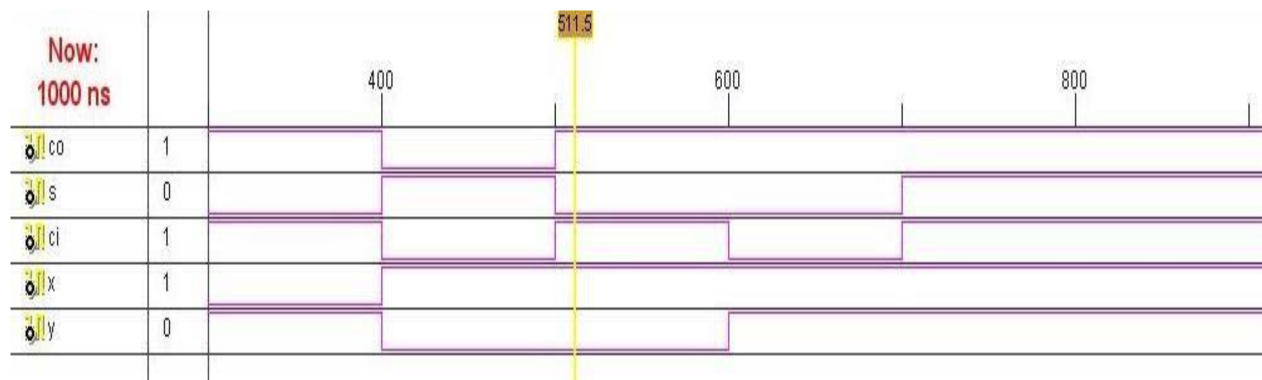
**LOGIC DIAGRAM:**



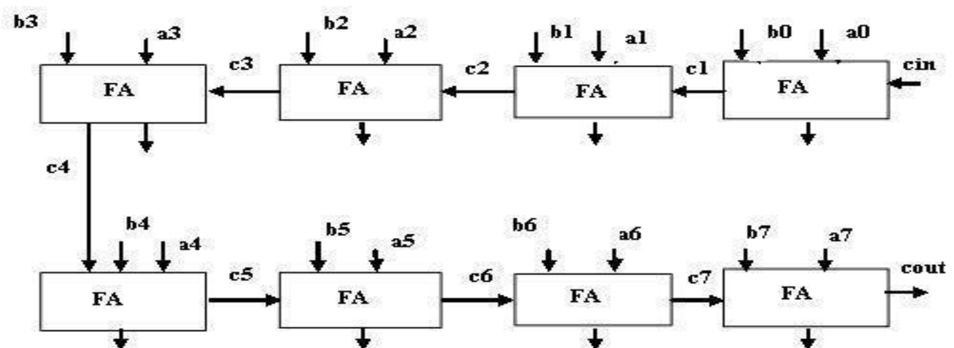

**EXPECTED OUTPUT WAVEFORM:**



**SIMULATION OUTPUT WAVEFORM:**




**Circuit diagram:**




**RESULT:**

Thus the logic circuit for the Full adder is designed in Verilog HDL and the output is verified.

# EXPERIMENT: 3
# DESIGN OF 2-TO-4 DECODER

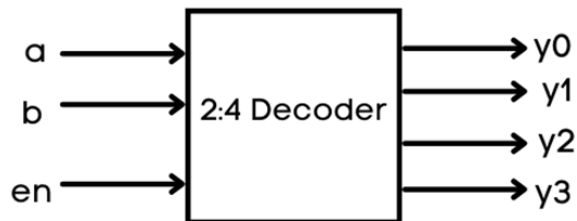
**AIM:**

To develop the source code for decoder by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

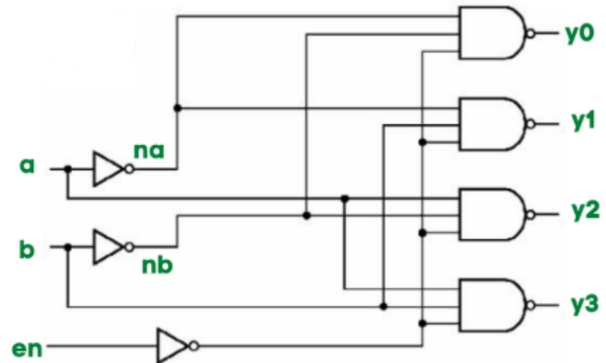
**SOFTWARE & HARDWARE:**

XILINX - VIVADO

LOGIC DIAGRAM:



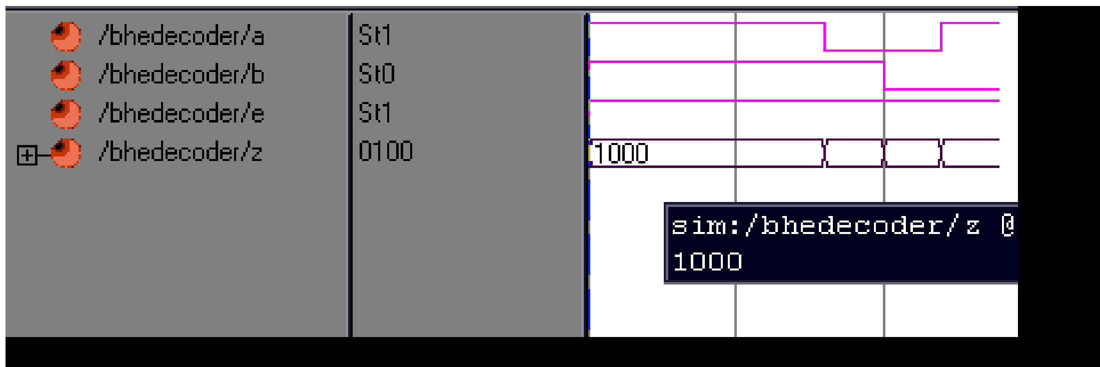

| En | Input | | Output | | | |
|---|---|---|---|---|---|---|
| | a | b | y3 | y2 | y1 | y0 |
| 1 | x | x | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |




**VERILOG SOURCE CODE:**

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

**SIMULATION OUTPUT:**




**RESULT:**

Thus the OUTPUT's of decoder are verified by synthesizing and simulating the VERILOG code.

# EXPERIMENT: 4
# DESIGN OF 8-TO-3 ENCODER

**AIM:**

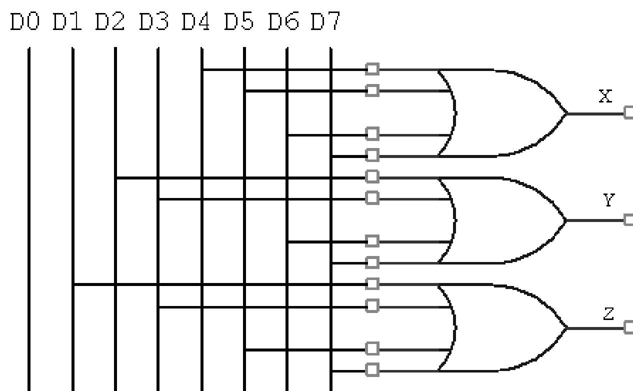
To develop the source code for 8-to-3 encoder by using VERILOG

**SOFTWARE & HARDWARE:**

XILINX -VIVADO

**ENCODER:**

LOGIC DIAGRAM:



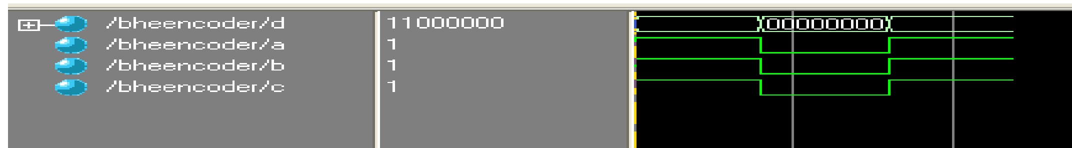

TRUTH TABLE:

| 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 a,b,c;
            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 synthesizing and simulating the VERILOG code.

# EXPERIMENT: 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, synthesis, place and route and implement into FPGA.

**SOFTWARE & HARDWARE:**

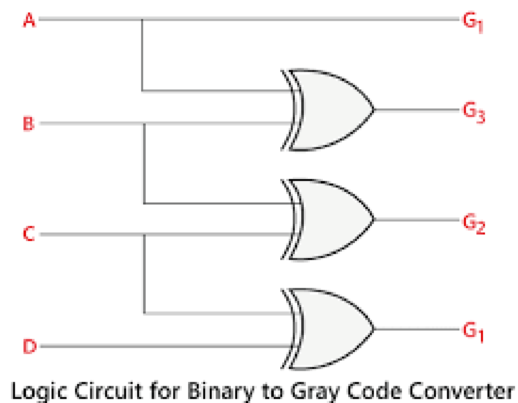
XILINX-VIVADO

**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:



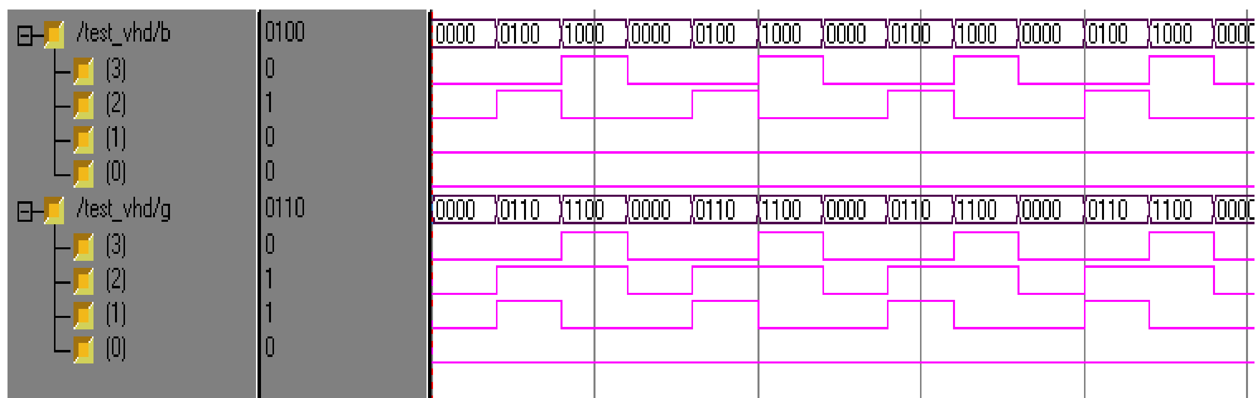
Logic Circuit for Binary to Gray Code Converter

**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 synthesizing and simulating the VERILOG code.

# EXPERIMENT: 6
# DESIGN OF FLIP FLOPS (SR, JK, D)

**AIM:**

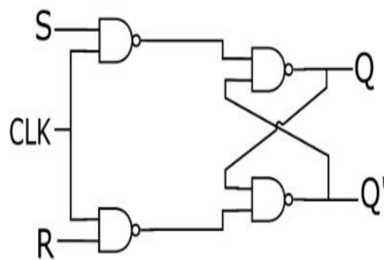
To develop the source code for FLIP FLOPS by using VERILOG and obtained the simulation, synthesis, place and route and implement into FPGA.

**SOFTWARE & HARDWARE:**

XILINX-VIVADO

**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 |

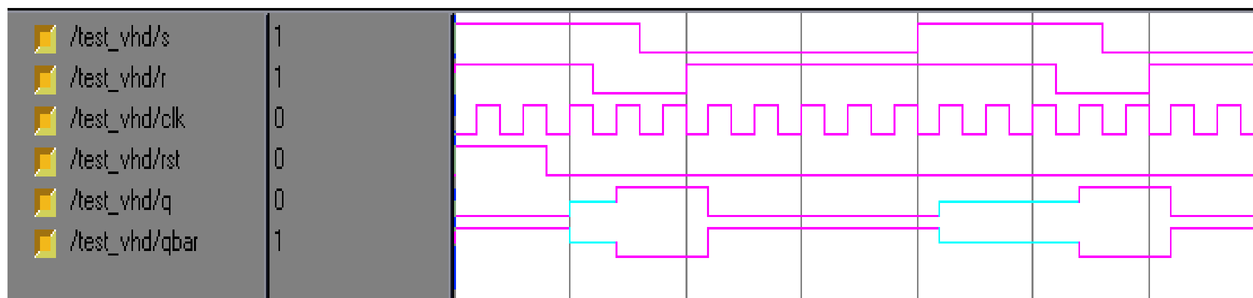
**VERILOG SOURCE CODE:**

**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;
```

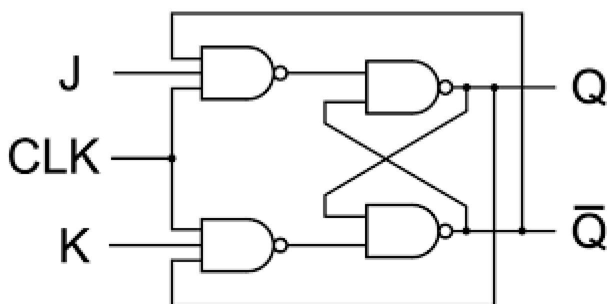
```
end
        else if(s==1'b0 &&      r==1'b1)
         begin
q= 1'b0; qbar= 1'b1;
end
        else  if(s==1'b1  && r==1'b0)
         begin
q= 1'b1; qbar= 1'b0;
end
else
begin
q=1'bx;qbar=1'bx;
end
end
endmodule
```

**SIMULATION OUTPUT:**




**JK FLIPFLOP:**

LOGIC DIAGRAM:




TRUTH TABLE:

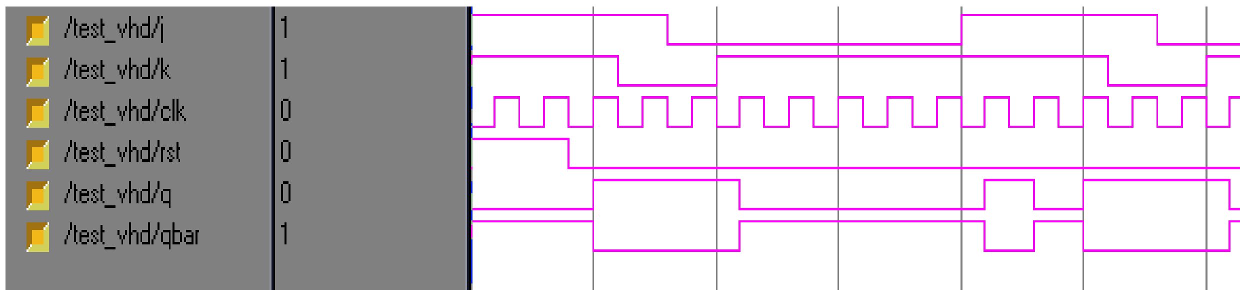
| Q(t) | J | K | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**VERILOG SOURCE CODE:**

**Behavioral Modeling:**

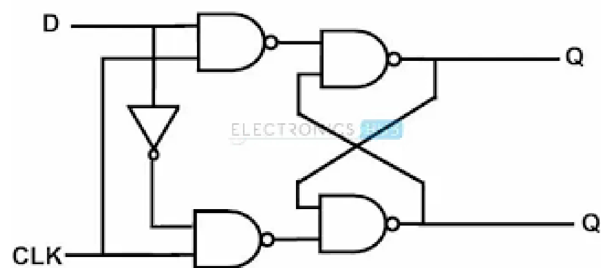
```
module jkff(j, k, clk, rst, q, qbar);
   input j;
   input k;
   input clk;
   input rst;
   output q;
   output qbar;
         reg q;
         reg qbar;
         always @ (posedge(clk) or posedge(rst)) begin
         if (rst==1'b1)
         begin
         q=1'b0;
         qbar=1'b1;
         end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

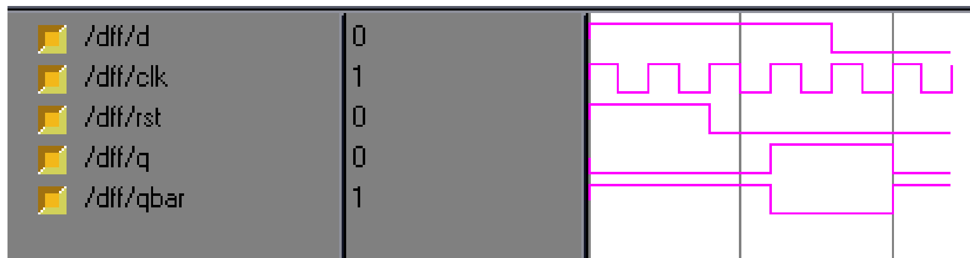
**VERILOG SOURCE CODE:**

**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
endmodule
```

**SIMULATION OUTPUT:**




**RESULT:**

Thus the OUTPUT's of Flip flops using three modeling styles are verified by synthesizing and simulating the VERILOG code.

# EXPERIMENT: 7
# DESIGN OF MULTIPLEXERS

**AIM:** Design a 4 to 1 Multiplexer circuit in Verilog.

**SOFTWARE REQUIRED:**

XILINX-VIVADO

**THEORY:**

A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. The selection of a particular input line is controlled by a set of selection lines. Normally, there are $2^n$ input lines and n selection lines whose bit combinations determine which input is selected. A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output lines. Multiplexer ICs may have an enable input to control the operation of the unit. When the enable input is in a given binary state (the disable state), the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as normal multiplexer. The enable input (sometimes called strobe) can be used to expand two or more multiplexer ICs to digital multiplexers with a larger number of inputs.The size of the multiplexeris specified by the number $2^n$ of its input lines and the single output line. In general, a $2^n$ – to – 1 line multiplexer is constructed from an n – to $2^n$ decoder by adding to it $2^n$ input lines, one to each AND gate. The outputs of the AND gates are applied to a single OR gate to provide the 1 – line output.

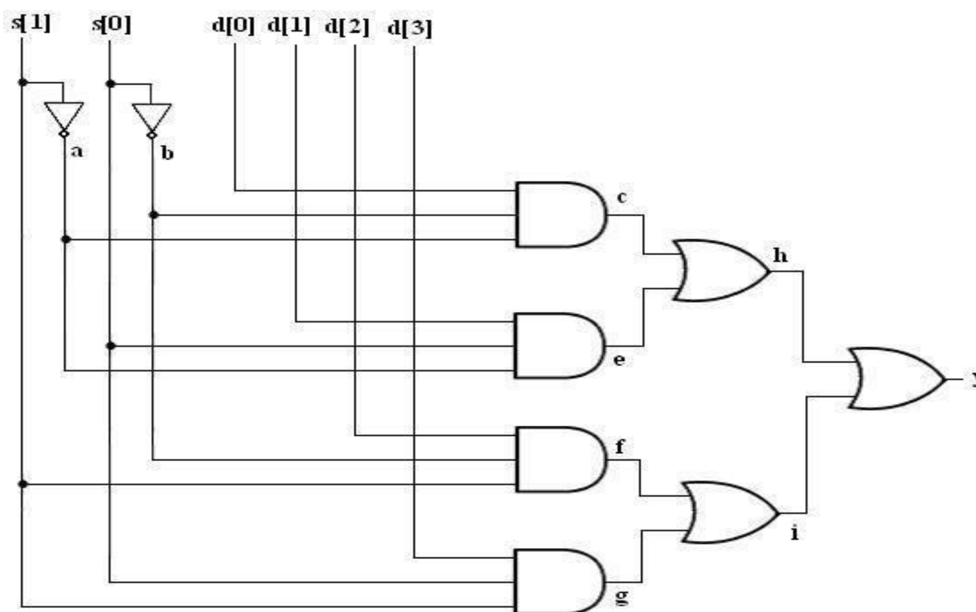
**PROCEDURE:**

1) The multiplexer circuit is designed and the Boolean function is found out.
2) The Verilog Module Source for the circuit is written.
3) It is implemented in Model Sim and Simulated.
4) Signals are provided and Output Waveforms are viewed.

**TRUTH TABLE:**

| INPUT | | OUTPUT |
|---|---|---|
| s[1] | s[0] | y |
| 0 | 0 | D[0] |
| 0 | 1 | D[1] |
| 1 | 0 | D[2] |
| 1 | 1 | D[3] |

**LOGIC DIAGRAM 4 TO 1 MULTIPLEXER:**

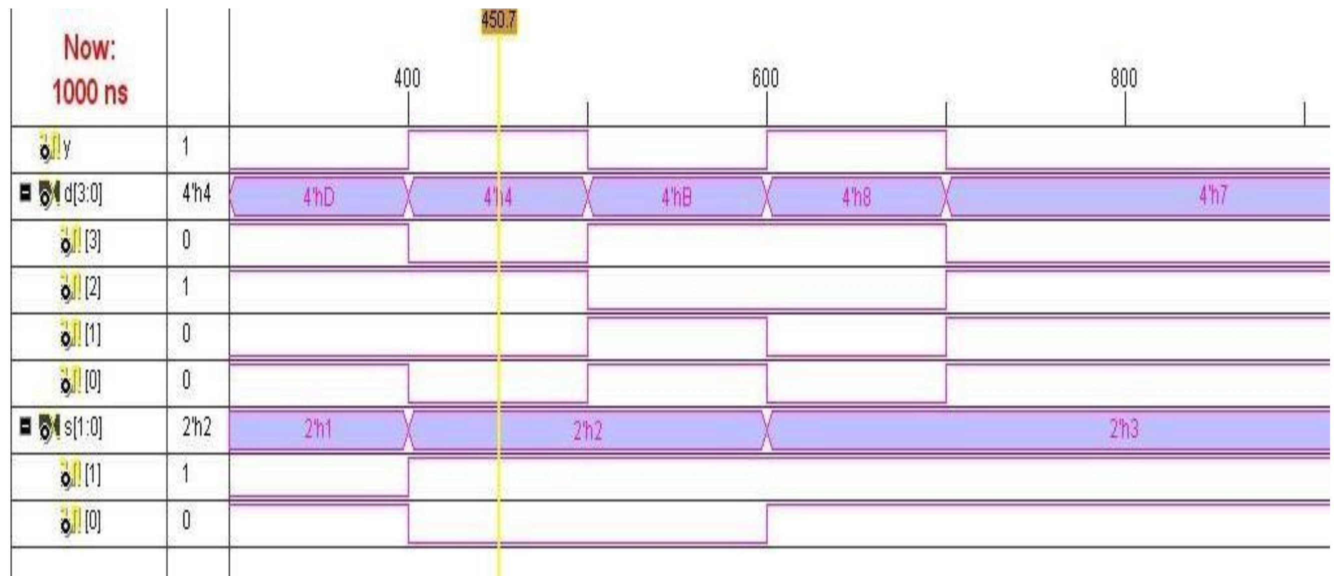



**MULTIPLEXER USING VERILOG CODE:**

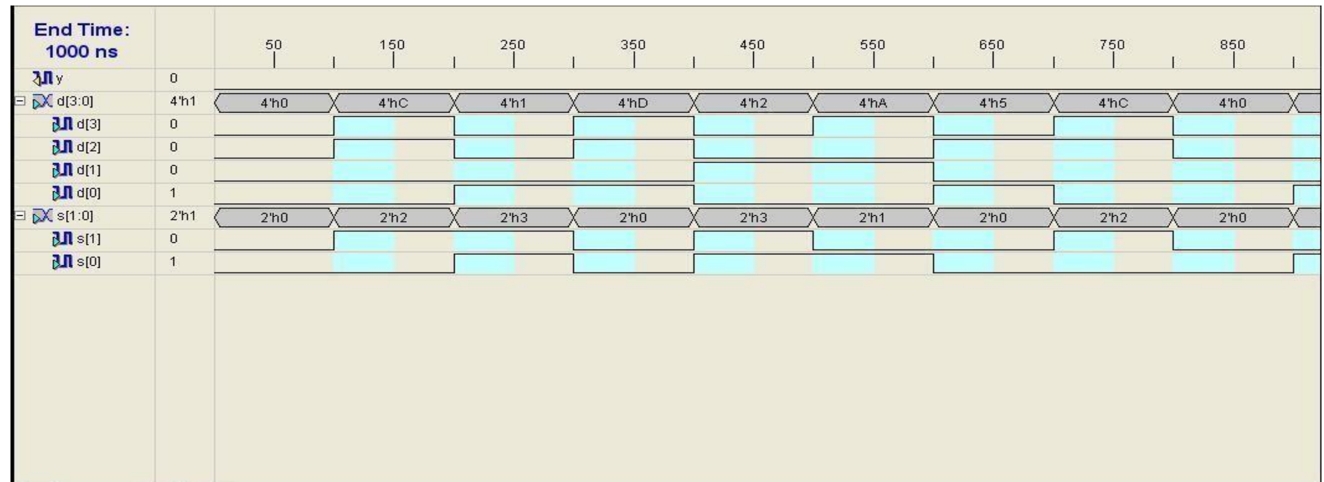
```
module
multiplexer(y,d,s);
output y;
input [3:0] d;
input [1:0] s;
wire a,b,c,e,f,g,h,i;
//Instantiate Primitive gates
not (a,s[1]);
```

```
not (b,s[0]);
and (c,d[0],b,a);
and (e,d[1],s[0],a);
and (f,d[2],b,s[1]);
and (g,d[3],s[0],s[1]);
or (h,c,e);
or (i,f,g);
or (y,h,i);
endmodule

//Stimulus for testing 4 to 1 Multiplexer

module simulation;
reg [3:0]d;
reg [1:0]s;
wire y;
//Instantiate the 4 to 1 Multiplexer
multiplexer mux_t(y,d,s);
initial begin
s=2'b00;d[0]=1'b1;d[1]= 1'b0;d[2]= 1'b0;d[3]= 1'b0; #100
s=2'b00;d[0]= 1'b0;d[1]= 1'b1;d[2]= 1'b1;d[3]= 1'b1; #100
s=2'b01;d[0]= 1'b0;d[1]= 1'b1;d[2]= 1'b0;d[3]= 1'b0; #100
s=2'b01;d[0]= 1'b1;d[1]= 1'b0;d[2]= 1'b1;d[3]= 1'b1; #100
s=2'b10;d[0]= 1'b0;d[1]= 1'b0;d[2]= 1'b1;d[3]= 1'b0; #100
s=2'b10;d[0]= 1'b1;d[1]= 1'b1;d[2]= 1'b0;d[3]= 1'b1; #100
s=2'b11;d[0]= 1'b0;d[1]= 1'b0;d[2]= 1'b0;d[3]= 1'b1; #100
s=2'b11;d[0]= 1'b1;d[1]= 1'b1;d[2]= 1'b1;d[3]= 1'b0;
end
endmodule
```

**WAVEFORM OF MULTIPLEXERS:**




**TEST BENCH WAVEFORM OF MULTIPLEXERS**:




**RESULT:** Thus the multiplexer is designed in Verilog HDL and the output is verified.

# EXPERIMENT: 8

## RIPPLE COUNTER REALIZATION IN VERILOG HDL

**AIM:** To realize an asynchronous ripple counter in Verilog

**SOFTWARE REQUIRED:**

XILINX-VIVADO

**THEORY:**

In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. In other words, the Clock Pulse inputs of all flip-flops (except the first) are triggered not by the incoming pulses, but rather by the transition that occurs in other flip-flops. A binary ripple counter consists of a series connection of complementing flip-flops (JK or T type), with theoutput of each flip-flop connected to the Clock Pulse input of the next higher-order flip-flop. The flip-flop holding the LSB receives the incoming count pulses. All J and K inputs are equal to 1. Thesmall circle in the Clock Pulse /Count Pulse indicates that the flip-flop complements during a negative-going transition or when the output to which it is connected goes from 1 to 0. The flip- flops change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion. A binary counter with reverse count is called a binary down-counter. In binary down-counter, the binary count is decremented by 1 with every input count pulse.

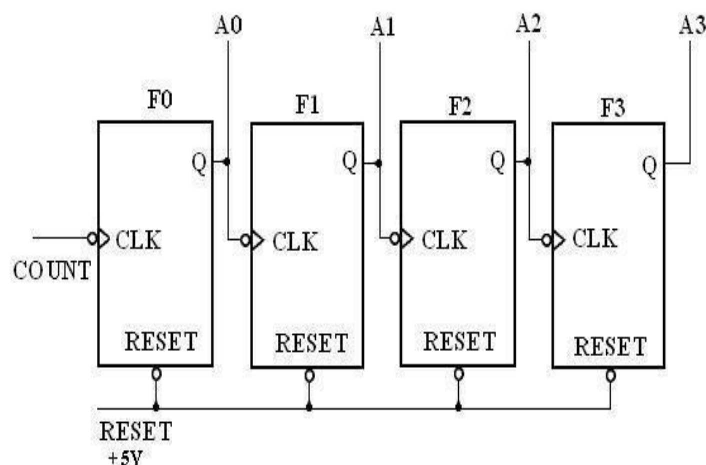
**PROCEDURE:**

1) The 4 bit asynchronous ripple counter circuit is designed.
2) The Verilog Module Source for the circuit is written.
3) It is implemented in Model Sim and Simulated.
4) Signals are provided and Output Waveforms are viewed.

```
//Structural description of Ripple Counter

module
ripplecounter(A0,A1,A2,A3,COUNT,RESET);
output A0,A1,A2,A3;
input COUNT,RESET;
//Instantiate Flip-Flop
```

```
 FF F0(A0,COUNT,RESET);
 FF F1(A1,A0,RESET);
 FF F2(A2,A1,RESET);
 FF F3(A3,A2,RESET);
 endmodule

//Description of Flip-Flop
module
 FF(Q,CLK,RESET);
 output Q;
input CLK,RESET; reg Q;
always @(negedge CLK or negedge RESET) if(~RESET)
Q=1'b0; else
Q=(~Q); endmodule

//Stimulus for testing Ripple Counter

module simulation; reg COUNT;
reg RESET;
wire A0,A1,A2,A3;

//Instantiate Ripple Counter
ripplecounter rc_t(A0,A1,A2,A3,COUNT,RESET);
 always
#5 COUNT=~COUNT;
 Initial begin
COUNT=1'b0;
RESET=1'b0; #10 RESET=1'b1; end
endmodule
```

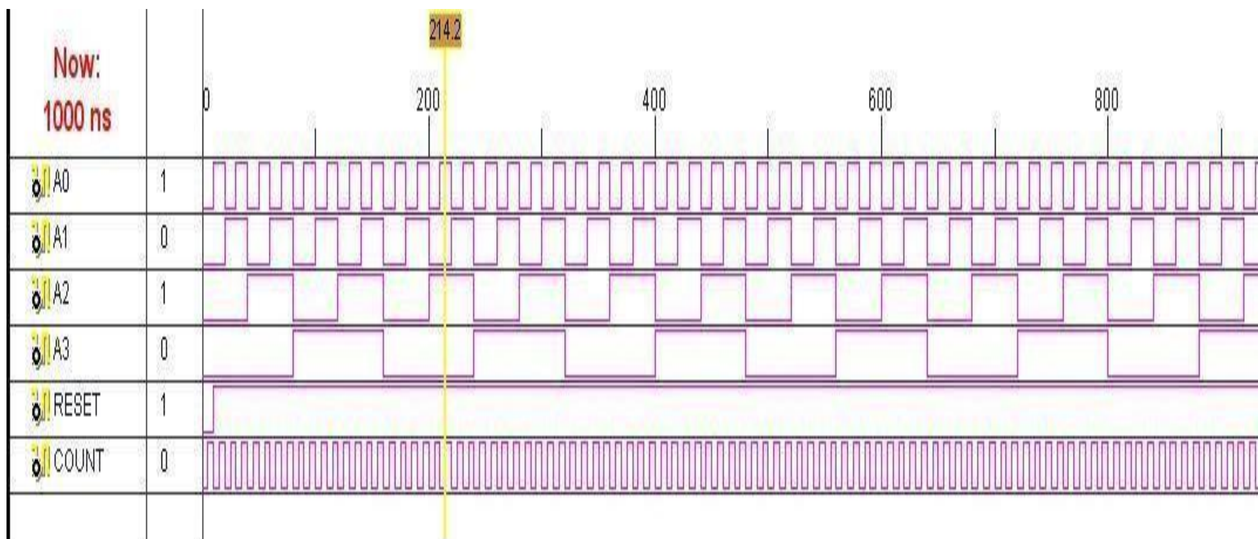
**LOGIC DIAGRAM:**
**4-BIT RIPPLE COUNTER:**



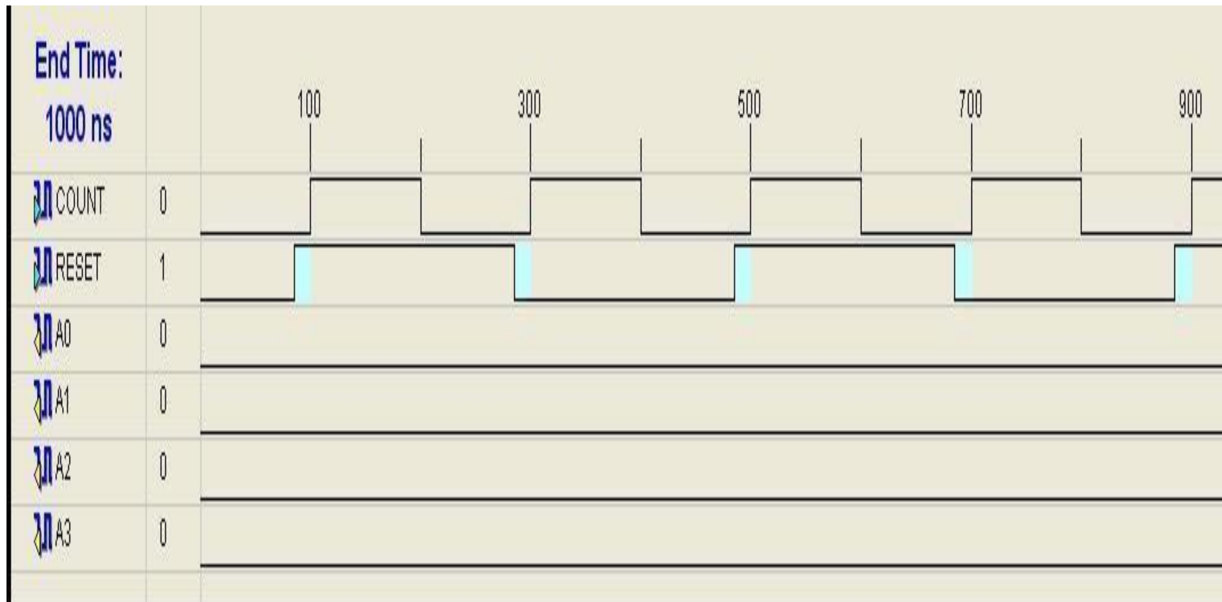
**TRUTH TABLE:**

| COUNT | A0 | A1 | A2 | A3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 1 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

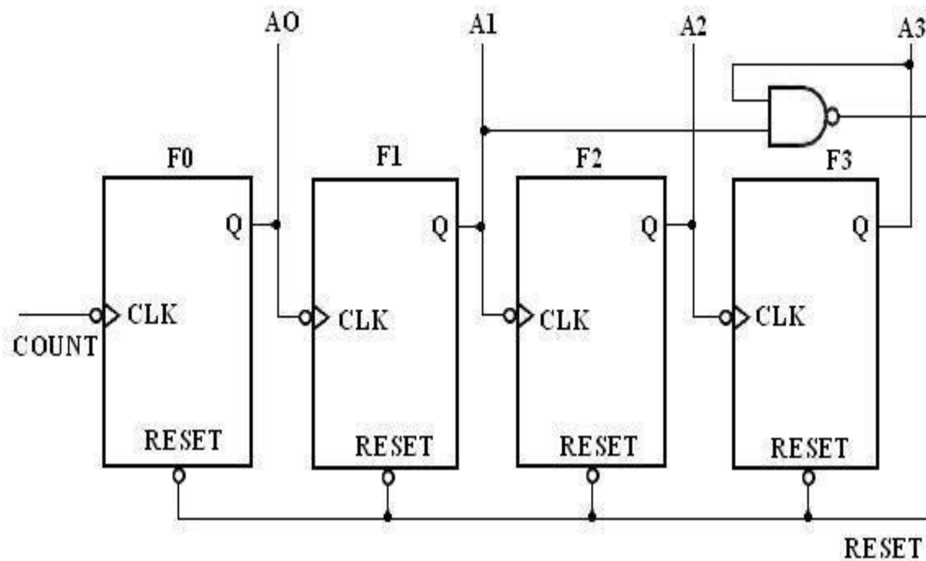
**WAVEFORM OF RIPPLE COUNTER:**



**TESTBENCHWAVEFORM OF RIPPLE COUNTER:**




**LOGIC DIAGRAM MOD-10 RIPPLE COUNTER:**



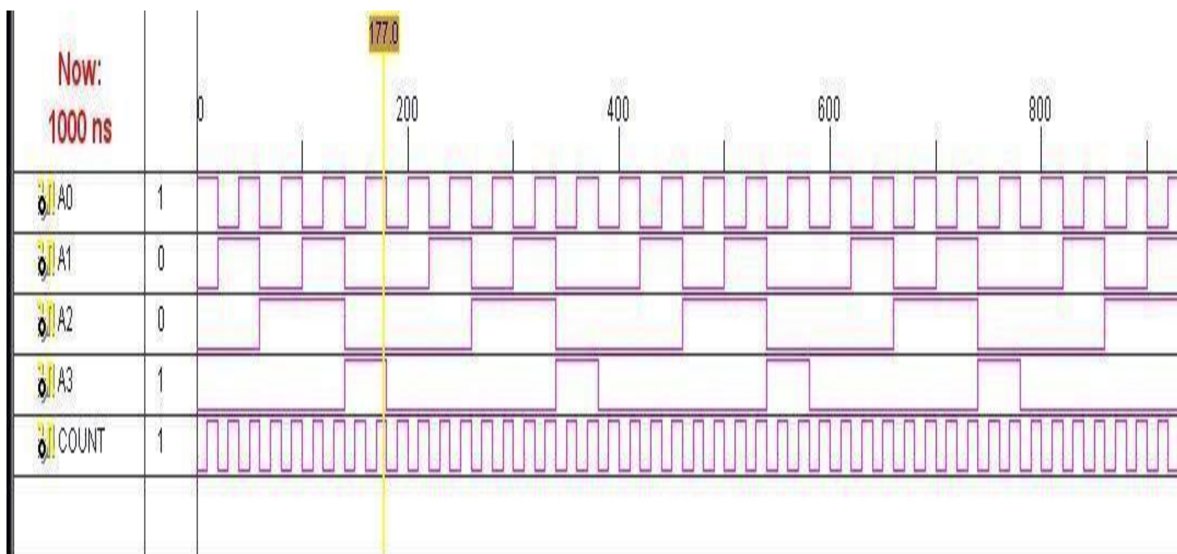
**TRUTH TABLE**:

| COUNT | A0 | A1 | A2 | A3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 0 | 0 | 0 |

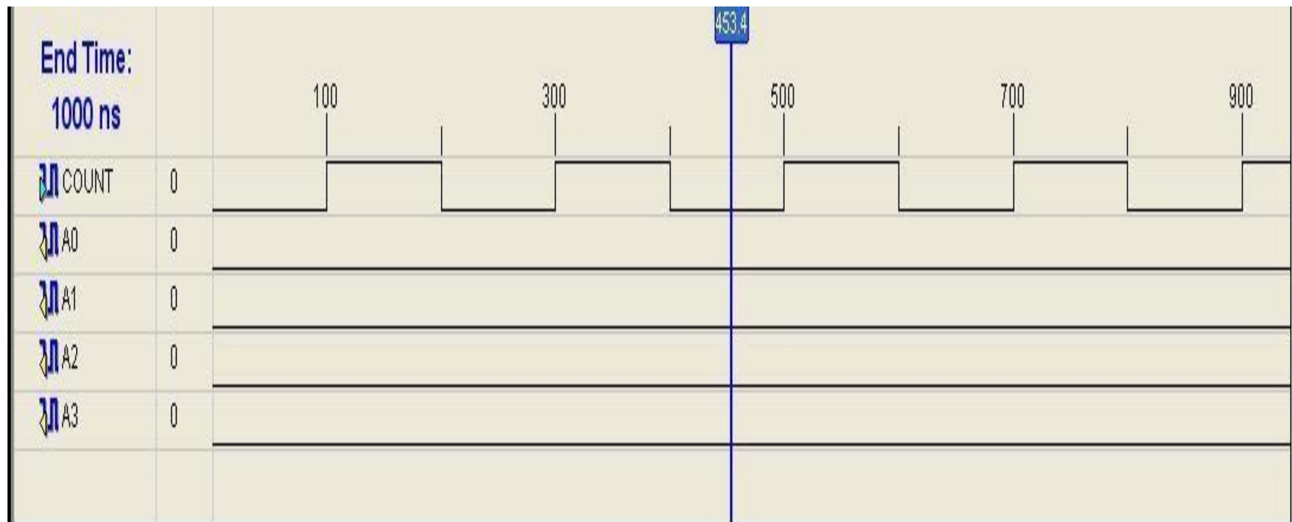
```
/Structural description of MOD10 Counter
module
 MOD10(A0,A1,A2,A3,COUNT);
 output A0,A1,A2,A3;
input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
//Instantiate Primitive gate
nand (RESET,A1,A3);
endmodule

//Description of Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q=1'b0;
always @(negedge CLK or negedge RESET)
if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

```
//Stimulus for testing MOD10 Counter module simulation;
reg COUNT;
wire A0,A1,A2,A3;
//Instantiate MOD10 Counter
MOD10 MOD10_TEST(A0,A1,A2,A3,COUNT);
Always #10 COUNT=~COUNT;
initial
begin
COUNT=1'b0;
end
Endmodule
```

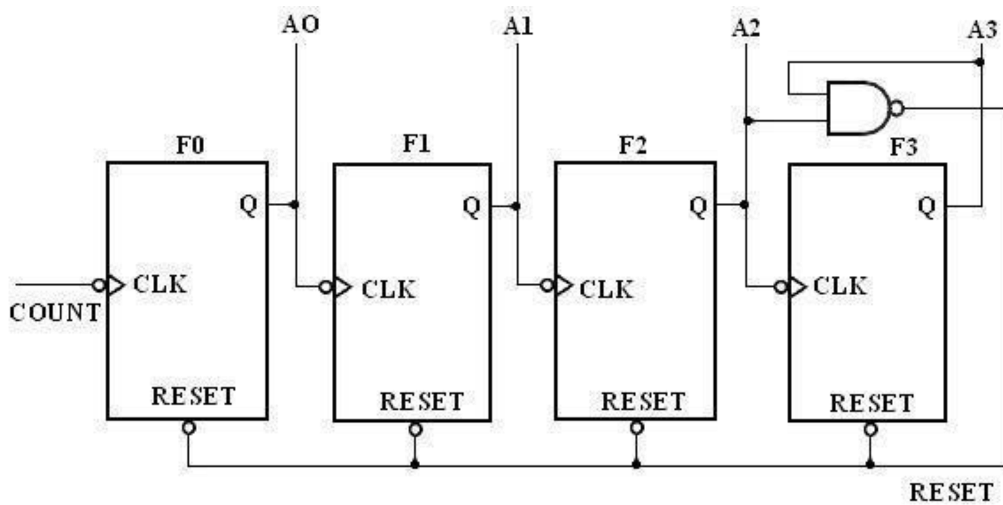
**WAVEFORM OF MOD 10 COUNTERS:**



**TESTBENCHWAVEFORM OF MOD 10:**




**LOGIC DIAGRAM MOD-12 RIPPLE COUNTER:**



**TRUTH TABLE**:

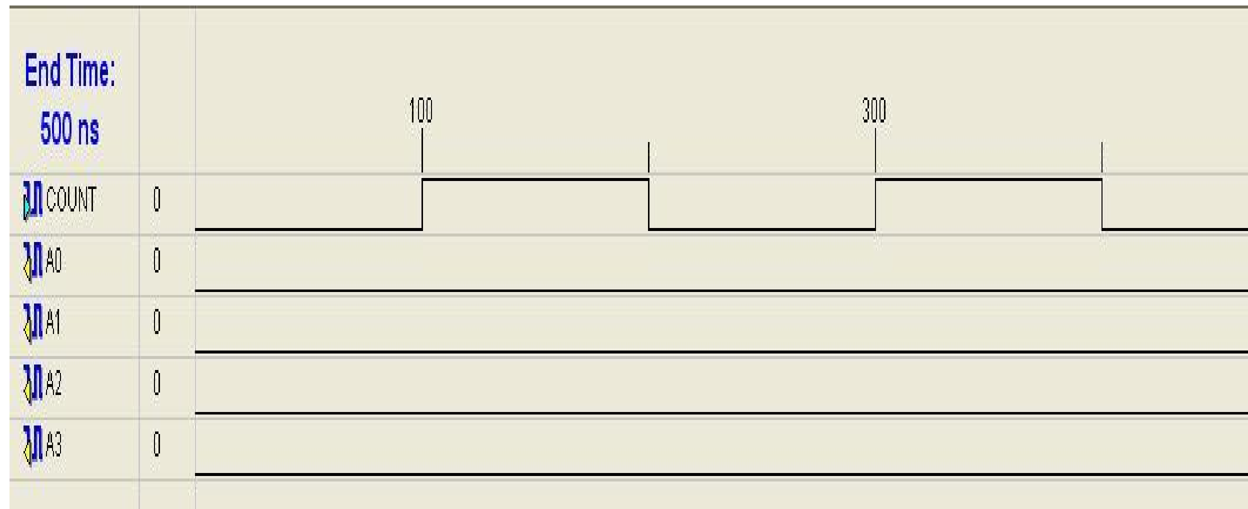
| COUNT | A0 | A1 | A2 | A3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 0 | 0 |

```
//Structural description of MOD12 Counter

Module
 MOD12(A0,A1,A2,A3,COUNT);
 output A0,A1,A2,A3;
 input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
//Instantiate Primitive gates
nand (RESET,A2,A3);
endmodule

//Description of Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q=1'b0;
always @(negedge CLK or negedge RESET) if(~RESET)
Q=1'b0;
```

```
else
Q=(~Q);
endmodule

//Stimulus for testing MOD12 Counter

module simulation;
reg COUNT;
wire A0,A1,A2,A3;
//Instantiate MOD12 Counter
MOD12 MOD12_TEST(A0,A1,A2,A3,COUNT);
 always
#10 COUNT=~COUNT;
initial
begin
COUNT=1'b0;
end
endmodule
```

**WAVEFORM OF MOD 12 COUNTERS:**

**TESTBENCHWAVEFORM OF MOD 12 COUNTERS:**




**RESULT:**

Thus the ripple counter is designed in Verilog HDL and the output is verified.

**EXPERIMENT: 9**

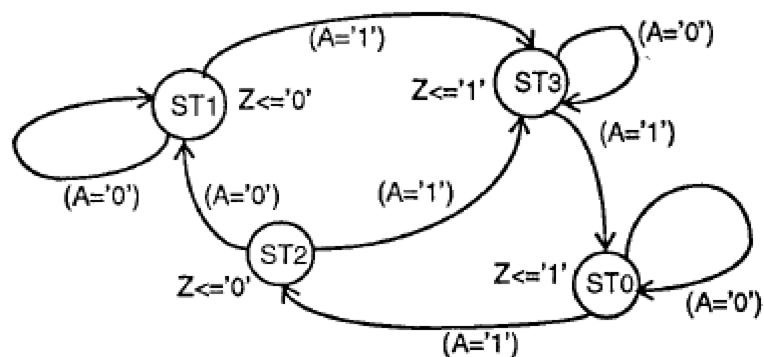
## Sequence Detector (Finite State Machine- Mealy and Moore Machines)

**AIM:**

To develop the source code for Sequence Detector (Finite State Machine- Mealy and Moore Machines) by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**SOFTWARE REQUIRED:**
XILINX- VIVADO

**MOORE FSM:**
LOGIC DIAGRAM:



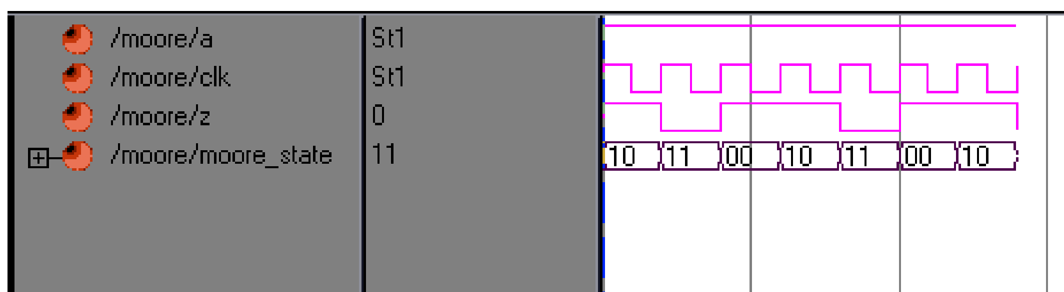

**Behavioral Modeling:**

```
module moorefsm(a,clk,z);
   input a;
   input clk;
   output z;
          reg z;

          parameter st0=0,st1=1,st2=2,st3=3;
          reg[0:1]moore_state;
          initial
          begin
          moore_state=st0;
          end
          always @ (posedge(clk))
          case(moore_state)
          st0:
          begin
          z=1;
          if(a)
          moore_state=st2;
```

end

```
st1:
begin
z=0;
if(a)
moore_state=st3;
end

st2:
begin
z=0;
if(~a)
moore_state=st1;
else
moore_state=st3;
end

st3:
begin
z=1;
if(a)
moore_state=st0;
end
endcase
endmodule
```

**Simulation output:**




**MEALY FSM:**

TRUTH TABLE:

| Present state | Input A = 0 | Input A = 1 |
|---|---|---|
| ST0 | ST0 / 0 | ST3 / 1 |
| ST1 | ST1 / 1 | ST0 / 0 |
| ST2 | ST2 / 0 | ST1 / 1 |
| ST3 | ST2 / 0 | ST1 / 0 |

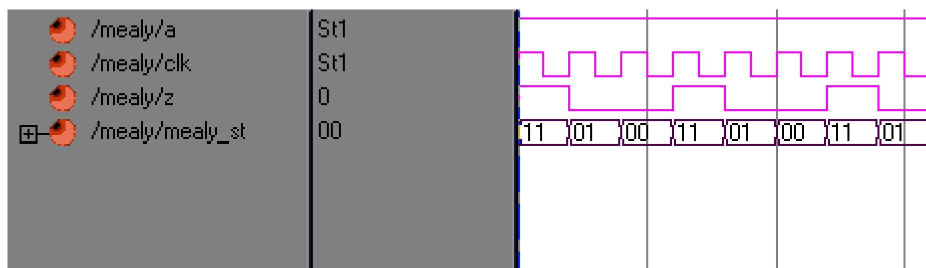
(Entries in table are next state and output Z)

**Behavioral Modeling:**

```
module mealayfsm(a, clk, z);
   input a;
   input clk;
   output z;
         reg z;

         parameter st0=0,st1=1,st2=2,st3=3;
         reg[0:1]mealy_state;
         initial
         begin
         mealy_state=st0;
         end
         always @ (posedge(clk))
         case(mealy_state)
         st0:
         begin
         if(a) begin
         z=1;
         mealy_state=st3; end
         else
         z=0;
         end

         st1:
         begin
         if(a) begin
         z=0;
         mealy_state=st0; end
         else
```

```
z=1;
end

st2:
begin
if(a) begin
z=1;
mealy_state=st1; end
else
z=0;
end

st3:
begin
z=0;
if(a) begin
mealy_state=st1;       end
else
mealy_state=st2;
end

endcase
endmodule
```

**SIMULATION OUTPUT:**




**RESULT:**

Thus the OUTPUT's of Moore and Mealy FSM is verified by synthesizing and simulating the VHDL and VERILOG code.

# CYCLE -II

## EXPERIMENT: I

## DESIGN AND IMPLEMENTATION OF AN INVERTER

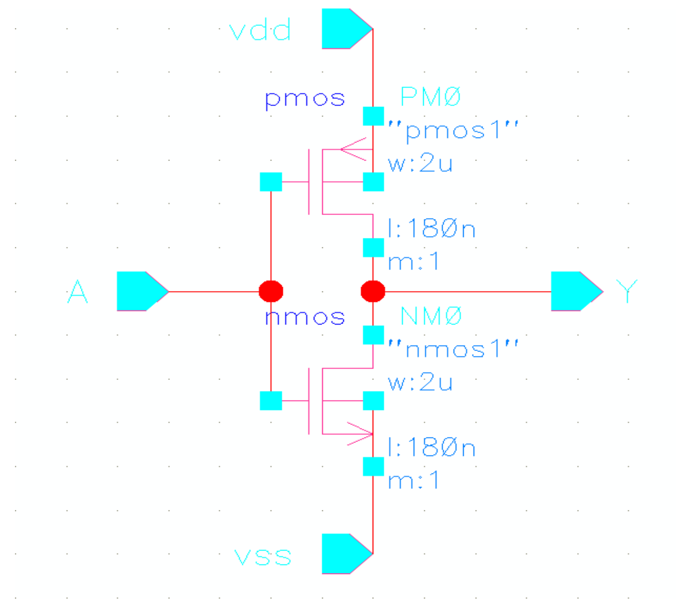
**AIM:** To design and Implementation of an Inverter

**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.

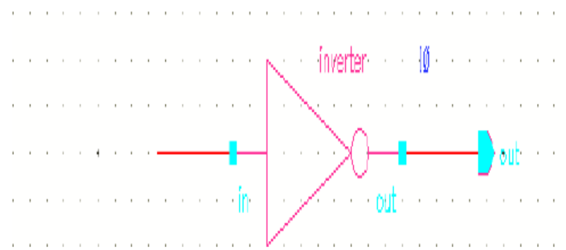
**Schematic Capture:**



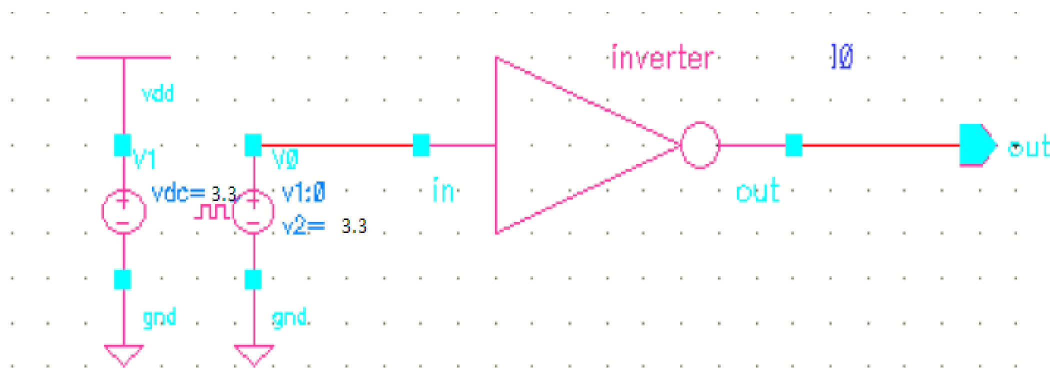

**Procedure:**

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.

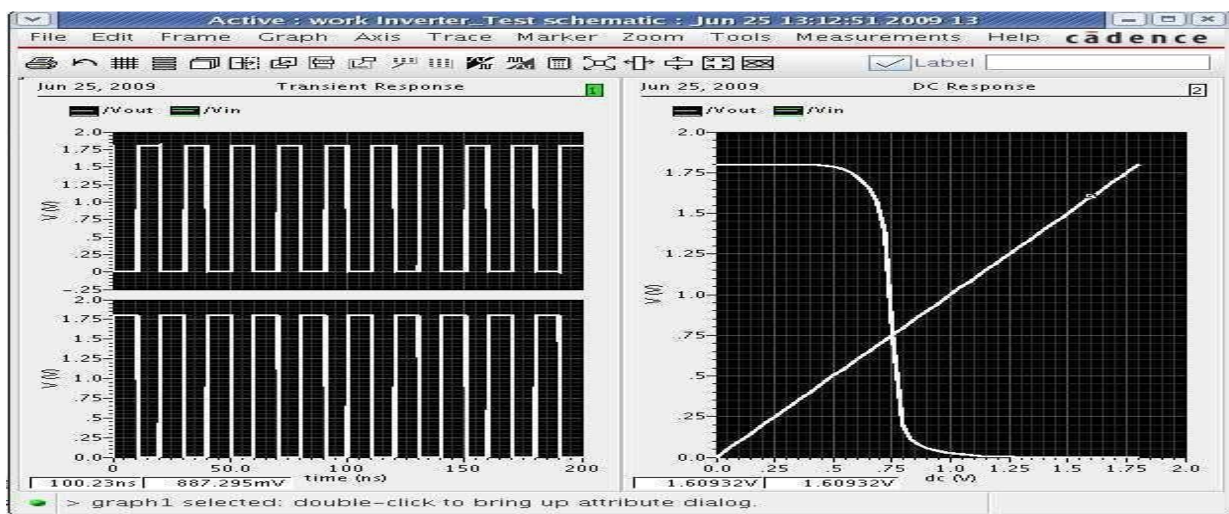
**Schematic Symbol:**



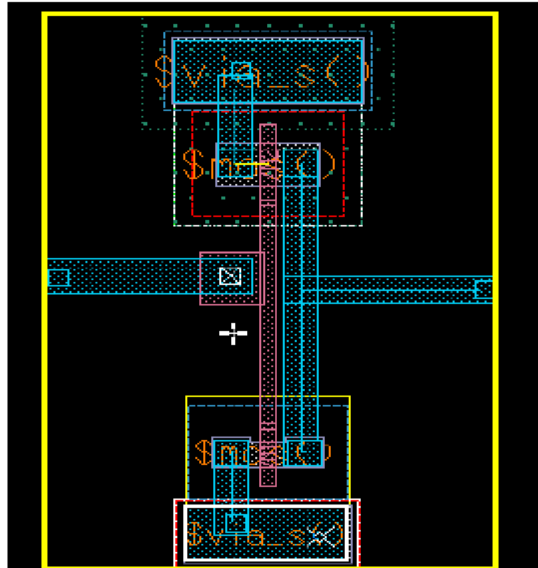

**Testing the Schematic:**




**Simulation Output: Input Vs Output Transient and DC Characteristics:**



**Layout of the Inverter:**



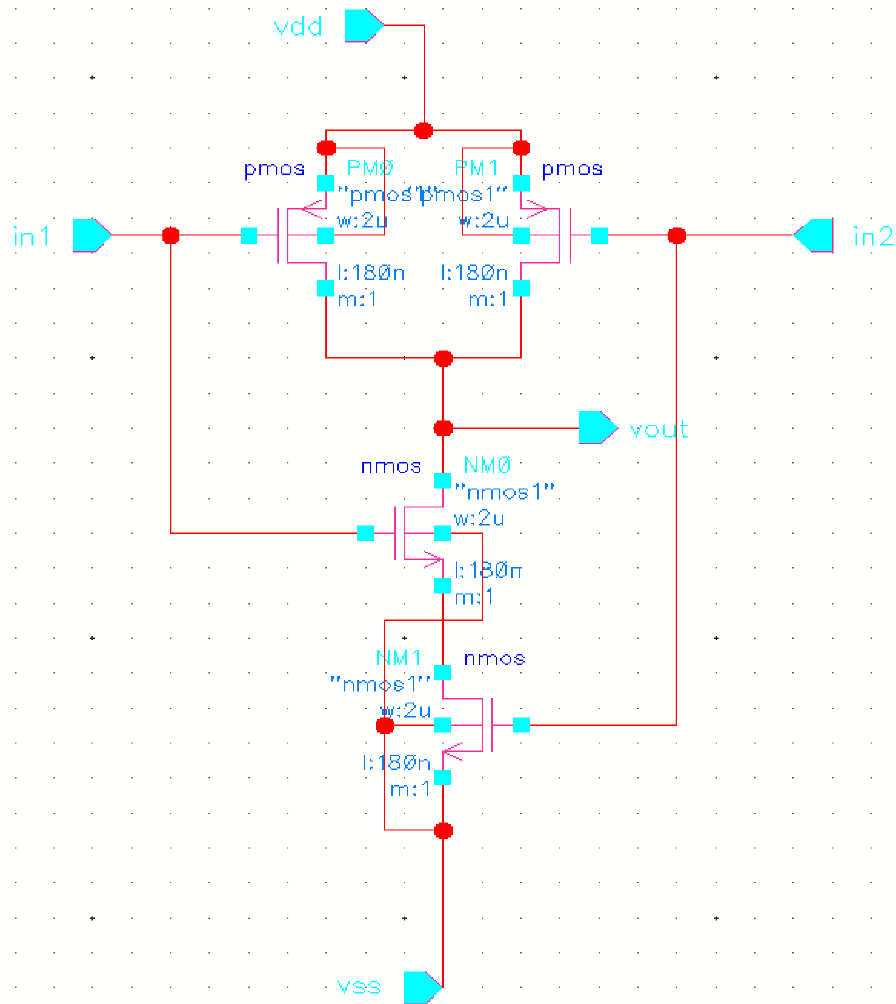
## EXPERIMENT: 2

## 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

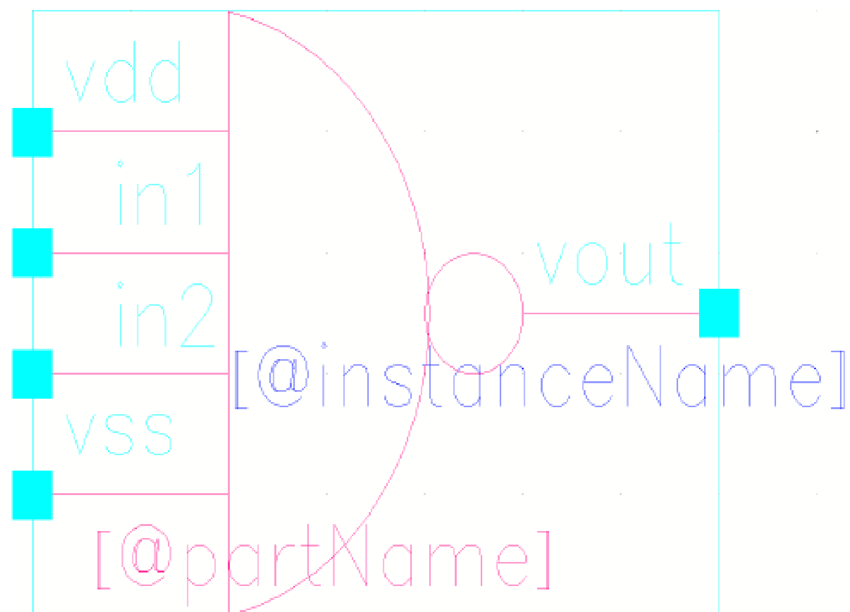
**Schematic Diagram**



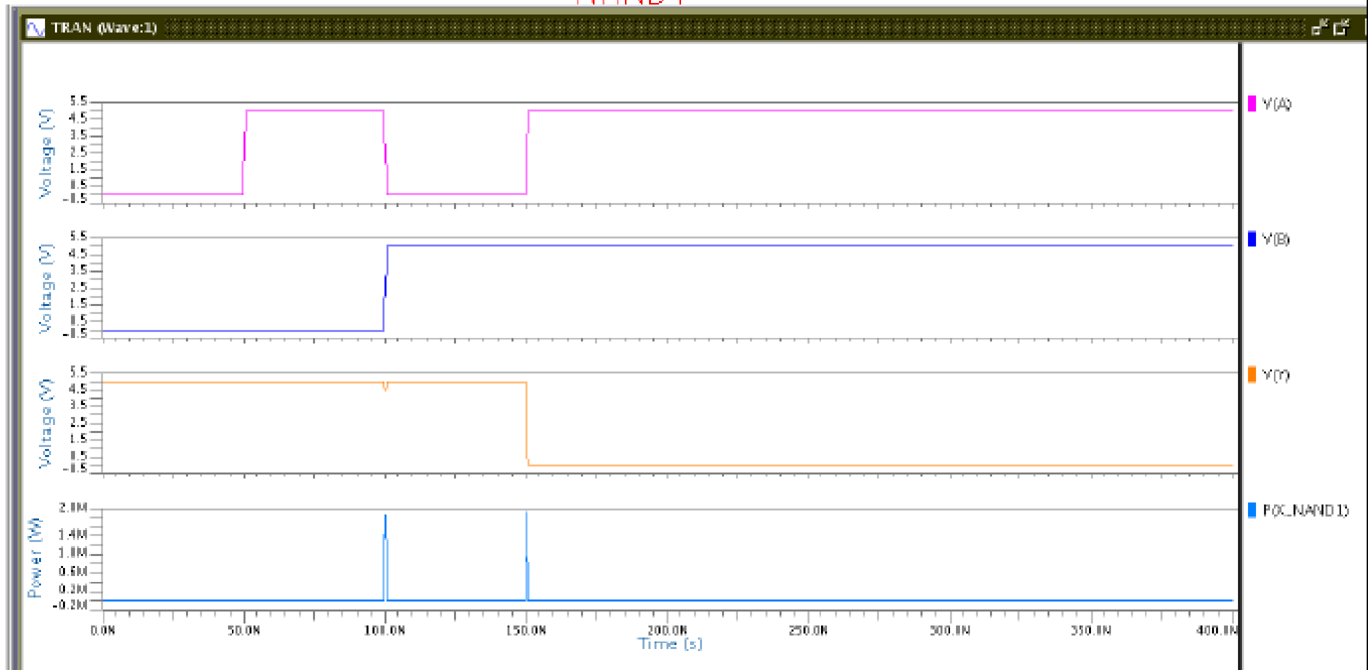
**PROCEDURE:**

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.

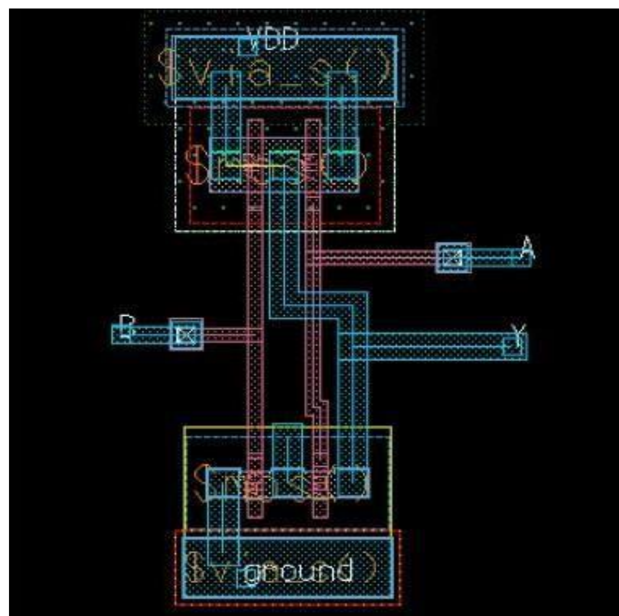
**Symbol Creation:**



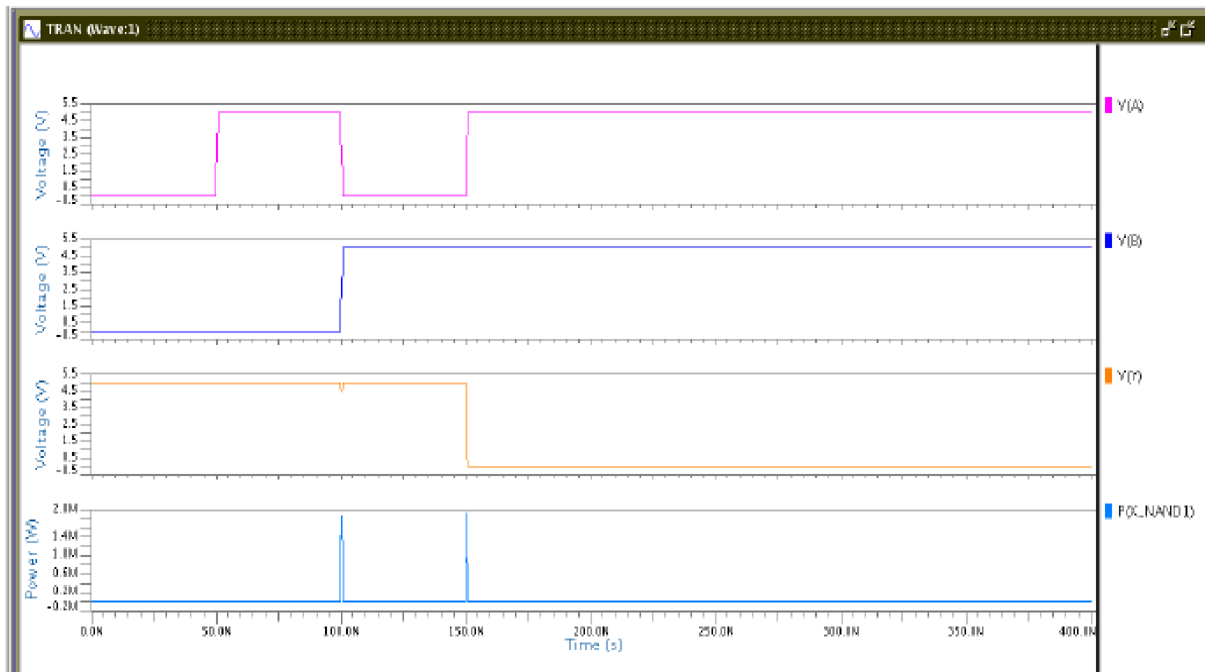
**Building the NAND Test Design**




**Creating a layout view of NAND gate**



**Simulation Output:**

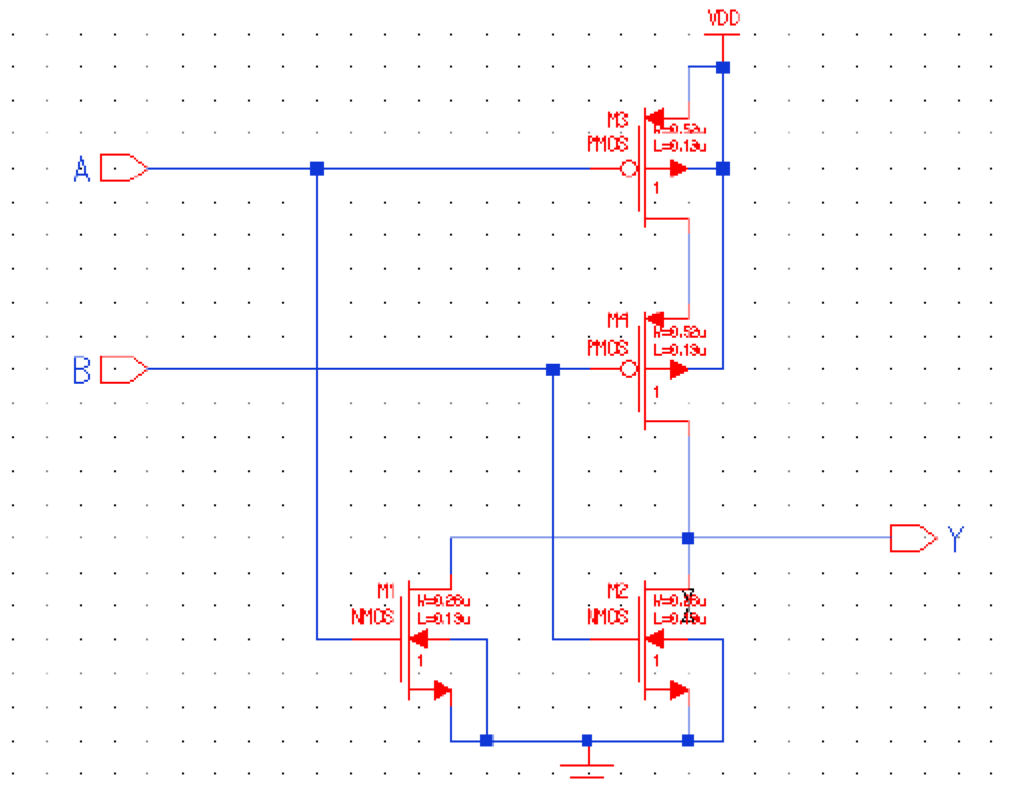


## EXPERIMENT: 3
## NOR GATE

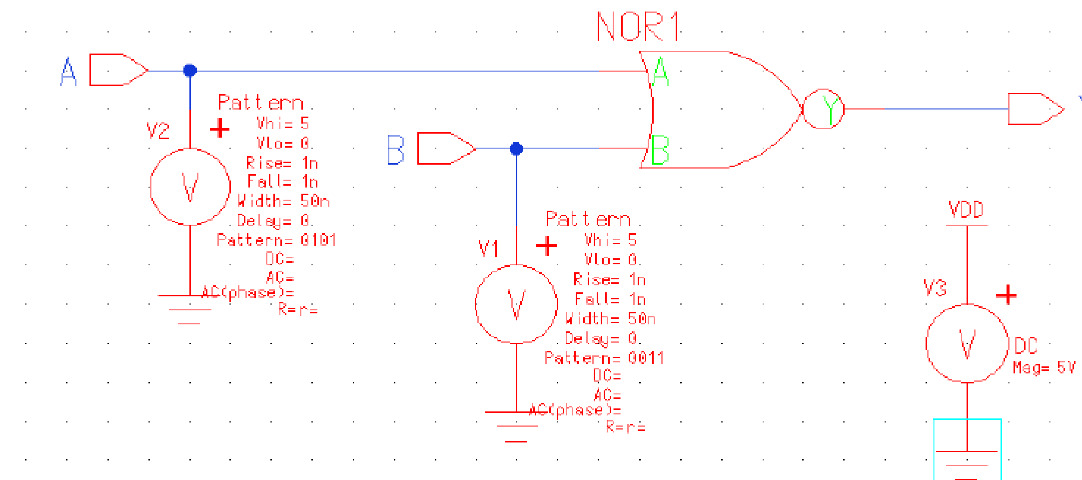
**AIM:** To design and simulate the CMOS NOR gate

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

**CIRCUIT DIAGRAM:**



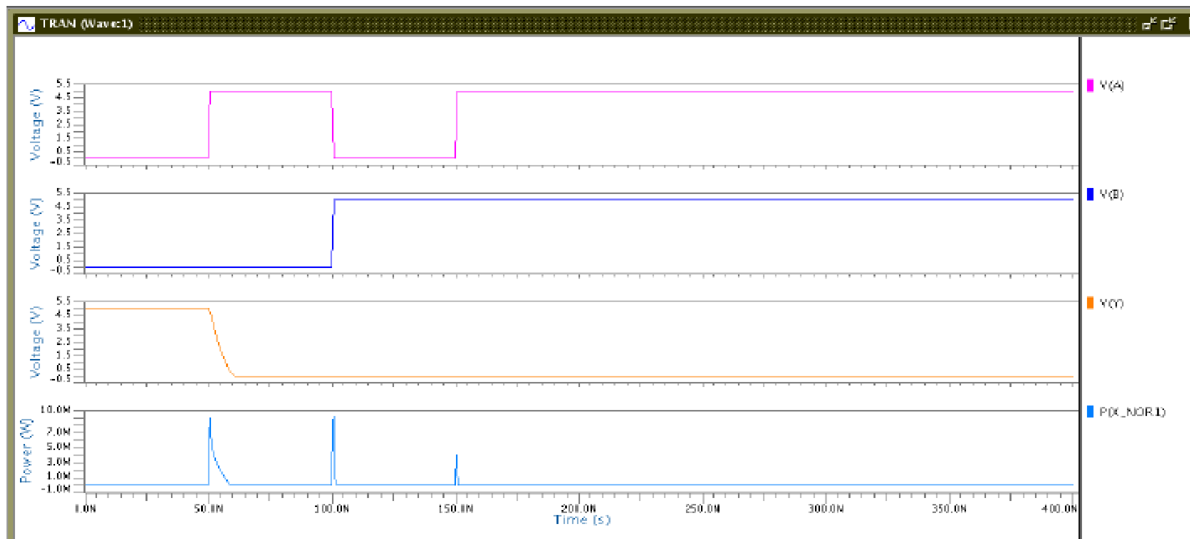
**SIMULATION CIRCUIT:**



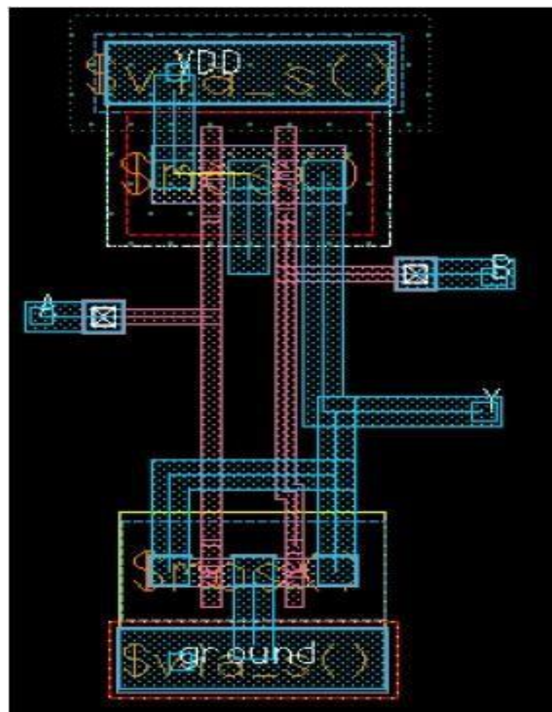

**PROCEDURE:**

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.

**Simulation Output:**



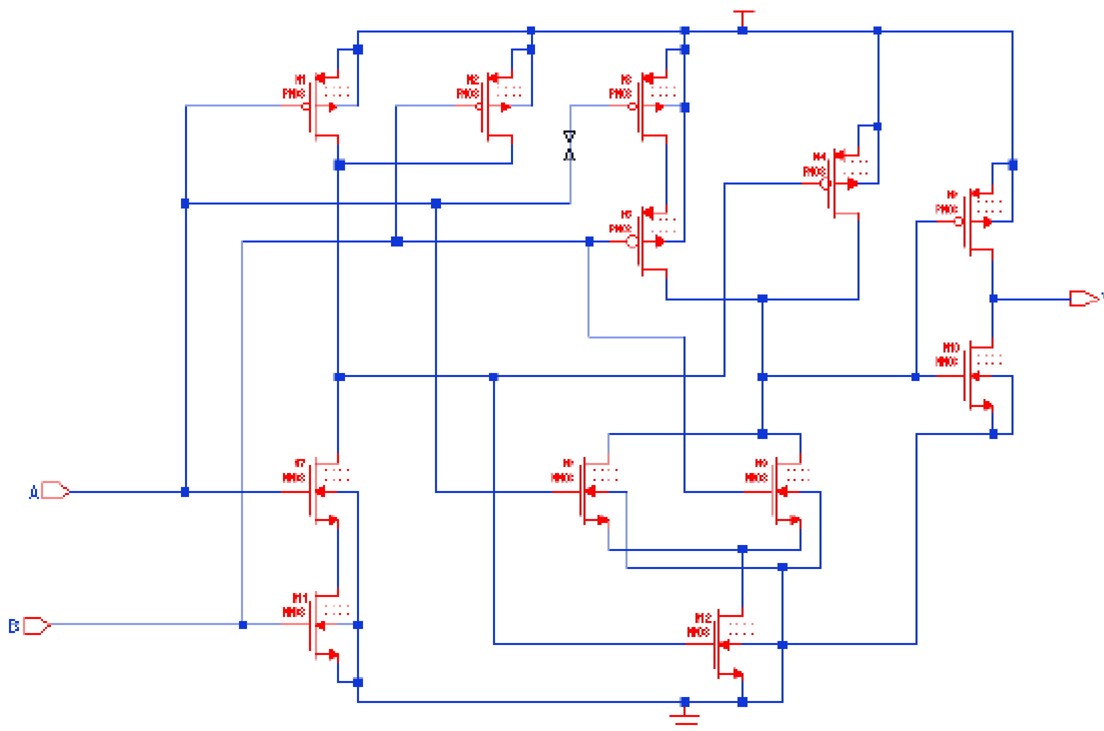

**Layout:**



# EXPERIMENT: 4
# 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

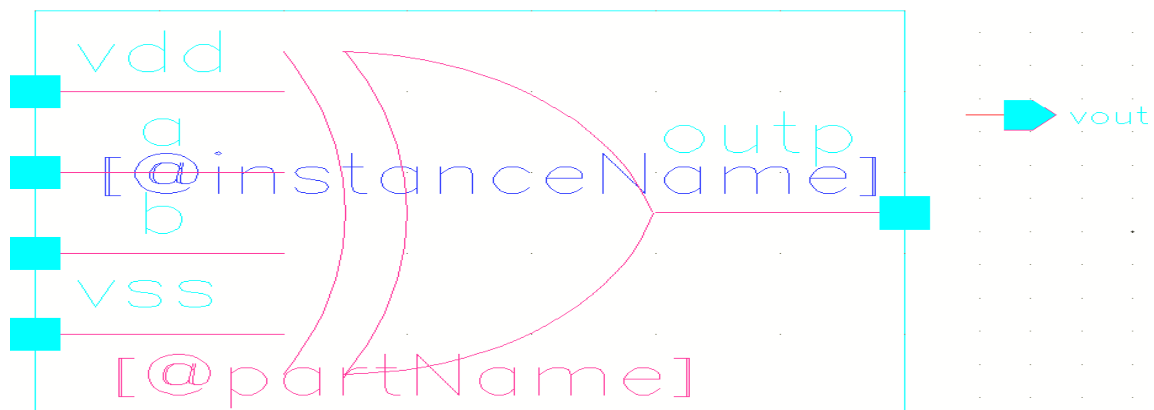



**PROCEDURE:**

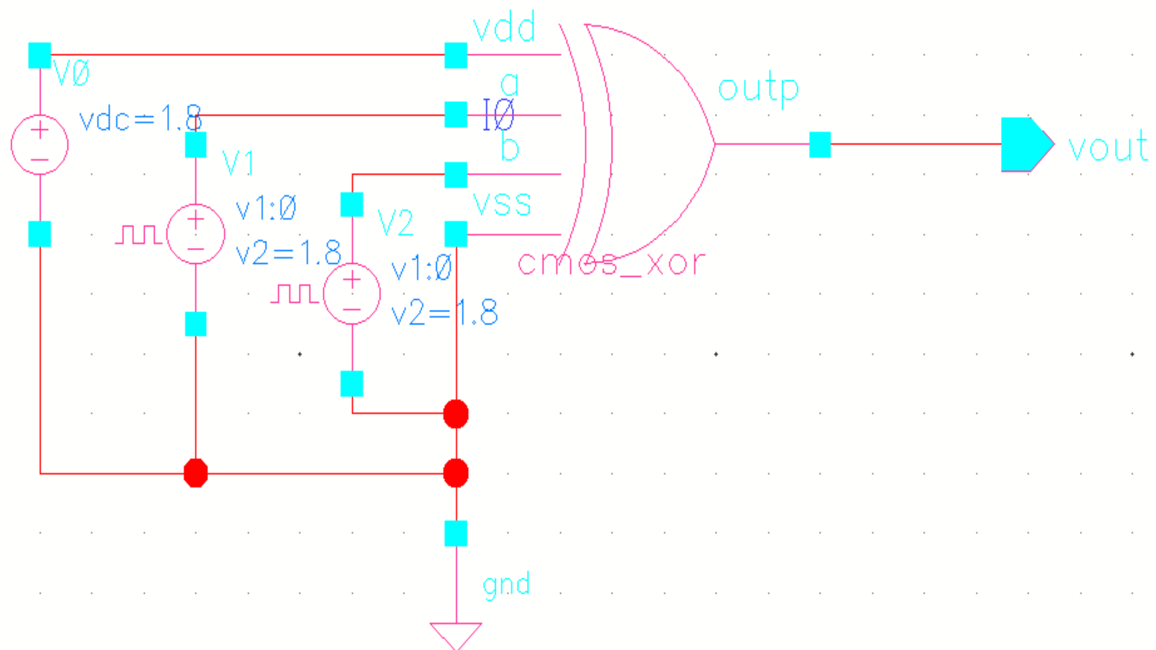
1. Connect the Circuit as shown in the circuit diagram using Pyxis schematic.
2. Create a simulation schematic for simulation.
3. Add necessary nets in outputs to view waveforms.
4. Run the Simulation and observe results in EZwave.
5. Draw the Layout for the circuit using Pyxis Layout.

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:**



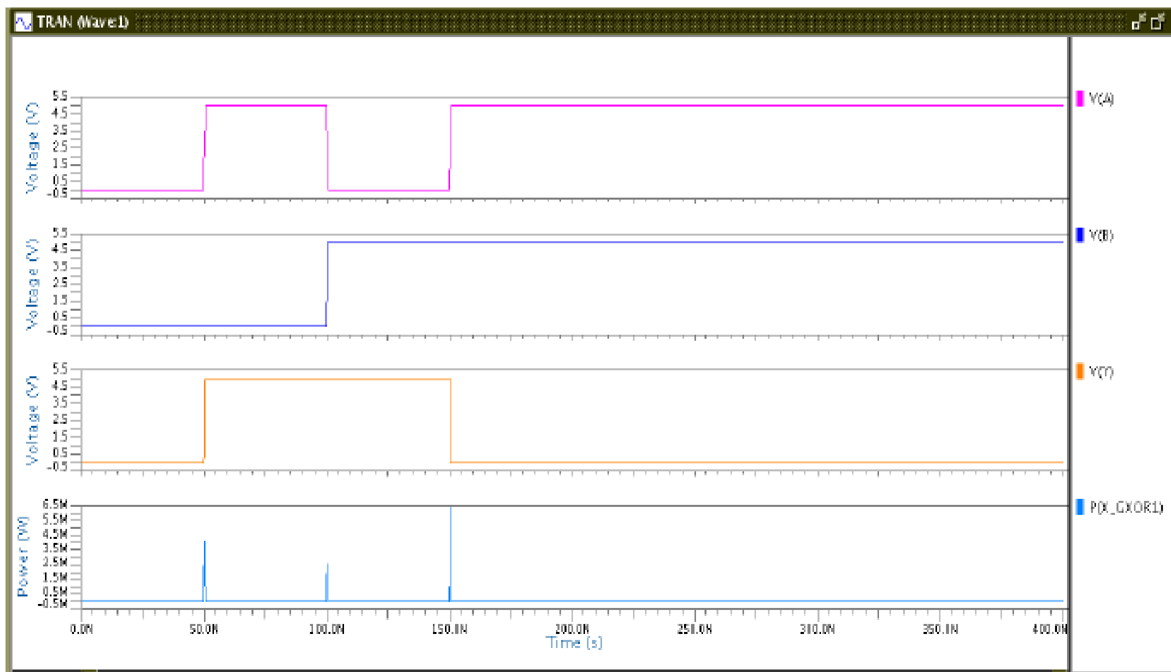

**Building the XOR Gate Test Design**



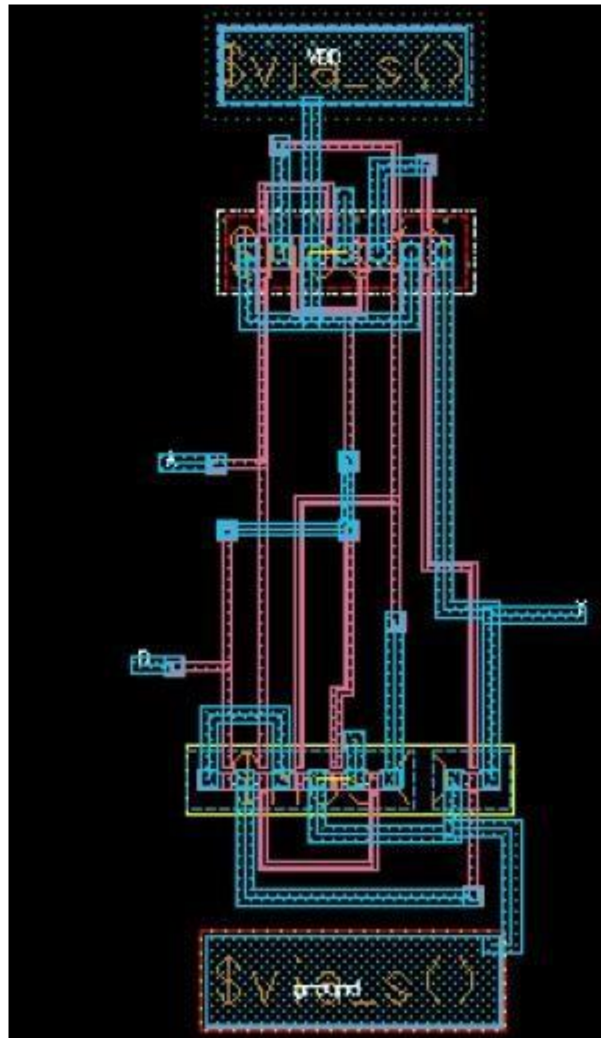
**PROCEDURE:**

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:**

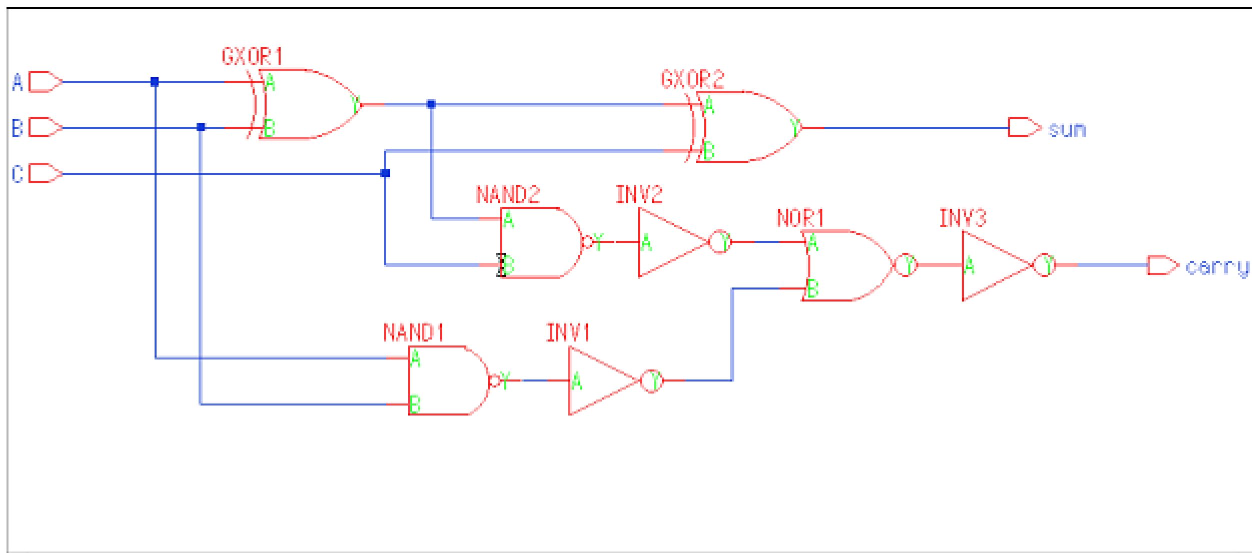


# EXPERIMENT: 5
# CMOS 1-BIT FULL ADDER

**AIM:** To design and simulate the CMOS 1 Bit Full Adder.

**TOOLS:** Pyxis Schematic, Pyxis Layout, Eldo, Ezwave, Calibre.

### 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.

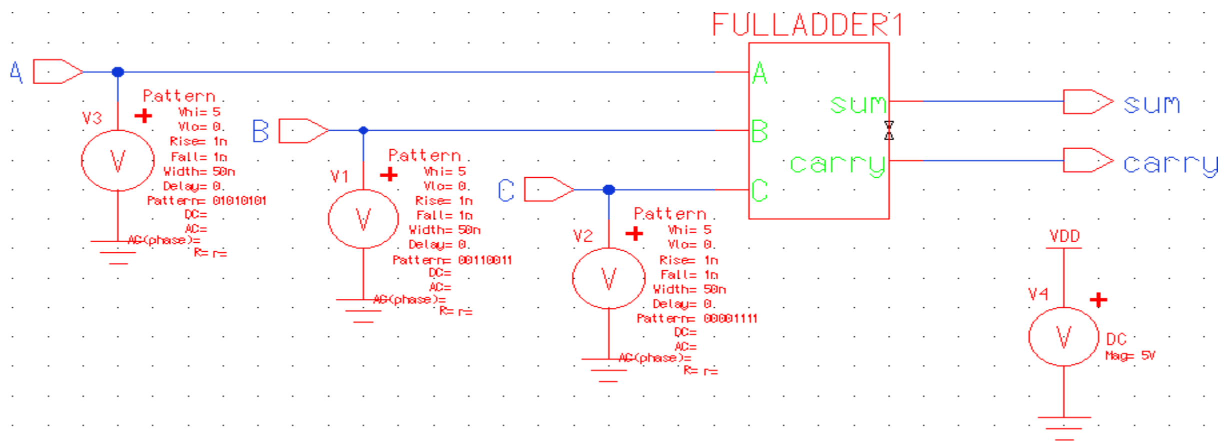
5. Draw the Layout for the circuit using Pyxis Layout.

7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .

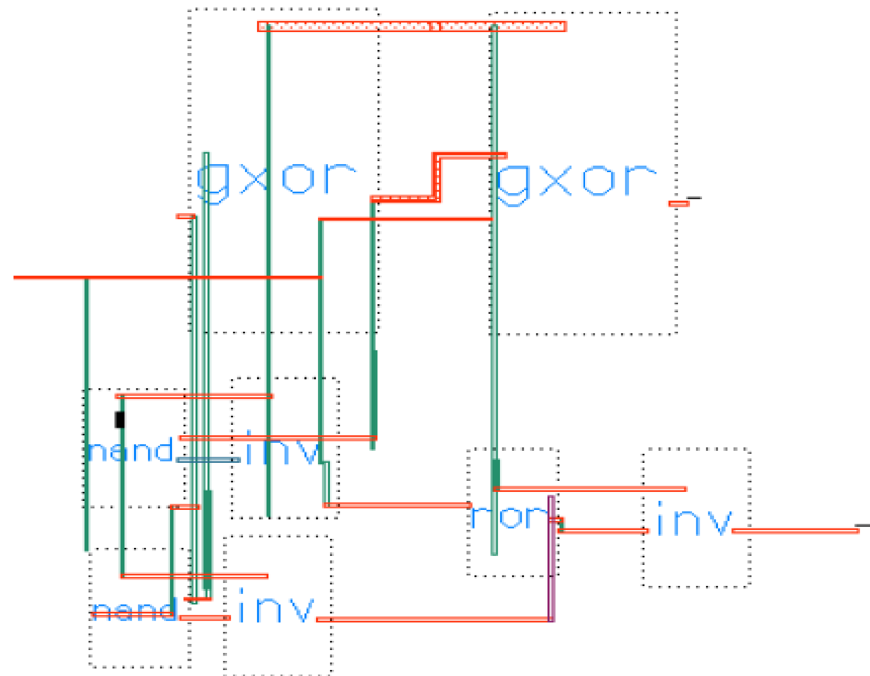
8. Run the post layout simulation by adding the .dspf file generated in PEX.

9. Observe the post layout results

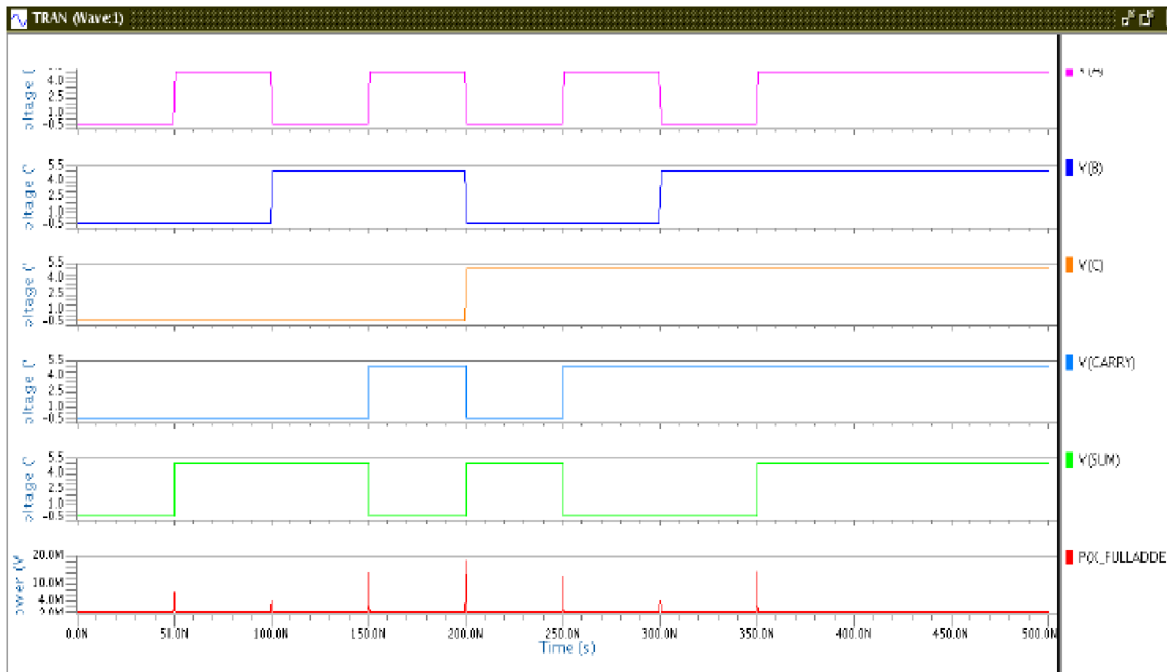
**Testing the Full Adder:**




**Layout:**



## Simulation Output:

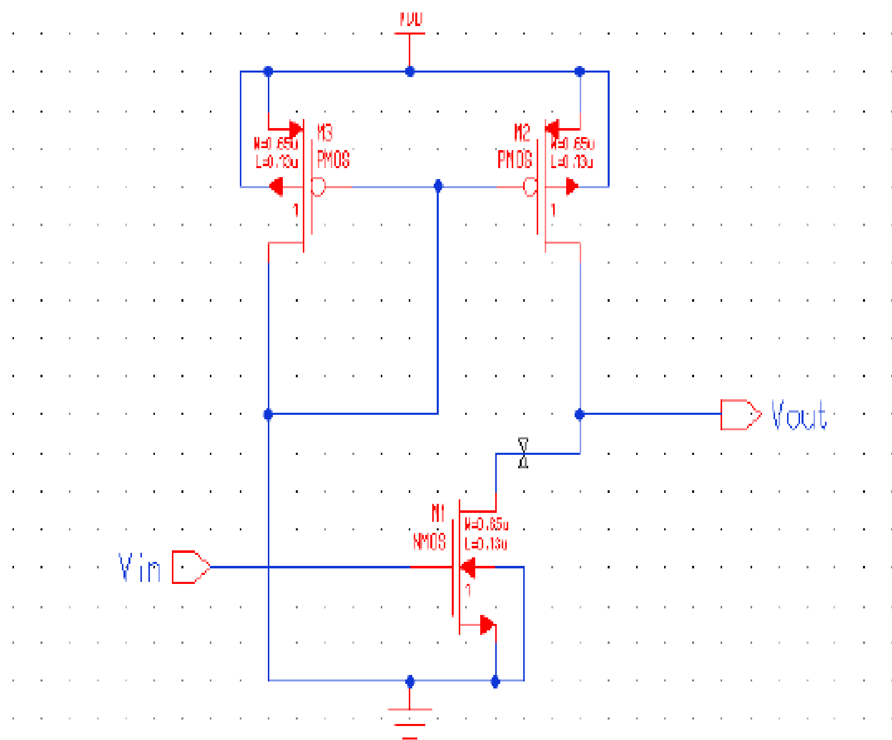


# EXPERIMENT: 6
# COMMON SOURCE AMPLIFIER

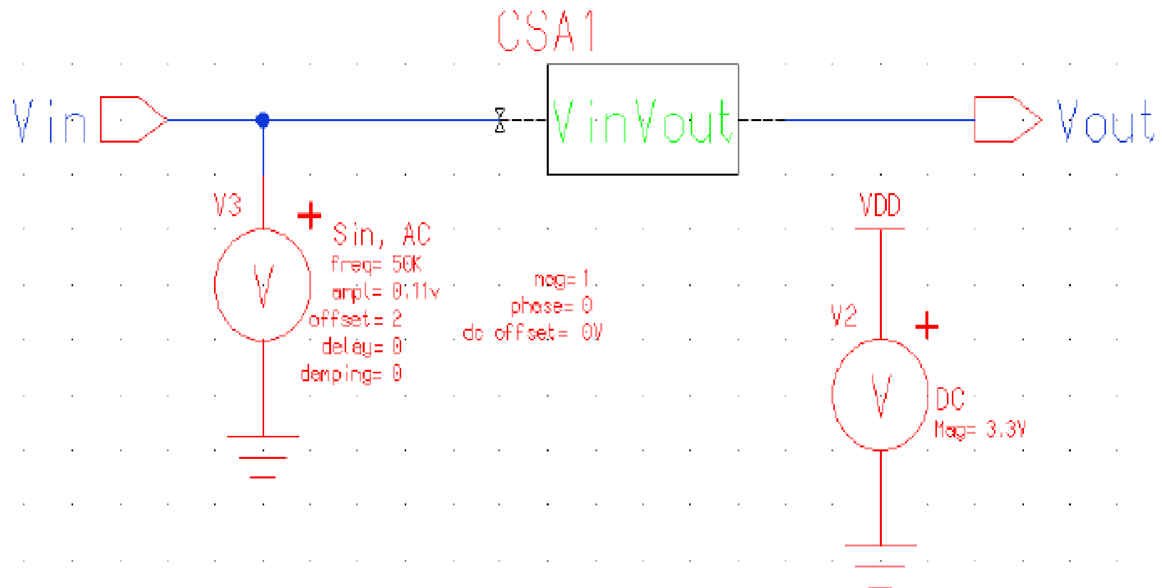
**AIM:** To design and simulate the Common Source Amplifier.

**TOOLS:** 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.

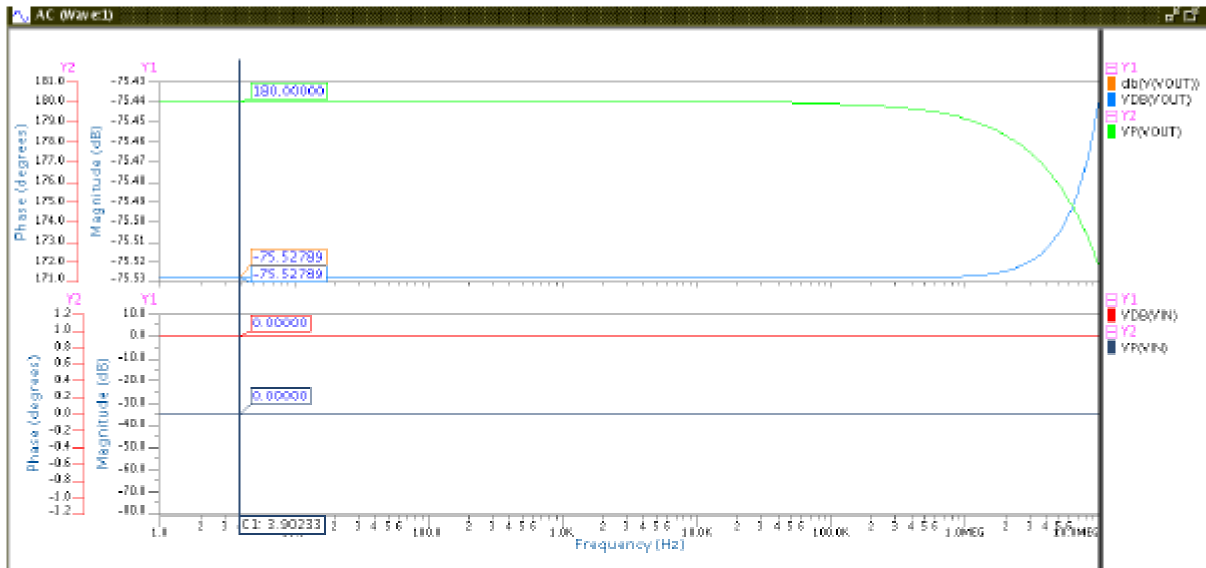
5. Draw the Layout for the circuit using Pyxis Layout.

7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .

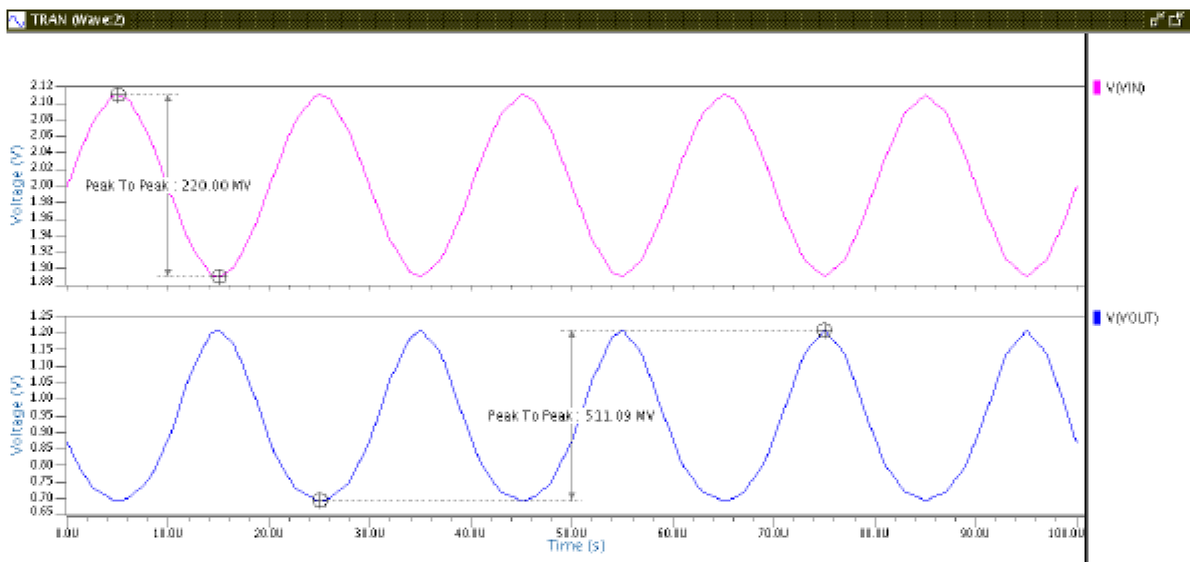
8. Run the post layout simulation by adding the .dspf file generated in PEX.

9. Observe the post layout results.

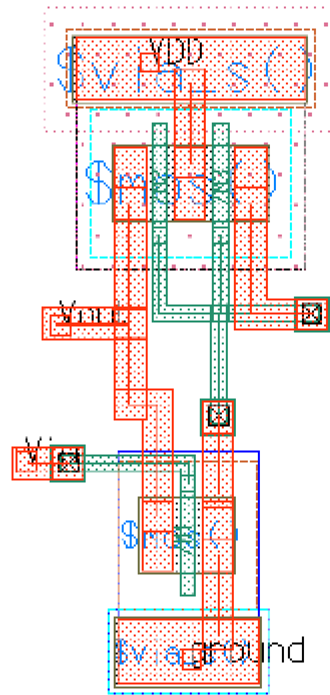
**AC Analysis:**




**Transient Analysis Result:**



**Layout:**

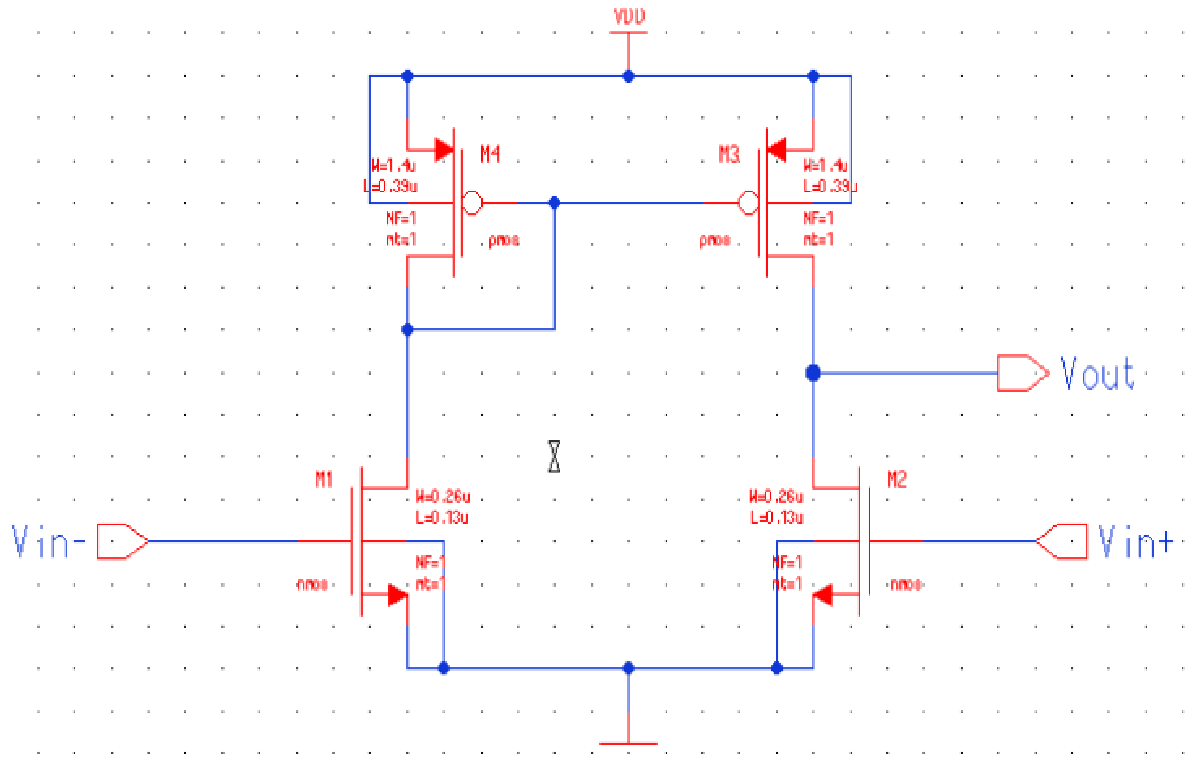


# EXPERIMENT: 7
# DIFFERENTIAL AMPLIFIER

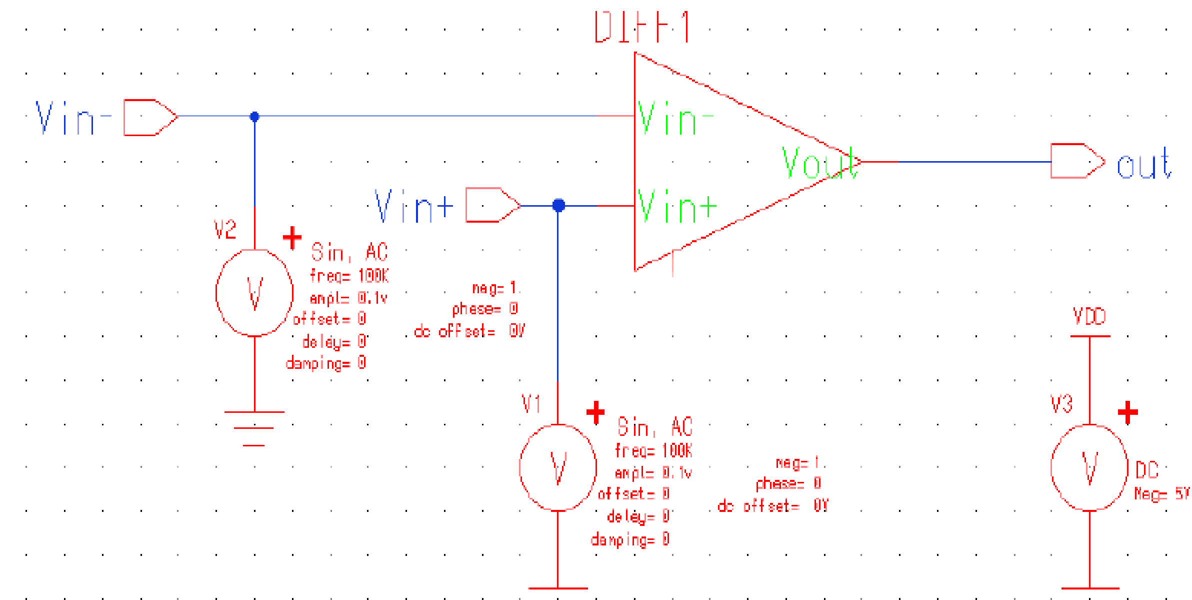
**AIM:** To design and simulate the Differential Amplifier.

**TOOLS:** 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.

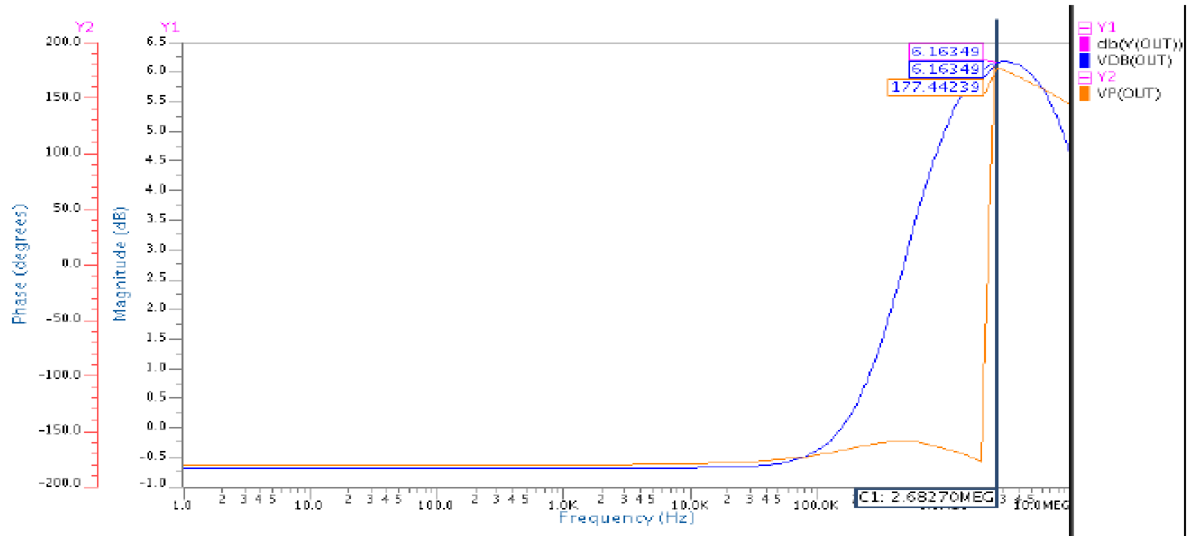
7. Run the physical verification (DRC, LVS, PEX) using Calibre tool .

8. Run the post layout simulation by adding the .dspf file generated in PEX.

9. Observe the post layout results.

**RESULTS:**

**AC Analysis result:**




**Transient Analysis**

**Result:**