

# **SRM VALLIAMMAI ENGINEERING COLLEGE**

**(An Autonomous Institution)**

**SRM NAGAR, KATTANKULATHUR – 603 203.**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**



## **LABORATORY MANUAL**

**EC3566 – VLSI LABORATORY**

**III YEAR- V SEMESTER**

**Regulation 2023**

**Academic Year 2026– 2027 (Odd semester)**

**Prepared by:**

**Dr. C. Saravanakumar, Associate Professor / ECE**

**Dr.N. Jothy, Assistant Professor (Sr.G) / ECE Mr.**

**K.R.Ganesh, Assistant Professor(Sr.G) / ECE**

## Syllabus

EC3566 VLSI LABORATORY

| L | T | P | C   |
|---|---|---|-----|
| 0 | 0 | 3 | 1.5 |

### OBJECTIVES:

- To learn Hardware Descriptive Language (Verilog/VHDL).
- To enable the students to efficiently analyze and design VLSI circuits in digital domain
- To apply the fundamental principles and design of VLSI circuit in analog domain
- To design Digital System using HDL and implement on FPGA
- To explore the integration of VLSI circuits with EDA platforms
- To analyze the result of logic and timing simulations and to use these simulation results to debug digital system

### LIST OF EXPERIMENTS:

#### **Simulation using HDL and implementation in FPGA**

1. Design an Adder and Multiplier (Minimum 8 Bit).
2. Design of basic combinational circuits.
3. Design of basic sequential circuits.
4. Design an Universal Shift Register.
5. Design of Memories.
6. Design Finite State Machine (Moore and Mealy).
7. Design 3-bit synchronous up/down counter.
8. Design 4-bit Asynchronous up/down counter.

#### **Simulation using circuit simulators**

9. Design CMOS Basic Gates & Flip-Flops. Generate Manual/Automatic Layout.
10. Design a 4-bit synchronous counter using Flip-Flops. Generate Manual/Automatic Layout
11. Design a CMOS Inverter
12. Design a simple 5 transistor differential amplifier.

## LIST OF EXPERIMENTS

### Simulation using HDL and implementation in FPGA

| Expt.No | Experiment                                                                            | Page |
|---------|---------------------------------------------------------------------------------------|------|
| 1.      | Design an Adder and Multiplier (Minimum 8 Bit).                                       | 4    |
| 2.      | Design of basic combinational circuits.                                               | 10   |
| 3.      | Design of basic sequential circuits.                                                  | 15   |
| 4.      | Design an Universal Shift Register.                                                   | 18   |
| 5.      | Design of Memories.                                                                   | 22   |
| 6.      | Design Finite State Machine (Moore and Mealy)                                         | 24   |
| 7.      | Design 3-bit synchronous up/down counter.                                             | 29   |
| 8       | Design 4-bit Asynchronous up/down counter                                             | 31   |
|         | <b>Simulation using circuit simulators</b>                                            |      |
| 9.      | Design CMOS Basic Gates & Flip-Flops. Generate Manual/Automatic Layout.               | 35   |
| 10.     | Design a CMOS Inverter                                                                | 44   |
| 11.     | Design a 4-bit synchronous counter using Flip-Flops. Generate Manual/Automatic Layout | 55   |
| 12.     | Design a simple 5 transistor differential amplifier                                   | 72   |
|         | <b>Content beyond Syllabus</b>                                                        |      |
| 13      | Design and Layout of a 6-Transistor (6T) CMOS SRAM Memory Cell Using Cadence Virtuoso | 85   |

## Expt No.1.A

### Design and Implementation of 8 Bit Adder Using HDL

#### Aim:

To design, simulate and implement 8 bit adder using Xilinx simulator and implement using Xilinx FPGA

#### Apparatus Required:

Windows 11 PC

Xilinx Vivado Simulator

Xilinx Zedboard

#### Procedure:

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

#### Program:

8 BIT ADDER

```
module adder(s,cout,a,b,cin);
output[7:0]s;
output cout;
input[7:0]a,b;
input cin;
wire c1,c2,c3,c4,c5,c6,c7;
fulladd fa0(s[0],c1,a[0],b[0],cin);
fulladd fa1(s[1],c2,a[1],b[1],c1);
fulladd fa2(s[2],c3,a[2],b[2],c2);
    fulladd fa3(s[3],c4,a[3],b[3],c3);
fulladd fa4(s[4],c5,a[4],b[4],c4);
fulladd fa5(s[5],c6,a[5],b[5],c5);
fulladd fa6(s[6],c7,a[6],b[6],c6);
fulladd fa7(s[7],cout,a[7],b[7],c7);
endmodule
```

```
module fulladd(s,cout,a,b,cin);
```

```

output s,cout;
input a,b,cin; xor

(s,a,b,cin);

assign cout = ((a & b)|(b& cin)|( a & cin)) ;

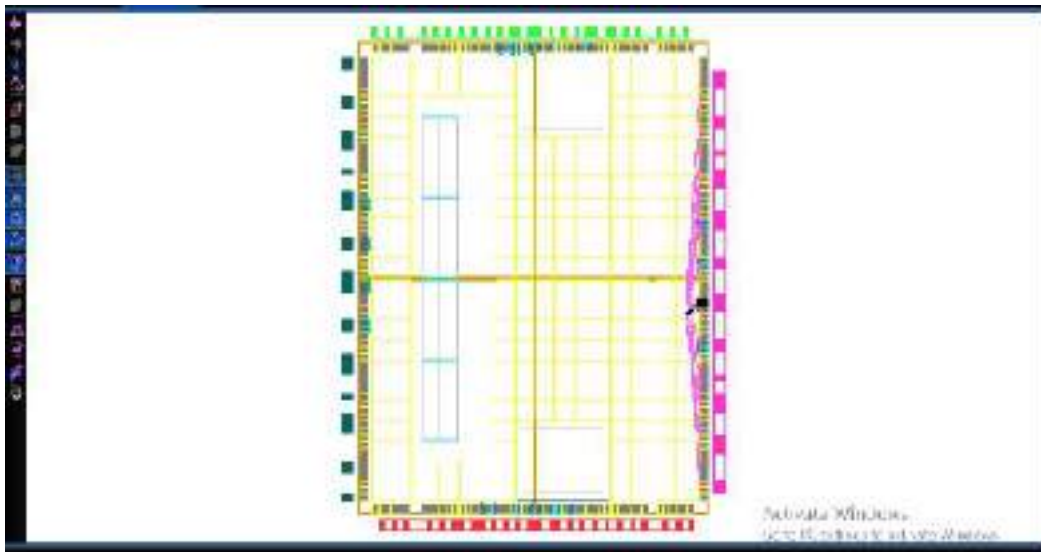
endmodule

```

### Simulation output:



### Floor Plan:



### Viva-Voce Questions

1. What is the difference between a Half Adder and a Full Adder?
2. Why is carry propagation delay important in multi-bit adders?
3. What is the difference between combinational and sequential multipliers?
4. What is the advantage of using Verilog/VHDL for arithmetic circuit design?
5. How is the design verified before implementation on an FPGA?

### Result:

Thus the program for 8-bit adder is simulated and implemented using Xilinx tools and outputs are verified successfully.

## Expt No.1.B

### Design and Implementation of 4 Bit Multiplier Using HDL

#### Aim:

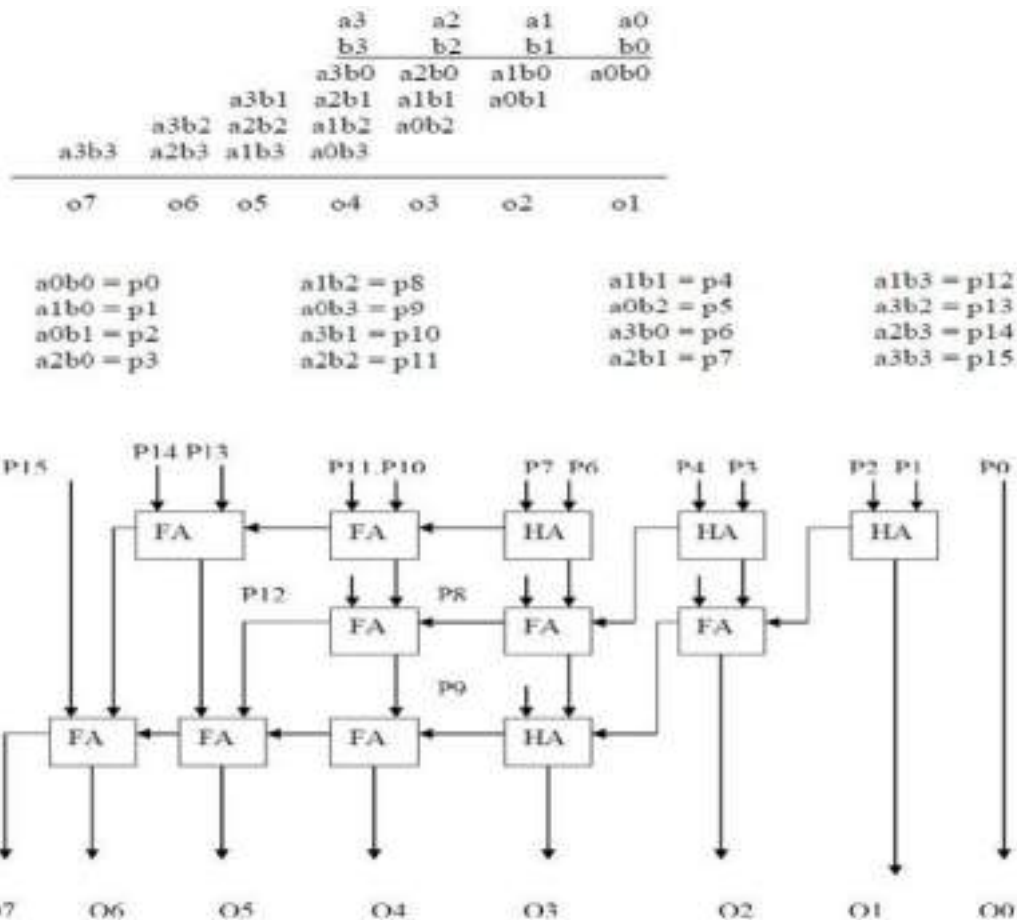
To design, simulate and implement 4 bit Multiplier using Xilinx simulator and implement using Spartan 3e FPGA.

#### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.



## Procedure:

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

## Program:

```
module fourbitmulti(m,a,b);
input[3:0]a;
input[3:0]b;
output[7:0]m;
wire[15:0]p;
wire[12:1]s;
wire[12:1]c;

and(p[0],a[0],b[0]);
and(p[1],a[1],b[0]);
and(p[2],a[0],b[1]);
and(p[3],a[2],b[0]);
and(p[4],a[1],b[1]);
and(p[5],a[0],b[2]);
and(p[6],a[3],b[0]);
and(p[7],a[2],b[1]);
and(p[8],a[1],b[2]);
and(p[9],a[0],b[3]);
and(p[10],a[3],b[1]);
and(p[11],a[2],b[2]);
and(p[12],a[1],b[3]);
and(p[13],a[3],b[2]);
and(p[14],a[2],b[3]);
and(p[15],a[3],b[3]);
```

```

half ha1(s[1],c[1],p[1],p[2]);
half ha2(s[2],c[2],p[4],p[3]);
half ha3(s[3],c[3],p[7],p[6]);
full fa4(s[4],c[4],p[11],p[10],c[3]);
full fa5(s[5],c[5],p[14],p[13],c[4]);
full fa6(s[6],c[6],p[5],s[2],c[1]);
full fa7(s[7],c[7],p[8],s[3],c[2]);
full fa8(s[8],c[8],p[12],s[4],c[7]);
full fa9(s[9],c[9],p[9],s[7],c[6]);
half ha10(s[10],c[10],s[8],c[9]);
full fa11(s[11],c[11],s[5],c[8],c[10]);
full fa12(s[12],c[12],p[15],s[5],c[11]);

```

```

buf (m[0],p[0]);
buf (m[1],s[1]);
buf (m[2],s[6]);
buf (m[3],s[9]);

```

```

buf (m[4],s[10]);
buf (m[5],s[11]);
buf (m[6],s[12]);
buf (m[7],c[12]);
endmodule

```

```

module half (s,c0,x,y);
input x,y;
output s,c0;
xor(s,x,y);
and(c0,x,y);
endmodule

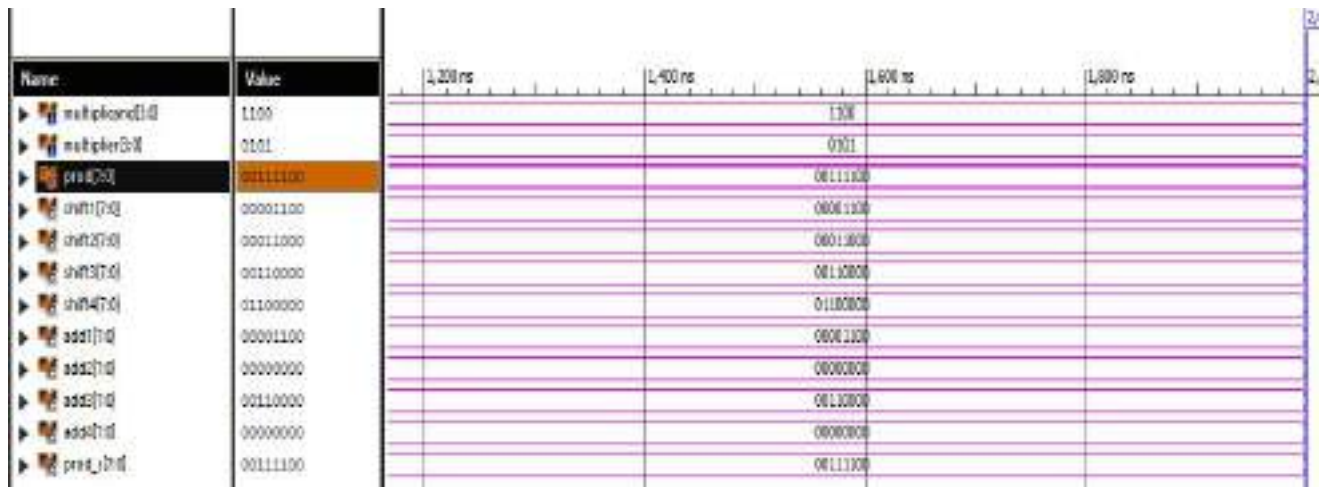
```

```

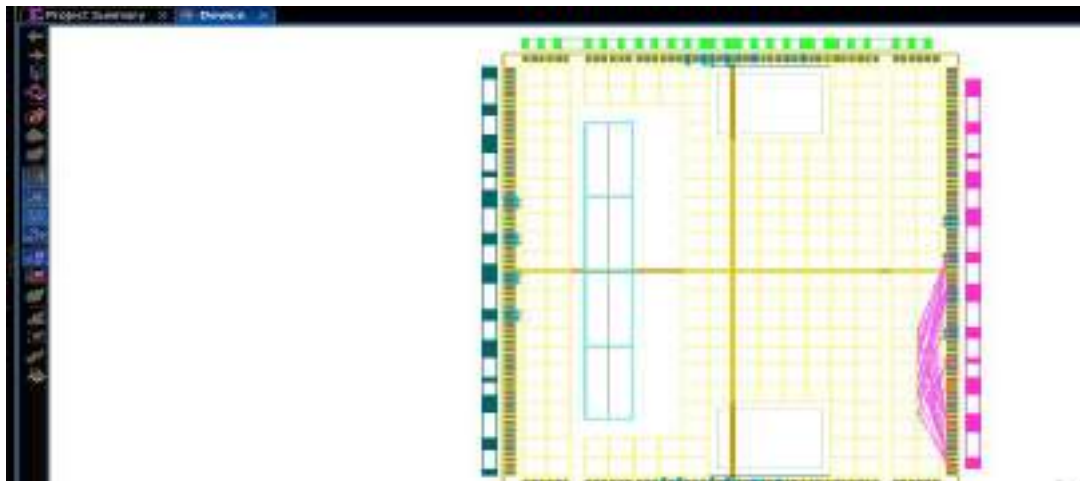
module full(s,c0,x,y,cin);
input x,y,cin;
output s,c0;
wire s1,d1,d2;
half ha_1(s1,d1,x,y);
half ha_2(s,d2,s1,cin);
or or_gate(c0,d2,d1);
endmodule

```

## Simulation Output:



## Floor Plan:



## Viva-Voce Questions

1. What is the difference between a Ripple Carry Adder (RCA) and a Carry Look-Ahead Adder (CLA)?
2. What is overflow in binary addition, and how is it detected?
3. Why is a multiplier circuit more complex than an adder circuit?
4. How does FPGA implementation differ from HDL simulation?
5. What factors affect the speed and performance of an 8-bit adder or multiplier circuit?

## Result:

Thus the program for 4 bit multiplier is simulated and implemented and outputs are verified successfully.

## Expt.No.2

### Design and Implementation of basic combinational circuits

Aim:

To design, simulate and implement basic combinational circuits like multiplexer, demultiplexer, encoder and decoder using Xilinx.

#### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

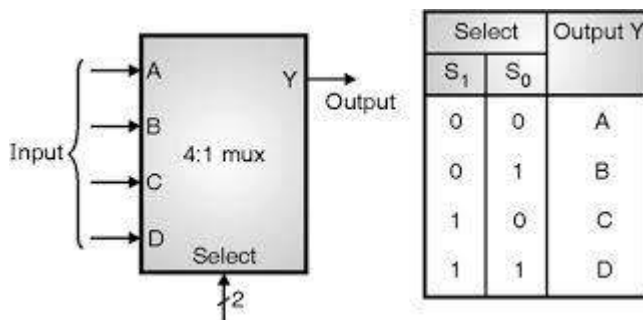
Xilinx Zedboard.

#### Procedure:

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

#### Program:

##### 4:1 Mux



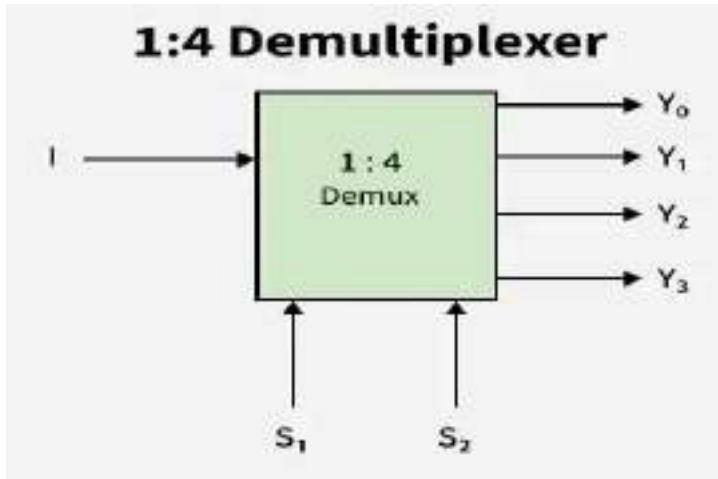
```
module mux4to1(input I0,input I1,input I2,input I3,input S1,input S0,output reg Y);  
always @(*)  
begin  
case({S1,S0})  
2'b00: Y = I0;  
2'b01: Y = I1;  
2'b10: Y = I2;  
2'b11: Y = I3;
```

```

default: Y = 1'b0;
endcase
end
endmodule

```

### 1:4 demux



```

module demux1to4(input D,input S1,input S0,output reg Y0,output reg Y1,output
reg Y2,output reg Y3);
always @(*)
begin
Y0 = 0;
Y1 = 0;
Y2 = 0;
Y3 = 0;
case({S1,S0})
2'b00: Y0 = D;
2'b01: Y1 = D;
2'b10: Y2 = D;
2'b11: Y3 = D;
endcase
end
endmodule

```

### 8:3 Encoder

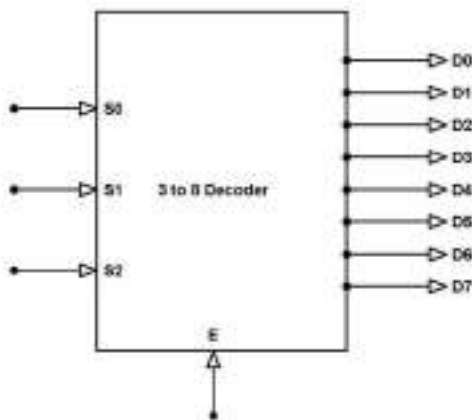


```

module encoder8to3( input [7:0] D,output reg [2:0] Y);
always @(*)
begin
case(D)
8'b00000001: Y = 3'b000;
8'b00000010: Y = 3'b001;
8'b00000100: Y = 3'b010;
8'b00001000: Y = 3'b011;
8'b00010000: Y = 3'b100;
8'b00100000: Y = 3'b101;
8'b01000000: Y = 3'b110;
8'b10000000: Y = 3'b111;
default: Y = 3'b000;
endcase
end
endmodule

```

### 3:8 decoder



```

module decoder3to8(input [2:0] A,output reg [7:0] Y);

```

```

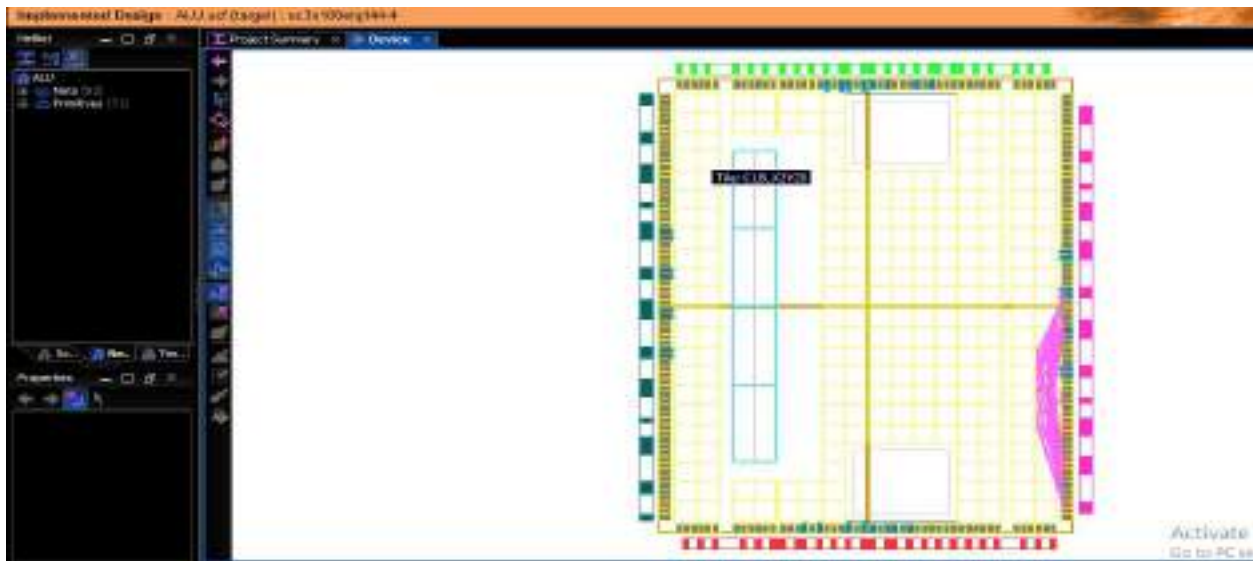
always @(*)
begin
case(A)
3'b000: Y = 8'b000000001;
3'b001: Y = 8'b000000010;
3'b010: Y = 8'b000000100;
3'b011: Y = 8'b000001000;
3'b100: Y = 8'b000010000;
3'b101: Y = 8'b000100000;
3'b110: Y = 8'b001000000;
3'b111: Y = 8'b010000000;
default: Y = 8'b000000000;
endcase
end
endmodule

```

### Simulation Output:



## Floor Plan:



## Viva-Voce Questions

1. What are combinational circuits?
2. Differentiate between an Encoder and a Decoder.
3. What is the purpose of a Multiplexer?
4. What is the difference between SOP and POS expressions?
5. Why are combinational circuits free from memory elements?

## Result

Thus the basic combinational circuits are designed, simulated and implemented using Xilinx tools and outputs are verified successfully.

## Expt No.3 Design and Implementation of Basic Sequential Circuits

### Aim:

To design, simulate and implement basic sequential circuits like D flip flop, JK flip flop, and SR flip flop using Xilinx.

### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

### Procedure:

Start the Xilinx project Navigator.

Create a new project and new source.

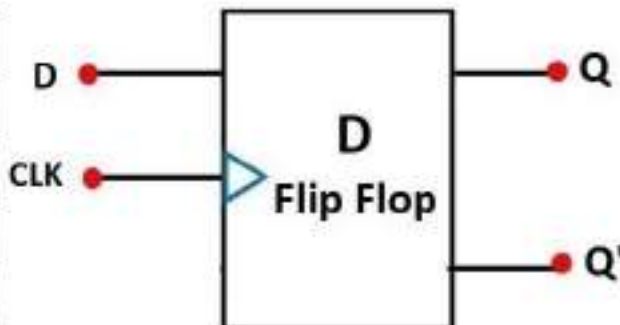
Write a Verilog code and synthesize and simulate the coding using ISE Simulator.

Assign the package pins for input and output using XILINX plan ahead.

Implement and verify the output in Spartan-3 FPGA kit.

**Truth Table of DFlip Flop**

| D | CLK | Q | Q' |
|---|-----|---|----|
| 0 | 0   | 0 | 1  |
| 0 | 1   | 0 | 1  |
| 1 | 0   | 0 | 1  |
| 1 | 1   | 1 | 0  |



```
module d_flipflop(input clk,input D,output reg Q);
```

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

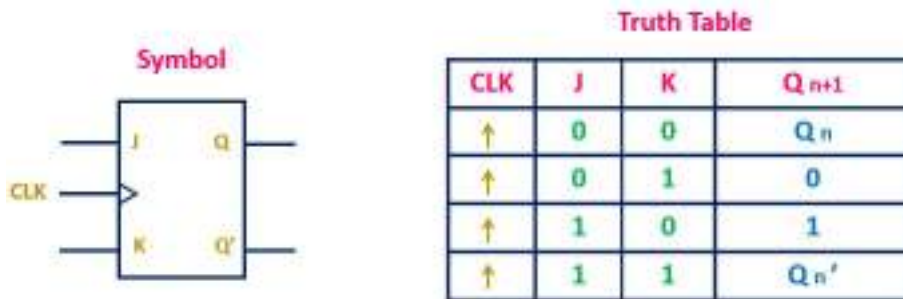
```
begin
```

```
Q <= D;
```

```
end
```

```
endmodule
```

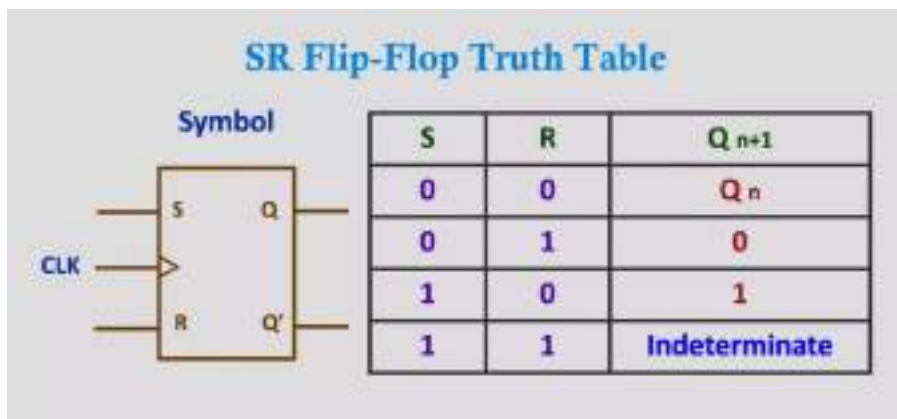
## JK flip flop



```

module jk_flipflop( input clk,input J,input K,output reg Q);
always @(posedge clk)
begin
case({J,K})
2'b00: Q <= Q;
2'b01: Q <= 1'b0;
2'b10: Q <= 1'b1;
2'b11: Q <= ~Q;
endcase
end
endmodule
SR FLIP FLOP

```



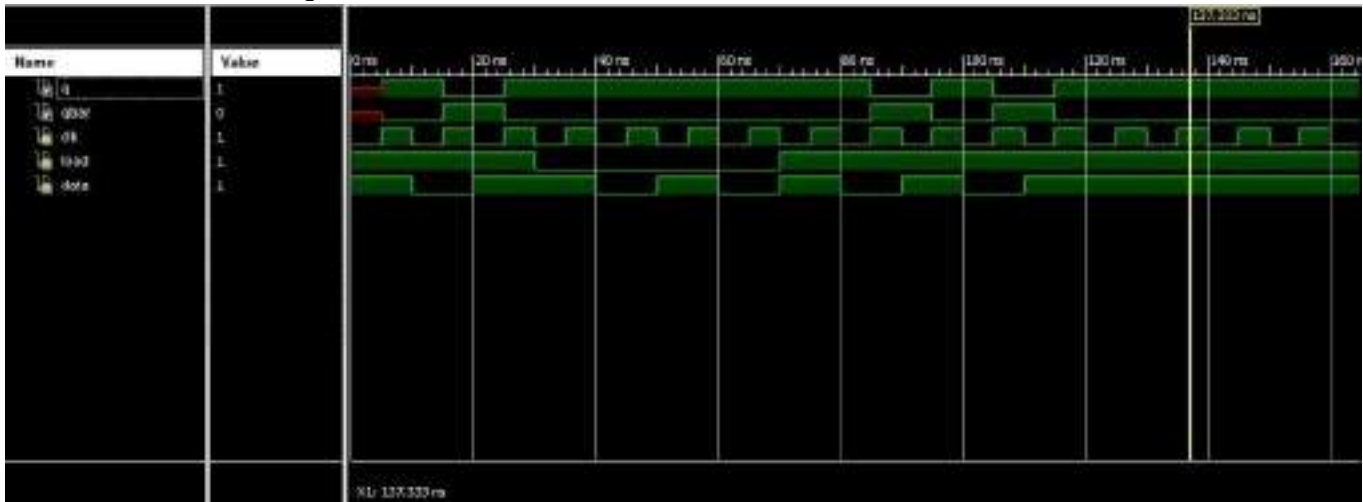
```

module sr_flipflop(input clk,input S,input R,output reg Q);

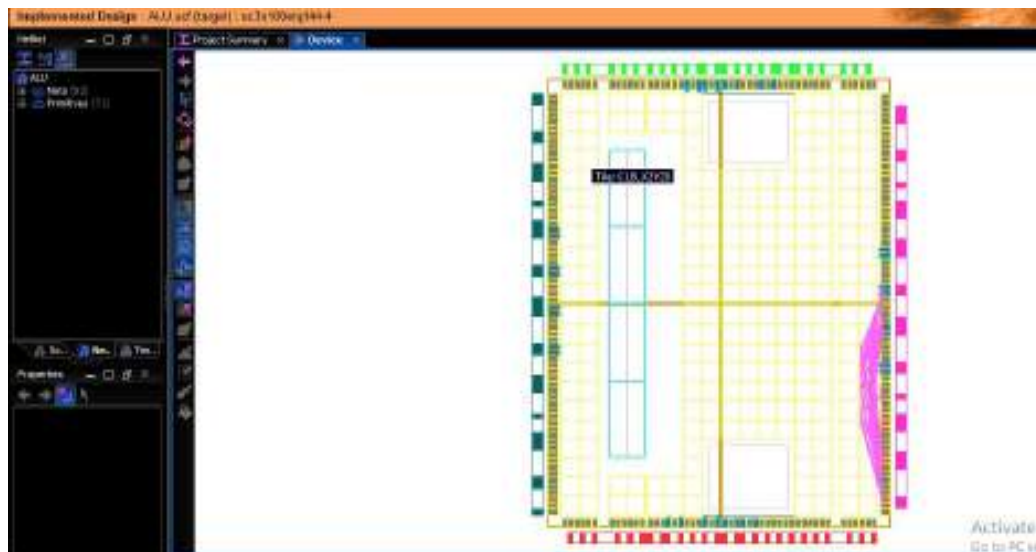
always @(posedge clk)
begin
case({S,R})
2'b00: Q <= Q;
2'b01: Q <= 1'b0;
2'b10: Q <= 1'b1;
2'b11: Q <= 1'bx;
endcase
end
end

```

### *simulation output*



### Floor Plan:



### Viva-Voce Questions

1. What is the difference between combinational and sequential circuits?
2. What is the function of a Flip-Flop?
3. Differentiate between synchronous and asynchronous reset.
4. What is setup time and hold time?
5. Why is a clock signal required in sequential circuits?

### Result

Thus the program for basic sequential circuits like D flip flop, JK flip flop and SR flip flop using Xilinx tools are simulated, implemented and outputs are verified successfully.

## Expt No. 4

### Design and Implementation of Universal Shift Register

#### Aim:

To design, simulate and implement universal shift register using Xilinx.

#### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

#### Procedure:

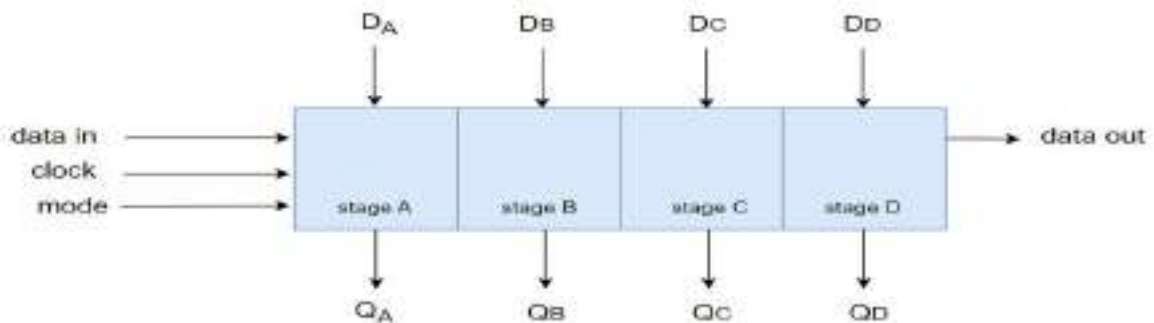
Start the Xilinx project Navigator.

Create a new project and new source.

Write a Verilog code and synthesize and simulate the coding using ISE Simulator.

Assign the package pins for input and output using XILINX plan ahead.

Implement and verify the output in Spartan-3 FPGA kit.



#### Program:

```
module universal_shift(a,s,clk,p);  
    input [3:0]a;  
    input [1:0]s;  
    input clk;  
    output reg [3:0]p;  
    initial  
        p<=4'b0110;  
    always@(posedge clk)  
    begin  
        case (s)  

```

```

        2'b00:
        begin
            p[3]<=p[3];
        p[2]<=p[2];
        p[1]<=p[1];
        p[0]<=p[0];

        end

        2'b01:
        begin
            p[3]<=p[0];
            p[2]<=p[3];
            p[1]<=p[2];
            p[0]<=p[1];
        end

        2'b10:
        begin
            p[0]<=p[3];
            p[1]<=p[0];
            p[2]<=p[1];
            p[3]<=p[2];
        end

        2'b11:
        begin
            p[0]<=a[0];
            p[1]<=a[1];
            p[2]<=a[2];
            p[3]<=a[3];
        end

    endcase
end
endmodule

```



## **Viva-Voce Questions**

1. What is a Universal Shift Register?
2. What are the different modes of operation of a Universal Shift Register?
3. What is the difference between serial and parallel data transfer?
4. Why are shift registers used in digital systems?
5. Name some practical applications of shift registers.

### **Result:**

Thus the program to simulate Universal Shift Register using Xilinx tools is executed and output is verified successfully.

## Expt.No. 5

## Design of Memories

### Aim:

To design, simulate and implement single port RAM using Xilinx simulator and implement using Xilinx FPGA.

### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

### Procedure:

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

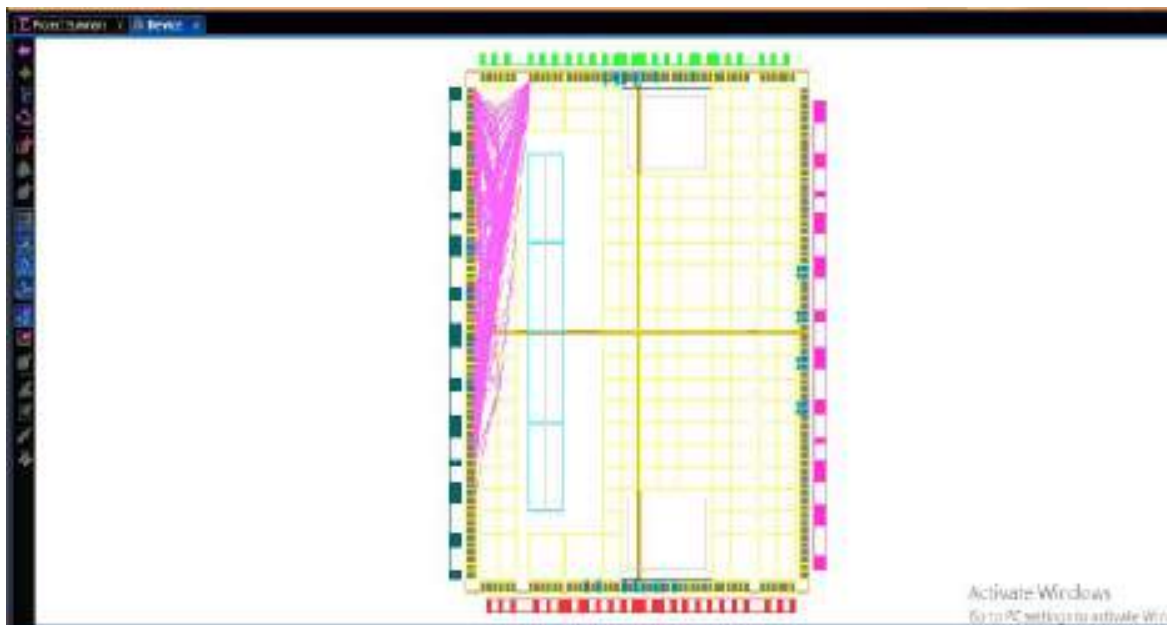
### Program:

```
module single_port_ram (input [7:0] data,input [5:0] addr, input we, clk, output [7:0]q);
    reg [7:0] ram[63:0];
    reg [5:0] addr_reg;
    always @ (posedge clk)
        begin
            if (we)
                ram[addr] <=
                    data; addr_reg
                    <= addr;
            end
            assign q = ram[addr_reg];
        endmodule
```

### Simulation Output:

[illegible]

### Floor Plan:



## Viva-Voce Questions

1. What is the difference between RAM and ROM?
2. Differentiate between SRAM and DRAM.
3. What is the function of address lines in a memory?
4. Why is SRAM faster than DRAM?
5. What are the applications of semiconductor memories?

**Result:** Thus the program to design memory (single port RAM) using Xilinx tools is executed and output is verified successfully.

## Expt. No. 6 Design of Finite State Machine (Moore and Mealy) using HDL

### Aim:

To design, simulate and implement Finite State Machine using Xilinx tool.

### Apparatus Required:

Windows 11 PC.

Xilinx Vivado

Simulator. Xilinx

Zedboard.

### Procedure:

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.



Fig. 1: Moore State Machine

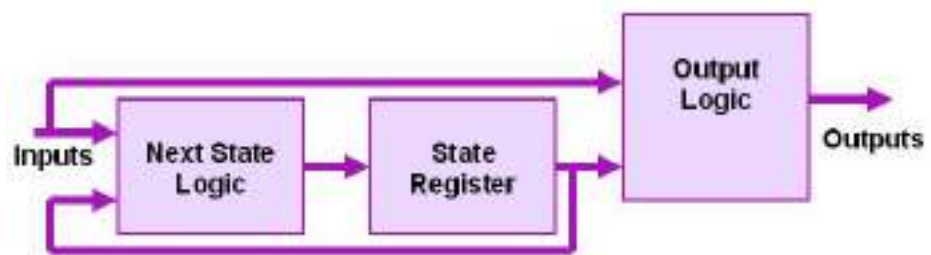
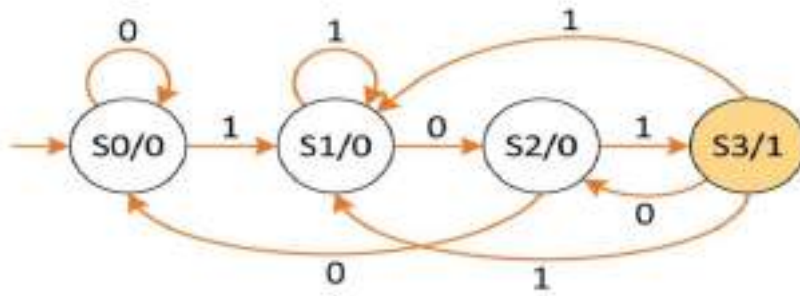


Fig. 2: Mealy State Machine

### Program:

## 1. Moore FSM (Sequence Detector for "101")

Moore Machine State Diagram for Detecting '101'



A Moore 101 detector needs four states; the output 1 belongs to state S3, not to any transition.

```
module moore_fsm(input clk,input reset,input x,output reg y);
```

```
reg [1:0] state, next_state;
```

```
// State Encoding
```

```
parameter S0 = 2'b00,
```

```
S1 = 2'b01,
```

```
S2 = 2'b10,
```

```
S3 = 2'b11;
```

```
// State Register
```

```
always @(posedge clk or posedge reset)
```

```
begin
```

```
if(reset)
```

```
state <= S0;
```

```
else
```

```
state <= next_state;
```

```
end
```

```
// Next State Logic
```

```
always @(*)
```

```
begin
```

```
case(state)
```

```
S0:
```

```
if(x)
```

```
next_state = S1;
```

```
else
```

```
next_state = S0;
```

```
S1:
```

```
if(x)
```

```
next_state = S1;
```

```
else
```

```
next_state = S2;
```

```
S2:
```

```
if(x)
```

```
next_state = S3;
```

```
else
```

```
next_state = S0;
```

```
S3:
```

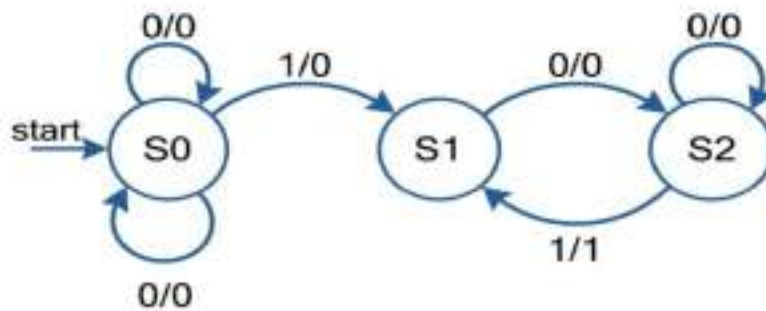
```

if(x)
next_state = S1;
else
next_state = S2;

default:
next_state = S0;
endcase
end
// Output Logic (Moore)
always @(*)
begin
case(state)
S3: y = 1'b1;
default: y = 1'b0;
endcase
end
endmodule

```

## 2. Mealy FSM (Sequence Detector for "101")



A Mealy 101 detector needs only three states; the output 1 appears on the S2 to S1 transition on input 1.

### Verilog Code (Mealy FSM)

```

module mealy_fsm(input clk,input reset,input x,output reg y);

reg [1:0] state, next_state;

// State Encoding
parameter S0 = 2'b00,
S1 = 2'b01,
S2 = 2'b10;

// State Register
always @(posedge clk or posedge reset)
begin
if(reset)
state <= S0;

```

```

else
state <= next_state;
end

// Next State Logic and Output
always @(*)
begin

y = 1'b0;

case(state)

S0:
begin
if(x)
next_state = S1;
else
next_state = S0;
end

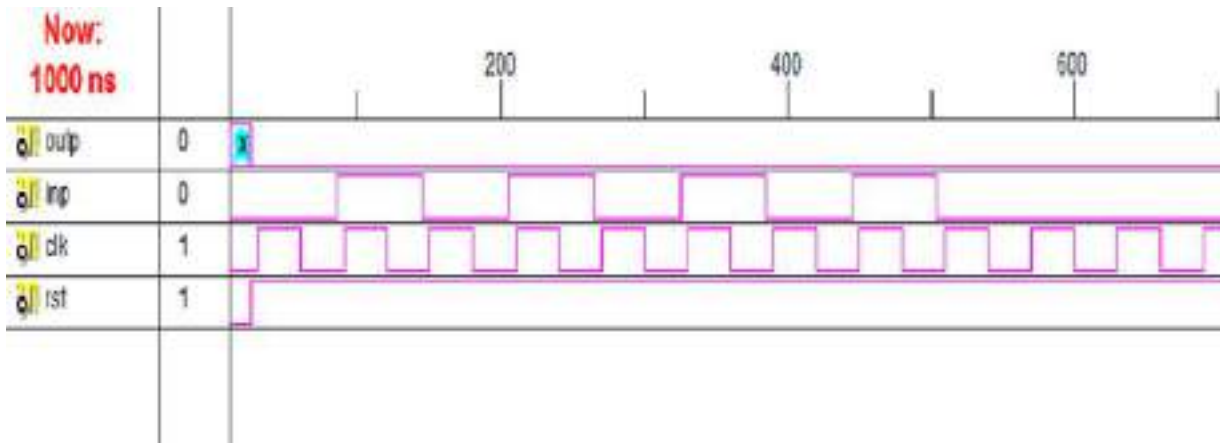
S1:
begin
if(x)
next_state = S1;
else
next_state = S2;
end

S2:
begin
if(x)
begin
next_state = S1;
y = 1'b1;
end
else
next_state = S0;
end

default:
begin
next_state = S0;
y = 1'b0;
end
endcase
end
endmodule

```

**Simulation Output:**



## Viva-Voce Questions

1. What is a Finite State Machine (FSM)?
2. What is the difference between Moore and Mealy machines?
3. Which FSM generally requires fewer states?
4. What is state encoding?
5. Where are FSMs commonly used?

## Result:

Thus the program to simulate Finite State Machine (Moore and mealy machine) using Xilinx tools is executed and output is verified successfully.

## Expt. No. 7     Design of 3-bit Synchronous Up/Down Counter

### Aim:

To design, simulate and implement 3-bit Synchronous Up/Down Counter using Xilinx simulator and implement using Xilinx FPGA.

### Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

### Procedure:

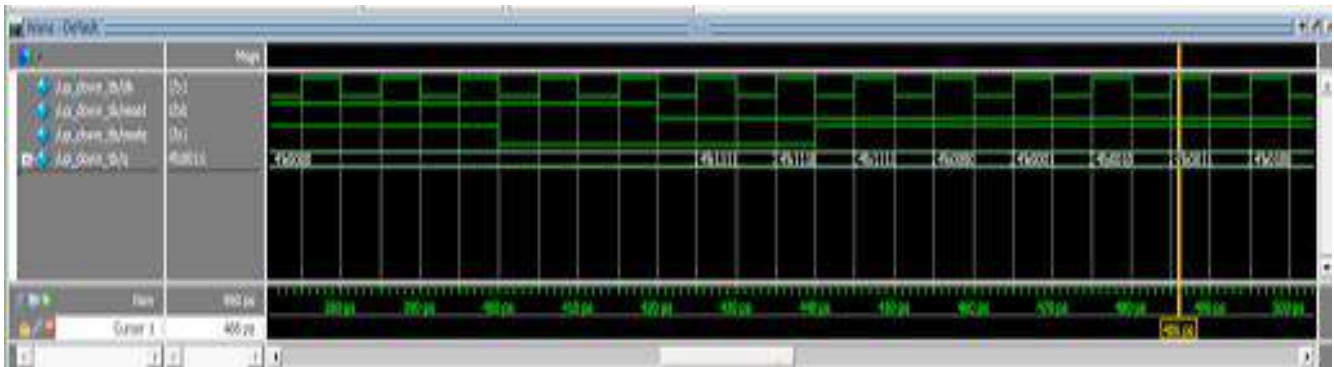
- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

### Program:

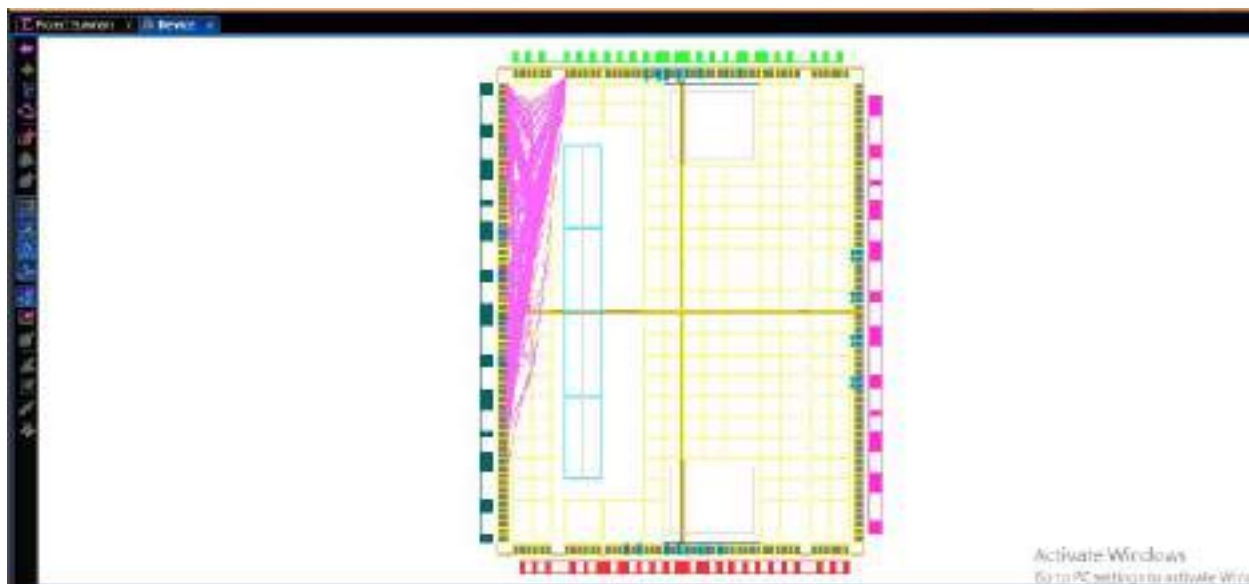
```
module up_down_counter (input clk,input reset,input mode, output reg [2:0] count);
always @(posedge clk)
begin
if (reset)
count <= 3'b000;
else
begin
if (mode)
count <= count + 1'b1; // Up Count
else
count <= count - 1'b1; // Down Count
end
end

endmodule
```

—:



## Floor Plan:



## Viva-Voce Questions

1. What is a synchronous counter?
2. How does an Up counter differ from a Down counter?
3. Why do synchronous counters have less propagation delay?
4. What is the modulus of a 3-bit counter?
5. Name one practical application of counters.

**Result:** Thus the program to design 3-bit Synchronous Up/Down Counter using Xilinx tools is executed and output is verified successfully.

## **Expt.no .8    Design 4-bit Asynchronous up/down counter**

### **Aim:**

To design, simulate and implement 4-bit Asynchronous up/down counter using Xilinx simulator and implement using Xilinx FPGA.

### **Apparatus Required:**

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

### **Procedure:**

- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

### **Program**

```
// -----  
  
// Design: 4-bit Asynchronous (Ripple) Up/Down Counter  
// Clocking scheme: Positive-edge triggered T flip-flops  
// Up mode (up_down = 1): Next stage clock = ~Q[prev]  
// Down mode (up_down = 0): Next stage clock = Q[prev]  
// -----  
  
// Toggle Flip-Flop with asynchronous active-high reset  
module T_FF (  
    output reg Q,  
    input T,  
    input clk,  
    input reset  
);  
  
always @(posedge clk or posedge reset) begin
```

```

    if(reset)
        Q <= 1'b0;
    else if(T)
        Q <= ~Q;
    end
endmodule

// Top-level: 4-bit Asynchronous Up/Down Counter
module async_up_down_counter_4bit (
    output [3:0] count,
    input    clk,    // Clock for the LSB stage
    input    up_down, // 1 = up, 0 = down
    input    reset   // Asynchronous reset, active high
);
    wire [3:0] q;    // Flip-flop outputs
    wire [3:0] t = 4'b1111; // All toggle inputs tied high (always toggle)
    wire [2:0] stage_clk; // Derived clocks for stages 1..3
    // LSB stage (bit 0) toggles directly on the external clock
    T_FF ff0 (
        .Q(q[0]),
        .T(t[0]),
        .clk(clk),
        .reset(reset)
    );
    // Generate clock for stage 1 (bit 1)
    assign stage_clk[0] = up_down ? ~q[0] : q[0];
    T_FF ff1 (
        .Q(q[1]),
        .T(t[1]),
        .clk(stage_clk[0]),
        .reset(reset)
    );
    // Generate clock for stage 2 (bit 2)
    assign stage_clk[1] = up_down ? ~q[1] : q[1];

```

```

T_FF ff2 (
    .Q(q[2]),
    .T(t[2]),
    .clk(stage_clk[1]),
    .reset(reset)
);

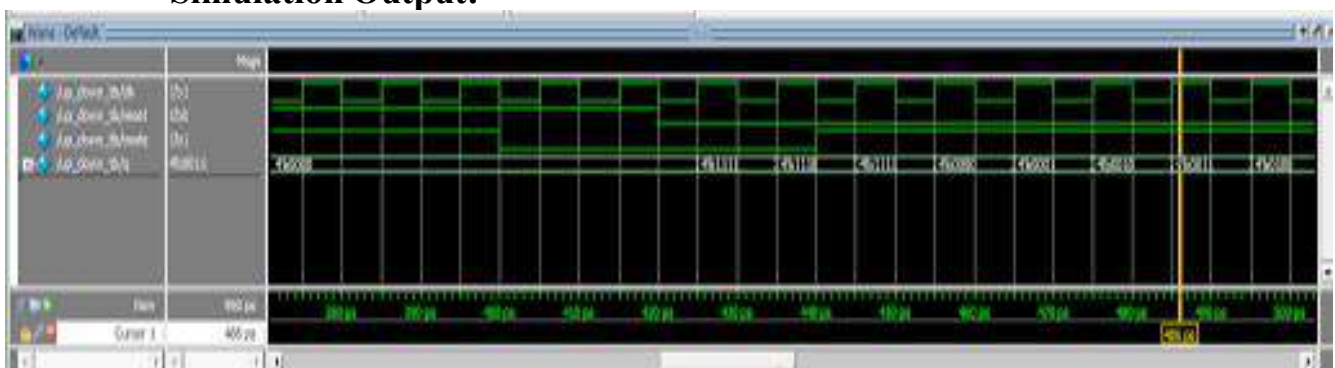
// Generate clock for stage 3 (bit 3)
assign stage_clk[2] = up_down ? ~q[2] : q[2];

T_FF ff3 (
    .Q(q[3]),
    .T(t[3]),
    .clk(stage_clk[2]),
    .reset(reset)
);

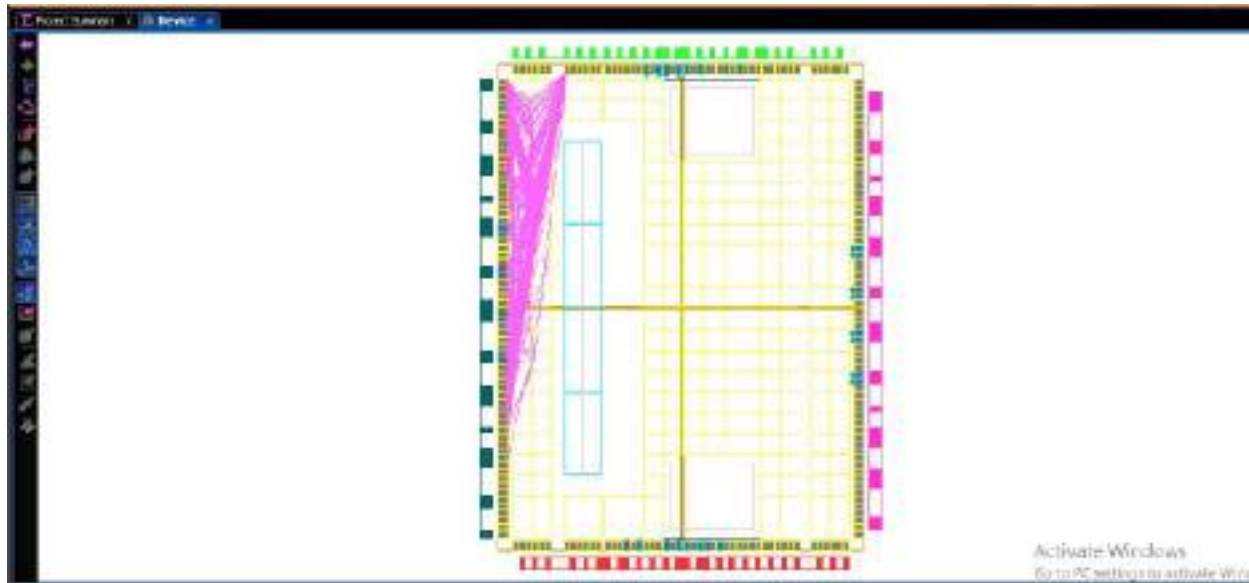
// Output assignment
assign count = q;
endmodule

```

### Simulation Output:



### Floor Plan:



## Viva-Voce Questions

1. What is an asynchronous (Ripple) counter?
2. Why is propagation delay higher in ripple counters?
3. What is the counting sequence of a 4-bit Up counter?
4. Differentiate between synchronous and asynchronous counters.
5. Mention one advantage and one disadvantage of ripple counters.

**Result:** Thus the program to design 4-bit Asynchronous Up/Down Counter using Xilinx tools is executed and output is verified successfully.

## Layout Generation Experiments

### Expt.No. 9 Design of CMOS Basic Gates & Flip-Flops and Generate Manual/Automatic Layout Using Cadence Virtuoso

#### Aim

To design CMOS basic logic gates (Inverter, NAND, NOR) and a CMOS D Flip-Flop using **Cadence Virtuoso**, simulate their functionality using the **Spectre** simulator, generate **manual and connectivity-driven (automatic-assisted) layouts** using **Virtuoso Layout Suite XL**, and verify the layouts through **DRC, LVS, parasitic extraction, and post-layout simulation**.

#### Software Required

- Cadence Virtuoso IC Design Platform
- Virtuoso Schematic Editor
- Virtuoso Layout Suite XL
- ADE Explorer / ADE Assembler
- Spectre Circuit Simulator
- Assura DRC/LVS or Pegasus DRC/LVS (depending on installation)
- Quantus (QRC) Parasitic Extraction (if licensed)
- Process Design Kit (PDK)
  - GPD180 / GPD90 / 45 nm PDK / Foundry PDK (as available)

#### Hardware Required

- Linux Workstation (Recommended)
- Intel Core i5/i7 Processor
- Minimum 8 GB RAM
- Cadence License Server

#### Theory

##### CMOS Logic Gates

CMOS (Complementary Metal Oxide Semiconductor) technology combines PMOS and NMOS transistors to realize digital logic circuits with very low static power consumption and high noise immunity.

##### CMOS Inverter

The CMOS inverter consists of:

- One PMOS transistor
- One NMOS transistor

Operation:

- Input LOW → PMOS ON, NMOS OFF → Output HIGH.
- Input HIGH → PMOS OFF, NMOS ON → Output LOW.

### CMOS NAND Gate

The CMOS NAND gate consists of:

- PMOS transistors connected in parallel (Pull-Up Network)
- NMOS transistors connected in series (Pull-Down Network)

### CMOS NOR Gate

The CMOS NOR gate consists of:

- PMOS transistors connected in series
- NMOS transistors connected in parallel

### CMOS D Flip-Flop

The D Flip-Flop is a sequential circuit that stores one bit of information. On the active clock edge, the output Q follows the input D. In Cadence, it may be built from transmission gates and inverters or instantiated from the standard-cell library provided by the PDK.

#### Truth Tables

##### Inverter

| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

##### NAND Gate

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

##### NOR Gate

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

##### D Flip-Flop

| Clock | D | Q(next) |
|-------|---|---------|
| ↑     | 0 | 0       |

| Clock | D | Q(next) |
|-------|---|---------|
| ↑     | 1 | 1       |

## Design Flow

Specification



Library Creation



Schematic Design



Spectre Simulation



Manual Layout



Connectivity-Driven Layout (Layout XL)



DRC



LVS



Parasitic Extraction



Post-Layout Simulation

## Procedure

### Part A – Library Creation

#### Step 1

Launch Cadence Virtuoso.

#### Step 2

Open Library Manager.

#### Step 3

Create a new library:

File → New → Library

#### Step 4

Attach the required technology library (PDK).

Example:

- gpdk180
- gpdk90
- Foundry PDK (if provided)

## **Part B – CMOS Schematic Design**

### **1. CMOS Inverter**

Create Cell:

Library : VLSI\_LAB

Cell : CMOS\_INV

View : schematic

Place Components:

- PMOS
- NMOS
- VDD
- GND
- Input Pin
- Output Pin

Assign transistor dimensions (example for GPDK180):

- NMOS:  $W = 1\ \mu\text{m}$ ,  $L = 180\ \text{nm}$
- PMOS:  $W = 2\ \mu\text{m}$ ,  $L = 180\ \text{nm}$

Perform:

Design → Check and Save

### **2. CMOS NAND Gate**

Create Cell:

CMOS\_NAND

Place:

- 2 PMOS (parallel)
- 2 NMOS (series)
- VDD
- GND
- Inputs A, B
- Output Y

Run:

Check and Save

### **3. CMOS NOR Gate**

Create Cell:

CMOS\_NOR

Place:

- 2 PMOS (series)
- 2 NMOS (parallel)

Run:

Check and Save

### **4. CMOS D Flip-Flop**

Create Cell:

CMOS\_DFF

Implement using either:

- Transmission Gates + CMOS Inverters (recommended for learning), or
- Standard-cell DFF from the PDK (if instructed by the laboratory).

Pins:

- D
- CLK
- $\overline{Q}$
- $\overline{Q}$
- VDD
- GND

Run:

Check and Save

### **Part C – Spectre Simulation**

Launch:

Launch → ADE Explorer

Select:

Simulator : Spectre

Configure:

- DC Analysis (for logic verification if required)
- Transient Analysis

Typical settings:

- VDD = 1.8 V (GPDK180)
- Pulse source = 0–1.8 V
- Clock period (for DFF) = 20 ns
- Simulation time = 200 ns

Verify:

- Inverter output is the complement of the input.
- NAND and NOR outputs match their truth tables.
- D Flip-Flop output follows D on the active clock edge.

## **Part D – Manual Layout Using Virtuoso Layout Suite**

Create a **layout** view for each cell.

Draw:

- N-Well
- Active (Diffusion)
- Poly
- N+ Implant
- P+ Implant
- Contacts
- Metal-1
- Via
- Metal-2 (if required)

Layout guidelines:

### **Inverter**

- PMOS inside N-Well.
- NMOS in P-Substrate.
- Shared Poly gate.
- Common drain as output.

### **NAND**

- PMOS in parallel.
- NMOS in series.

### **NOR**

- PMOS in series.
- NMOS in parallel.

## D Flip-Flop

- Place transmission gates and inverters.
- Route CLK, D, Q, and  $\bar{Q}$  carefully.
- Use wider power rails for VDD and GND.

Save the layout.

## Part E – Connectivity-Driven Layout (Automatic-Assisted)

Cadence Virtuoso does **not** perform one-click automatic layout generation like FPGA place-and-route tools. Instead, use **Virtuoso Layout XL**:

1. Open the layout in **Virtuoso Layout XL**.
2. Choose:

Connectivity → Generate → All From Source

This imports:

- Devices
- Pins
- Nets

1. Use:

Connectivity → Update Components and Nets

1. Arrange devices with placement assistance.
2. Complete routing manually or using available routing assistants.

## Part F – Design Rule Check (DRC)

Run:

Verify → DRC

Check for:

- Minimum width
- Minimum spacing
- Poly extension
- Contact enclosure
- Well spacing
- Metal spacing

Expected result:

DRC Clean  
No Errors

## Part G – Layout Versus Schematic (LVS)

Run:

Verify → LVS

Compare:

- Schematic
- Layout

Expected result:

LVS Matched

## **Part H – Parasitic Extraction**

If Quantus/QRC is licensed:

Launch → Quantus Extraction

Extract:

- Parasitic resistance
- Parasitic capacitance

Generate the extracted view.

## **Part I – Post-Layout Simulation**

Open ADE Explorer.

Select the **extracted** view.

Run Spectre transient simulation.

Compare:

- Rise time
- Fall time
- Propagation delay
- Dynamic power
- Static power      with the schematic simulation.

### **Viva-Voce Questions**

1. Why is CMOS technology preferred over TTL?
2. Explain the operation of a CMOS inverter.
3. What is the purpose of Design Rule Check (DRC)?
4. What is Layout Versus Schematic (LVS)?
5. What is meant by parasitic capacitance in VLSI layouts?

### **Result**

CMOS Inverter, NAND Gate, NOR Gate, and D Flip-Flop were successfully designed using **Cadence Virtuoso**. Functional verification was completed using **Spectre ADE Explorer**. Manual layouts were created using **Virtuoso Layout Suite**, and connectivity-driven layout generation was performed using **Virtuoso Layout XL**. The layouts passed **DRC** and **LVS** verification. Parasitic extraction and post-layout simulation confirmed the correct operation of all circuits.

## Layout Generation Experiments

### Expt.No. 10      4-BIT SYNCHRONOUS COUNTER

**Aim:** Design and Simulate a 4-bit Synchronous counter using a Flip-Flops , perform  
· Manual/Automatic Layout Generation and Post Layout Extraction.

**Tool Required:**

- Functional Simulation: INCISIVE Simulator (ncvlog, ncelab, ncsim)
- Synthesis: GENUS Synthesis Solution Tool
- Physical Design Implementation/Automatic Layout Generation: INNOVUS Implementation Tool

#### Module 1: Work Space Creation & Writing RTL Code for 4Bit Synchronous Counter

**Getting Started:**

1. Create **sub directory/Folder** or folder with name **counter**
2. Open the “**counter**” directory and make a Right Click to “**Open in Terminal**”.
3. Create and Save the Verilog & Test-bench source code for counter using command shown below:  
**gedit counter.v counter\_test.v**
4. Invoke the the simulation tool for checking the functionality of the code with help of simulation waveform window.



```
saraswati@saraswati:~$  
File Edit View Search Terminal Help  
[saraswati@saraswati ~]$ csh  
[saraswati@saraswati ~]$ source /home/saraswati/install/cshrc  
  
Welcome to Cadence Tools Suite  
[saraswati@saraswati ~]$ nclaunch -new&
```

## Verilog and Test bench Code for 4BIT Synchronous Counter using Flip Flop:

```
counter.v
~/Desktop/VLSI_Lab_Annam_University_2022/counter

// 4-BIT Synchronous Counter Verilog Code
`timescale 1ns/1ps
module counter(clk,m,rst,count);
input clk,m,rst;
output reg [3:0] count;
always@(posedge clk or negedge rst)
begin
if(!rst)
count=0;
else if(m)
count=count+1;
else
count=count-1;
end
endmodule
```

```
counter_test.v
~/Desktop/VLSI_Lab_Annam_University_2022/counter

// 4-BIT Synchronous Counter Testbench Code
`timescale 1ns/1ps
module counter_test;
reg clk, rst,m;
wire [3:0] count;
initial
begin
clk=0;
rst=0;#100;
rst=1;
end
initial
begin
m=0;
#600 m=1;
#500 m=0;
end

counter counter1(clk,m,rst, count);

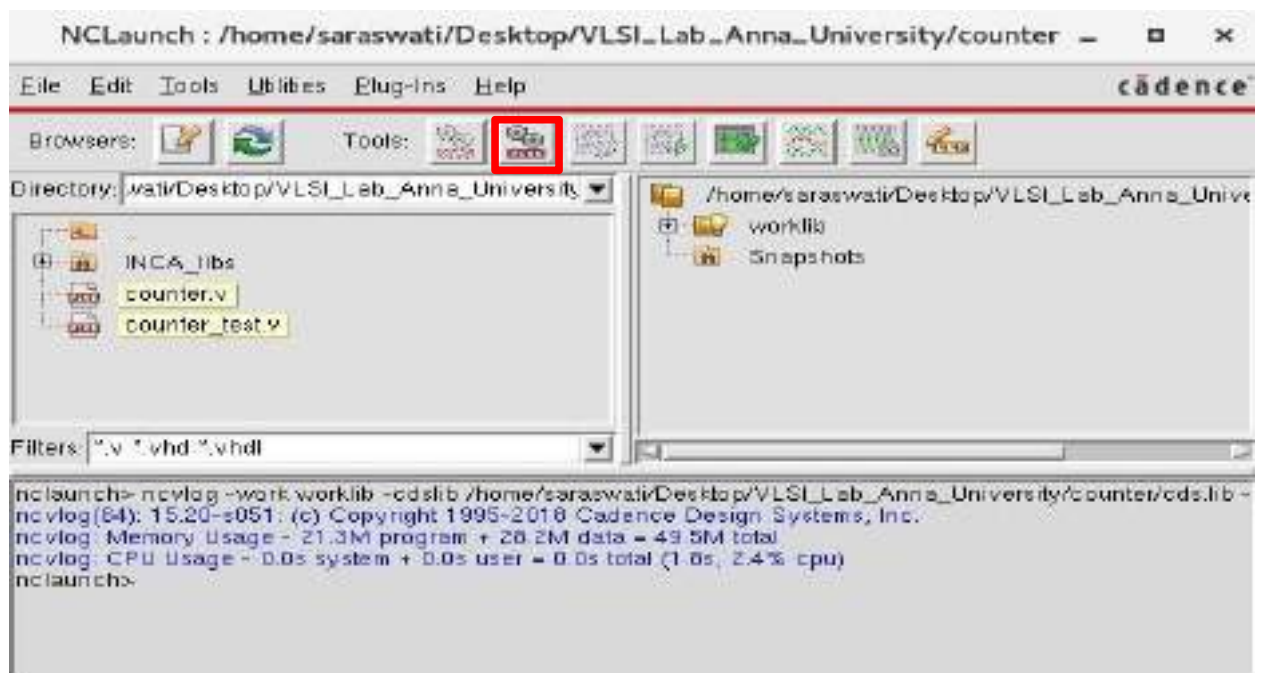
always #5 clk=~clk;

initial $monitor("Time=%t rst=%b clk=%b count=%b", $time,rst,clk,count);

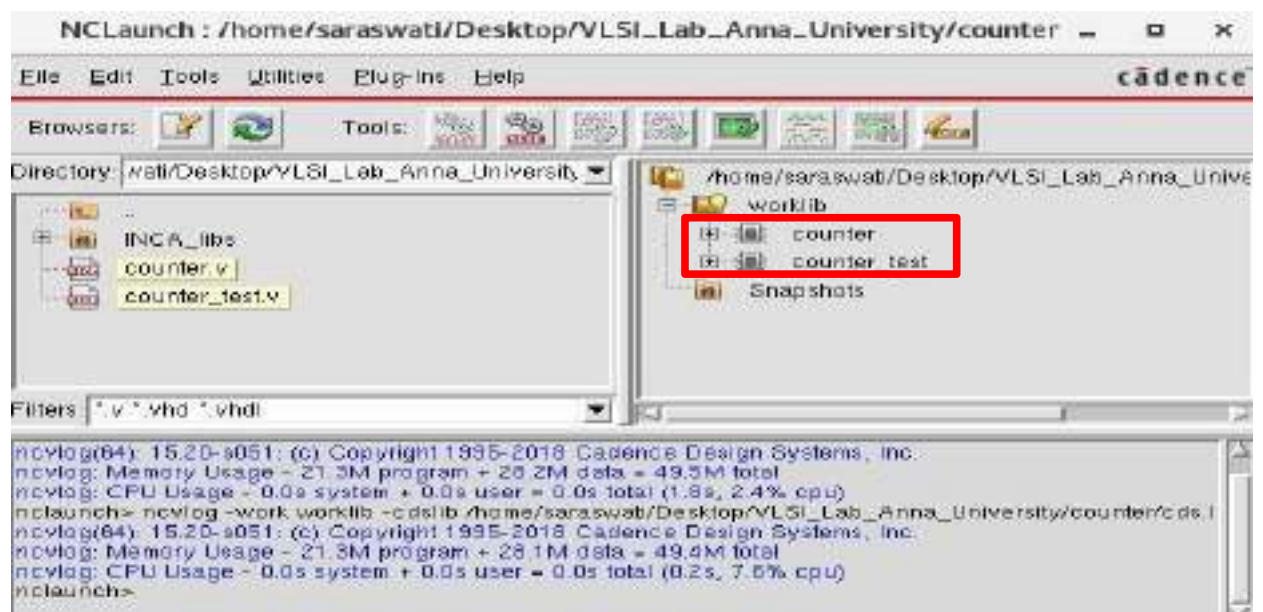
initial
#1400 $finish;

endmodule
```

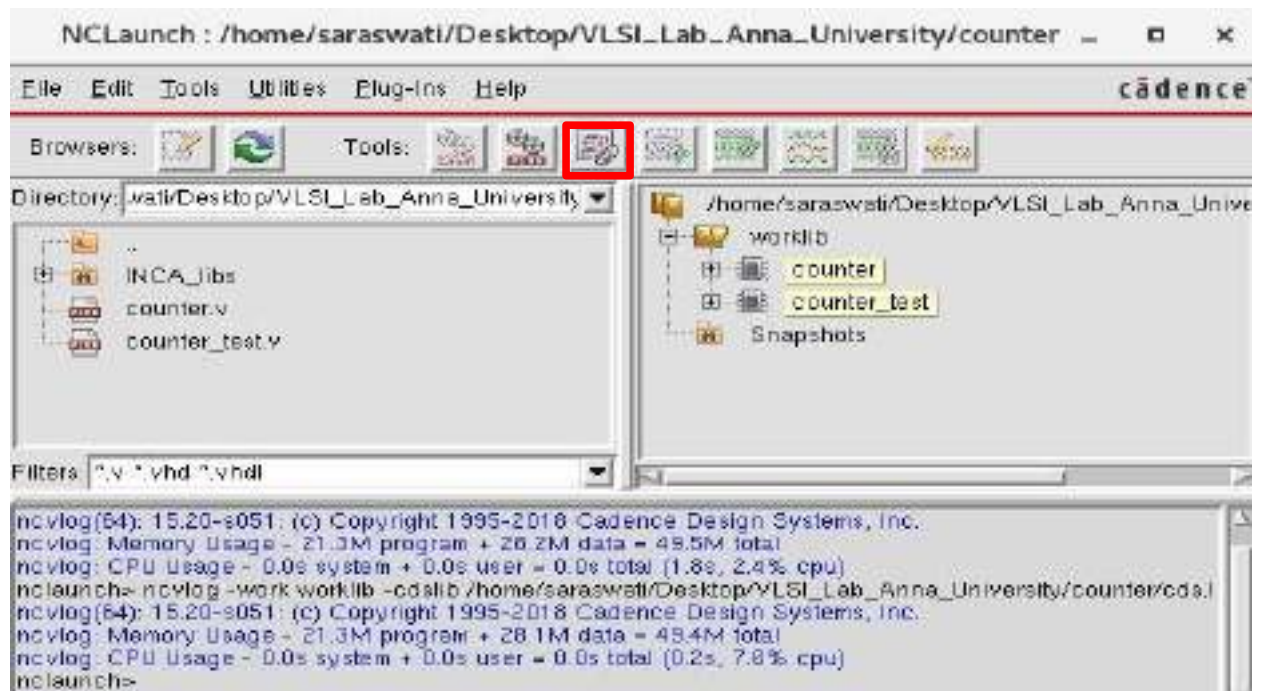
## Module 2: Design Simulation using INCISIVE Simulator:



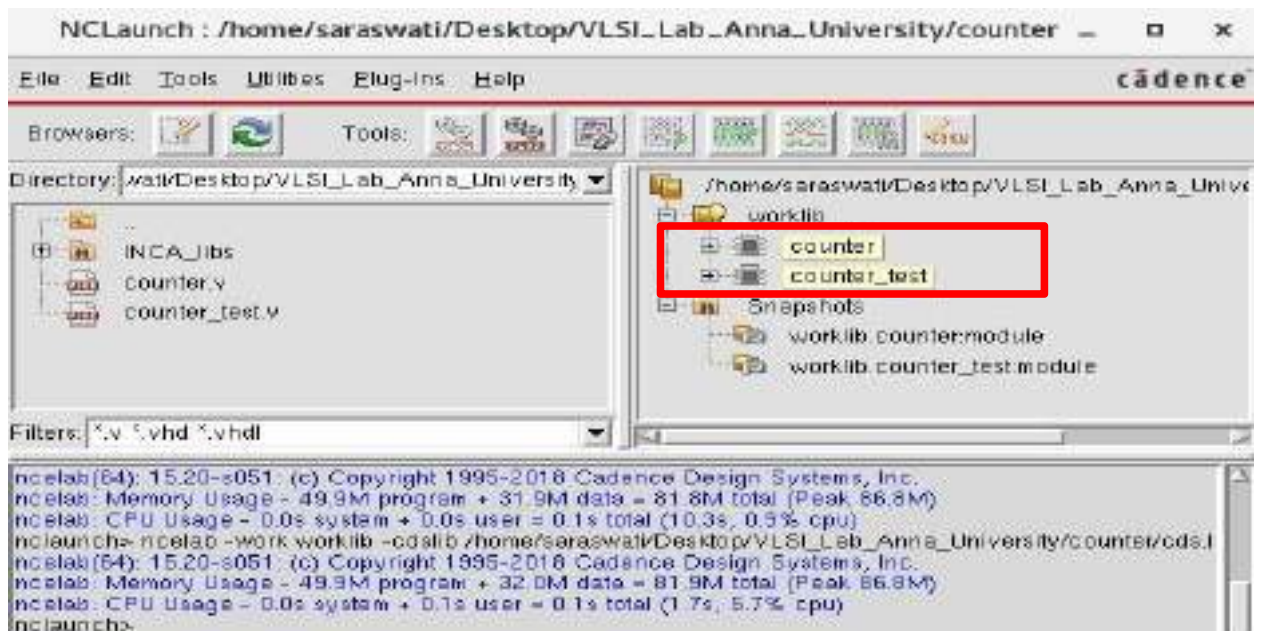
Compiled Data stored in worklib:



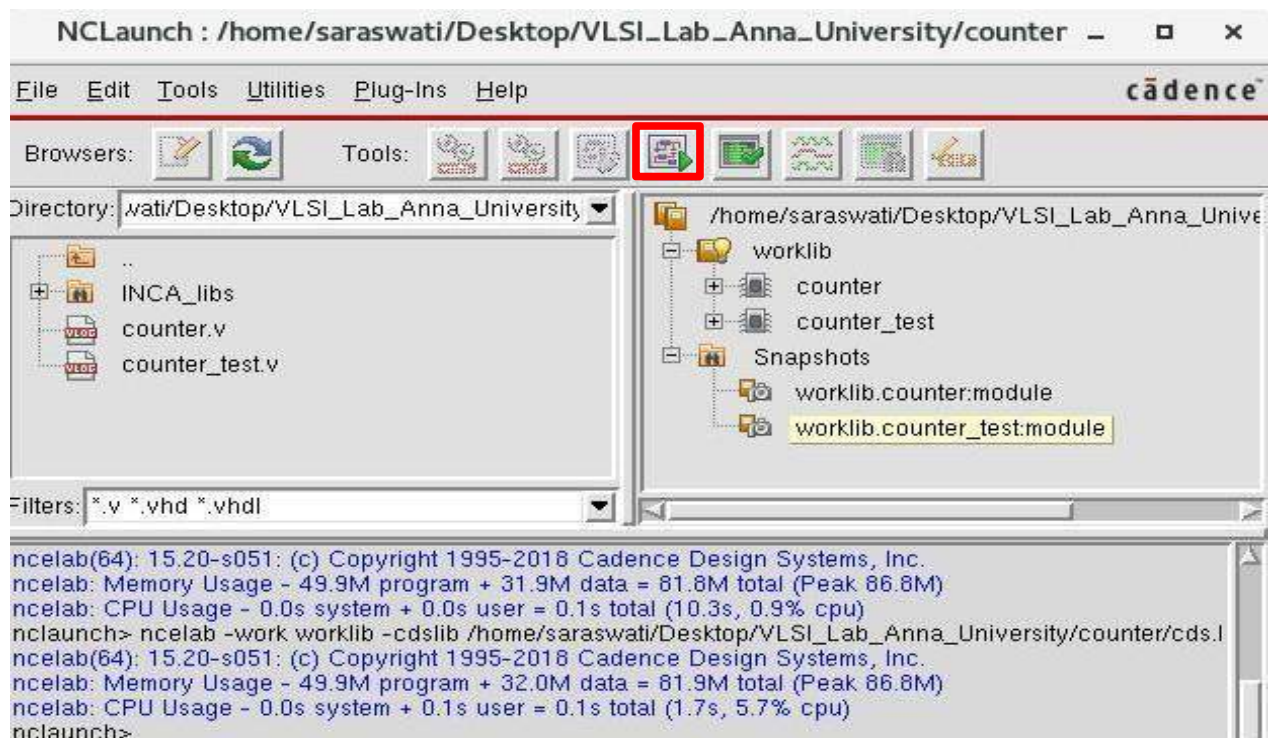
Elaboration:



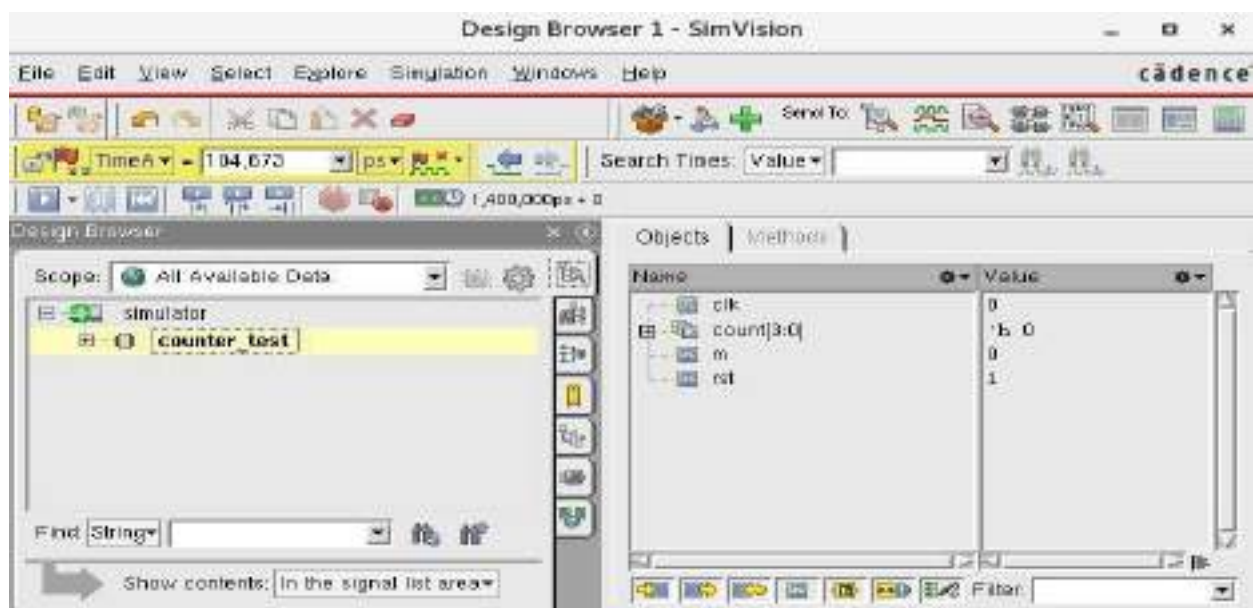
Elaborated Data Stored in Snapshots:



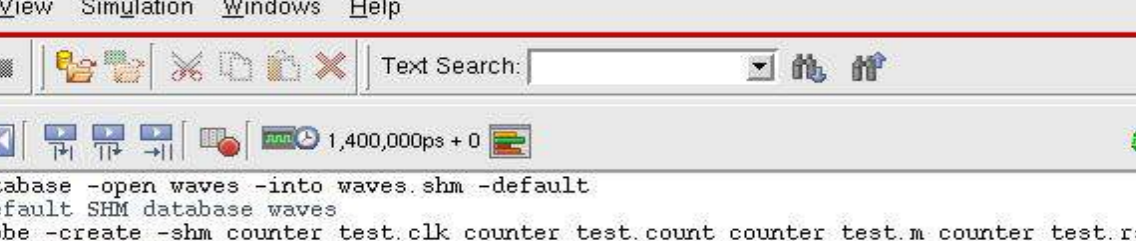
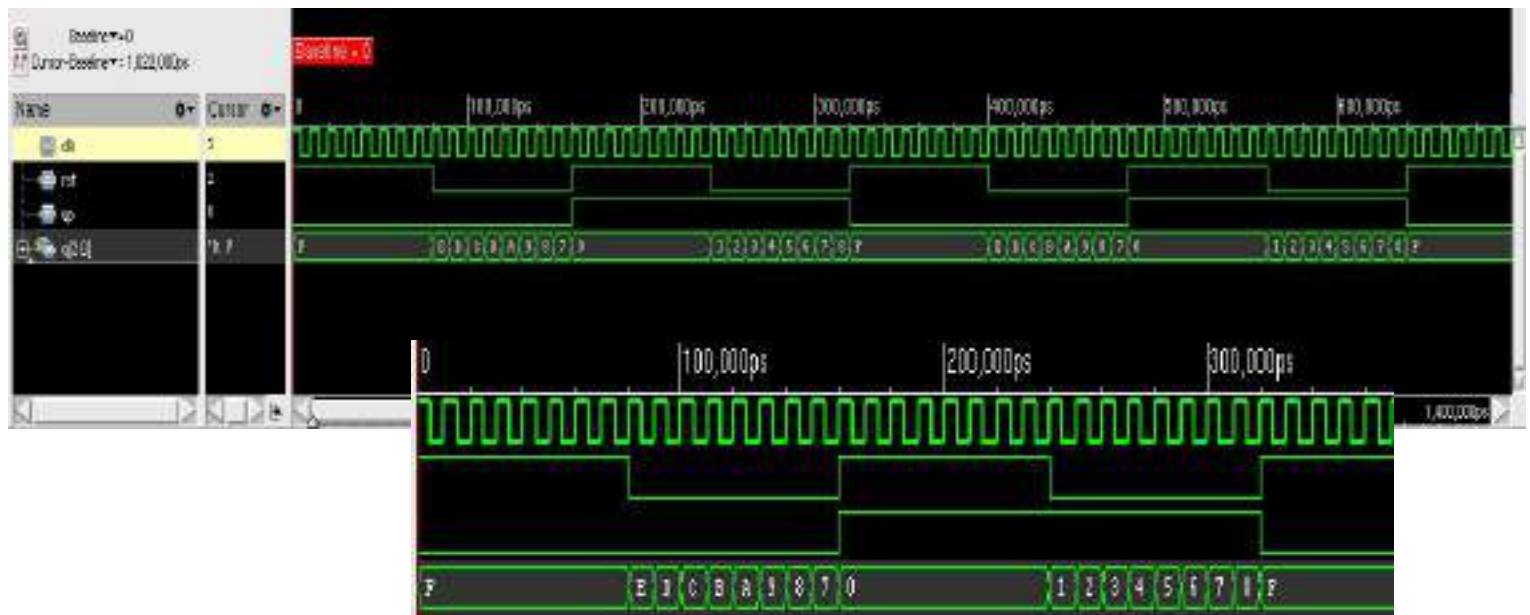
Simulation:



Chose the test scope and make a Right Click on the selected and Select “**Send to Waveform Window**”.



### Simulation Result:



The screenshot shows the Cadence SimVision console window. The title bar is "Console - SimVision". The menu bar includes File, Edit, View, Simulation, Windows, and Help. The toolbar contains various icons for simulation and editing. The text search field is empty. The console output shows the following commands and results:

```
ncsim> database -open waves -into waves.shm -default
Created default SHM database waves
ncsim> probe -create -shm counter_test.clk counter_test.count counter_test.rst
Created probe 1
ncsim> run
```

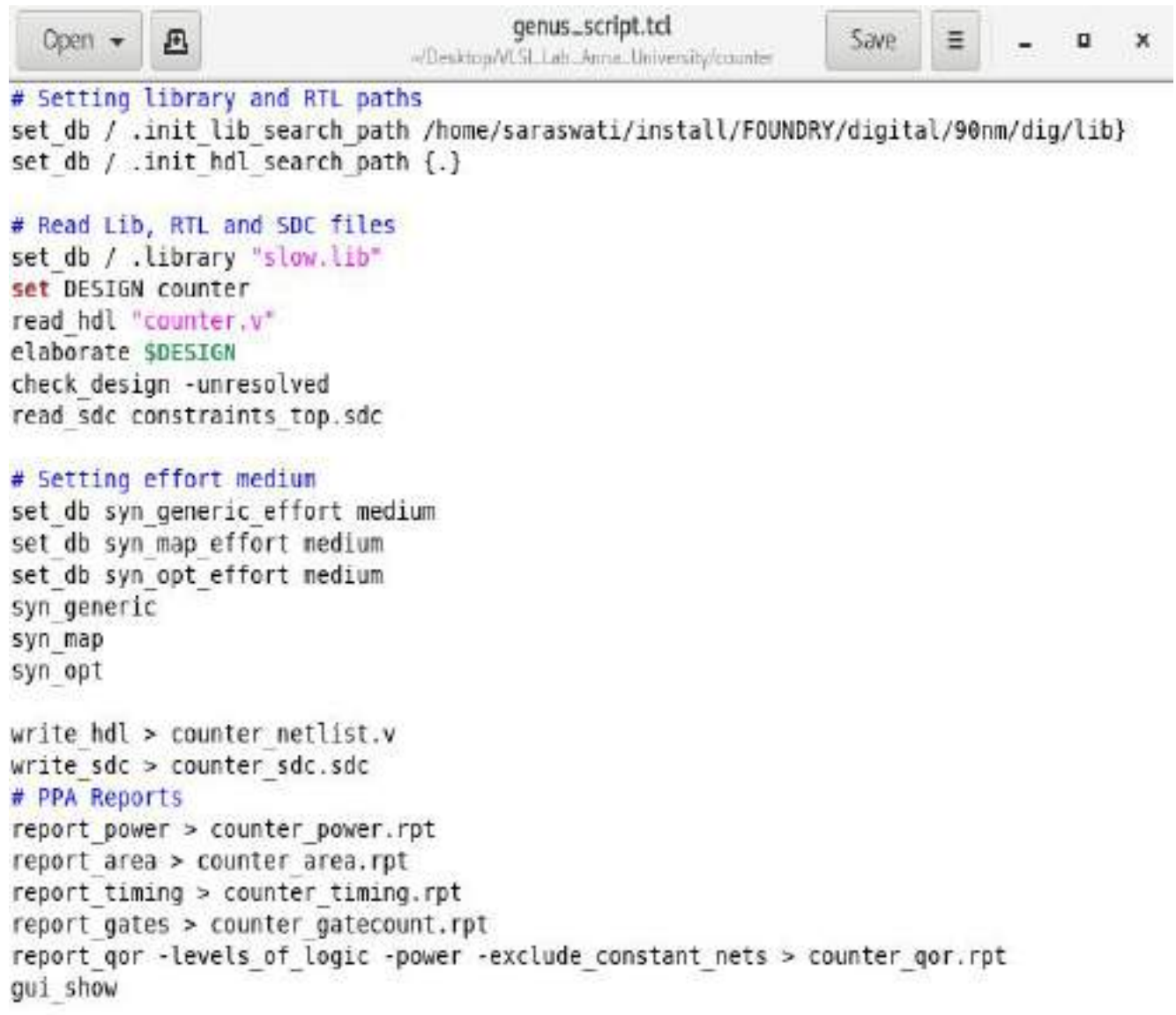
| Time= | 0     | rst=0 | clk=0 | count=0000 |
|-------|-------|-------|-------|------------|
| Time= | 5000  | rst=0 | clk=1 | count=0000 |
| Time= | 10000 | rst=0 | clk=0 | count=0000 |
| Time= | 15000 | rst=0 | clk=1 | count=0000 |
| Time= | 20000 | rst=0 | clk=0 | count=0000 |
| Time= | 25000 | rst=0 | clk=1 | count=0000 |
| Time= | 30000 | rst=0 | clk=0 | count=0000 |
| Time= | 35000 | rst=0 | clk=1 | count=0000 |
| Time= | 40000 | rst=0 | clk=0 | count=0000 |
| Time= | 45000 | rst=0 | clk=1 | count=0000 |
| Time= | 50000 | rst=0 | clk=0 | count=0000 |
| Time= | 55000 | rst=0 | clk=1 | count=0000 |
| Time= | 60000 | rst=0 | clk=0 | count=0000 |
| Time= | 65000 | rst=0 | clk=1 | count=0000 |
| Time= | 70000 | rst=0 | clk=0 | count=0000 |
| Time= | 75000 | rst=0 | clk=1 | count=0000 |
| Time= | 80000 | rst=0 | clk=0 | count=0000 |
| Time= | 85000 | rst=0 | clk=1 | count=0000 |

### Module 3: Synthesis Using Genus Tool

Synthesis runs in following stages :

- Translation
- Mapping
- Optimization

To run the synthesis, the following script can be used. Inside the genus script file we have to mention the commands as shown below.



```
genus_script.tcl
~/Desktop/VLSI_Lab_Arma_University/counter

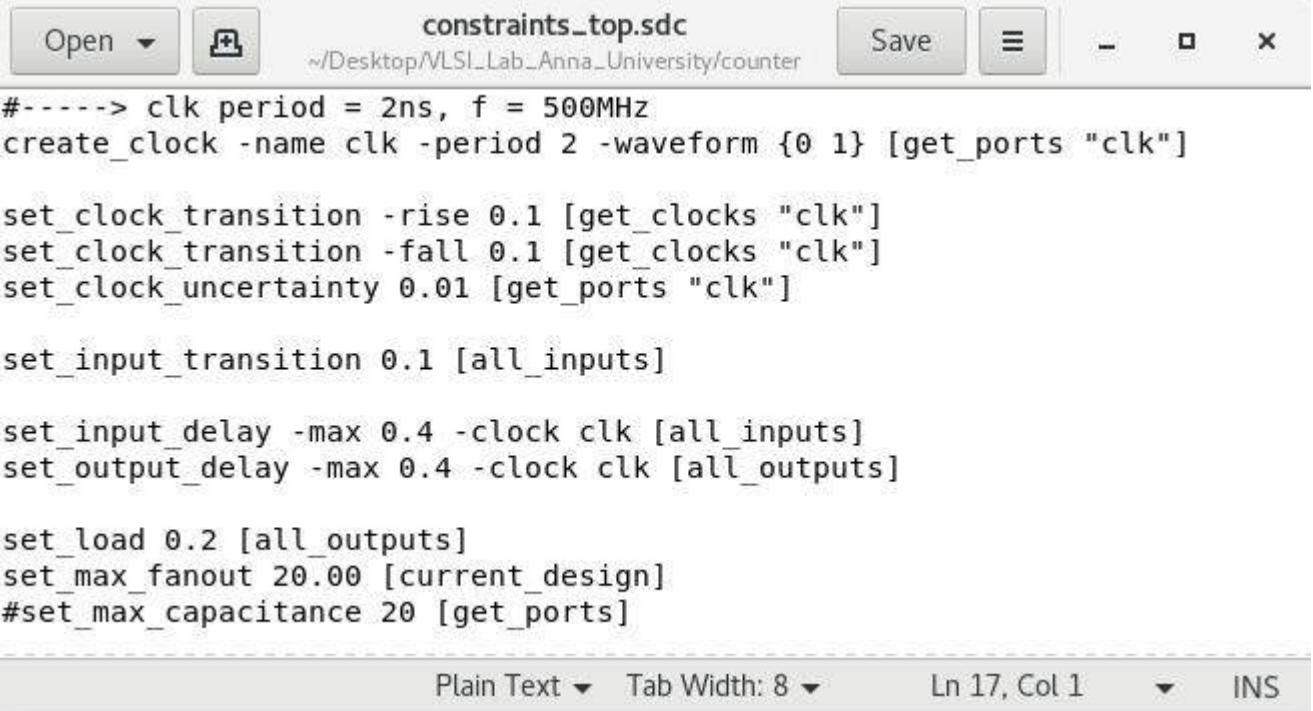
# Setting library and RTL paths
set_db / .init_lib_search_path /home/saraswati/install/FOUNDRY/digital/90nm/dig/lib}
set_db / .init_hdl_search_path {.}

# Read Lib, RTL and SDC files
set_db / .library "slow.lib"
set DESIGN counter
read_hdl "counter.v"
elaborate $DESIGN
check_design -unresolved
read_sdc constraints_top.sdc

# Setting effort medium
set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium
syn_generic
syn_map
syn_opt

write_hdl > counter_netlist.v
write_sdc > counter_sdc.sdc
# PPA Reports
report_power > counter_power.rpt
report_area > counter_area.rpt
report_timing > counter_timing.rpt
report_gates > counter_gatecount.rpt
report_qor -levels_of_logic -power -exclude_constant_nets > counter_qor.rpt
gui_show
```

Chip Level SDC is as follows:



The screenshot shows a text editor window titled "constraints\_top.sdc" with a file icon and a "Save" button. The file path is "~/Desktop/VLSI\_Lab\_Anna\_University/counter". The script content is as follows:

```
#-----> clk period = 2ns, f = 500MHz
create_clock -name clk -period 2 -waveform {0 1} [get_ports "clk"]

set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]

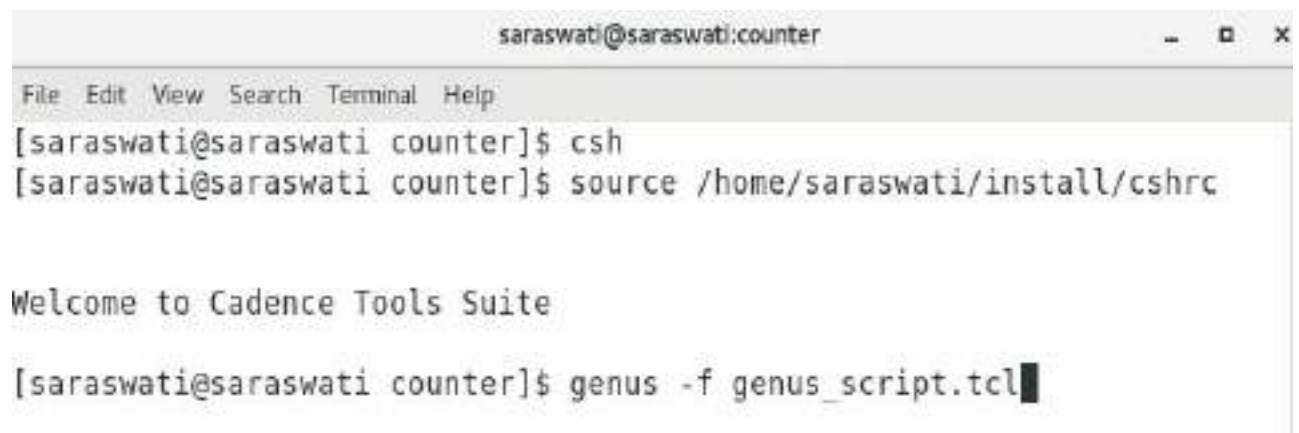
set_input_transition 0.1 [all_inputs]

set_input_delay -max 0.4 -clock clk [all_inputs]
set_output_delay -max 0.4 -clock clk [all_outputs]

set_load 0.2 [all_outputs]
set_max_fanout 20.00 [current_design]
#set_max_capacitance 20 [get_ports]
```

The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 17, Col 1", and "INS".

Invoke the Genus tool by typing the below command on your terminal: **genus -f genus\_script.tcl**  
The tcl [Tool Command Language] script runs executing each command one after the other.

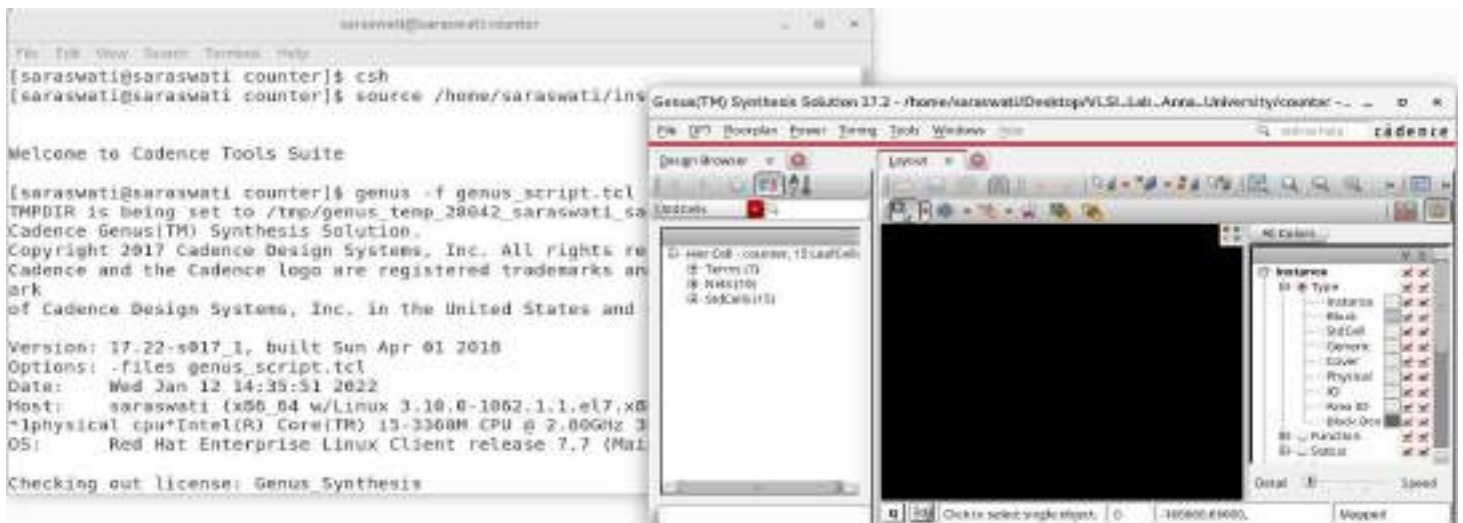


The screenshot shows a terminal window titled "saraswati@saraswati:counter". The terminal output is as follows:

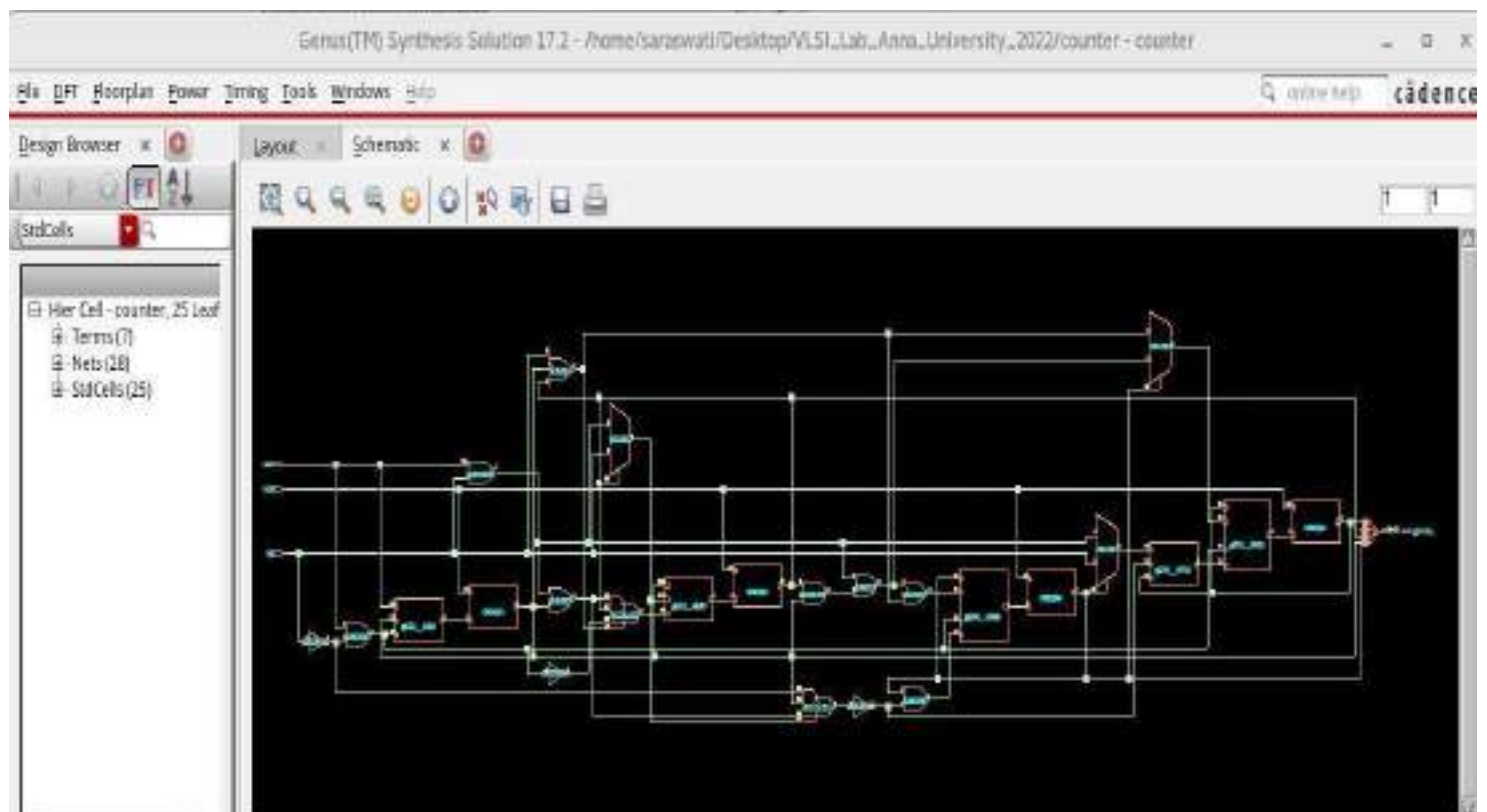
```
File Edit View Search Terminal Help
[saraswati@saraswati counter]$ csh
[saraswati@saraswati counter]$ source /home/saraswati/install/cshrc

Welcome to Cadence Tools Suite

[saraswati@saraswati counter]$ genus -f genus_script.tcl
```



A window of **Genus GUI** pops – up with the top hier cell on the left top. Make a **Right Click** and select **Schematic Viewer** → **In Main**.



## Power Report

```
=====
Generated by:      Genus(TM) Synthesis Solution 17.22
Generated on:      Mar 14 2022  04:14:40 pm
Module:           counter
Technology library: slow
Operating conditions: slow (balanced_tree)
Wireload mode:    enclosed
Area mode:        timing library
=====
```

|          |       | Leakage   | Dynamic    | Total      |
|----------|-------|-----------|------------|------------|
| Instance | Cells | Power(nW) | Power(nW)  | Power(nW)  |
| counter  | 25    | 1060.618  | 211192.463 | 212253.081 |

## Gate Count Report

```
=====
Module:           counter
Operating conditions: slow (balanced_tree)
Wireload mode:    enclosed
Area mode:        timing library
=====
```

| Gate      | Instances | Area    | Library |
|-----------|-----------|---------|---------|
| AOI211X1  | 1         | 5.298   | slow    |
| AOI211XL  | 1         | 5.298   | slow    |
| CLKINVX1  | 3         | 6.812   | slow    |
| DFF0X4    | 4         | 78.718  | slow    |
| MXI2XL    | 3         | 18.166  | slow    |
| NAND2BX1  | 1         | 4.541   | slow    |
| NAND2XL   | 3         | 9.083   | slow    |
| NOR2XL    | 3         | 9.083   | slow    |
| NOR3X1    | 1         | 4.541   | slow    |
| OAI211X1  | 2         | 10.597  | slow    |
| OAI21X1   | 2         | 9.083   | slow    |
| OAI2BB1XL | 1         | 5.298   | slow    |
| total     | 25        | 166.518 |         |

| Type           | Instances | Area    | Area % |
|----------------|-----------|---------|--------|
| sequential     | 4         | 78.718  | 47.3   |
| inverter       | 3         | 6.812   | 4.1    |
| logic          | 18        | 80.988  | 48.6   |
| physical_cells | 0         | 0.000   | 0.0    |
| total          | 25        | 166.518 | 100.0  |

### Area Report:

```

=====
Generated by:      Genus(TM) Synthesis Solution 17.22-s017_1
Generated on:     Mar 14 2022  04:14:40 pm
Module:          counter
Technology library:  slow
Operating conditions:  slow (balanced_tree)
Wireload mode:    enclosed
Area mode:       timing library
=====

Instance Module  Cell Count  Cell Area  Net Area  Total Area  Wireload
-----
counter          25      166.518    0.000    166.518    <none> (D)

(D) = wireload is default in technology library

```

### Timing Report:

```

Module: counter
Operating conditions: slow (balanced_tree)
Wireload mode: enclosed
Area node: timing library
=====

Path 1: MET (595 ps) Setup Check with Pin q_reg[3]/CK->0
  Group: clk
  Startpoint: (R) q_reg[1]/CK
  Clock: (R) clk
  Endpoint: (F) q_reg[3]/0
  Clock: (R) clk

      Capture          Launch
  Clock Edge: +      2000          0
  Src Latency: +      0            0
  Net Latency: +      0 (I)        0 (I)
  Arrival: =      2000          0

  Setup: -      123
  Uncertainty: -      10
  Required Time: =      1867
  Launch clock: -      0
  Data Path: -      1272
  Slack: =      595

#-----
# Timing Point   Flags   Arc   Edge   Cell          Fanout   Load   Trans   Delay   Arrival   Instance
#                                     (ff)   (ps)   (ps)   (ps)   Location
#-----
q_reg[1]/CK      -      -      R      (arrival)      4      -      100      -      0      (-,-)
q_reg[1]/0       -      CK->0 F      DFFQX4          5      209.8   376      605      605      (-,-)
g583_7675/Y      -      50->Y F      MUX12XL         2      4.5     222      241      846      (-,-)
g578_9682/Y      -      C0->Y R      AOI211X1        1      2.7     157      158      1003      (-,-)
g577/Y           -      A->Y F      CLKINVX1         2      4.4     72       65      1068      (-,-)
g572_3772/Y      -      A1->Y R      AOI211X1        1      2.9     134      93      1161      (-,-)
g570_8780/Y      -      C0->Y F      AOI211X1        1      1.6     149      111      1272      (-,-)
q_reg[3]/0       -      -      F      DFFQX4          1      -      -      0      1272      (-,-)

```

## Viva-Voce Questions

1. Why are Flip-Flops used in counters?
2. Why is a common clock used in synchronous counters?
3. What is the difference between DRC and LVS?
4. What is Automatic Layout Generation?
5. Why is timing analysis important in VLSI design?

**Result :**

Design and Simulation of a 4-bit Synchronous counter using a Flip-Flops is done and , performed Manual/Automatic Layout Generation and Post Layout Extraction.

# Analog Circuit Design Experiments

## Expt.No.11. Design of CMOS Inverting Amplifier

### Aim:

Design and Simulate a CMOS Inverting Amplifier and Analyze the Input impedance, Output impedance, Gain and Bandwidth .

### Tool Required:

- IC 6.1.8
- Spectre

### Getting Started:

1. Make sure the Licensing Server is switched ON and the client is connected to server.
2. **General Note:** Before starting to work on a design, create a Workspace (Folder) for the project individually.
3. **Work Space Creation:** Make a right click on the Desktop and select the option “New Folder” and name the folder (for example: VLSI\_Lab\_Anna\_University) and click on “Create”. Open the folder by a double click
4. Open the “VLSI\_Lab\_Anna\_University” folder or directory and make a right click to “**Open in Terminal**”.
5. In a terminal window, type `csch` at the command prompt to invoke the C shell.  
**#csch**
6. To verify that the path to the software is properly set in the `cschrc` file  
**#source /home/install/cschrc**

A welcome string “**Welcome to Cadence Tool Suite**” appears indicating terminal ready to invoke Cadence Tools available for you.

7. In the same terminal window, enter:  
**# virtuoso &**



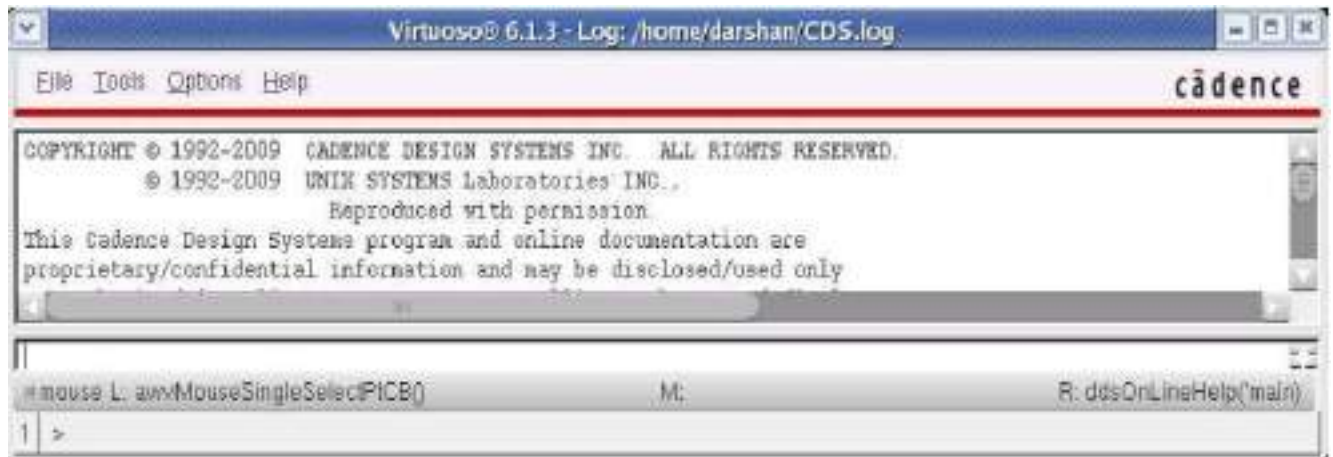
```
saraswati@saraswati:VLSI_Lab_Anna_University
File Edit View Search Terminal Help
[saraswati@saraswati VLSI_Lab_Anna_University]$ csch
[saraswati@saraswati VLSI_Lab_Anna_University]$ source /home/saraswati/install/cschrc

Welcome to Cadence Tools Suite

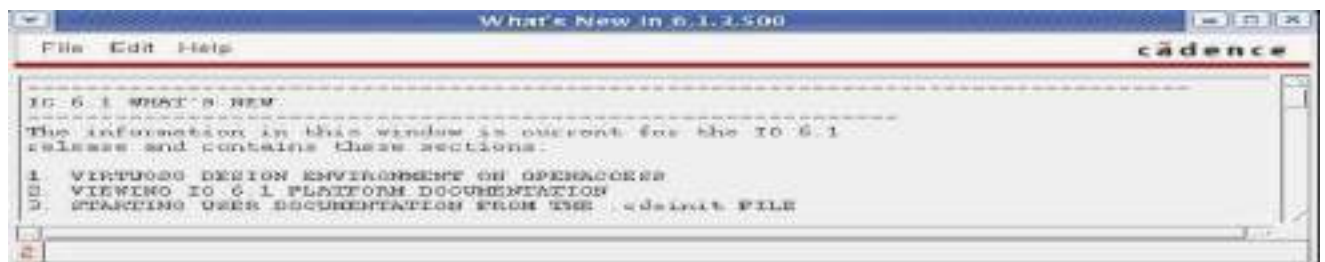
[saraswati@saraswati VLSI_Lab_Anna_University]$ virtuoso &
```

The virtuoso or Command Interpreter Window (CIW) appears at the bottom of the screen.

EC3566 VLSI Design Laboratory – Academic Year (2026-2027)

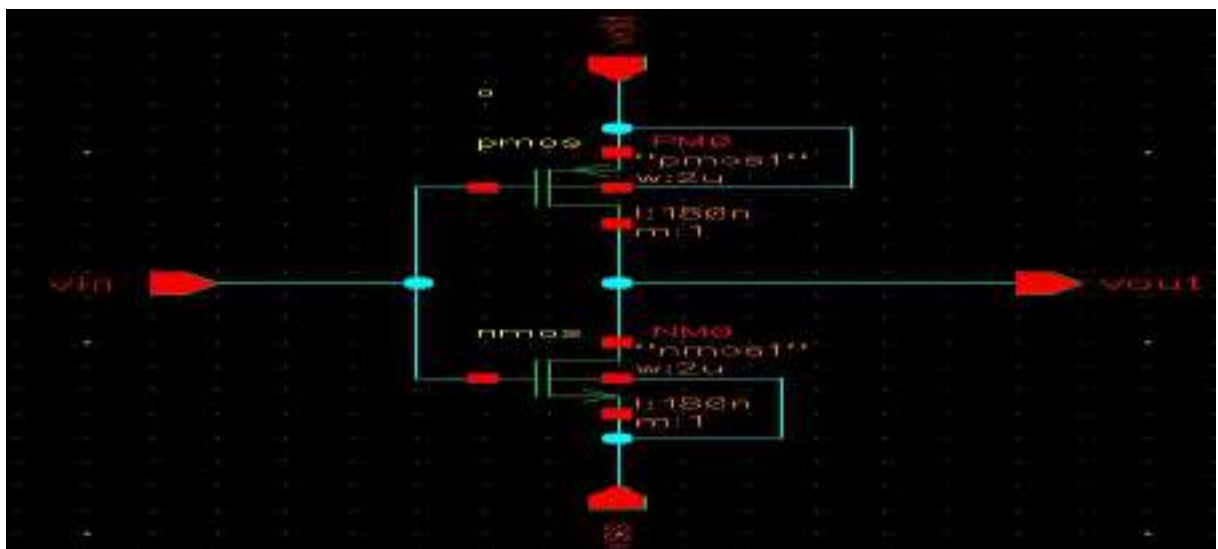


8. If the “What’s New ...” window appears, close it with the, **File— Close** command.



## Schematic Capture of CMOS Inverter:

### Adding Components to schematic



## Schematic Entry:

**Objective:** To create a library and build a schematic of an CMOS Inverting Amplifier or Inverter.

Below steps explain the creation of new library “**myDesignLib**” and we will use the same throughout this course for building various cells that we are going to create in the next labs.

Execute **Tools – Library Manager** in the CIW or Virtuoso window to open Library Manager.

### Creating a New Library:

1. In the Library Manager, execute **File - New – Library**. The new library form appears.
2. In the “New Library” form, type “**myDesignLib**” in the Name section.
3. In the next “**Technology File for New library**” form, select option **Attach to an existing techfile** and click **OK**.



4. In the “**Attach Design Library to Technology File**” form, select **gpd180** from the cyclic field and click **OK**.



7. After creating a new library you can verify it from the library manager.
8. If you right click on the “**myDesignLib**” and select properties, you will find that **gpdK180** library is attached as techlib to “**myDesignLib**”.



### Creating a Schematic Cellview:

In this section we will learn how to open new schematic window in the new “**myDesignLib**” library and build the inverter schematic as shown in the figure at the start of this lab.

1. In the CIW or Library manager, execute **File – New – Cellview**.
2. Set up the New file form as follows:



Do not edit the **Library path file** and the one above might be different from the path shown in your EC3566 VLSI Design Laboratory – Academic Year (2026-2027)

form. Click **OK** when done the above settings. A blank schematic window for the **Inverter** design appears



1. In the Inverter schematic window, click the **Instance** fixed menu icon to display the Add Instance form

**Tip:** You can also execute **Create — Instance** or press **i**.

2. Click on the **Browse** button. This opens up a Library browser from which you can select components and the **symbol** view.

You will update the Library Name, Cell Name, and the property values given in the table on the next page as you place each component.

3. After you complete the Add Instance form, move your cursor to the schematic window and click **left** to place a component.

This is a table of components for building the Inverter schematic.

| Library name | Cell Name | Properties/Comments               |
|--------------|-----------|-----------------------------------|
| gpd180       | nmos      | Model Name = nmos1; W=2u; L= 180n |
| gpd180       | pmos      | Model Name = pmos1; W=2u; L= 180n |

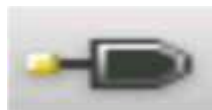
If you place a component with the wrong parameter values, use the **Edit— Properties— Objects** command to change the parameters along with selection of component. Use the **Edit— Move** command if you place components in the wrong location.



You can rotate components at the time you place them or use the **Edit— Rotate** command after they are placed.

4. After entering components, click **Cancel** in the Add Instance form or press **Esc** with your cursor in the schematic window.

### Adding pins to Schematic



1. Click the **Pin** fixed menu icon in the schematic window.

EC3566 VLSI Design Laboratory – Academic Year (2026-2027)

You can also execute **Create — Pin** or press **p**. The Add pin form appears.

2. Type the following in the Add pin form in the exact order leaving space between the pin names.

| Pin Name      | Direction |
|---------------|-----------|
| vin, vdd, vss | input     |
| vout          | output    |

Make sure that the direction field is set to **input/output/inputoutput** when placing the **input/output/inout** pins respectively and the usage field is set to schematic.

3. Select Cancel from the Add – pin form after placing the pins. In the schematic window, to fit the schematic design click on the window and press the **f** bind-key.



### Adding Wires to a Schematic:



Add wires to connect components and pins in the design.

1. Click the **Wire (narrow)** icon in the schematic window. You can also press the **w** key or execute **Create — Wire (narrow)**.

2. In the schematic window, click on a pin of one of your components as the first point for your wiring. A diamond shape appears over the starting point of this wire.

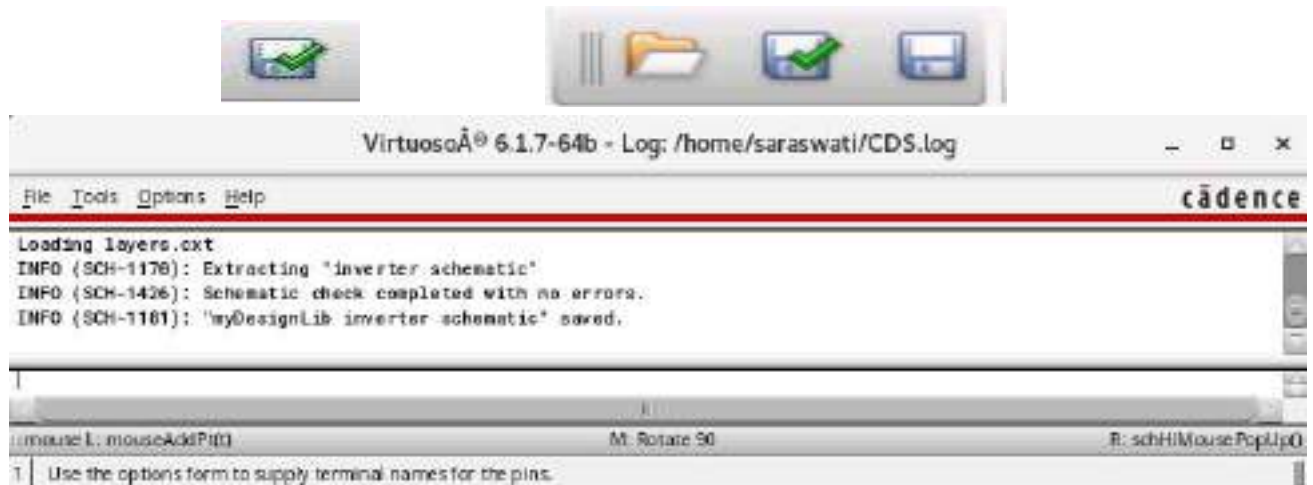
3. Follow the prompts at the bottom of the design window and click **left** on the destination point for your wire. A wire is routed between the source and destination points.

4. Complete the wiring as shown in figure and when done wiring press **ESC** key in the

### Saving the Design:

1. Click the Check and Save icon in the schematic editor window.

EC3566 VLSI Design Laboratory – Academic Year (2026-2027)



2. Observe the CIW output area for any errors.

## Symbol Creation:

**Objective:** To create a symbol for the Inverter

In this section, you will create a symbol for your inverter design, so you can place it in a test circuit for simulation. A symbol view is extremely important step in the design process.

1. In the Inverter schematic window, execute, **Create — Cellview— From Cellview**. The **Cellview From Cellview** form appears. With the Edit Options function active, you can control the appearance of the symbol to generate.
2. Verify that the **From View Name** field is set to schematic, and the **To View Name** field is set to **symbol**, with the **Tool/Data Type** set as **SchematicSymbol**



The 'Cellview From Cellview' dialog box is shown with the following settings:

- Library Name: myDesignLib
- Cell Name: inverter
- From View Name: schematic
- To View Name: symbol
- Tool / Data Type: schematicSymbol
- Display Cellview: ☒
- Edit Options: ☒

Buttons at the bottom: OK, Cancel, Defaults, Apply, Help.

3. Click **OK** in the **Cellview From Cellview** form. The Symbol Generation Form appears.

4. Modify the **Pin Specifications** as follows:

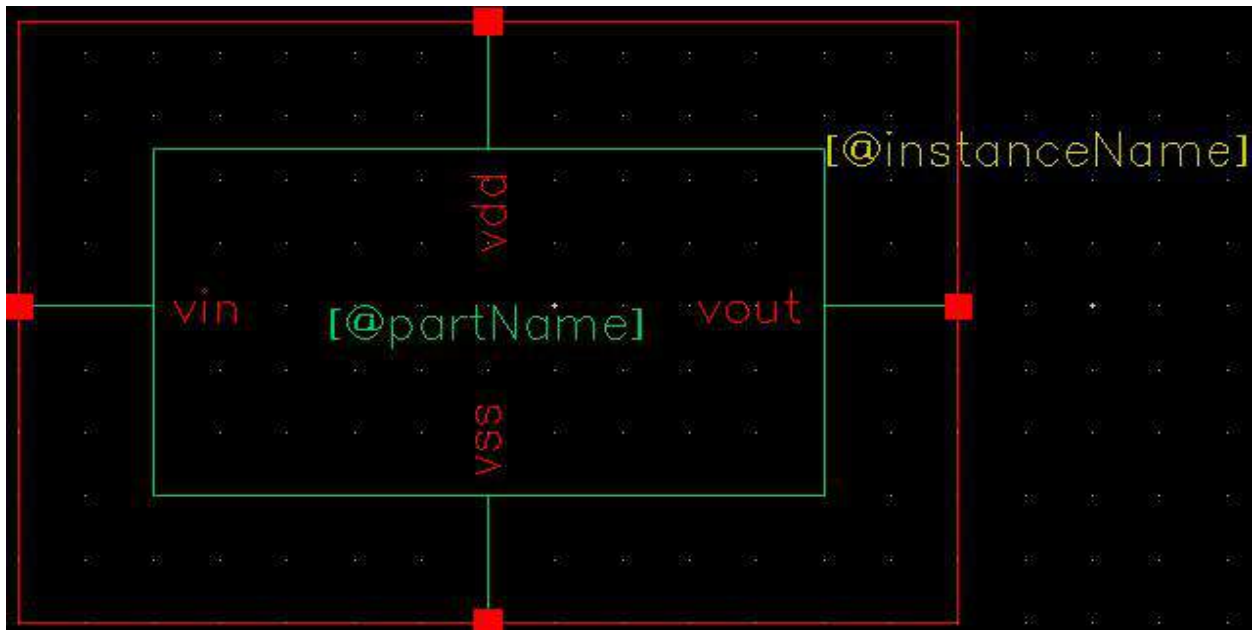


The 'Symbol Generation Options' dialog box is shown with the following settings:

- Library Name: myDesignLib
- Cell Name: inverter
- View Name: symbol
- Pin Specifications:
  - Left Pins: vin
  - Right Pins: vout
  - Top Pins: vdd
  - Bottom Pins: vss
- Exclude Inherited Connection Pins:
  - ☒ None
  - ☐ All
  - ☐ Only these:
- Buttons at the bottom: OK, Cancel, Apply, Help.

5. Click **OK** in the Symbol Generation Options form.

6. A new window displays an automatically created Inverter symbol as shown here.

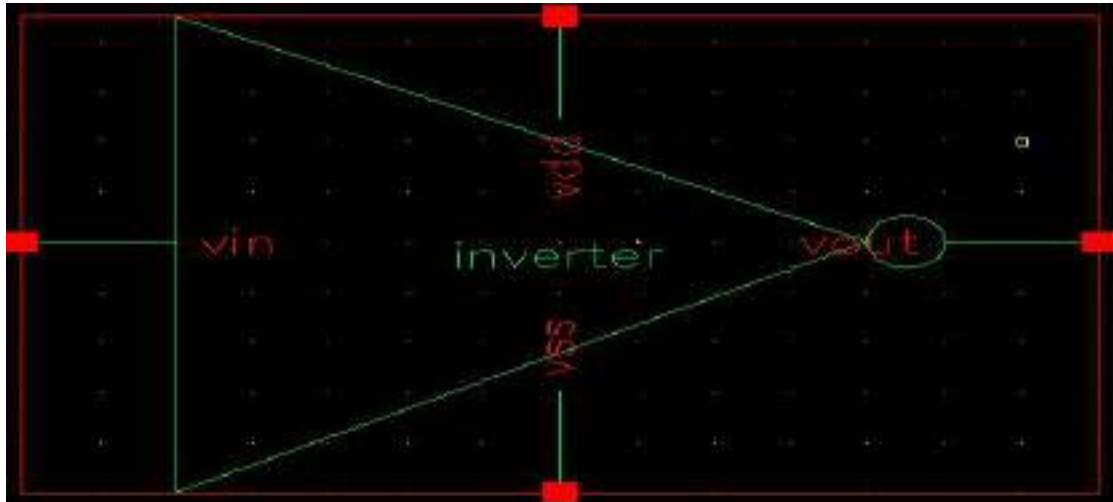


### Editing a Symbol:

In this section we will modify the inverter symbol to look like a Inverter gate symbol.



1. Move the cursor over the automatically generated symbol, until the green rectangle is highlighted, click **left** to select it.
  2. Click **Delete** icon in the symbol window, similarly select the red rectangle and delete that.
  3. Execute **Create – Shape – polygon** and draw a shape similar to triangle.
  4. After creating the triangle press **ESC** key.
  5. Execute **Create – Shape – Circle** to make a circle at the end of triangle.
  6. You can move the pin names according to the location.
  7. Execute **Create — Selection Box**. In the Add Selection Box form, click **Automatic**. A new red selection box is automatically added.
  8. After creating symbol, click on the **save** icon in the symbol editor window to save the symbol.
- In the symbol editor, execute **File — Close** to close the symbol view window.



### Building the Inverter\_Test Design

**Objective:** To build an Inverter Test circuit using your Inverter

You will create the Inverter\_Test cellview that will contain an instance of the inverter cellview. In the next section, you will run simulation on this design

1. In the **CIW** or Library Manager, execute **File— New— Cellview**.
2. Set up the New File form as follows:



3. Click **OK** when done. A blank schematic window for the **Inverter\_Test** design appears.

### Building the Inverter\_Test Circuit:

1. Using the component list and Properties/Comments in this table, build the **Inverter\_Test** schematic.

| Library Name | Cell-view Name | Properties/Comments |
|--------------|----------------|---------------------|
|--------------|----------------|---------------------|

|             |          |                                                          |
|-------------|----------|----------------------------------------------------------|
| myDesignLib | Inverter | Symbol                                                   |
| analogLib   | vpulse   | For V0: v1=0, v2=1.8v, td=0, tr=tf=1ns, ton=10ns, T=20ns |
| analogLib   | vdd, gnd | vdc=1.8v                                                 |

2. Add the above components using **Create — Instance** or by pressing **I**.

3. Click the **Wire (narrow)** icon and wire your schematic.



**Tip:** You can also press the w key, or execute **Create— Wire (narrow)**

4. Click **Create — Wire Name** or press **L** to name the input (**vin**) and output (**vout**) wires as in the below schematic.

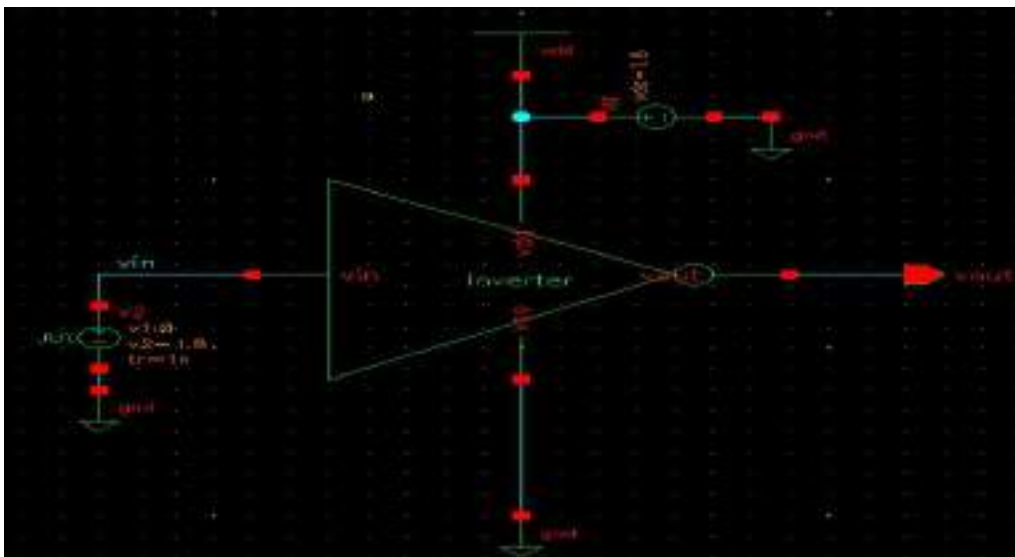


5. Click on the **Check and Save** icon to save the design. The schematic should look like this.



6. The schematic should look like this.

**Schematic Test Capture of CMOS Inverter:**



7. Leave your **Inverter\_Test** schematic window open for the next section.

## Analog Simulation with Spectre

Objective: To set up and run simulations on the Inverter\_Test design

In this section, we will run the simulation for Inverter and plot the transient, DC characteristics.

### Starting the Simulation Environment:

Start the Simulation Environment to run a simulation.

1. In the **Inverter\_Test** schematic window, execute **Launch – ADE L**

The **Virtuoso Analog Design Environment (ADE)** simulation window appears.

Set the environment to use the **Spectre® tool**, a high speed, highly accurate analog simulator. Use this simulator with the Inverter\_Test design, which is made-up of analog components.

1. In the simulation window (ADE), execute **Setup— Simulator/Directory/Host**.

2. In the Choosing Simulator form, set the Simulator field to **spectre** (Not spectreS) and click **OK**.

### Setting the Model Libraries:

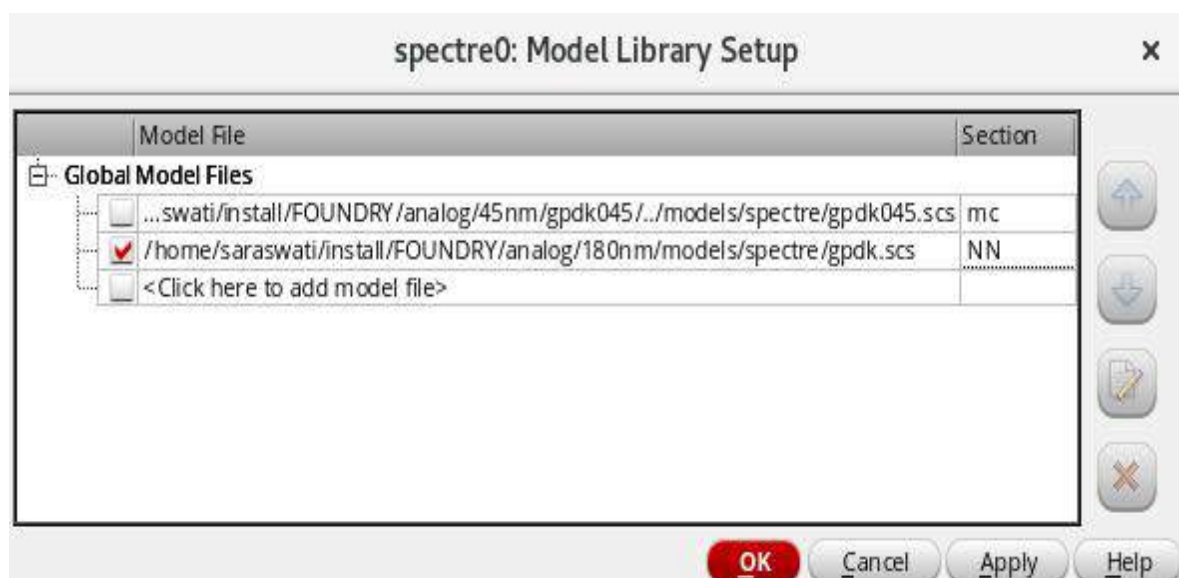
The Model Library file contains the model files that describe the nmos and pmos devices during simulation.

1. In the simulation window (ADE), Execute **Setup - Model Libraries**.

The Model Library Setup form appears. Click the **browse** button to add **gpdsk.scs** if not added by default as shown in the **Model Library Setup** form. Remember to select the section type as **NN** in front of the gpdsk.scs file.



Your Model Library Setup window should now look like the below figure.



To view the model file, highlight the expression in the Model Library File field and Click **Edit File**.



### Choosing Analyses:

This section demonstrates how to view and select the different types of analyses to complete the circuit when running the simulation.

1. In the Simulation window (ADE), click the **Choose - Analyses** icon. You can also execute  
EC3566 VLSI Design Laboratory – Academic Year (2026-2027)

## Analyses - Choose.



The Choosing Analysis form appears. This is a dynamic form, the bottom of the form changes based on the selection above.

### 2. To setup for **transient analysis**:

- a. In the Analysis section select **tran**
- b. Set the stop time as **200n**
- c. Click at the **moderate** or **Enabled** button at the bottom, and then click **Apply**

Choosing Analyses -- ADE L (1)

Analysis: ☒ tran ☐ dc ☐ ac ☐ noise  
☐ vf ☐ sens ☐ dcmatch ☐ acmatch  
☐ stb ☐ pz ☐ lf ☐ sp  
☐ envp ☐ pss ☐ pac ☐ pttb  
☐ pncba ☐ pxf ☐ psp ☐ qpaa  
☐ qpac ☐ qpnoise ☐ qpaf ☐ qpap  
☐ hb ☐ hbac ☐ hbtb ☐ hbnose  
☐ hbap ☐ hbaa

Transient Analysis

Stop Time:

Accuracy Defaults (errpreset)  
☐ conservative ☒ moderate ☐ liberal

☐ Transient Noise

Dynamic Parameter ☐

Enabled ☒

Options...

OK Cancel Defaults Apply Help

To set up for **DC Analyses**:

- d. In the Analyses section, select **dc**.
- e. In the DC Analyses section, turn on **Save DC Operating Point**.
- f. Turn on the **Component Parameter**.
- g. Double click the **Select Component**, which takes you to the schematic window.
- h. Select input signal **vpulse source** in the test schematic window.
- i. Select **—DC Voltage** in the **Select Component Parameter** form and click OK.
- j. In the analysis form type **start** and **stop** voltages as **0** to **1.8** respectively.
- k. Check the enable button and then click **Apply**.

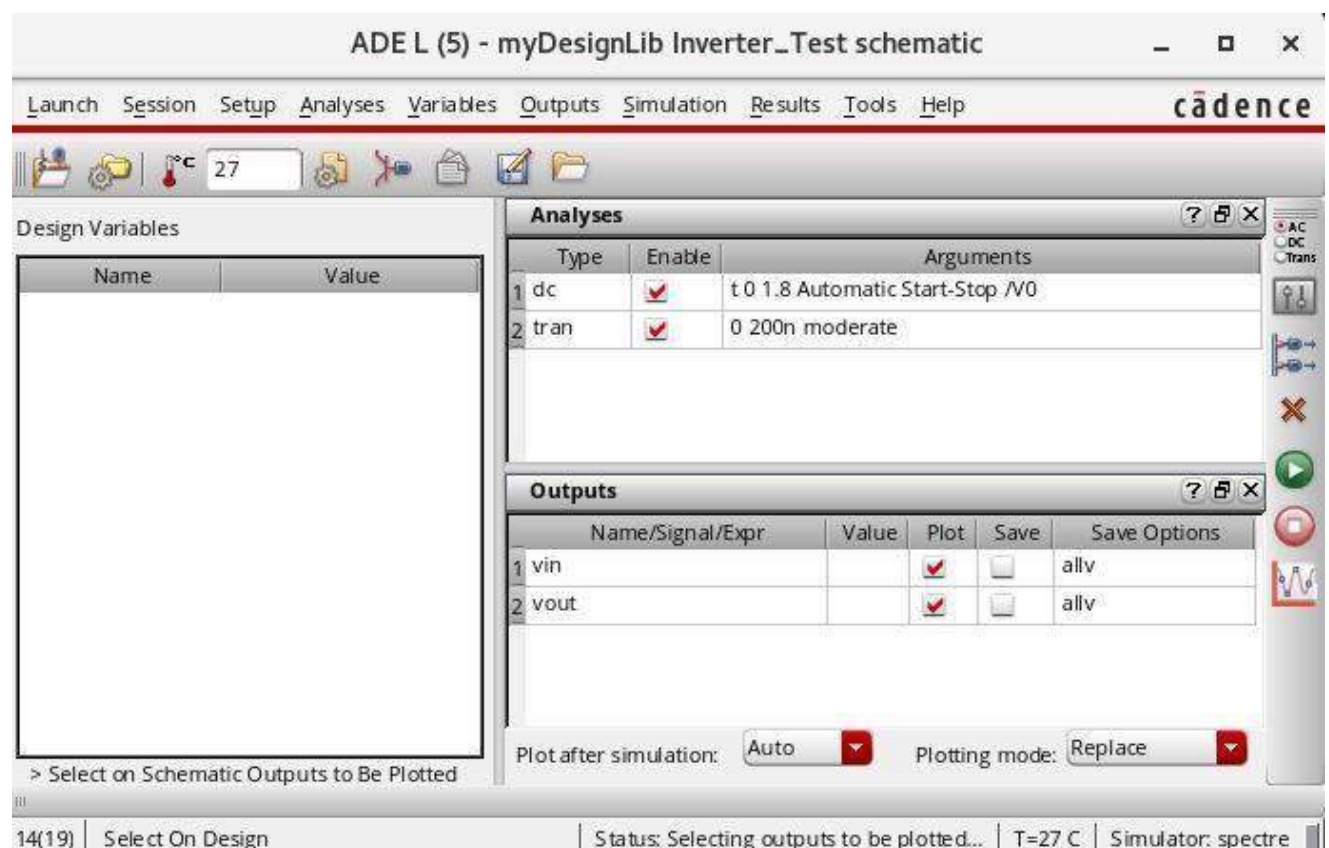
The image shows a screenshot of the "Choosing Analyses -- ADE L (1)" dialog box. The "Analysis" section contains a grid of radio buttons for various analysis types: tran, dc (selected), ac, noise, xf, sens, dcmatch, acmatch, stb, pz, lf, sp, envlp, pss, pac, pstb, pnoise, pxf, psp, qpss, qpac, qpnoise, qpxf, qpss, hb, hbac, hbstb, hbnoise, hbsp, and hbsf. Below this, the "DC Analysis" section has "Save DC Operating Point" checked and "Hysteresis Sweep" unchecked. The "Sweep Variable" section has "Temperature", "Design Variable", "Component Parameter" (checked), and "Model Parameter" options. The "Component Name" field contains "AVE" and the "Parameter Name" field contains "dc". The "Sweep Range" section has "Start-Stop" (selected) with "Start" at 0 and "Stop" at 1.8, and "Center-Span" is unselected. The "Sweep Type" is set to "Automatic". The "Add Specific Points" checkbox is unchecked. At the bottom, the "Enabled" checkbox is checked. The dialog box has buttons for "OK", "Cancel", "Defaults", "Apply", and "Help".

3. Click OK in the Choosing Analyses Form.

## Selecting Outputs for Plotting:

1. Execute **Outputs – To be plotted – Select on Schematic** in the simulation window.
2. Follow the prompt at the bottom of the schematic window, Click on output net **vout**, input net **vin** of the Inverter. Press **ESC** with the cursor in the schematic after selecting it.

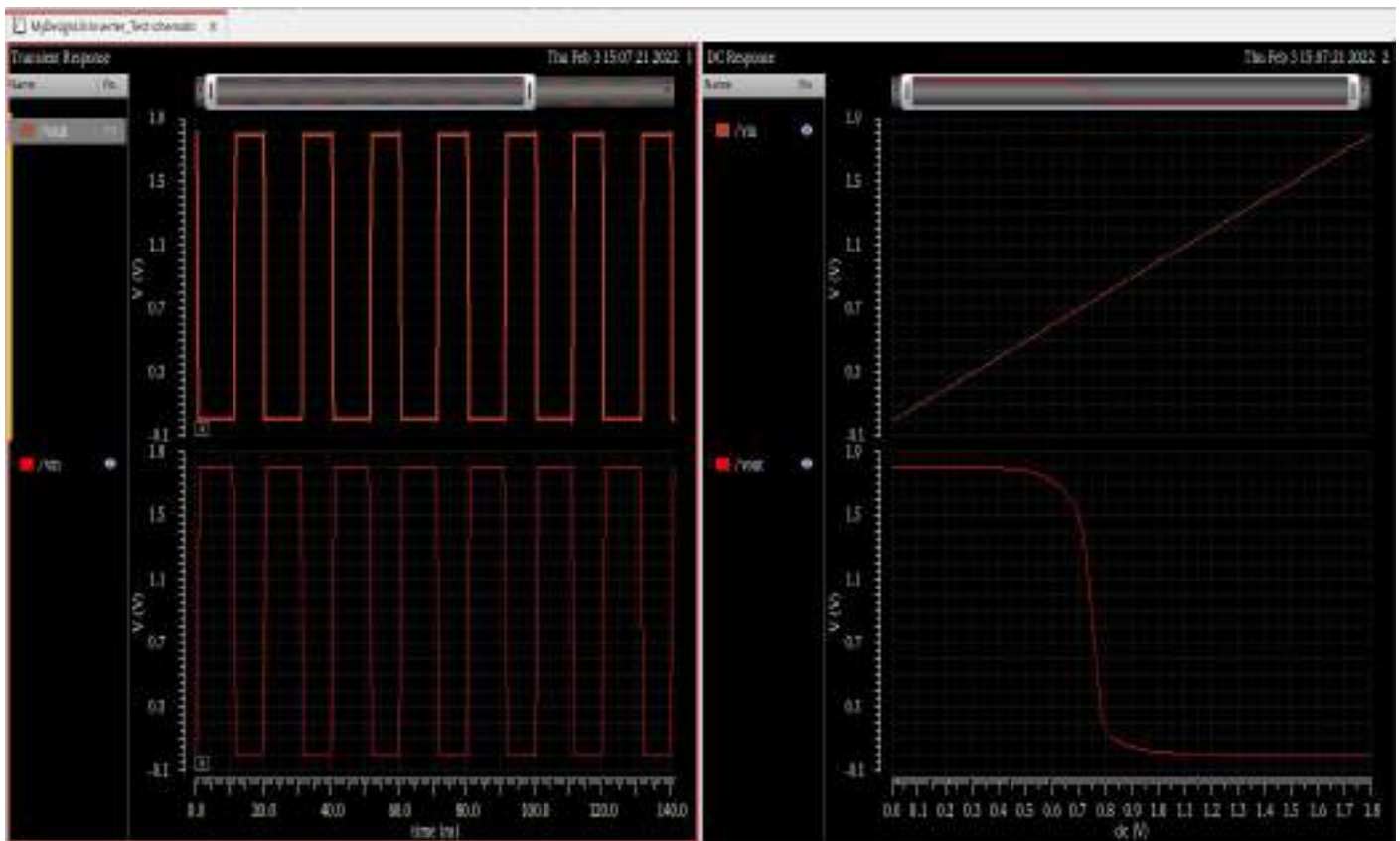
Does the simulation window look like this?



## Running the Simulation:



1. Execute **Simulation – Netlist and Run** in the simulation window to start the Simulation or the icon, this will create the netlist as well as run the simulation.
2. When simulation finishes, the Transient, DC plots automatically will be popped up along with log file.



### Saving the Simulator State:

We can save the simulator state, which stores information such as model library file, outputs, analysis, variable etc.

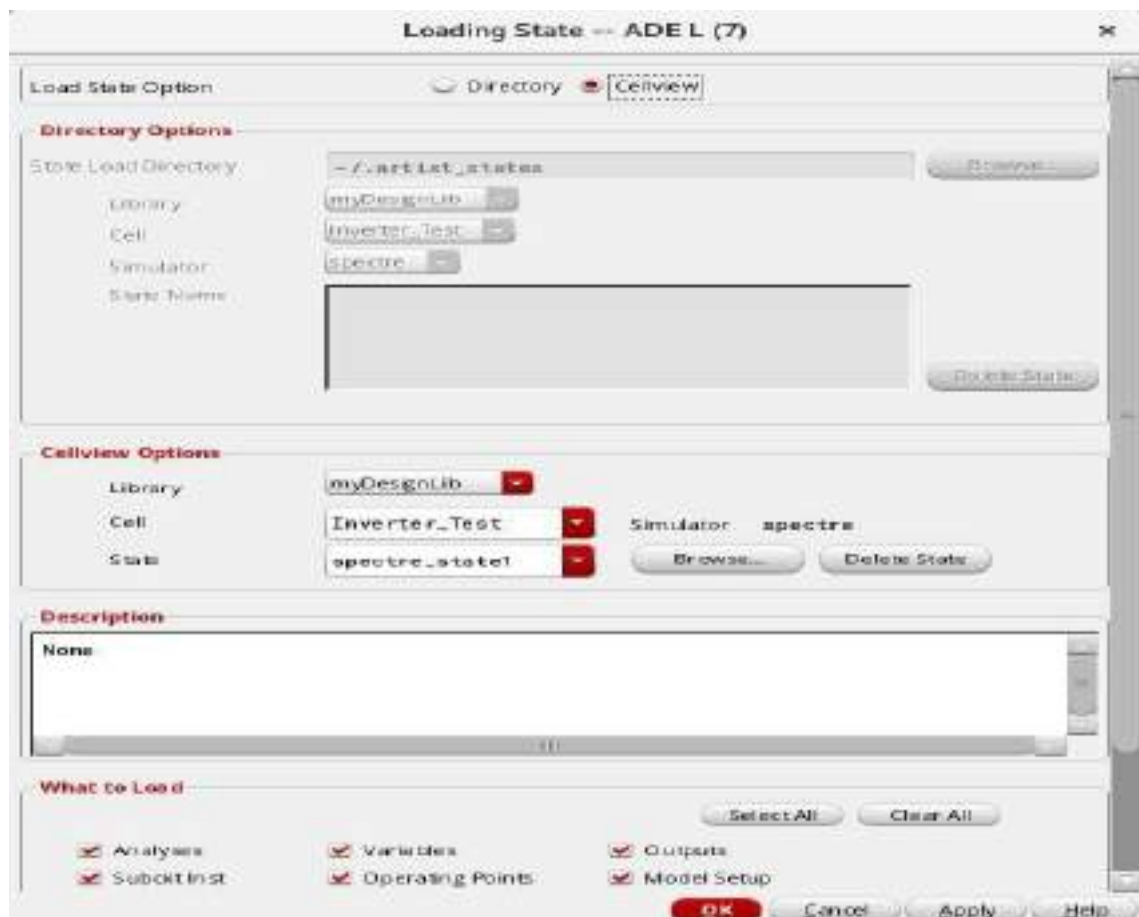
This information restores the simulation environment without having to type in all of setting again.

1. In the Simulation window, execute **Session – Save State**. The Saving State form appears.
2. Set the **Save as field** to **state1\_inv** and make sure all options are selected under what to save field.
3. Click **OK** in the saving state form. The Simulator state is saved.

### Loading the Simulator State:

1. From the ADE window execute **Session – Load State**.
2. In the Loading State window, set the State name to **state1\_inv** as shown

3. Click **OK** in the Loading State window.



### Viva-Voce Questions

1. Why is a CMOS inverter called a complementary circuit?
2. What is the voltage transfer characteristic (VTC) of a CMOS inverter?
3. Why is the static power consumption of CMOS very low?
4. What is propagation delay in an inverter?
5. Why is transistor sizing important in CMOS design?

### Result :

Thus the CMOS Inverting Amplifier is Designed Simulated and Analyze the Input impedance, Output impedance, Gain and Bandwidth.

## **Expt.No.12.**

### **Design of Differential Amplifier (5 Transistor )**

#### **Objective:**

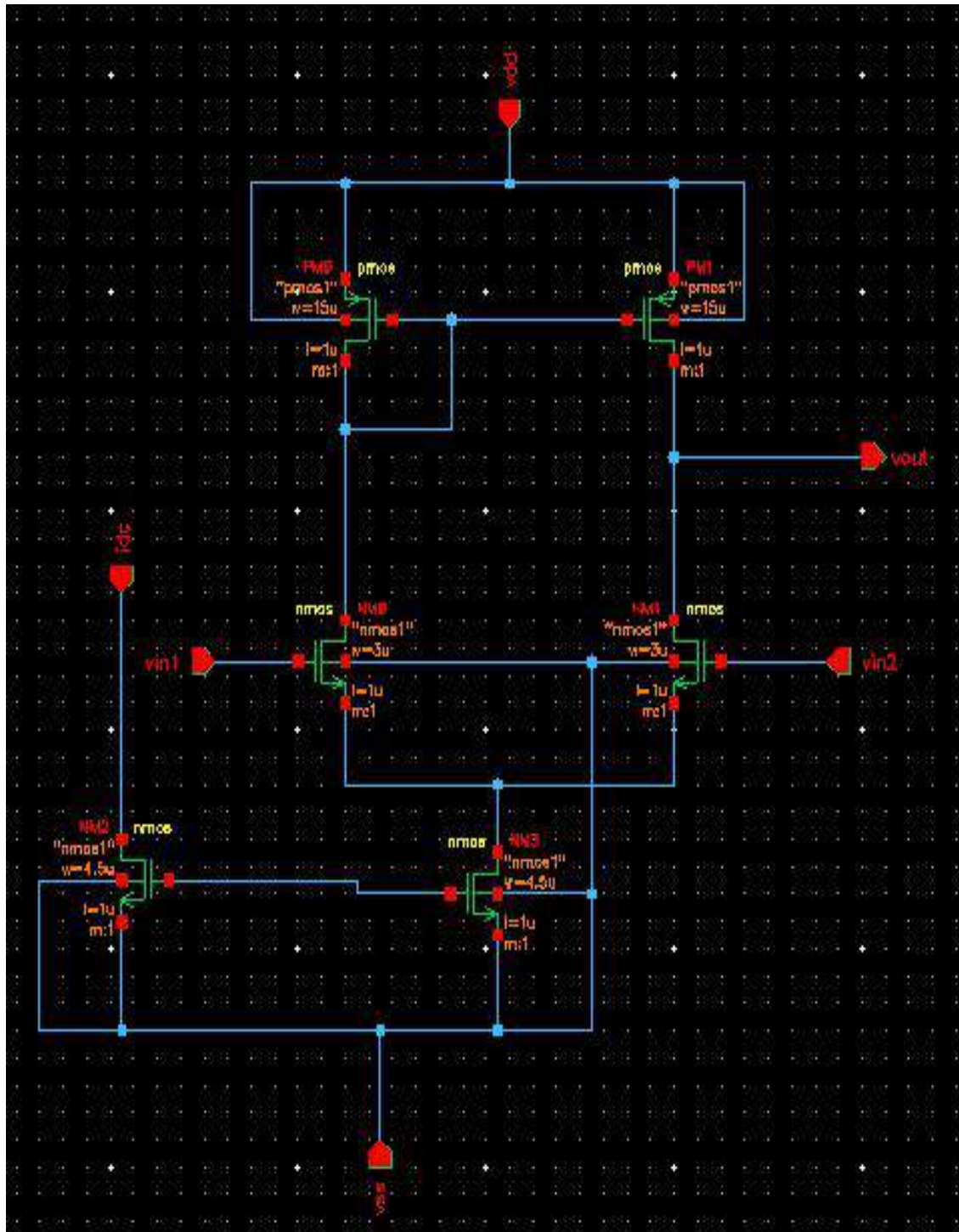
- To set up and run simulations on the Differential Amplifier Test design.
- In this section, we will run the simulation for Differential Amplifier and plot the Transient, DC and AC characteristics.
- Analyzing Gain, Bandwidth and CMRR by performing schematic simulations.

#### **Theory:**

The differential amplifier is probably the most widely used circuit building block in analog integrated circuits, principally op amps. The differential amplifier can be implemented with BJTs or MOSFET's.

A differential amplifier multiplies the voltage difference between two inputs ( $V_{in+} - V_{in-}$ ) by some constant factor  $A_d$ , the differential gain. It may have either one output or a pair of outputs where the signal of interest is the voltage difference between the two outputs. A differential amplifier also tends to reject the part of the input signals that are common to both inputs  $(V_{in+} + V_{in-})/2$ . This is referred to as the common mode signal.

## Schematic Capture of Differential Amplifier:



## 1: Schematic Entry

Objective: To create a new cell view and build Differential Amplifier

### Creating a Schematic cellview:

Open a new schematic window in the **myDesignLib** library and build the Diff\_amplifier design.

1. In the CIW or Library manager, execute **File – New – Cellview**. Set up the Create New file form as follows:



3. Click OK when done. A blank schematic window for the design appears.

### Adding Components to schematic:

1. In the Differential Amplifier schematic window, execute **Create— Instance** to display the Add Instance form.
2. Click on the **Browse** button. This opens up a Library browser from which you can select components and the **Symbol** view. You will update the Library Name, Cell Name, and the property values given in the table on the next page as you place each component.
3. After you complete the Add Instance form, move your cursor to the schematic window and click

**left** to place a component.

This is a table of components for building the Differential Amplifier schematic.

| Library Name | Cell Name | Properties/Comments                           |
|--------------|-----------|-----------------------------------------------|
| gpd180       | nmos      | Model name = nmos1, (NM0, NM1); W= 3u; L=1u   |
| gpd180       | nmos      | Model name = nmos1, (NM2, NM3); W= 4.5u; L=1u |
| gpd180       | pmos      | Model name = pmos1, (PM0, PM1); W= 15u; L=1u  |

3. After entering components, click Cancel in the Add Instance form or press Esc with your cursor in the schematic window

### Adding pins to Schematic:

Use **Create – Pin** or the menu icon to place the pins on the schematic window.

1. Click the **Pin** fixed menu icon in the schematic window.

You can also execute **Create – Pin** or **press p**. The Add pin form appears.

2. Type the following in the Add pin form in the exact order leaving space between the pin names.

| Pin Names                 | Directions |
|---------------------------|------------|
| vin1, vin2, Idc, vdd, vss | input      |
| vout                      | output     |

Make sure that the direction field is set to **input/output/inputoutput** when placing the **input/output/inout** pins respectively and the Usage field is set to schematic.

3. Select **Cancel** from the Add pin form after placing the pins. In the schematic window, execute **View — Fit** or press the **f** bindkey.

### Adding Wires to a Schematic:

Add wires to connect components and pins in the design.

1. Click the **Wire (narrow)** icon in the schematic window.

You can also press the **w** key, or execute **Create - Wire (narrow)**.

2. Complete the wiring as shown in figure and when done wiring press ESC key in the schematic window to cancel wiring.

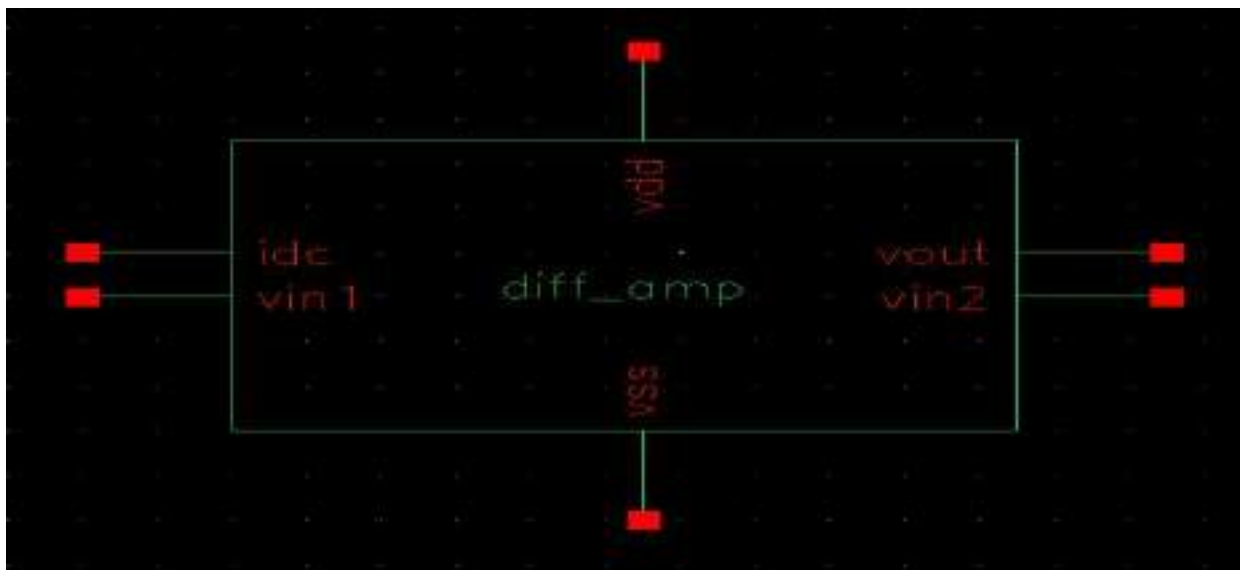
### Saving the Design:

1. Click the Check and **Save** icon in the schematic editor window.
2. Observe the **CIW** output area for any errors.

### Symbol Creation:

Objective: To create a symbol for the Differential Amplifier

1. In the Differential Amplifier schematic window, execute **Create — Cellview— From Cellview**. The Cellview from Cellview form appears. With the Edit Options function active, you can control the appearance of the symbol to generate.
2. Verify that the **From View Name** field is set to **schematic**, and the To View Name field is set to **symbol**, with the Tool/Data Type set as **SchematicSymbol**.
3. Click **OK** in the Cellview from Cellview form. The **Symbol Generation Form** appears.
4. Modify the Pin Specifications as in the below symbol.
5. Click **OK** in the Symbol Generation Options form.
6. A new window displays an automatically created Differential Amplifier symbol.
7. Modifying automatically generated symbol so that it looks like below Differential Amplifier symbol.
8. Execute **Create— Selection Box**. In the Add Selection Box form, click **Automatic**. A new red selection box is automatically added.



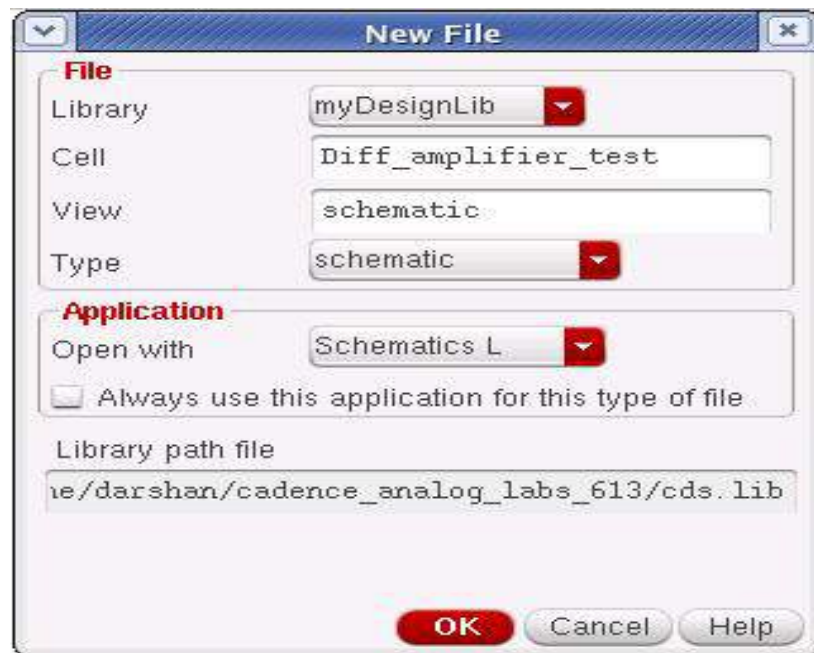
9. After creating symbol, click on the **save** icon in the symbol editor window to save the symbol. In the symbol editor, execute **File— Close** to close the symbol view window.

## 2: Building the Diff\_amplifier\_test Design

Objective: To build Differential Amplifier Test circuit using your Differential Amplifier

### Creating the Differential Amplifier Test Cellview:

1. In the CIW or Library Manager, execute **File— New— Cellview**.
2. Set up the Create New File form as follows:



3. Click **OK** when done. A blank schematic window for the Diff\_amplifier\_test design appears.

### Building the Diff\_amplifier\_test Circuit:

1. Using the component list and Properties/Comments in this table, build the Diff\_amplifier\_test schematic.

| Library name | Cellview name  | Properties/Comments                                                                                  |
|--------------|----------------|------------------------------------------------------------------------------------------------------|
| myDesignLib  | Diff_amplifier | Symbol                                                                                               |
| analogLib    | vsin           | Define specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude= 5m; Frequency= 1K |
| analogLib    | vdd, vss, gnd  | vdc=2.5; vdc=-2.5;                                                                                   |
| analogLib    | Idc            | DC current=30u                                                                                       |

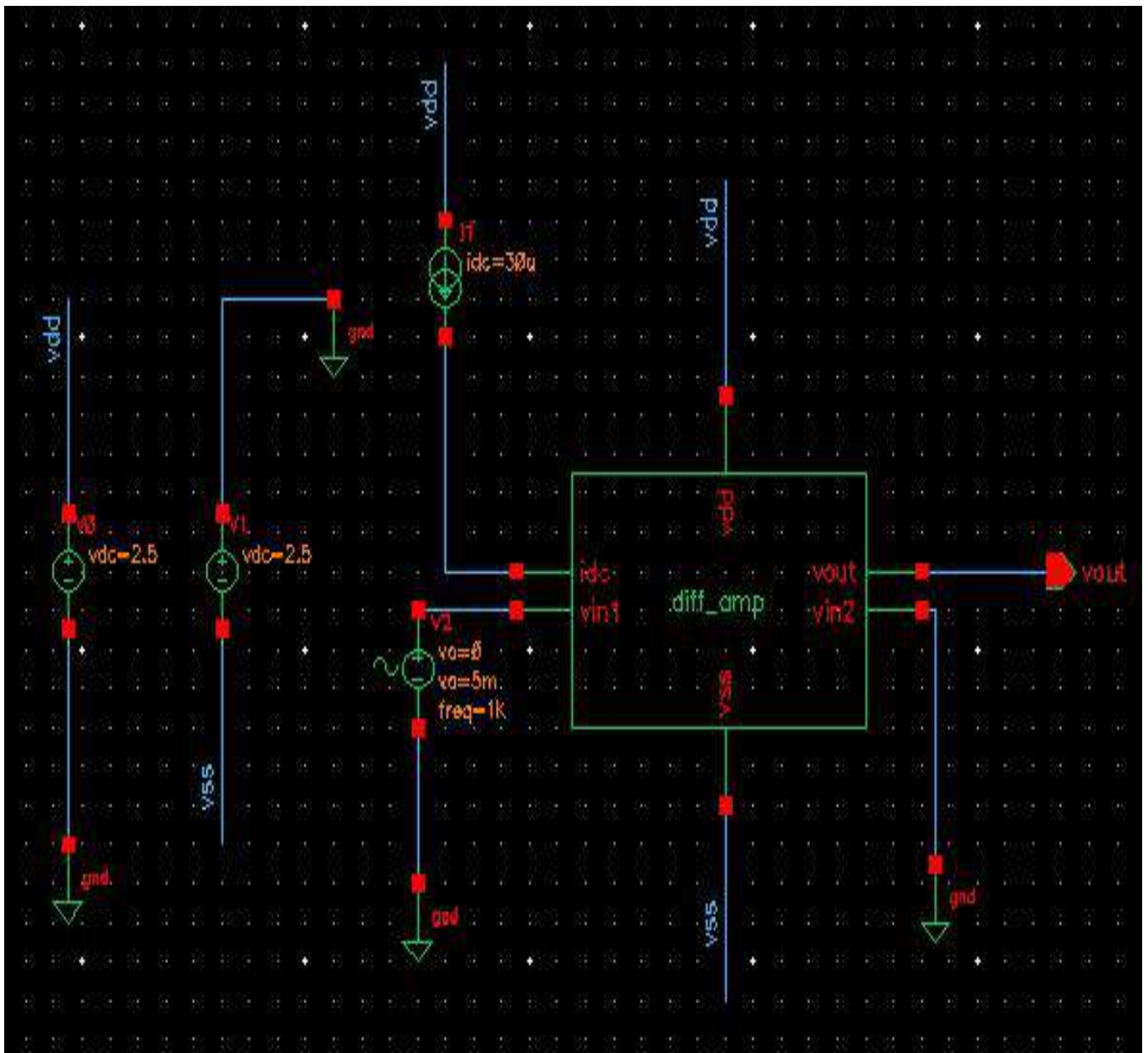
3. Click the Wire (narrow) icon and wire your schematic.

Tip: You can also press the w key, or execute Create— Wire (narrow).

4. Click on the Check and save icon to save the design.

5. The schematic should look like shown below.

Schematic Capture of Diff\_amplifier\_test Circuit:



### 3: Analog Simulation with Spectre:

#### Objective:

- To set up and run simulations on the Differential Amplifier Test design.
- In this section, we will run the simulation for Differential Amplifier and plot the transient, DC and AC characteristics.
- Analyzing Gain, Bandwidth and CMRR by performing Schematic Simulations

#### Starting the Simulation Environment:

1. In the Diff\_amplifier\_test schematic window, execute **Launch – ADE L**. The Analog Design Environment simulation window appears.

#### Choosing a Simulator:

1. In the simulation window or ADE, execute **Setup— Simulator/Directory/Host**.
2. In the **Choosing Simulator** form, set the Simulator field to **spectre** (Not spectreS) and click **OK**.

#### Setting the Model Libraries:

1. Click **Setup - Model Libraries**.

Note: Step 2 should be executed only if the model file not loaded by default.

2. In the Model Library Setup form, click **Browse** and find the **gpd180.scs** file in the **./models/spectre** directory. Select **NN** in Section field, click **Add** and click **OK**.

#### Choosing Analyses:

1. In the Simulation window, click the **Choose - Analyses** icon.

You can also execute **Analyses - Choose**.

The Choosing Analysis form appears. This is a dynamic form, the bottom of the form changes based on the selection above.

2. To setup for transient analysis
  - a. In the Analysis section select **tran**
  - b. Set the stop time as **10m**
  - c. Click at the **moderate** or Enabled button at the bottom, and then click **Apply**.
3. To set up for DC Analyses:
  - a. In the Analyses section, select **dc**.
  - b. In the DC Analyses section, turn on Save DC Operating Point.
  - c. Turn on the **Component Parameter**
  - d. Double click the Select Component, Which takes you to the schematic window.
  - e. Select input signal **Vsin** for **dc** analysis.
  - f. In the analysis form, select **start** and **stop** voltages as **-5** to **5** respectively.
  - g. Check the enable button and then click **Apply**.
4. To set up for AC Analyses form is shown in the previous page.
  - a. In the Analyses section, select **ac**.
  - b. In the AC Analyses section, turn on **Frequency**.
  - c. In the Sweep Range section select **start** and **stop** frequencies as **150** to **100M**

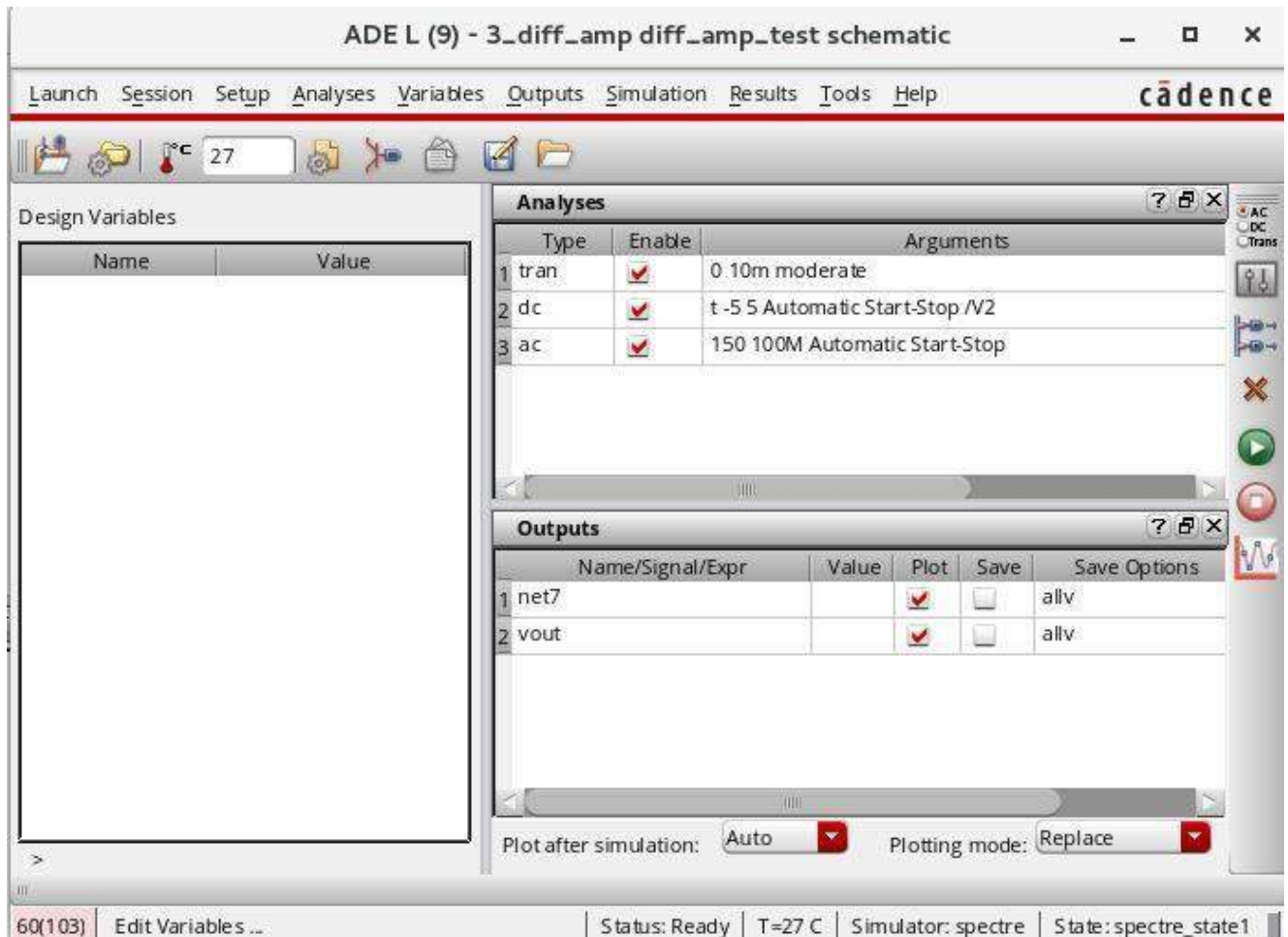
- d. Sweep type **Automatic**.
- e. Check the enable button and then click **Apply**.
5. Click **OK** in the Choosing Analyses Form

### Selecting Outputs for Plotting:

Select the nodes to plot when simulation is finished.

1. Execute **Outputs – To be plotted – Select on Schematic** in the simulation window.
2. Follow the prompt at the bottom of the schematic window, Click on output net **Vo**, input net **Vin** of the Diff\_amplifier. Press **ESC** with the cursor in the schematic after selecting node.

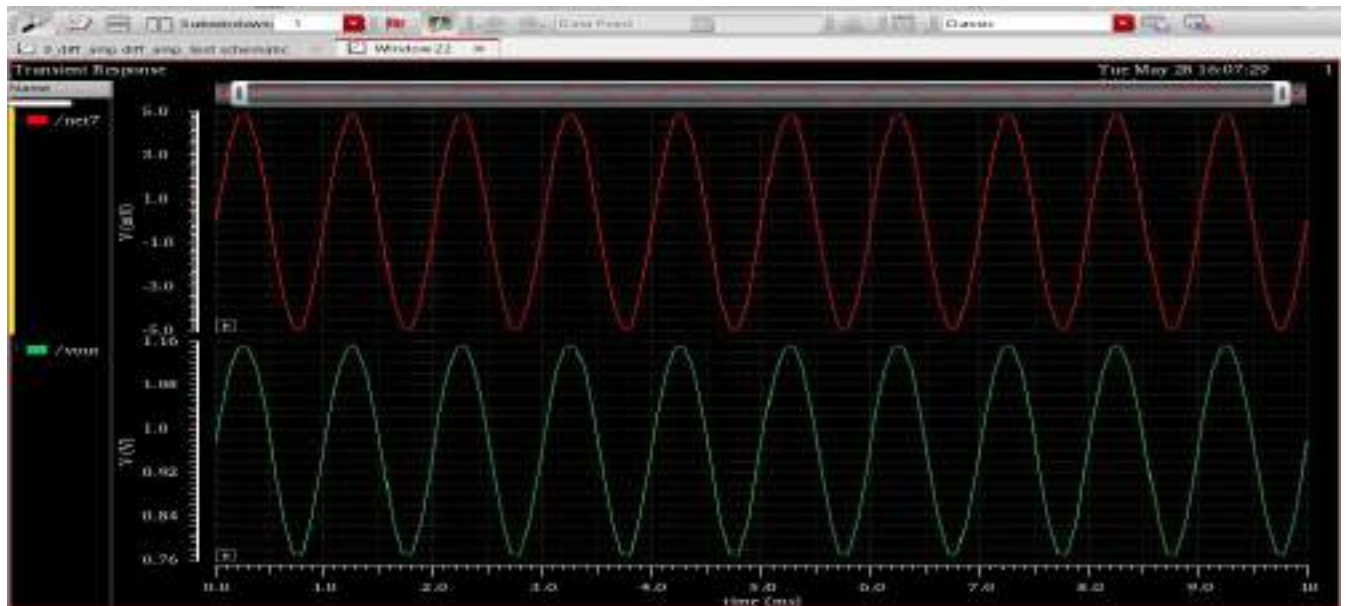
### Settings in ADE L window:



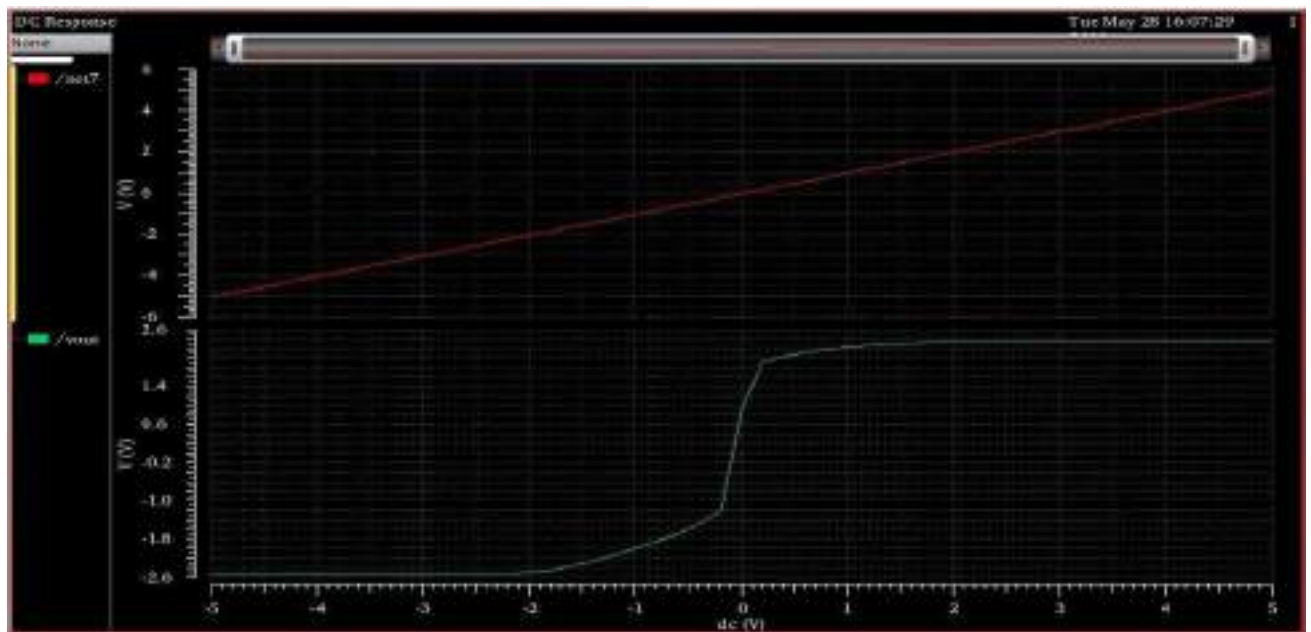
### Running the Simulation:

1. Execute **Simulation – Netlist and Run** in the simulation window to start the simulation, this will create the netlist as well as run the simulation.
2. When simulation finishes, the Transient, DC and AC plots automatically will be popped up along with netlist.

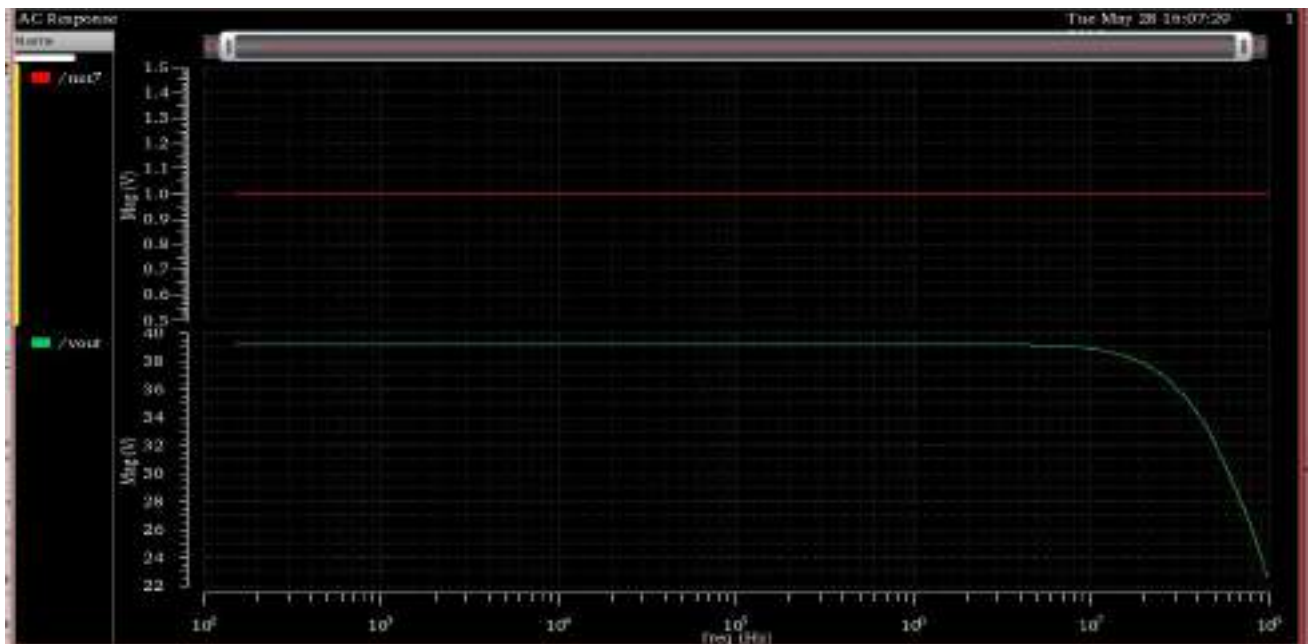
## Transient Response:



## DC Response:



## AC Response:

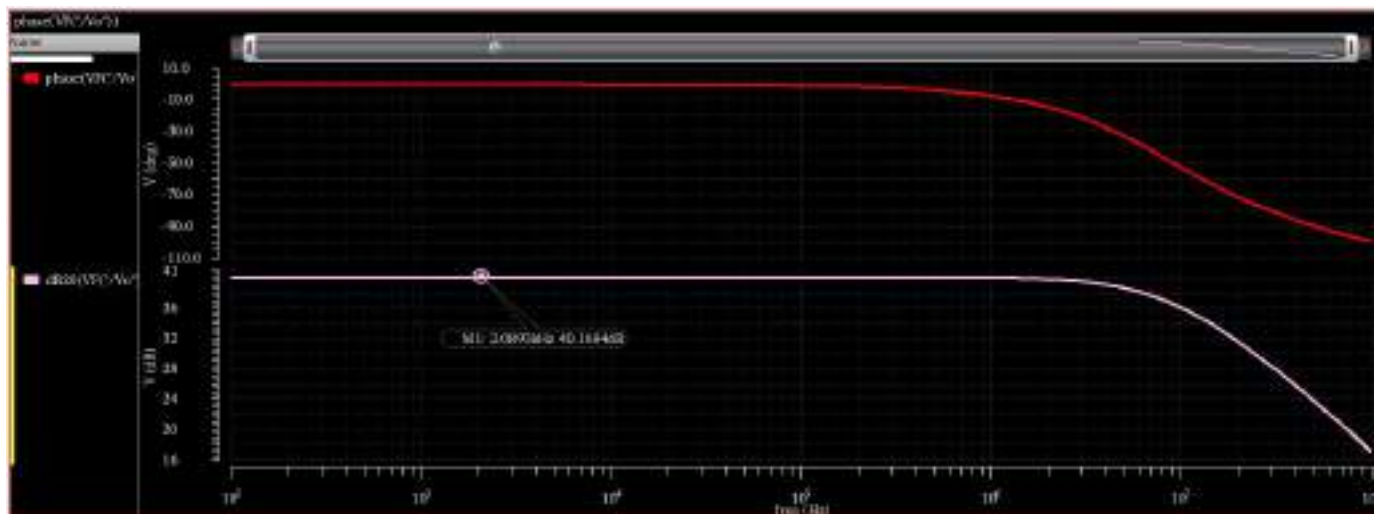


## 4: Gain Calculation

To Calculate the Gain of Differential Pair:- Configure the Differential pair schematic as shown below

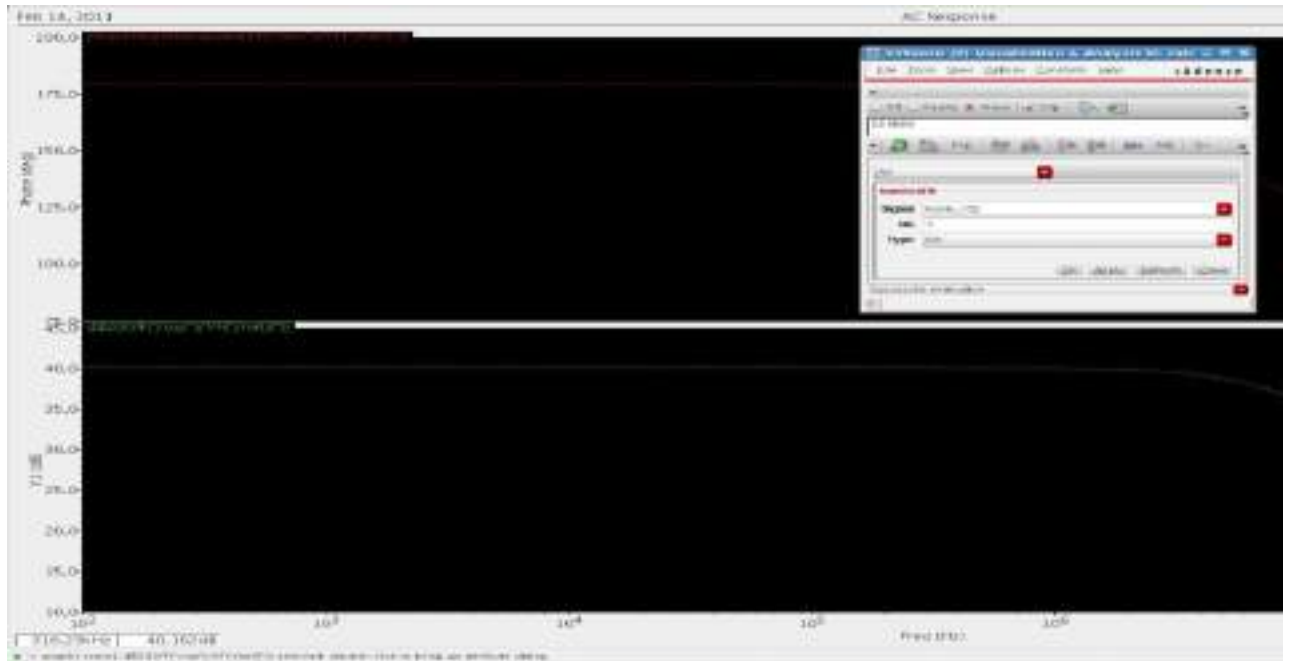
1. Schematic for Differential Gain:

Differential mode Gain ( $A_d$ ):



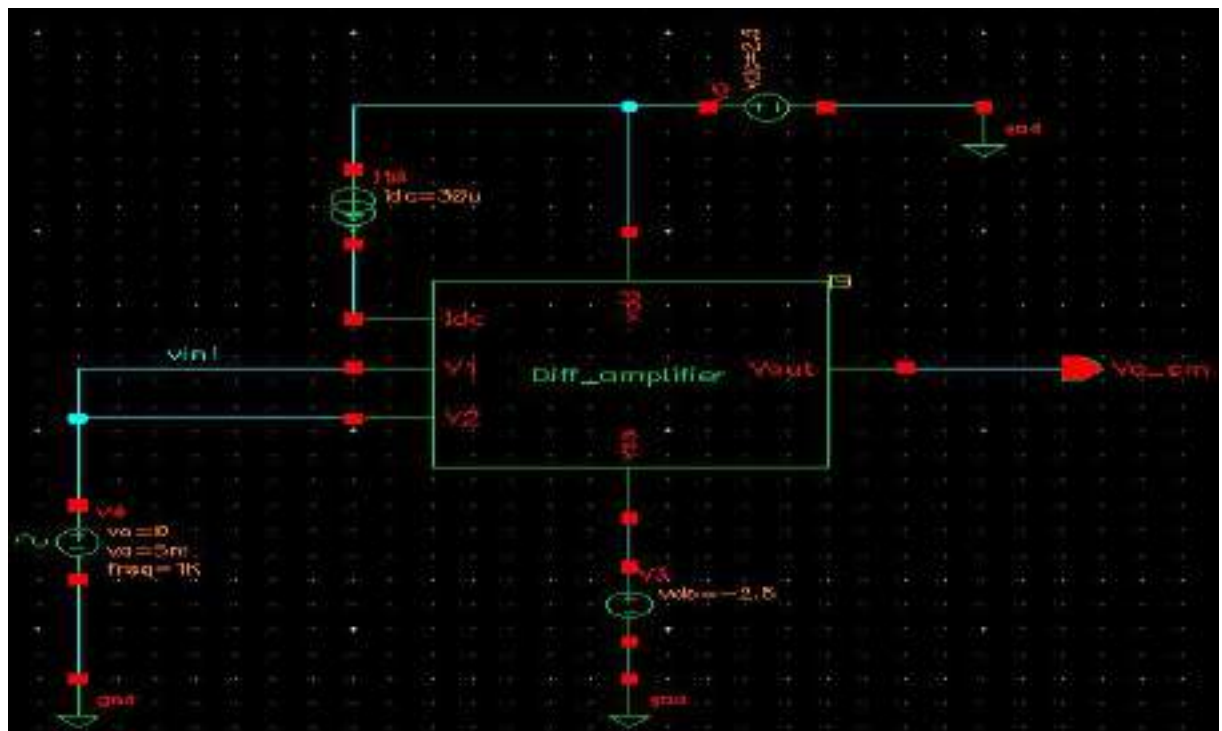
## 5: Bandwidth Calculation

To Calculate the BW of the Differential pair: Open the calculator and select the bandwidth option, select the waveform of the gain in dB and press Evaluate the buffer -

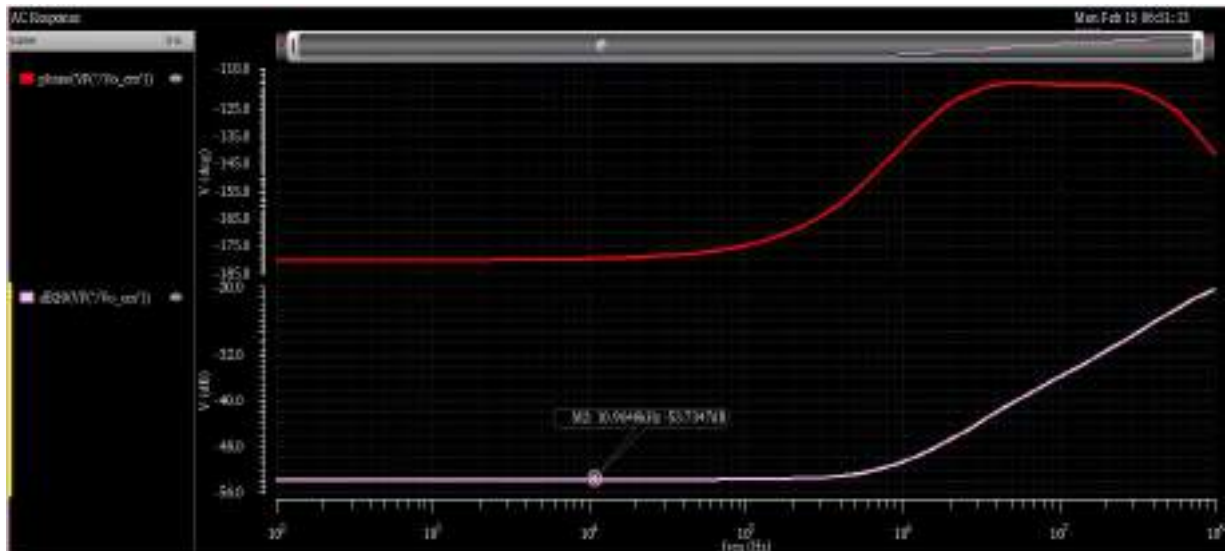


To Calculate the Gain of Common Mode Pair:- Configure the Differential pair schematic as shown below

## 2. Schematic for Common Mode Gain:



Common mode Gain (Ac):



## 6: Common Mode Rejection Ratio Calculation:

To measure CMRR, subtract Common Mode Gain from Differential mode Gain

$$\text{CMRR} = A_d / A_c$$

## Viva-Voce Questions

1. What is a differential amplifier?
2. Why is differential gain important?
3. What is Common Mode Rejection Ratio (CMRR)?
4. Why is a current source used in a differential amplifier?
5. Mention one application of differential amplifiers.

## Result:

Thus Differential Amplifier using five transistor is designed, simulated and functionality is verified successfully.

## **Content Beyond Syllabus Experiment**

### **Experiment No. 13**

#### **Design and Layout of a 6-Transistor (6T) CMOS SRAM Memory Cell Using Cadence Virtuoso**

##### **Aim**

To design, simulate, and generate the manual/automatic layout of a 6-Transistor (6T) CMOS Static Random Access Memory (SRAM) Cell using Cadence Virtuoso and verify its Read, Write, and Hold operations.

##### **Objectives**

To understand the architecture of a CMOS SRAM cell.

To design a 6T SRAM memory cell using CMOS transistors.

To perform schematic capture using Cadence Virtuoso.

To verify Read, Write and Hold operations through simulation.

To generate manual/automatic layout.

To perform Design Rule Check (DRC), Layout Versus Schematic (LVS), and Parasitic Extraction (PEX).

##### **Theory**

Static Random Access Memory (SRAM) is one of the most important memory circuits used in modern VLSI systems. Unlike DRAM, SRAM does not require periodic refreshing because it stores data using bistable latches.

A 6T SRAM cell consists of

Two CMOS Inverters connected back-to-back

Two NMOS access transistors

The access transistors connect the storage nodes to the Bit Lines (BL and BLB) when the Word Line (WL) is enabled.

The SRAM cell operates in three modes:

Hold Mode

WL = 0

Access transistors OFF

Data retained indefinitely.

Write Mode

WL = 1

BL and BLB driven with complementary data

New data written into the latch.

Read Mode

WL = 1

BL and BLB precharged

Stored data sensed through bit lines.

### **Software Required**

Cadence Virtuoso

Spectre Simulator

Assura/PVS

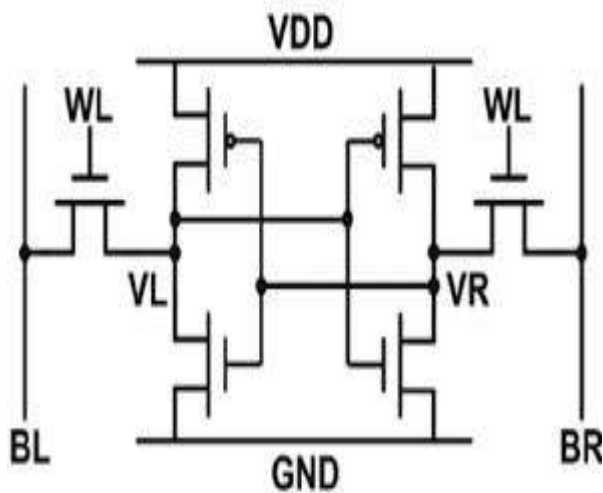
Technology Library (180 nm / 90 nm / 45 nm)

Circuit Components

—

| Component           | Quantity |
|---------------------|----------|
| PMOS                | 2        |
| NMOS                | 4        |
| VDD                 | 1        |
| Ground              | 1        |
| Bit Lines (BL, BLB) | 2        |
| Word Line (WL)      | 1        |

## Circuit Diagram



## Design Procedure

### Part A – Schematic Design

Open Cadence Virtuoso.

Create a new library.

Attach the technology library.

Create a new Cell View (Schematic).

Place:

2 PMOS

4 NMOS

Connect two CMOS inverters back-to-back.

Add two access transistors.

Connect BL, BLB and WL.

Add VDD and Ground.

Check and Save the schematic.

### Part B – Simulation

Hold Operation

WL = 0

Verify stored data remains unchanged.

Write Operation

WL = 1

BL = 1

BLB = 0

Verify logic '1' stored.

Repeat for

BL = 0

BLB = 1

Verify logic '0' stored.

Read Operation

Precharge BL and BLB.

Enable WL.

Observe voltage difference.

Verify correct data retrieval.

## Part C – Layout Design

Create Layout View.

Place PMOS in N-Well.

Place NMOS in P-Substrate.

Draw Active regions.

Draw Poly Gates.

Create Source/Drain Diffusions.

Add Contacts.

Route Metal1.

Route Metal2 if required.

Connect VDD and GND rails.

Save Layout.

## Part D – Verification

### Perform

Design Rule Check (DRC)

Layout Versus Schematic (LVS)

Parasitic Extraction (PEX)

### Ensure

No DRC Errors

LVS Matched

Functional Simulation Successful

### Observation

SRAM cell successfully stores one bit.

Hold operation retains data.

Read operation retrieves stored data.

Write operation updates stored data.

Layout passes DRC and LVS verification.

## Viva-Voce Questions

1. Why does a standard SRAM cell use six transistors (6T) instead of four?
2. Explain the Read, Write, and Hold operations of a 6T SRAM cell.
3. What are the functions of the Word Line (WL) and Bit Lines (BL and BLB)?
4. Why are DRC, LVS, and Parasitic Extraction (PEX) performed after layout generation?
5. What are the advantages of SRAM over DRAM, and where is SRAM commonly used?

### Result

A 6-Transistor CMOS SRAM Cell was successfully designed, simulated, and verified using Cadence Virtuoso. The manual/automatic layout was generated, DRC and LVS checks were completed successfully, and the Read, Write, and Hold operations were verified through simulation.