SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM NAGAR, KATTANKULATHUR – 603 203.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC3566 – VLSI LABORATORY MANUAL

III YEAR- V SEMESTER

Regulation 2023

Academic Year 2025– 2026(Odd semester)

Prepared by:

Dr. C. Saravanakumar, AP (Sel.G) / ECE

Mr. S. Senthilmurugan, AP (Sel.G) / ECE

Mr. A. Pandian, AP(Sr.G) / ECE

LIST OF EXPERIMENTS

Digital System Design using HDL & FPGA

1. Design an Adder (Min 8 Bit) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.
2. Design a Multiplier (4 Bit Min) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.
3. Design an ALU using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
4. Design an ALU using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.
5. Design a Universal Shift Register using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.
6. Design Finite State Machine (Moore/Mealy) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.
7. Design Memories using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA.

Digital Circuit Design

8. Design and simulate a CMOS inverter using digital flow.
9. Design and simulate a CMOS Basic Gates & Flip-Flops.
10. Design and simulate a 4-bit synchronous counter using a Flip-Flops.

Analog Circuit Design

11. Design and Simulate a CMOS Inverting Amplifier.
12. Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.
13. Design and simulate simple 5 transistor differential amplifier. Analyze Gain, Bandwidth and CMRR by performing Schematic Simulations.

Expt No.1

Design and Implementation of 8 Bit Adder Using HDL

Aim:

To design, simulate and implement8 bit adder 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:

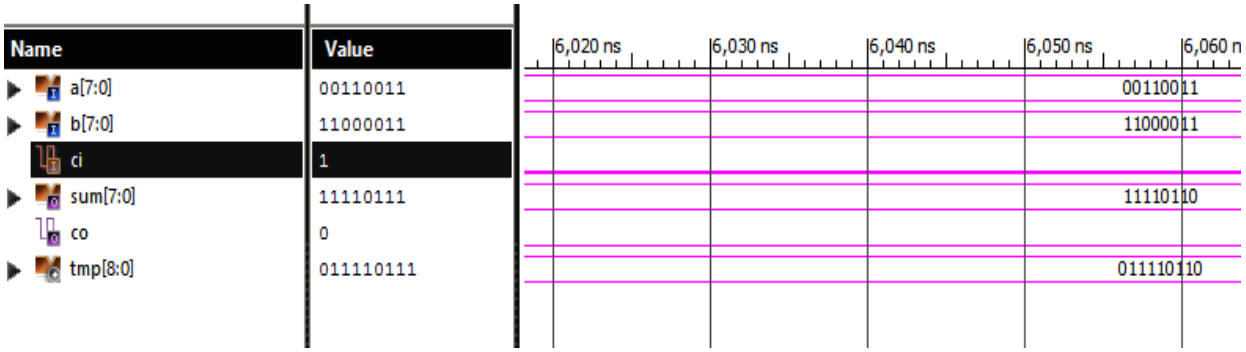
8 BIT ADDER

```
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
```

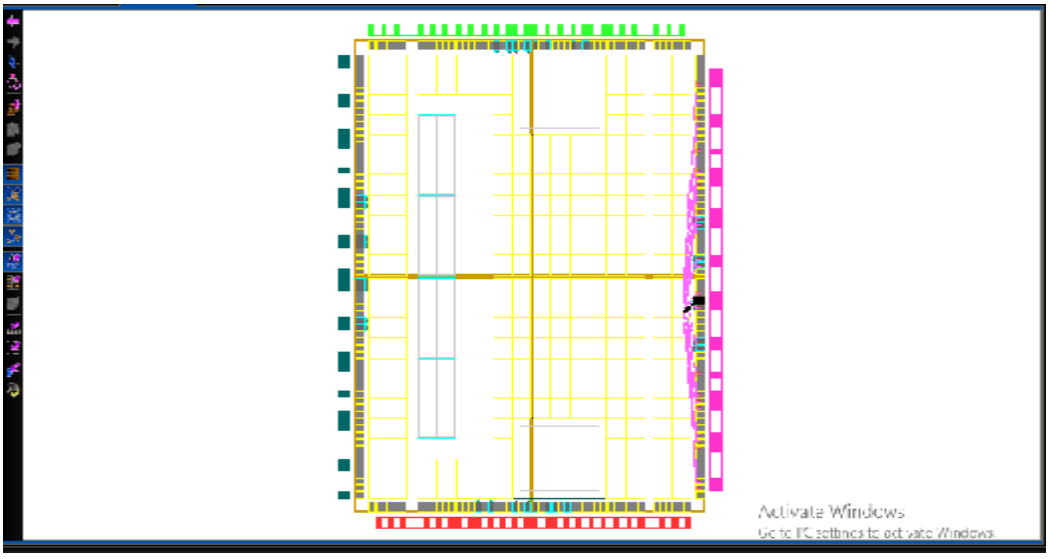
```
module fulladd(s,cout,a,b,cin);
output s,cout;
```

```
input a,b,cin; xor
(s,a,b,cin);
assign cout = ((a & b )|(b& cin)|( a & cin)) ;
endmodule
```

Simulation output:



Floor Plan:



Result:

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

Expt No.2

Design and Implementation of 4 Bit Multiplier Using HDL

Aim:

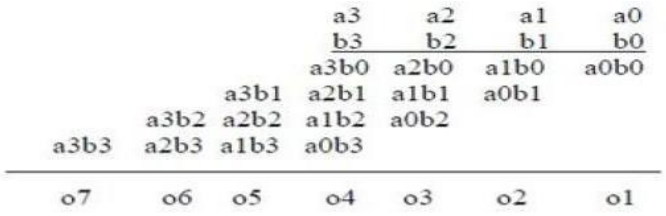
To design, simulate and implement 4 bit Multiplier using Xilinx simulator and implement using Spartan 3e FPGA

Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

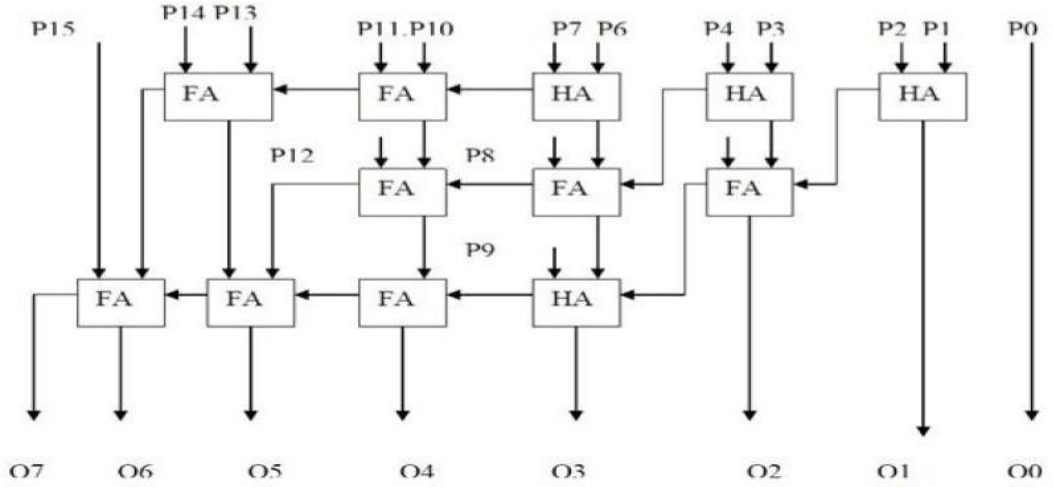


a0b0 = p0
a1b0 = p1
a0b1 = p2
a2b0 = p3

a1b2 = p8
a0b3 = p9
a3b1 = p10
a2b2 = p11

a1b1 = p4
a0b2 = p5
a3b0 = p6
a2b1 = p7

a1b3 = p12
a3b2 = p13
a2b3 = p14
a3b3 = p15



Procedure:

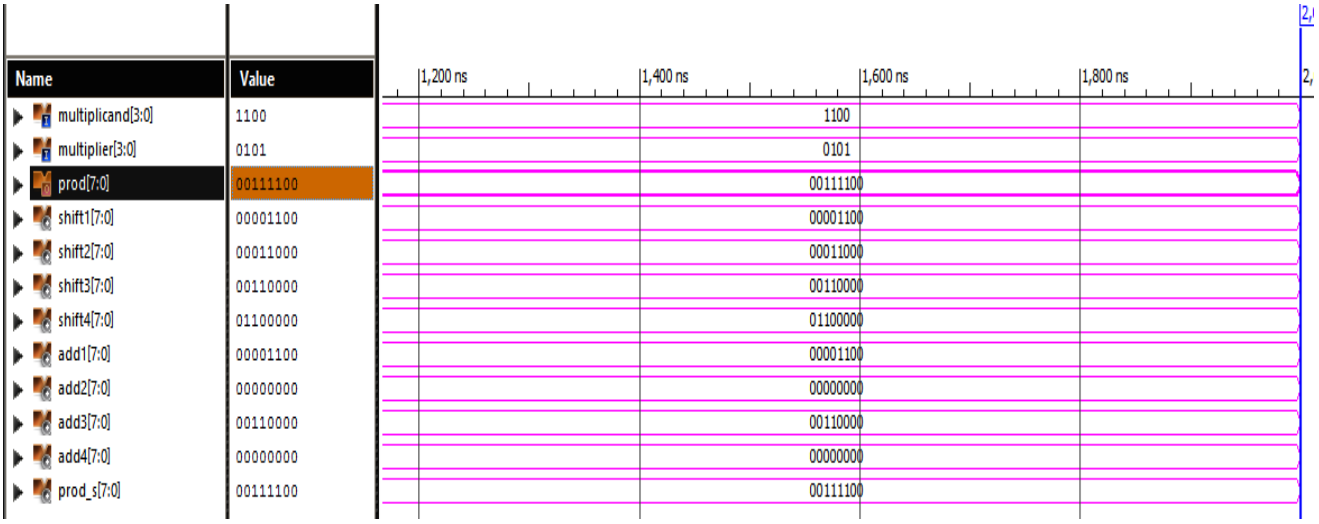
- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

Program:

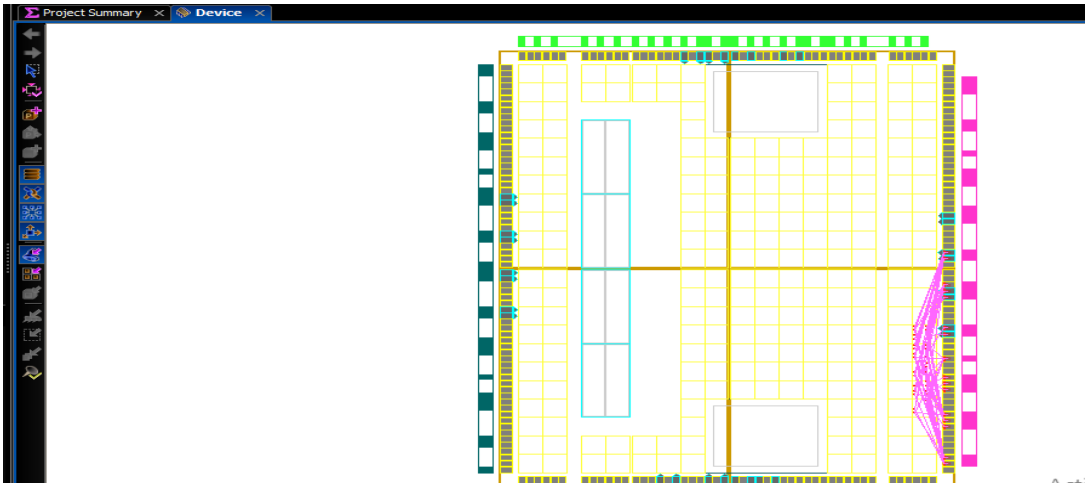
```
module fourbitmulti(m,a,b);
input[3:0]a;
input[3:0]b;
output[7:0]m;
wire[15:0]p;
wire[12:1]s;
wire[12:1]c;



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:



Result:

Thus the program to simulate 4 bit multiplier using Xilinx tools is executed and output is verified successfully.

Expt No.3

Design and Implementation of ALU Using HDL

Aim:

To design, simulate and implement ALU using Xilinx.

Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

Procedure:

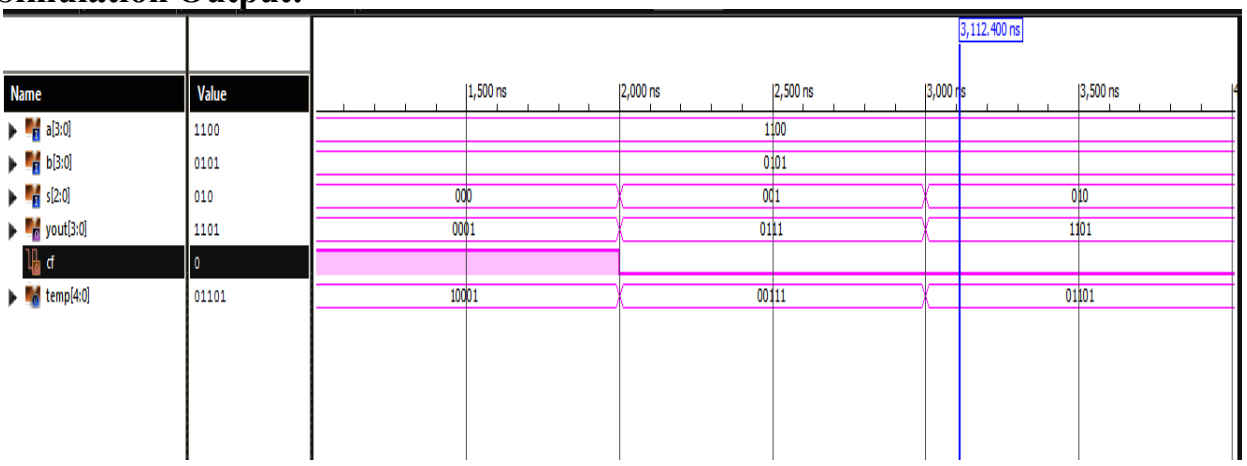
- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

Program:

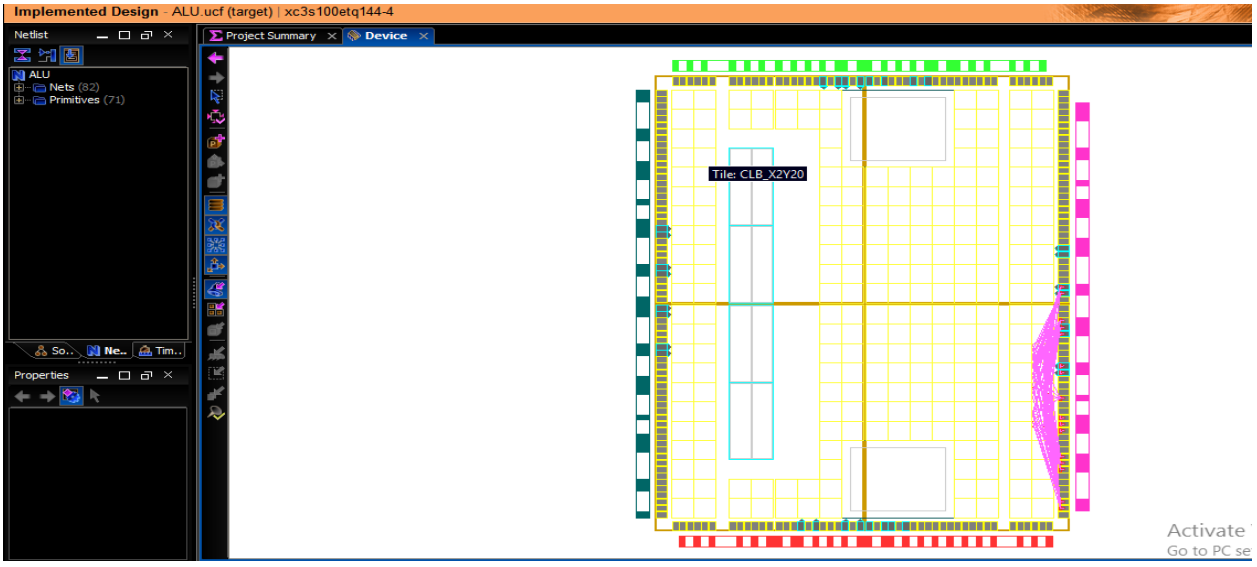
```
module alu(s,a,b,f);
input[2:0]s;
input[3:0]a,b;
output[3:0]f;
reg[3:0]f;
always@(s or a or b)
begin
case(s)
3'b000:f<=4'b0000;
3'b001:f<=a-b;
3'b010:f<=a+b;
3'b011:f<=b-a;
3'b100:f<=a&b;
3'b101:f<=a|b;
```

```
3'b110:f<=~a;
default:f<=4'b1111;
endcase
end
endmodule
```

Simulation Output:



Floor Plan:



Result

Thus the program to simulate 4 bit ALU using Xilinx tools is executed and output is verified successfully.

Expt No.4 Design and Implementation of Code converter Using HDL

Aim:

To design, simulate and implement binary to gray and gray to binary code converters using Xilinx.

Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

Procedure:

Start the Xilinx project Navigator.

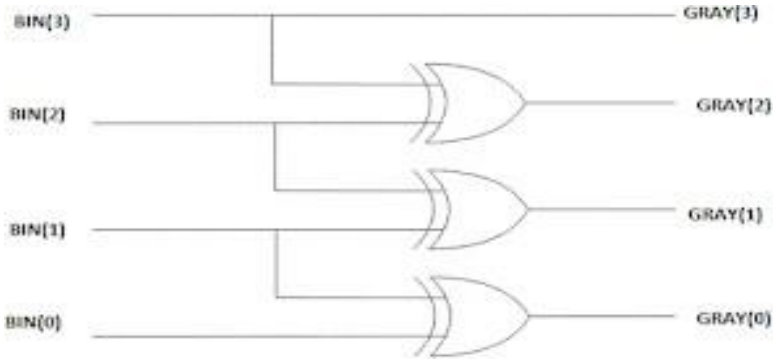
Create a new project and new source.

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

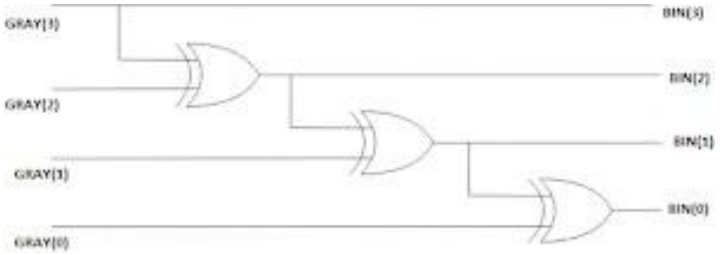
Assign the package pins for input and output using XILINX plan ahead.

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

Logic circuit for 4 bit Binary to Gray code converter



Logic circuit for 4 bit Gray code to Binary converter:



Program:

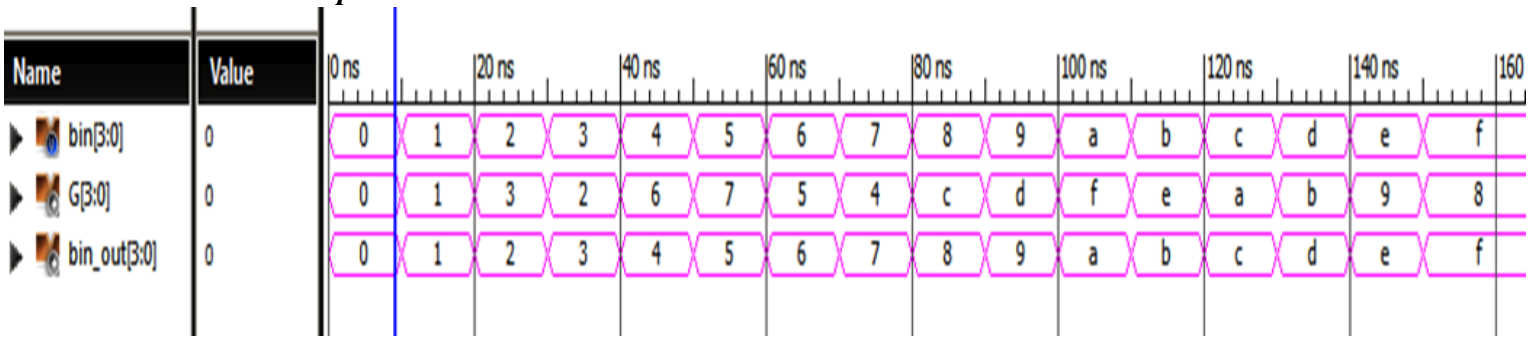
Verilog Code for Binary to Gray code conversion:

```
module bin2gray
   (input [3:0] bin, //binary input
    output [3:0] G //gray code output
   );

//xor gates.
assign G[3] = bin[3];
assign G[2] = bin[3] ^ bin[2];
assign G[1] = bin[2] ^ bin[1];
assign G[0] = bin[1] ^ bin[0];

endmodule
```

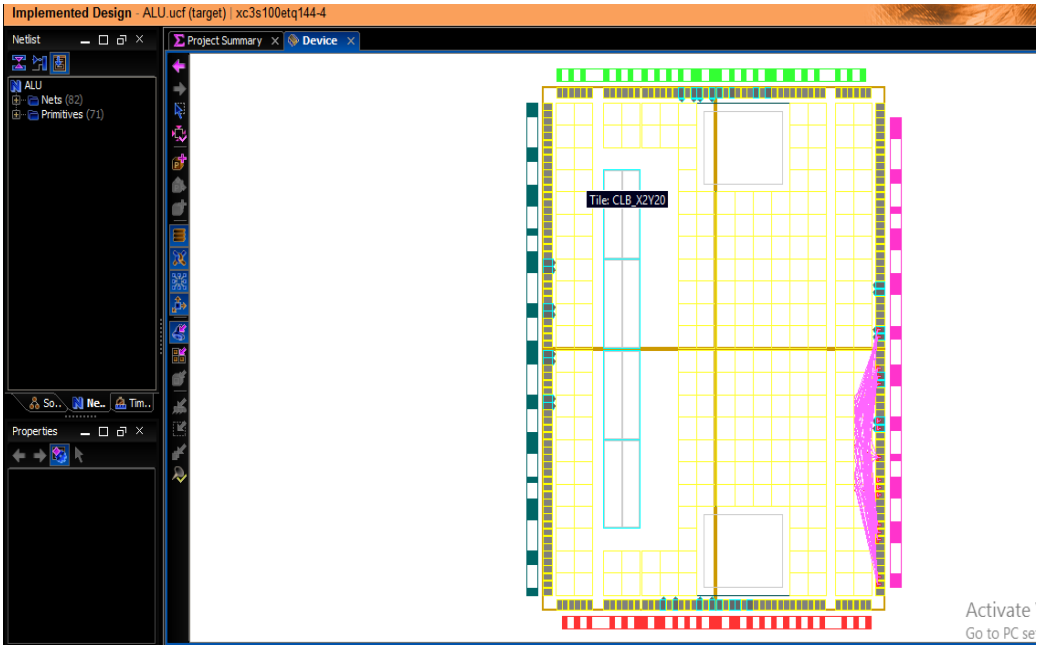
Verilog Code for Gray code to Binary conversion:

```
module gray2bin
   (input [3:0] G, //gray code output
    output [3:0] bin  //binary input
   );

assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];

endmodule
```

simulation output



Floor Plan:



Result

Thus the program to simulate 4 bit code converter using Xilinx tools is executed and output is verified successfully.

Expt No. 5

Design and Implementation of Universal Shift Register Using HDL

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.

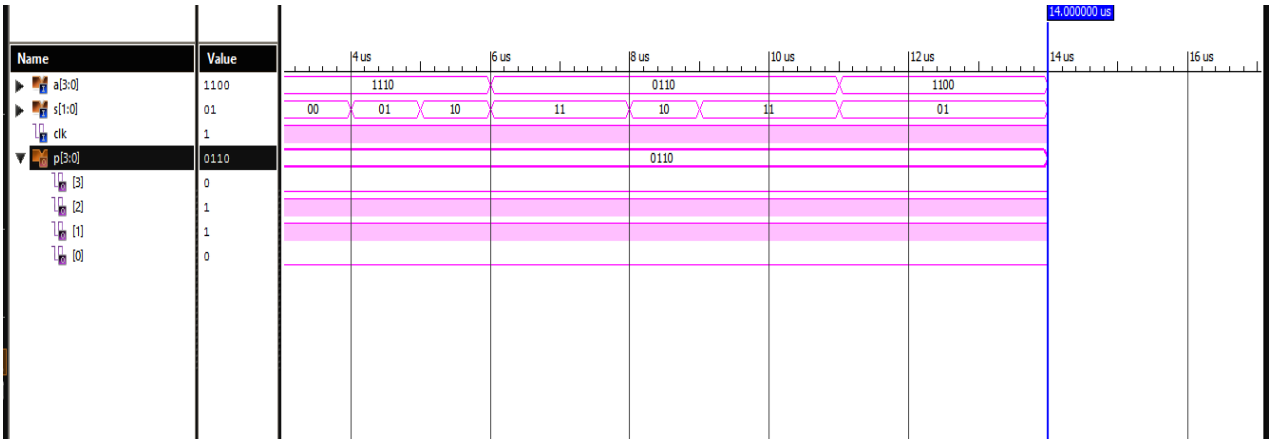
Program:

```
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];
```

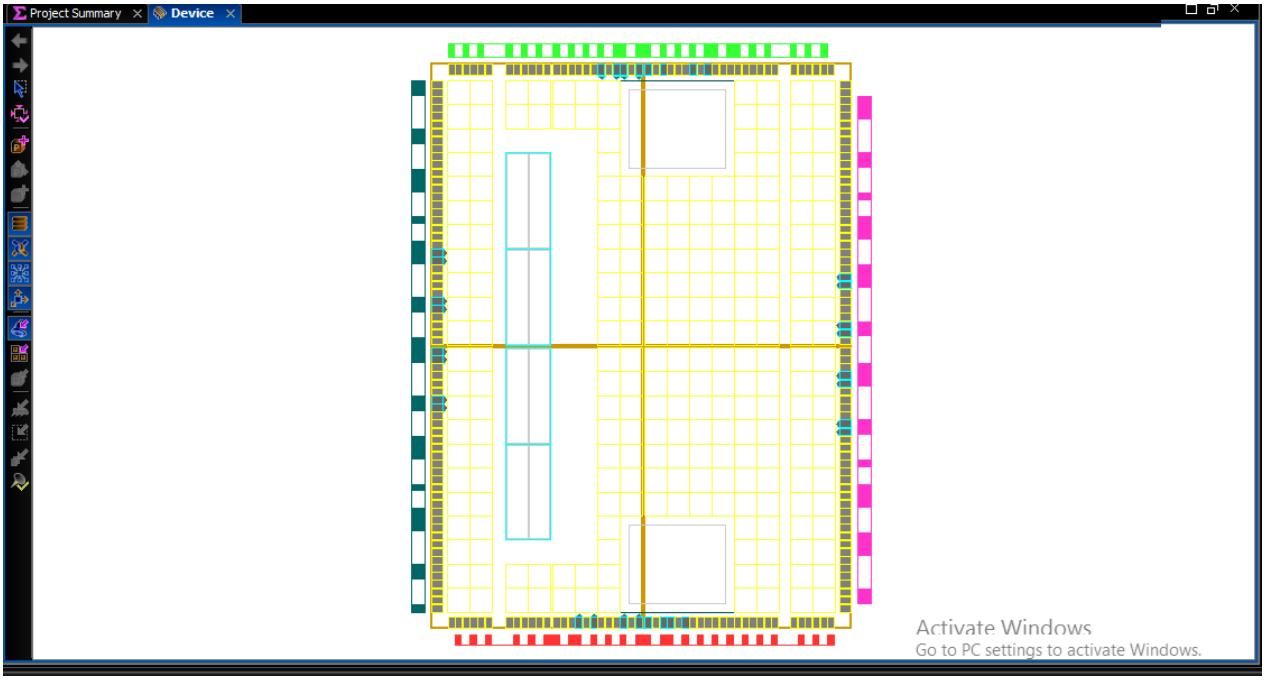
```
    end
    2'b01:
    begin
p[3]<=p[0];
p[2]<=p[3];
p[1]<=p[2];
p[0]<=p[1];
end
    2'b10:
    begin
p[0]<=p[3];
p[1]<=p[0];
p[2]<=p[1];
p[3]<=p[2];
end
    2'b11:
    begin
p[0]<=a[0];
p[1]<=a[1];
p[2]<=a[2];
p[3]<=a[3];
end
    endcase
    end
endmodule
```

Select (S0, S1)	Operation
00	Previous State
01	Shift Right
10	Shift Left
11	Parallel load

Simulation Output:



Floor Plan:



Result:

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

Expt No:6

Design Finite State Machine (Moore/Mealy) using HDL

Aim:

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

Apparatus Required:

Windows 11PC.

Xilinx Vivado Simulator.

Xilinx Zedboard.

Procedure:

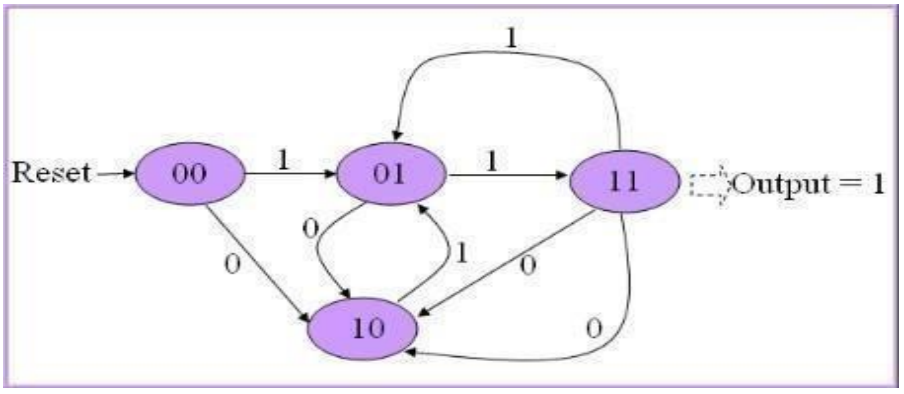
- Start the Xilinx project Navigator.
- Create a new project and new source.
- Write a Verilog code and synthesize and simulate the coding using ISE Simulator.
- Assign the package pins for input and output using XILINX plan ahead.
- Implement and verify the output in Spartan-3 FPGA kit.

Program:

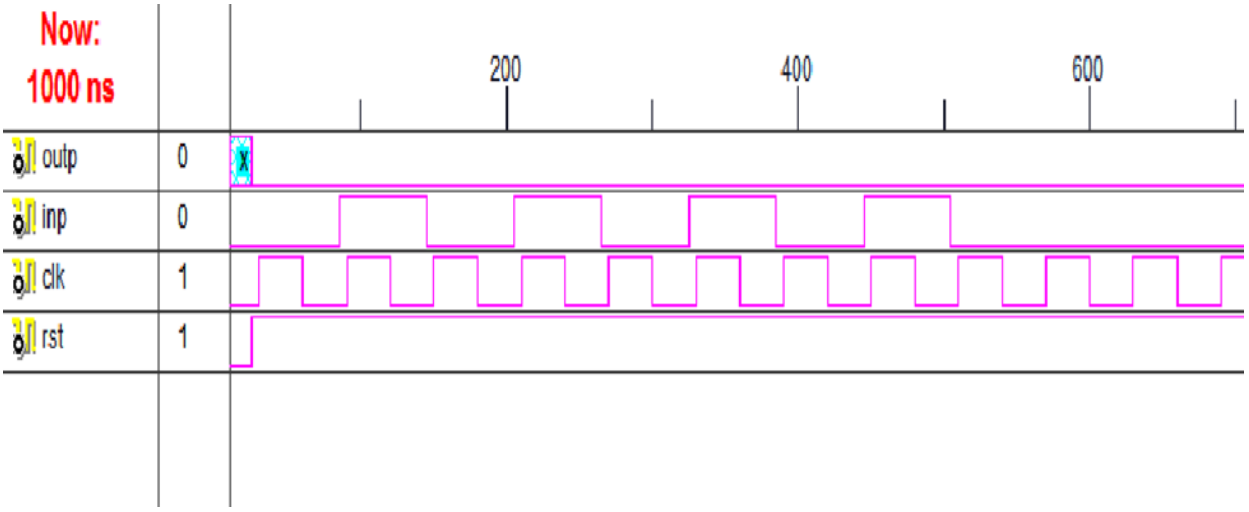
```
module moore (clk, rst, inp, outp);
input clk, rst, inp;
    output outp;
    reg [0:1] state;
    reg outp;
    always @ (posedge clk, posedege rst)
begin
    if (rst) state<=2’b00;
    else
    begin
    case (state)

    2’b00
begin
if( inp ) state <= 2'b01;
    else state <= 2'b10;
end

    2'b01:
    begin
    if( inp ) state <= 2'b11;
```

```
    else state <=
2'b10;  end


    2'b10:
    begin
    if( inp ) state <=
2'b01; else state <=
2'b11;
    end

    2'b11:
    begin
    if( inp ) state <= 2'b01;
    else state <= 2'b10;
    end
    endcase
    end
    end
    always @(posedge clk, posedge rst)
begin
    if( rst )
    outp <= 0;
    else if( state == 2'b11 ) outp <= 1;
else outp =0;

    end

    endmodule
```

Moore Machine:



Simulation Output:



Result:

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

Expt No.7 Design of Memories using HDL.

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.

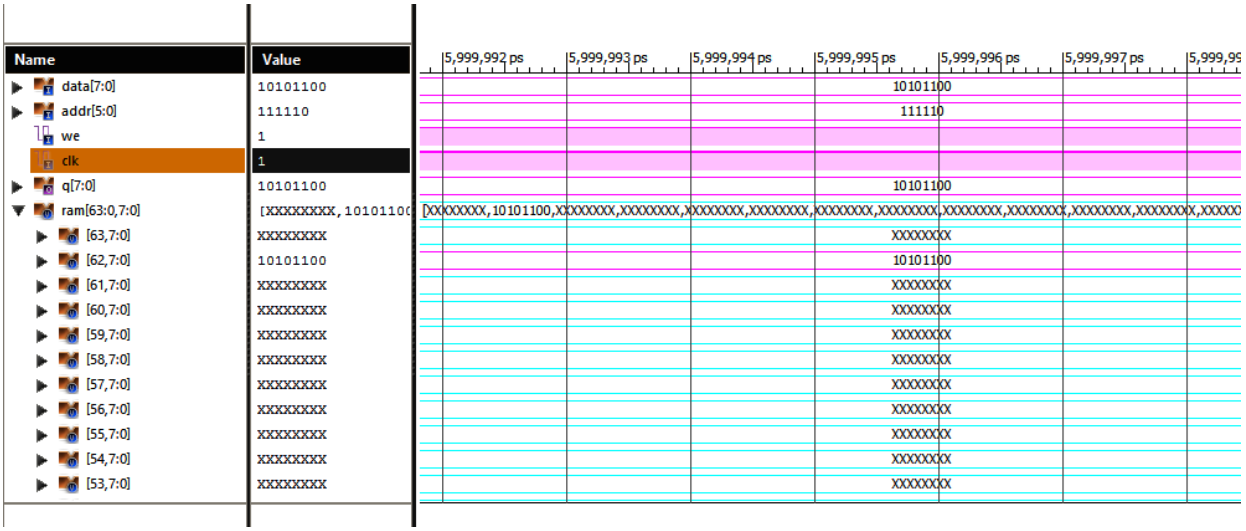
Procedure:

- 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.

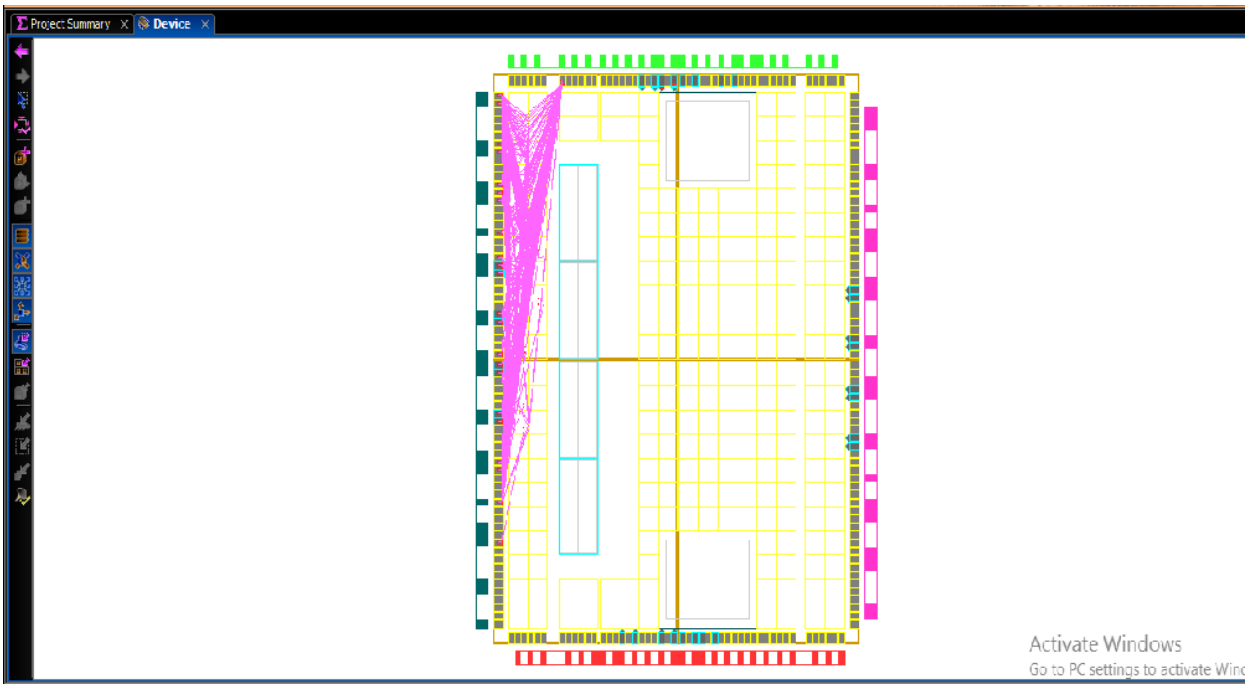
Program:

```
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:



Floor Plan:



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

8: CMOS INVERTER

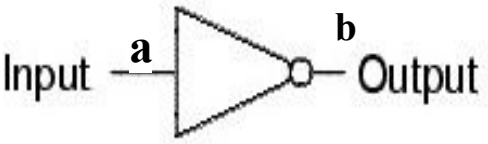
Aim:

Design and simulate a **CMOS Inverter** using Digital Flow.

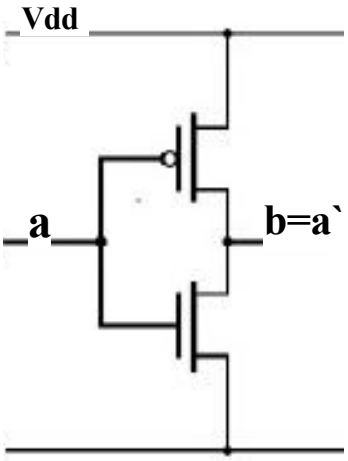
To write verilog code for an CMOS inverter circuit and its test bench for verification using incisive simulator, observe the waveform and synthesize the code with technological library with given constraints.

Theory:

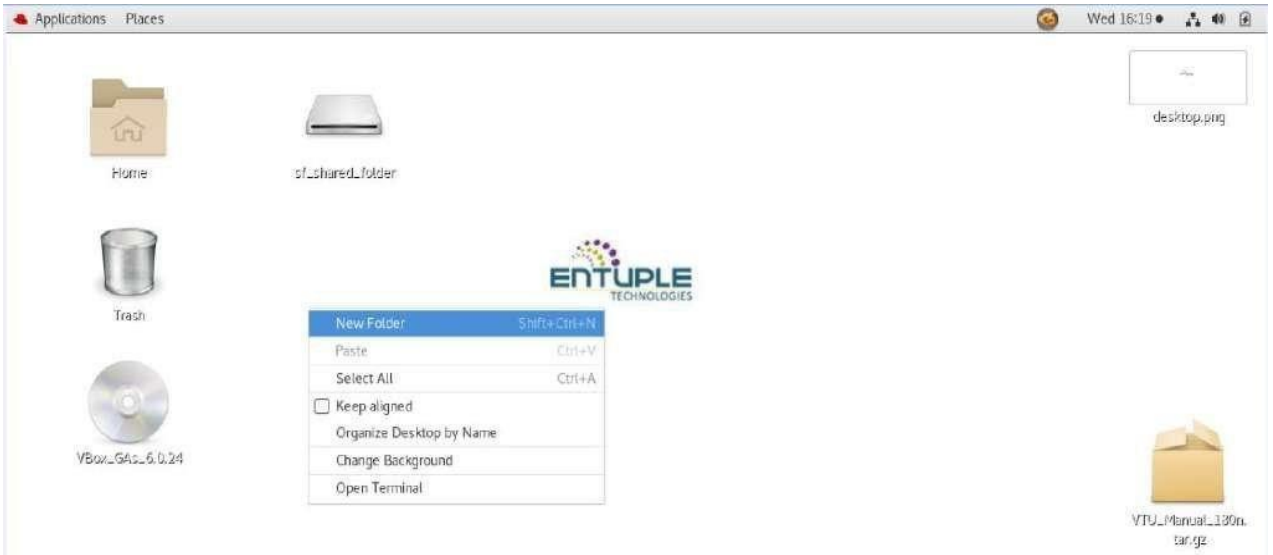
The NOT gate or an CMOS inverter is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is **"a"**, the inverted output is known as "**NOT a"**. This is also shown as **a'**, or **a** with a **bar** over the top, as shown at the outputs.



Input	Output
1	Vss
0	1

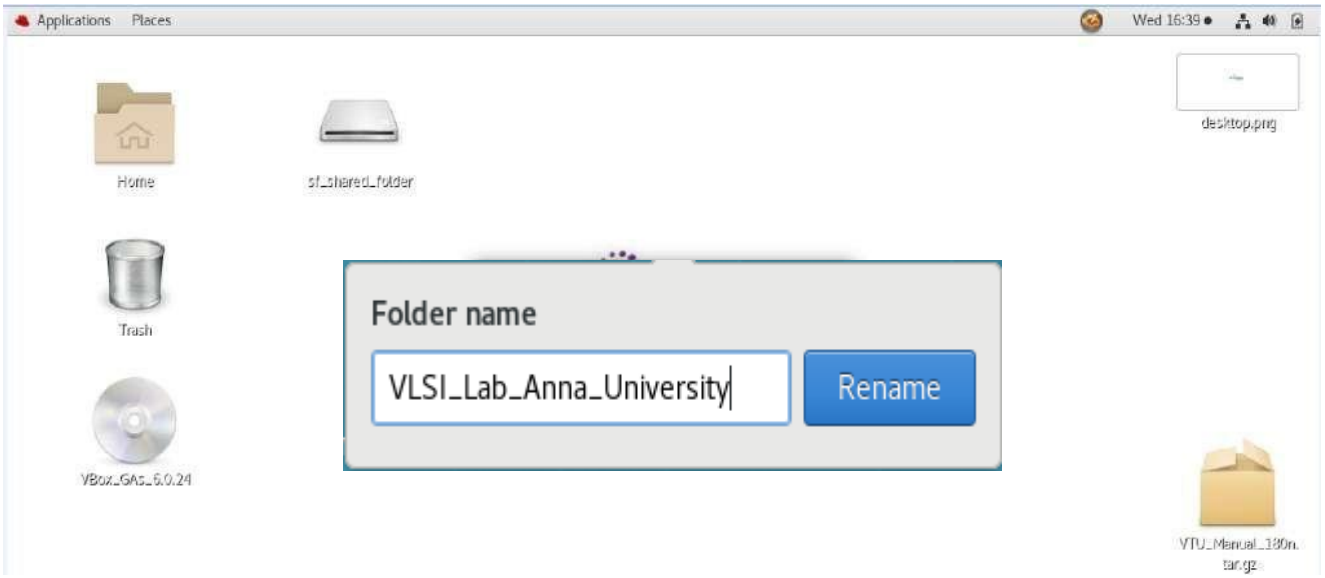


Module 1: Work Space Creation & Writing RTL Code for Inverter

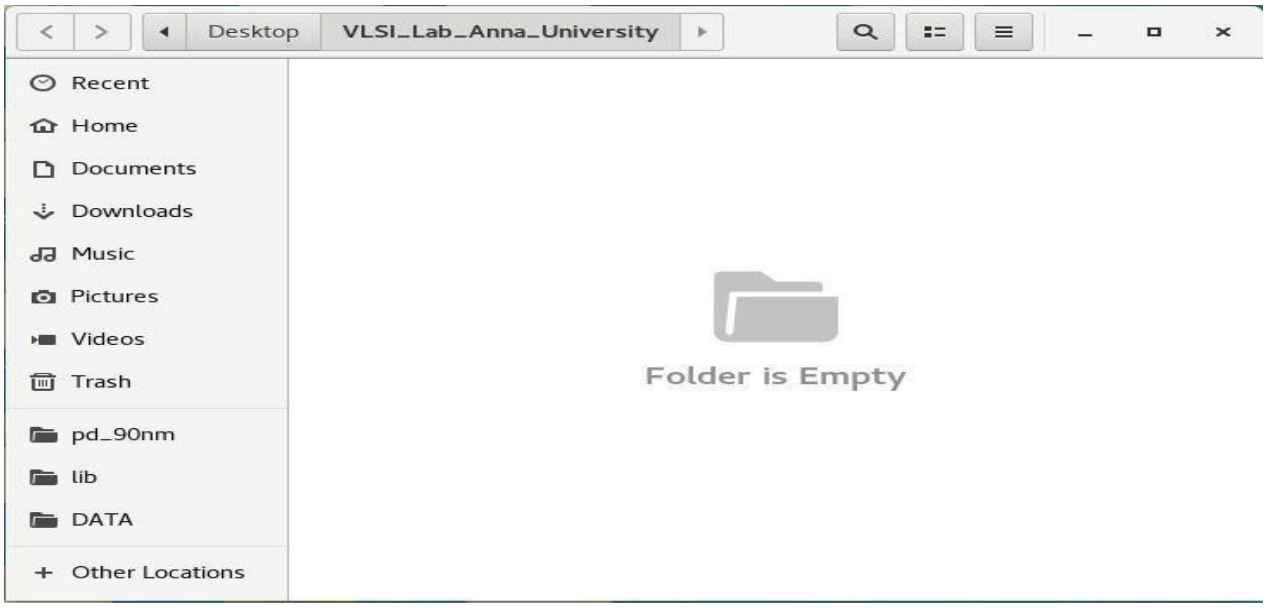


1. Make a **Right click** on the Desktop and select the option "**New Folder**" as shown in Figure below.
2. Name the folder (for example: VLSI_Lab_Anna_University) and click on "**Create**" as shown

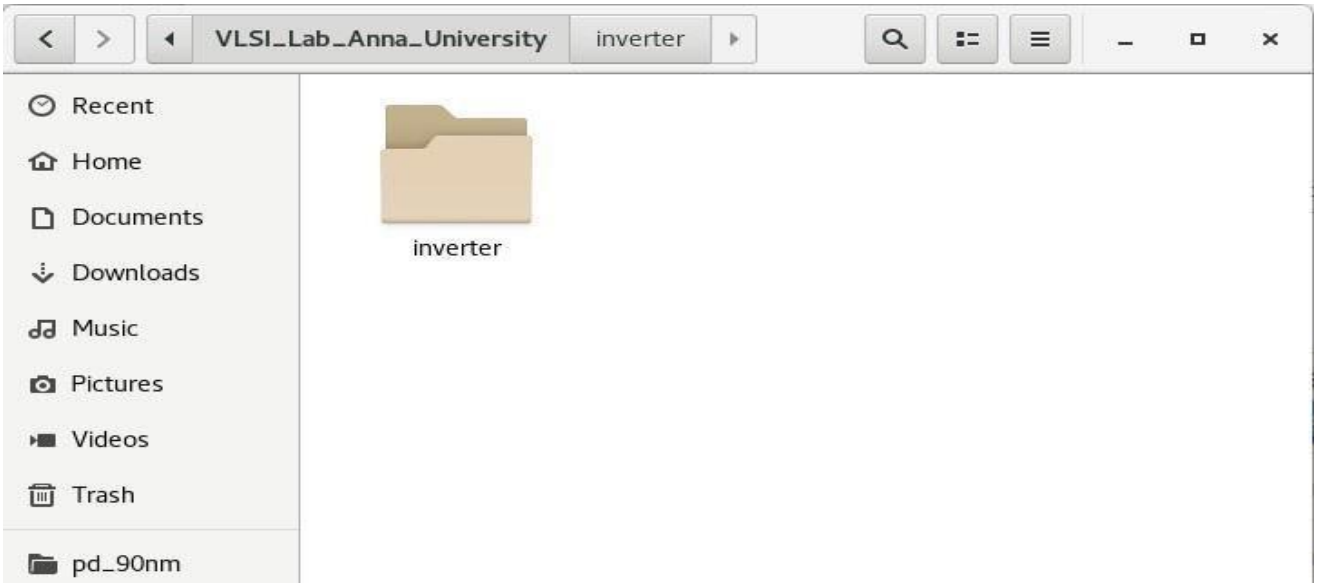
below.



3. Open the folder by a **double click** and the window can be seen as shown in below.



4. Create **sub directory** or folder with name inverter as shown below.

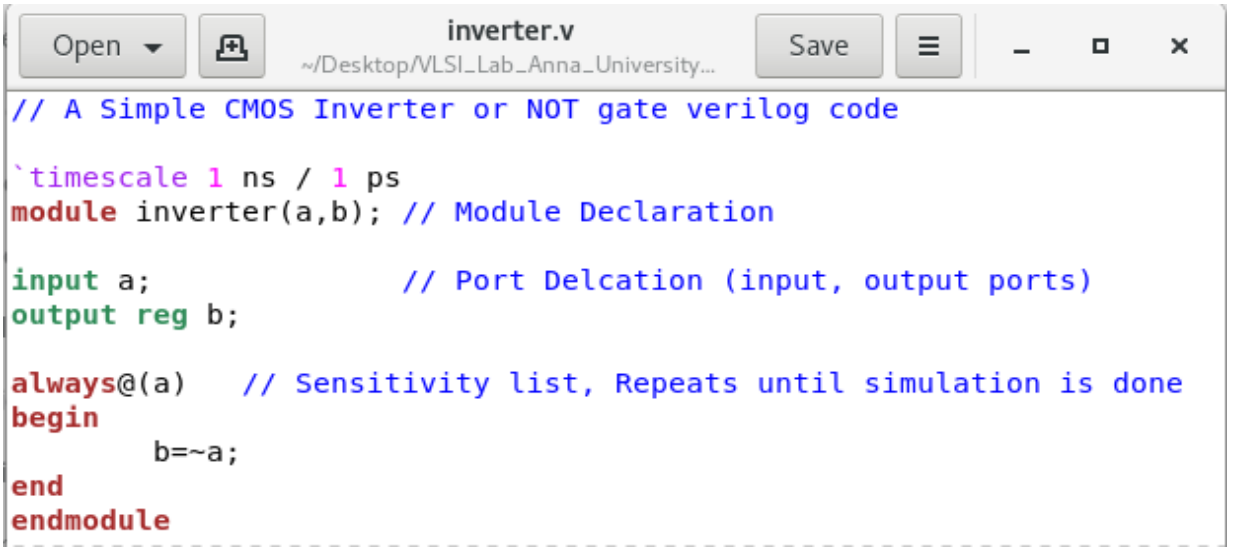


Note: Give folder name without any space

4. In order to create an RTL code for inverter, you can open a text editor and type verilog code or VHDL code.

5. Open the inverter directory and **Right click** and select "**Open terminal**" type the command given below

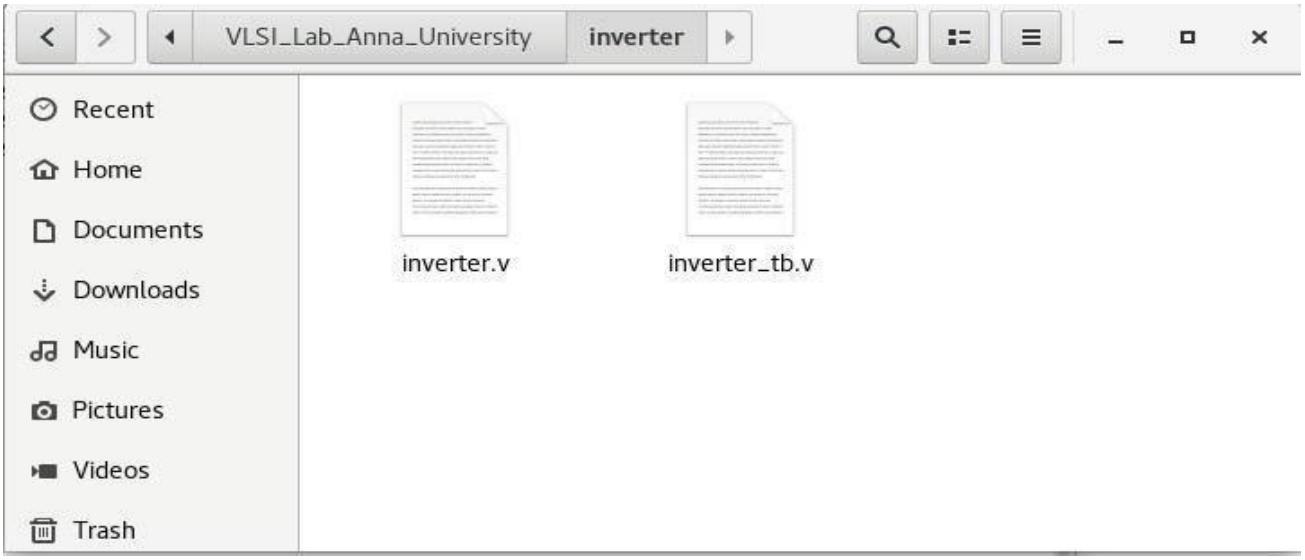
 # gedit inverter.v inverter_test.v



Open ▾ | inverter_tb.v | ~/Desktop/VLSI_Lab_Anna_University/... | Save

```
//A Simple CMOS Inverter or NOT gate Testbench code
`timescale 1 ns / 1 ps
module inverter_test(); // Module Declaration
reg a;
wire b;

inverter g1(.a(a), .b(b)); // Instantiation

initial  // drive test vectors
begin
        a=0;
#10     a=1;
#20     a=0;
end

initial
begin

$monitor("time=%d, a=%d, b=%d", $time,a,b); // Log based data
monitor

#50 $finish; // Control simulation time
end
endmodule
```

6. Once after writing the verilog and test-bench code for inverter you can **save** the files and **close** it.



Steps to Invoke Tools:

Invoke the cadence environment by type the below commands

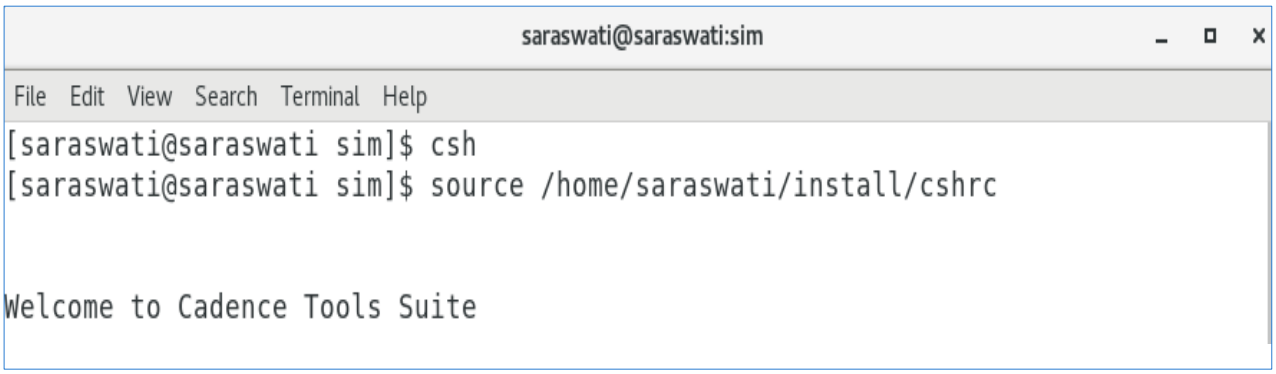
7. Before invoking any tool, invoke C shell be typing ‘csh’ in terminal.

#csh

8. Source the cshrc file by typing ‘source <cshrc file>’ e.g.: - source cshrc

#Source /home/install/cshrc (mention the path of the tools)

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

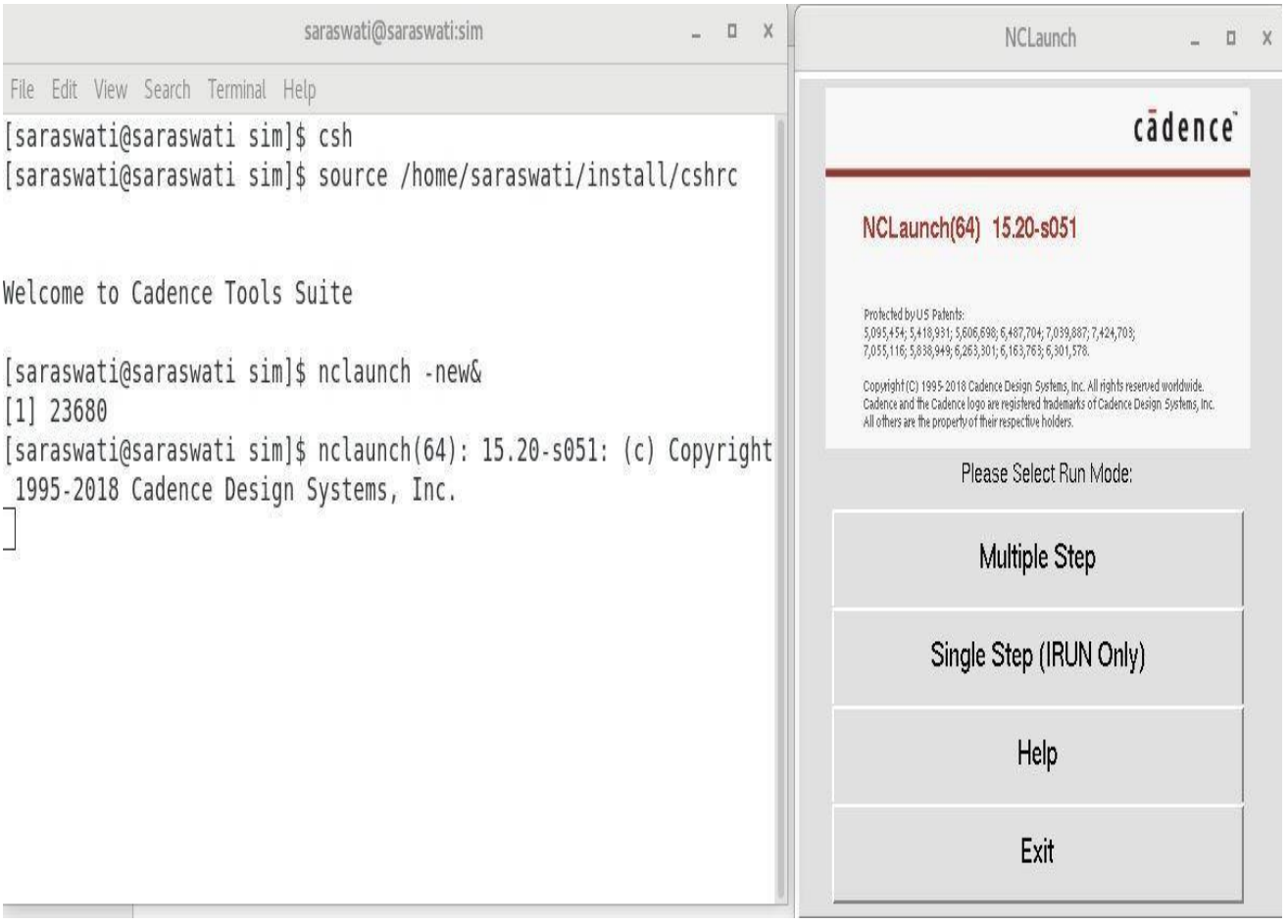


Module 2: Design Simulation Using the INCISIVE Simulator

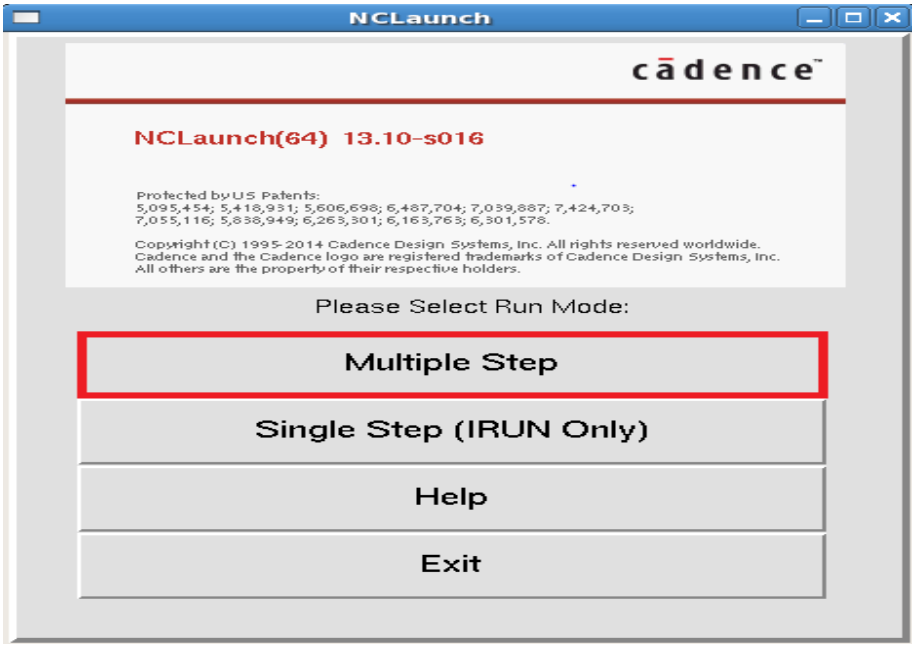
Objective: To simulate a simple counter design with a testbench using the Incisive Enterprise Simulator or IES tool.

10. To perform functional simulation, "INCISIVE" simulator tool is to be used.

11. In your terminal, type the command "**nclaunch -new&**" to open the tool as shown below.



Note: The '**-new**' switch is used only for the first time the design is being run. For the next time on wards, the command to be used could be 'nclaunch' only.



12. Select "**Multiple Step**" option.

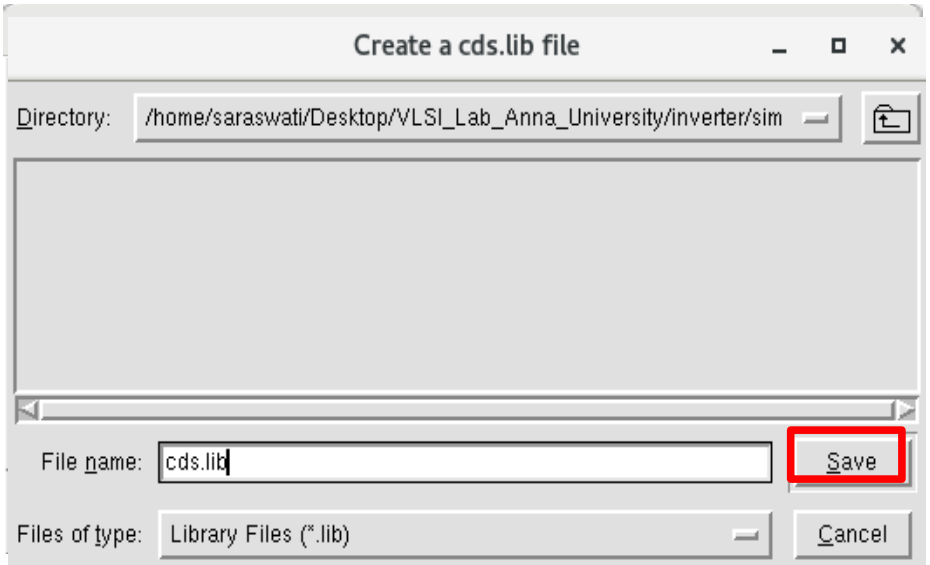
It will invoke the nclaunch window for functional simulation we can compile, elaborate and simulate it using Multistep

we can simulate a design using the Incisive simulator. For that we have to Create the **cds.lib** and **hdl.var** files for to Compile, elaborate and simulate the design and test bench

- **cds.lib:** It maps logical library names to physical directory paths. It also defines the path to the work library and to any reference libraries that the design required.

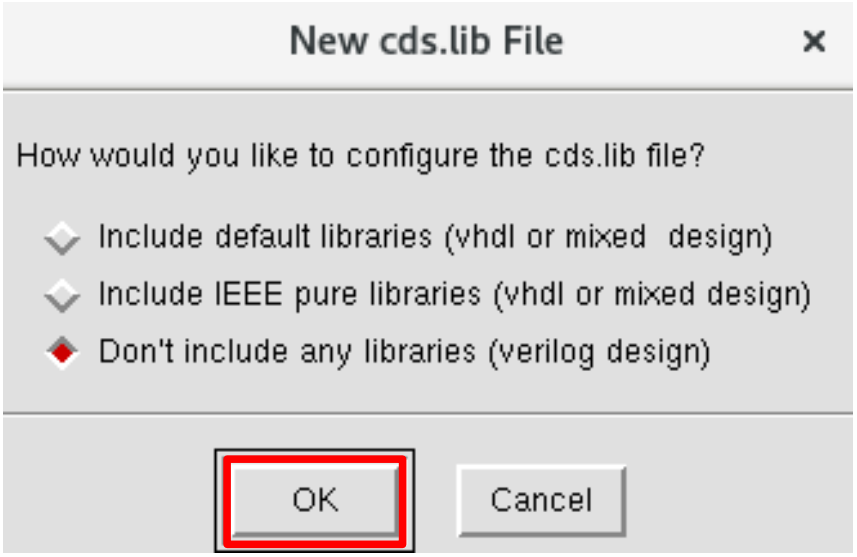
- **hdl.var:** Defines the work library. It may also define variables that configure your design environment, control the operation of incisive simulation tool and specify the location of supported files and invocation scripts.

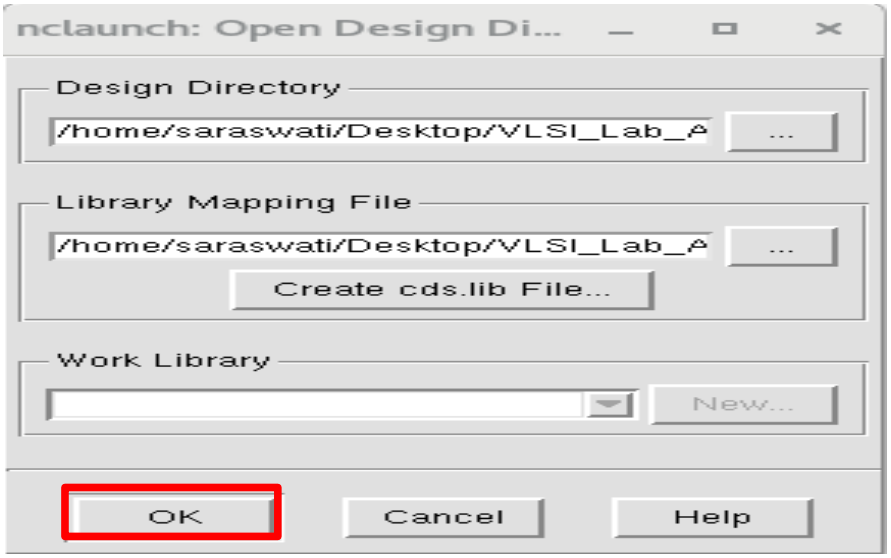
13. Click the **Create cds.lib file** and **save** it.



14. Choose any of the option listed below. Based on the libraries available and the

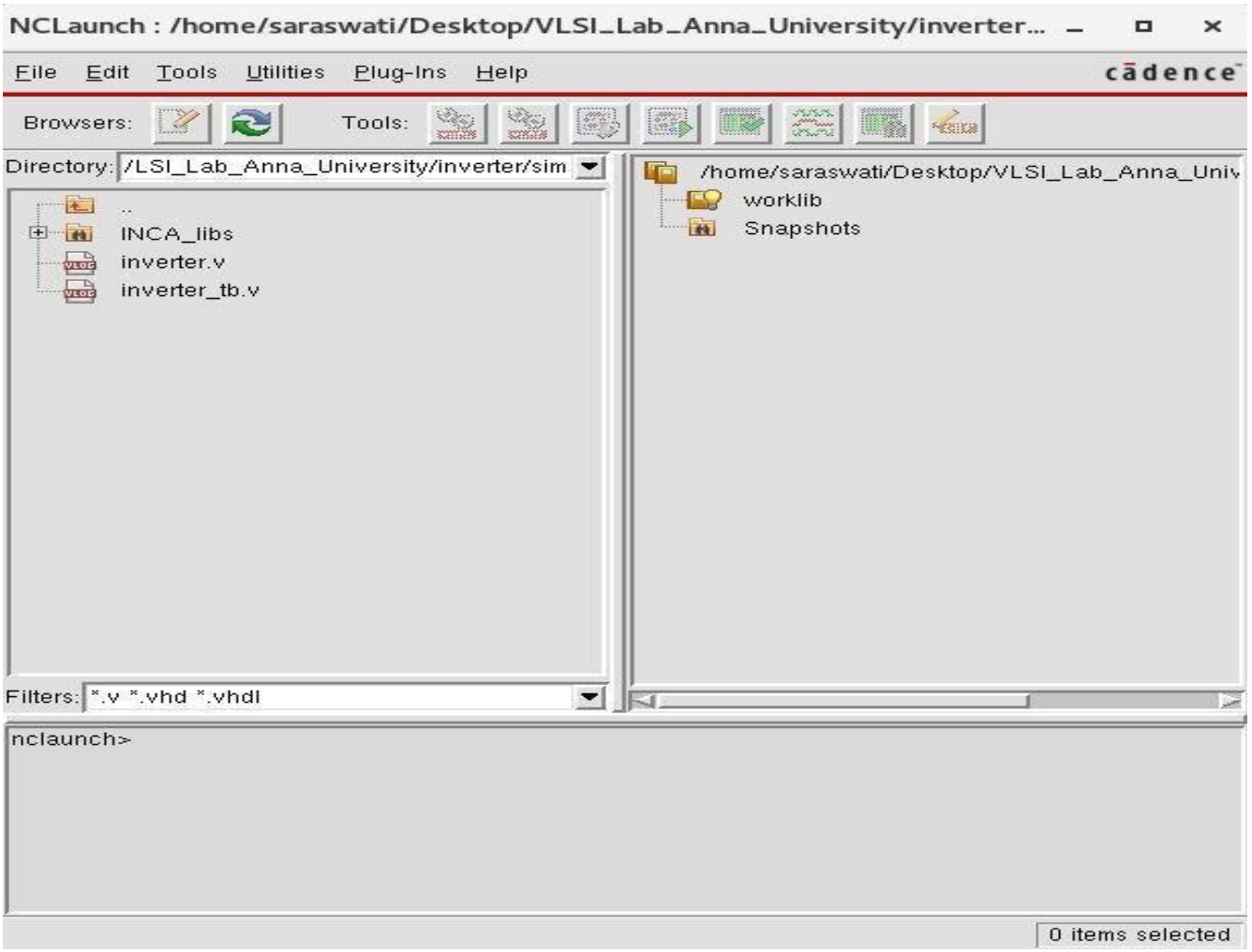
type of RTL code written in verilog so, you need to choose "**Don't include any libraries (verilog design)**" as shown below.





You can see the below window after giving "O**K**"

15. A new window pop-up as shown below that is a **NCLaunch window.**

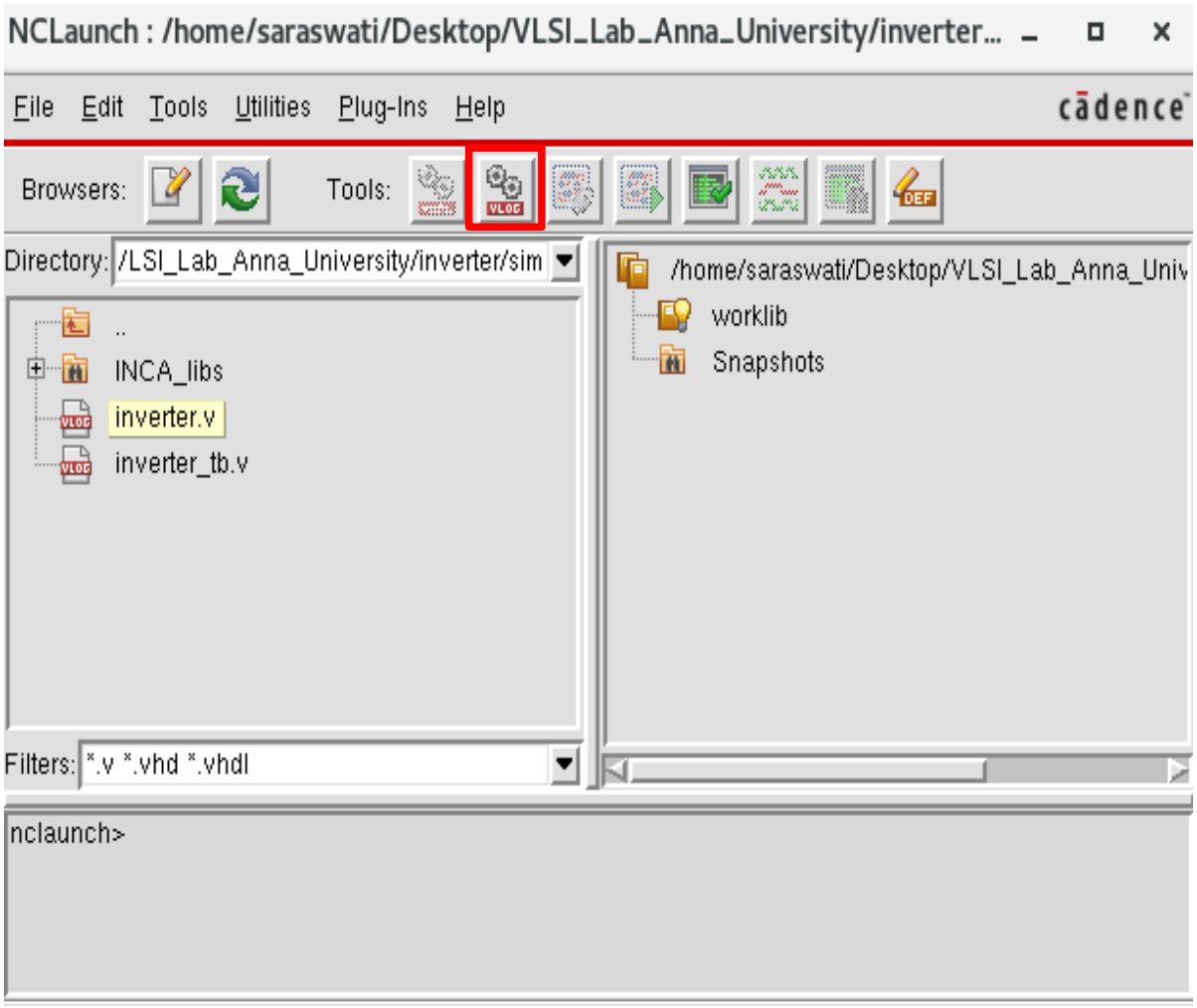


Left side you can see the **HDL files** and right side of the window has **worklib** and **snapshots** directories listed.

Worklib is the directory where all the compiled codes are stored while **Snapshot** will have output of elaboration which in turn goes for simulation

Functional simulation using cadence runs in 3 stages:
i. Compilation
ii. Elaboration
iii. Simulation

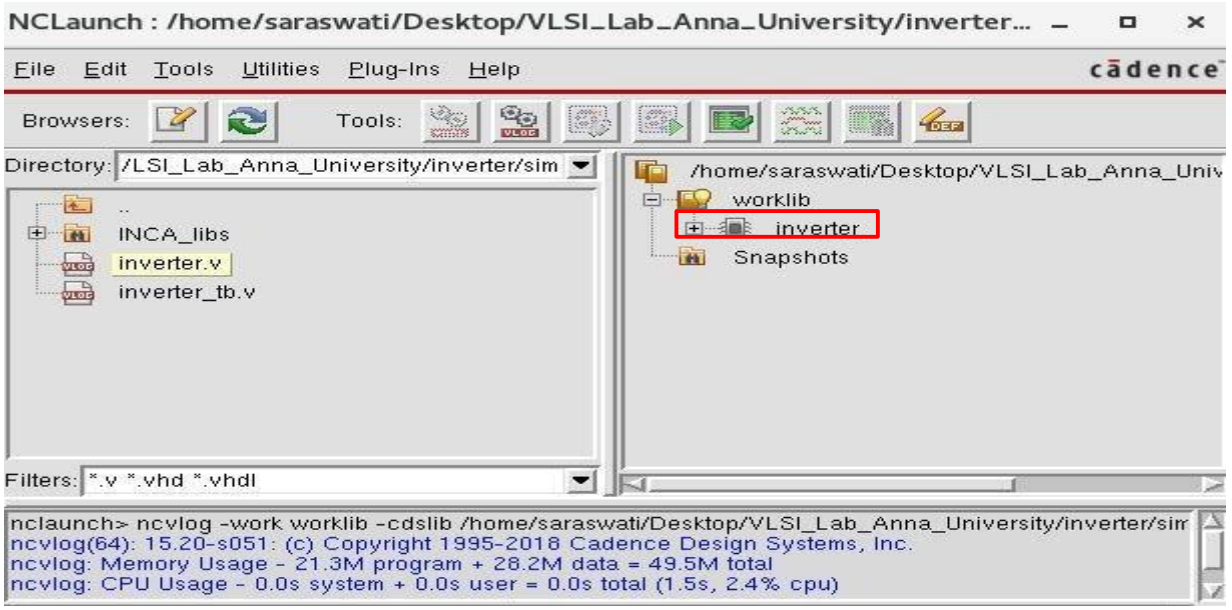
Compilation: (Checks syntax and semantics errors)



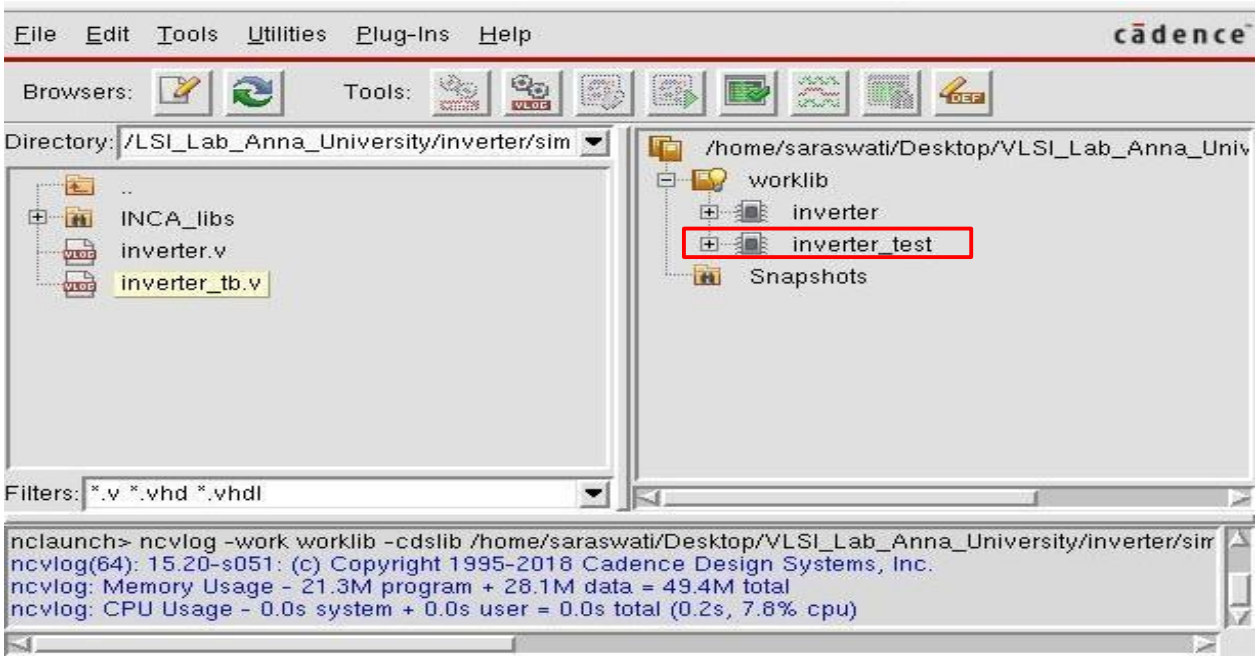
16. Left side select the file (inverter.v) and in **Tools : launch verilog compiler with current selection** will get enable.

Click it to compile the code

17. After compilation it will come under **worklib** you can see in right side window.



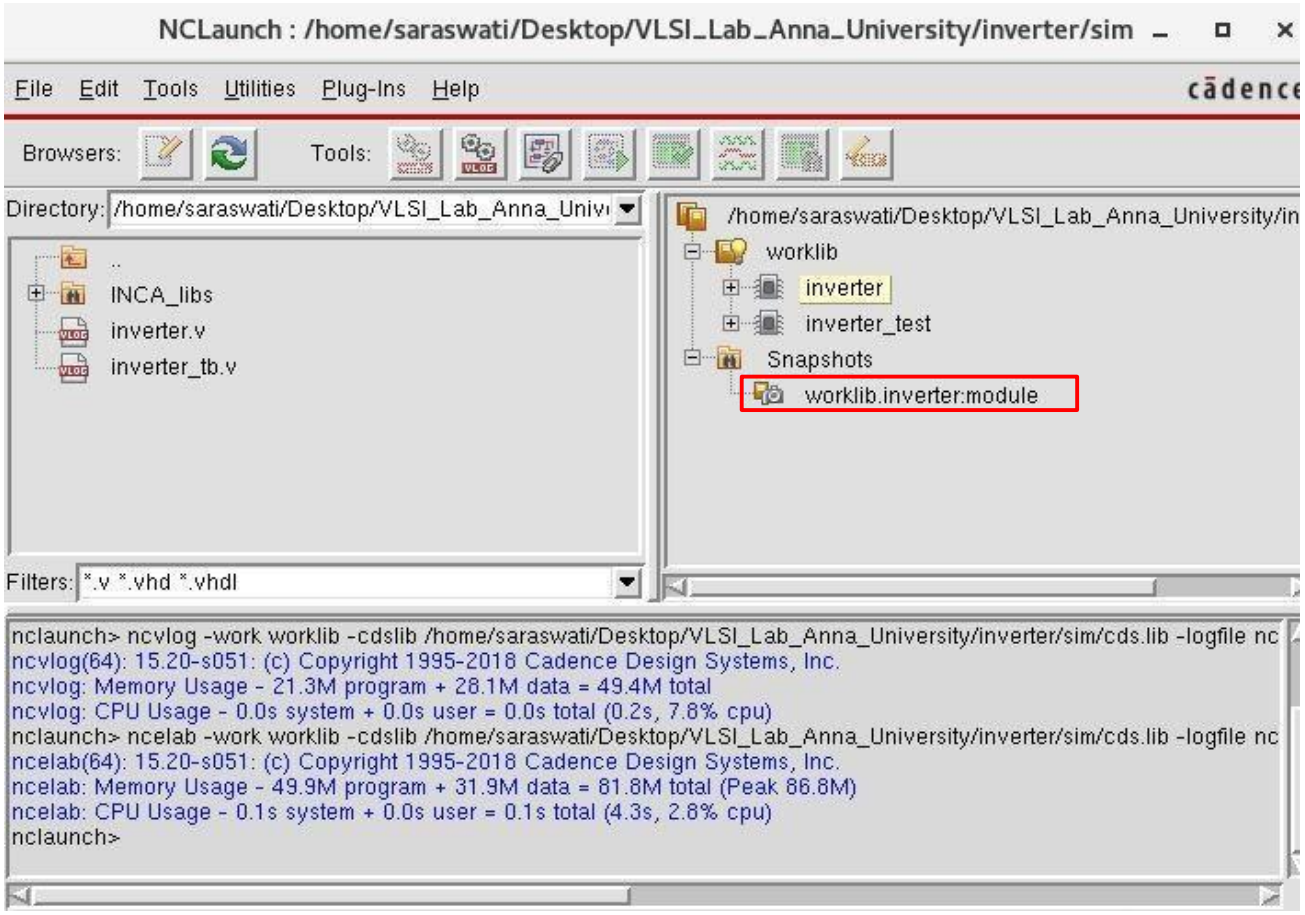
18. Select the test bench and compile it. It will come under **worklib**. Under Worklib you can see the module and testbench. Next is to elaborate the design.

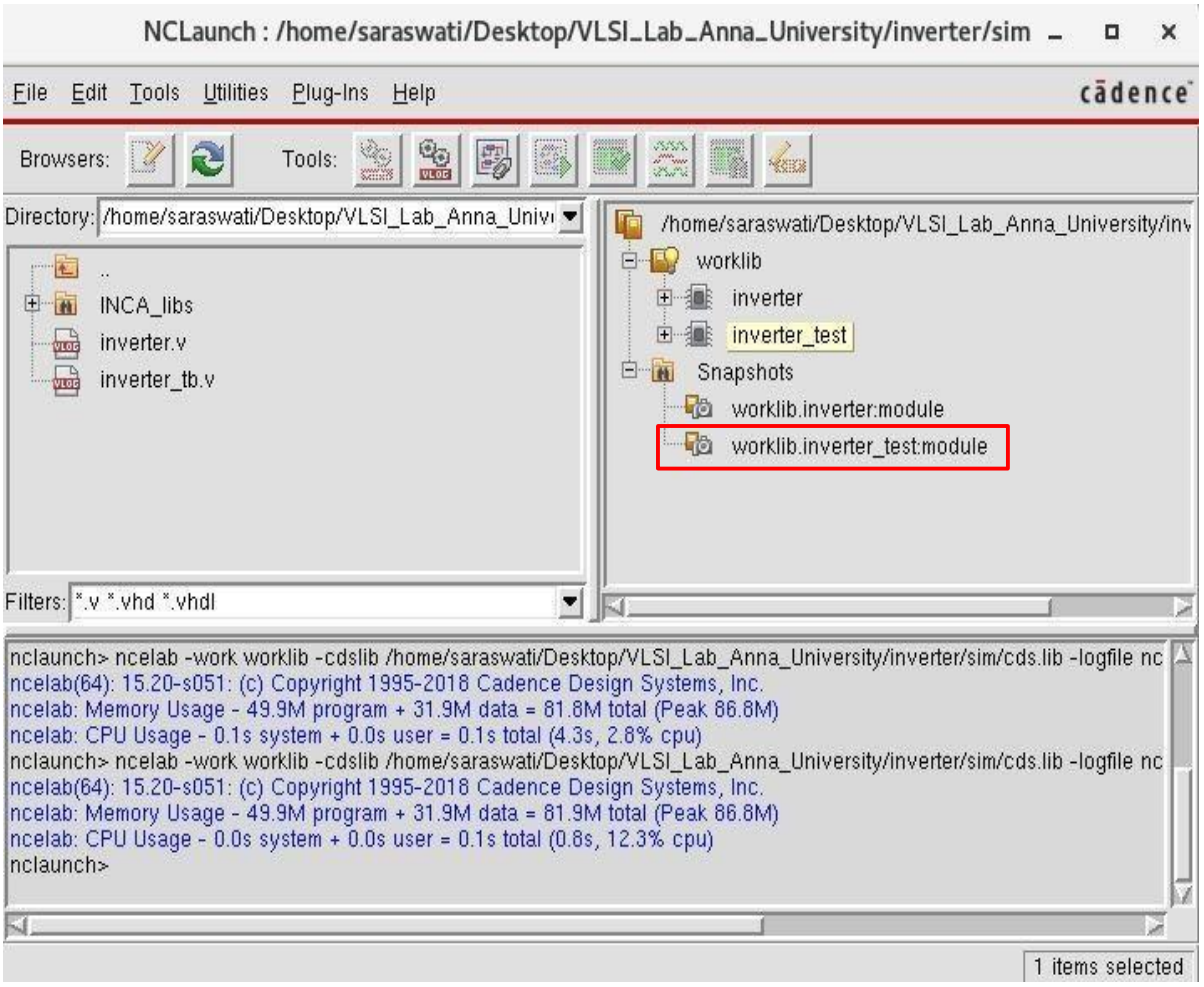


Elaboration:

19. select the file under worklib and in **Tools : launch elaborator with current selection** will get enable. select the **elaborator** to elaborate the design.

Choose the module and test bench and elaborate the design.

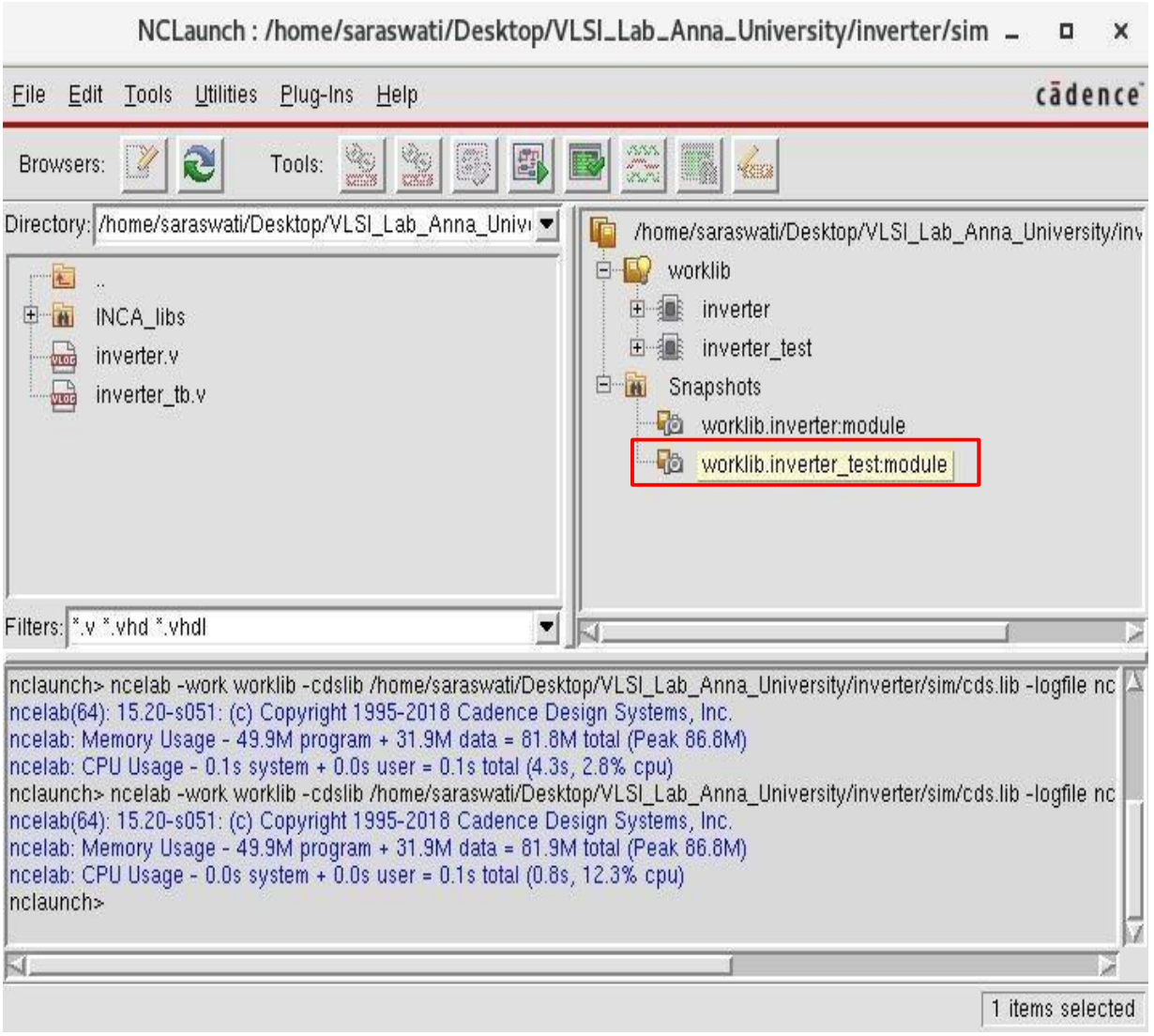

NCLaunch : /home/saraswati/Desktop/VLSI_Lab_Anna_University/inverter/sim
File Edit Tools Utilities Plug-Ins Help
cadence
Browsers:
Tools:
Directory: /home/saraswati/Desktop/VLSI_Lab_Anna_Univ
..
INCA_libs
inverter.v
inverter_tb.v
Filters: *.v *.vhd *.vhdl
/home/saraswati/Desktop/VLSI_Lab_Anna_University/in
worklib
inverter
inverter_test
Snapshots
worklib.inverter:module
nclaunch> ncvlog -work worklib -cdslib /home/saraswati/Desktop/VLSI_Lab_Anna_University/inverter/sim/cds.lib -logfile nc
ncvlog(64): 15.20-s051: (c) Copyright 1995-2018 Cadence Design Systems, Inc.
ncvlog: Memory Usage - 21.3M program + 28.1M data = 49.4M total
ncvlog: CPU Usage - 0.0s system + 0.0s user = 0.0s total (0.2s, 7.8% cpu)
nclaunch> ncelab -work worklib -cdslib /home/saraswati/Desktop/VLSI_Lab_Anna_University/inverter/sim/cds.lib -logfile nc
ncelab(64): 15.20-s051: (c) Copyright 1995-2018 Cadence Design Systems, Inc.
ncelab: Memory Usage - 49.9M program + 31.9M data = 81.8M total (Peak 86.8M)
ncelab: CPU Usage - 0.1s system + 0.0s user = 0.1s total (4.3s, 2.8% cpu)
nclaunch>



After elaboration the file will come under snapshot. Select the test bench and elaborate it.

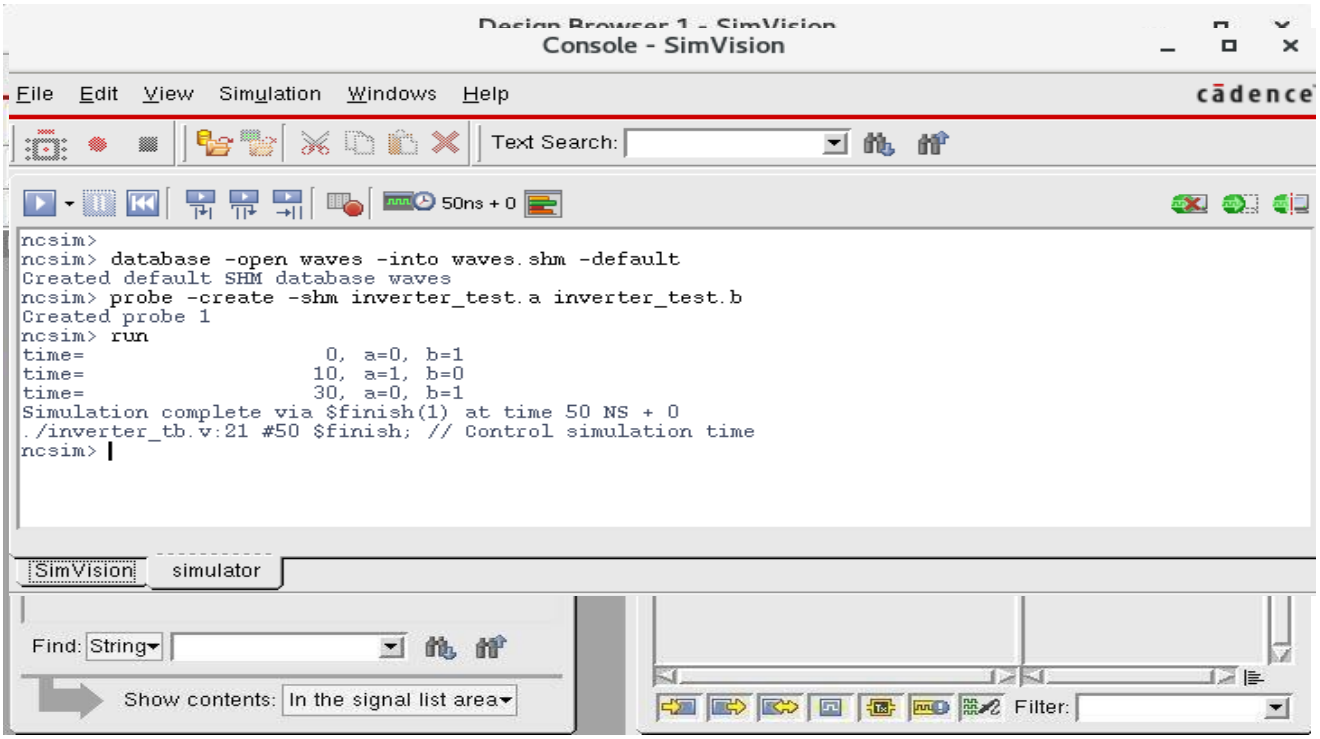
Simulation:

20. Select the test-bench file under snapshot and in **Tools: Launch simulator with current selection** will get enable.

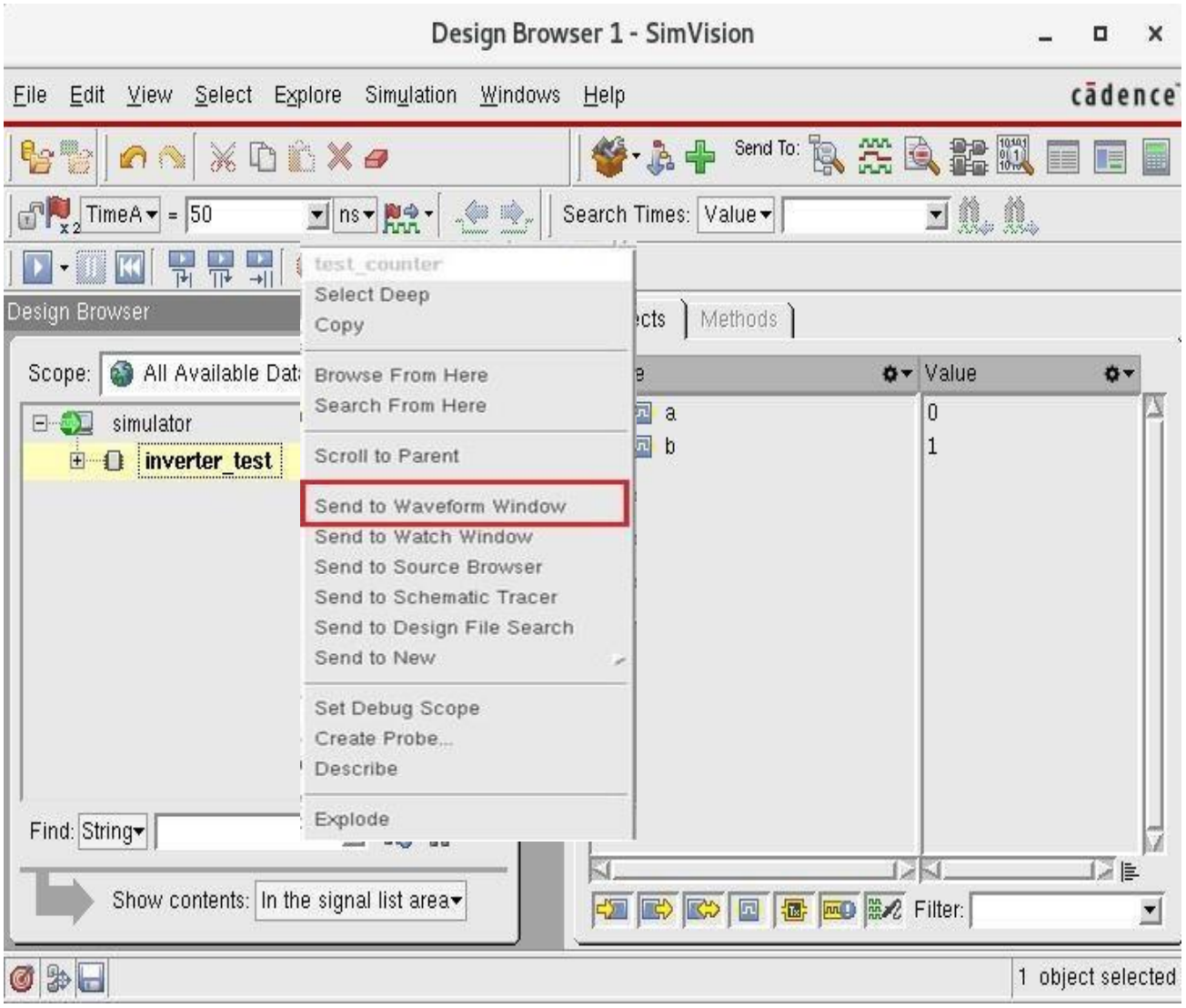


Select simulator to simulate the design. After simulation you will get the two windows like below image.

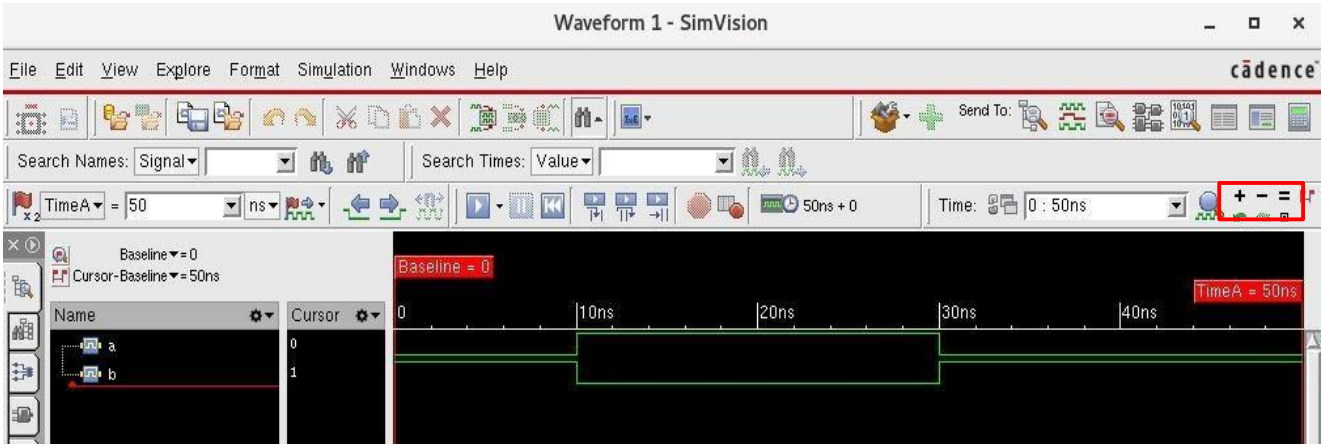
21. You will get the two windows **Design Browser 1 - SimVision** and Console **- SimVision**.In design browser you can see the test bench in left side window.


Console - SimVision
File Edit View Simulation Windows Help
cadence
Text Search:
50ns + 0
ncsim>
ncsim> database -open waves -into waves.shm -default
Created default SHM database waves
ncsim> probe -create -shm inverter_test.a inverter_test.b
Created probe 1
ncsim> run
time= 0, a=0, b=1
time= 10, a=1, b=0
time= 30, a=0, b=1
Simulation complete via $finish(1) at time 50 NS + 0
./inverter_tb.v:21 #50 $finish; // Control simulation time
ncsim>
SimVision
simulator
Find: String
Show contents: In the signal list area
Filter:

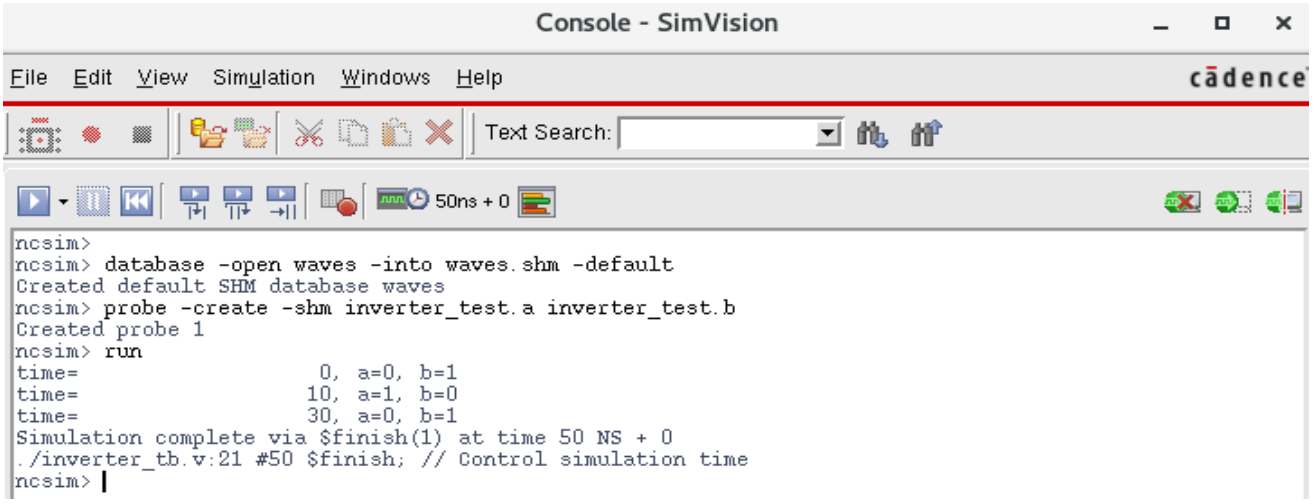
22. Select the test bench for the inverter and **Right click** it. **Select the send to waveform window** or select the **waveform icon**



23. You can see the waveform window after that click the **Run tool** to see the functional simulation for the inverter.



The equivalent command terminal output can be observed in the Simvision console window and also in the nclaunch console terminal.



3: Synthesis Using GENUS Tool

Synthesis is the process of converting the **RTL Coding** into optimized **Gate Level Netlist**. The Tool used for doing the synthesis is **GENUS**. The tcl file (Tool Command Language) is used for scripting.

Inputs for Synthesis:

1. RTL Code (.v or .vhdl)
2. Chip Level SDC (System Design Constraints)

3. Liberty Files (.lib)

Expected Outputs of Synthesis:

1. Gate Level Netlist
2. Block Level Netlist
3. Timing, Area and Power Reports

Synthesis runs in following 3 stage process:

i. **Translation** – Converts RTL code to boolean expression
ii. **Mapping** – Boolean expression is mapped to logic /standard cells available from libraries
iii. **Optimization** - Tool tries to reduce cell count without affecting the functionality

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

Script for synthesis (genus_script.tcl):

Open | genus_script.tcl ~/Desktop/VLSI_Lab_Anna_University/inverter/synthesis | Save

```
# 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 inverter
read_hdl "inverter.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 > output/inverter_netlist.v
write_sdc > output/inverter_sdc.sdc
# PPA Reports
report_power > output/inverter_power.rpt
report_area > output/inverter_area.rpt
report_timing -unconstrained > output/inverter_timing.rpt
report_gates > output/inverter_gatecount.rpt
gui_show
```

Tcl | Tab Width: 8 | Ln 27, Col 45 | INS

Chip Level SDC (constraints_top.sdc):

Open | constraints_top.sdc ~/Desktop/VLSI_Lab_Anna_... | Save

```
set_input_delay -max 1.0 [get_ports "a"]
set_output_delay -max 1.0 [get_ports "b"]
```

Script File Explanation:

1. Give the path of the library w.r.t to the directory you are using the command:
   set_db / .init_lib_search_path {/home/install/FOUNDRY/digital/90nm/dig/lib}

2. Give the path of the RTL files w.r.t to the directory you are using the command:
   set_db / .init_hdl_search_path {.}

3. Read the library file from the directory specified in giving the path for the library files in First line using the command:
   set_db / .library "slow.lib"

4. Set the top module name for the design you are running using the command:
   set DESIGN inverter

5. Read the RTL files from the directory specified in the second line. The RTL files are in the directory name : **read_hdl "inverter.v"**

6. Now Elaborate the design using the command:
   elaborate $DESGIN

7. To check the any unresolved references with the RTL code you can check by using the command:
   check_design -unresolved

8. Constraint File : Not Mandatory, If you are having constraint file then you can read the constraint file using command:
   read_sdc constraints_top.sdc

9. Set the effort level to (low, medium & high) the all the 3-stages of synthesis based on the user specification by using the commands:
   set_db syn_generic_effort medium
   set_db syn_map_effort medium
   set_db syn_opt_effort medium
10. Synthesize the circuit using the commands:
   syn_generic
   syn_map
   syn_opt

OUTPUT Files Generated from Synthesis Process

11. Write the Gate level netlist (GLN) or hdl code in terms of library components for the synthesized circuit using the command:
    write_hdl > inverter_netlist.v

12. Similarly write the constraint file using:
    write_sdc > inverter_const.sdc

REPORTS Files Generated from Synthesis Process

13. Check Power dissipation using:
    report_power > inverter_pwr.rpt

14. Check area using:
    report_area > inverter_area.rpt

15. Timing could be check using command:
    report_timing -unconstrained (only for combinational circuit)

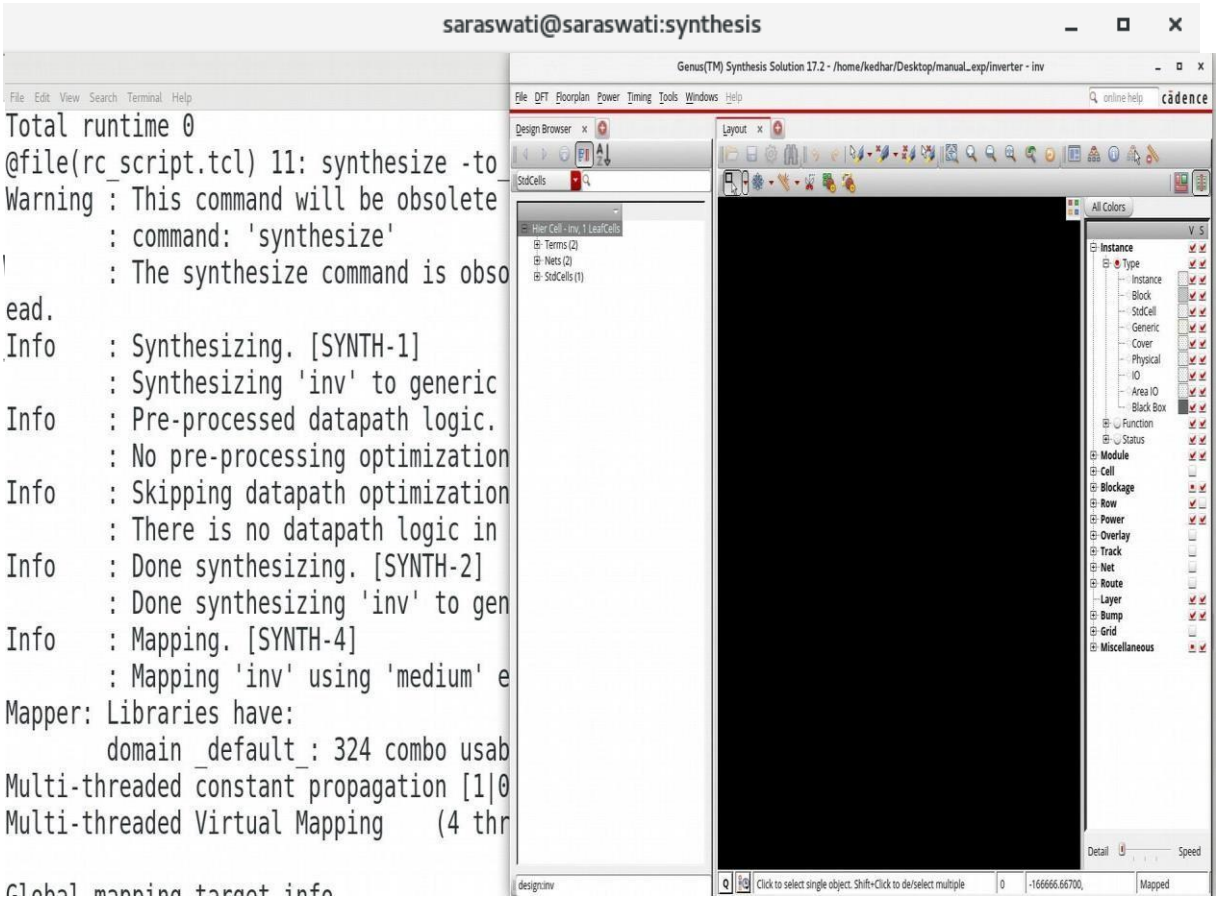
16. Similarly for Gates:
    report_gates > inverter_gates.rpt

17. To see the schematic capture of invert use the command:
    gui_show

Power Report:
Invoke the Genus tool by typing the below command on your terminal:

genus -f genus_script.tcl

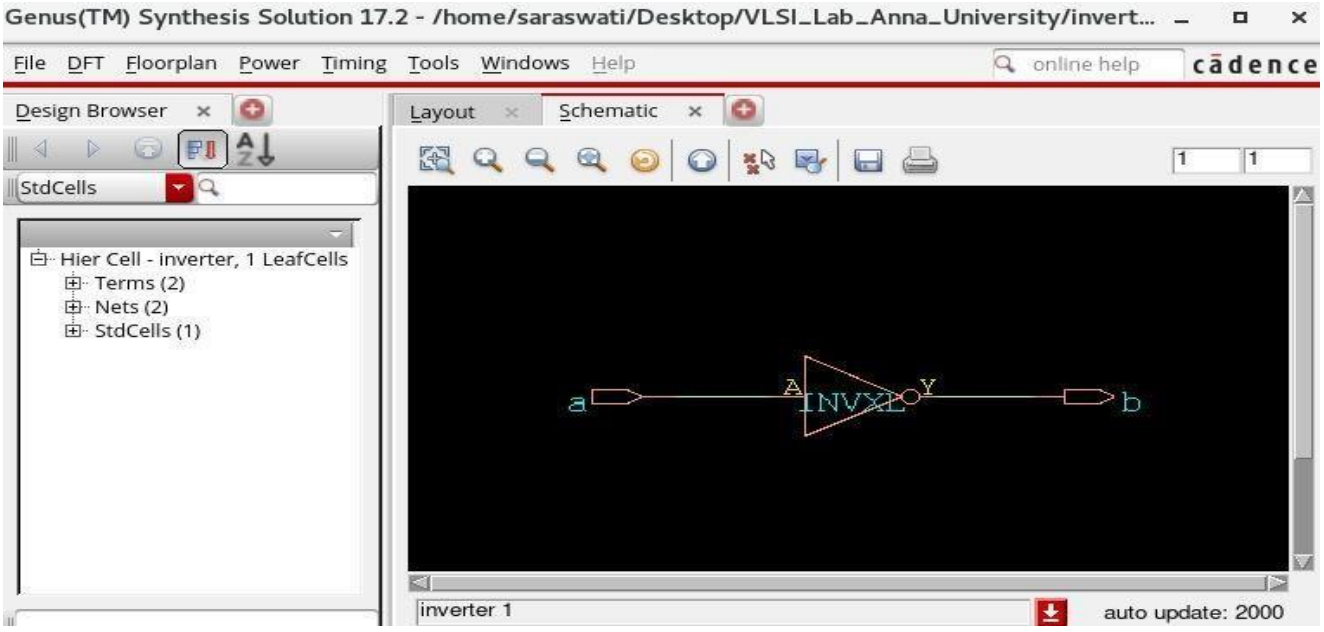
The tcl [Tool Command Language] script runs executing each command one after the other.



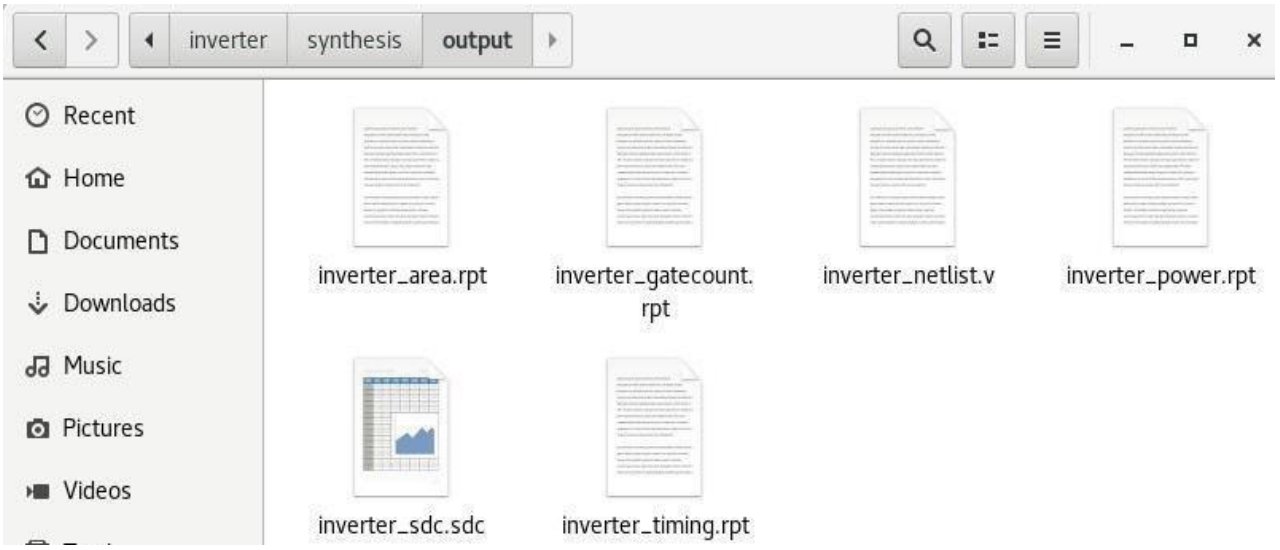
A window of **Genus GUI** pops – up with the top hier cell on the left top.

Make a **Right Click** and select **Schematic Viewer → In Main.**

Power Report:



It will generate the **Gate Level Netlist** (inverter_neltist.v), **SDC constraints** (inverter_sdc.sdc) and Report (**Timing, Area, Power and Gates**) for the design as shown below.



Pow

```
╞=============================================================
    Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
    Generated on:           Jan 11 2022  02:22:09 pm
    Module:                 inverter
    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)
  ------------------------------------------------
  inverter     1     8.757    44.417    53.174
```

Area Report:

```
 ╞=============================================================
   Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
   Generated on:           Jan 11 2022  02:22:09 pm
   Module:                 inverter
   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
 ---------------------------------------------------------------------------
 inverter                 1      2.271     0.000        2.271    <none> (D)

  (D) = wireload is default in technology library
```

Timing Report:

```
╞=============================================================
  Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
  Generated on:           Jan 11 2022  02:22:09 pm
  Module:                 inverter
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
=============================================================


Path 1: UNCONSTRAINED Late External Delay Assertion at pin b
     Startpoint: (F) a
       Endpoint: (R) b


   Output Delay:-    1000
    Input Delay:-    1000
      Data Path:-      12

Exceptions/Constraints:
  input_delay              1000           constraints_top.sdc_line_1
  output_delay             1000           constraints_top.sdc_line_2

#-------------------------------------------------------------------------------------------
# Timing Point   Flags   Arc   Edge   Cell     Fanout Load Trans Delay Arrival Instance
#                                                (fF)  (ps)  (ps)   (ps)  Location
#-------------------------------------------------------------------------------------------
  a              -       -     F     (arrival)      1  1.6     0     0    1000    (-,-)
  g2/Y           -       A->Y  R     INVXL          1  0.0     6    12    1012    (-,-)
  b              -       -     R     (port)         -    -     -     0    1012    (-,-)
#-------------------------------------------------------------------------------------------
```

Gates Report:

```
   ╞=============================================================
     Generated by:           Genus(TM) Synthesis Solution 17.22-s017_1
     Generated on:           Jan 11 2022  02:22:09 pm
     Module:                 inverter
     Operating conditions:   slow (balanced_tree)
     Wireload mode:          enclosed
     Area mode:              timing library
   =============================================================


    Gate  Instances  Area  Library
   ----------------------------------
   INVXL           1  2.271     slow
   ----------------------------------
   total           1  2.271


        Type       Instances  Area Area %
   ----------------------------------------
   inverter                1 2.271  100.0
   physical_cells          0 0.000    0.0
   ----------------------------------------
   total                   1 2.271  100.0
```

9: CMOS BASIC GATES & FLIP FLOPS

1 CMOS BASIC GATES

Aim:

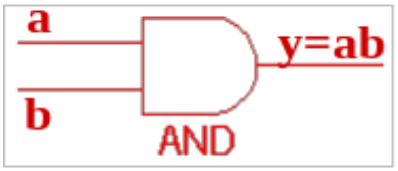
Design and Simulate a **CMOS Basic Gates and Flip Flops**

To write verilog code for a CMOS Basic Gates and its test bench for verification using incisive simulator, observe the waveform and synthesize the code with technological library with given constraints.

Theory:

Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NAND, NOR and XOR gates. The basic operations are described below with the aid of truth tables.

a: AND Gate:



2 Input AND gate		
a	b	y=ab
0	0	0
0	1	0
1	0	0
1	1	1

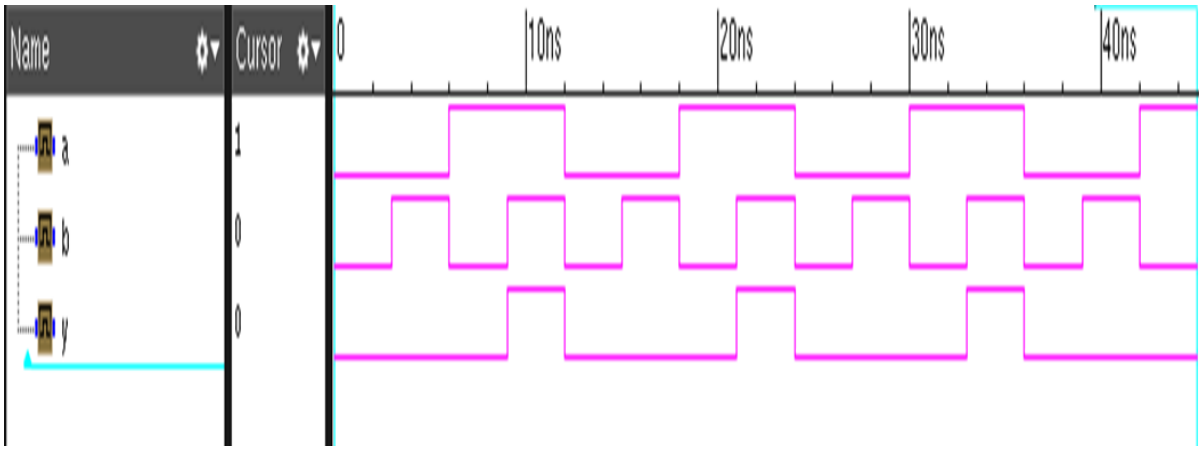
The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. "**a**" dot (.) is used to show the AND operation i.e. "**a.b**". Bear in mind that this dot is sometimes omitted i.e. **ab**

Verilog and Test bench Code for AND Gate:

```
// Verilog source code:
    module andgate (a, b, y);
    input a, b;
    output y;
    assign y = a & b;
    endmodule

// Test-bench source code :
    module tb_and_gate;
    reg a,b;
```

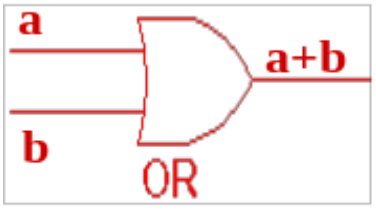
```
wire y;
andgate test (.a(a), .b(b), .y(y));
//Above style is connecting by names
initial begin
a =1'b0;
b = 1'b0;
#45 $finish;
end
always #6 a =~a;
always #3 b =~b;
always @(y)
$display( "time =%0t \t INPUT VALUES: \t a=%b b =%b \t output value y =%b",$time,a,b,y);
endmodule
```

Simulation Result For AND Gate:



Schematic Capture of AND Gate:

b: OR Gate:



2 Input OR gate		
a	b	y=a+b
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (**1**) if one or more of its inputs are high. "**a**" plus (+) is used to show the **OR operation**.

Verilog and Test bench Code for OR Gate:

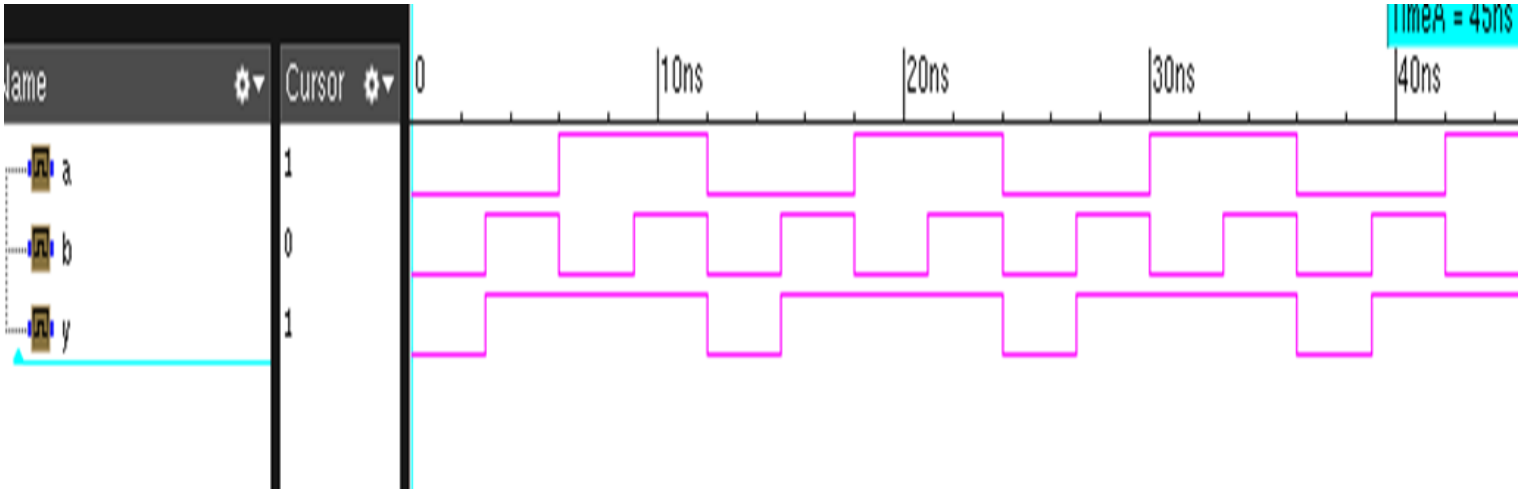
// Verilog source code:

```
module orgate (a, b, y);
input a, b;
output  y;
assign y = a | b;
endmodule
```

// Test bench source code:

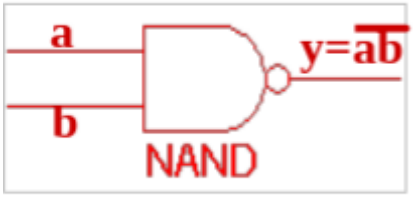
```
module  tb_or_gate;
reg a,b;
wire y;
orgate test (.a(a), .b(b), .y(y));
//Above style is connecting by names
initial begin
a =1'b0;
b = 1'b0;
#45 $finish;
end
always #6 a =~a;
always #3 b =~b;
always @(y)
$display( "time =%0t \t INPUT VALUES: \t a=%b b =%b \t output value y =%b"$time,a,b,y);
endmodule
```

Simulation Result for OR Gate:



Schematic Capture of OR Gate:

c: NAND Gate:



2 Input NAND Gate		
a	b	$y=\overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Verilog and Test bench Code for NAND Gate:

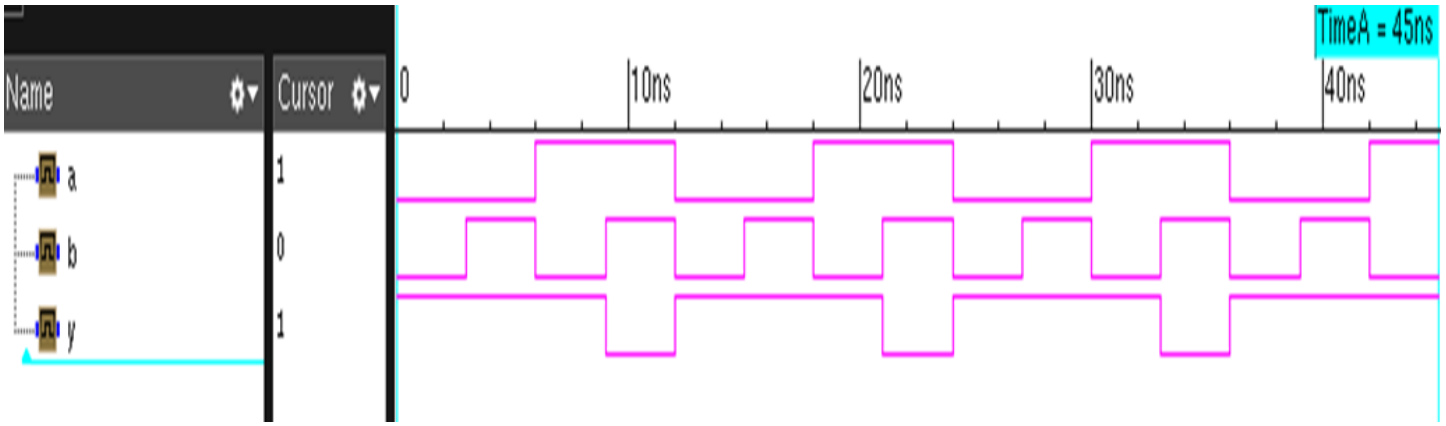
// Verilog source code:

```
module nandgate (a, b, y);
input a, b;
output y;
assign y = ~(a & b);
endmodule
```

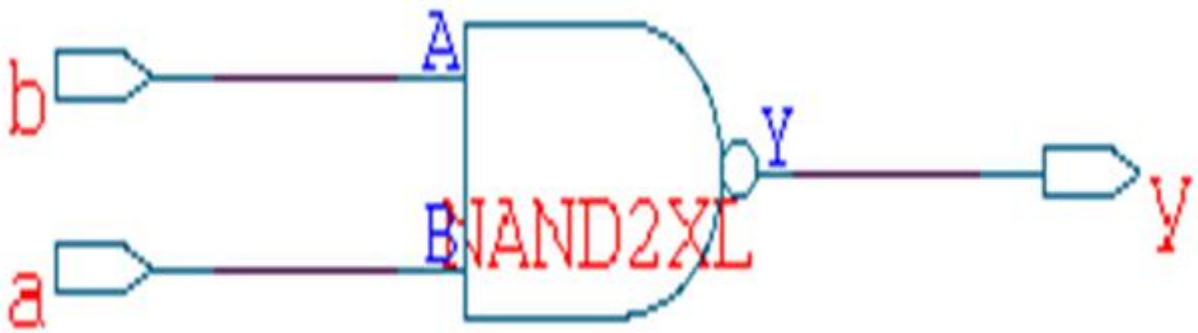
// Test-bench source code:

```
module tb_nand_gate;
reg a,b;
wire y;
nandgate test (.a(a), .b(b), .y(y));
//Above style is connecting by names
initial begin
a =1'b0;
b = 1'b0;
#45 $finish;
end
always #6 a =~a;
always #3 b =~b;
always @(y)
$display( "time =%0t \t INPUT VALUES: \t a=%b b =%b \t output value y =%b",$time,a,b,y);
endmodule
```

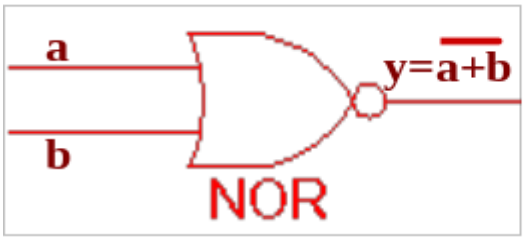
Simulation Result for NAND Gate:



Schematic Capture of NAND Gate:



d: NOR Gate:



2 Input NOR Gate		
a	b	$y=\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

This is a **NOT-OR** gate which is equal to an OR gate followed by a NOT gate. The outputs of all **NOR** gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

Verilog and Test bench Code for NOR Gate:

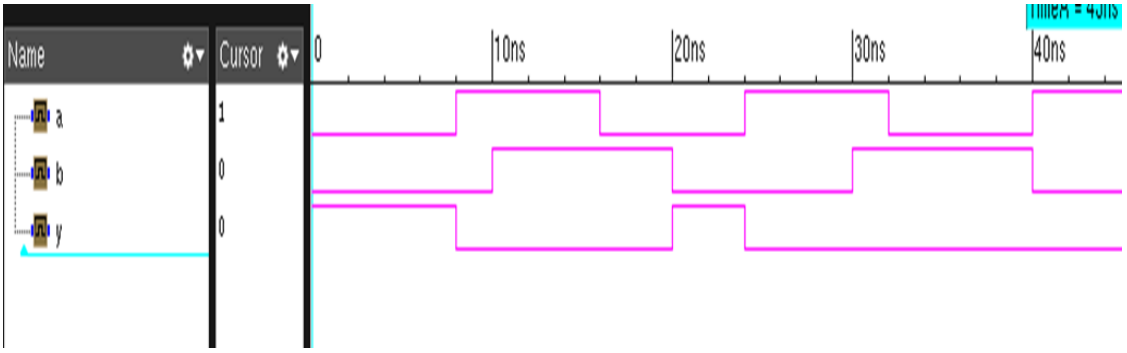
// Verilog Source Code:

```
module norgate (a, b, y);
input a, b;
output y;
assign y = ~(a | b);
endmodule
```

// Test-bench Source Code:

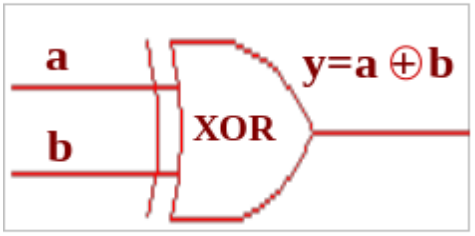
```
module  tb_nor_gate;
reg a,b;
wire y;
norgate test (.a(a), .b(b), .y(y));
//Above style is connecting by names
initial begin
a =1'b0;
b = 1'b0;
#45 $finish;
end
always #8 a =~a;
always #10 b =~b;
always @(y)
$display( "time =%0t \t INPUT VALUES: \t a=%b b =%b \t output value y =%b",$time,a,b,y);
endmodule
```

Simulation Result for NOR Gate:



Schematic Capture of NOR Gate:

e: XOR Gate:



2 Input XOR Gate		
a	b	y= a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

The '**Exclusive-OR**' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign (⊕) is used to show the **XOR** operation.

Verilog and Test bench Code for XOR Gate:

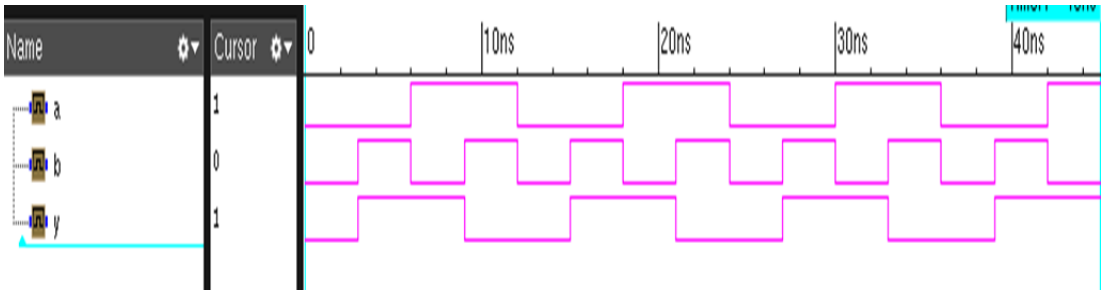
// Verilog Source Code:

```
module xorgate (a, b, y);
input a, b;
output y;
assign y = a ^ b;
endmodule
```

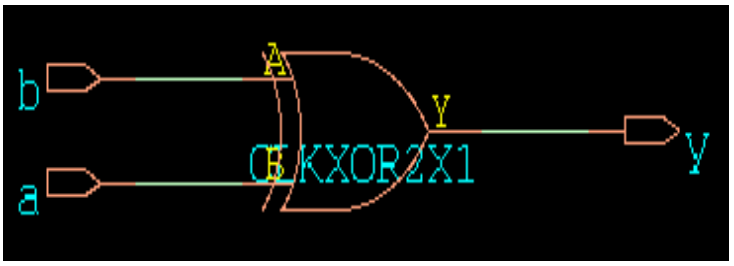
// Test-bench Source Code:

```
module  tb_xor_gate;
reg a,b;
wire y;
xorgate test (.a(a), .b(b), .y(y));
//Above style is connecting by names
initial begin
a =1'b0;
b = 1'b0;
#45 $finish;
end
always #6 a =~a;
always #3 b =~b;
always @(y)
$display( "time =%0t \t INPUT VALUES: \t a=%b b =%b \t output value y =%b",$time,a,b,y);
endmodule
```

Simulation Result for XOR Gate:



Schematic Capture for XOR Gate:



2 FLIP FLOPS

Aim:

Design and Simulate a **Flip Flops** (SR, JK, MS, D and T)

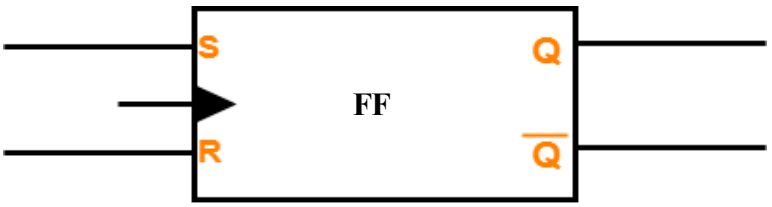
To write verilog code for Flip-Flops and its test bench for verification using incisive simulator, observe the waveform and synthesize the code with technological library with given constraints.

Theory:

In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multi-vibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

Lab 2.2a: SR-Flip Flop:

SR flip-flop is one of the fundamental **sequential circuits**. **SR flip flop is the simplest type of flip flop basically a one-bit memory storage device that has two inputs, one which will "Set" the device (i.e. the output is 1), and is labelled as S and other which will Reset the device (i.e. the output is 0), labelled R. The name SR** stands for "**Set-Reset**".



Logic Symbol

INPUTS			OUTPUTS	REMARKS
S	R	Q_n (Present State)	Q_{n+1} (Next State)	States and Conditions
0	0	X	Q_n	Hold State condition S = R = 0
0	1	X	0	Reset state condition S = 0 , R = 1
1	0	X	1	Set state condition S = 1 , R = 0
1	1	X	Indeterminate	Indeterminate state condition S = R = 1

Truth Table

Verilog and Test bench Code for SR Flip Flop:

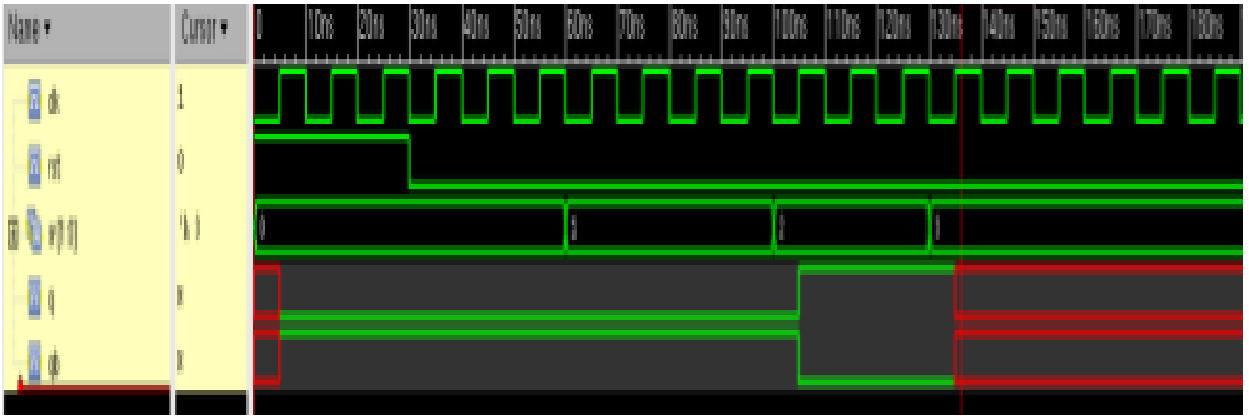
// Verilog Source Code:

```
module SR_flipflop(q,q1,r,s,clk);
output q,q1;
input r,s,clk;
reg q,q1;
initial begin
q=1'b0; q1=1'b1;
end
always @(posedge clk)
begin
case({s,r})
{1'b0,1'b0}: begin q=q; q1=q1; end
{1'b0,1'b1}: begin q=1'b0; q1=1'b1; end
{1'b1,1'b0}: begin q=1'b1; q1=1'b0; end
{1'b1,1'b1}: begin q=1'bx; q=1'bx; end
endcase
end
endmodule
```

// Test-bench Source Code:

```
module test;
reg clk=0; reg s=0; reg r=0;
wire q, qnot;
jkff dut(reset, clk, j, k, q, qnot);
initial begin
s=1'b1; r=1'b1;
#25 $finish;
end
always #1 clk=~clk;
endmodule
```

Simulation Result for SR-Flip Flop:


Name
Cursor
clk
rst
q
qb

2b: JK-Flip Flop:

JK flip flop is a refined & improved version of SR flip-flop that has been introduced to solve the problem of indeterminate state that occurs in SR flip flop when both the inputs are 1. The JK flip-flop is probably the most widely used and is considered the universal flip-flop because it can be used in many ways.

Logic Symbol

Verilog and Test bench Code for JK Flip Flop:

INPUTS			OUTPUTS	REMARKS
J	K	Q_n (Present State)	Q_{n+1} (Next State)	States and Conditions
0	0	X	Q_n	Hold State condition J = K = 0
0	1	X	0	Reset state condition J = 0 , K = 1
1	0	X	1	Set state condition J = 1 , K = 0
1	1	X	Q'_n	Toggle state condition J = K = 1

Truth Table

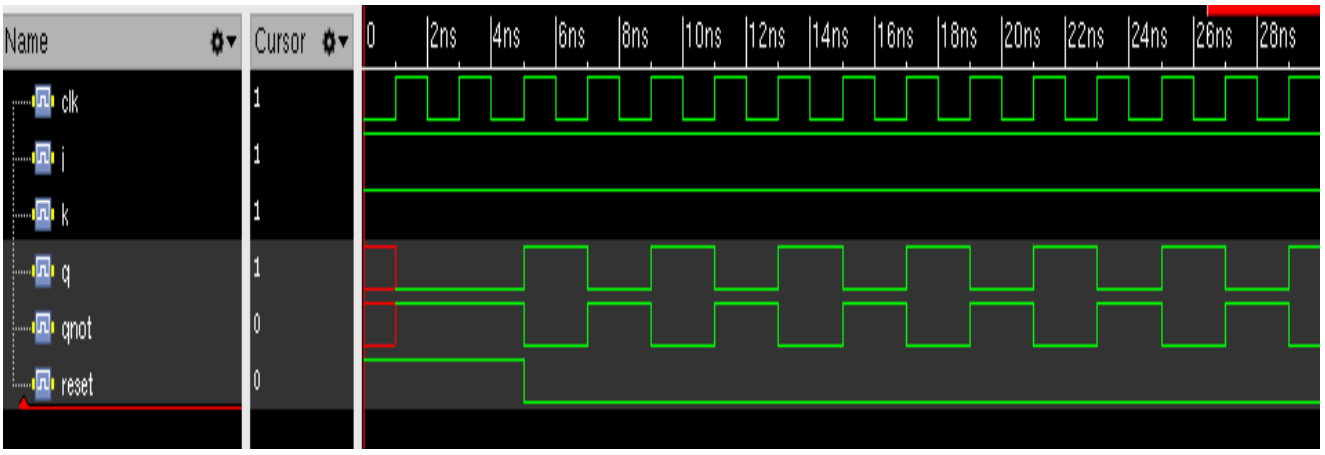
// Verilog Source Code:

```
module jkff(input reset, input clk, input j, input k, output reg q, output qnot);
assign qnot=~q;
always @(posedge clk)
if (reset) q<=1'b0;
else
case ({j, k})
2'b00: q<=q;
2'b01: q<=1'b0;
2'b10: q<=1'b1;
2'b11: q<=~q;
endcase
endmodule
```

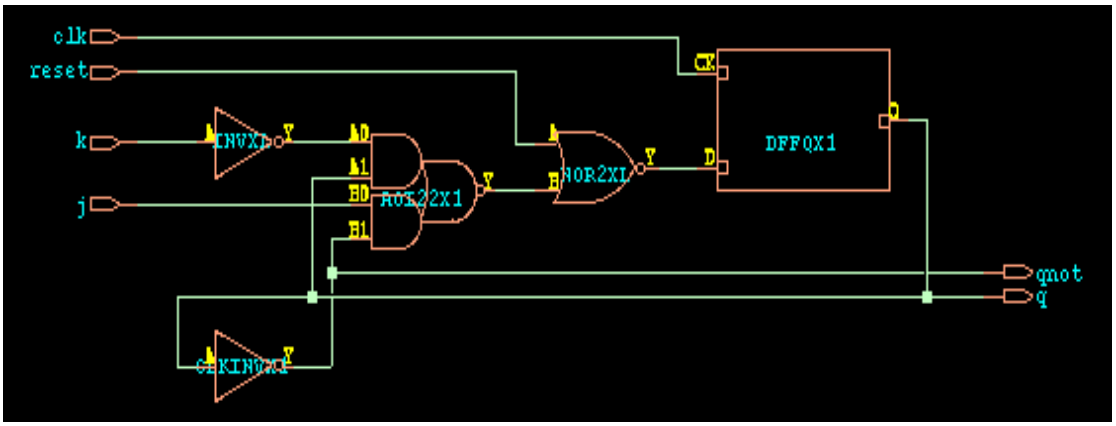
// Test-bench Source Code:

```
module test;
reg clk=0;
reg j=0;
reg  k=0;
reg reset=1;
wire q, qnot;
jkff dut(reset, clk, j, k, q, qnot);
initial
begin
$dumpfile("dump.vcd");
$dumpvars(1);
j=1'b1; // set your JK here
k=1'b1;
#5 reset=1'b0;
#25 $finish;
end
always #1 clk=~clk;
endmodule
```

Simulation Result for JK-Flip Flop:



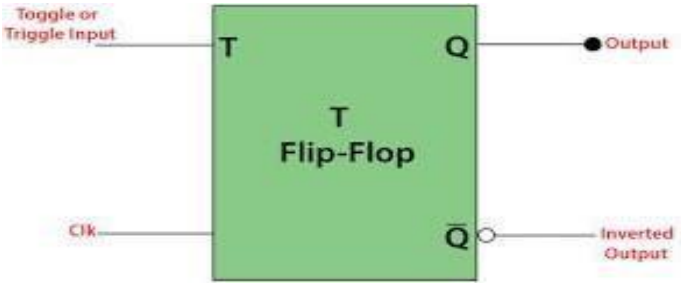
Schematic Capture of JK Flip Flop:


clk
reset
k
j
INVX1
AOI22X1
NOR2XI
DFFQX1
CK
D
Q
qnot
q

c: T-Flip Flop:

T flip flop is known as a **toggle** flip flop because of its toggling operation. It is a modified form of the JK flip flop. A T-flip flop is constructed by connecting J and K inputs, creating a single input called T. Hence why a T flip flop is also known as a **single input JK flip flop**.

The defining characteristic of T flip flop is that it can change its output state. You can change the output signal from one state (on or off) to another state (off or on). The clock signal must set high to toggle the output. When the clock is set low, the output remains as it is whether the input signal is set high or low. So, to change the output condition, the clock signal has to be high.



Symbol of T-FF

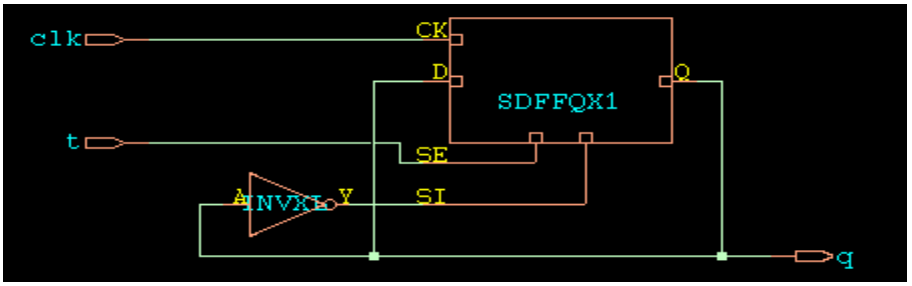
T	Qn	Qn+1	Action
0	0	0	Unchanged/hold
0	1	1	Unchanged/hold
1	0	1	Toggle
1	1	0	Toggle

Truth Table of T-FF

Verilog and Test bench Code for T Flip Flop:

// Verilog Source Code:

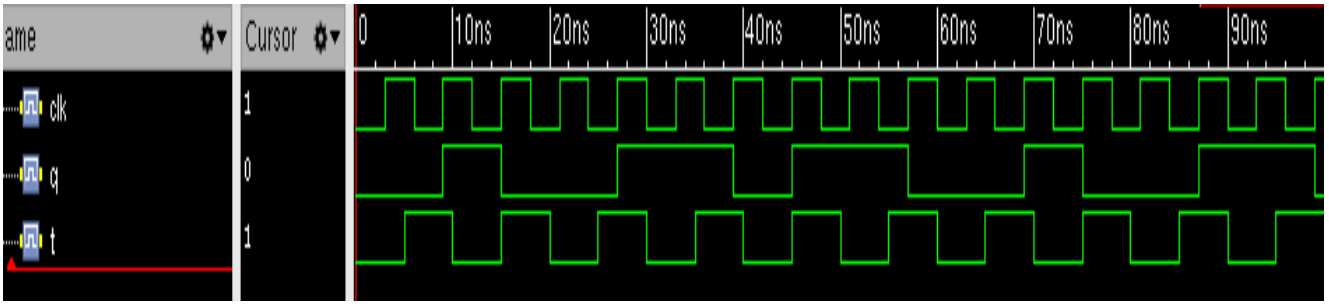
```
module tffmod(t, clk, q);
input t; input clk;
output q;
reg q;
initial
q <= 0;
always @(posedge clk)
q <= q^t;
endmodule
```



// Test-bench Source Code:

```
module tflipflopt_b;
reg t;
reg clk;
wire q;
tffmod uut (.t(t), .clk(clk), .q(q));
initial begin
t = 0; clk = 0;
#100;
end
always #3 clk=~clk;
always #5 t=~t;
initial
#100 $stop;
endmodule
```

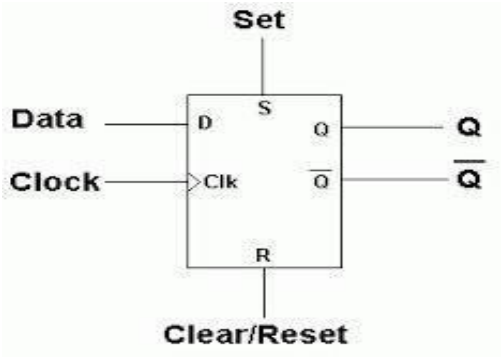
Simulation Result for T-Flip Flop:



d: D-Flip Flop:

D Flip-Flop is one of that Flip Flop that can store data. It can be used to store data statically or dynamically depends on the design of the circuit. D Flip-Flop is used in many sequential circuits as register, counter, etc.

D flip-flop or Data flip flop is a type of flip Flop that has only one data input that is ‘D’ and one clock pulse input with two outputs Q and Q bar. This Flip Flop is also called a delay flip flop because when the input data is provided into the d flip-flop, the output follows the input data delay by one clock pulse.



Symbol of D-FF

Input			Output	
D	reset	clock	Q	Q'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Characteristics table of D-FF

Module 1: Writing RTL Code for D-Flip Flop

Verilog and Test bench Code for D Flip Flop:

// Verilog Source Code:

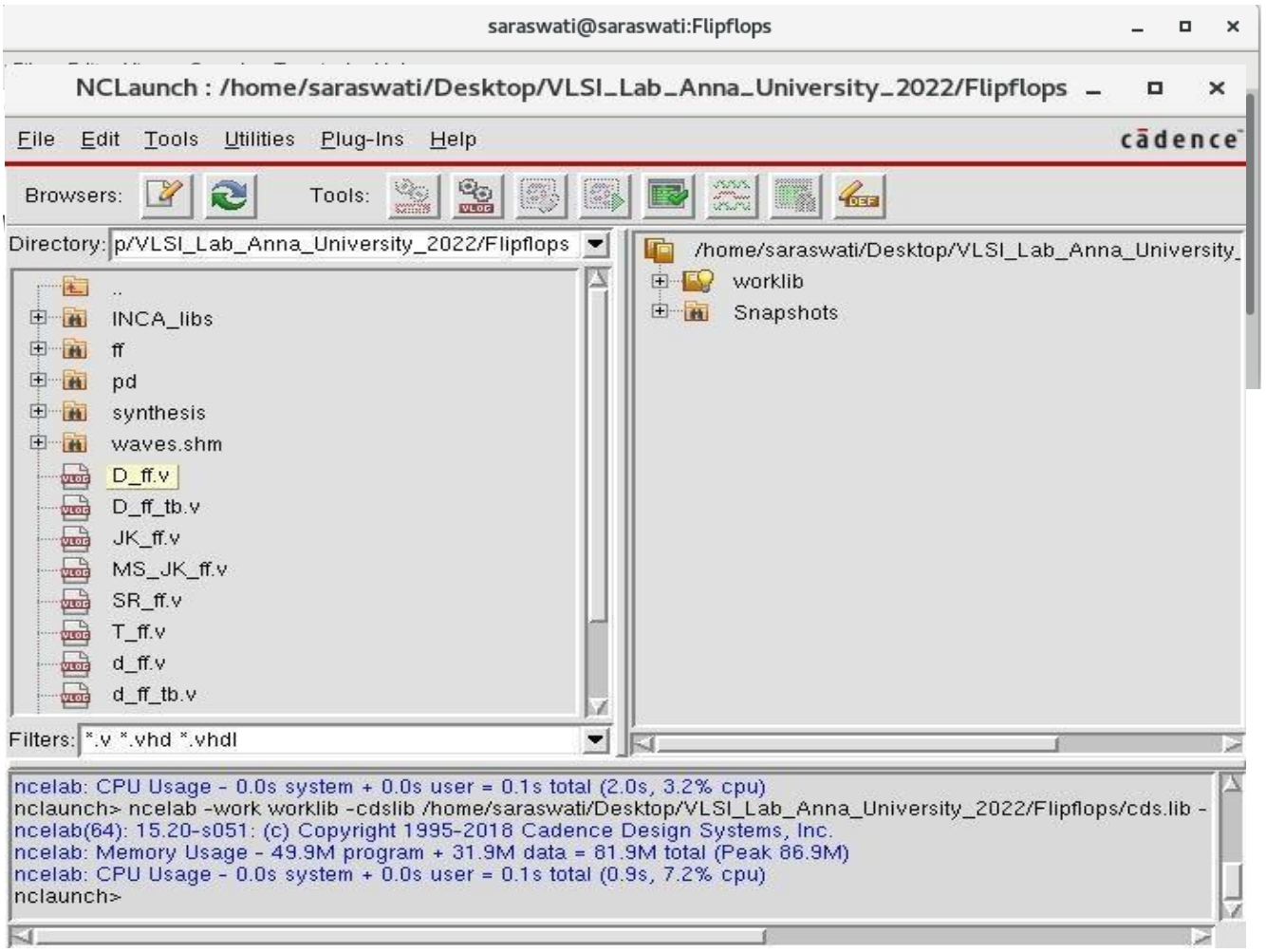
```
module dff_sync_reset (data,clk,reset,q);
input data, clk, reset ;
output q;
reg q;
always @ ( posedge clk)
if (~reset) begin
q <= 1'b0;
end else begin
q <=  data;
end
endmodule
```

// Test-bench Source Code:

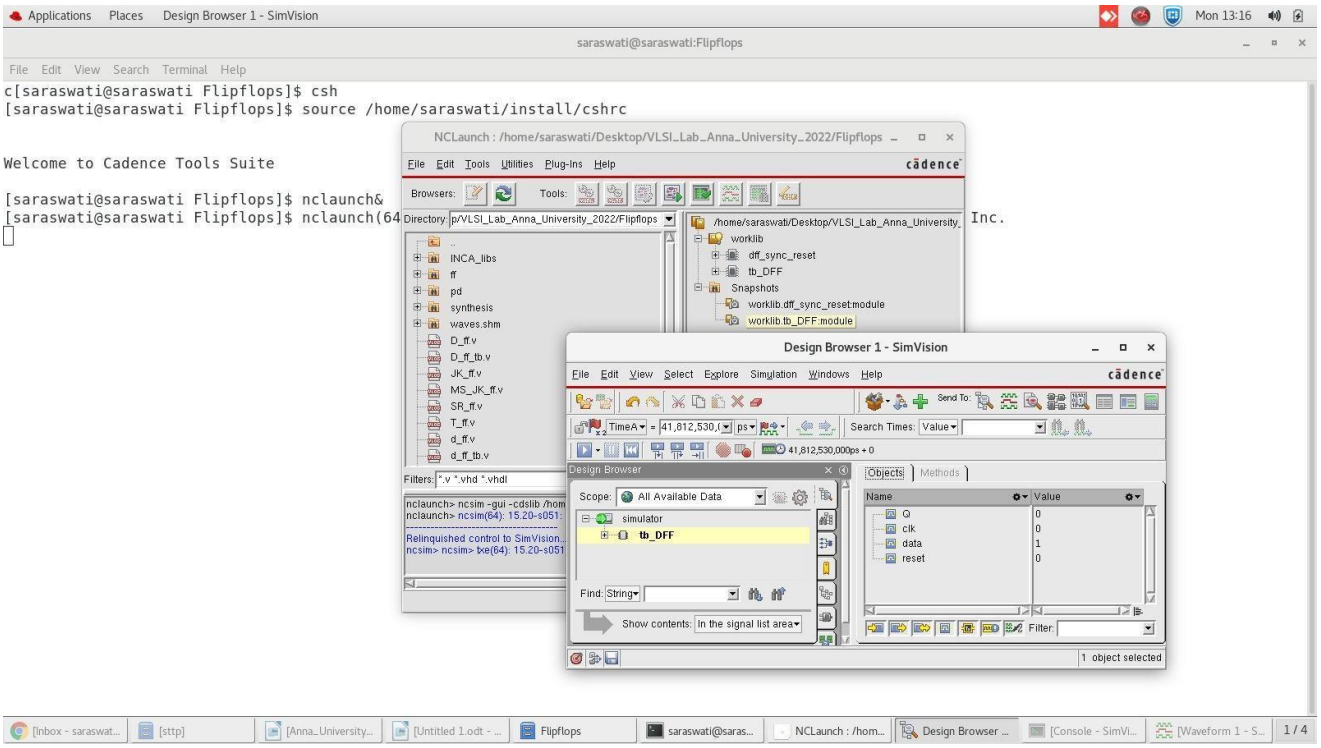
```
module tb_DFF();
reg data;
reg clk;
reg reset;
wire Q;
dff_sync_reset dut(data, clk, reset, Q);
initial begin
clk=0;
forever #10 clk = ~clk;
end
initial begin
reset=1;
data <= 0;
#100;
reset=0;
data <= 1;
```

```
#100;
data <= 0;
#100;
data <= 1;
end
endmodule
```

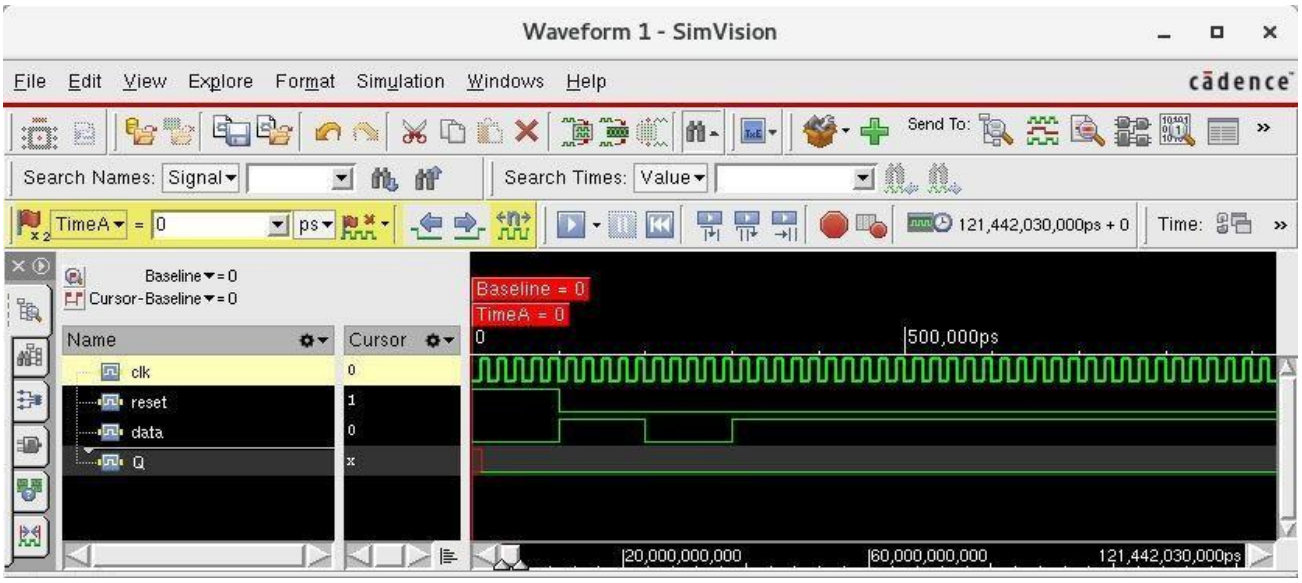
Module 2: Design Simulation using INCISIVE Simulator



You can Compile, Elaborate and Simulate the D-flip flop with source codes available with verilog and test-bench and observe the simulation output as shown below.



Simulation Result for D-Flip Flop:



Module 3: Synthesis Using GENUS Tool

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

Open ▾ genus_script.tcl ~/Desktop/VLSI_Lab_Anna_University_2022/Flipflops/ff Save

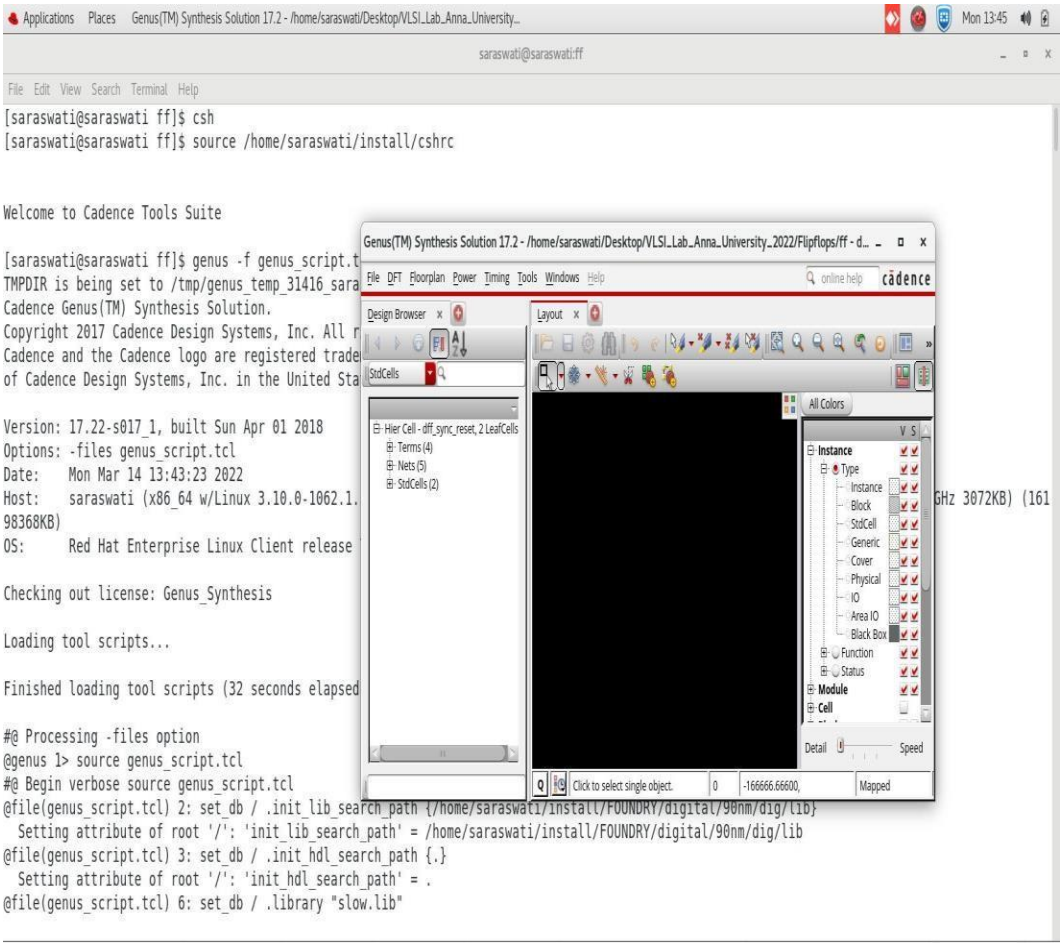
```
# Setting library and RTL paths
set_db / .init_lib_search_path {/home/saraswati/install/FOUNDRY/digital/90nm/dig/lib}
set_db / .init_hdl_search_path {.}

# Read Lib, RTL and SDC files
set_db / .library "slow.lib"
set DESIGN dff_sync_reset
read_hdl "D_ff.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 > D_ff_netlist.v
write_sdc > D_ff_sdc.sdc
# PPA Reports
report_power > D_ff_power.rpt
report_area > D_ff_area.rpt
report_timing > D_ff_timing.rpt
report_gates > D_ff_gatecount.rpt
report_qor -levels_of_logic -power -exclude_constant_nets > D_ff_qor.rpt
gui_show
```

Tcl ▾ Tab Width: 8 ▾ Ln 31, Col 1 ▾ INS

Chip Level SDC (constraints_top.sdc):

```
#-----> 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]
```

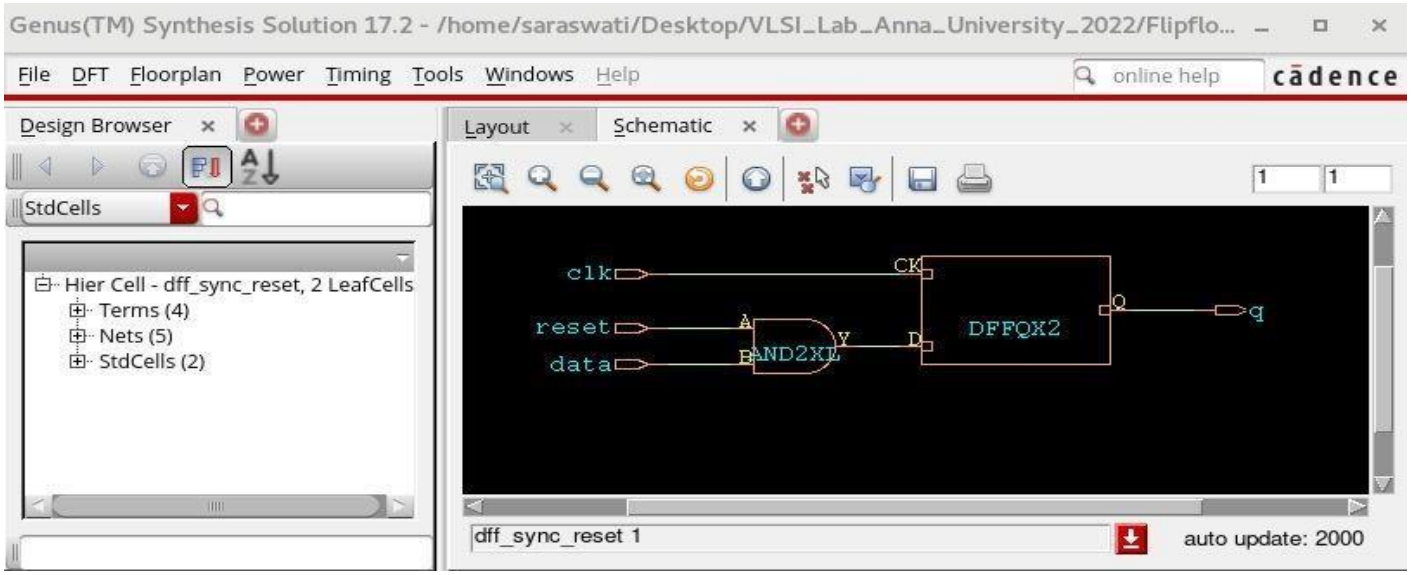
Invoke the Genus tool by typing the below command on your terminal: **genus -f genus_script.tcl**
The tcl [Tool Command Language] script runs executing each command one after the other.



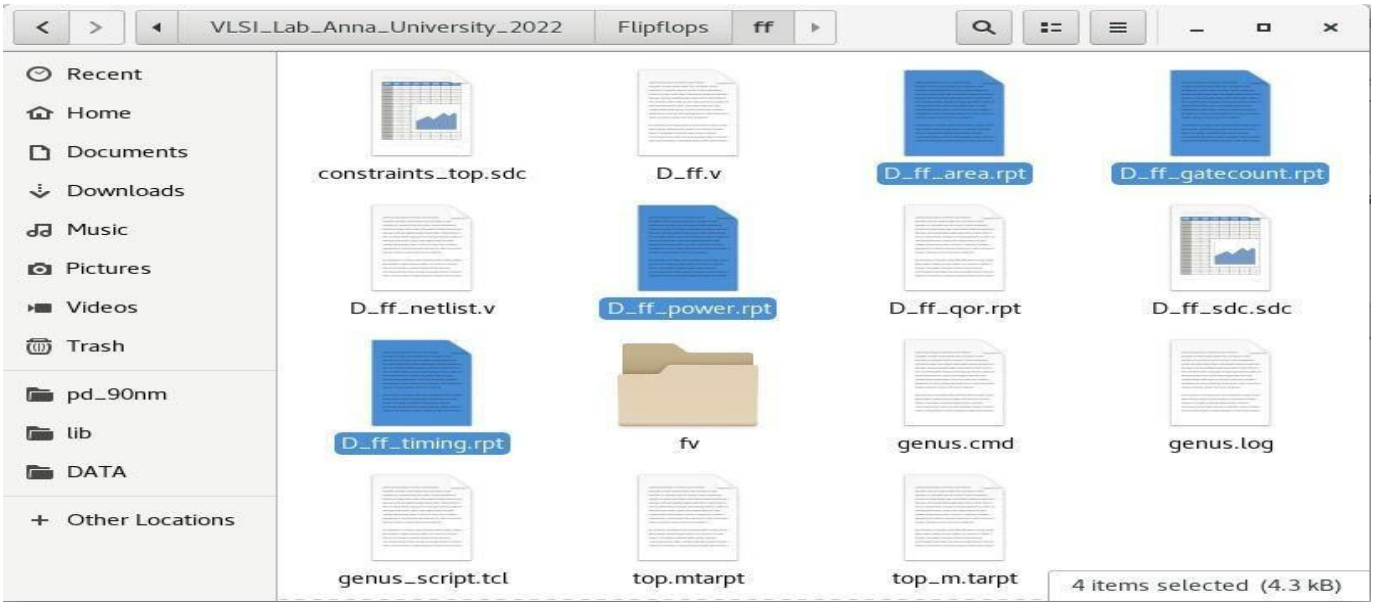
A window of **Genus GUI** pops – up with the top hier cell on the left top. Make a **Right Click** and select **Schematic Viewer → In Main**.

Or

You can click on + **option** form the Genus GUI where you can choose the **Schematic** option see the schematic capture of D-flip flop Gate level netlist.



Output & Report Generation:



10: 4-BIT SYNCHRONOUS COUNTER

Aim: Design and Simulate a 4-bit Synchronous counter using a Flip-Flops

- Manual/Automatic Layout Generation and Post Layout Extraction
- Analyzing the Power, Area and Timing for Exp 7 to 9 by Performing Pre Layout and Post Layout Simulations.

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

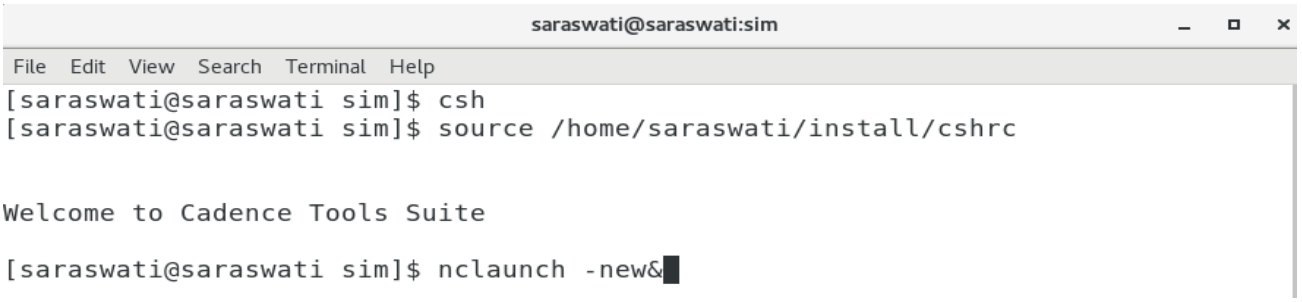
Getting Started:

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.



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

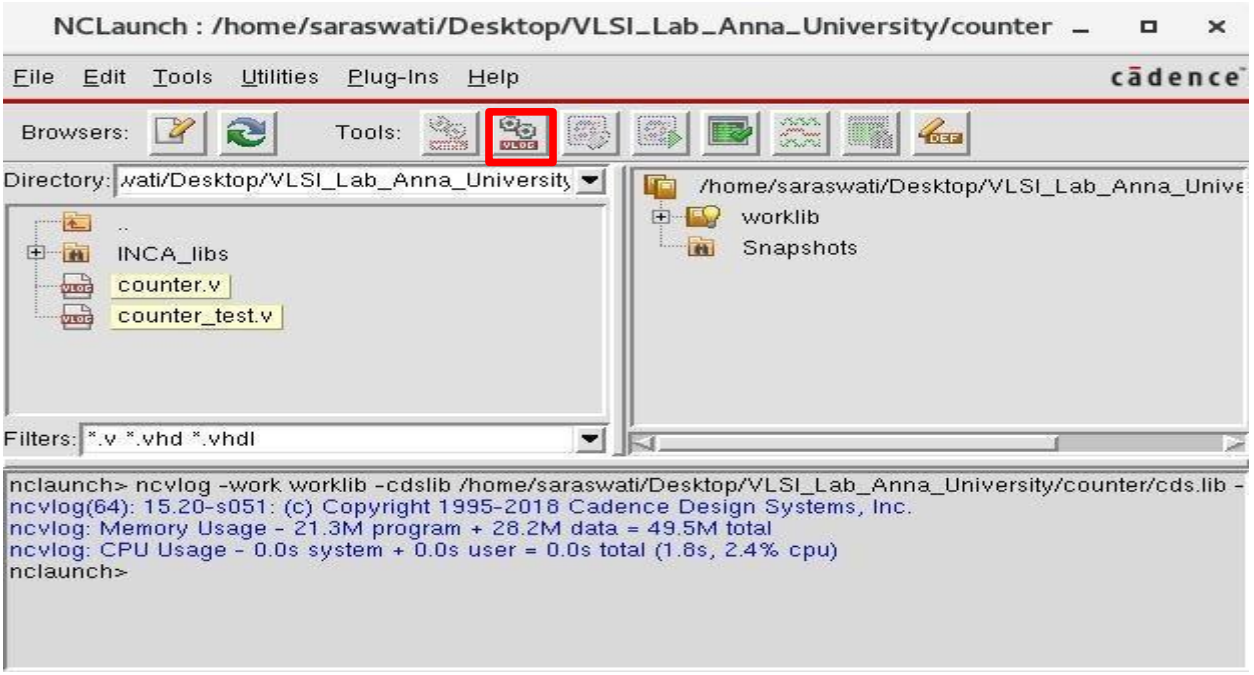
counter.v
~/Desktop/VLSI_Lab_Anna_University_2022/counter

```
// 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
```

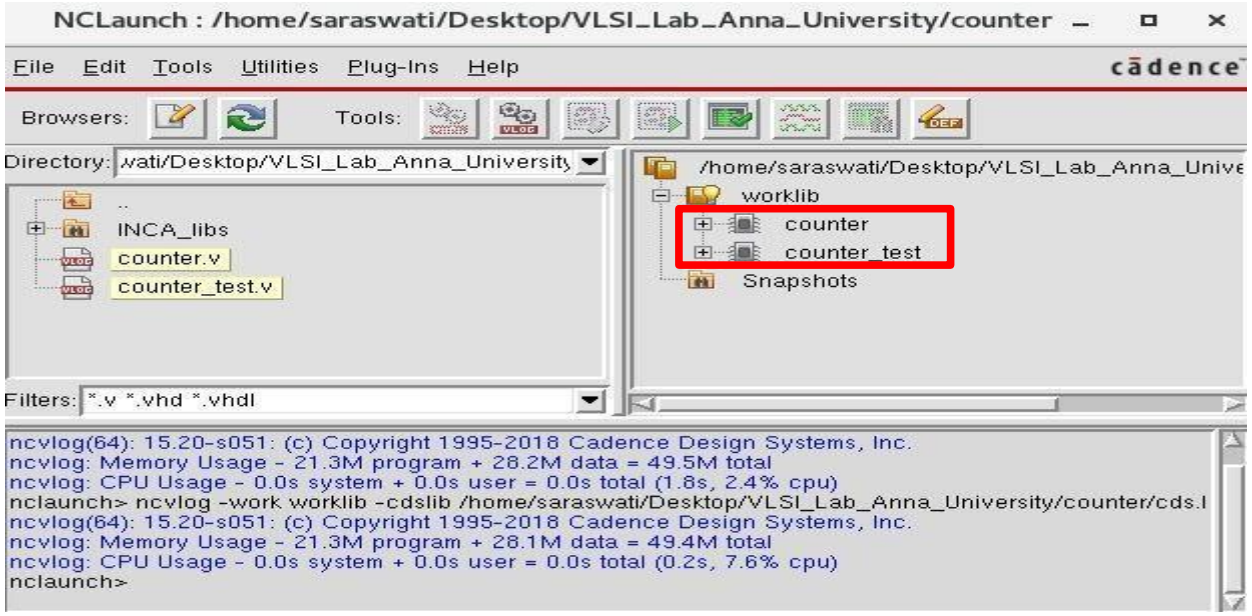
counter_test.v
~/Desktop/VLSI_Lab_Anna_University_2022/counter

```
// 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
```

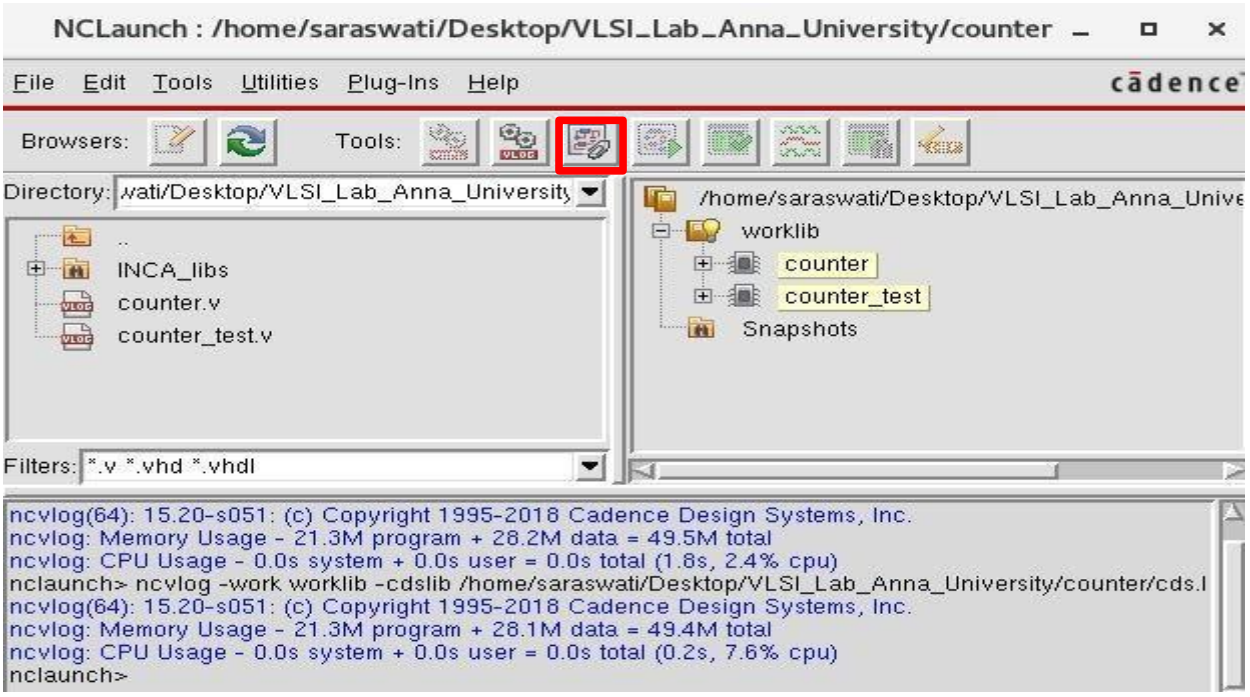
Module 2: Design Simulation using INCISIVE Simulator:



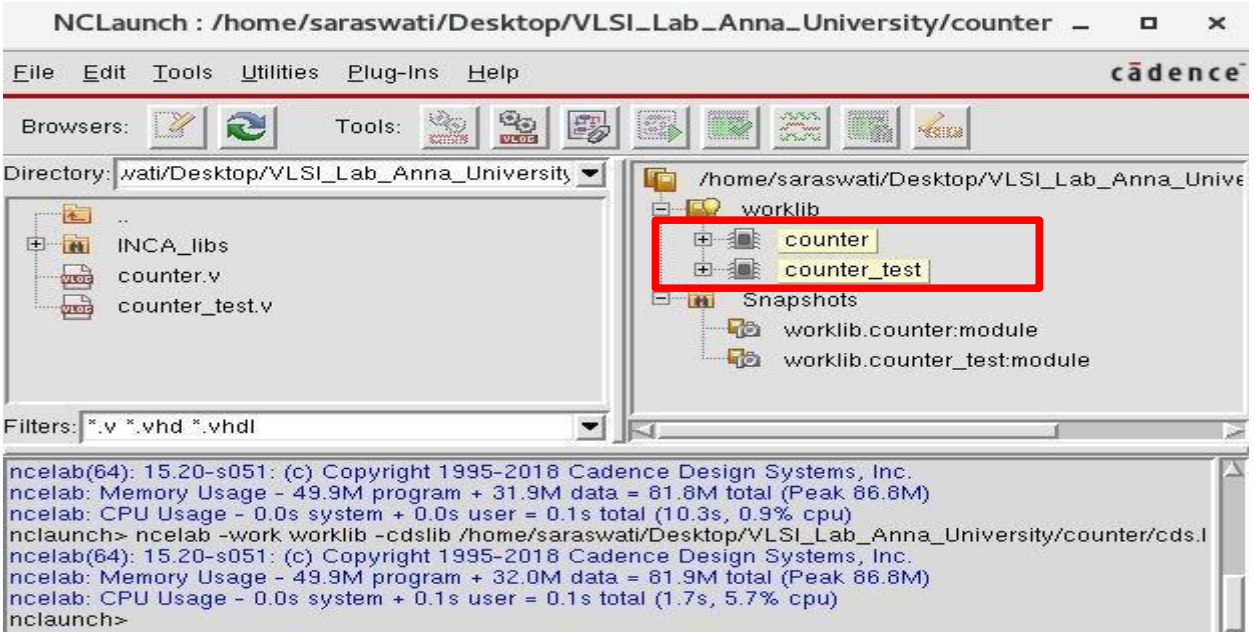
Compiled Data stored in worklib:



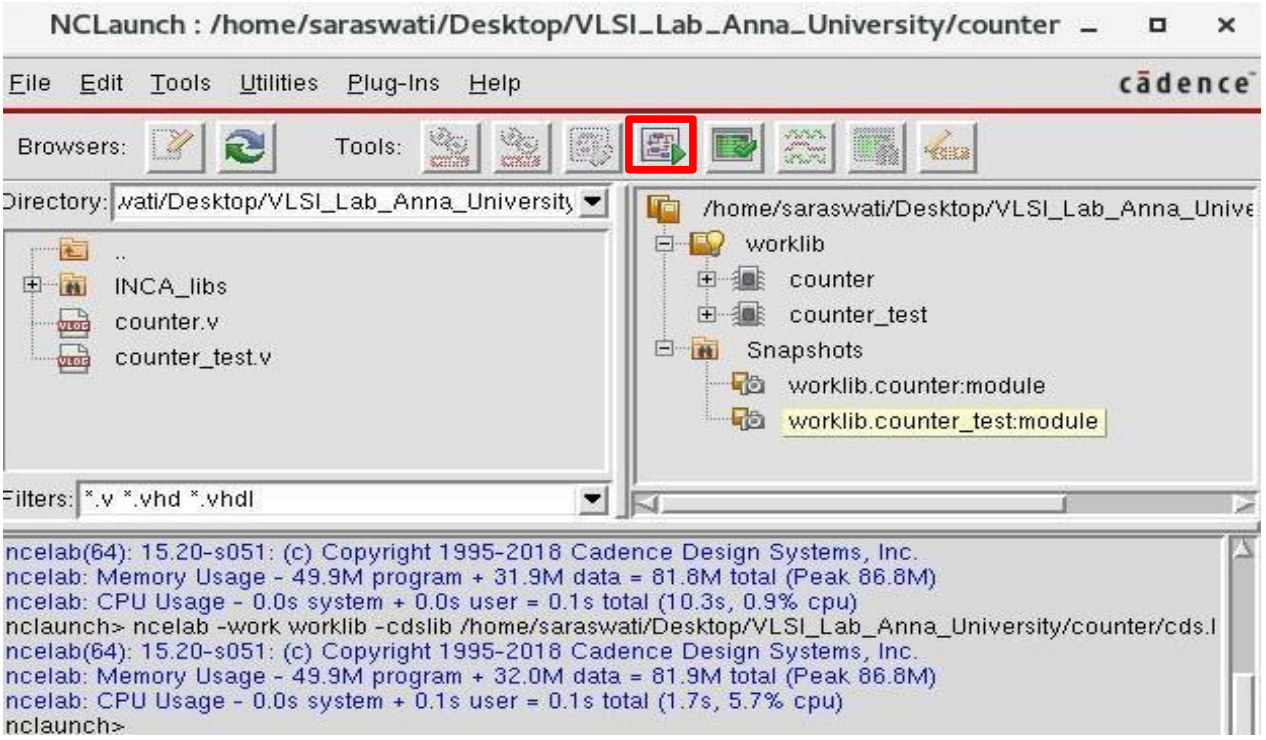
Elaboration:



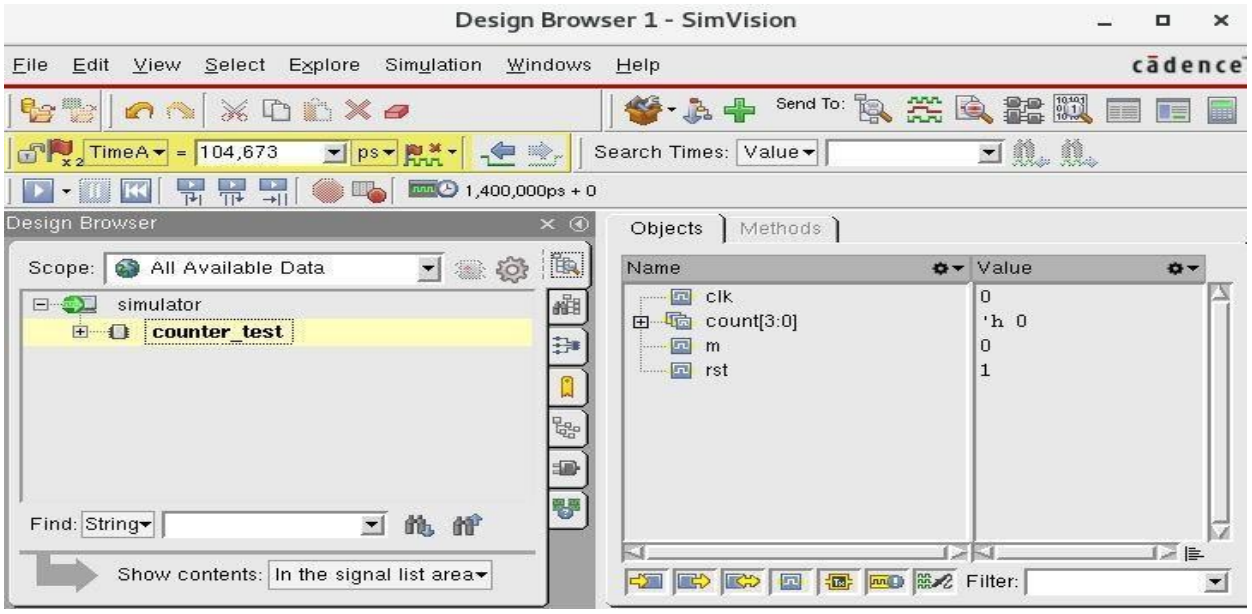
Elaborated Data Stored in Snapshots:



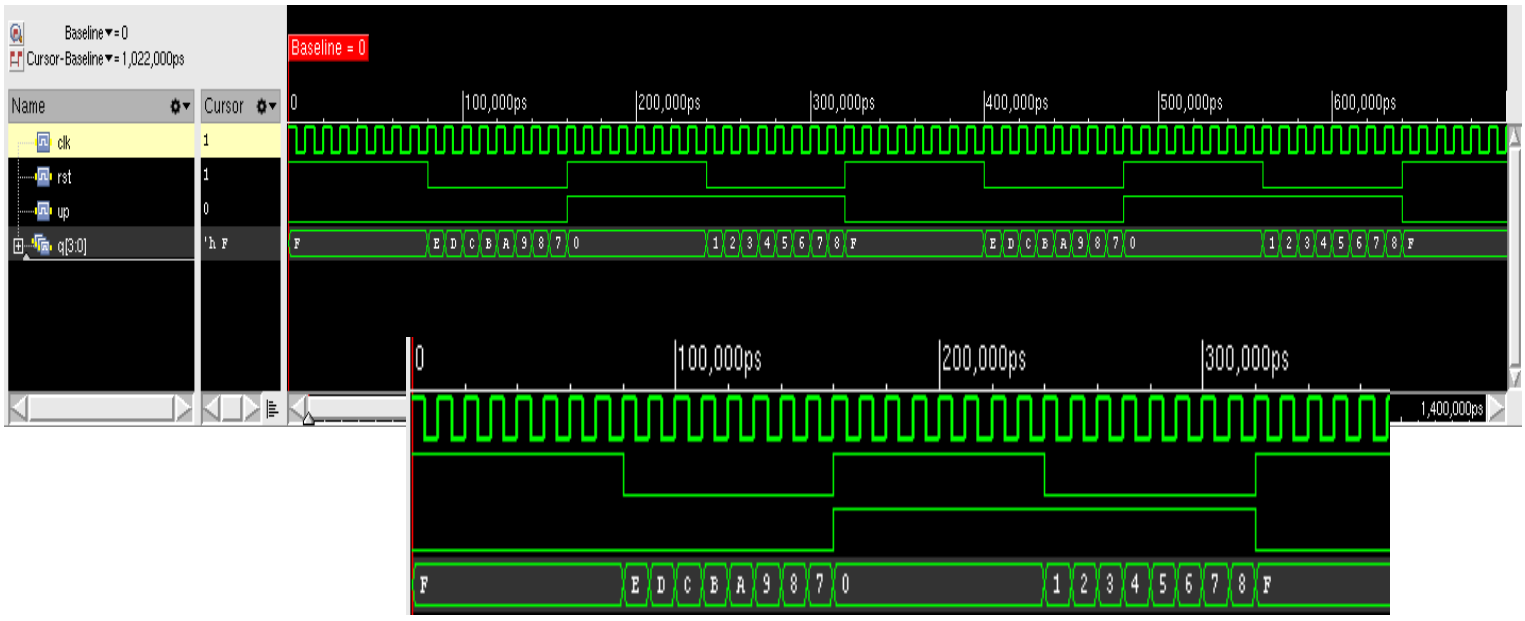
Simulation:



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



Simulation Result:



Console - SimVision

File Edit View Simulation Windows Help cadence

Text Search:

1,400,000ps + 0

```
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.m 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.

Open | genus_script.tcl ~/Desktop/VLSI_Lab_Anna_University/counter | Save

```
# 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:

Open ▾ | constraints_top.sdc — ~/Desktop/VLSI_Lab_Anna_University/counter | Save

```
#-----> 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]
```

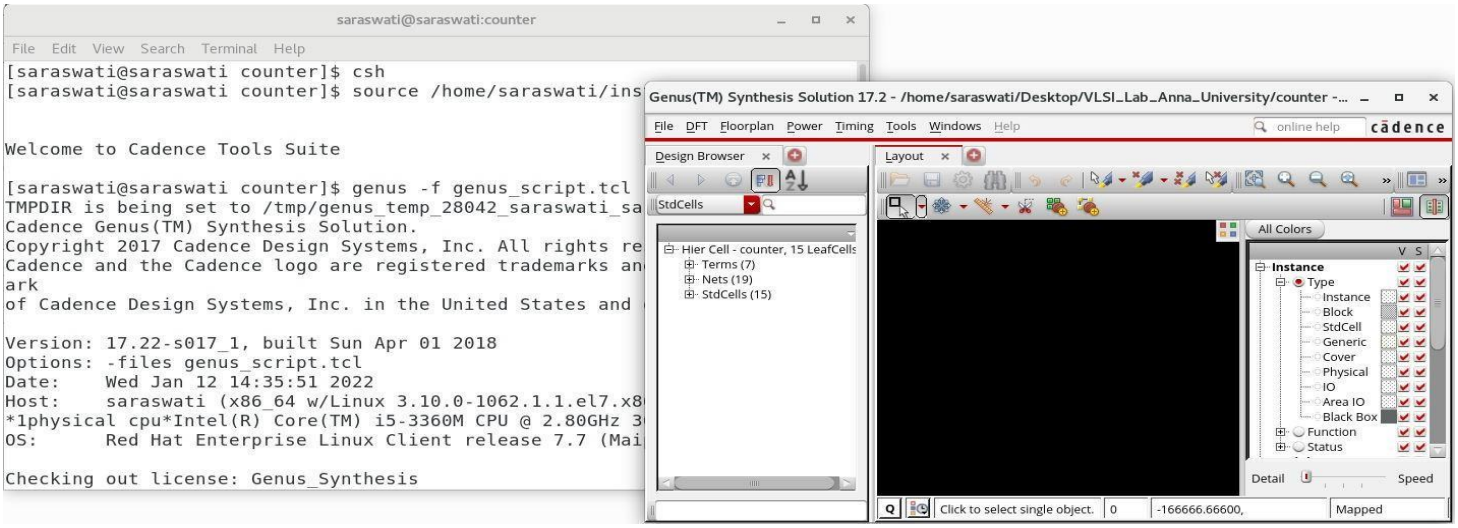
Plain Text ▾ Tab Width: 8 ▾ Ln 17, Col 1 ▾ INS

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.

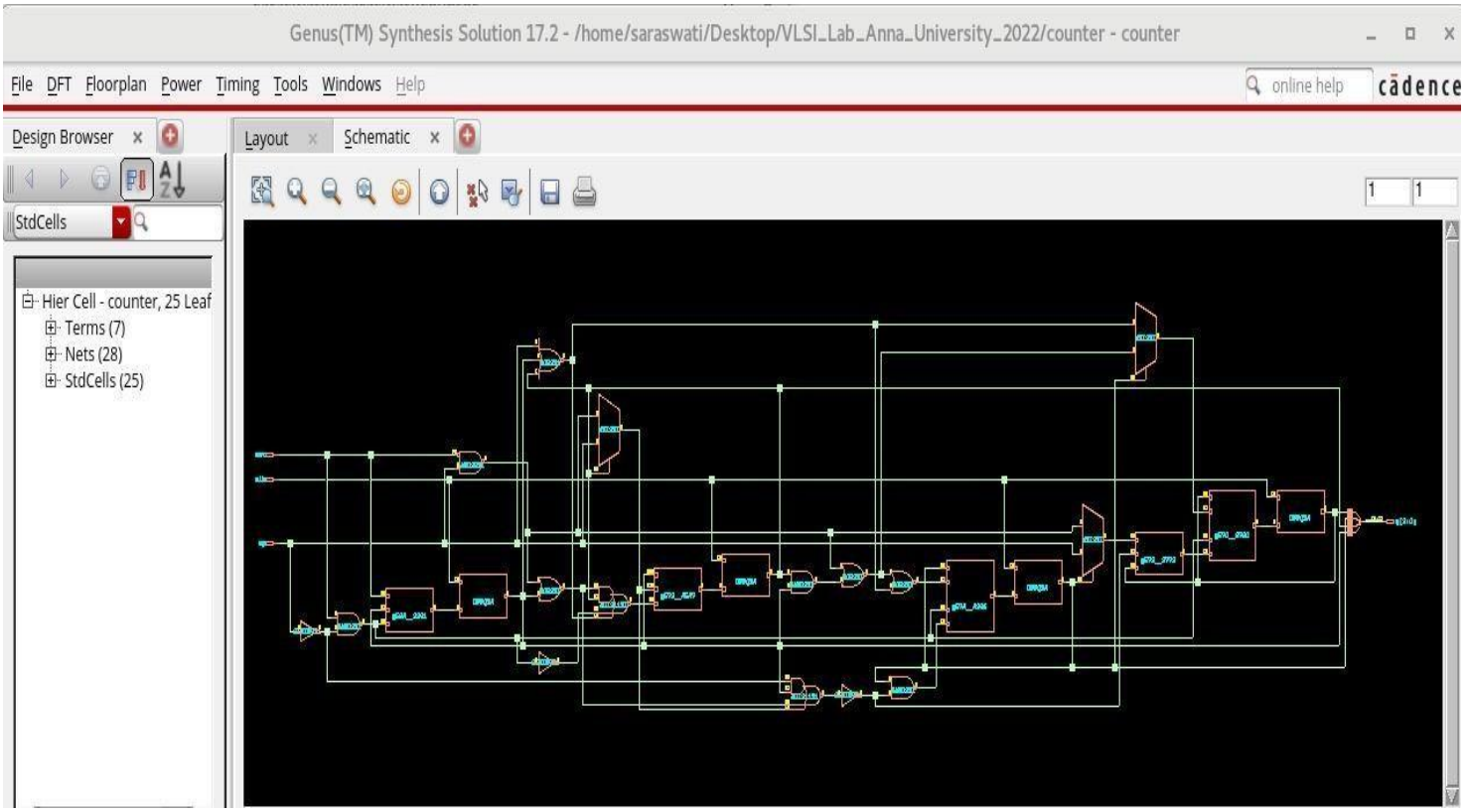
saraswati@saraswati:counter

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
DFFQX4               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 mode:              timing library
============================================================


Path 1: MET (595 ps) Setup Check with Pin q_reg[3]/CK->D
          Group: clk
     Startpoint: (R) q_reg[1]/CK
          Clock: (R) clk
       Endpoint: (F) q_reg[3]/D
          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]/Q     -       CK->Q F     DFFQX4         5 209.8   376   605     605    (-,-)
  g583__7675/Y   -       S0->Y F     MXI2XL         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     OAI21X1        1   2.9   134    93    1161    (-,-)
  g570__8780/Y   -       C0->Y F     OAI211X1       1   1.6   149   111    1272    (-,-)
  q_reg[3]/D     <<<     -     F     DFFQX4         1     -     -     0    1272    (-,-)
#-------------------------------------------------------------------------------------------
```

11. CMOS Inverting Amplifier

Aim:

➔ Design and Simulate a CMOS Inverting Amplifier

➔ Analyze the Input impedance, Output impedance, Gain and Bandwidth for experiments 10 by performing Schematic Simulations.

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 csh at the command prompt to invoke the C shell.

#csh

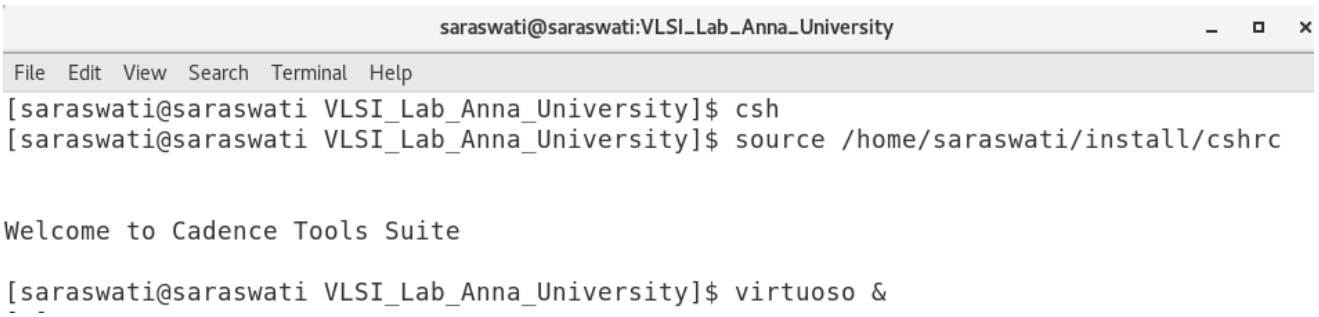
6. To verify that the path to the software is properly set in the cshrc file

#source /home/install/cshrc

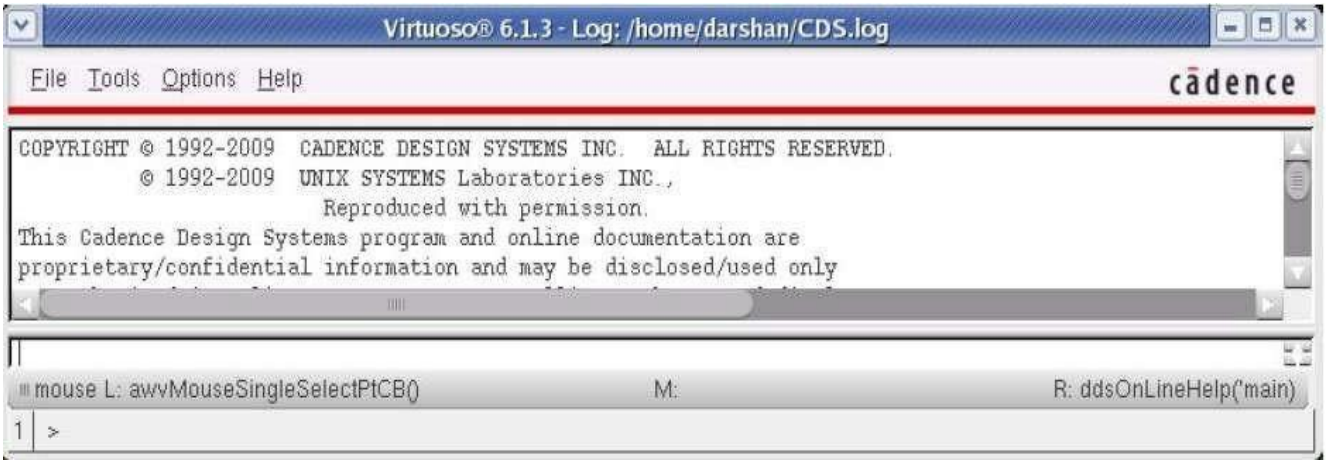
A welcome string "**Welcome to Cadence Tool Suite**" appears indicating terminal ready to invoke Cadence Tools available for you.

7. In the same terminal window, enter:

 # virtuoso &



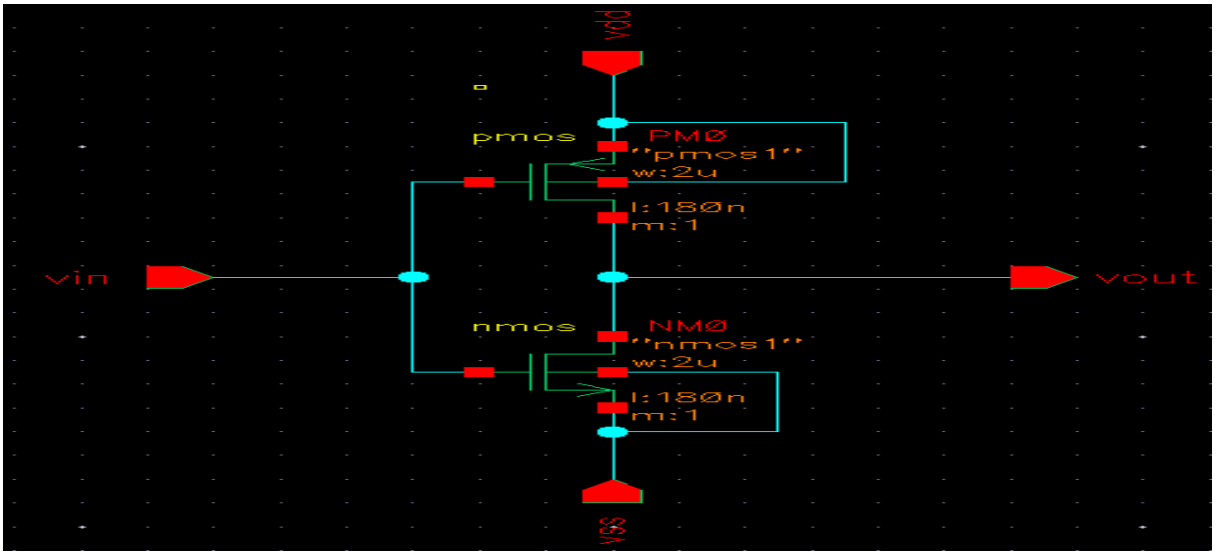
The virtuoso or Command Interpreter Window (CIW) appears at the bottom of the screen.



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 **gpdk180** from the cyclic field and click **OK**.

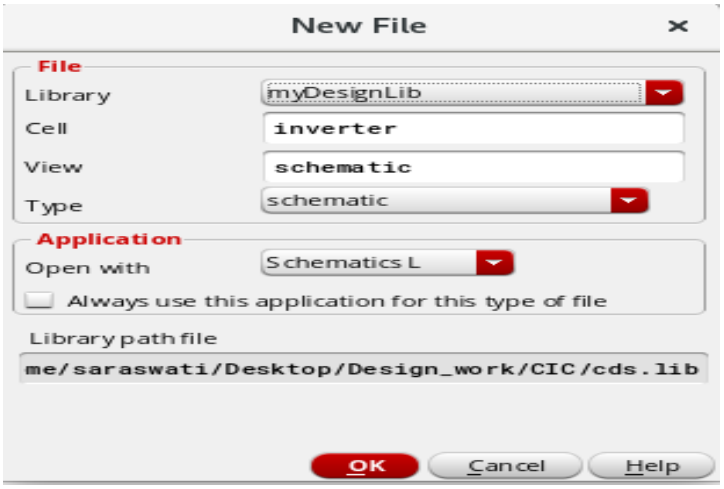
7. After creating a new library you can verify it from the library manager.
8. If you right click on the "**myDesignLib**" and select properties, you will find that **gpdk180** library is attached as techlib to "**myDesignLib**".

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

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.

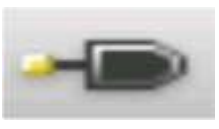
Library name	Cell Name	Properties/Comments
gpdk180	nmos	Model Name = nmos1; W=2u; L= 180n
gpdk180	pmos	Model Name = pmos1; W=2u; L= 180n

If you place a component with the wrong parameter values, use the **Edit— Properties— Objects** command to change the parameters along with selection of commponent. 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.

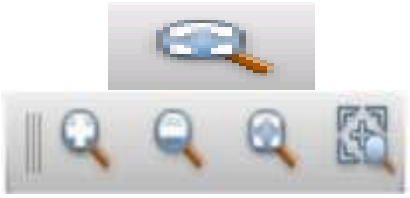
You can also execute **Create — Pin** or press **p**. The Add pin form appears.

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

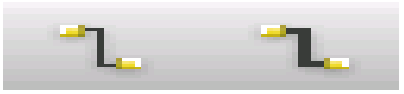
Pin Name	Direction
vin, vdd, vss	input
vout	output

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.

2. Observe the CIW output area for any errors.

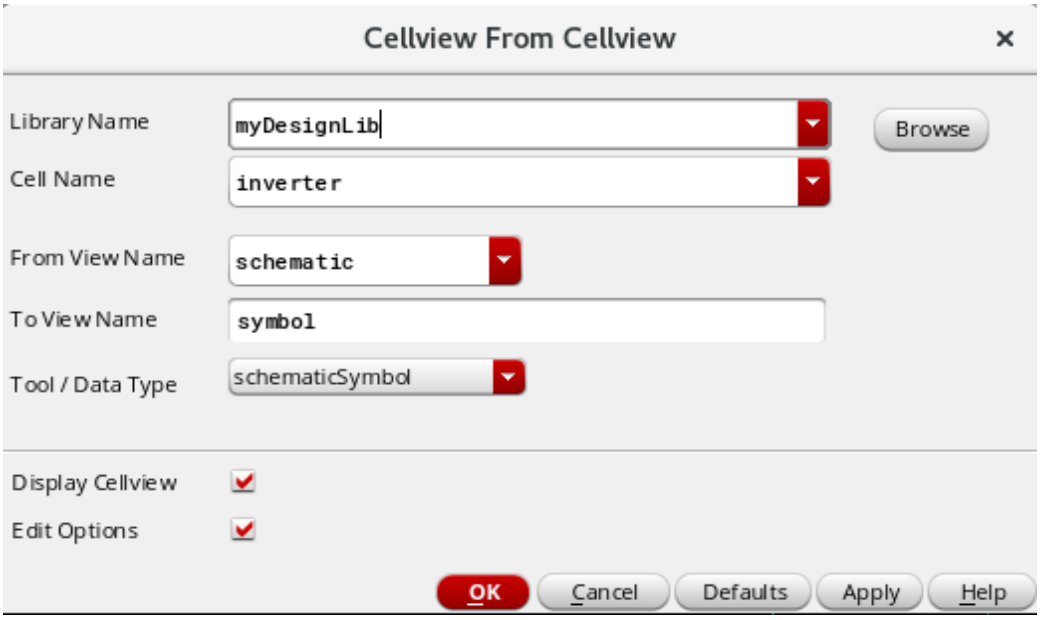
Symbol Creation:

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.

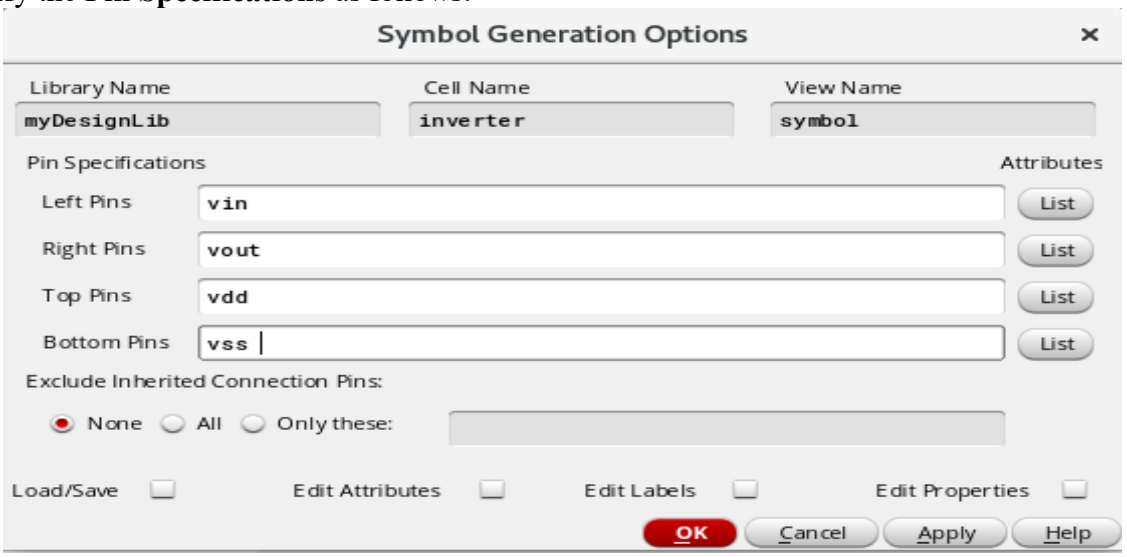
1. In the Inverter schematic window, execute, **Create — Cellview— From Cellview**. The **Cellview From Cellview** form appears. With the Edit Options function active, you can control the appearance of the symbol to generate.

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**



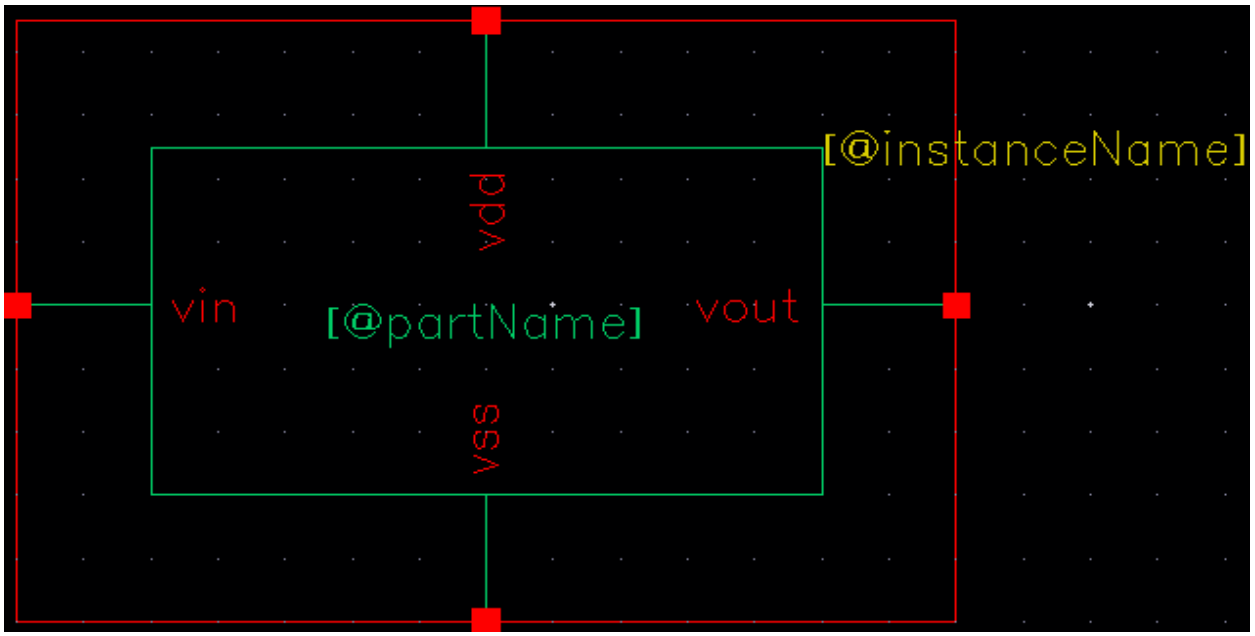
3. Click OK in the **Cellview From Cellview** form. The Symbol Generation Form appears.

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



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:

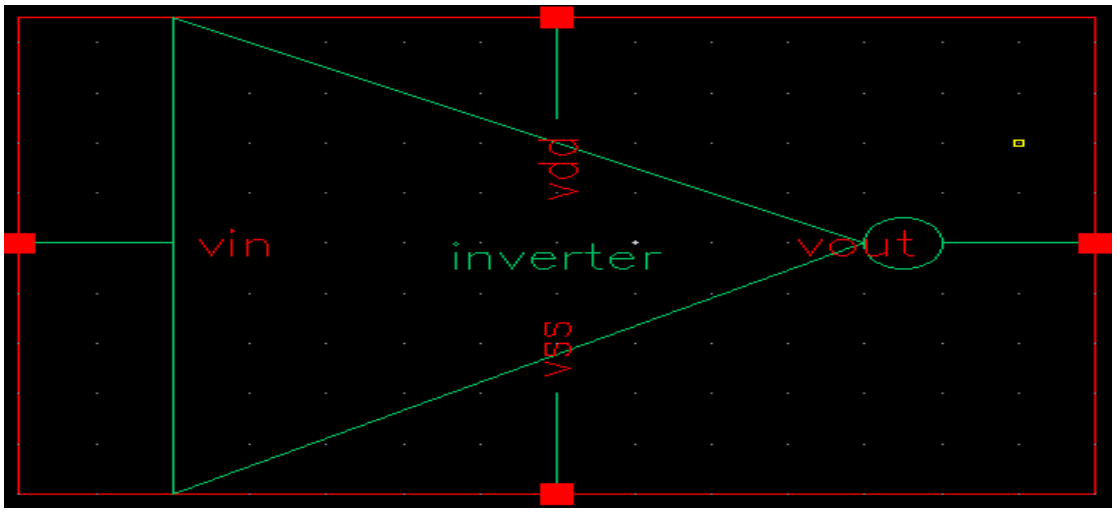
In this section we will modify the inverter symbol to look like a Inverter gate symbol.

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.

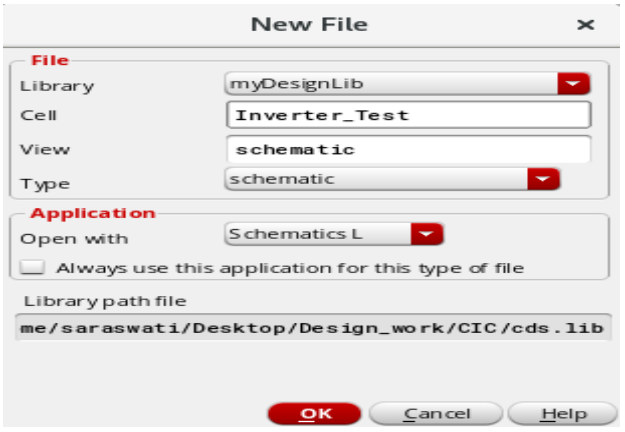


Building the Inverter_Test Design

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.

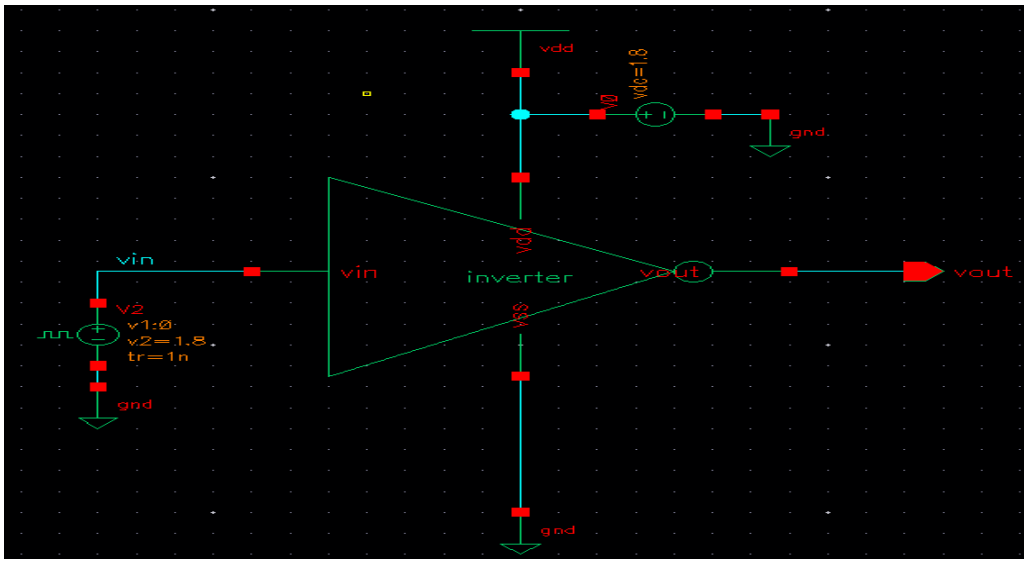

Tip: You can also press the w key, or execute **Create— Wire (narrow)**

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:

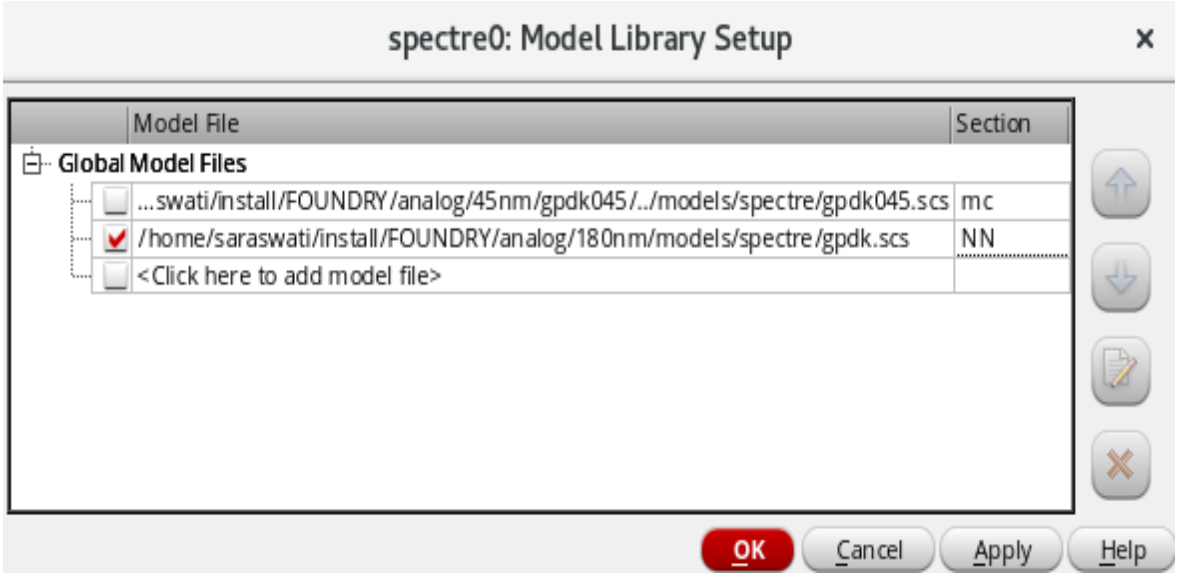
Start the Simulation Environment to run a simulation.

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 **gpdk.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 gpdk.scs file. 

Your Model Library Setup window should now looks 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

Analyses - Choose.

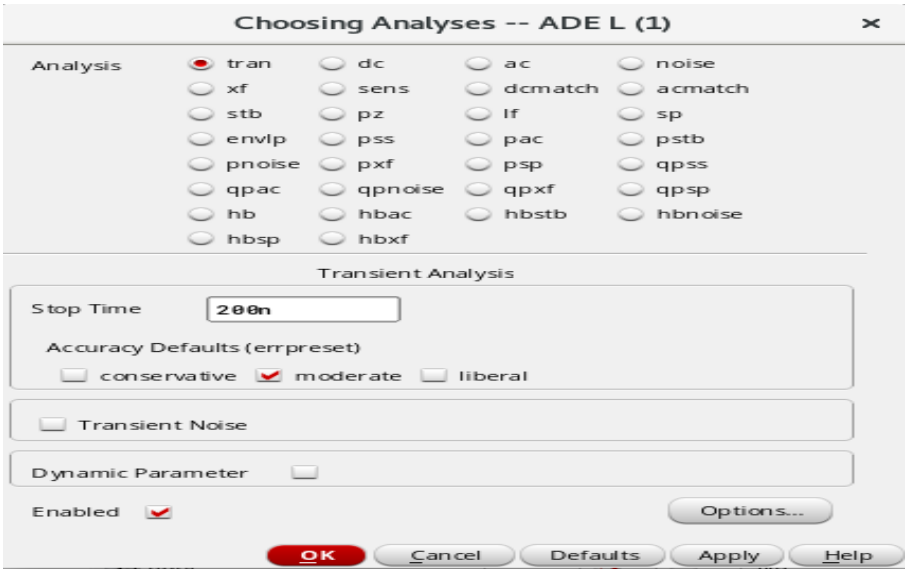
The Choosing Analysis form appears. This is a dynamic form, the bottom of the form changes based on the selection above.

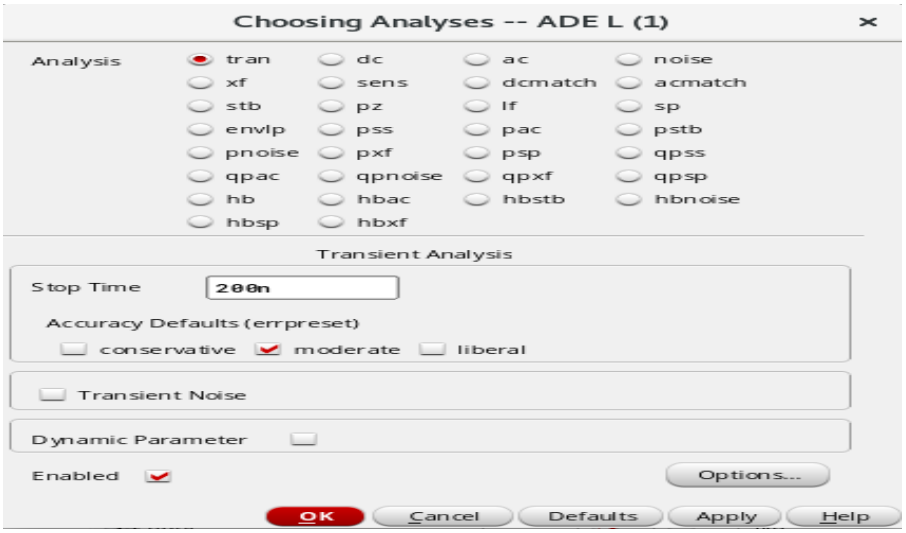
2. To setup for **transient analysis:**

a. In the Analysis section select **tran**

b. Set the stop time as **200n**

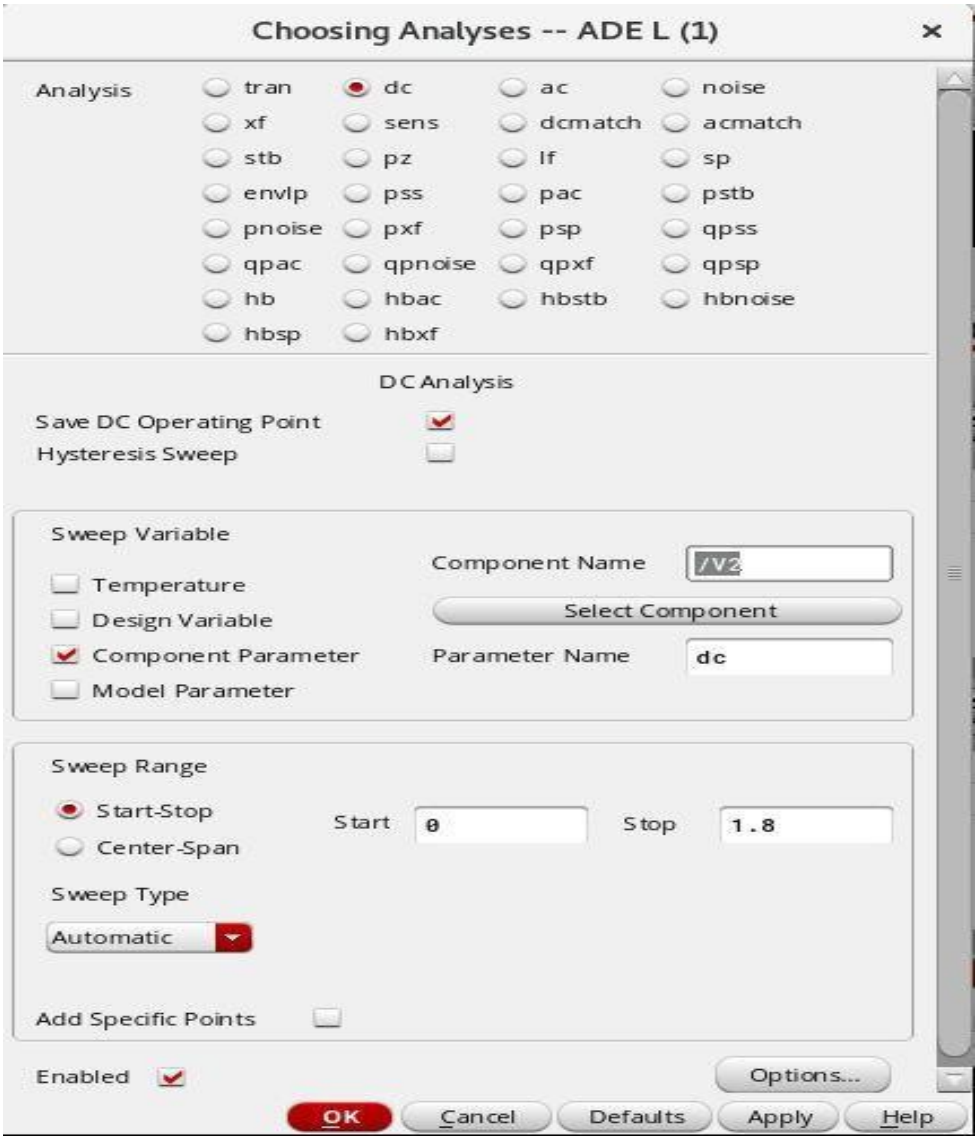
c. Click at the **moderate** or **Enabled** button at the bottom, and then click **Apply**





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**.



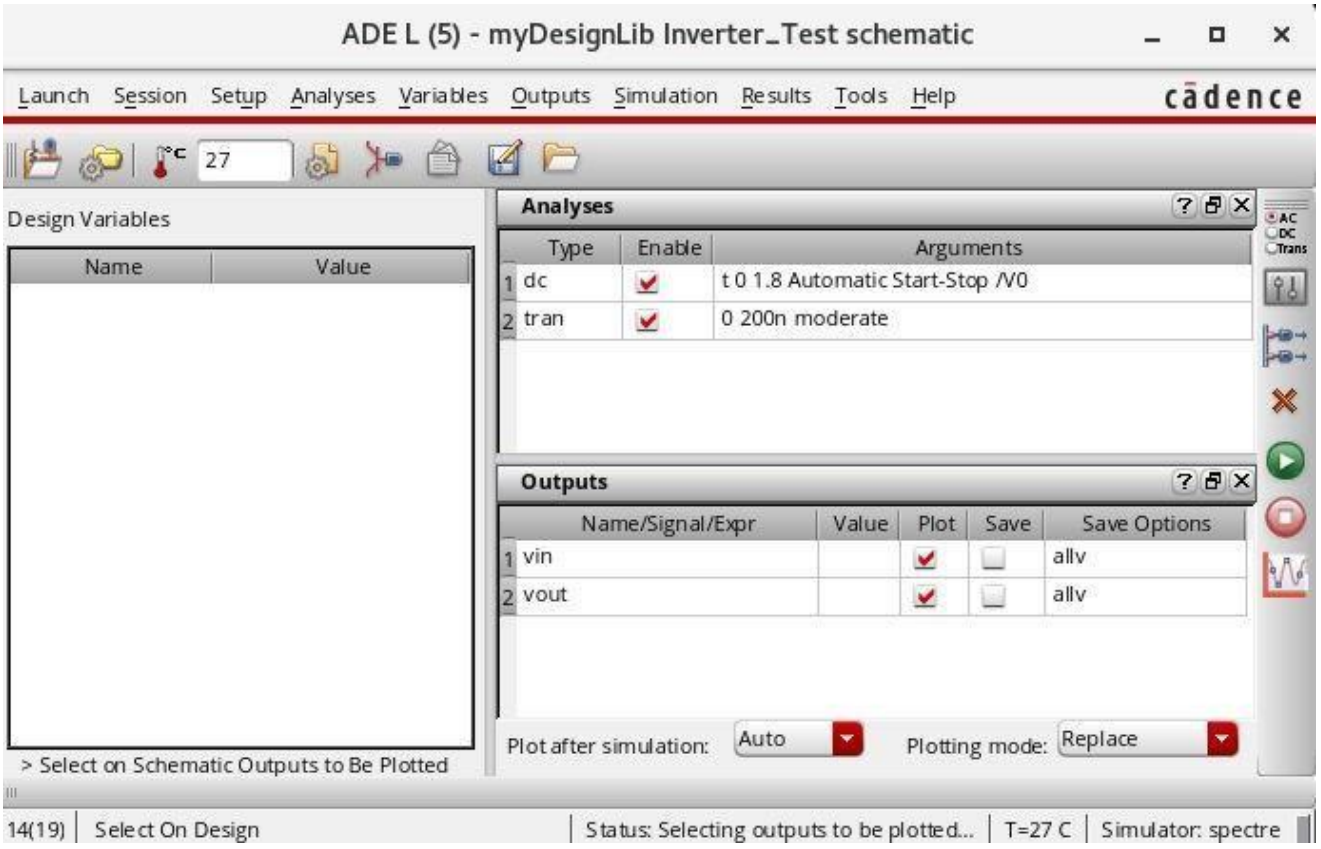
3. Click OK in the Choosing Analyses Form.

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.

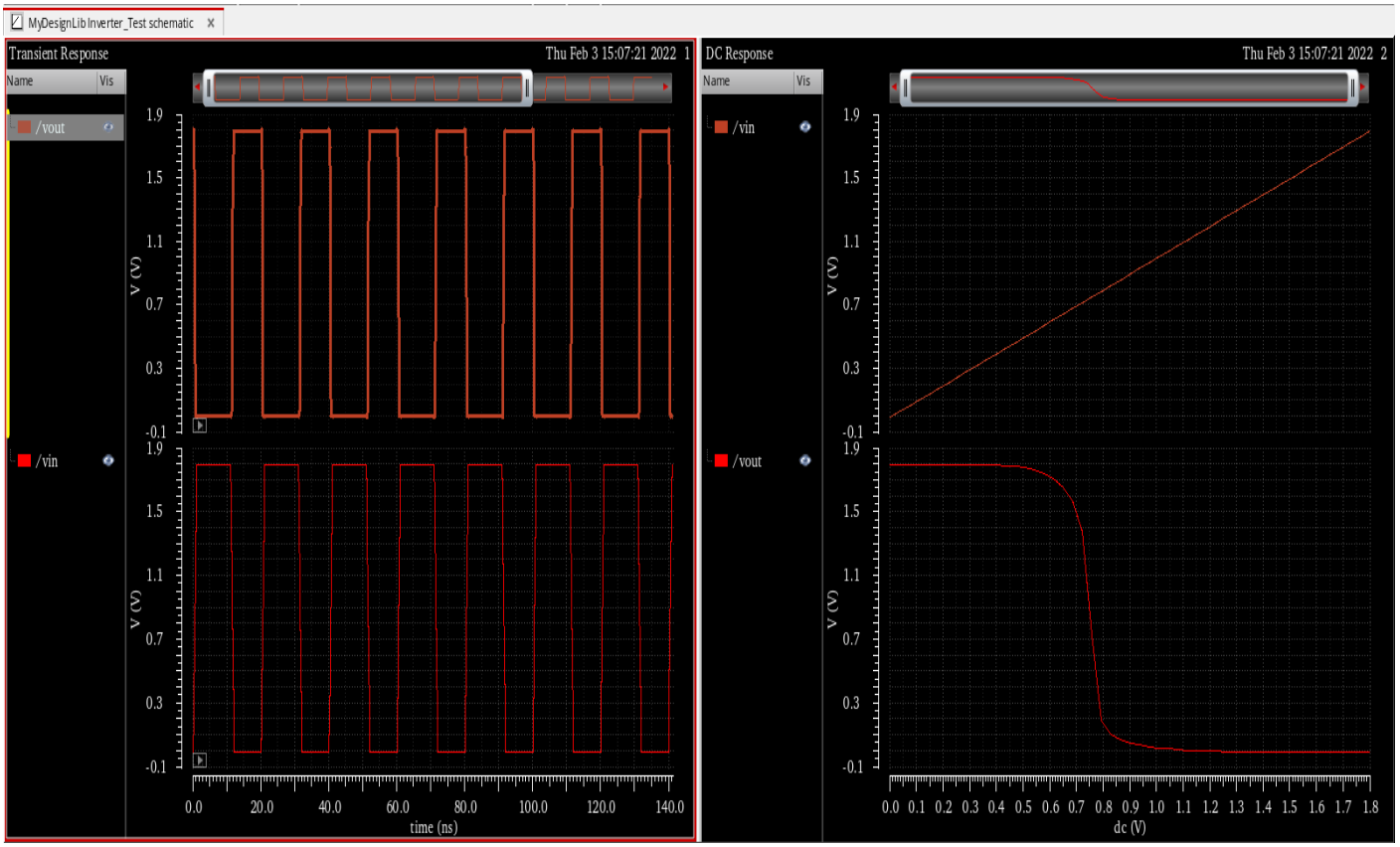
Does the simulation window look like this?



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.

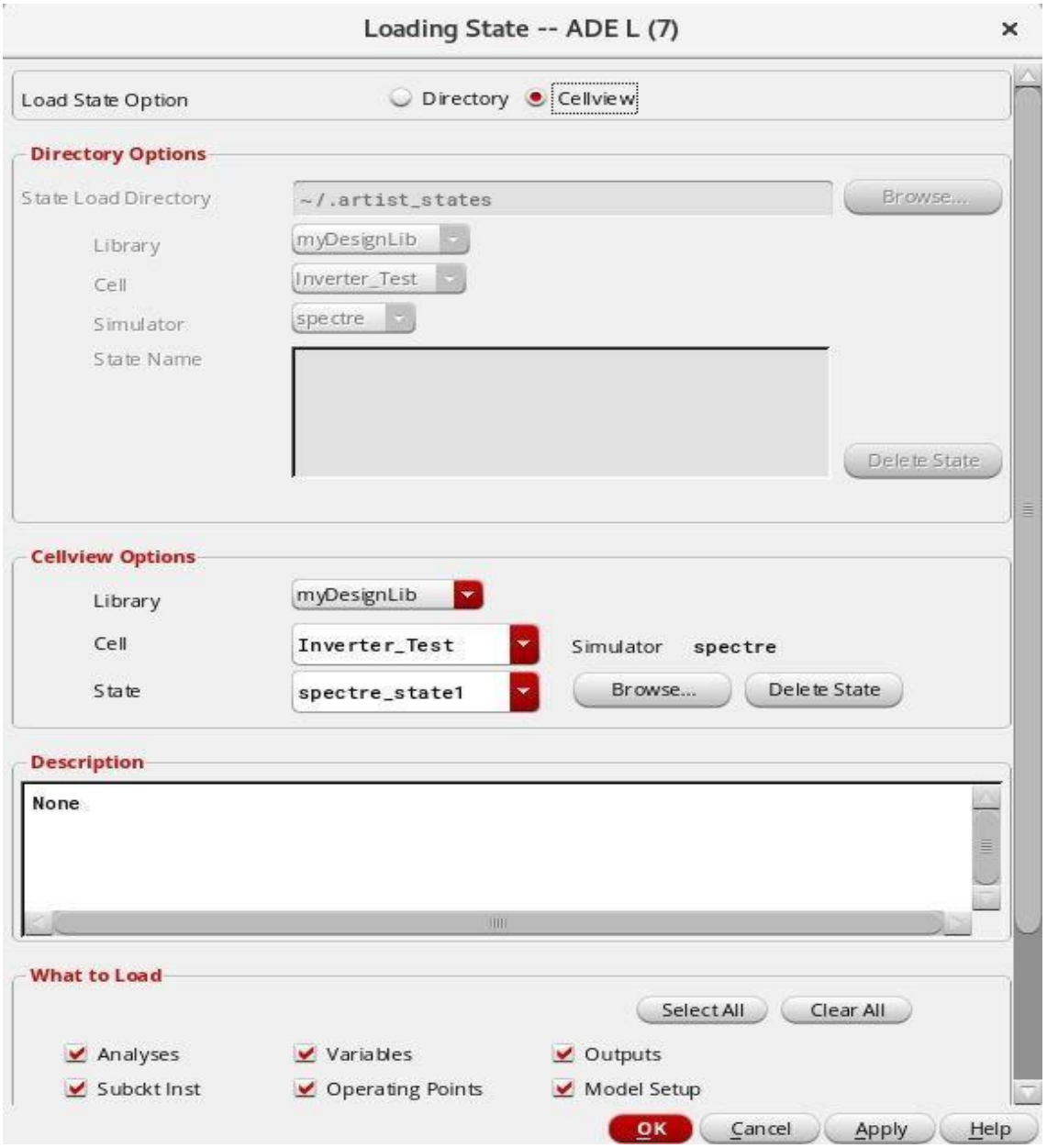
2. Set the **Save as field** to **state1_inv** and make sure all options are selected under what to save field.

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.



12. Design and Simulate Basic Common Source, Common Gate & Common Drain Amplifiers

Aim:

➔ Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.

➔ Analyze given below by performing Schematic Simulations

i. Input impedance

ii. Output impedance

iii. Gain

iv. Bandwidth

1. Basic Common Source Amplifier

Objective:

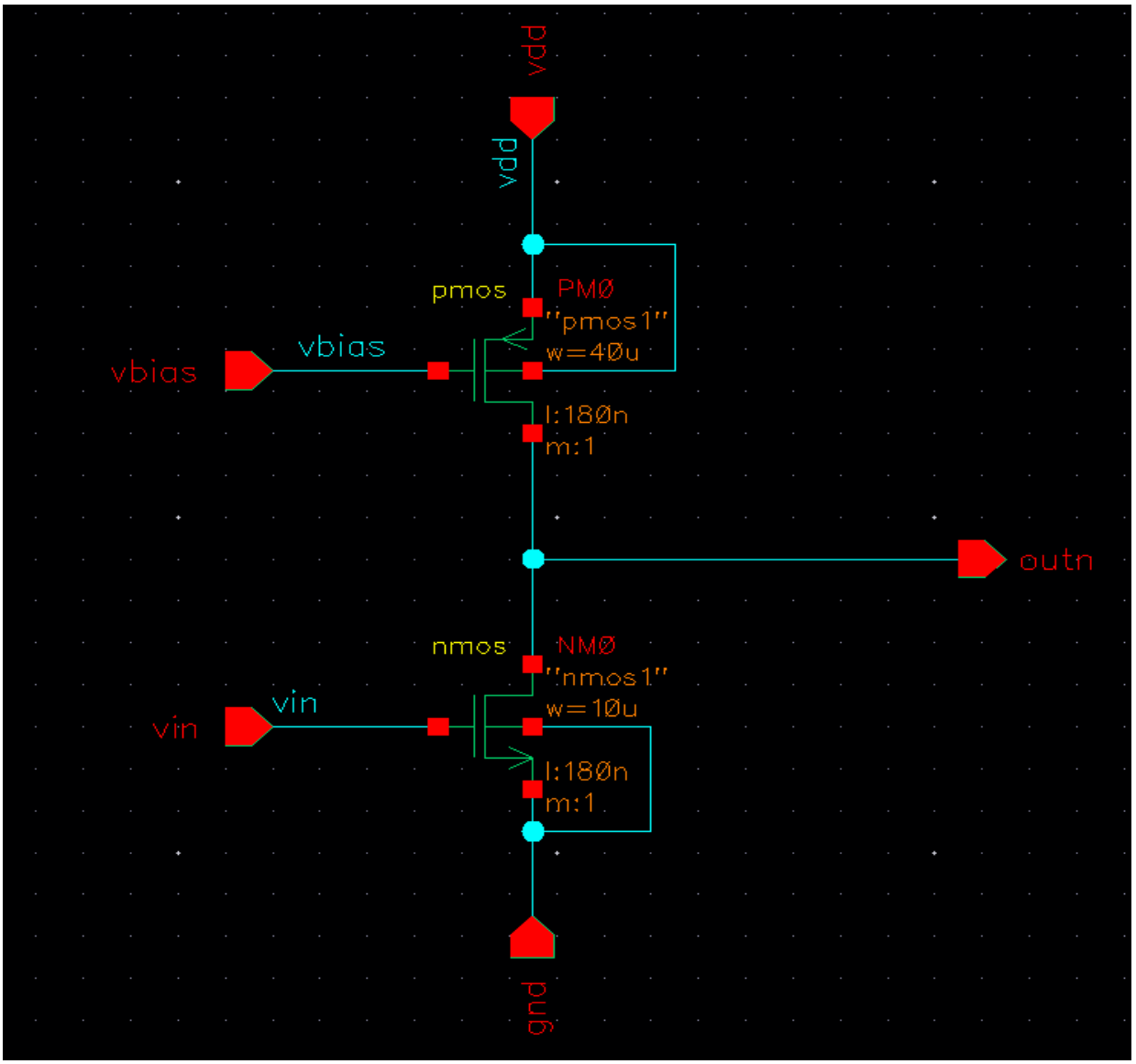
- To set up and run simulations on the CS Amplifier Test design.
- In this section, we will run the simulation for CS Amplifier and plot the Transient, DC and AC characteristics.
- Anayzing Gain, Bandwidth and Input and Output Impedance by simulating schematic test of cs amplifier

Theory:

In electronics, a common-source amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a voltage or transconductance amplifier. The easiest way to tell if a FET is common source, common drain, or common gate is to examine where the signal enters and leaves. The remaining terminal is what is known as "common". In this example,the signal enters the gate, and exits the drain. The only terminal remaining is the source. This is a common source FET circuit. The analogous bipolar junction transistor circuit is the common-emitteramplifier.

The common-source (CS) amplifier may be viewed as a transconductance amplifier or as a voltage amplifier. (See classification of amplifiers). As a transconductance amplifier, the input voltage is seen as modulating the current going to the load. As a voltage amplifier, input voltage modulates the amount of current flowing through the FET, changing the voltage across the output resistance according to Ohm's law. However, the FET device's output resistance typically is not high enough for a reasonable transconductance amplifier (ideally infinite), nor low enough for a decent voltage amplifier (ideally zero). Another major drawback is the amplifier's limited high-frequency response.

Schematic Capture:

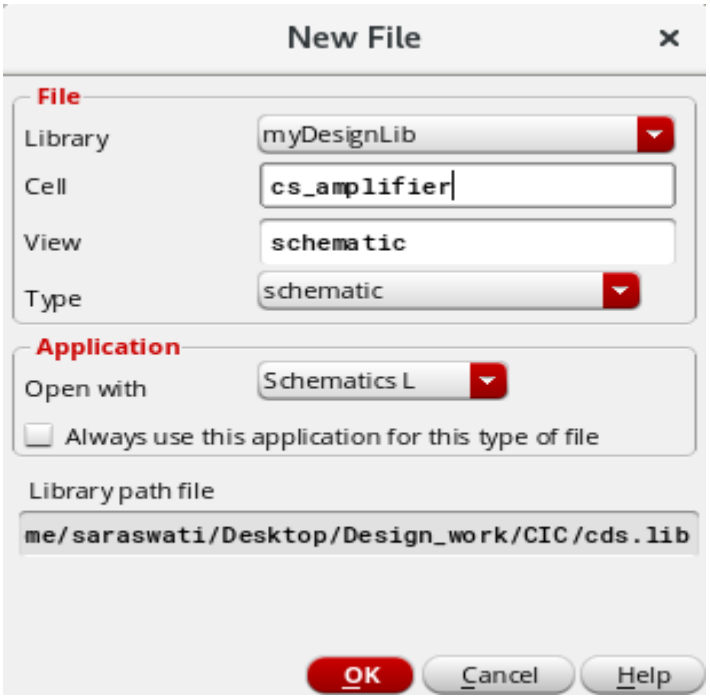


Schematic Entry

Creating a Schematic Cellview:

In this section we will learn how to open new schematic window in the new "**myDesignLib**" library and build the CS Amplifier 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 form.

3. Click OK when done the above settings. A blank schematic window for the CS Amplifier design appears

Adding Components to schematic:

1. In the CS Amplifier 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.

Library Name	Cell Name	Properties/Comments
gpdk 180	pmos	Model Name = pmos1; W=40u; L=180n
gpdk 180	nmos	Model Name = pmos1; W=40u; L=180n

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

If you place a component with the wrong parameter values, use the **Edit— Properties— Objects** command to change the parameters.

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. 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, vbias, vdd, gnd	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 **Window— Fit** or 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.

Saving the Design:

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

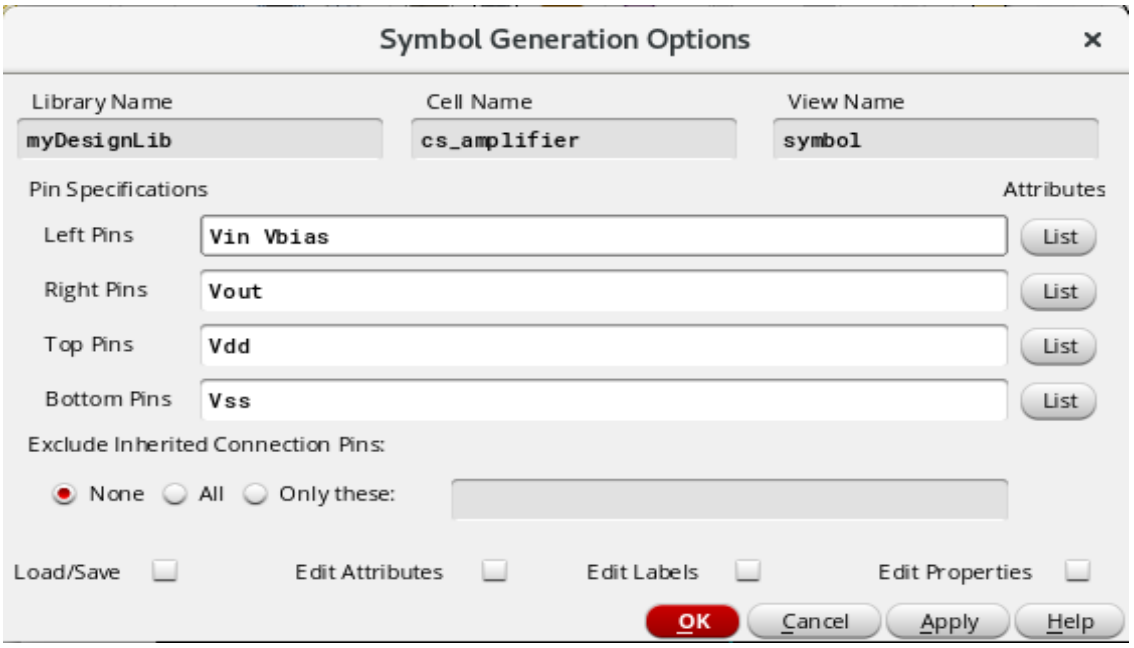
Symbol Creation:

Objective: To create a symbol for the Common Source Amplifier

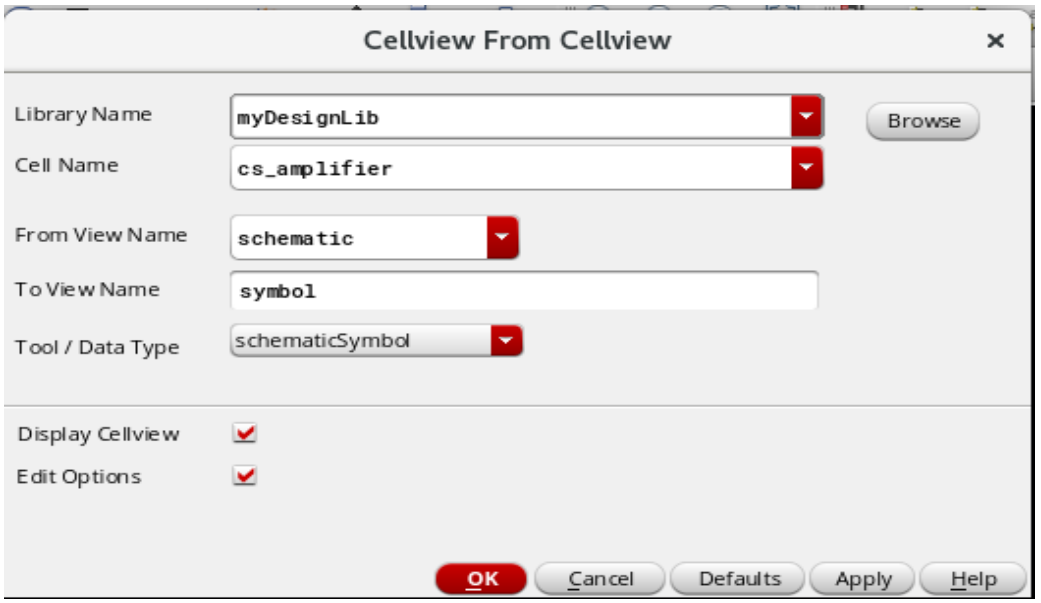
In this section, you will create a symbol for your cs_amplfier design, so you can place it in a test circuit for simulation. A symbol view is extremely important step in the design process.

1. In the CS 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.

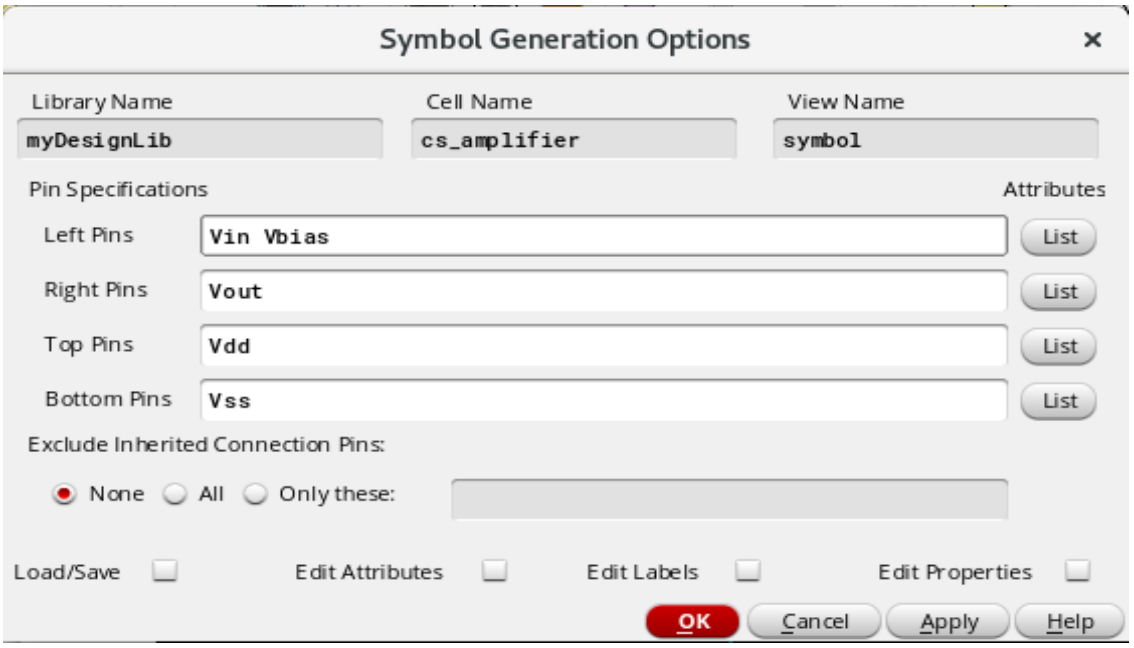
2. Verify that the **From View Name** field is set to **schematic**, and the To View Name field is set to **symbol**, with the Tool/Data Type set as **SchematicSymbol**
3. Click **OK** in the **Cellview From Cellview** form. The Symbol Generation Form appears.
4. Modify the Pin Specifications as follows:



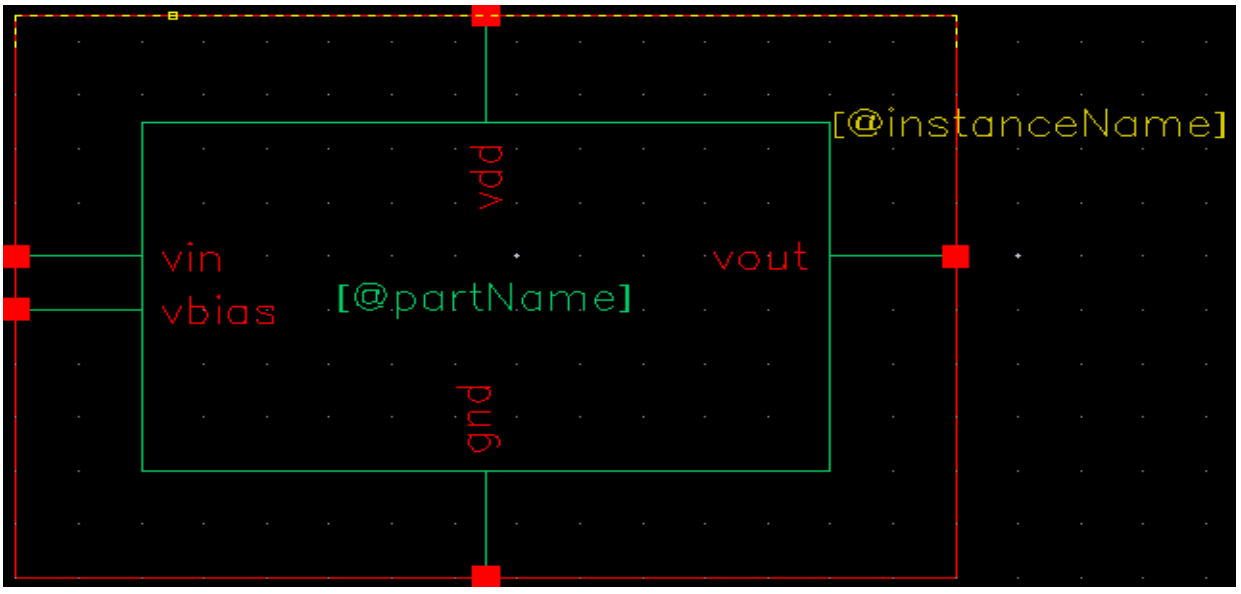
5. Click **OK** in the Symbol Generation Options form.
6. A new window displays an automatically created CS Amplifier symbol as shown here.



7. Click **OK** in the **Cellview From Cellview** form. The Symbol Generation Form appears.
8. Modify the Pin Specifications as follows:



9. Click **OK** in the Symbol Generation Options form.
10. A new window displays an automatically created CS Amplifier symbol as shown here.



Editing a Symbol: In this section we will modify the CS Amplifier 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.

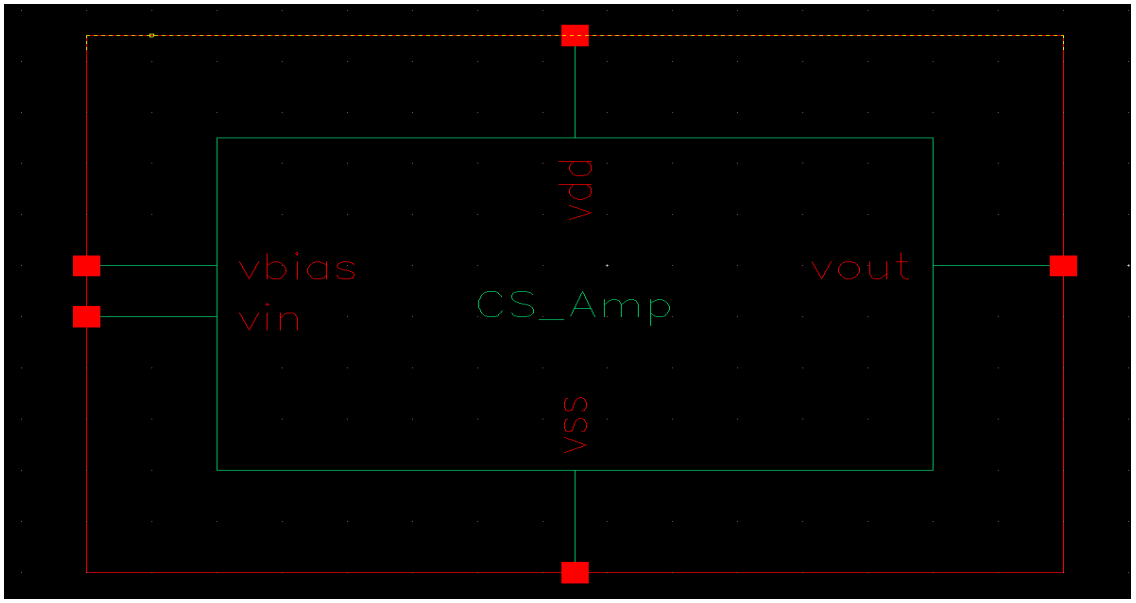
3. Execute **Create – Shape – polygon** and draw a shape similar to triangle.

4. After creating the rectangle press **ESC** key.

5. You can move the pin names according to the location.

6. Execute **Create — Selection Box**. In the Add Selection Box form, click **Automatic**. A new red selection box is automatically added.

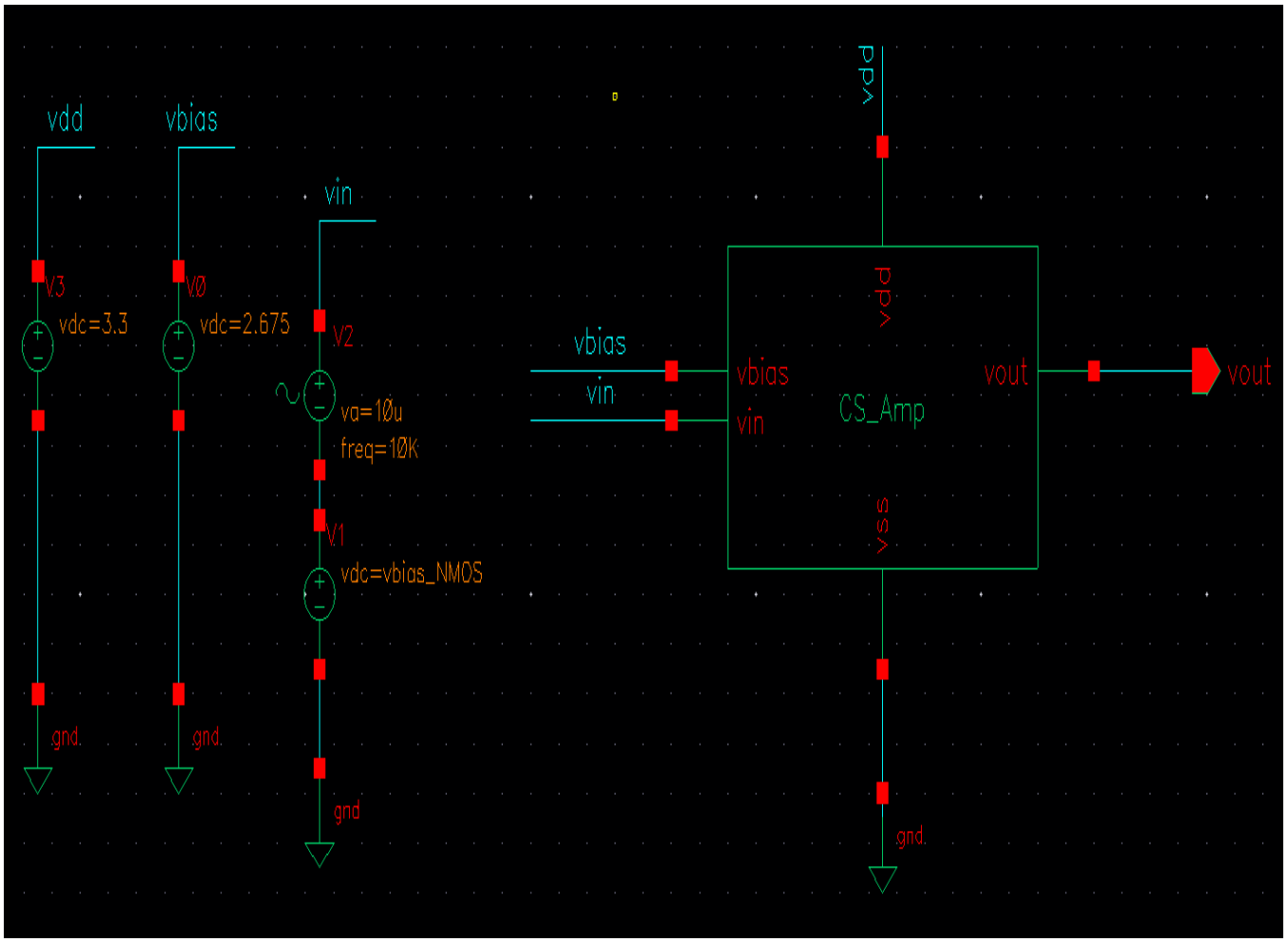
7. After creating symbol, click on the **save** icon in the symbol editor window to save the symbol. In the symbol editor, execute **File — Close** to close the symbol view window



Design entry for Test Schematic:

Library name	Cellview name	Properties
myDesignLib	cs_amplifier	Symbol
analogLib	vsin	Define pulse specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude=10u; Frequency=10K
analogLib	vdd, vbias & gnd	vdd=3.3v; vbias=2.675v: vbias_NMOS=662mv

Building the cs_amplifer_test Design



Analog Simulation with Spectre

Objective:

- To set up and run simulations on the CS Amplifier Test design.
- In this section, we will run the simulation for CS Amplifier and plot the Transient, DC and AC characteristics.
- Anayzing Gain, Bandwidth and Input and Output Impedance by simulating schematic test of cs amplifier

Starting the Simulation Environment:

1. In the cs amplifier_test schematic window, execute **Launch – ADE L.** The Analog Design Environment (ADE) simulation window appears.

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

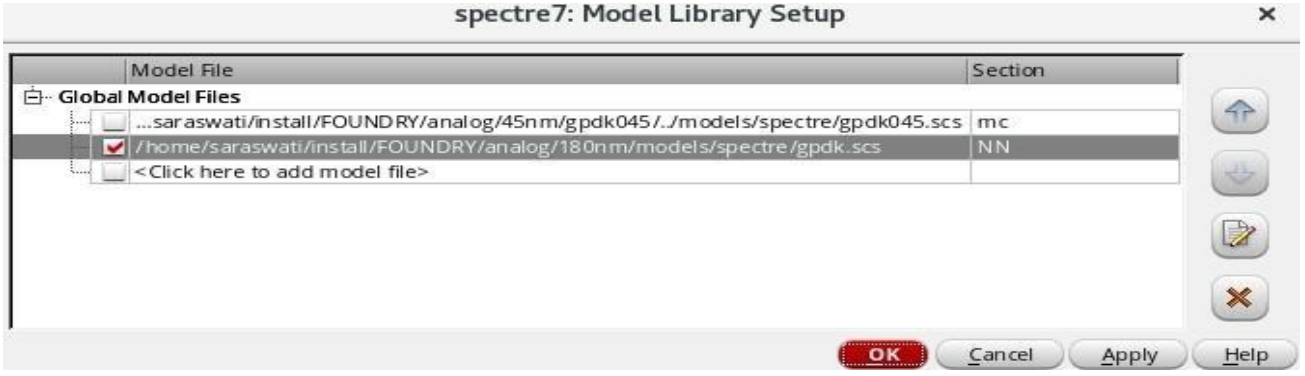
3. In the Choosing Simulator form, set the Simulator field to **spectre** (Not spectreS) and click **OK**.

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 gpdk.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 gpdk.scs 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 **Analyses – Choose** icon. 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 1**m**
   - 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 **0** to **3.3** 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 **1** to **100G**
   - **d.** Sweep type **Automatic.**
   - e. Check the enable button and then click **Apply**.

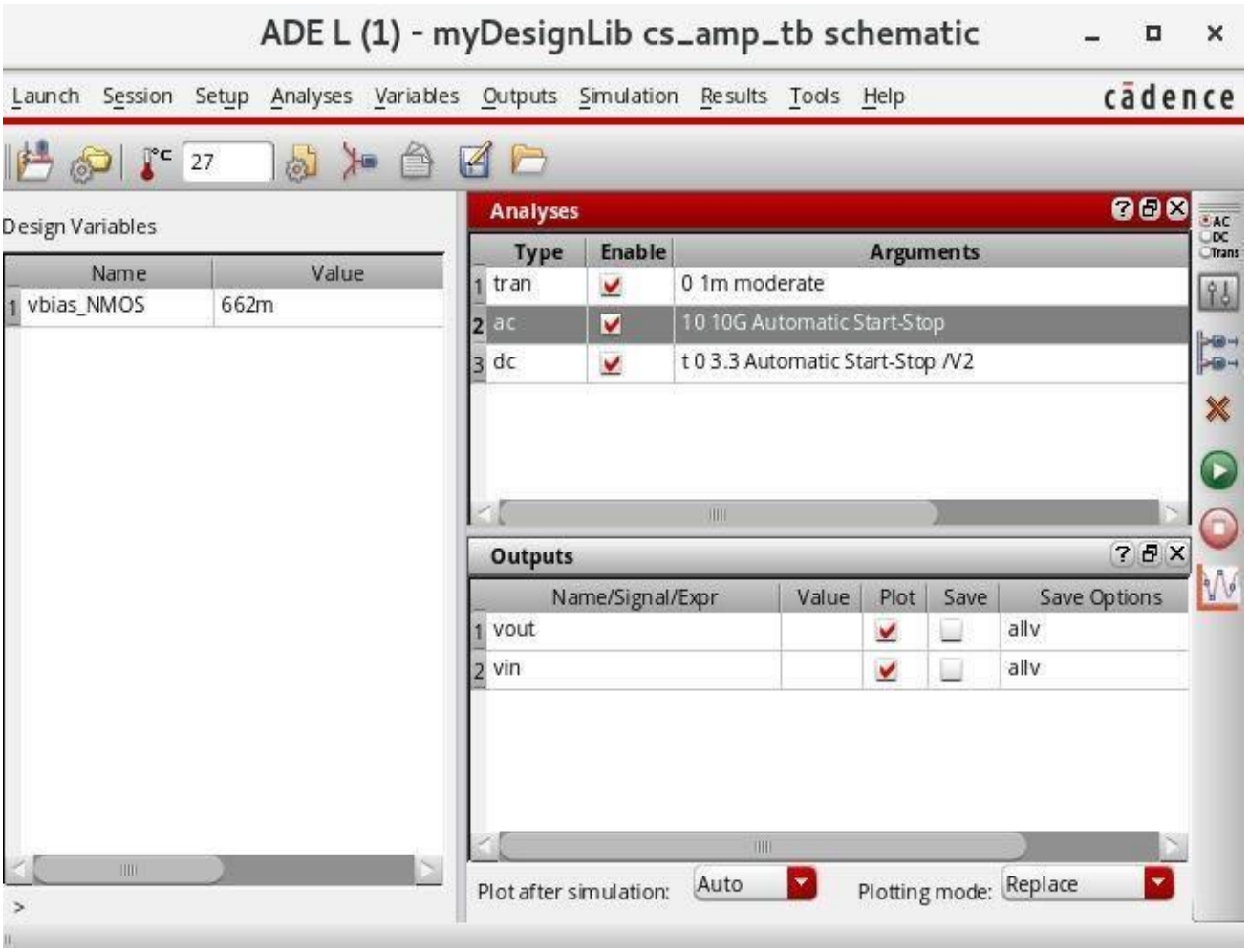
5. Click **OK** in the Choosing Analyses Form.

Selecting Outputs for Plotting:

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?

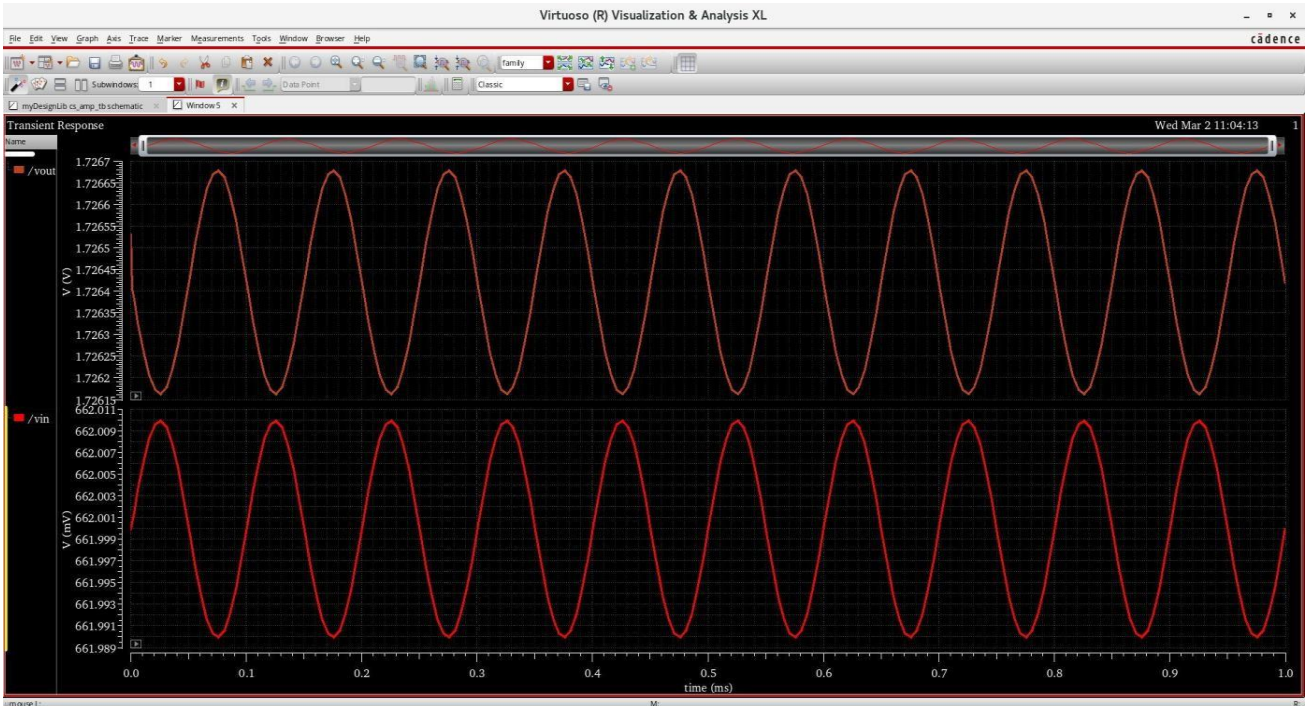


Running the Simulation:

1. Execute Simulation – Netlist and Run in the simulation window to start the simulation, this will create the netlist as well as run the simulation.

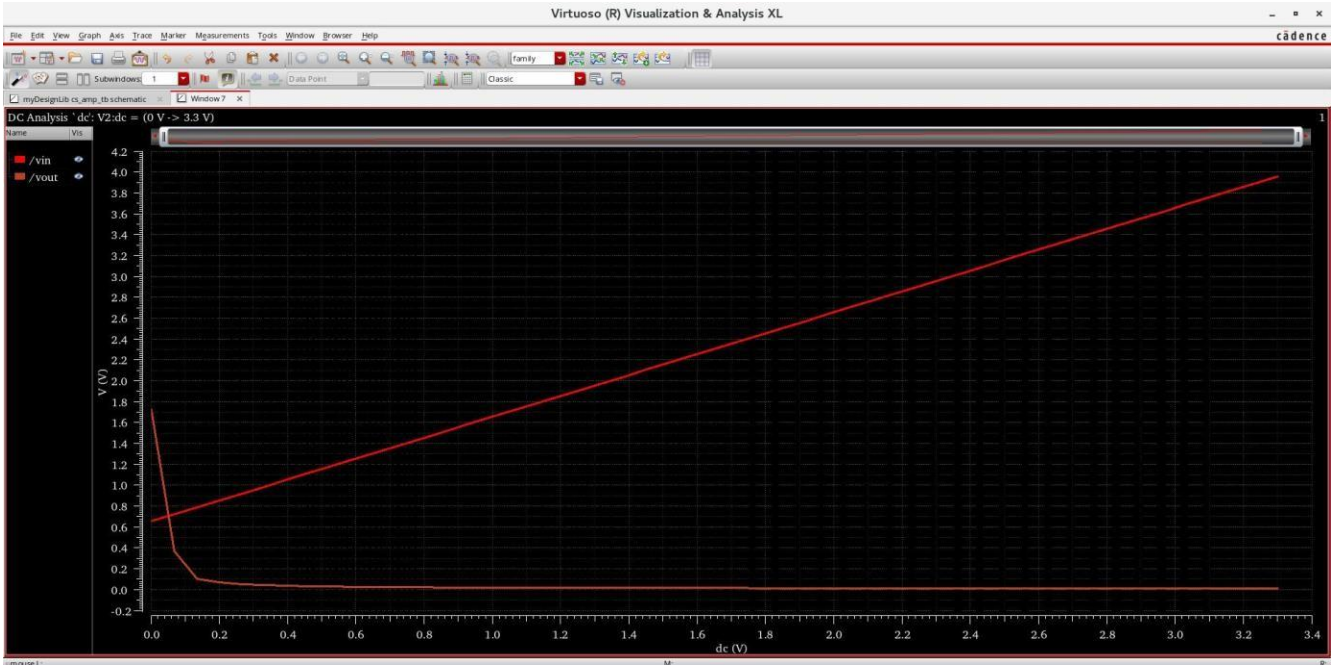
2. When simulation finishes, the Transient, DC and AC plots automatically will be popped up along with netlist.

Transient Response :



AC Responses:

DC Responses:

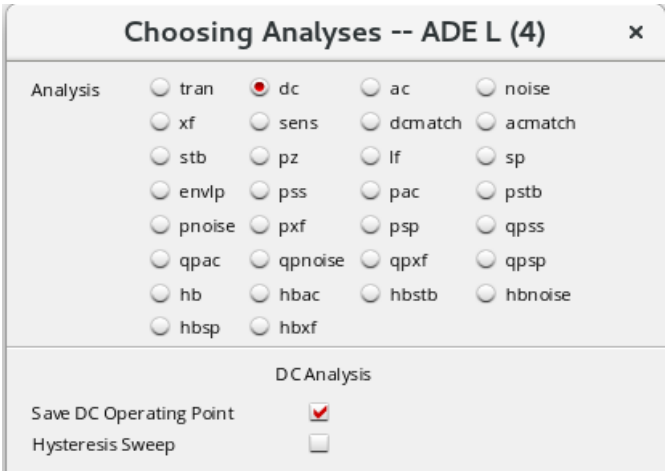


Operating Regions for Transistors:

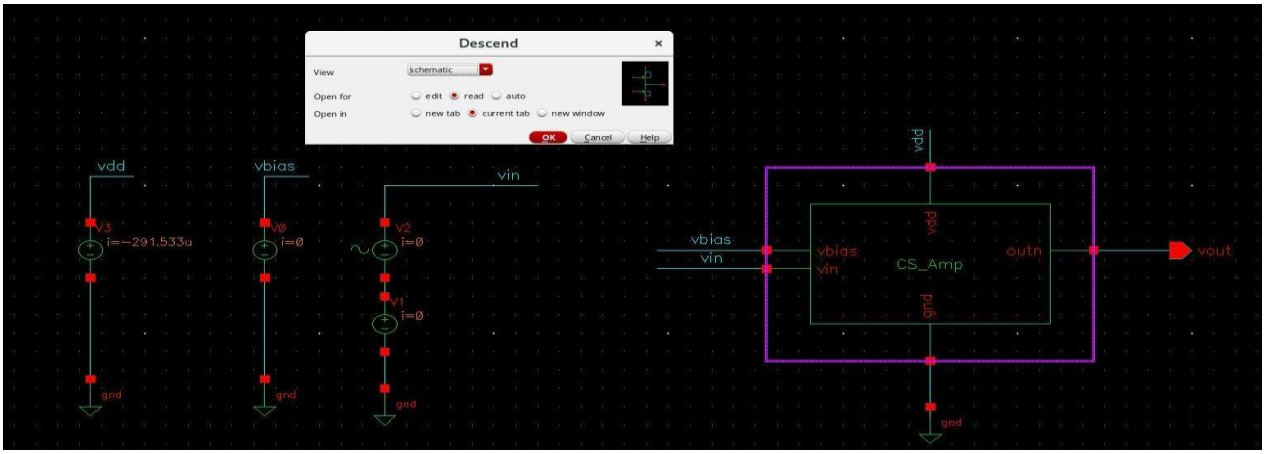
For amplification all the transistor should be in saturation region (region-2)

You can map the 5 regions as follow:Here you can map 5 regions that are **region-0**: cut-off, **region-1**: triode, **region-2** saturation, **region-3** subth and **region-4** breakdown.

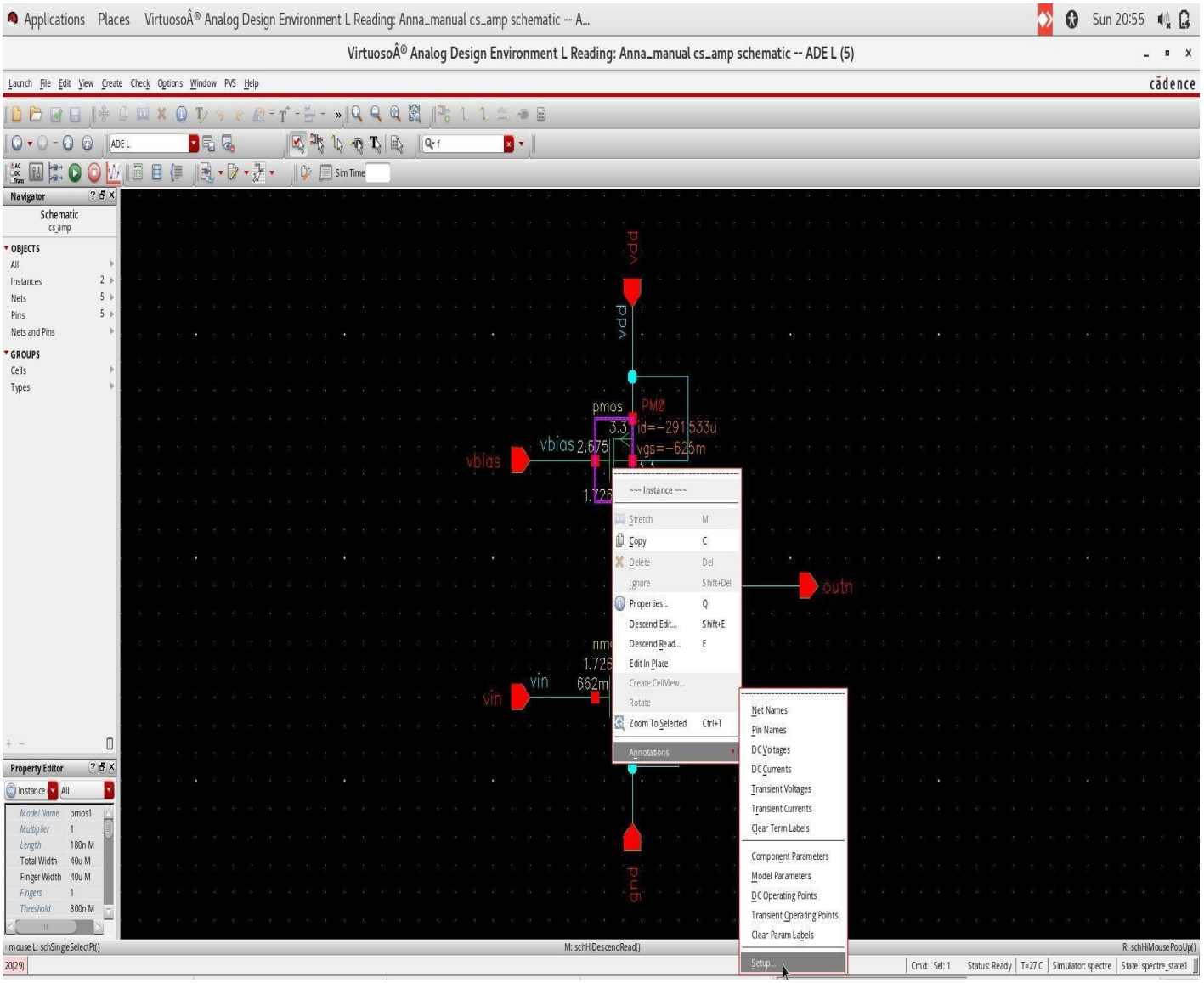
1. Do the DC analysis along with enabling save DC Operating Point



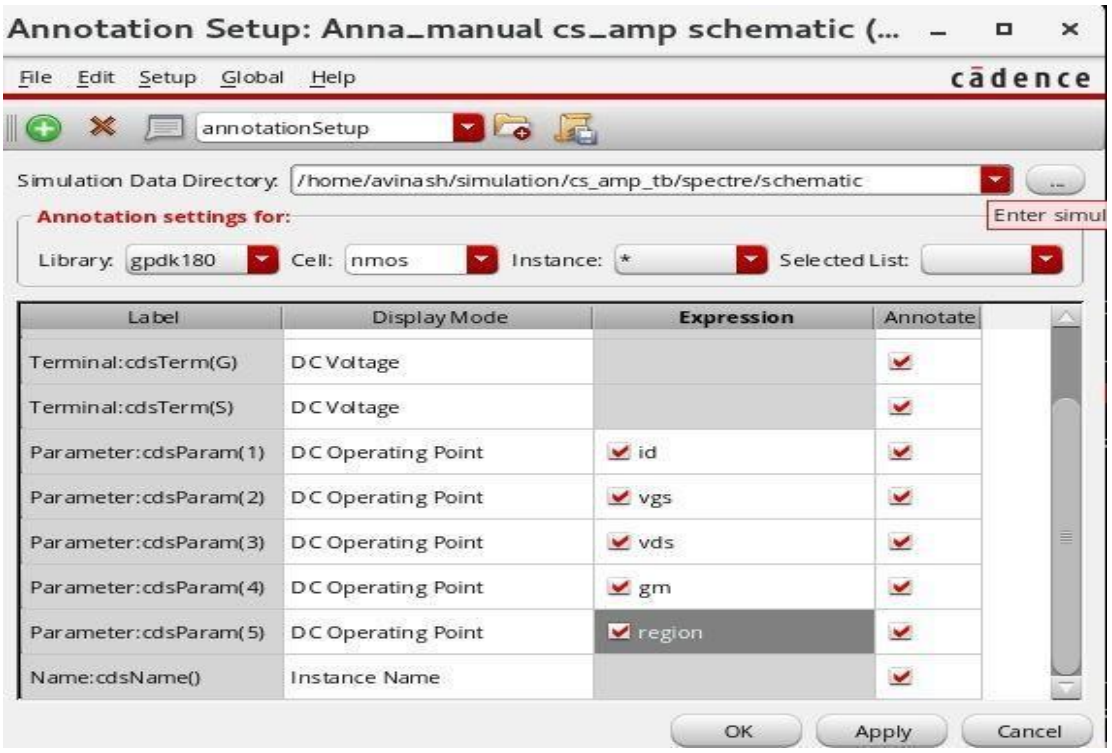
2. Select the symbol of **CS Amplifier** in schematic Test-bench and press "**E**" from key board and click on **OK**

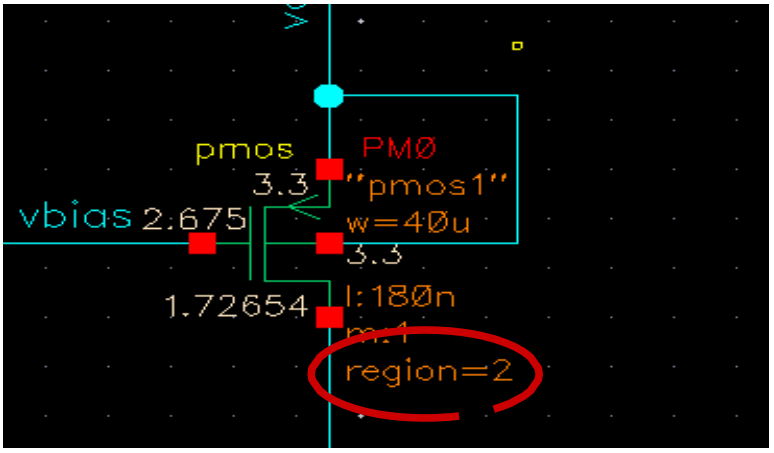


3. Select the transistor and '**Right Click**' and choose annotation→ '**Setup**'



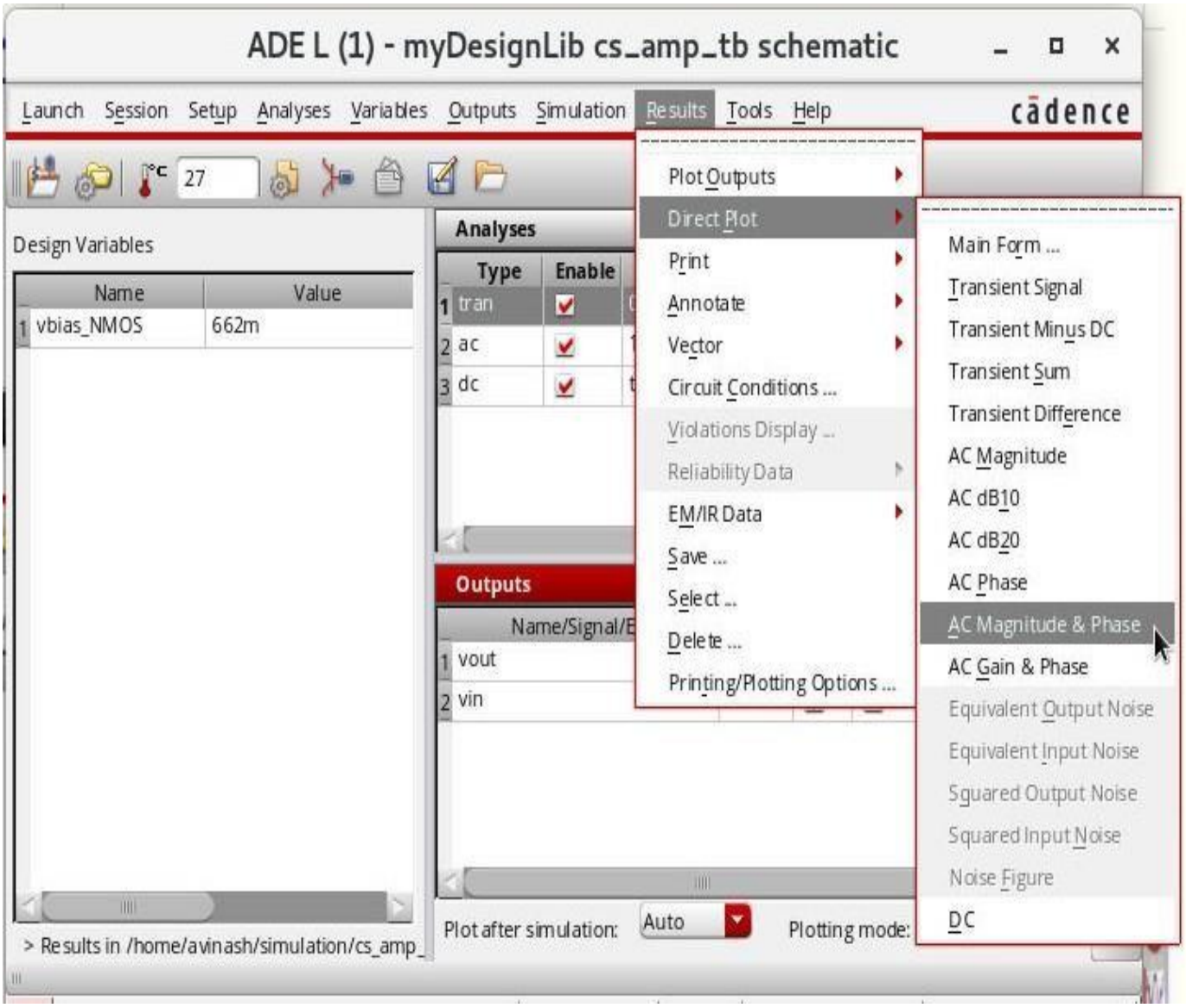
4. It will take you to annotation setup window. Under **Display mode** select the **DC Operating Point** and under **Expression** select **region** option and **Cell** select **nmos/pmos**. Click on **OK** to see the region of operations in schematic near the transistor.



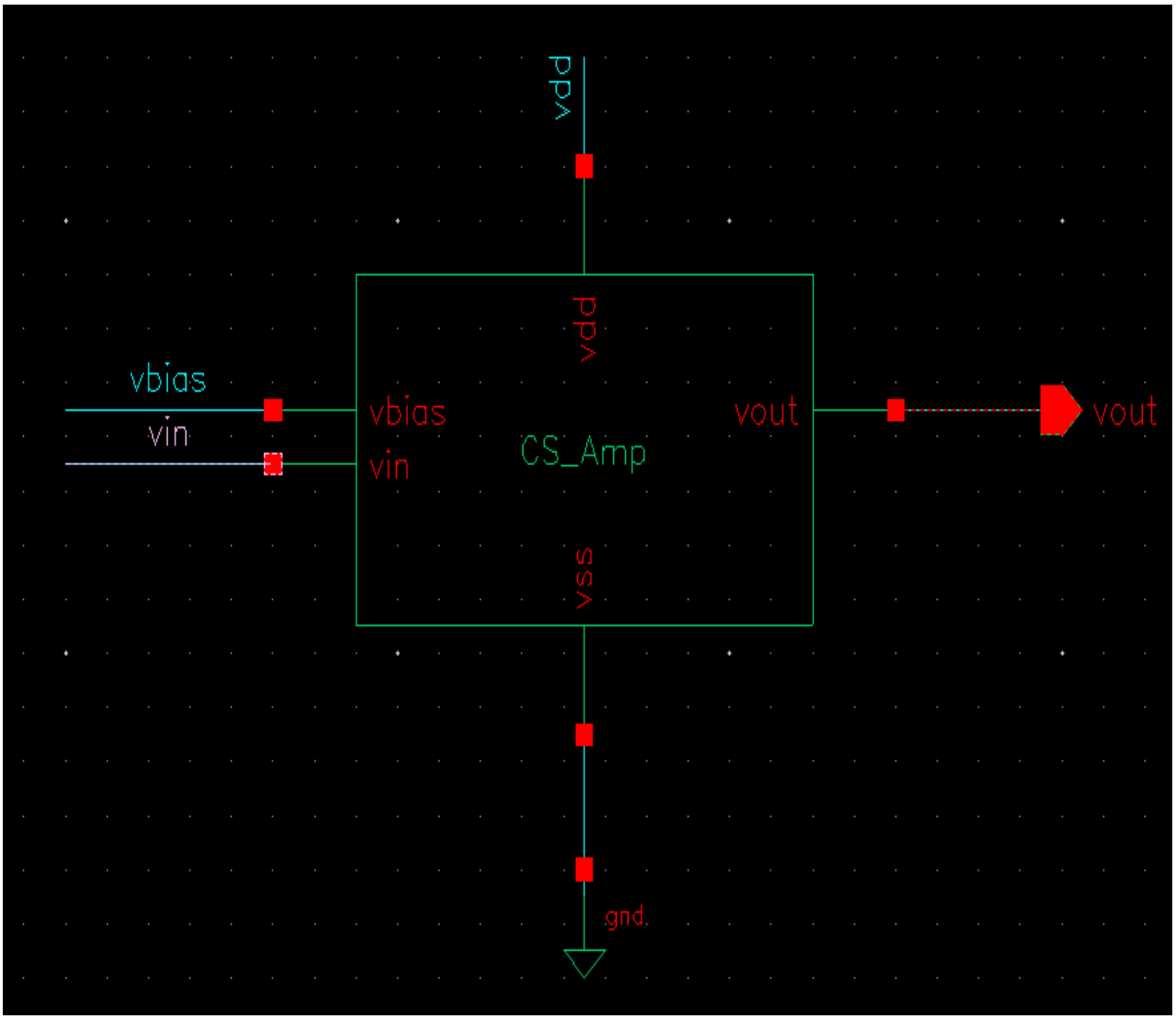


Gain Calculation

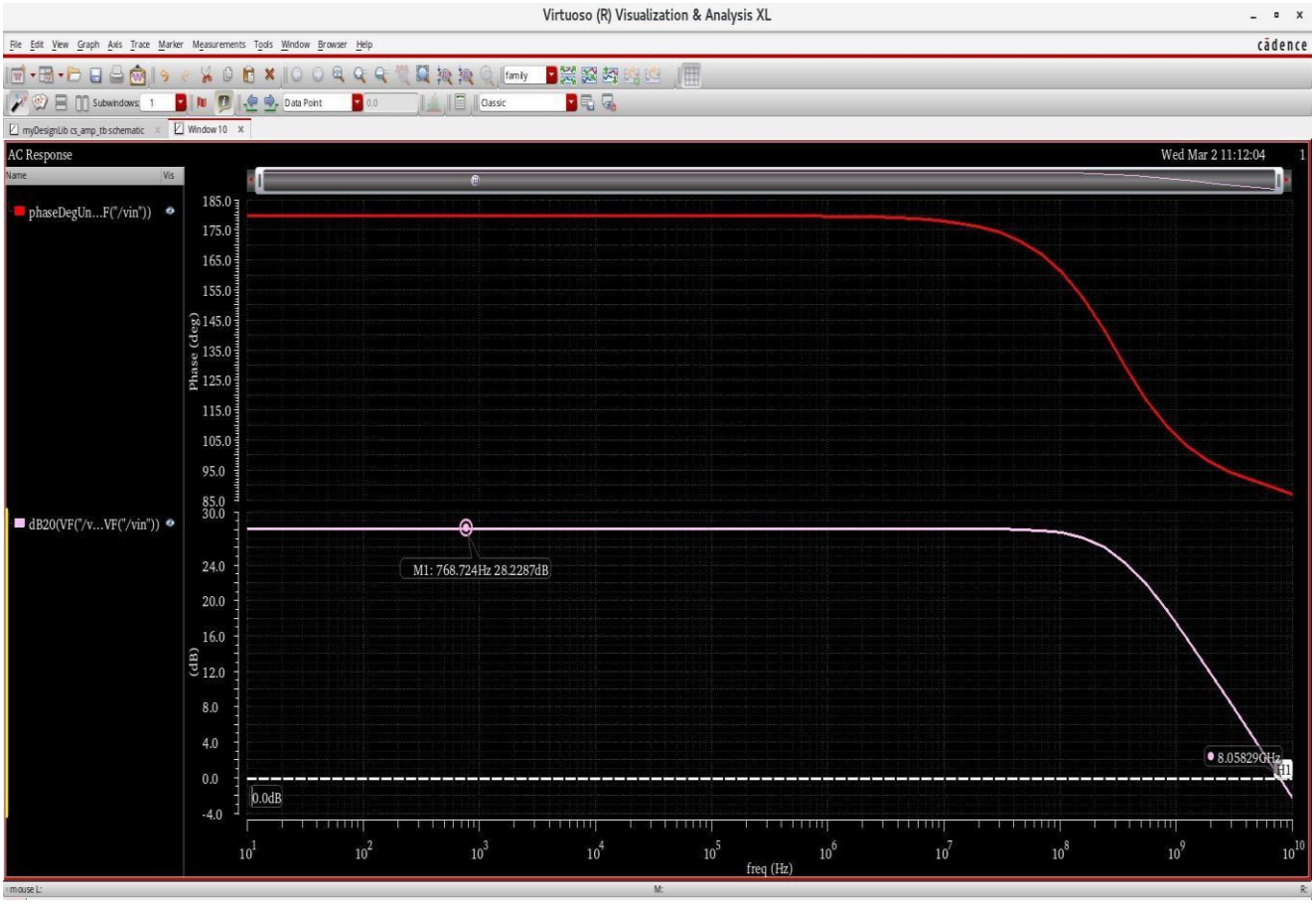
After running AC Analysis, Go back to **ADE L**
Select "Results →Direct Plot → AC Magnitude & Phase"



- It will take you back to the Test Schematic
- Select the **output net** and click on '**Esc**' on the keyboard



Gain Plot:

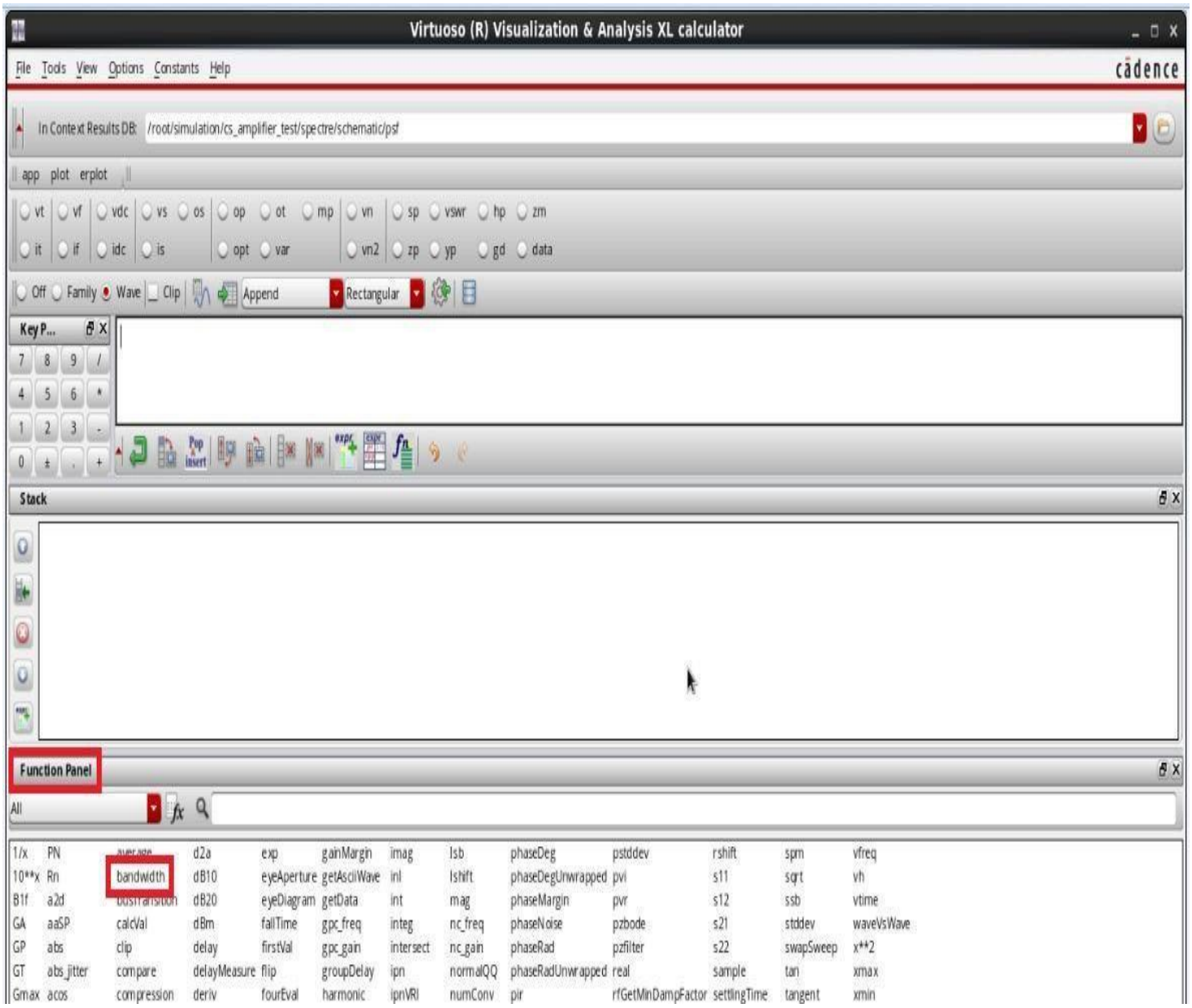


- Use the bind key '**M**' to place 'Marker' as in the above graph
- The value that can be seen is the '**Gain**' in dB

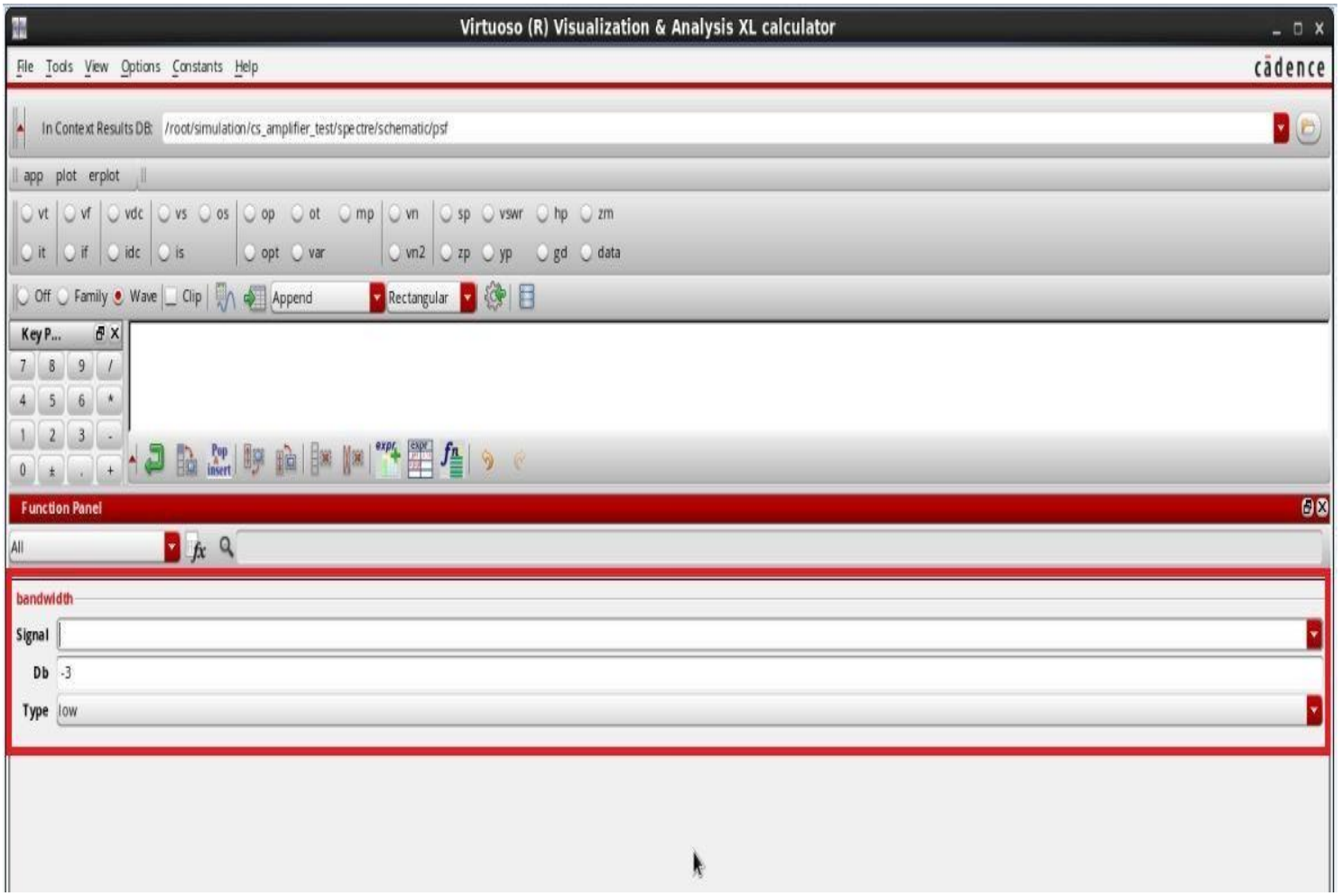
Bandwidth Calculation

From the Graph window,

- Select “Tools →Calculator”
- Select “**Bandwidth**” from the “**Function Panel**”



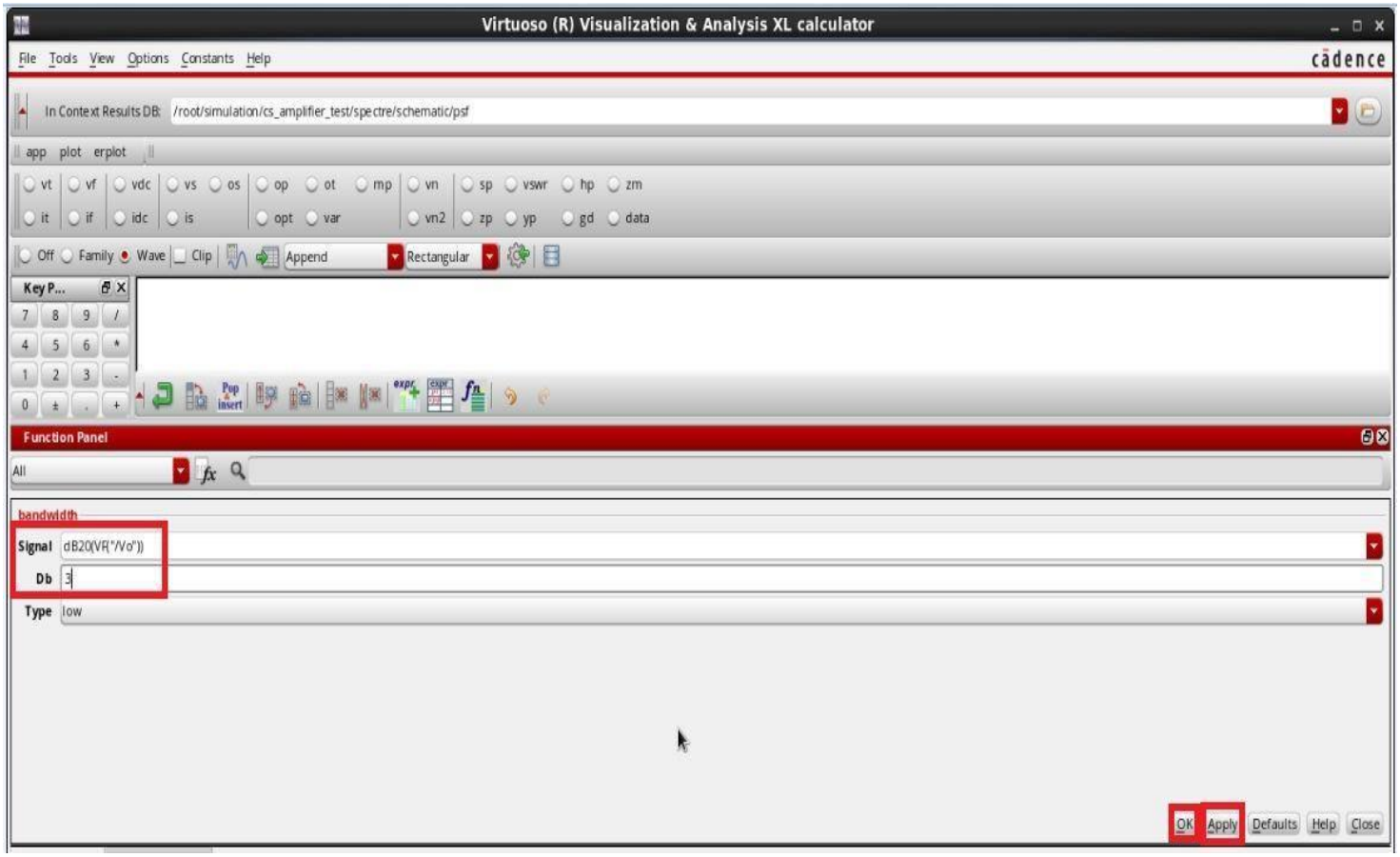
Bandwidth in Function Panel:



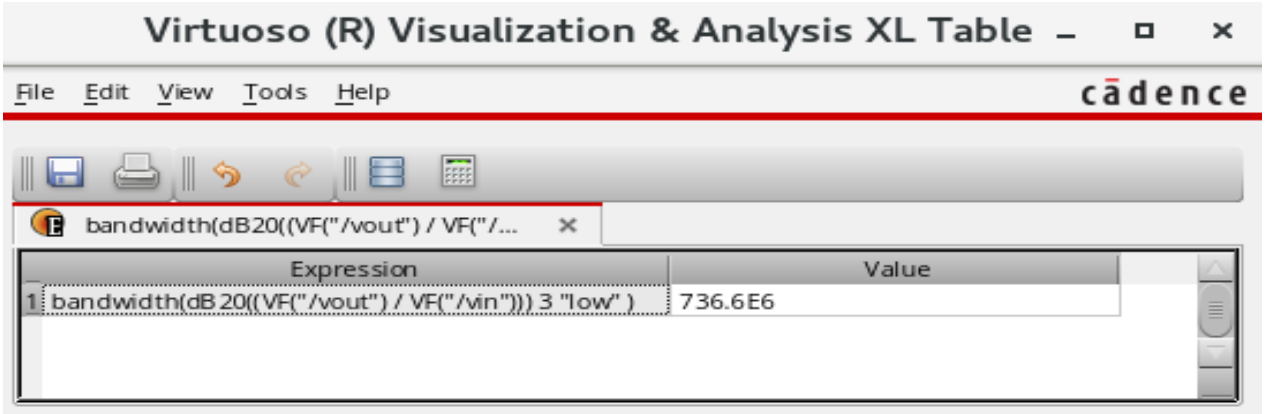
Bandwidth

- Place the cursor in '**Signal**'
- Go back to the waveform window
- Select the '**dB20**' curve
- Mention '**3**' as '**Db**' value to calculate '**3dB Bandwidth**'
- Click on '**Apply**' and then '**OK**'

Bandwidth:



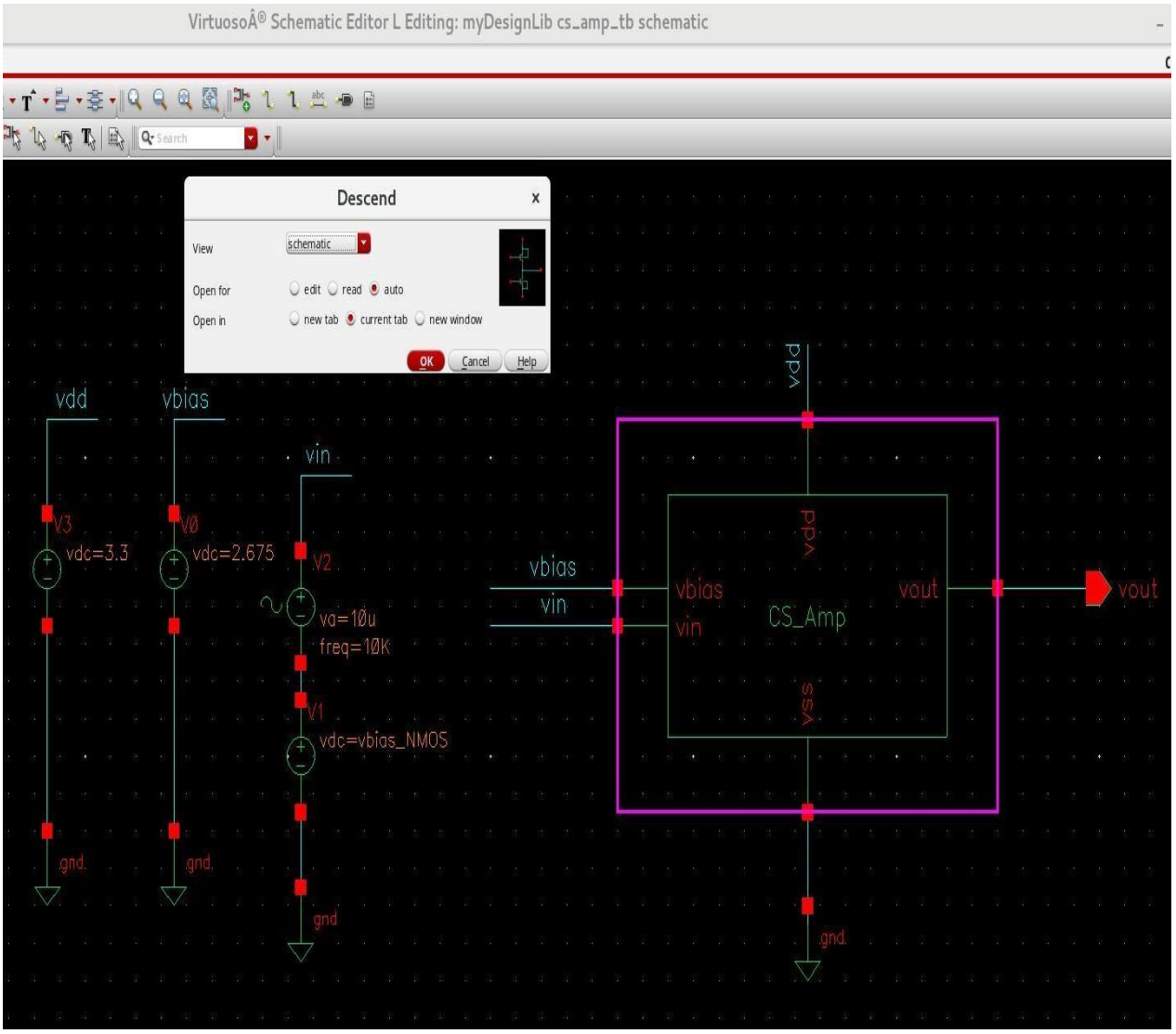
Evaluated Bandwidth:



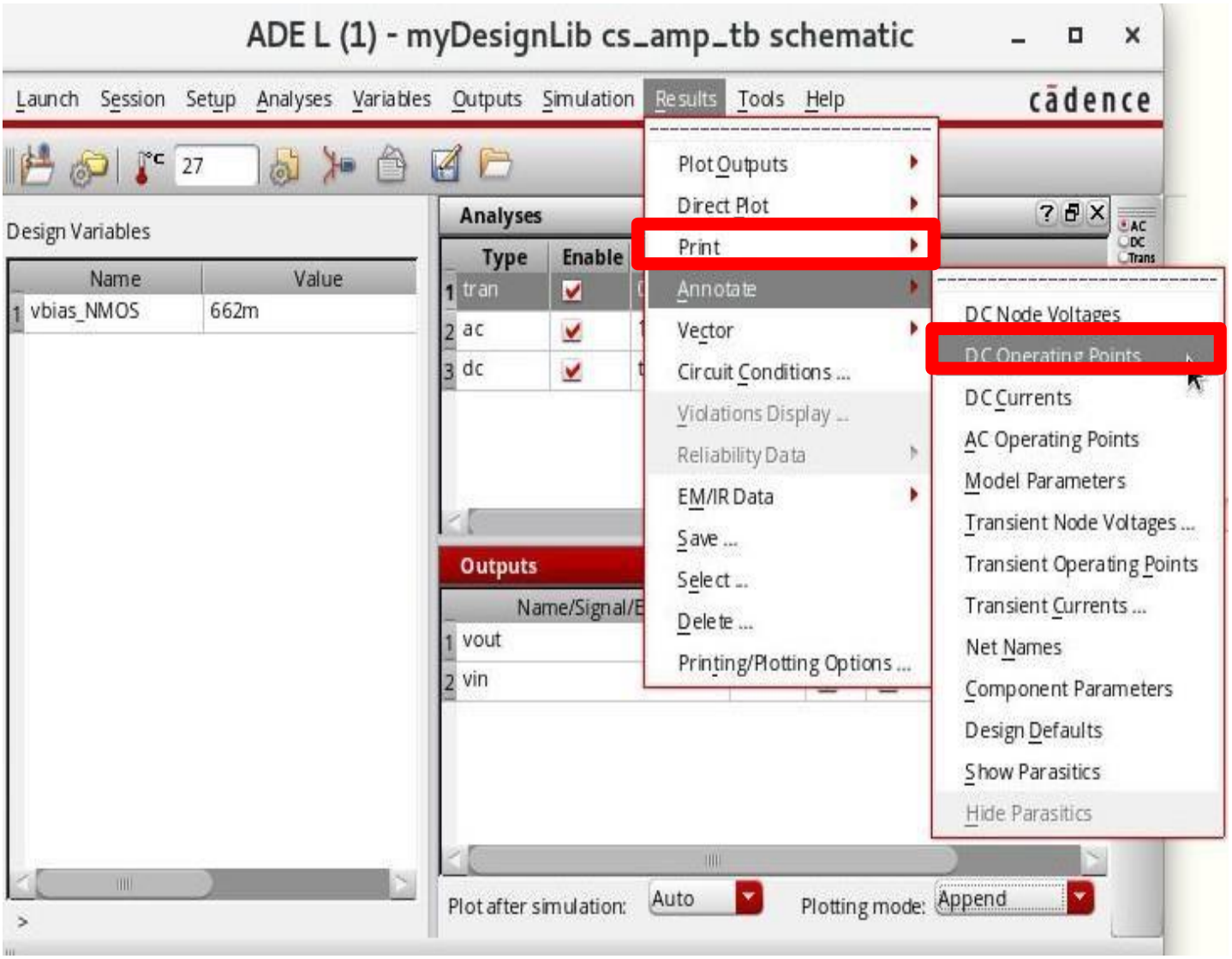
Input & Output Impedance Calculation

- From the Test Circuit, double click on the symbol to peep down the hierarchy
- Select '**new tab**' under '**Descend**' and click on '**OK**'
- Run a DC Analysis
- Go back to 'ADE L' window
- Select 'Results →Print →DC Operating Points'

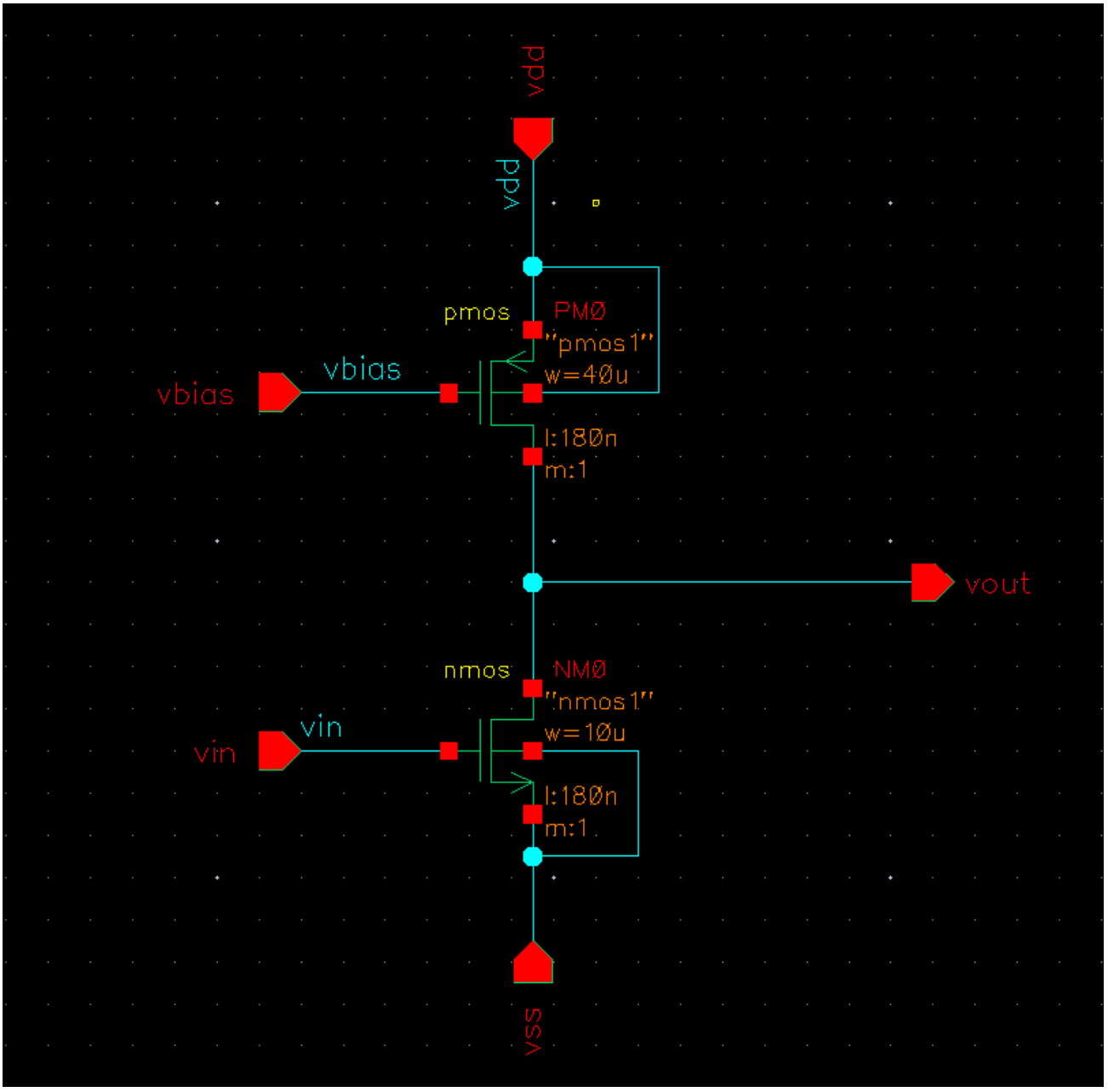
To peep down the hierarchy



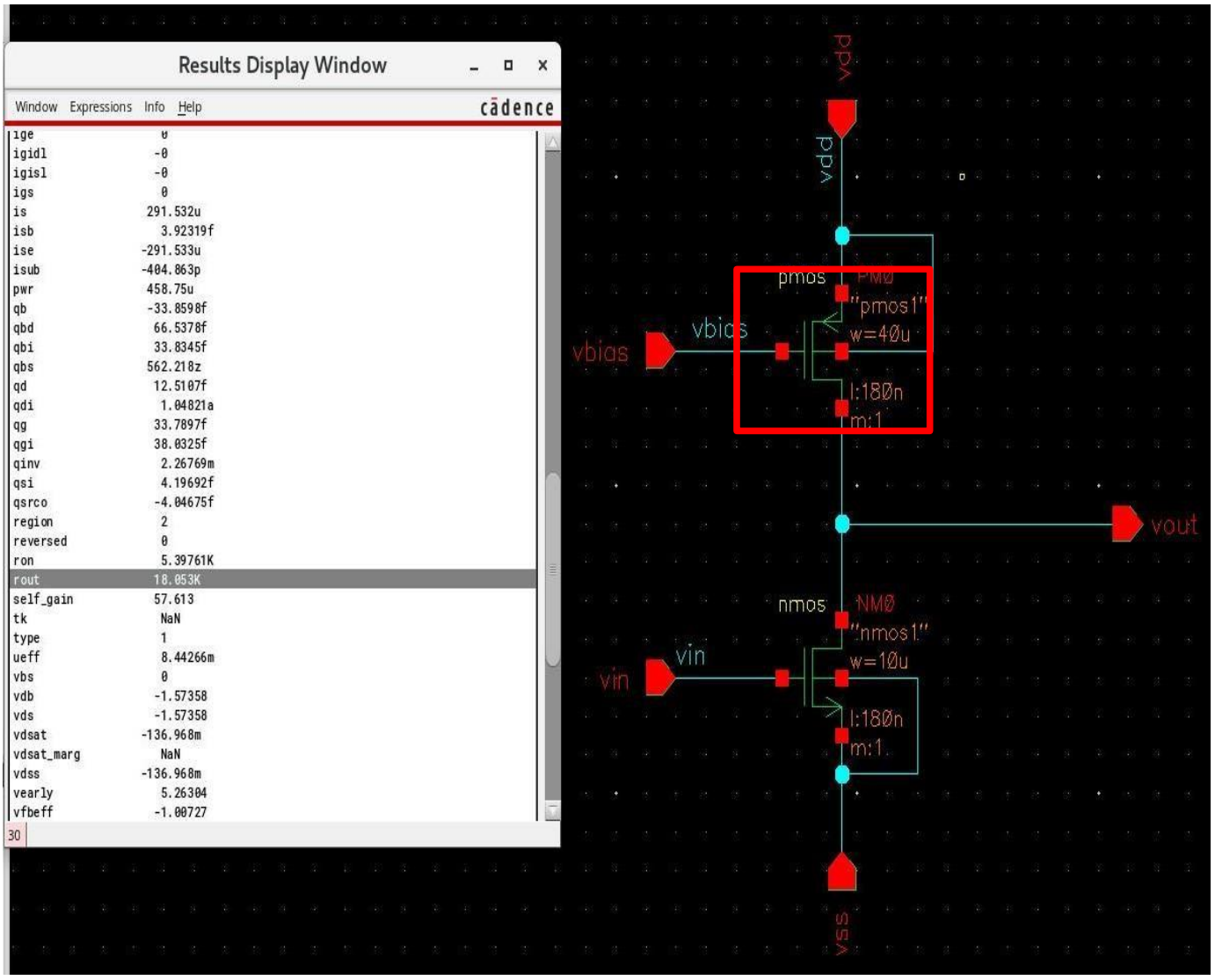
Go back to ‘ADE L’ window, Select ‘Results →Print →DC Operating Points’



Transistor level Schematic:



- Click on **PMOS** / **NMOS** transistor to print the Operating Points
- Scroll down to get the value of '**rout**'
- Similarly, **repeat** the process for the other transistor
- Get the equivalent resistance by using the formula for example '**ro1** || **ro2**'



2: Basic Common Gate Amplifier

Objective:

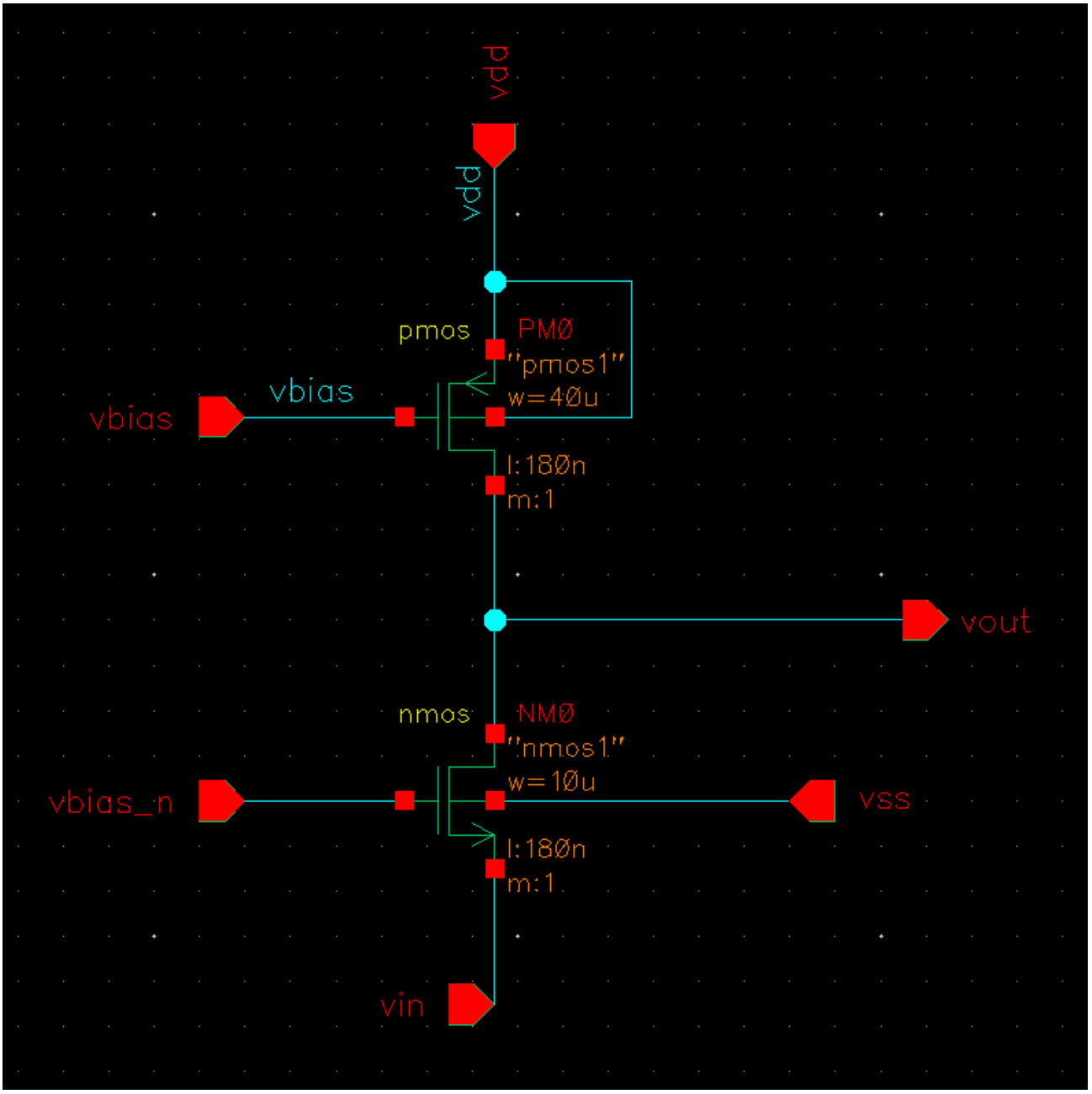
- To set up and run simulations on the CG Amplifier Test design.
- In this section, we will run the simulation for CG Amplifier and plot the Transient, DC and AC characteristics.
- Anayzing Gain, Bandwidth and Input and Output Impedance by simulating schematic test of cg amplifier

Theory:

A common-gate amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a current buffer or voltage amplifier. In the circuit the source terminal of the transistor serves as the input, the drain is the output and the gate is connected to ground, or common, hence its name.

The analogous bipolar junction transistor circuit is the common-base amplifier. Input signal is applied to the source, output is taken from the drain. Current gain is about unity, input resistance is low, output resistance is high a CG stage is a current buffer. It takes a current at the input that may have a relatively small Norton equivalent resistance and replicates it at the output port, which is a good current source due to the high output resistance

Schematic Capture of Common Gate Amplifier:



1: Schematic Entry

Objective: To create a new cell view and build Common Gate Amplifier

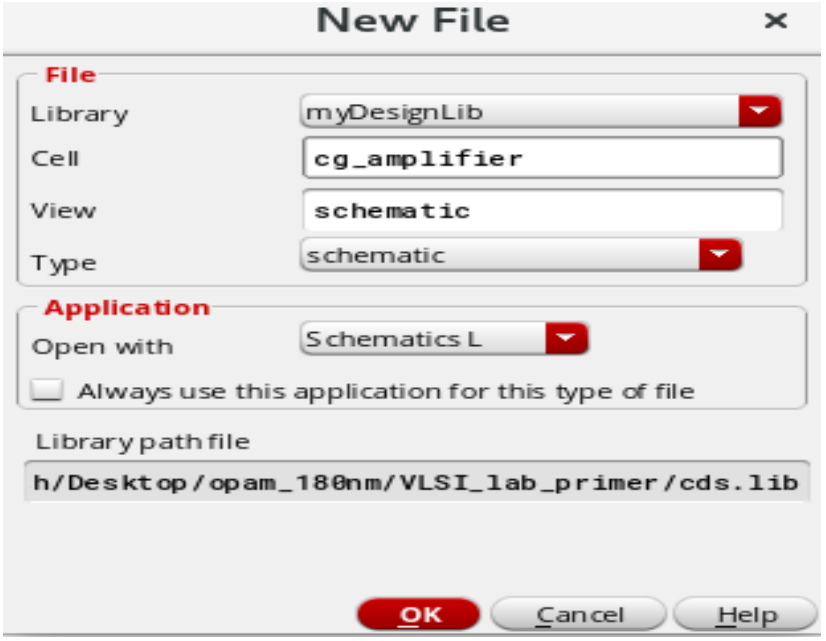
Creating a Schematic cellview:

Open a new schematic window in the myDesignLib library and build the cg_amplifier design.

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

2. Do not edit the library path file and the one above might be different from the path shown in your form.

3. Click OK when done the above settings. A blank schematic window for the CG Amplifier 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.

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

Library Name	Cell Name	Properties/Comments
gpdk 180	pmos	Model Name = pmos1; W=40u; L=180n
gpdk 180	nmos	Model Name = nmos1; W=10u; L=180n

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 Name	Direction
vdd , vss, vbias, vbias_n, vin	input
vout	output

Make sure that the direction field is set to **input/ouput/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.

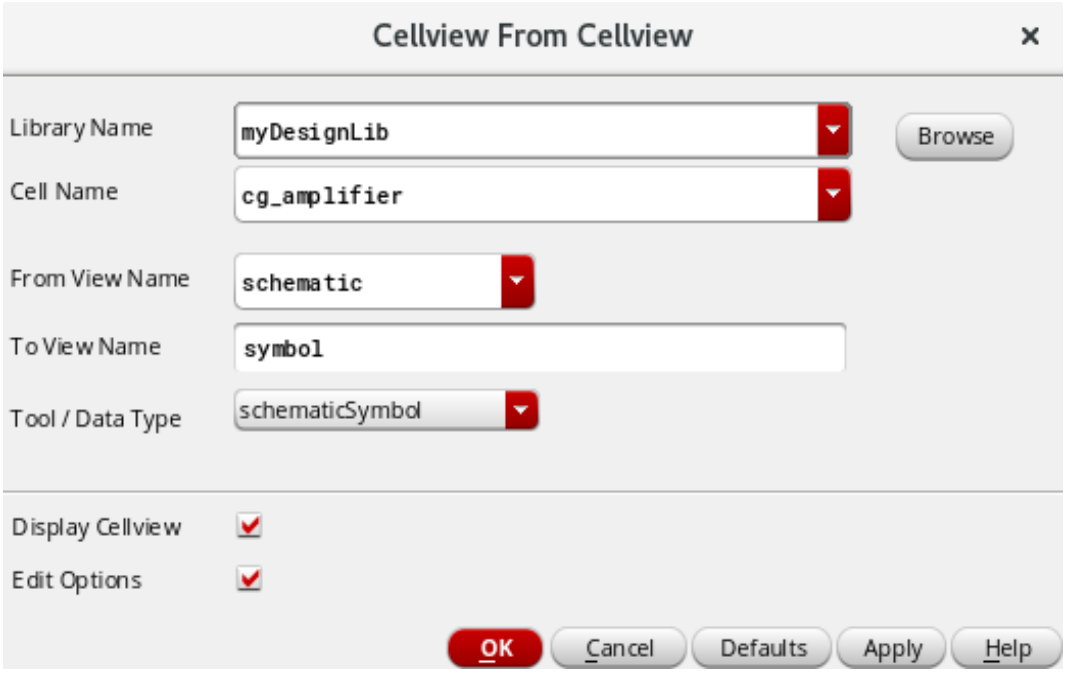
Saving the Design:
1. Click the **Check and Save** icon in the schematic editor window.
2. Observe the **CIW** output area for any errors.

Symbol Creation:
Objective: To create a symbol for the Common Gate Amplifier

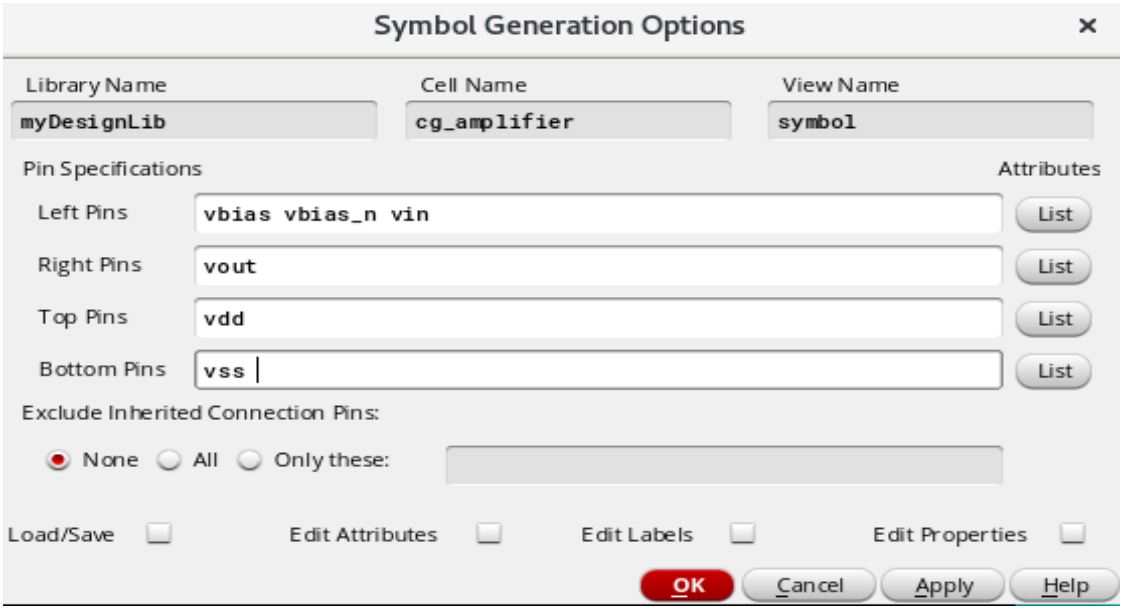
In this section, you will create a symbol for your cg_amplifier design, so you can place it in a test circuit for simulation. A symbol view is extremely important step in the design process.

1. In the CS 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**

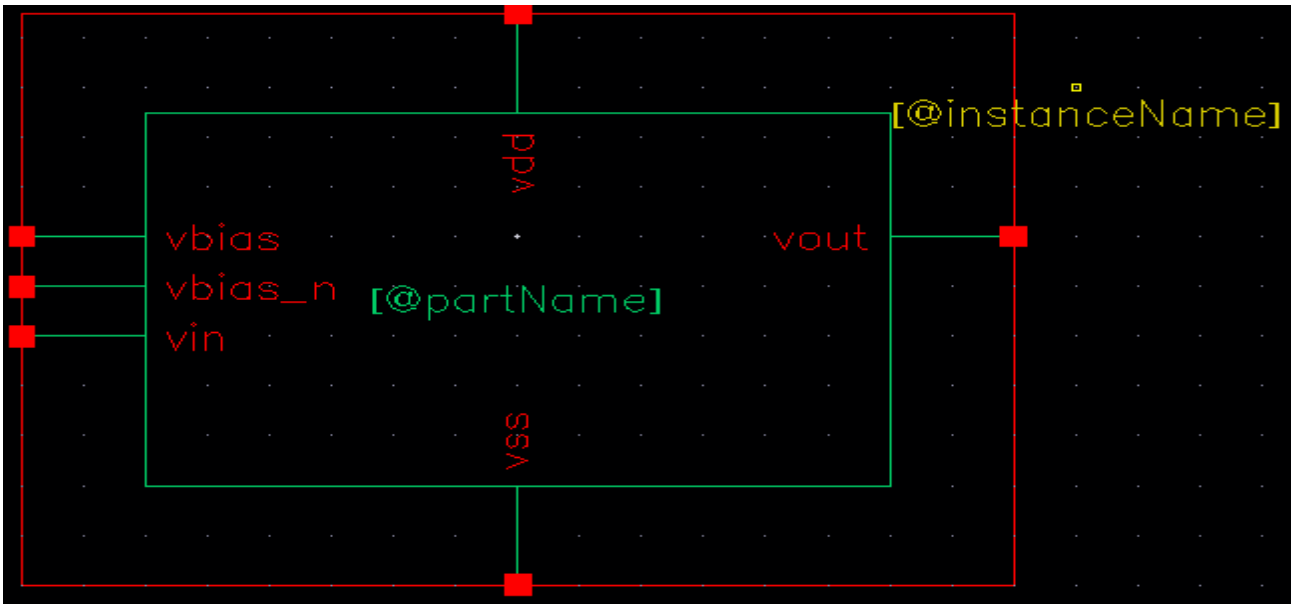

Cellview From Cellview
Library Name
myDesignLib
Browse
Cell Name
cg_amplifier
From View Name
schematic
To View Name
symbol
Tool / Data Type
schematicSymbol
Display Cellview
Edit Options
OK
Cancel
Defaults
Apply
Help

3. Click **OK** in the **Cellview From Cellview** form. The Symbol Generation Form appears.
4. Modify the Pin Specifications as follows:



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

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



Editing a Symbol: In this section we will modify the CG Amplifier symbol.

1. Move the cursor over the automatically generated symbol, until the green rectangle is highlighted, click **left** to select it.

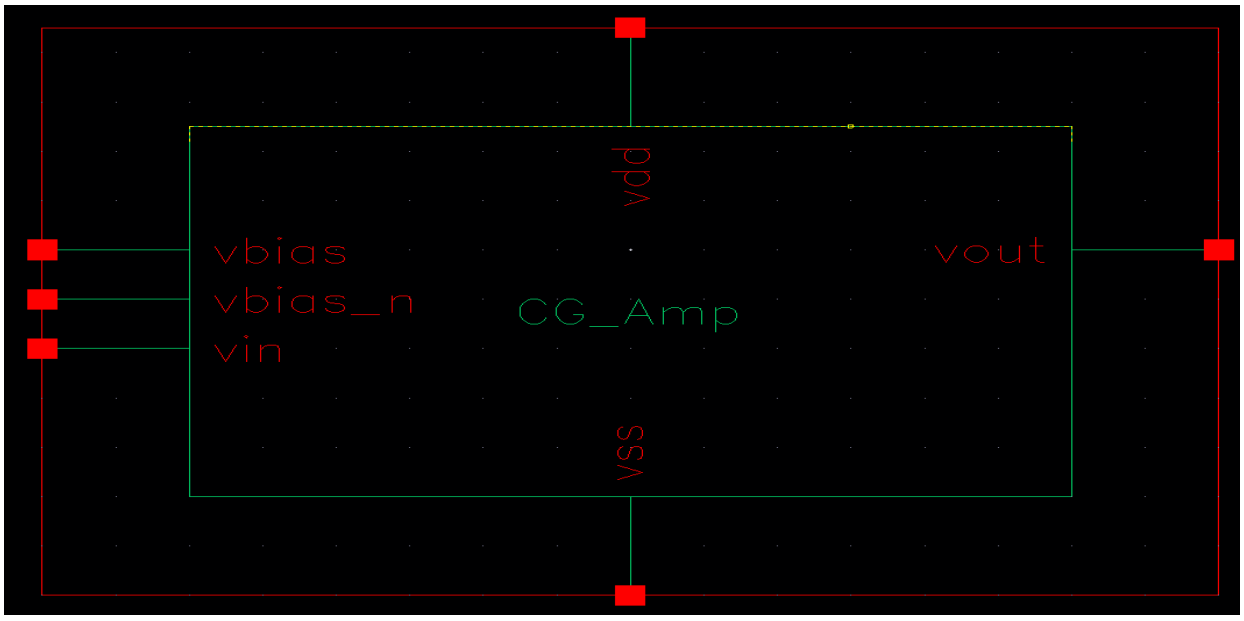
2. Click **Delete** icon in the symbol window, similarly select the red rectangle and delete that.

3. Execute **Create – Shape – polygon** and draw a shape similar to triangle.

4. After creating the rectangle press **ESC** key.

5. You can move the pin names according to the location.

6. Execute **Create — Selection Box**. In the Add Selection Box form, click **Automatic**. A new red selection box is automatically added.



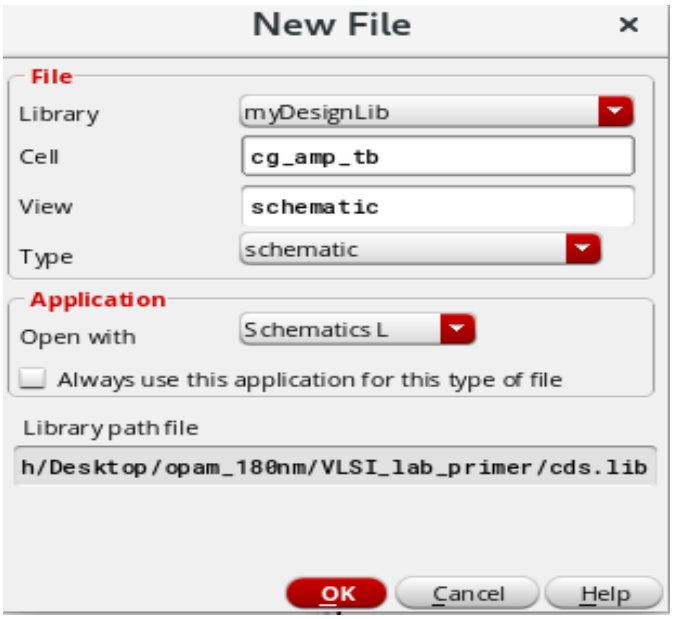
7. After creating symbol, click on the **save** icon in the symbol editor window to save the symbol. In the symbol editor, execute **File — Close** to close the symbol view window

2: Building the cg_amplifier_test Design

Objective: To build Common Gate Amplifier Test circuit using your CG Amplifier

Creating the Common Gate 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 cg_ amplifier_test design appears.

Building the cg_amplifier_test Circuit:

1. Using the component list and Properties/Comments in this table, build the cg_amplifier_test schematic.

Design entry for test circuit:

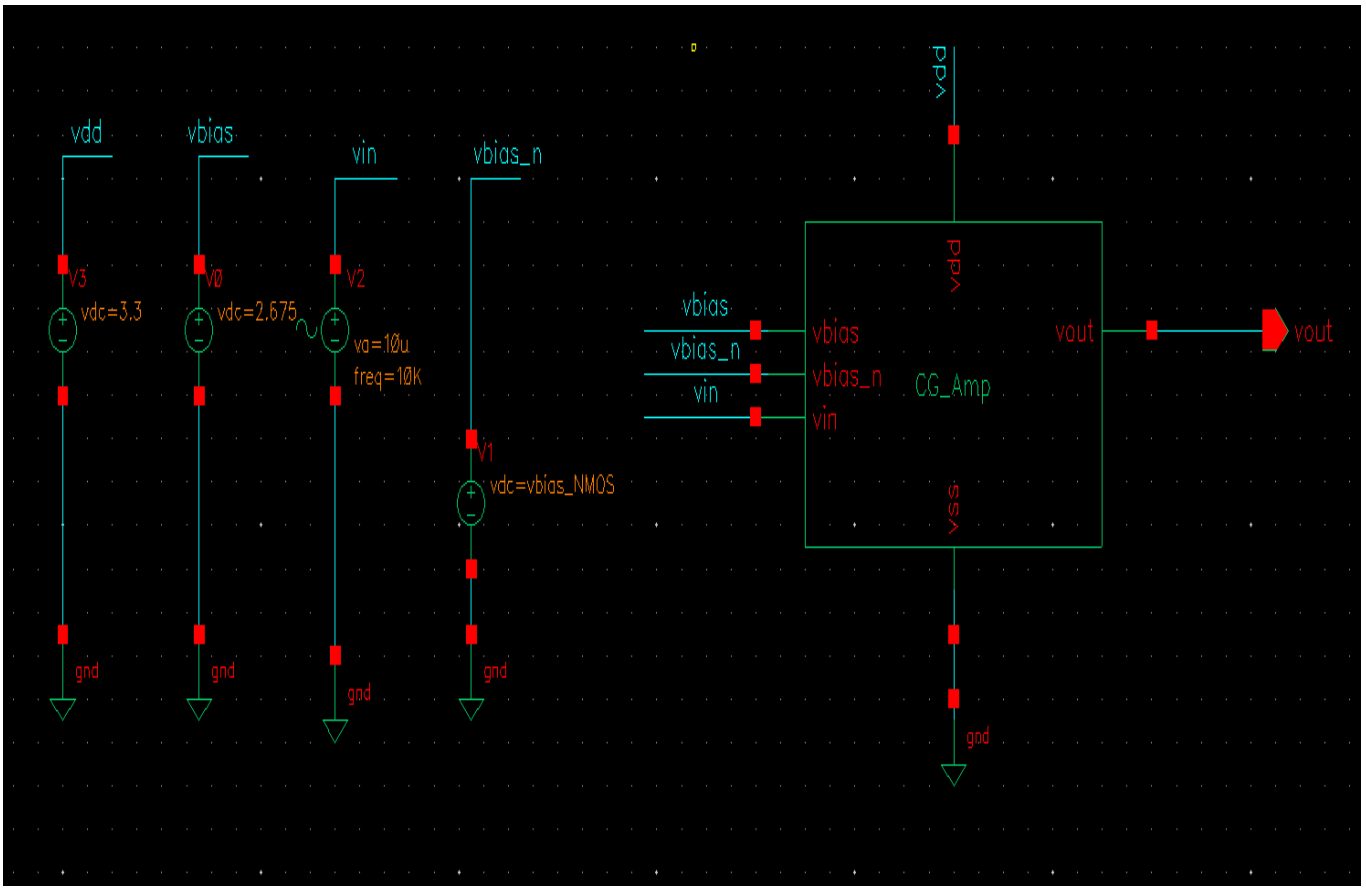
Library Name	Cellview Name	Properties
myDesignLib	cg_amplifier	symbol
analogLib	vsin	Define pulse specification as AC Magnitude =1; Amplitude =10u; Frequency =10k
analogLib	vdc, vbias, vbias_NMOS	vdc=3.3v; vbias=2.675; vibas_NMOS=662m

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 this.



6. Leave your cg_amplifier_test schematic window open for the next section.

3: Analog Simulation with Spectre

Objective:

- To set up and run simulations on the CG Amplifier Test design.
- In this section, we will run the simulation for CG Amplifier and plot the Transient, DC and AC characteristics.
- Anayzing Gain, Bandwidth and Input and Output Impedance by simulating schematic test of cg amplifier

Starting the Simulation Environment:

1. In the cg amplifier_test schematic window, execute **Launch – ADE L.** The Analog Design Environment simulation window appears.

Choosing a Simulator:

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:

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 **gpdk180.scs** file in the **./models/spectre** directory. Select **NN** in Section field, click **Add** and click **OK.**

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 **Analyses – Choose** icon. 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**

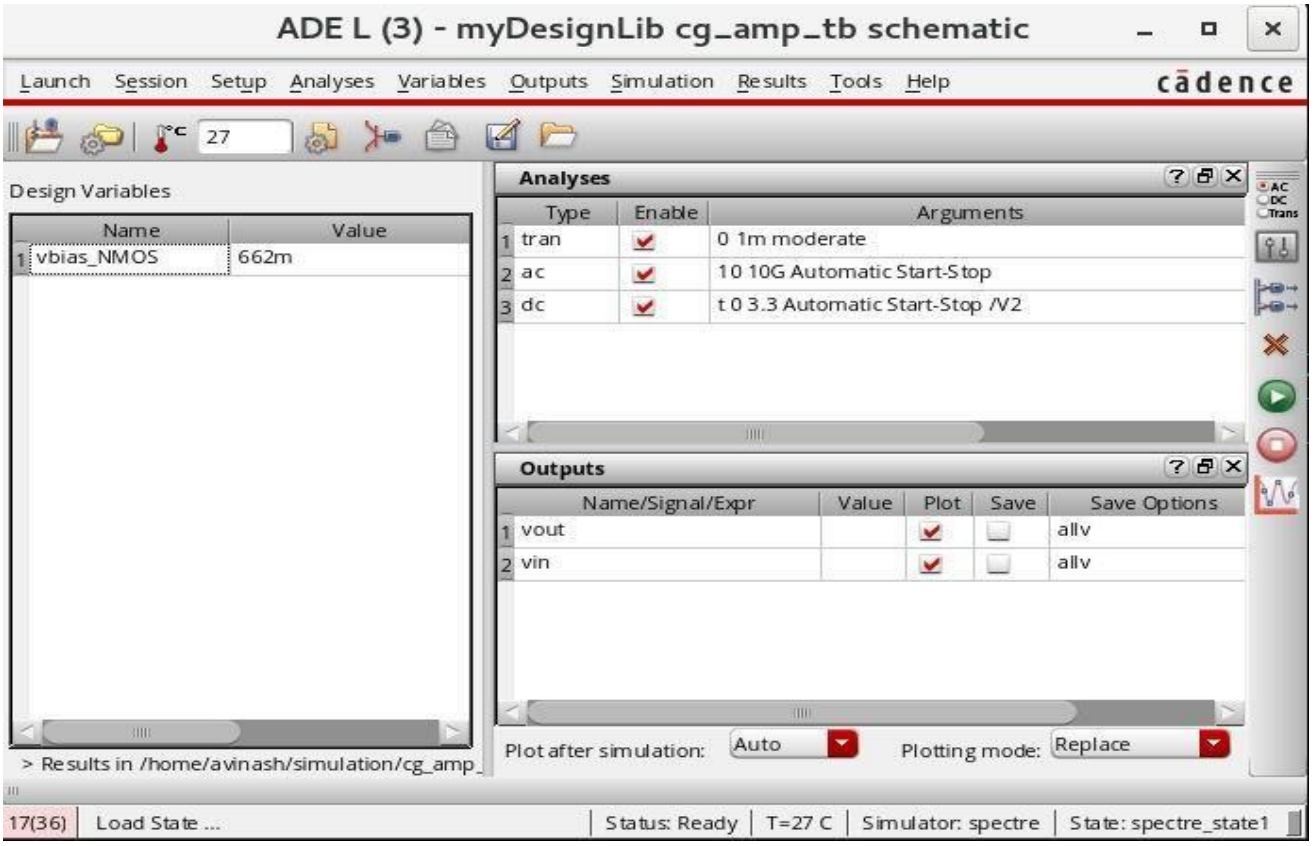
b. Set the stop time as **1m**

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 **0 to 3.3** 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 **10** to **10G**
   d. Sweep type **Automatic**.
   e. Check the enable button and then click **Apply**.
5. Click **OK** in the Choosing Analyses Form.

Settings in ADE L Window:



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.

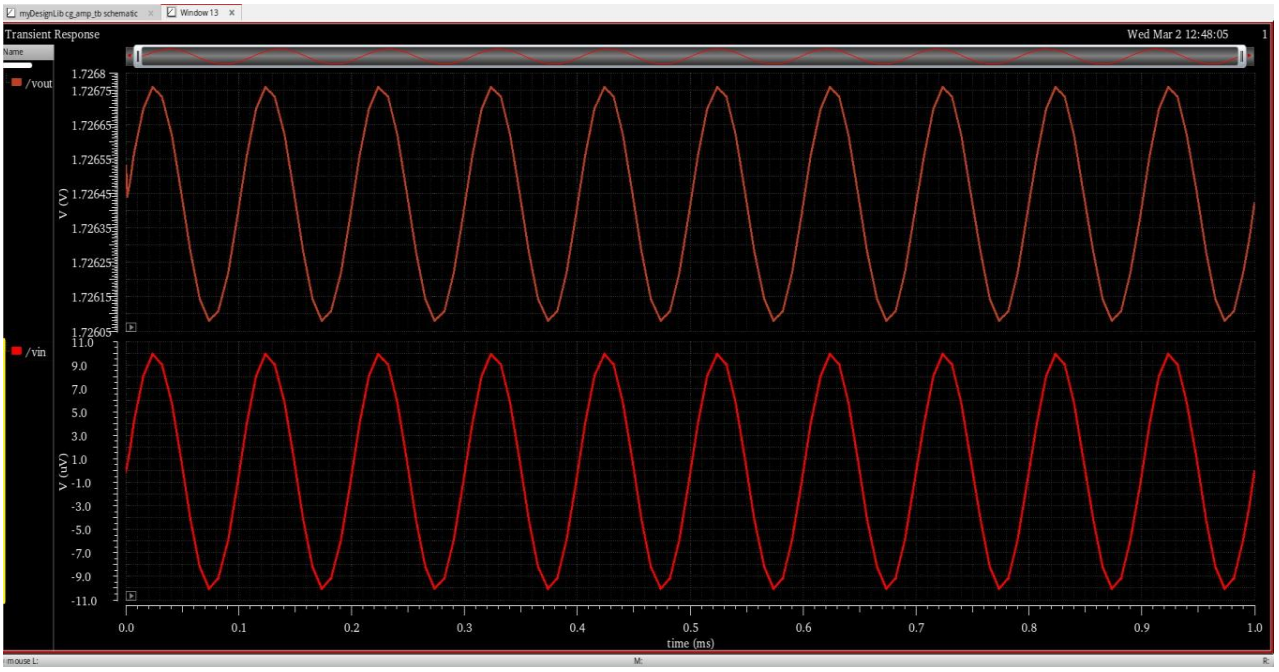
2. Follow the prompt at the bottom of the schematic window, Click on output net **Vout**, input net **Vin** of the cg_amplifier. Press **ESC** with the cursor in the schematic after selecting node.

Running the Simulation:

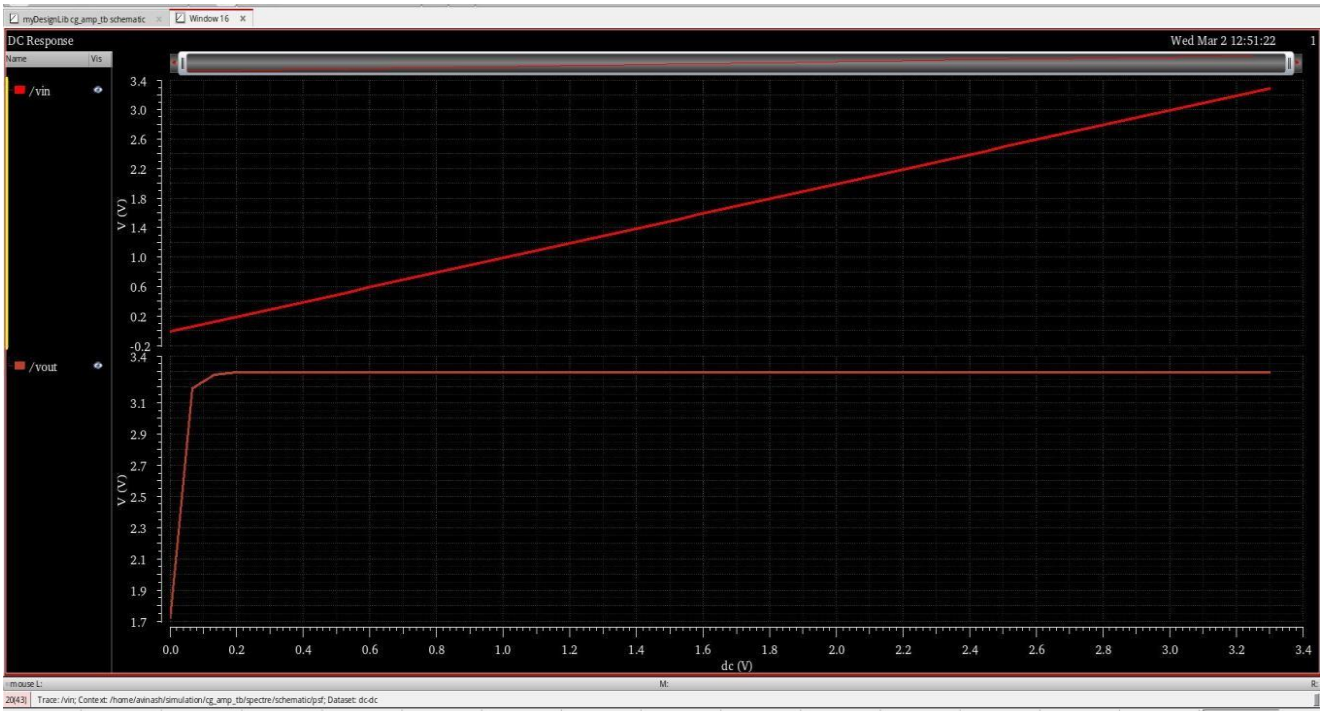
1. Execute Simulation – Netlist and Run in the simulation window to start the simulation, this will create the netlist as well as run the simulation.

2. When simulation finishes, the Transient, DC and AC plots automatically will be popped up along with netlist.

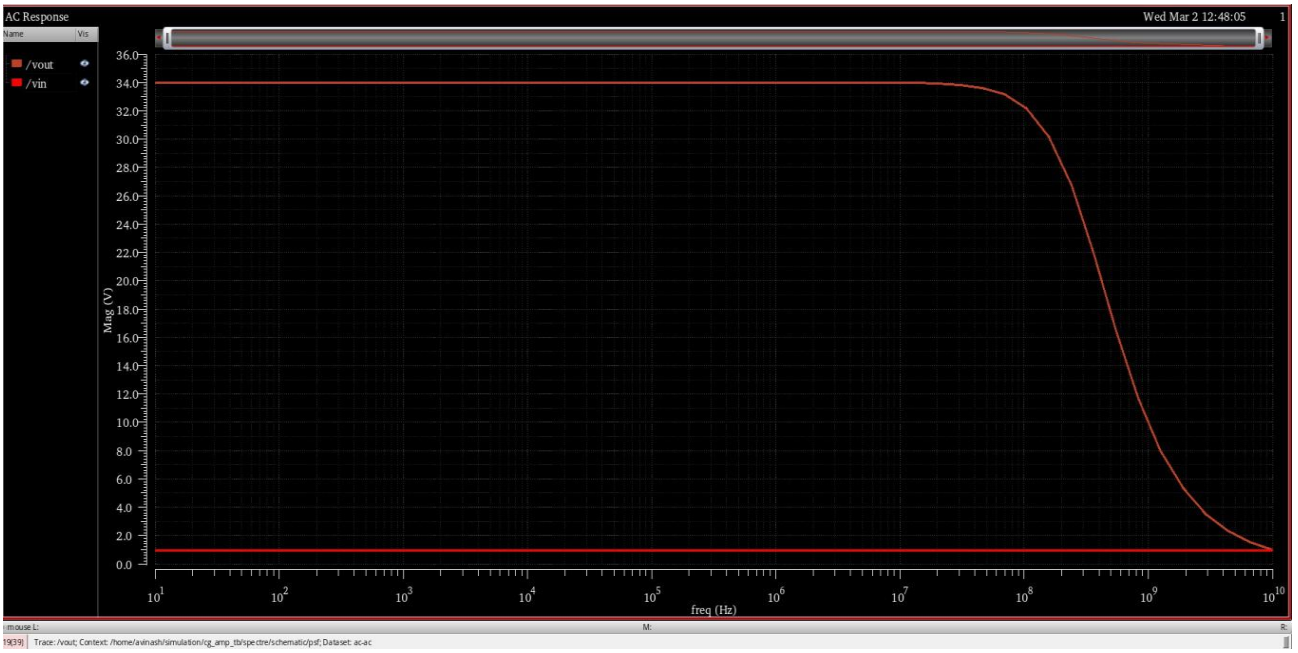
Transient Response :



DC Response

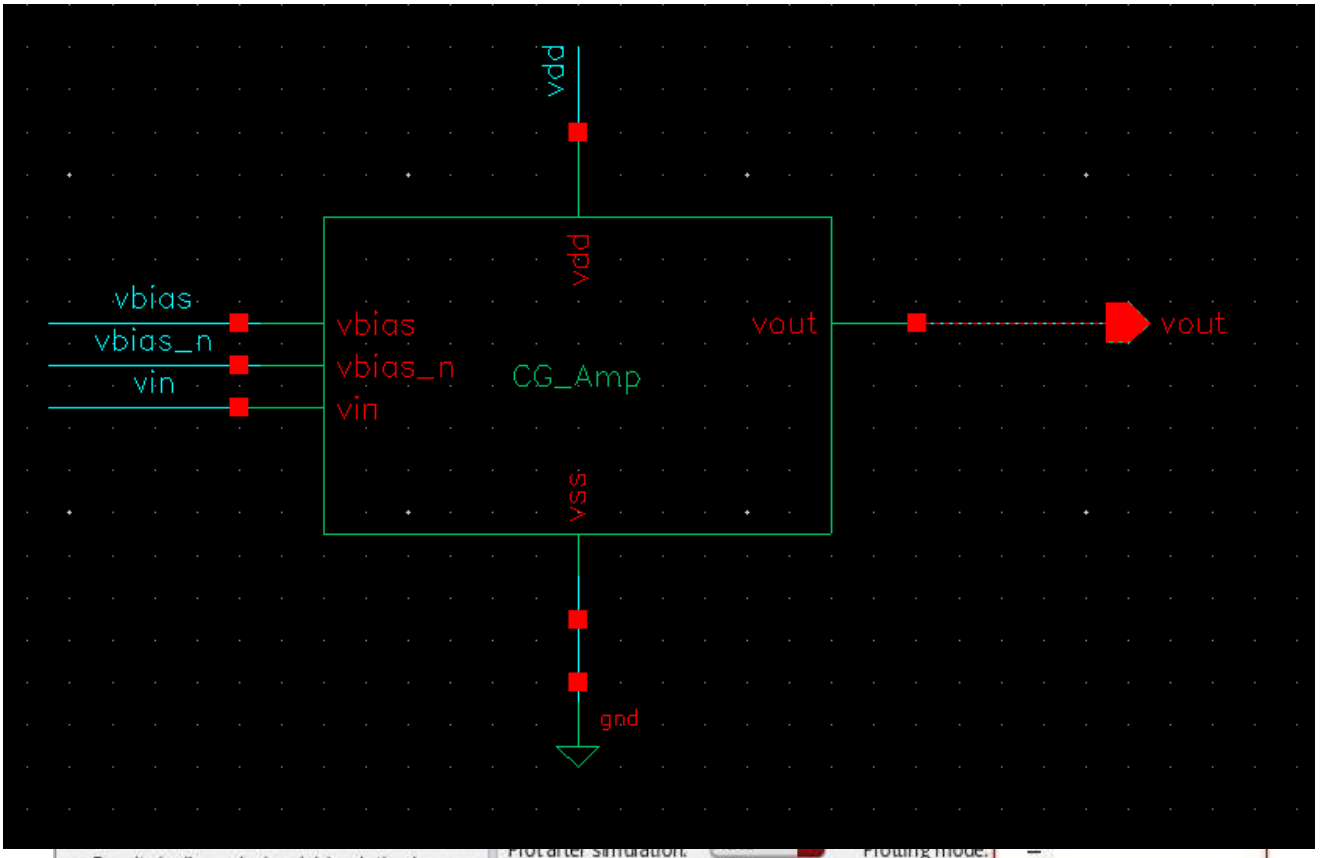


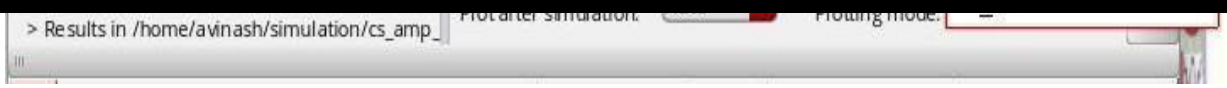
AC Response:



4: Gain Calculation:

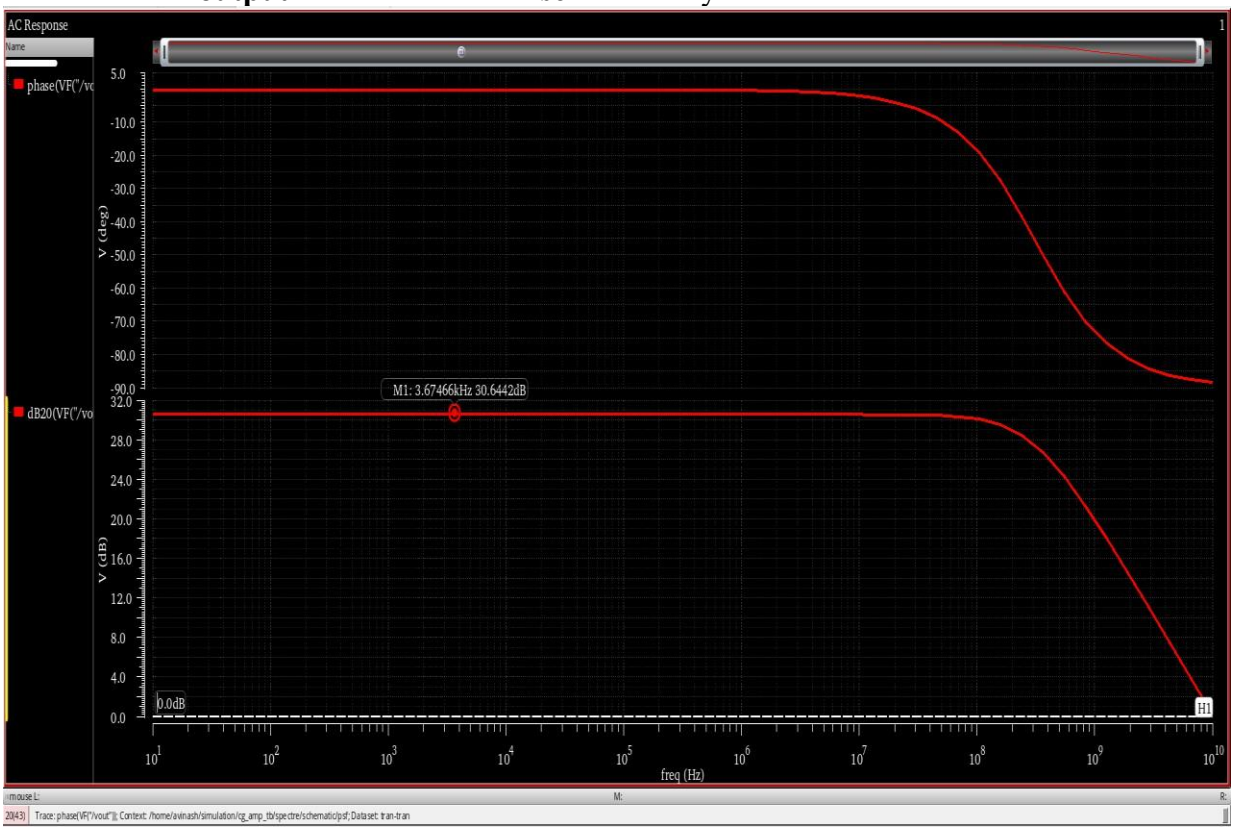
After running AC Analysis, Go back to **ADE L**
Select "Results-→Direct Plot → AC Magnitude & Phase"

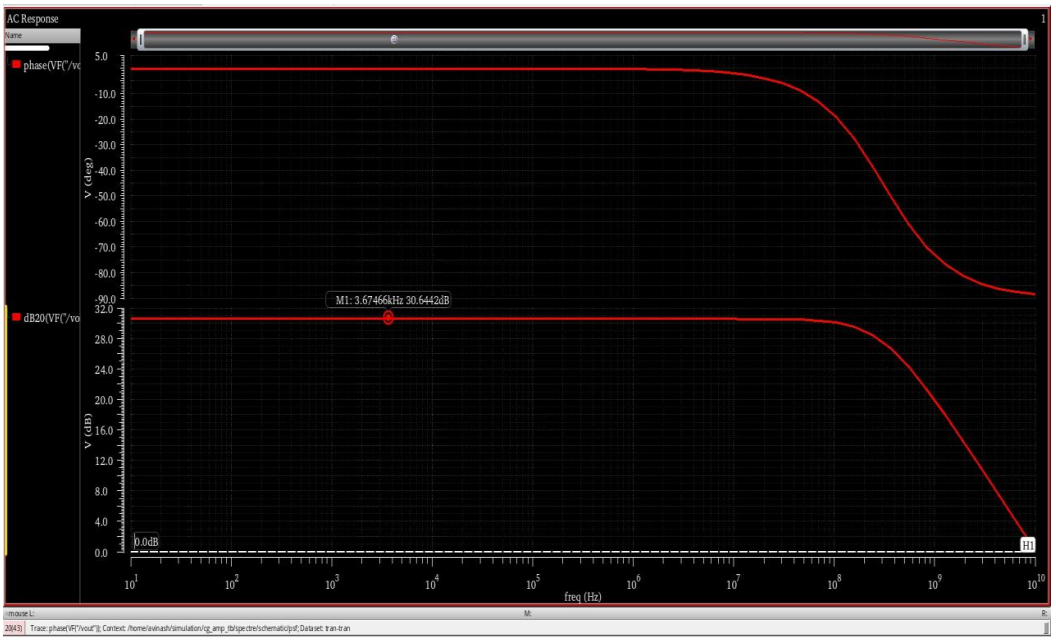




It will take you back to the Test Schematic

- Select the **output** net and click on '**Esc**' on the keyboard



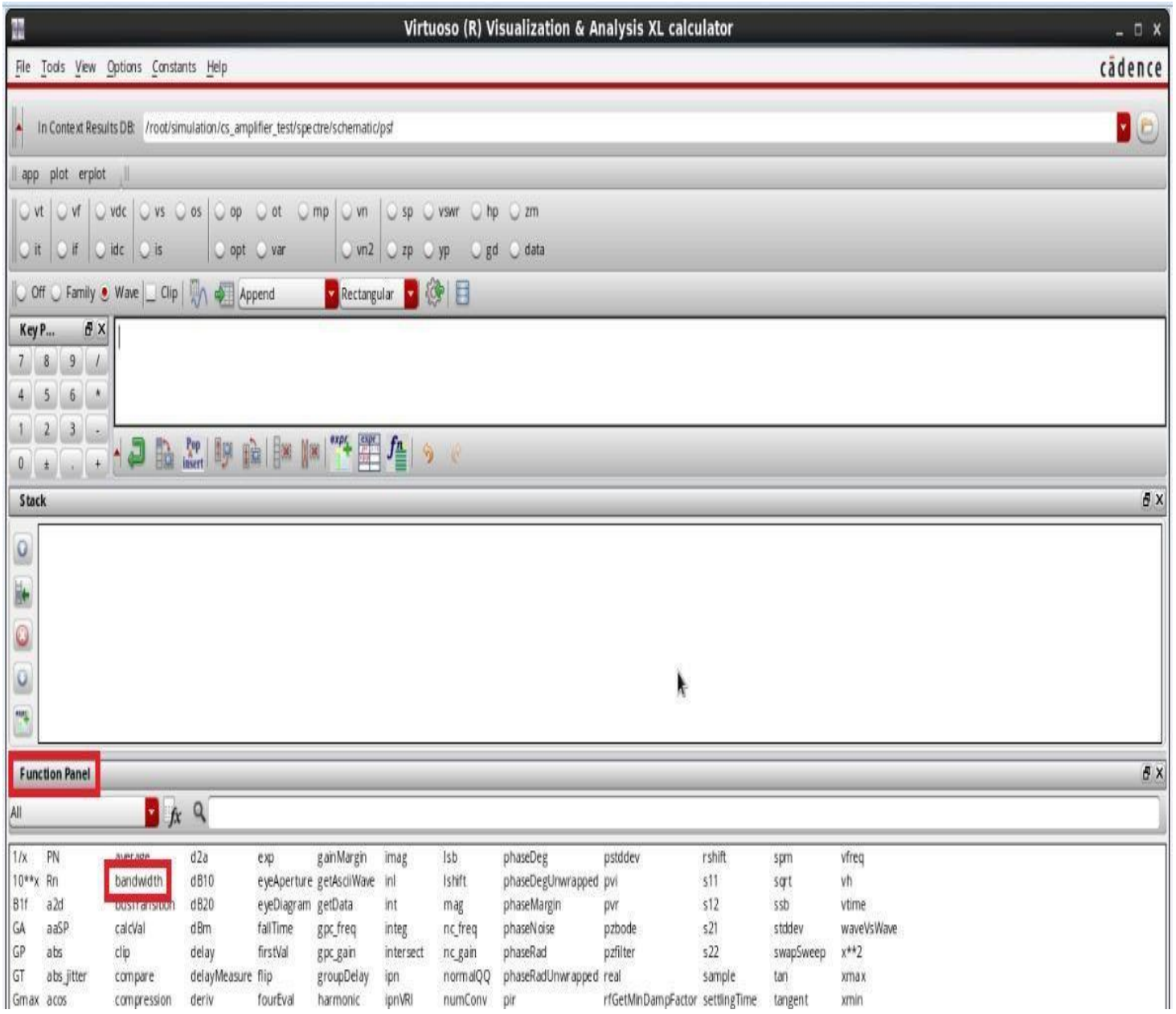


- Use the bind key '**M**' to place '**Marker**' as in the above graph
- The value that can be seen is the '**Gain**' in dB

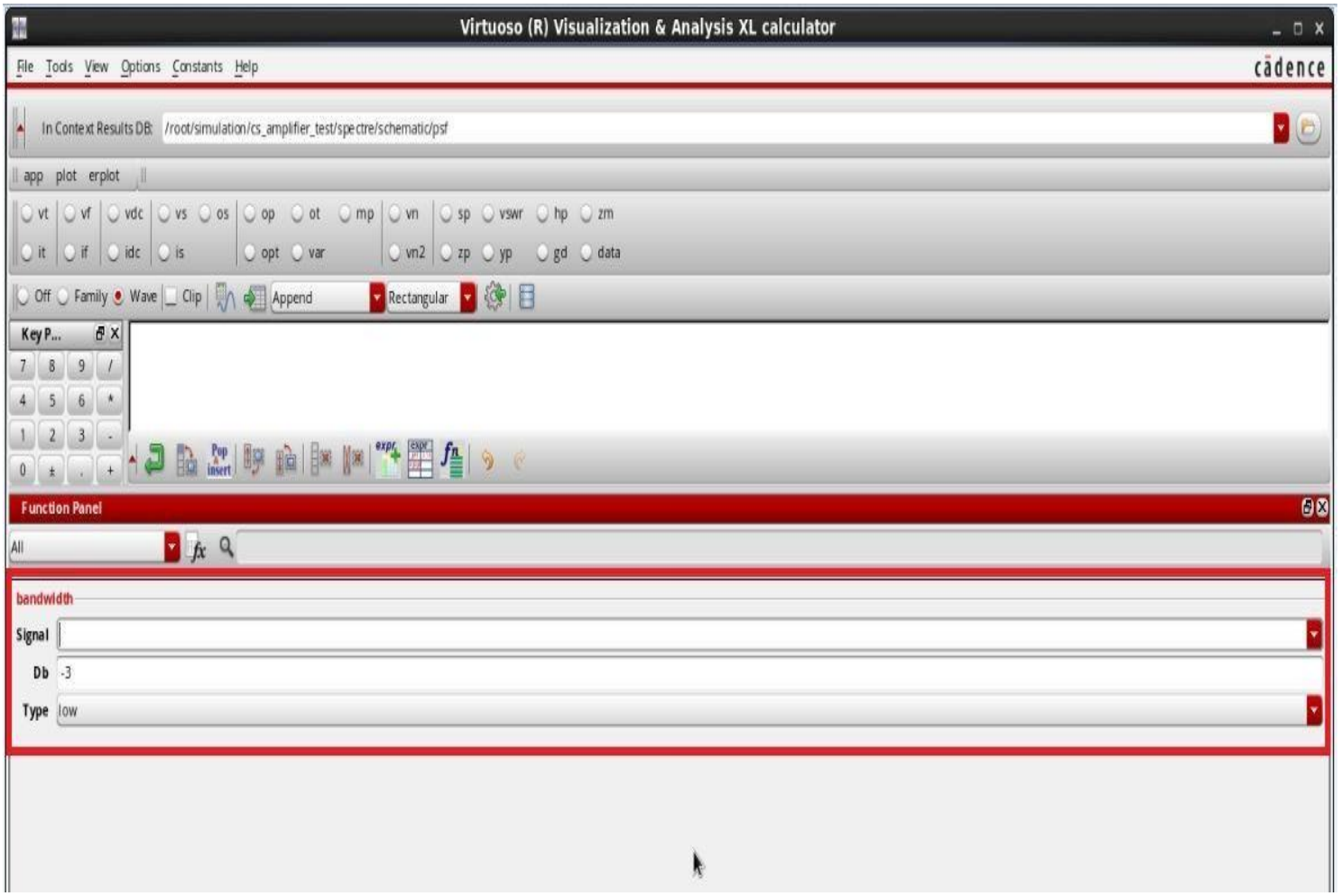
5: Bandwidth Calculation:

From the Graph window,

- Select " **Tools → Calculator** "
- Select "**Bandwidth**" from the "**Function Panel**"


Virtuoso (R) Visualization & Analysis XL calculator
File Tools View Options Constants Help
cadence
In Context Results DB: /root/simulation/cs_amplifier_test/spectre/schematic/psf
app plot erplot
vt vf vdc vs os op ot mp vn sp vswr hp zm
it if idc is opt var vn2 zp yp gd data
Off Family Wave Clip Append Rectangular
KeyP...
Stack
Function Panel
All
1/x PN d2a exp gainMargin imag lsb phaseDeg pstddev rshift spm vfreq
10**x Rn bandwidth dB10 eyeAperture getAsciiWave inl lshift phaseDegUnwrapped pvi s11 sqrt vh
B1f a2d dB20 eyeDiagram getData int mag phaseMargin pvr s12 ssb vtime
GA aaSP calcVal dBm fallTime gpc_freq integ nc_freq phaseNoise pzbode s21 stddev waveVsWave
GP abs clip delay firstVal gpc_gain intersect nc_gain phaseRad pzfilter s22 swapSweep x**2
GT abs_jitter compare delayMeasure flip groupDelay ipn normalQQ phaseRadUnwrapped real sample tan xmax
Gmax acos compression deriv fourEval harmonic ipnVRI numConv pir rfGetMinDampFactor settlingTime tangent xmin

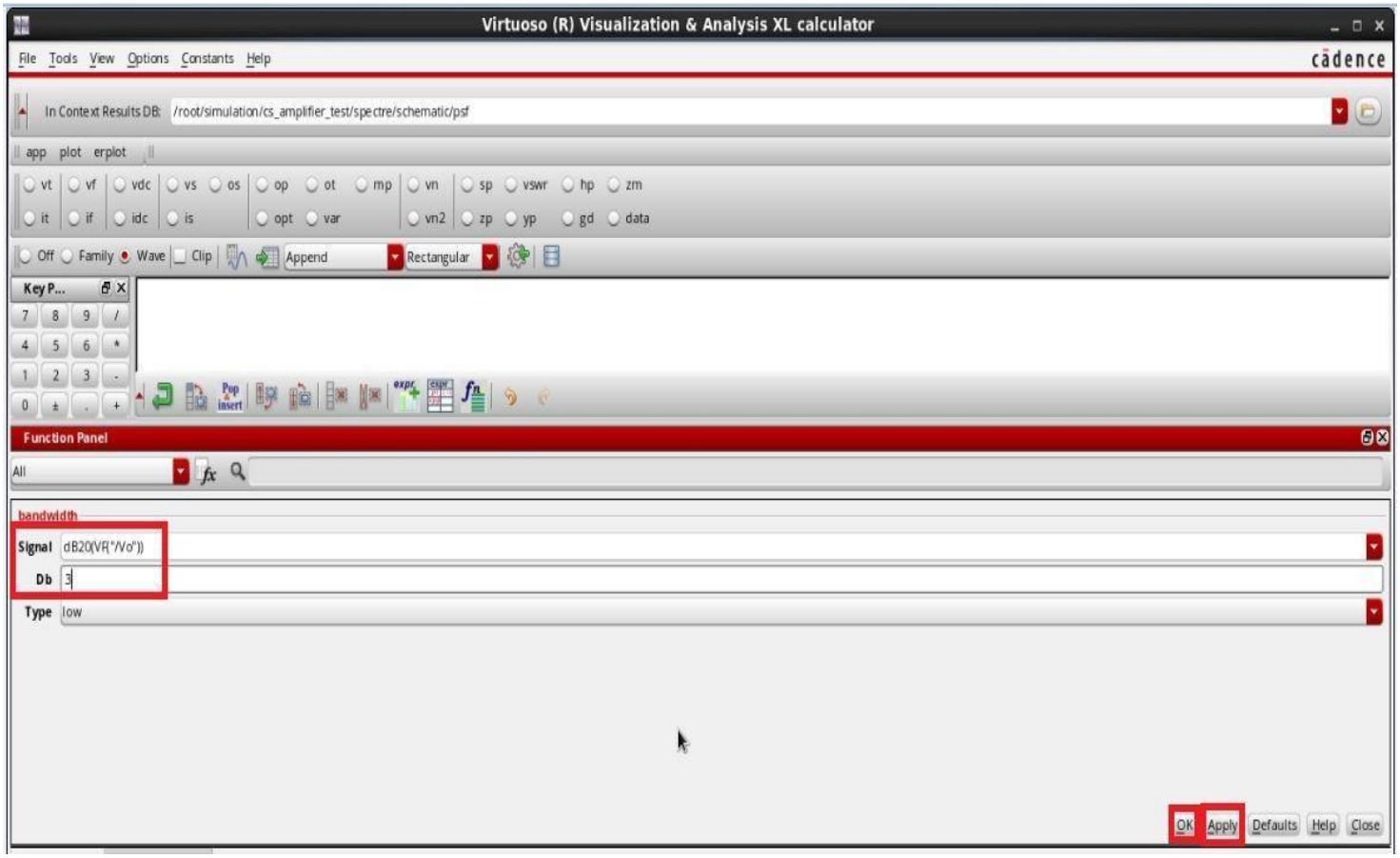
Bandwidth in Function Panel:



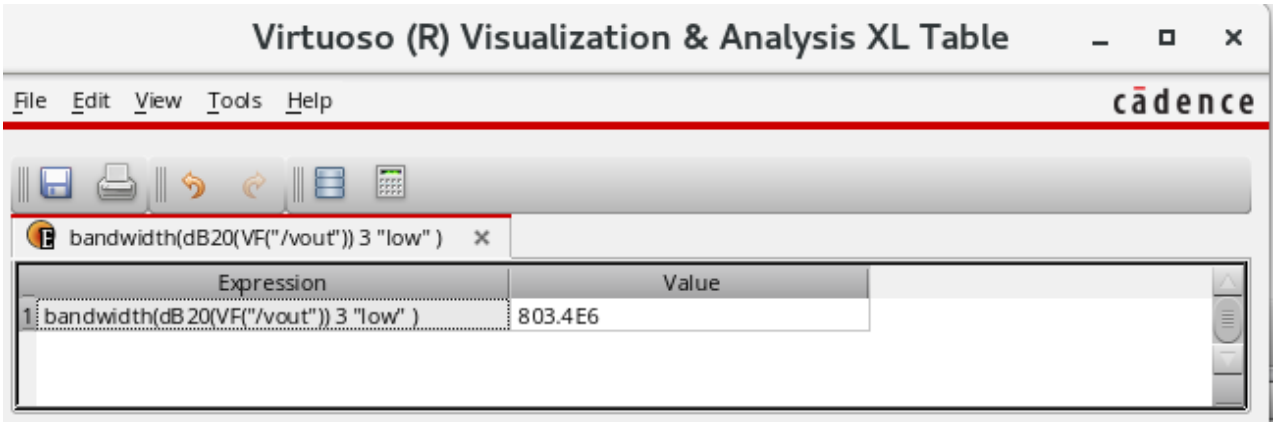
Bandwidth

Place the cursor in '**Signal**'
Go back to the waveform window
Select the '**dB20**' curve
Mention '**3**' as 'Db' value to calculate '**3dB** Bandwidth'
Click on '**Apply**' and then '**OK**'

Bandwidth:



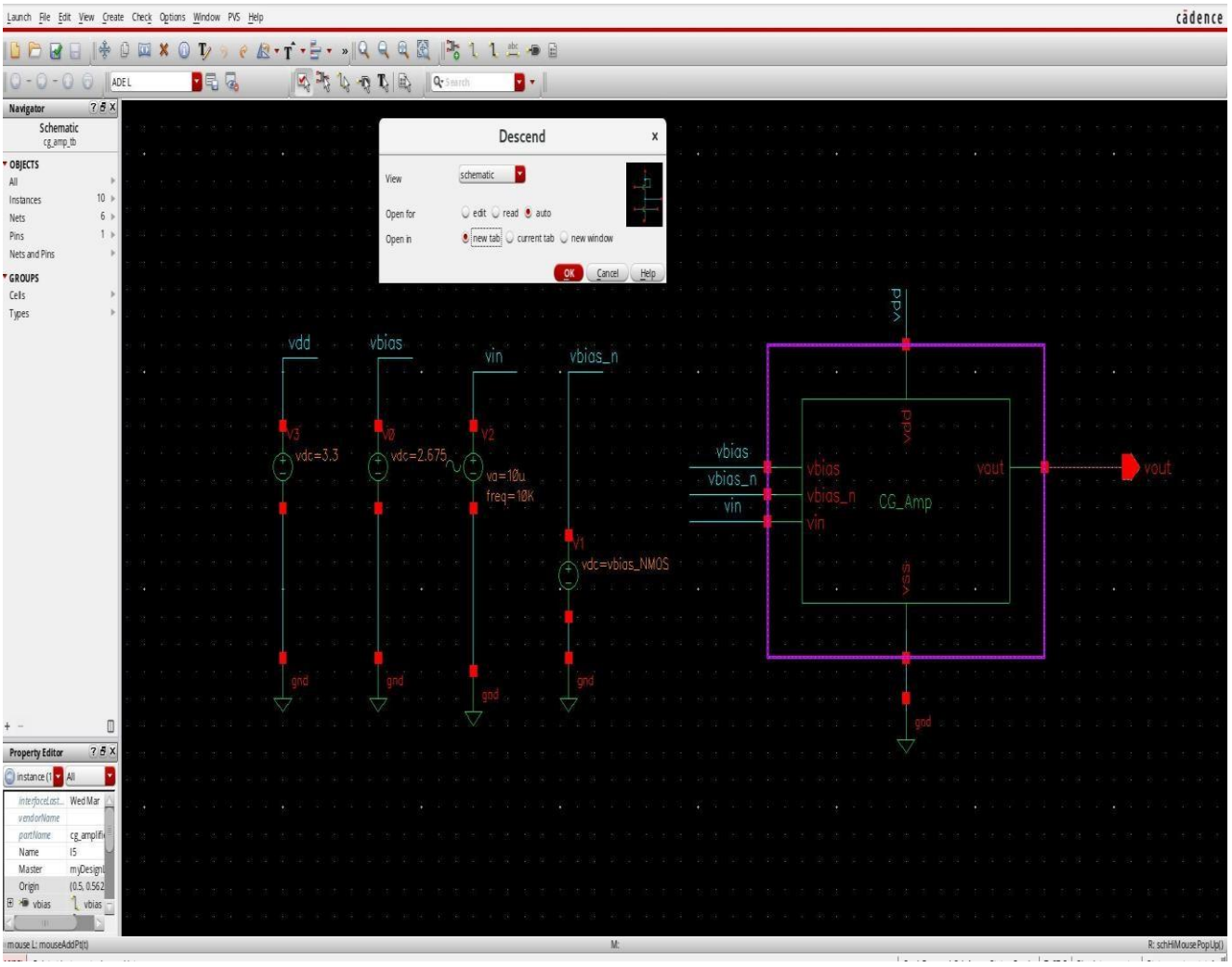
Evaluated Bandwidth:



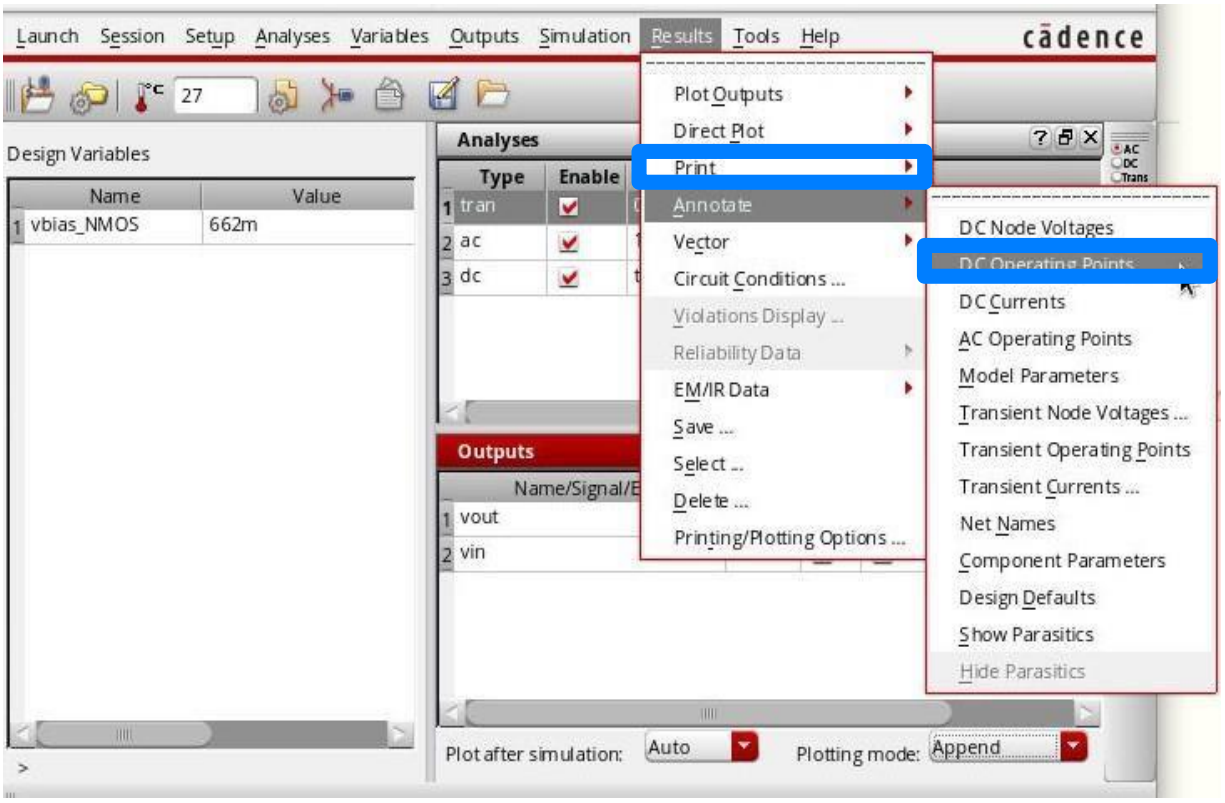
6: Input & Output Impedance Calculation:

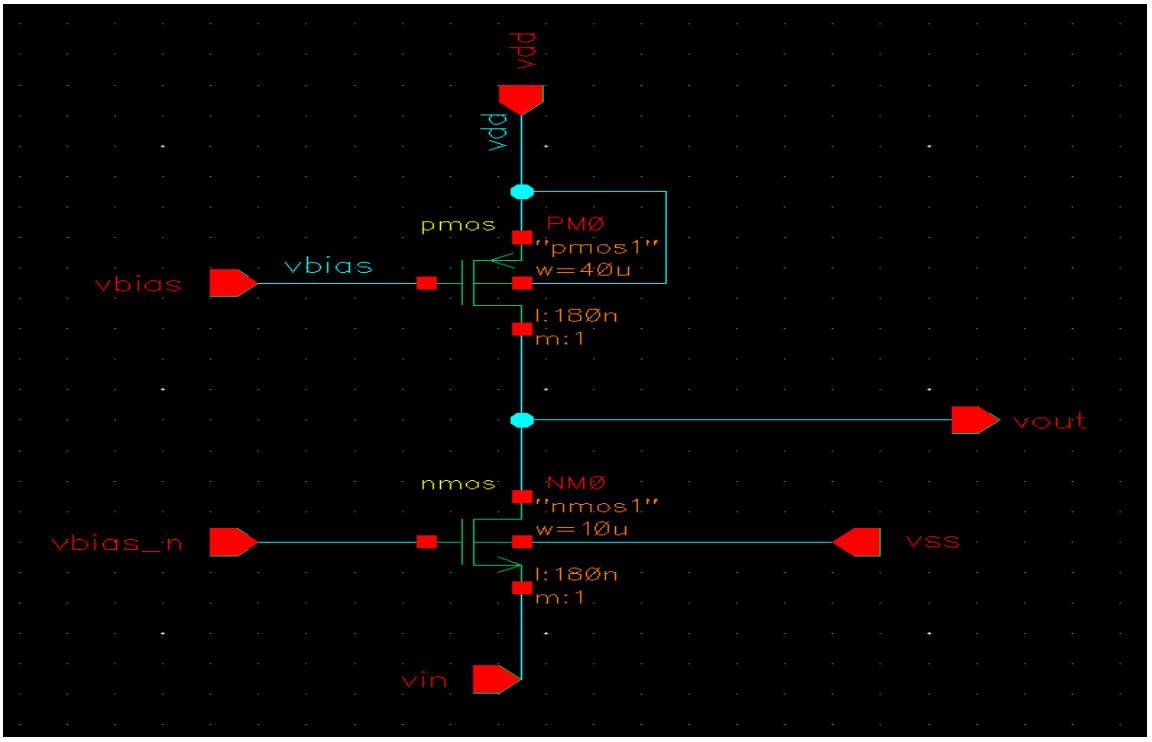
- From the Test Circuit, double click on the symbol to peep down the hierarchy
- Select '**new tab**' under '**Descend**' and click on 'OK'
- Run a DC Analysis
- Go back to 'ADE L' window
- Select 'Results → Print → DC Operating Points'

To peep down the hierarchy



Go back to ‘ADE L’ window, Select ‘Results →Print →DC Operating Points’

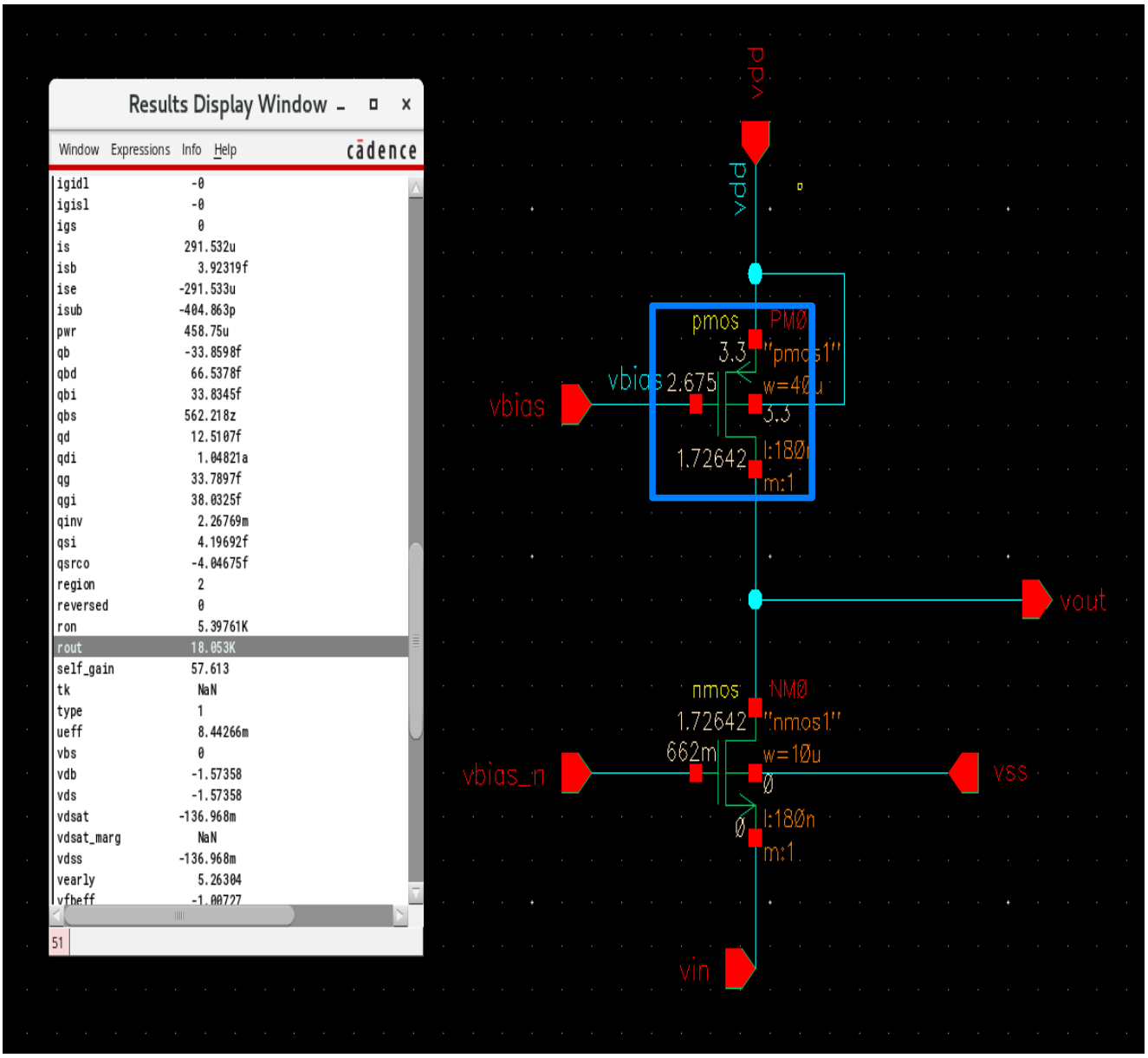




Transistor level Schematic:

- Click on **PMOS** / **NMOS** transistor to print the **Operating Points**
- Scroll down to get the value of ‘**rout**’
- Similarly, **repeat** the process for the other transistor

- Get the equivalent resistance by using the formula for example '**ro1 || ro2**'



13: Differential Amplifier

Objective:

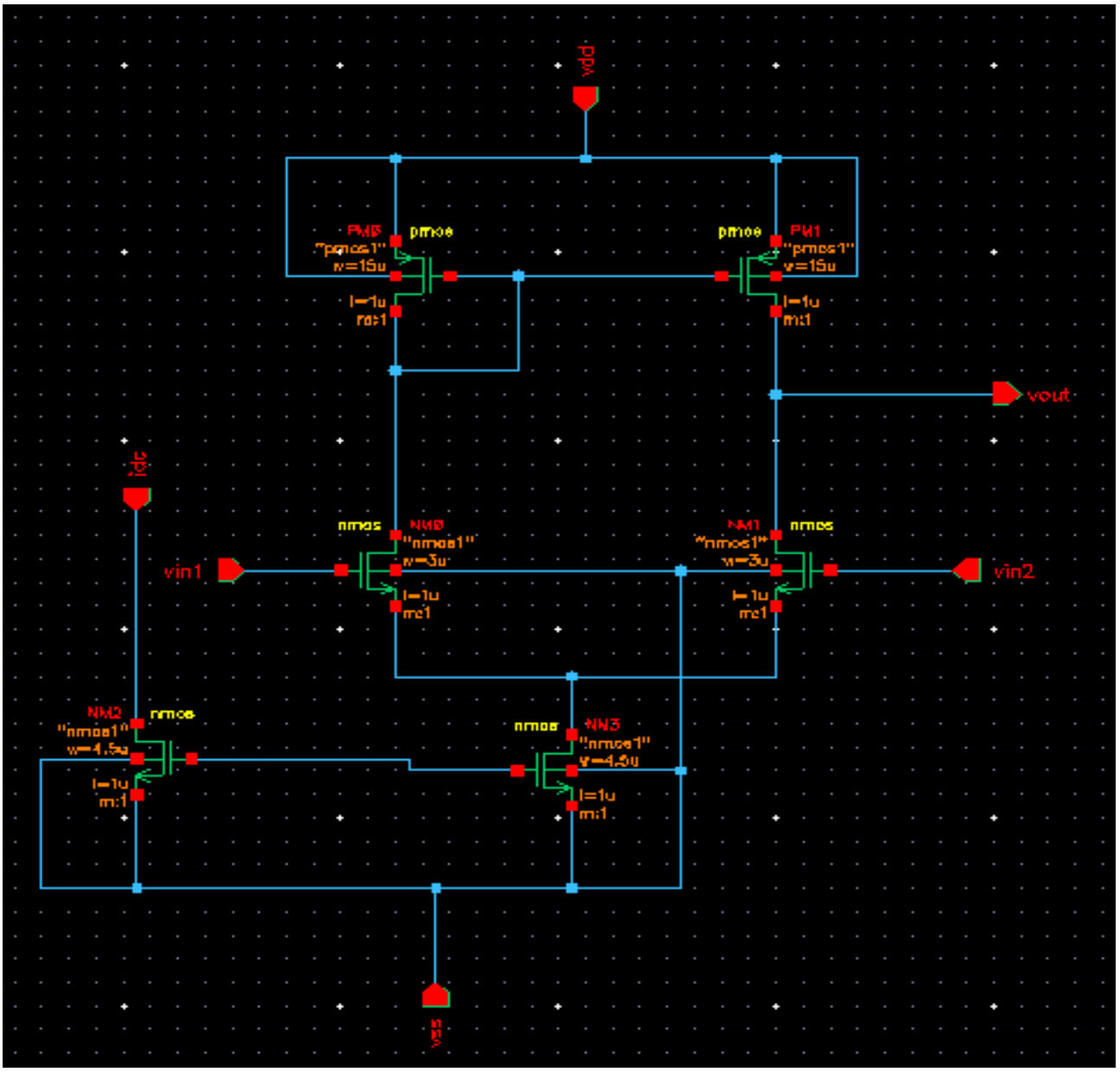
- To set up and run simulations on the Differential Amplifier Test design.
- In this section, we will run the simulation for Differential Amplifier and plot the Transient, DC and AC characteristics.
- Anayzing Gain, Bandwidth and CMRR by performing schematic simulations.

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 (Vin+ - Vin-) by some constant factor Ad, 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 (Vin+ + Vin-)/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
gpdk180	nmos	Model name = nmos1, (NM0, NM1); W= 3u; L=1u
gpdk180	nmos	Model name = nmos1, (NM2, NM3); W= 4.5u; L=1u
gpdk180	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/ouput/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:

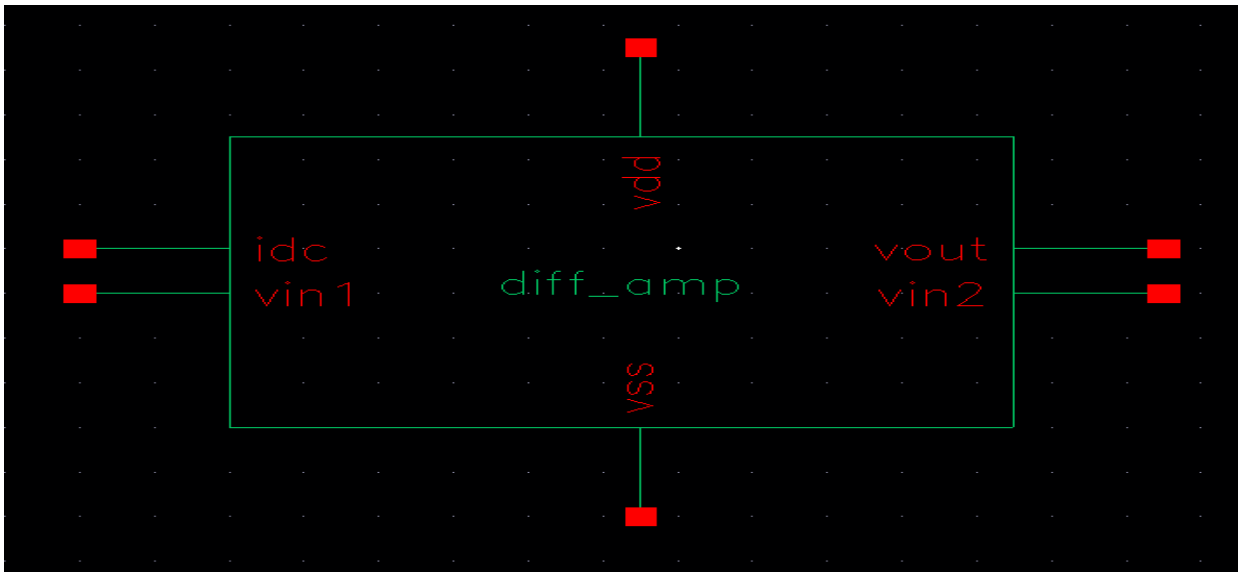
Objective: To create a symbol for the Differential Amplifier

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.

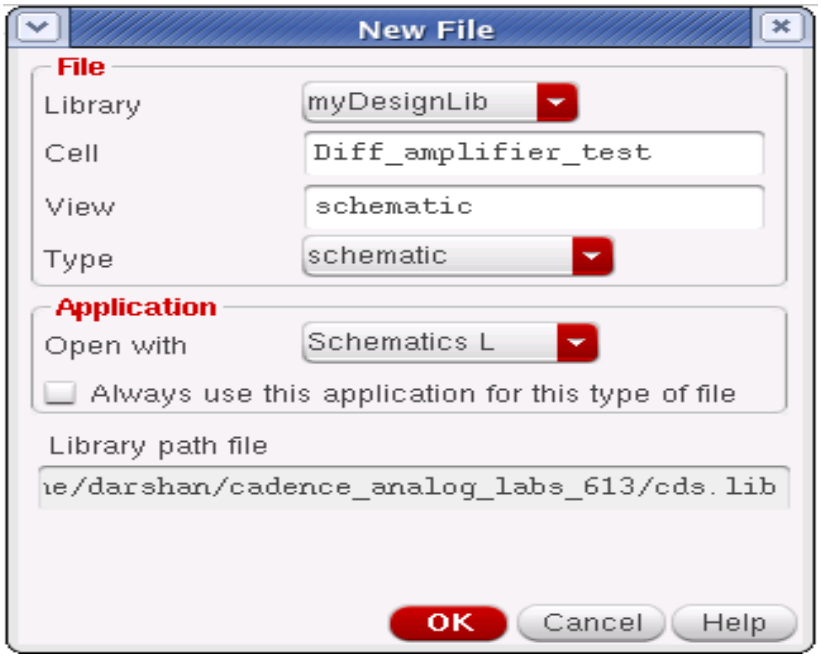
2: Building the Diff_amplifier_test Design

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

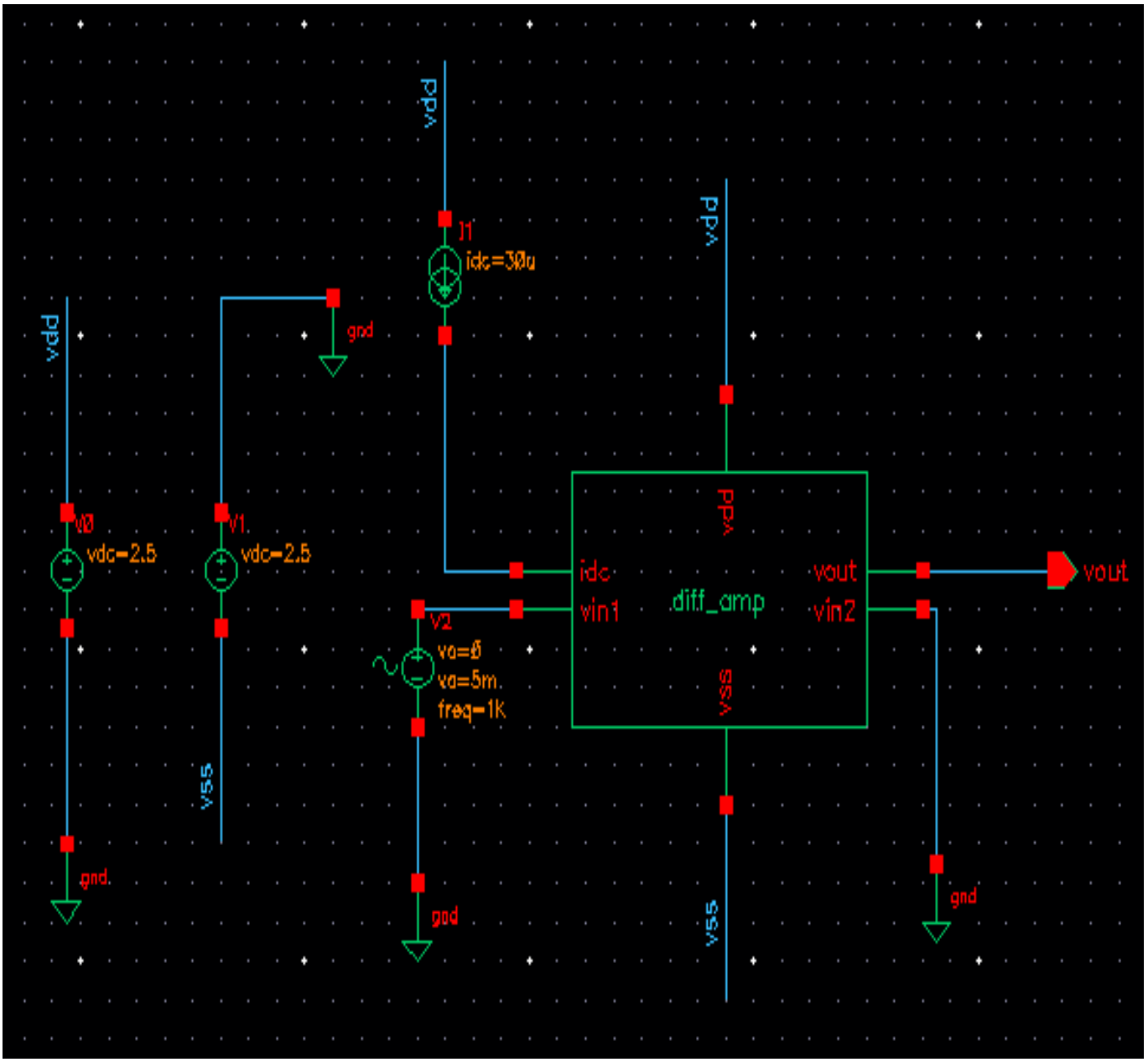
3. Click the Wire (narrow) icon and wire your schematic.

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 **gpdk180.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.**

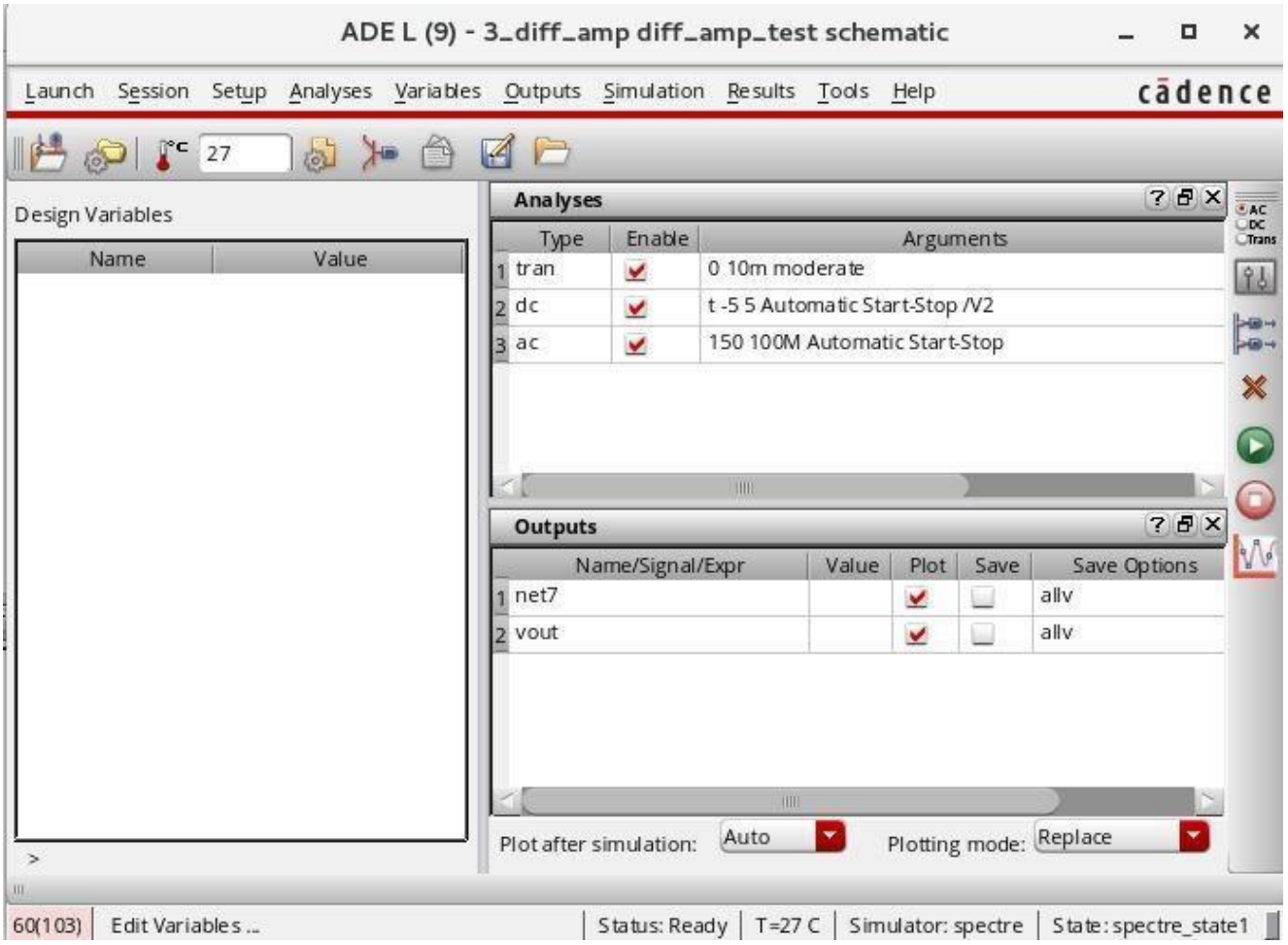
5. Click **OK** in the Choosing Analyses Form

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:

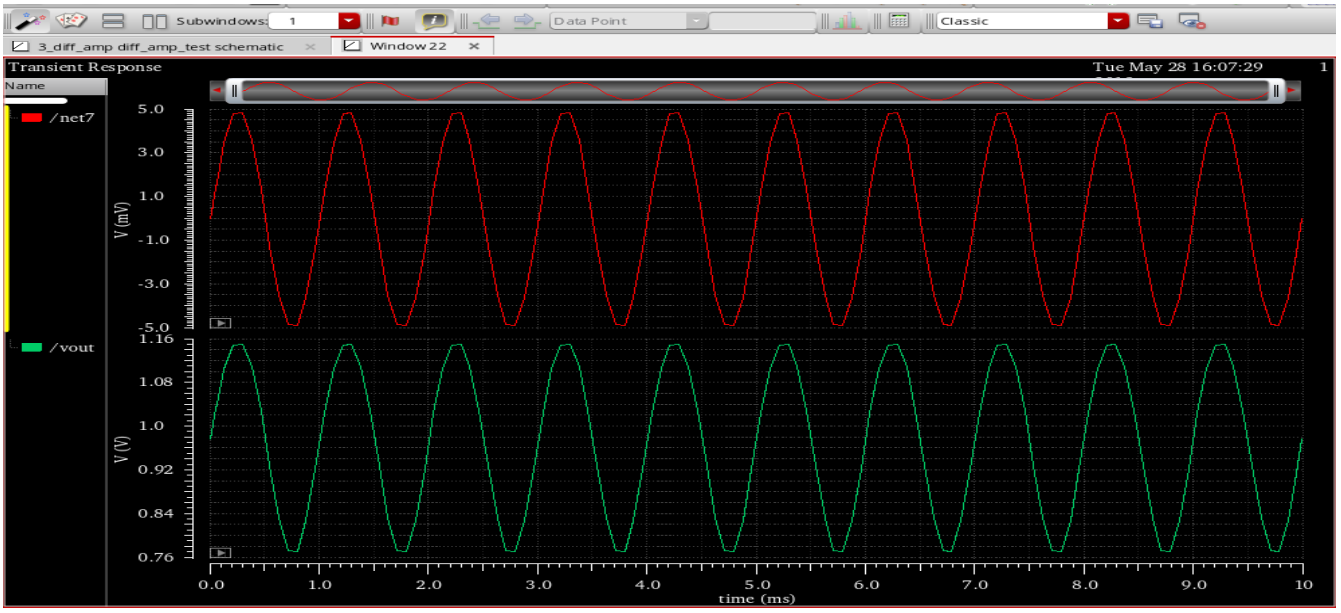


Running the Simulation:

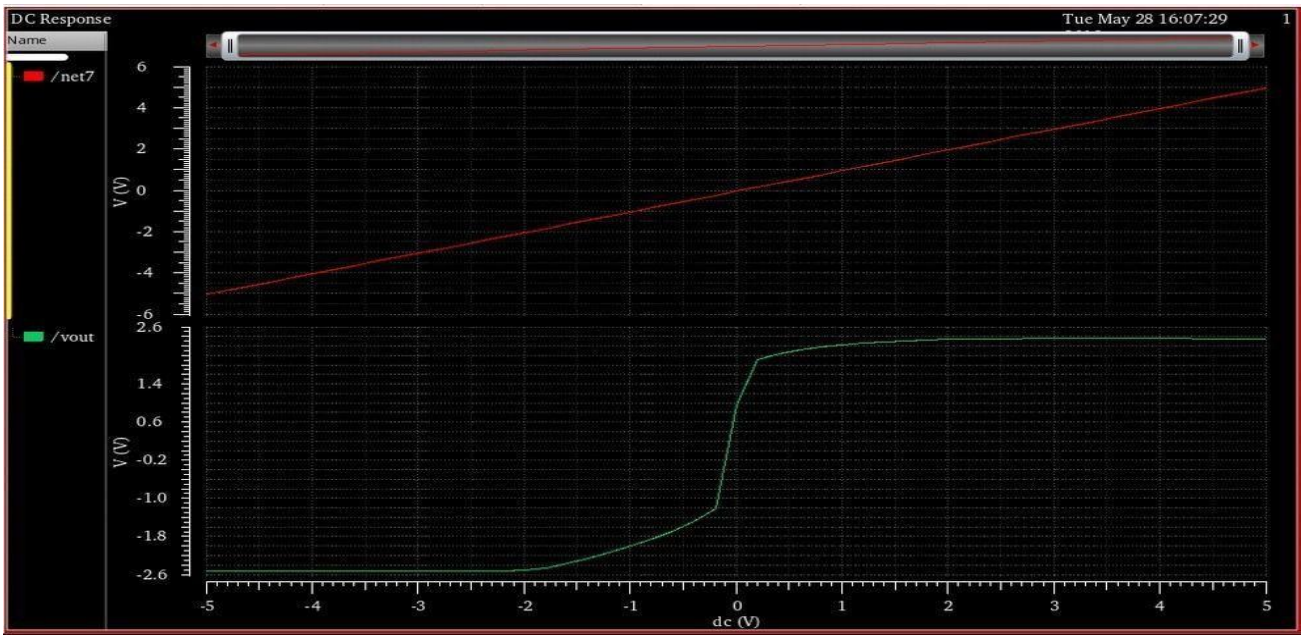
1. Execute **Simulation – Netlist and Run** in the simulation window to start the simulation, this will create the netlist as well as run the simulation.

2. When simulation finishes, the Transient, DC and AC plots automatically will be popped up along with netlist.

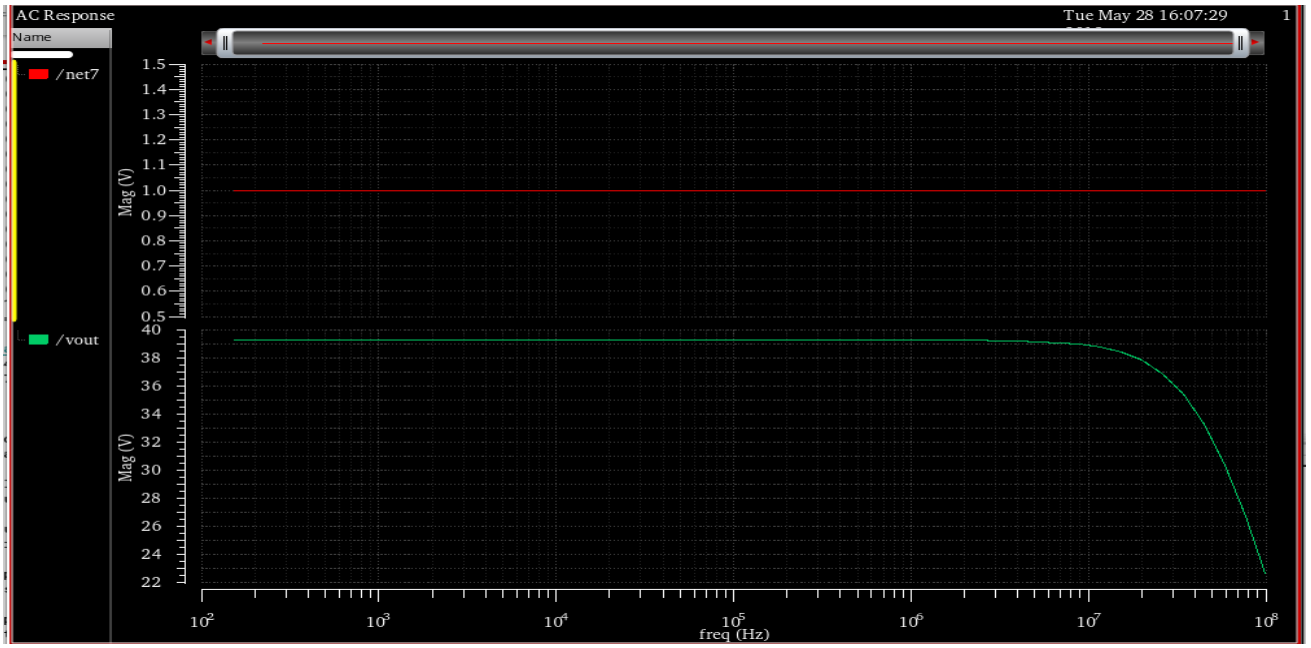
Transient Response:



DC Response:



AC Response:

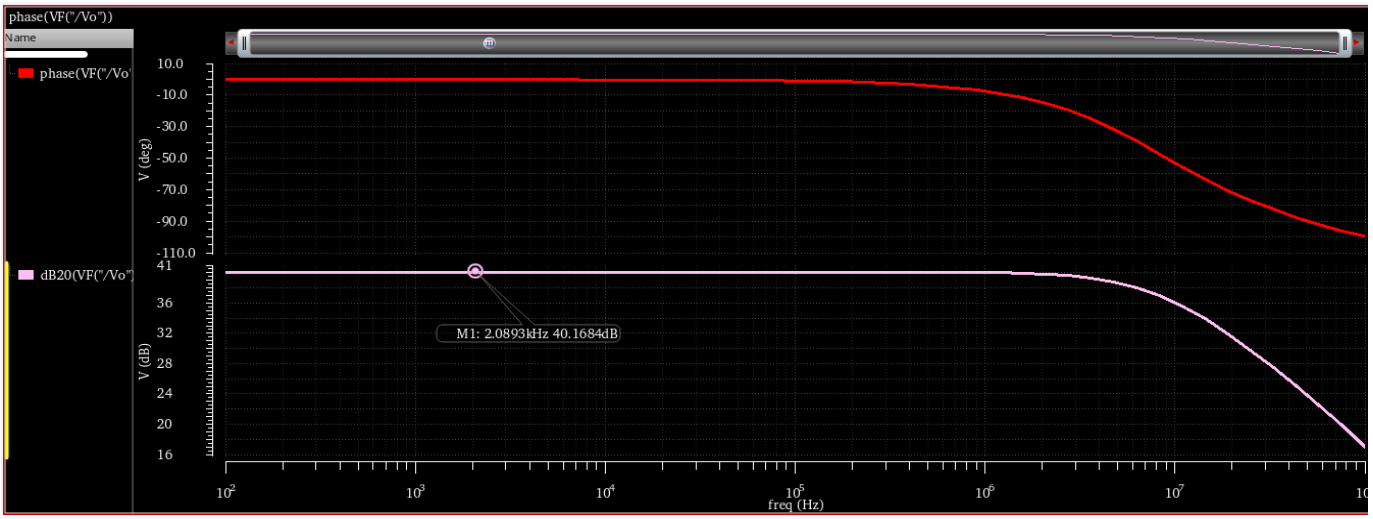


4: Gain Calculation

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

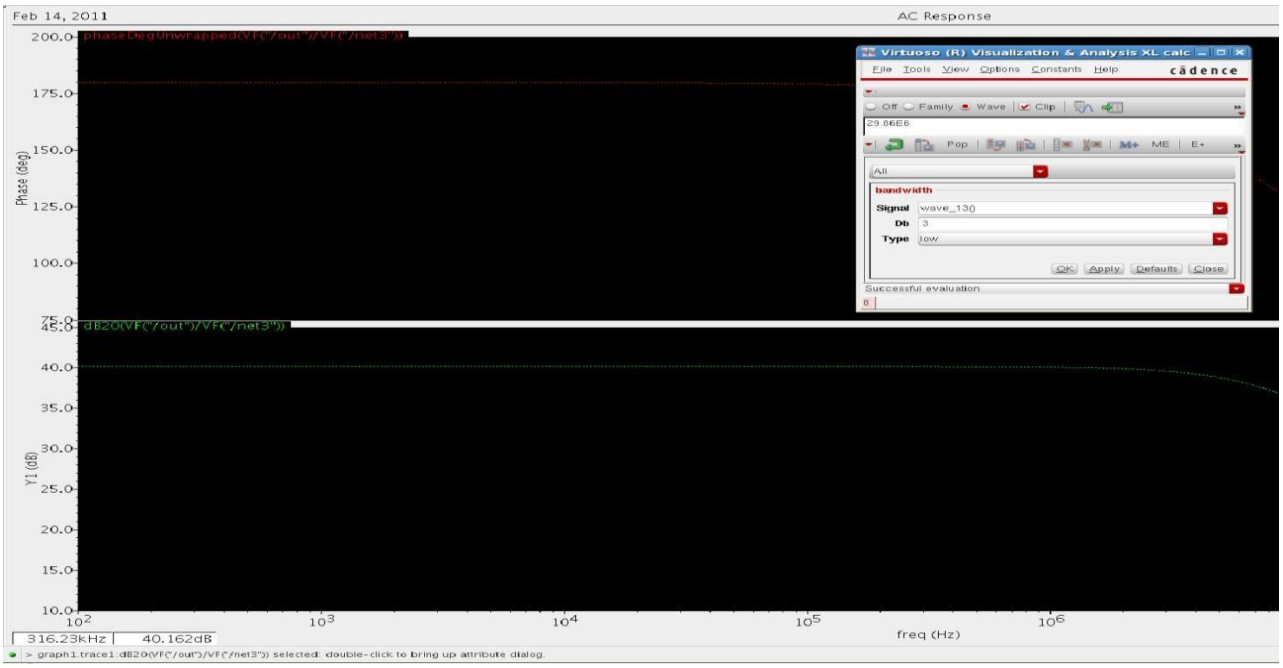
1. Schematic for Differential Gain:

Differential mode Gain (Ad):



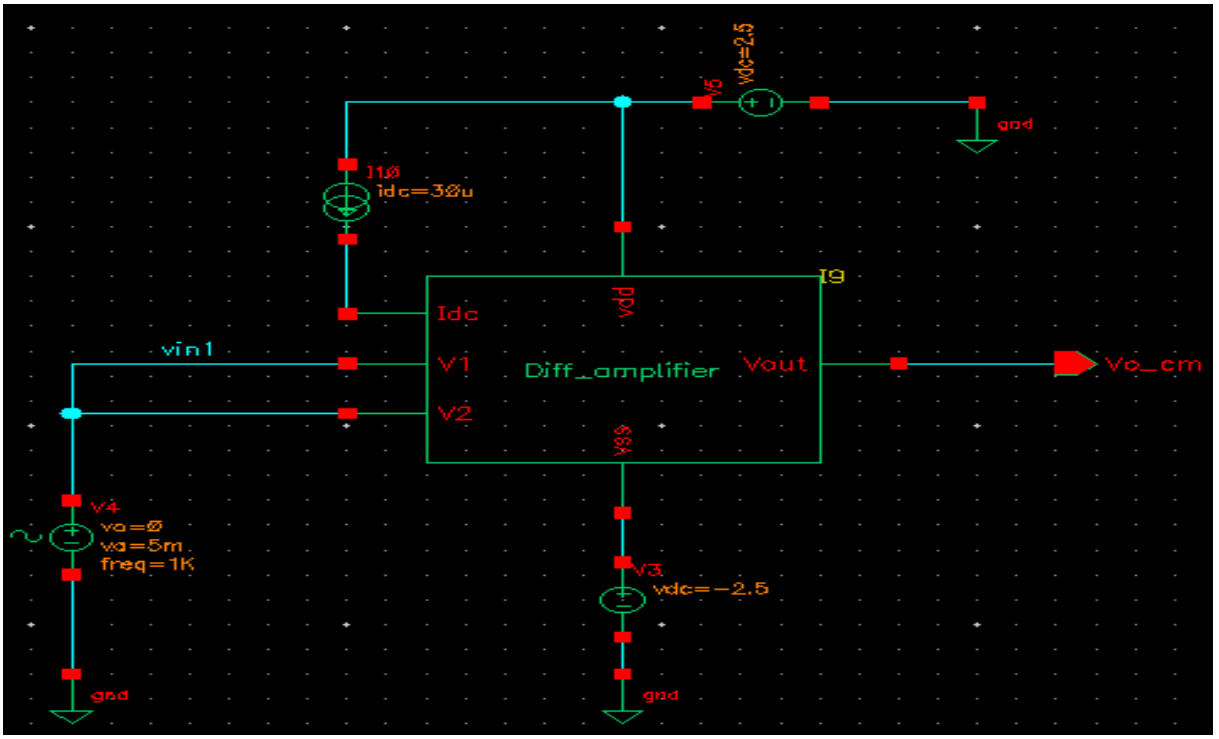
5: Bandwidth Calculation

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 -

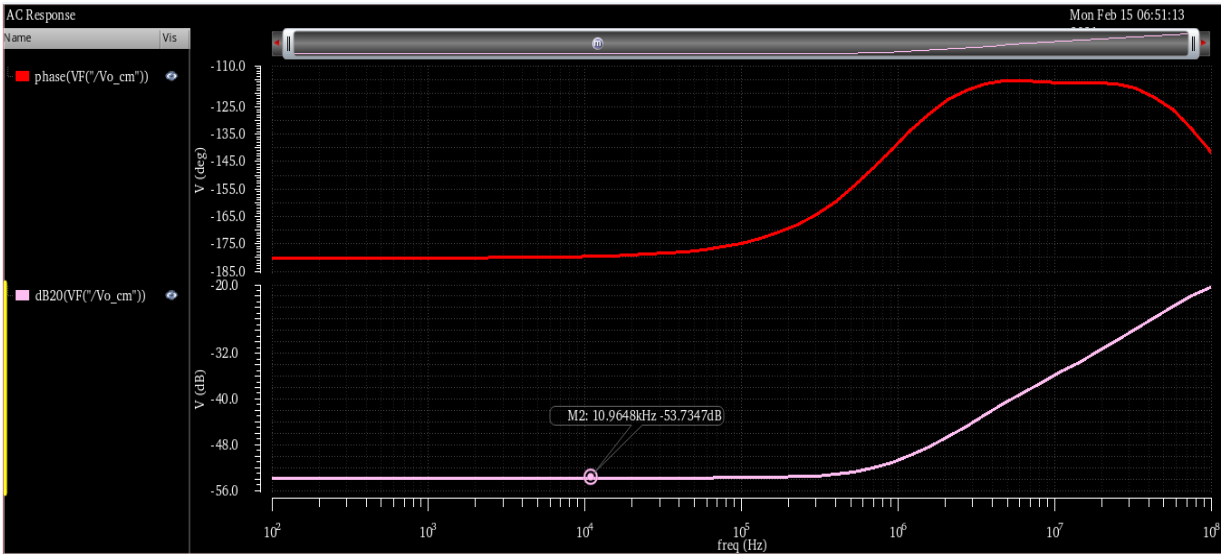


To Calculate the Gain of Common Mode Pair:- Configure the Differential pair schematic as shown below

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

$$CMRR = A_d - (-A_c)$$

Result:

Thus the set up and run simulations on the Differential Amplifier verified successfully.