DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGINEERING

VLSI LABORATORY MANUAL

VI Semester
(18EC6DLVLS)
Autonomous Course 2019

HOD : **Dr. T.C.Manjunath**

In-charge: **Prof.Madhura.R**

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

Dayananda Sagar College of Engineering
Shavige Malleshwara Hills, Kumaraswamy Layout, Banashankari, Bangalore-560078, Karnataka
Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789
Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu
(An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified) (Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade)

Dayananda Sagar College of Engineering

Dept. of E & C Engg.

Name of the Laboratory	**:**	**VLSI LAB**
Semester/Year	**:**	VI/ 2021 (Autonomous)
No. of Students/Batch	**:**	20
Major Equipment's	**:**	Cadence (20 user licenses) HP Server Computers Projectors HP LaserJet
Area in square meters	**:**	68 Sq. mts
Location	**:**	Room No. 17219
Total Cost of Lab	**:**	₹. 45,00,000 .00
Staff In-Charge	**:**	Prof. Madhura R
Instructor	**:**	**Mr.Naveen**.
HOD	**:**	Dr. T.C. Manjunath, Ph.D. (IIT Bombay)

About the college & the department

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programs under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of "Dayananda Sagar Institutions" (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centers in different branches of Engineering catering to research scholars for obtaining Ph.D. under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.'s & 30^{+} staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 12 sophisticated labs & dozen class room, R & D center, etc… The VLSI lab is equipped with high end cadence tool with 20 users & caters both for the UG & PG courses.

Vision & Mission of the Institute

Vision of the Institute

To impart quality technical education with a focus on Research and Innovation emphasizing on Development of Sustainable and Inclusive Technology for the benefit of society.

Mission of the Institute

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence.
- To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.
- To train the students to the changing technical scenario and make them to understand the importance of Sustainable and Inclusive technologies.

Vision & Mission of the Department

Vision:

To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional success.

Mission:

- Offering quality education in Electronics and Communication Engineering with effective teaching learning process in multidisciplinary environment.
- Training the students to take-up projects in emerging technologies and work with team spirit.
- To imbibe professional ethics, development of skills and research culture for better placement opportunities.

Program Educational Objectives

After four years, the students will be

PEO1: ready to apply the state-of-art technology in industry and meeting the societal needs with knowledge of Electronics and Communication Engineering due to strong academic culture.

PEO2: competent in technical and soft skills to be employed with capability of working in multidisciplinary domains.

PEO3: professionals, capable of pursuing higher studies in technical, research or management programs.

Program Specific Outcomes

Students will be able to

PSO1 : Design, develop and integrate electronic circuits and systems using current practices and standards.

PSO2 : Apply knowledge of hardware and software in designing Embedded and Communication systems.

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge.

PO2: Problem Analysis.

PO3: Design Development of solutions.

PO4: Conduct investigations on complex problems.

PO5: Modern tool usage.

PO6: The Engineer and Society.

PO7: Environment and Sustainability.

PO8: Ethics.

PO9: Individual and team work.

PO10: Communication.

PO11: Project management and finance.

PO12: Life Long learning.

VLSI LABORATORY (SYLLABUS)

VI SEMESTER B.E. (E & C)

AUTONOMOUS COURSE

Course code : 18EC6DLVLS Credits: 2 & 2 hrs per lab

L : P : T : S : 1 : 2 : 0 : 0 CIE Marks: 50

Exam Hours : 3 SEE Marks: 50

Hrs / Week : 3

Course Objectives:

1. To get a practical experience in analysis of the MOSFET circuits.
2. To get a practical experience in design of the MOSFET circuits.
3. To know the art of debugging the digital circuits using the V-codes.
4. To know the use of tools in the design of CMOS circuits.
5. To get acquainted with the VLSI verification techniques.
6. To investigate the layout design.

Syllabus:

Module	Expt No.	Content of the Lab Module with Expt. Nos.	Hours	COs
Digital Design				
Part A	1	Inverter and Buffer design and verification.	03	CO1 CO3
	2	Transmission Gate and basic/universal gates design and verification	03	CO1 CO2 CO3
	3	Design and verification of Flip flop -RS, D, JK, MS, T, etc…	03	CO1 CO3 CO4
	4	Design and verification of Serial& Parallel adder.	03	CO1 CO3 CO4
	5	Design and verification of 4-bit counter 1. Synchronous 2. Asynchronous counter	03	CO1 CO3 CO4

Analog Design				
Part B	6	Design an Inverter with given specifications.	03	CO4 CO5 CO6
	7	Design of Common source amplifier.	03	CO4 CO5 CO6
	8	Design of Common Drain amplifier.	03	CO4 CO5 CO6
	9	Design of Single Stage differential amplifier.	03	CO4 CO5 CO6
	10	Design of an op-amp with given specification.	03	CO4 CO5 CO6
	11	Design a 4 bit R-2R based DAC for the given specification.	03	CO4 CO5 CO6

Course Outcomes:

At the end of the course, student will be able to

CO1	Understand digital design flow using NCSIM and debug digital circuit design using Verilog Code.
CO2	Analyze and verification by writing Test Benches using NCSIM and further model subsystem blocks using Verilog code.
CO3	Perform the initial timing verification of Verilog code.
CO4	Design front-end digital/ analog circuit using industry standard cadence tool.
CO5	Apply the knowledge of amplifier design and analyze DC, AC and transient characteristics
CO6	Design and analyze layout for the circuits using back-end tool and verify the DRC and ERC and check for LVS, Extract RC.

CO-PO mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	2	2	3	-	-	1	2	-	1	2
CO2	3	3	2	2	3	-	-	1	2	-	1	2
CO3	3	3	2	2	3	-	-	1	2	-	1	2
CO4	3	3	2	2	3	-	-	1	2	-	1	2
CO5	3	3	2	2	3	-	-	1	2	-	1	2
CO6	3	3	2	2	3	-	-	1	2	-	1	2

DO's

- Students should follow the dress code of the laboratory compulsorily.
- Keep your belongings in the corner of the laboratory.
- Students have to enter their name, USN, time in/out and signature in the log register maintained in the laboratory.
- Students are required to enter components in the components register related to the experiment and handle the equipment's smoothly.
- Check the components, range and polarities of the meters before connecting to the circuit.
- Come prepare for the experiment and background theory.
- Before connecting to the circuit refer the designed circuit diagram properly. Debug the circuit for proper output.
- Students should maintain discipline in the laboratory and keep the laboratory clean and tidy.
- Observation book and Record book should be complete in all respects and get it corrected by the staff members.
- Clarify the doubts with staff members and instructors.
- Experiment once conducted, in the next lab, the entire record should be complete in all respects, else the student will lose the marks.
- For programming lab, show the output to the concerned faculty.
- All the students should come to LAB on time with proper dress code and identity cards
- Keep your belongings in the corner of laboratory.
- Students have to enter their name, USN, time-in/out and signature in the log register maintained in the laboratory.
- All the students should submit their records before the commencement of Laboratory experiments.
- Students should come to the lab well prepared for the experiments which are to be performed in that particular session.
- Students are asked to do the experiments on their own and should not waste their precious time by talking, roaming and sitting idle in the labs.
- Observation book and record book should be complete in all respects and it should be corrected by the staff member.

- Before leaving the laboratory students should arrange their chairs and leave in orderly manner after completion of their scheduled time.
- Prior permission to be taken, if for some reasons, they cannot attend lab.
- Immediately report any sparks/ accidents/ injuries/ any other untoward incident to the faculty /instructor.
- In case of an emergency or accident, follow the safety procedure.
- Switch OFF the power supply after completion of experiment.

DONT's

- Do not switch on the power supply before verification of the connected circuits by concerned staff.
- Do not feed higher voltages than rated to the device.
- Do not upload, delete or alter any software on the laboratory PC's.
- Do not write or mark on the equipment's.
- Usage of mobile phone is strictly prohibited.
- Ragging is punishable.
- If student damages the equipment or any of the component in the lab, then he / she is solely responsible for replacing that entire amount of the equipment or else, replace the equipment.
- The use of mobile/ any other personal electronic gadgets is prohibited in the laboratory.
- Do not make noise in the Laboratory & do not sit on experiment table.
- Do not make loose connections and avoid overlapping of wires.
- Don't switch on power supply without prior permission from the concerned staff.
- Never point/touch the CRO/Monitor screen with the tip of the open.

VLSI LABORATORY (ECL68)

I - CYCLE

1. Inverter and Buffer design and verification.
2. Transmission Gate and basic/universal gates design and verification
3. Design and verification of Flip flop -RS, D, JK, MS, T, etc…
4. Design and verification of Serial & Parallel adder.
5. Design and verification of 4-bit counter : Synchronous and Asynchronous counter

II – CYCLE

6. Design an Inverter with given specifications.
7. Design of Common source amplifier.
8. Design of Common Drain amplifier.
9. Design of Single Stage differential amplifier.
10. Design of an op-amp with given specification.
11. Design a 4 bit R-2R based DAC for the given specification.

General Introduction

Cadence:
Cadence is an Electronic Design Automation (EDA) environment in which different applications and tools can are integrated together. This allows all the stages of IC design and verification to be done in a single environment. The different tools are supported by different fabrication technologies allowing for customization of the Cadence environment to fit the particular technology.

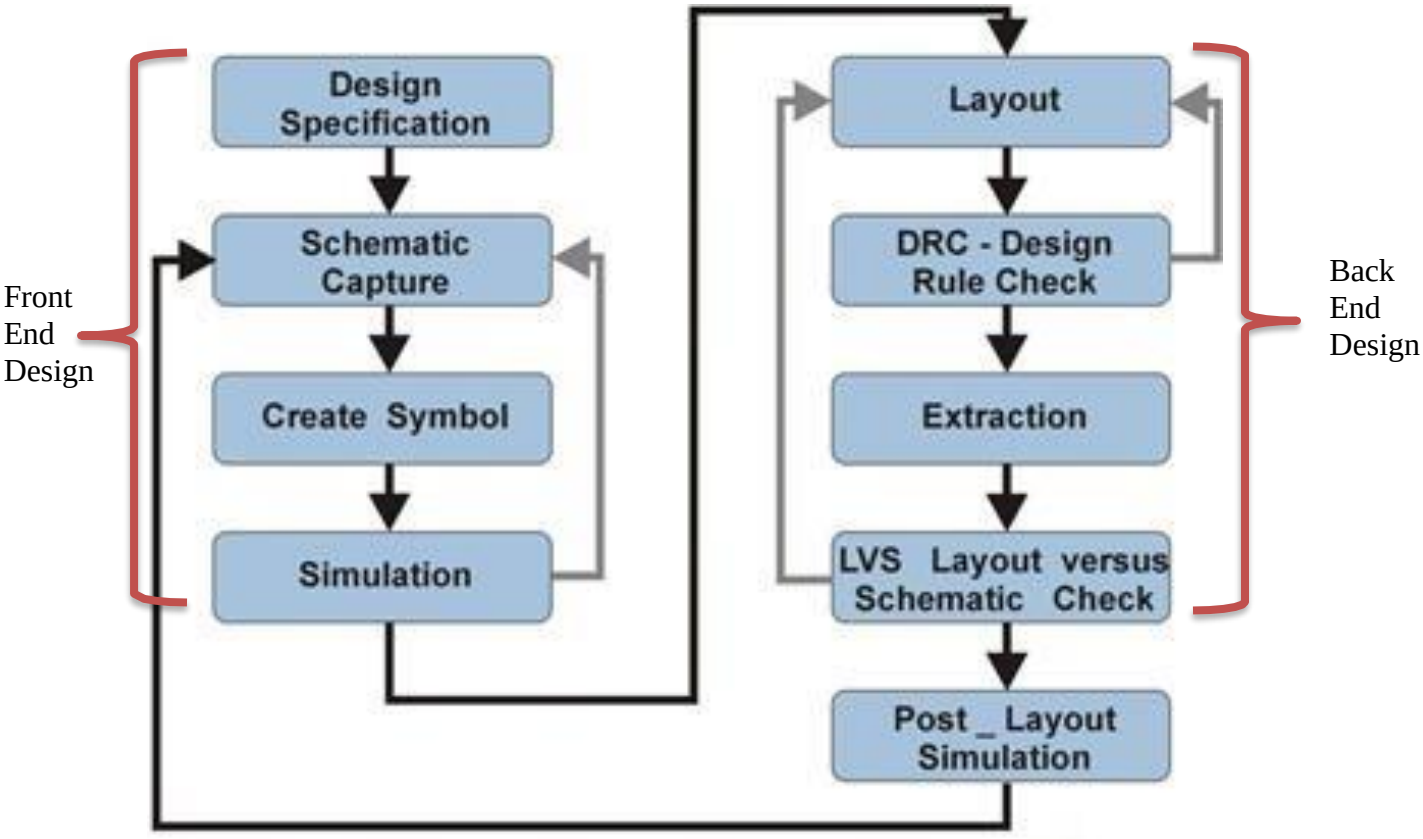
Linux:
Linux is an operating system that is a multi-user, multi-tasking system that can be used on servers, desktops and laptops. Linux was originally developed at Bell Labs in 1969. Linux is used to access cadence and manage the files in its libraries.

Basic Linux commands used for running cadence:

A. Command: ls
   Function: lists the files in the current directory
   Example: ls
B. Command: pwd
   Function: tells you what directory you are currently in
   Example: pwd
C. Command: mkdir
   Function: makes a new directory
   Example: mkdir project (makes a new directory called project)
D. Command: cd
   Function: takes into the specified directory
   Example: cd NCSU_AMI06 (takes you into the directory called NCSU_AMI06)
E. Command: cd ..
   Function: takes you back one directory
   Example: cd ..
F. Command: mv
   Function: moves a directory to the specified location
   Example: mv adder adder2
   (moves the directory adder to the current directory and changes the name to adder2)
G. Command: cp
   Function: copies a file
   Example: cp ../multi.
   (copies the file named multi from the previous directory to the current directory)
H. Command: --help
   Function: Tells you the function of a command
   Example: chmod –help
   (tells you the function of the command chmod)

ANALOG DESIGN

Custom IC Design Flow



Tools used:

IC616
ASSURA410
INCISIV131
RC142
EDI142
MMSIM131

Procedure to run the simulation using NCSIM

Example for SERIAL ADDER

Setup the design environment

1. **Create cds.lib and make the following entries**
   Define sadder_lib ./sadder.lib

2. **Create logical library by the following directories**
   Mkdir sadder.lib

3. **Create variable library and make the following entry**
   Define WORK sadder_lib

4. **Compilation process**
   - ncvlog sadder.v –messages
   - ncvlog sadder_test.v –mess

5. **Elaborate process**
   ncelab serial_adder_t –access +rwc -mess

6. **Simulate the Design**
   ncsim serial_adder_t –gui

When the Design Browser window pops up

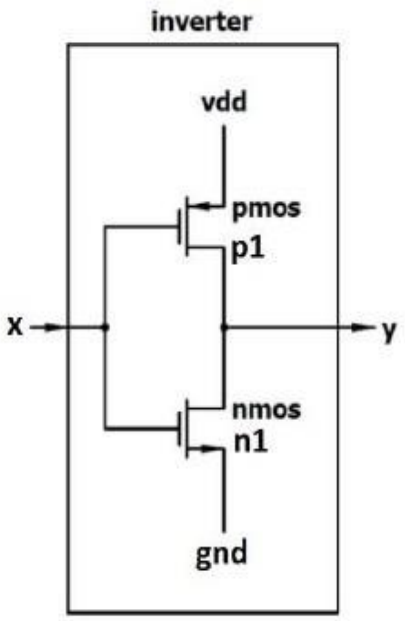
1. Select the serial_adder_t instance and click SEND the SELECTED SIGNALS TO THE TARGETED WAVEFORM WINDOW.
2. In the waveform window press RUN.
3. Now you will be able to visualize the waveforms.

Experiment No. : 1	Date : / / .
Inverter and Buffer design and verification	

INVERTER

Aim: To write Verilog code for an **INVERTER** and their Test Bench for verification

Module:



Truth table:

INPUT	OUTPUT
x	y
0	1
1	0

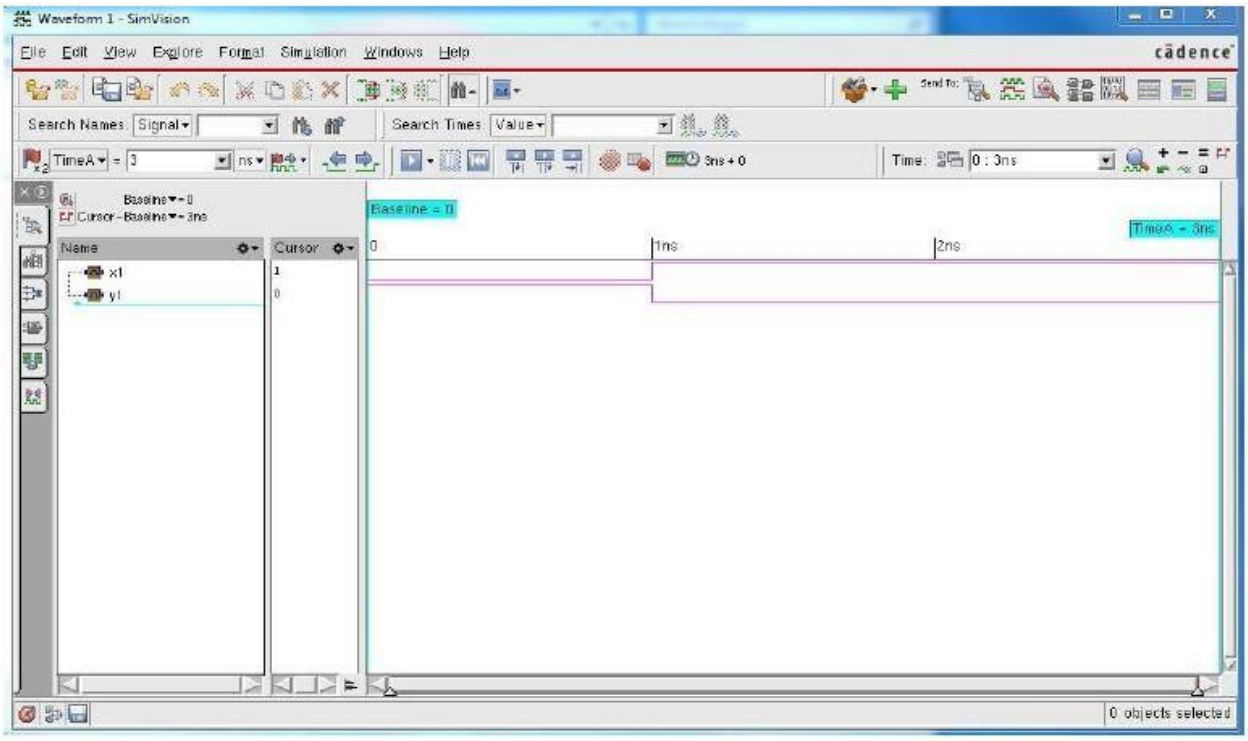
invr.v:

```
module invr(x,y);
input x;
output y;
wire vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (y,vdd,x); /* pmos name(drain, source, gate)*/
nmos n1 (y,gnd,x); /* nmos name(drain, source, gate)*/
endmodule
```

invr_tb.v:

```
module invr_tb;
reg x1;
wire y1;
invr inv1 (x1,y1);
initial begin
x1 = 1'b0; #1
x1 = 1'b1; #2
$finish;
end
endmodule
```

Simulation:



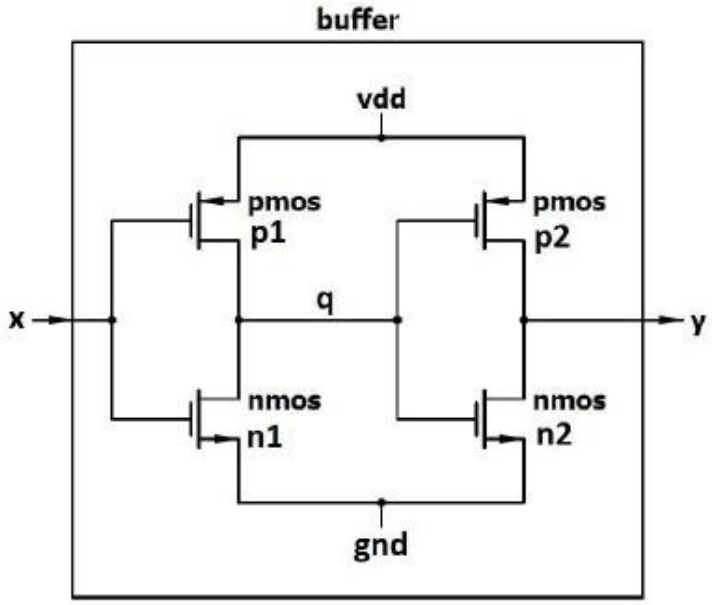
RESULT:

Verilog code for the Inverter is written, verification for the same is done using test bench and the waveform is observed

BUFFER

Aim: To write Verilog code for a **BUFFER** and their Test Bench for verification

Module:



Truth table:

INPUT	OUTPUT
x	y
0	0
1	1

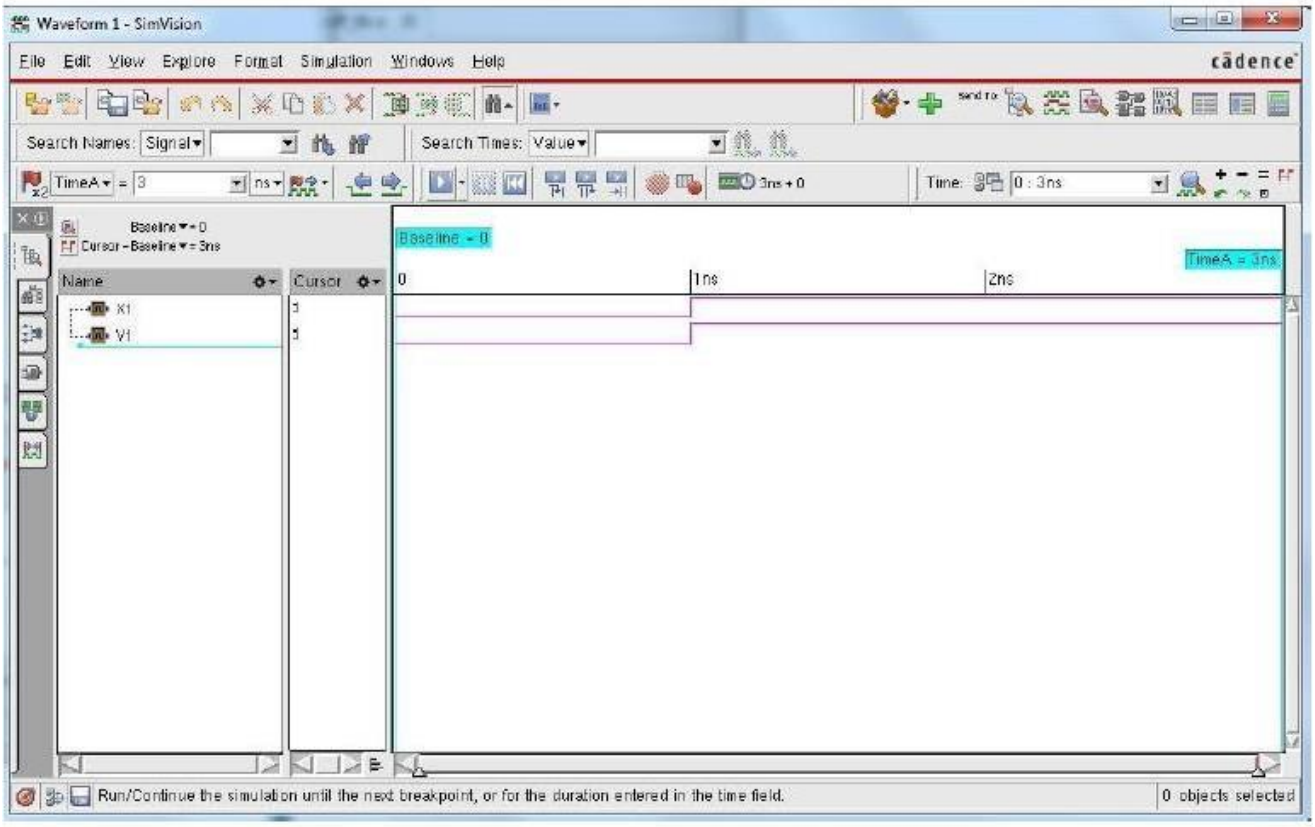
buff.v:

```
module buff(x,y);
input x;
output y;
wire q,vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (q,vdd,x); /* pmos name(drain, source, gate)*/
pmos p2 (y,vdd,q);
nmos n1 (q,gnd,x); /* pmos name(drain, source, gate)*/
nmos n2 (y,gnd,q);
endmodule
```

buff_tb.v:

```
module buff_tb;
reg X1;
wire Y1;
buff buf1 (X1,Y1);
initial begin
X1=1'b0; #1
X1=1'b1; #2
$finish;
end
endmodule
```

Simulation:



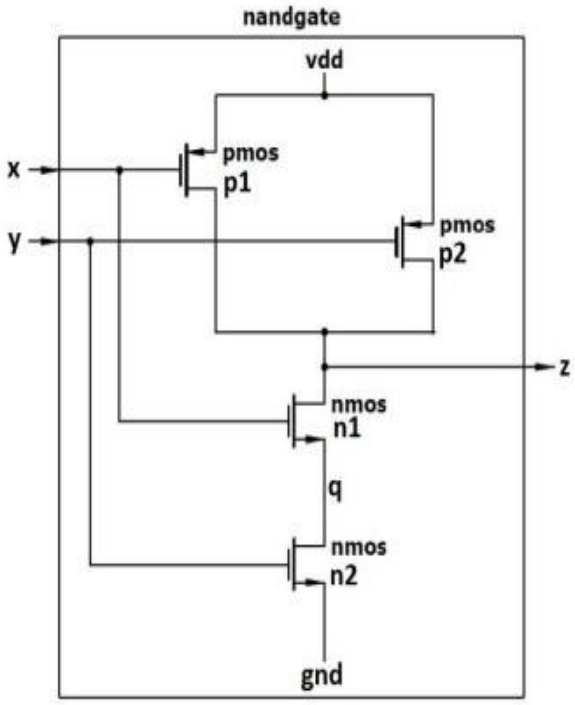
RESULT:

Verilog code for the buffer is written, verification for the same is done using test bench and the waveform is observed

Experiment No. : 2	**Date : / / .**
Transmission Gate and basic/universal gates design and verification	

Aim: To write Verilog code for a **NAND** gate and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUT
x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

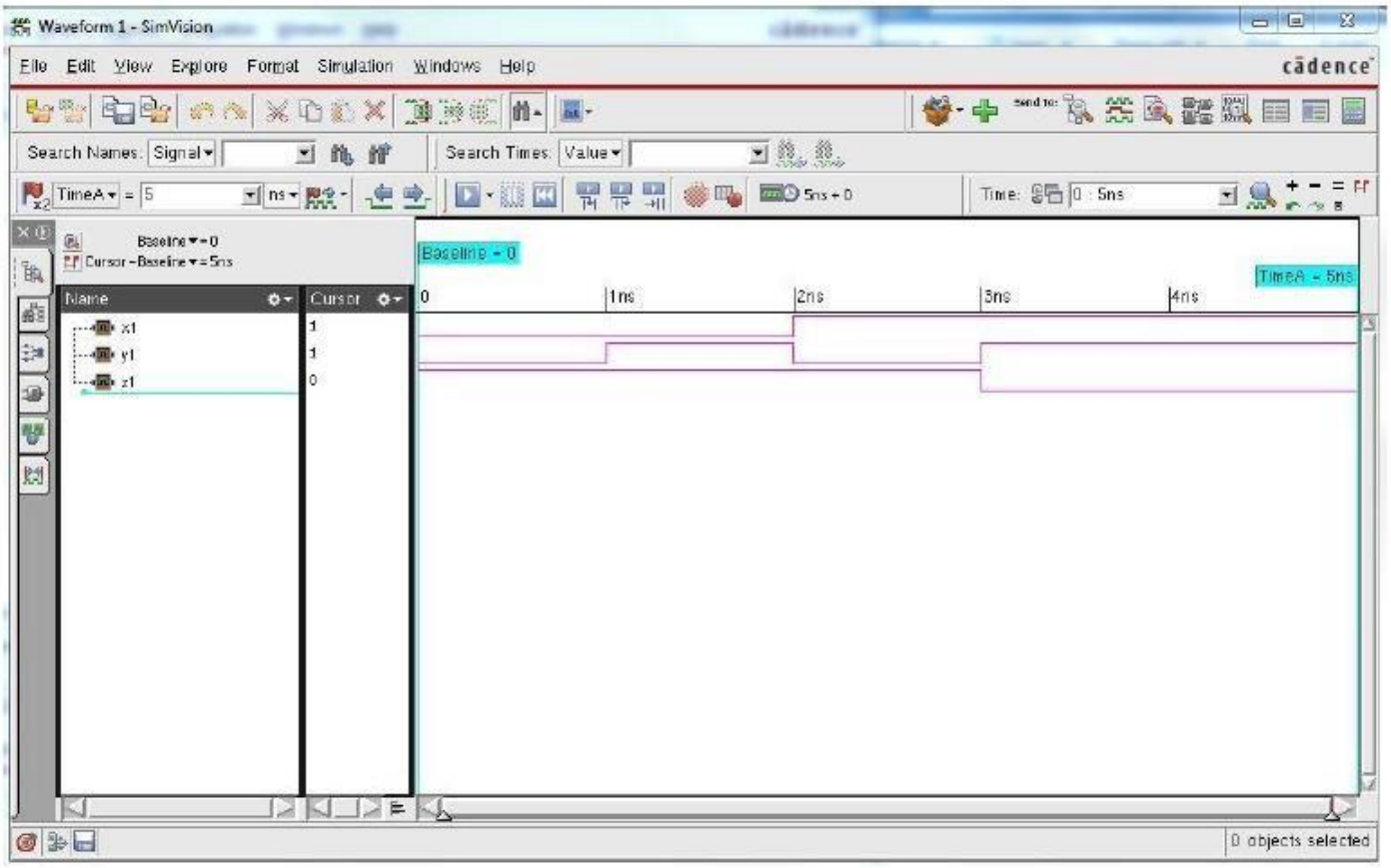
nand.v:

```
module nannd (x,y,z);
input x,y;
output z;
wire q, vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (z,vdd,x); /* pmos name(drain, source, gate)*/
pmos p2 (z,vdd,y);
nmos n1 (z,q,x); /* nmos name(drain, source, gate)*/
nmos n2 (q,gnd,y);
endmodule
```

nand_tb.v:

```
module nannd_tb;
reg x1, y1;
wire z1;
nannd nand1 (x1,y1,z1);
initial begin
x1 = 1'b0; y1 = 1'b0; #1
x1 = 1'b0; y1 = 1'b1; #1
x1 = 1'b1; y1 = 1'b0; #1
x1 = 1'b1; y1 = 1'b1; #2
$finish;
end
  endmodule
```

Simulation:

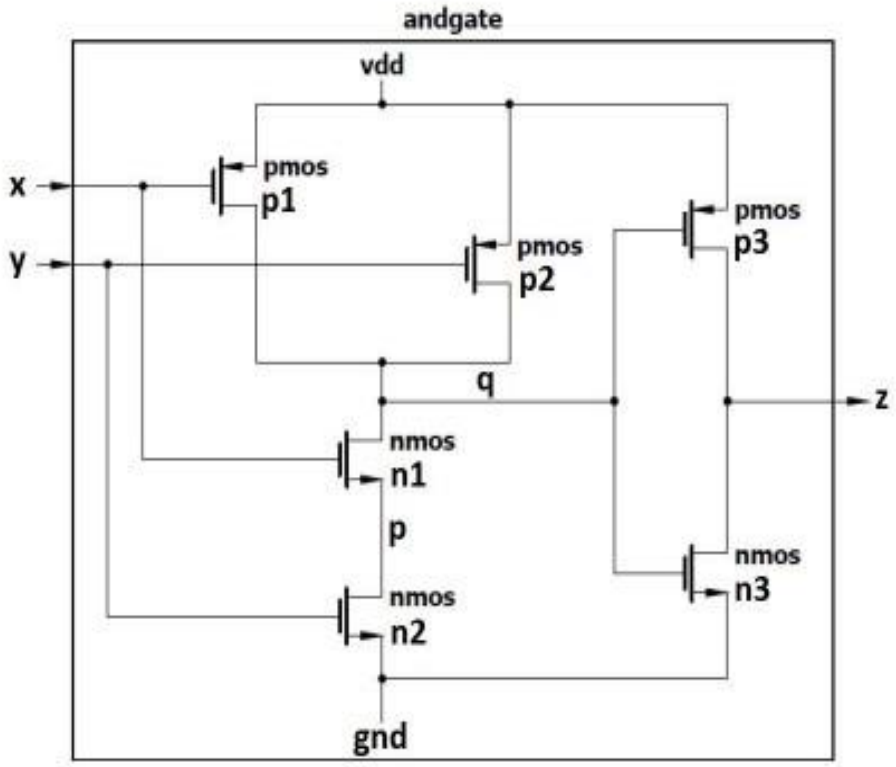


RESULT:

Verilog code for the NAND gate is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for a **AND** gate and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUT
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

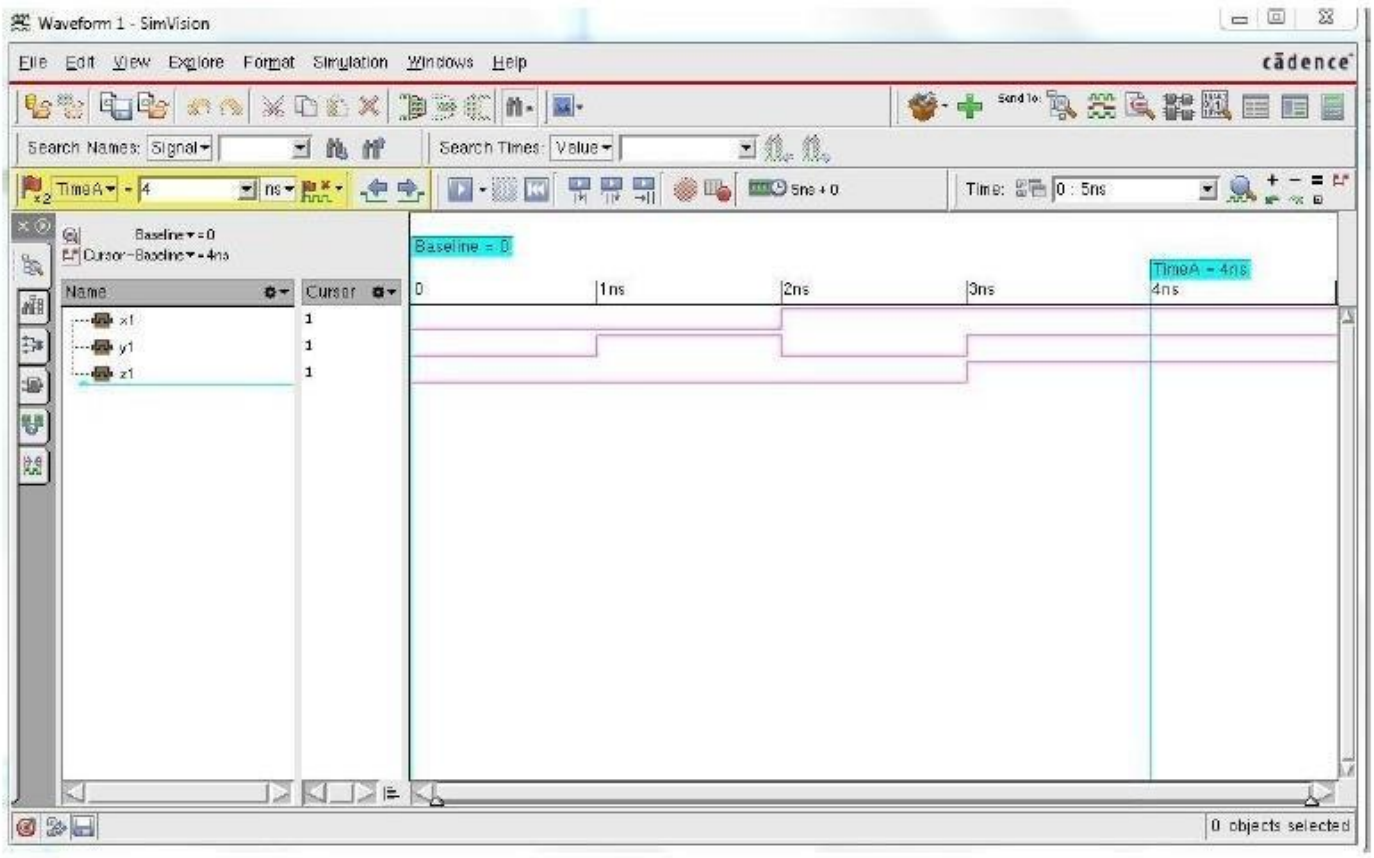
and.v:

```
module andmos (x,y,z);
input x,y;
output z;
wire p, q, vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (q,vdd,x); /* pmos name(drain, source, gate)*/
pmos p2 (q,vdd,y);
nmos n1 (q,p,x); /* nmos name(drain, source, gate)*/
nmos n2 (p,gnd,y);
pmos p3 (z,vdd,q);
nmos n3 (z,gnd,q);
  endmodule
```

and_tb.v:

```
module andmos1;
reg x1, y1;
wire z1;
andmos and1 (x1,y1,z1);
initial begin
x1 = 1'b0; y1 = 1'b0; #1
x1 = 1'b0; y1 = 1'b1; #1
x1 = 1'b1; y1 = 1'b0; #1
x1 = 1'b1; y1 = 1'b1; #2
$finish;
end
endmodule
```

Simulation:

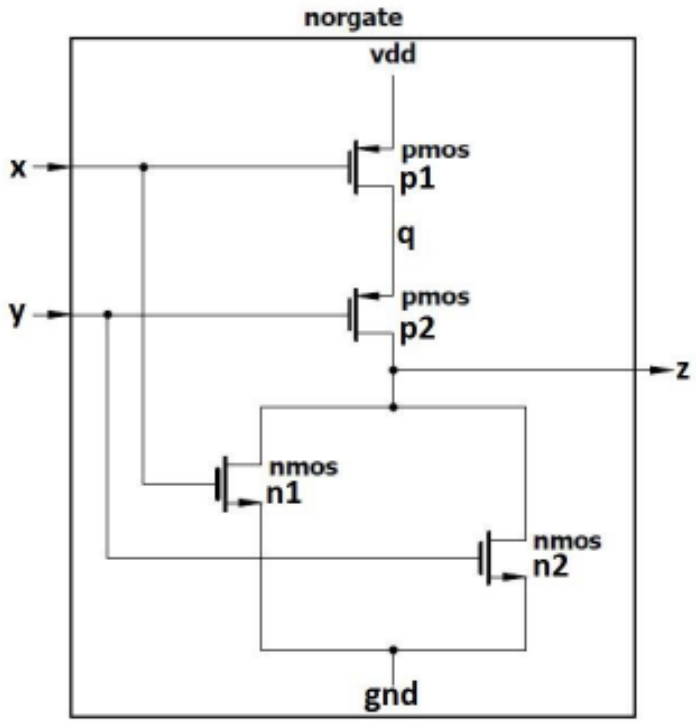


RESULT:

Verilog code for the AND gate is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for a **NOR** gate and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUT
x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

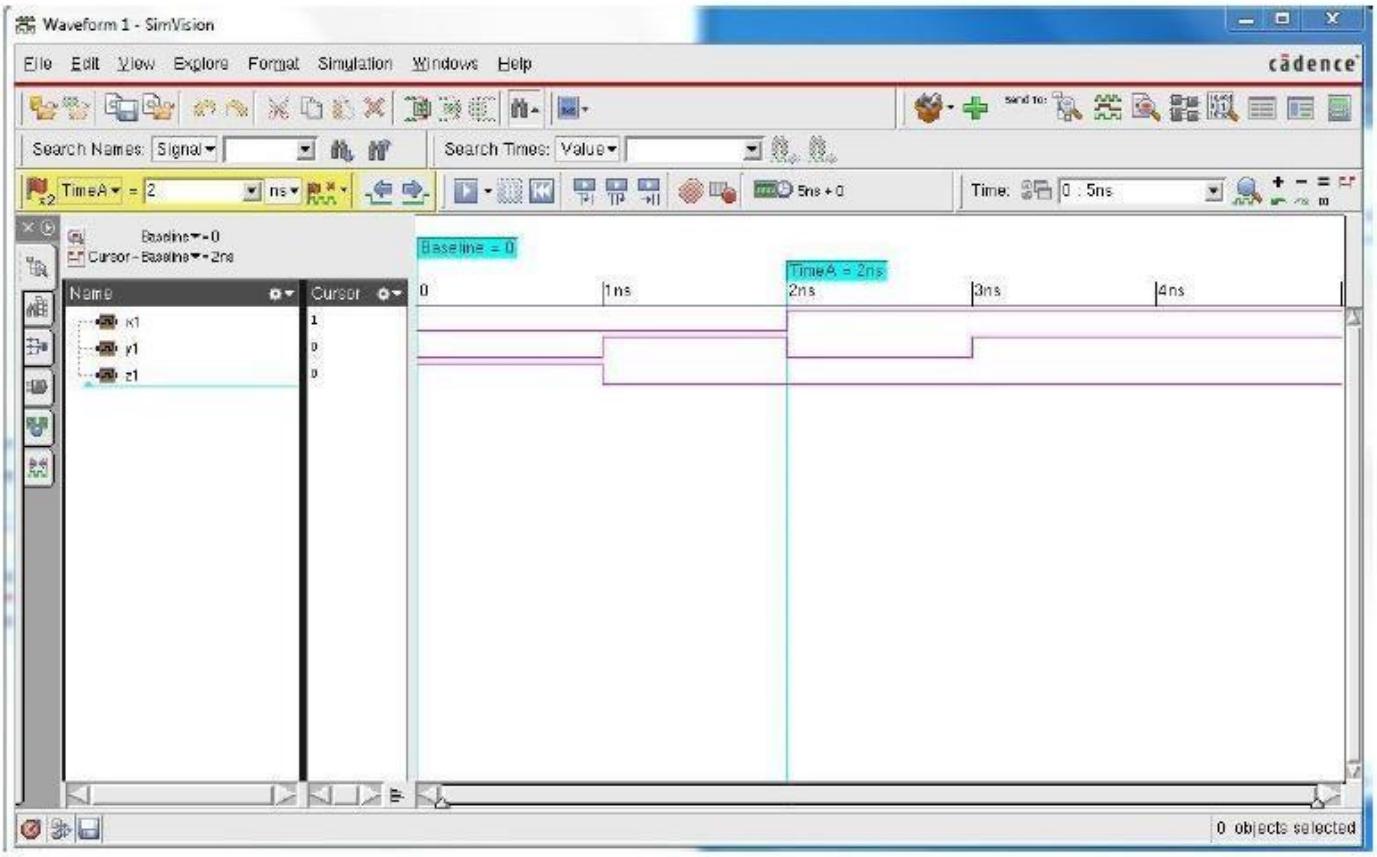
norr.v:

```
module norr (x,y,z);
input x,y;
output z;
wire q, vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (q,vdd,x); /* pmos name(drain, source, gate)*/
pmos p2 (z,q,y);
nmos n1 (z,gnd,x); /* pmos name(drain, source, gate)*/
nmos n2 (z,gnd,y);
  endmodule
```

norr_tb.v:

```
module norr_tb;
reg x1, y1;
wire z1;
norr nor1 (x1,y1,z1);
initial begin
x1 = 1'b0; y1 = 1'b0; #1
x1 = 1'b0; y1 = 1'b1; #1
x1 = 1'b1; y1 = 1'b0; #1
x1 = 1'b1; y1 = 1'b1; #2
$finish;
end
  endmodule
```

Simulation:

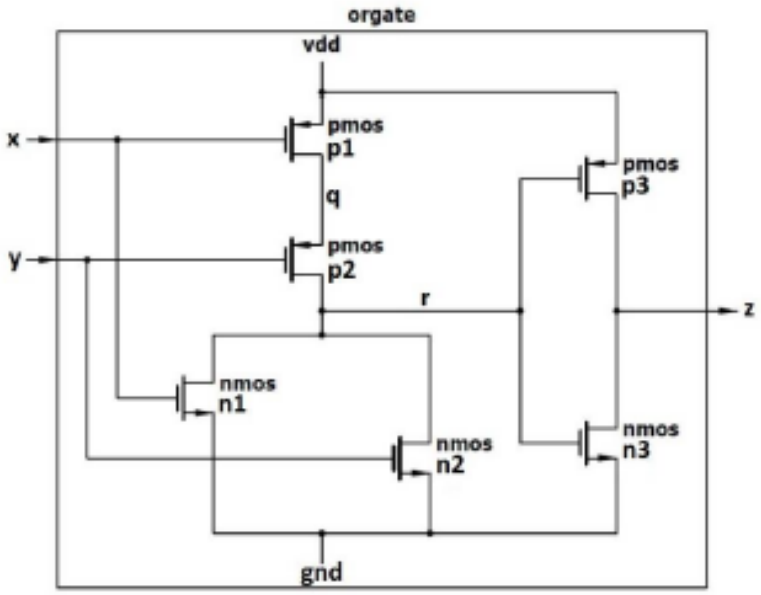


RESULT:

Verilog code for the NOR gate is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for an **OR** gate and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUT
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

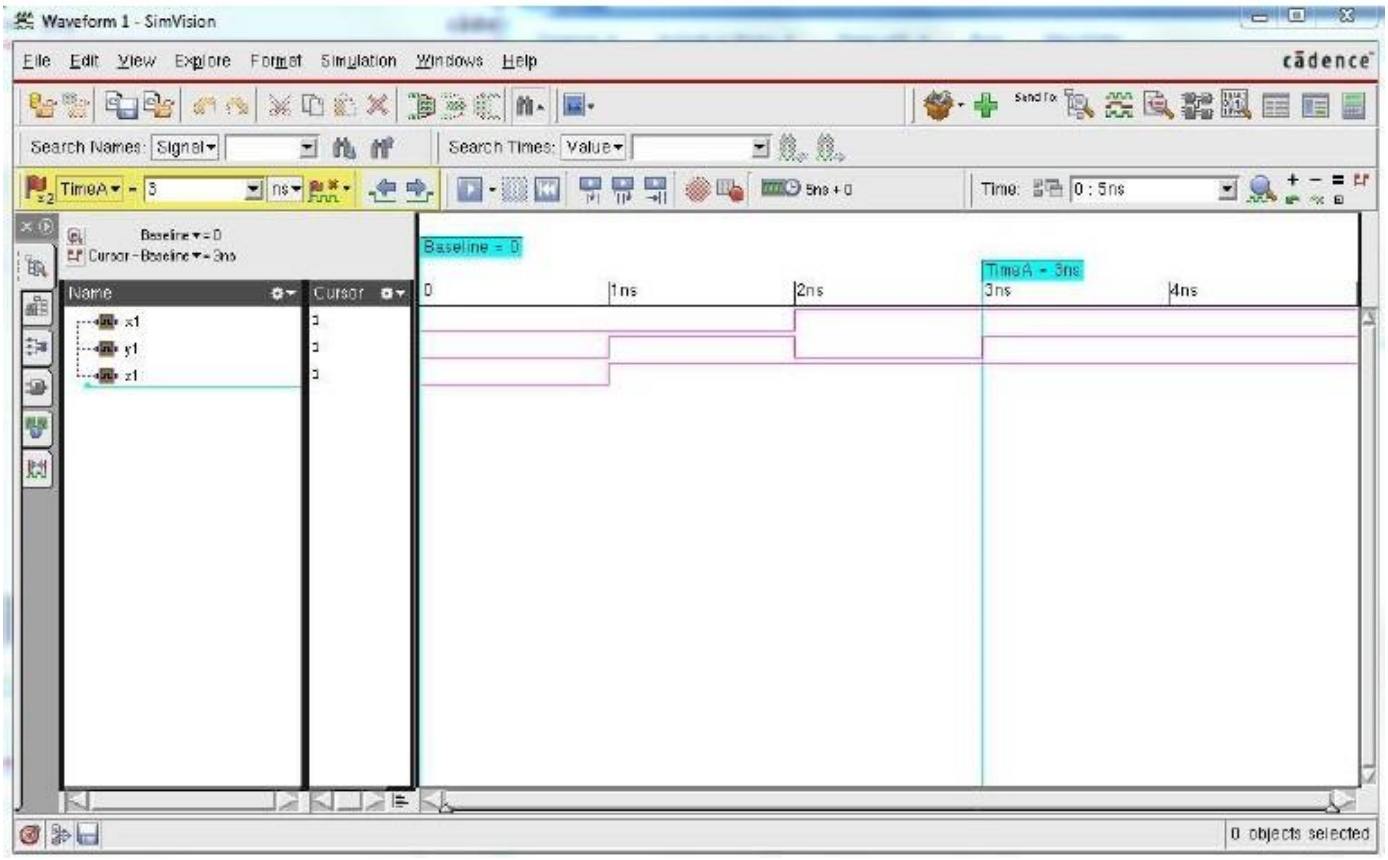
orr.v:

```
module orr (x,y,z);
input x,y;
output z;
wire q, r, vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
pmos p1 (q,vdd,x); /* pmos name(drain, source, gate)*/
pmos p2 (r,q,y);
pmos p3 (z,vdd,r);
nmos n1 (r,gnd,x); /* pmos name(drain, source, gate)*/
nmos n2 (r,gnd,y);
nmos n3 (z,gnd,r);
  endmodule
```

orr_tb.v:

```
module orr_tb;
reg x1, y1;
wire z1;
orr or1 (x1,y1,z1);
initial begin
x1 = 1'b0; y1 = 1'b0; #1
x1 = 1'b0; y1 = 1'b1; #1
x1 = 1'b1; y1 = 1'b0; #1
x1 = 1'b1; y1 = 1'b1; #2
$finish;
end
  endmodule
```

Simulation:

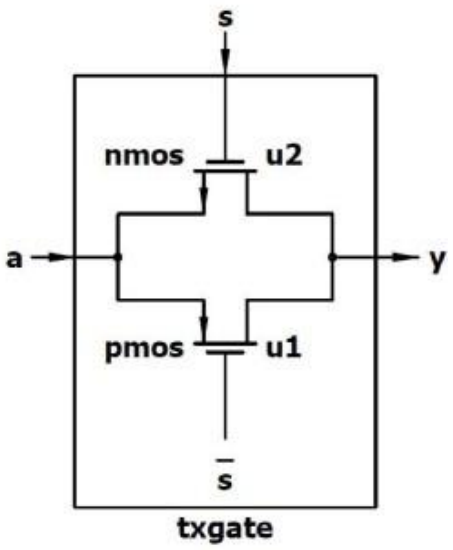


RESULT:

Verilog code for the OR gate is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for a **Transmission gate** and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUT
s	a	y
0	x	z
1	0	0
1	1	1

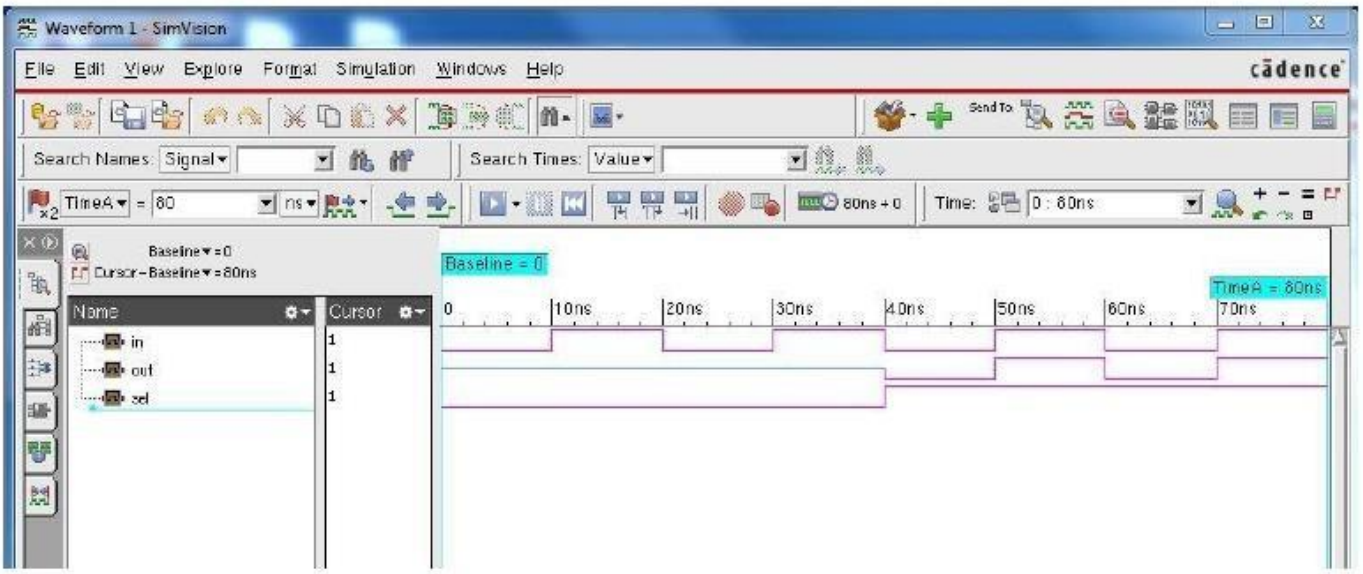
tgate.v:

```
module tgate(a,s,y);
input a,s;
output y;
pmos u1(y,a,~s); /* pmos name(drain, source, gate)*/
nmos u2(y,a,s); /* nmos name(drain, source, gate)*/
  endmodule
```

tgate_tb.v:

```
module tgate_tb;
reg in, sel;
wire out;
tgate u3(in,sel,out);
initial begin
in = 1'b0 ; sel = 1'b0 ; #10
in = 1'b1 ; sel = 1'b0 ; #10
in = 1'b0 ; sel = 1'b0 ; #10
in = 1'b1 ; sel = 1'b0 ; #10
in = 1'b0 ; sel = 1'b1 ; #10
in = 1'b1 ; sel = 1'b1 ; #10
in = 1'b0 ; sel = 1'b1 ; #10
in = 1'b1 ; sel = 1'b1 ; #10
$finish;
end
  endmodule
```

Simulation:



RESULT:

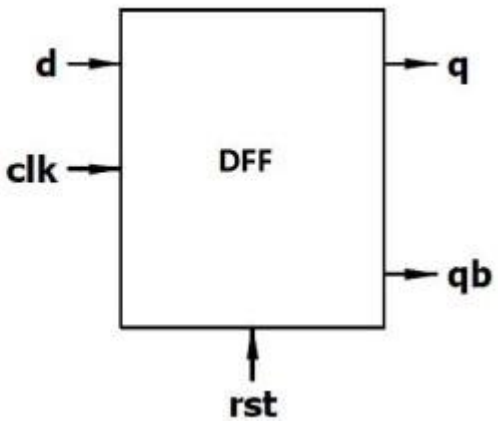
Verilog code for the Transmission gate is written, verification for the same is done using test bench and the waveform is observed

Experiment No. : 3	Date : / / .
Design and verification of Flip flop -RS, D, JK, MS, T, etc…	

FLIPFLOPS

Aim: To write Verilog code for **D Flip Flop** and their Test Bench for verification

Module:



Truth table:

INPUTS		OUTPUTS		
clk	rst	d	q	qb
↑	1	x	0	1
↑	0	0	0	1
↑	0	1	1	0

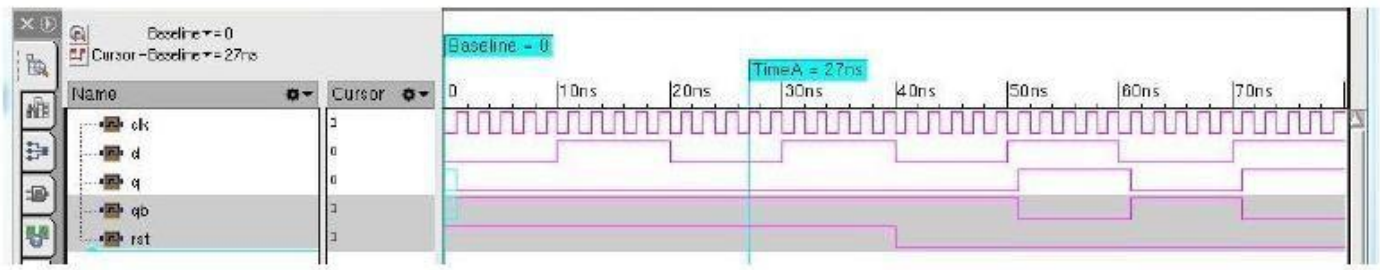
dff.v:

```
module DFF(clk,rst,d,q,qb);
input d,clk,rst;
output q,qb;
reg q;
assign qb = ~q;
always@ (posedge(clk))
begin
if(rst)
q <= 1'b0;
else
q <= d;
end
  endmodule
```

dff_tb.v:

```
module DFF_tb;
reg d,clk,rst;
wire q,qb;
DFF DFF1 (clk,rst,d,q,qb);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b1; d = 1'b0; #10
rst = 1'b1; d = 1'b1; #10
rst = 1'b1; d = 1'b0; #10
rst = 1'b1; d = 1'b1; #10
rst = 1'b0; d = 1'b0; #10
rst = 1'b0; d = 1'b1; #10
rst = 1'b0; d = 1'b0; #10
rst = 1'b0; d = 1'b1; #10
$finish;
end
  endmodule
```

Simulation:

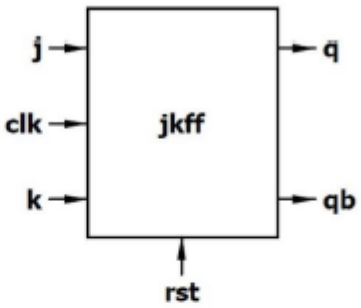


RESULT:

Verilog code for the D flip flop is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for **JK Flip Flop** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS		
clk	rst	j	k	q	qb
↑	1	x	x	0	1
↑	0	0	0	q	qb
↑	0	0	1	0	1
↑	0	1	0	1	0
↑	0	1	1	qb	q

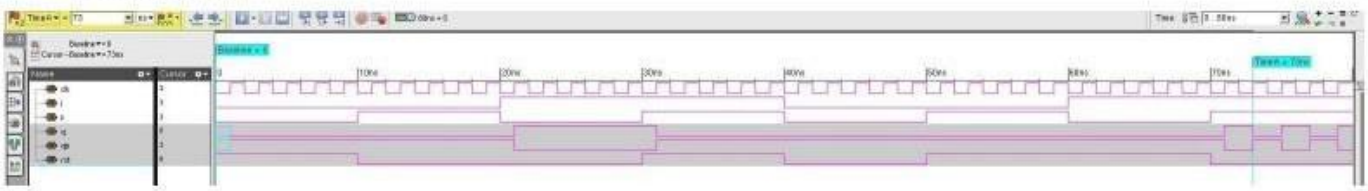
jkff.v:

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

jkff_tb.v:

```
module JKFF_tb;
reg j,k,clk,rst;
wire q,qb;
JKFF jkff1(clk,rst,j,k,q,qb);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b1; j = 1'b0; k = 1'b0 ; #10
rst = 1'b0; j = 1'b0; k = 1'b1 ; #10
rst = 1'b0; j = 1'b1; k = 1'b0 ; #10
rst = 1'b1; j = 1'b1; k = 1'b1 ; #10
rst = 1'b0; j = 1'b0; k = 1'b0 ; #10
rst = 1'b1; j = 1'b0; k = 1'b1 ; #10
rst = 1'b1; j = 1'b1; k = 1'b0 ; #10
rst = 1'b0; j = 1'b1; k = 1'b1 ; #10
$finish;
end
  endmodule
```

Simulation:

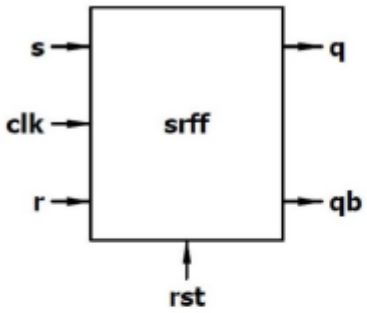


RESULT:

Verilog code for the JK flip flop is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for **SR Flip Flop** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS		
clk	rst	s	r	q	qb
↑	1	x	x	0	1
↑	0	0	0	q	qb
↑	0	0	1	0	1
↑	0	1	0	1	0
↑	0	1	1	x	x

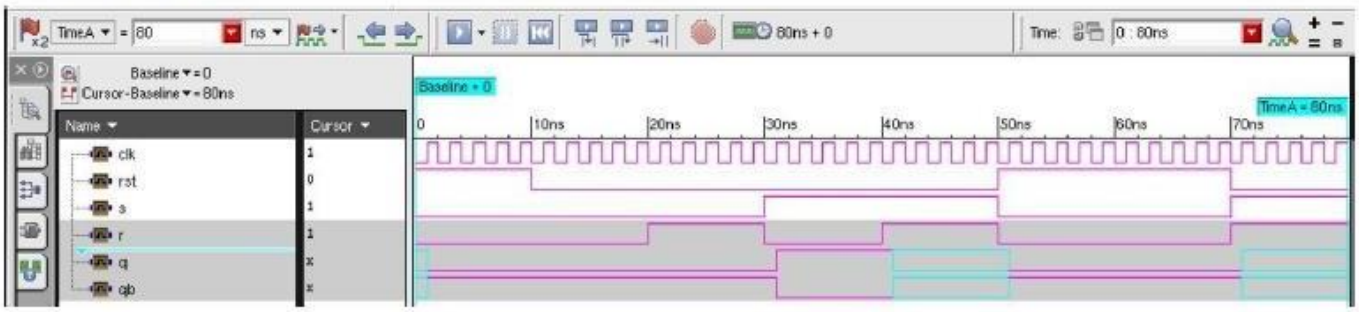
srf.v:

```
module srff(clk,rst,s,r,q,qb);
input s,r,clk,rst;
output q,qb;
reg q;
assign qb = ~q;
always @(posedge(clk))
begin
if(rst)
q <= 1'b0;
else
begin
case ({s,r})
2'b00 : q <= q;
2'b01 : q <= 1'b0;
2'b10 : q <= 1'b1;
2'b11 : q <= 1'bx;
endcase
end
end
endmodule
```

srf_tb.v:

```
module srff_tb;
reg s,r,clk,rst;
wire q,qb;
srff u1(clk,rst,s,r,q,qb);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b1 ; s = 1'b0 ; r = 1'b0 ; #10
rst = 1'b0 ; s = 1'b0 ; r = 1'b0 ; #10
rst = 1'b0 ; s = 1'b0 ; r = 1'b1 ; #10
rst = 1'b0 ; s = 1'b1 ; r = 1'b0 ; #10
rst = 1'b0 ; s = 1'b1 ; r = 1'b1 ; #10
rst = 1'b1 ; s = 1'b0 ; r = 1'b0 ; #10
rst = 1'b1 ; s = 1'b0 ; r = 1'b0 ; #10
rst = 1'b0 ; s = 1'b1 ; r = 1'b1 ; #10
$finish;
end
endmodule
```

Simulation:

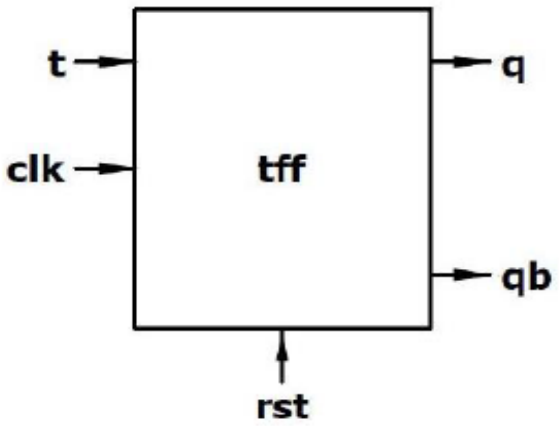


RESULT:

Verilog code for the SR flip flop is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for **T Flip Flop** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS	
clk	rst	t	q	qb
↑	1	x	0	1
↑	0	0	q	qb
↑	0	1	qb	q

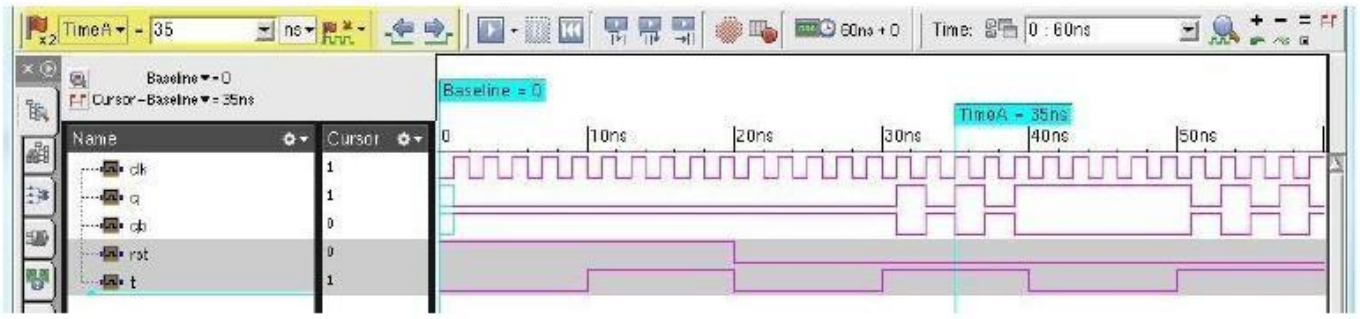
tff.v:

```
module tff(clk,rst,t,q,qb);
input t,clk,rst;
output q,qb;
reg q;
assign qb = ~q;
always @(posedge(clk))
begin
if(rst)
q <= 1'b0;
else
if(t) q <= ~q;
end
endmodule
```

tff_tb.v:

```
module tff_tb;
reg t,clk,rst;
wire q,qb;
tff u1 (clk, rst, t, q,qb);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b1; t=1'b0; #10
rst = 1'b1; t=1'b1; #10
rst = 1'b0; t=1'b0; #10
rst = 1'b0; t=1'b1; #10
rst = 1'b0; t=1'b0; #10
rst = 1'b0; t=1'b1; #10
$finish;
end
endmodule
```

Simulation:



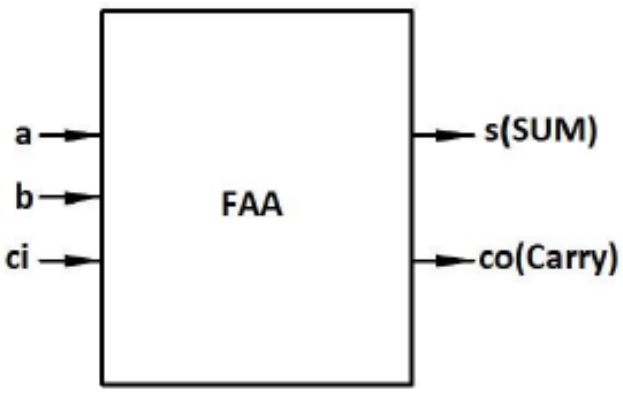
RESULT:

Verilog code for the T flip flop is written, verification for the same is done using test bench and the waveform is observed

Experiment No. : 5	Date : / / .
Design and verification of Serial & Parallel adder	

Aim: To write Verilog code for **FULL ADDER** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS	
a	b	ci	s	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

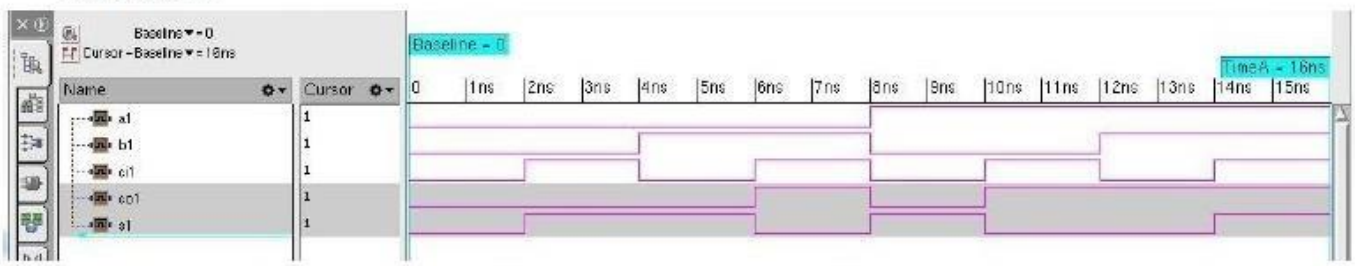
FAA.v:

```
module FAA (a,b,ci,s,co);
input a,b,ci;
output s,co;
wire vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
assign s = a^b^ci;
assign co = (a&b)|(b&ci)|(ci&a);
  endmodule
```

FAA_tb.v:

```
module FAA_tb;
reg a1,b1,ci1;
wire s1,co1;
FAA FA1 (a1,b1,ci1,s1,co1);
initial begin
a1=1'b0; b1=1'b0; ci1=1'b0; #2
a1=1'b0; b1=1'b0; ci1=1'b1; #2
a1=1'b0; b1=1'b1; ci1=1'b0; #2
a1=1'b0; b1=1'b1; ci1=1'b1; #2
a1=1'b1; b1=1'b0; ci1=1'b0; #2
a1=1'b1; b1=1'b0; ci1=1'b1; #2
a1=1'b1; b1=1'b1; ci1=1'b0; #2
a1=1'b1; b1=1'b1; ci1=1'b1; #2
$finish;
end
  endmodule
```

Simulation:



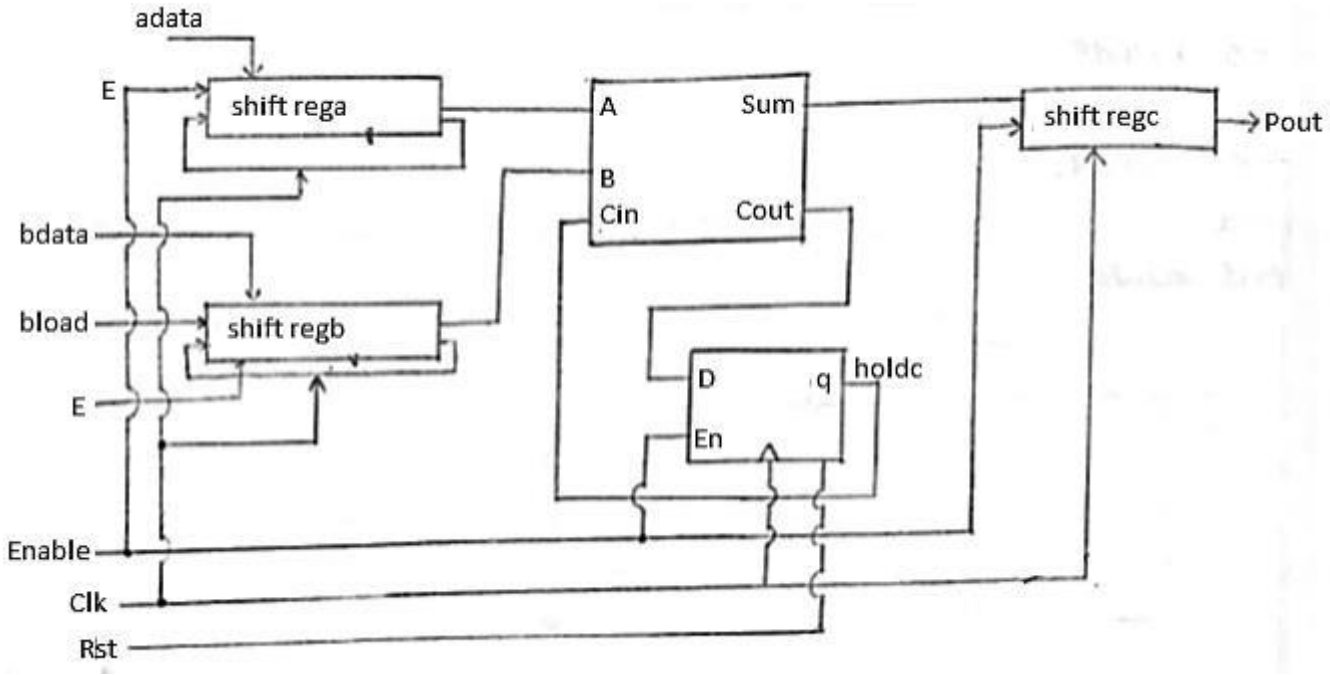
RESULT:

Verilog code for the Full adder is written, verification for the same is done using test bench and the waveform is observed

SERIAL ADDER

Aim: To write Verilog code for **Serial Adder** and their Test Bench for verification

Circuit Diagram:



Truth Table:

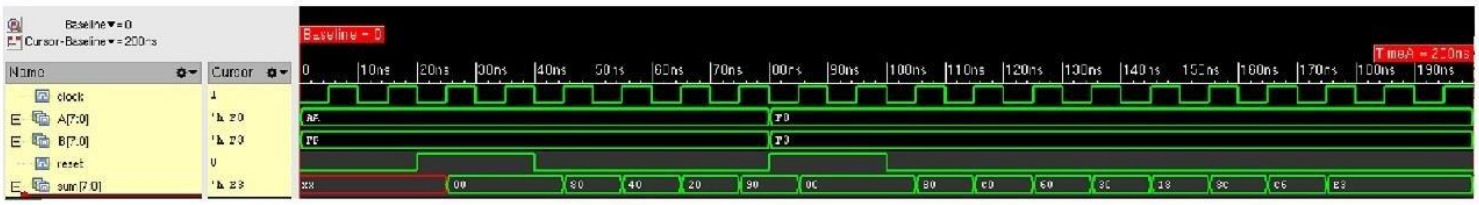
Shiftrega		Shiftregb		Shiftregc	
0000 0001	01	0000 0010	02	0000 0000	
1000 0000	80	0000 0001	01	1000 0000	80
0100 0000	40	1000 0000	80	1100 0000	C0
0010 0000	20	0100 0000	40	0110 0000	60
0001 0000	10	0010 0000	20	0011 0000	30
0000 1000	08	0001 0000	10	0001 1000	18
0000 0100	04	0000 1000	08	0000 1100	0C
0000 0010	02	0000 0100	04	0000 0110	06
0000 0001	01	0000 0010	02	0000 0011	03

sadd.v

```
module serial_adder(clk,rst,pload,adata,bdata,enable,pout);
input clk,rst,pload,enable;
input [7:0]pout;
reg shiftrega_lsb,shiftregb_lsb;
reg [7:0]shiftrega,shiftregb,shiftregc;
wire sum,cout;
reg holdc;
always@(posedge clk)
begin
  if(pload)
  begin
        shiftrega=adata;
        shiftregb=bdata;
        shiftregc=8'b0;
  end
else if(enable)

  begin
        shiftrega_lsb=shiftrega[0];
        shiftrega=shiftrega>>1;
        shiftrega[7]=shiftrega_lsb;
        shiftregb_lsb=shiftregb[0];
        shiftregb=shiftregb>>1;
        shiftregb[7]=shiftregb_lsb;
  end

  shiftregc=shiftregc>>1;
  shiftregc[7]=sum;

end

assign pout=shiftregc;
f1 s1(shiftrega[0], shiftregb[0],holdc,sum,cout);
always@(posedge clk,rst)
begin
  if(rst)
        holdc=1'b0;
  else if(enable)
        holdc=cout;
  else
        holdc=holda;
end

endmodule
```

sadd_tb.v

```
 module serial_adder_tb;

reg clk,rst,pload,enable;
reg [7:0]adata,bdata;
wire [7:0]pout;
serial_adder s1(clk,rst,pload,adata,bdata,enable,pout)
initial
begin
  clk=1'b0;
  rst=1'b1;
  pload=1'b0;
  #10 rst=1'b0; pload=1'b1;
  #5 adata=8'd1; bdata=8'd10;
#5 pload=1'b0; enable=1'b1;
#200
$stop;
End
always
#5 clk=~clk;
endmodule
```



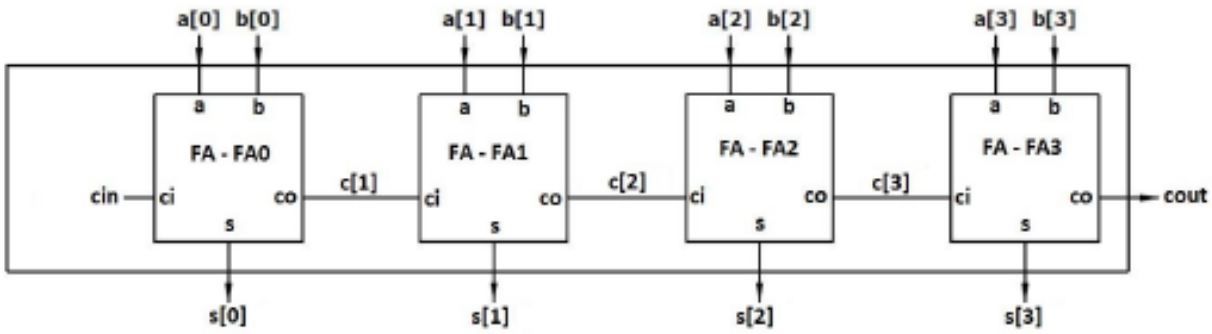
RESULT:

Verilog code for the Serial adder is written, verification for the same is done using test bench and the waveform is observed

PARELLEL ADDER

Aim: To write Verilog code for **Parallel Adder** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS	
a	b	cin	s	cout
1h(0001b)	2h(0010b)	1	4h(0100b)	0
Fh(1111b)	Fh(1111b)	1	Fh(1111b)	1
9h(1001b)	Ah(1010b)	1	4h(0100b)	1
1h(0001b)	2h(0010b)	0	3h(0011b)	0

padd.v:

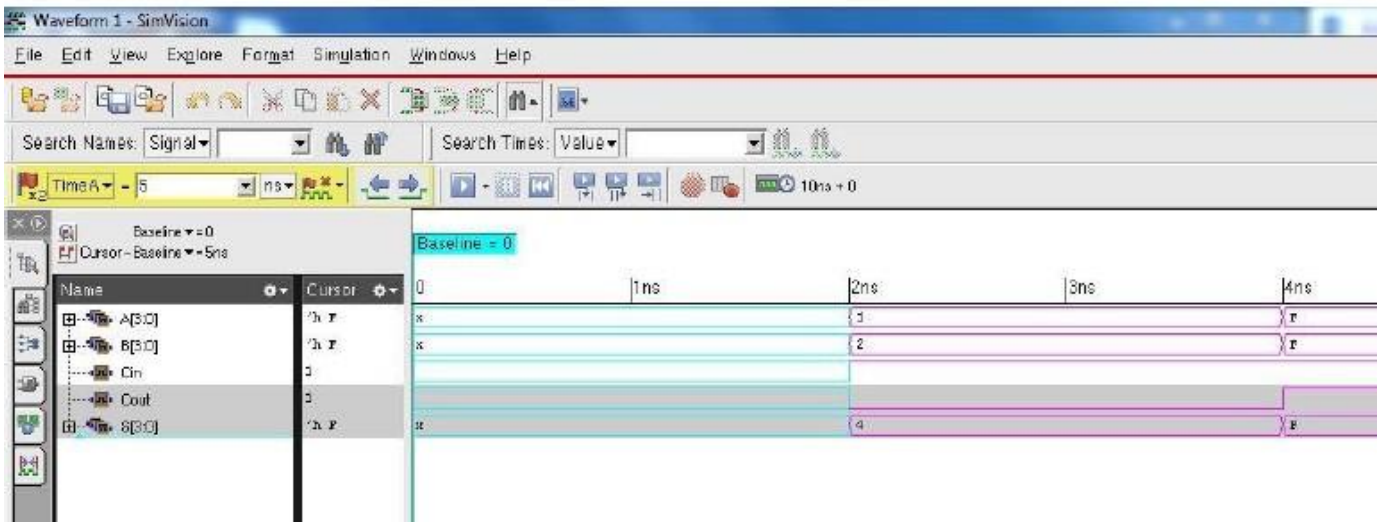
```
module padd (a,b,cin,s,cout);
input [3:0]a,b;
input cin;
output [3:0]s;
output cout;
wire [3:1]c;
FAA FA0 (a[0],b[0],cin,s[0],c[1]);
FAA FA1 (a[1],b[1],c[1],s[1],c[2]);
FAA FA2 (a[2],b[2],c[2],s[2],c[3]);
FAA FA3 (a[3],b[3],c[3],s[3],cout);
endmodule
module FAA (a,b,ci,s,co);
input a,b,ci;
output s,co;
wire vdd, gnd;
assign vdd = 1'b1;
assign gnd = 1'b0;
assign s = a^b^ci;
assign co = (a&b)|(b&ci)|(ci&a);
  endmodule
```

padd_tb.v:

```
module padd_tb;
```

```
reg [3:0]A,B;
reg Cin;
wire [3:0]S;
wire Cout;
padd PA1 (A,B,Cin,S,Cout);
initial begin
A=4'b0001; B=4'b0010; Cin=1'b1; #2
A=4'b1111; B=4'b1111; Cin=1'b1; #2
A=4'b1001; B=4'b1010; Cin=1'b1; #2
A=4'b0001; B=4'b0010; Cin=1'b0; #2
$finish;
end
  endmodule
```

Simulation:



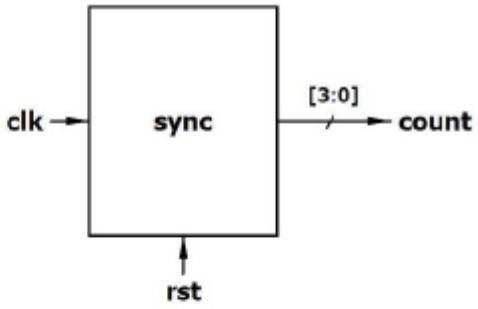
RESULT:

Verilog code for the Parallel adder is written, verification for the same is done using test bench and the waveform is observed

Experiment No. : 5	Date : / / .
Design and verification of 4-bit counter : Synchronous and Asynchronous counter	

Aim: To write Verilog code for **Synchronous Counter** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS		
clk	rst	count[3]	count[2]	count[1]	count[0]
↑	0	0	0	0	0
↑	1	0	0	0	1
↑	1	0	0	1	0
↑	1	0	0	1	1
↑	1	0	1	0	0
↑	1	0	1	0	1
↑	1	0	1	1	0
↑	1	0	1	1	1
↑	1	1	0	0	0
↑	1	1	0	0	1
↑	1	1	0	1	0
↑	1	1	0	1	1
↑	1	1	1	0	0
↑	1	1	1	0	1
↑	1	1	1	1	0
↑	1	1	1	1	1
↑	1	0	0	0	0

sync_cntr.v:

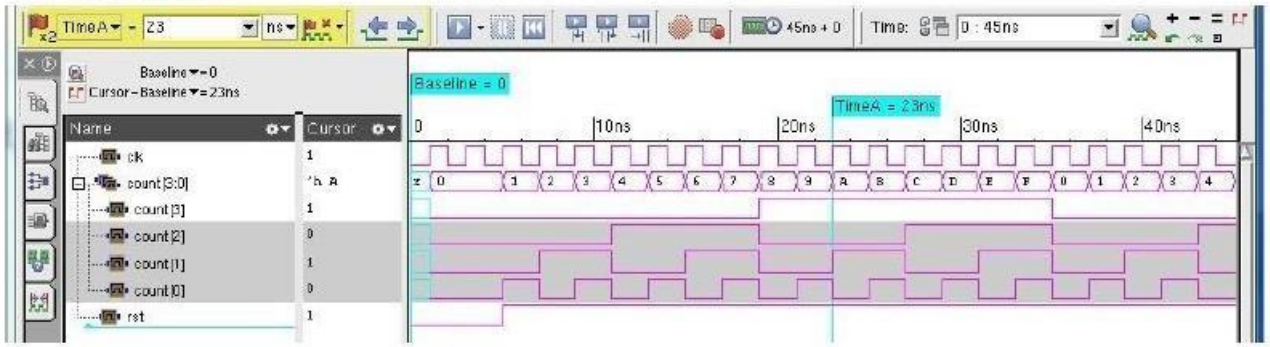
```
module syn_cntr (clk, rst, count);
input clk, rst;
output [3:0] count;
reg [3:0] count;
always @(posedge clk)
begin
if(!rst) /* considering active low reset */
count <= 4'b000;
```

```
else
count <= count + 1;
end
  endmodule
```

sync_cntr_tb.v:

```
module syn_cntr_tb;
reg clk,rst;
wire [3:0] count;
syn_cntr u1 (clk, rst, count);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b0; #5
rst = 1'b1; #40
$finish;
end
  endmodule
```

Simulation:

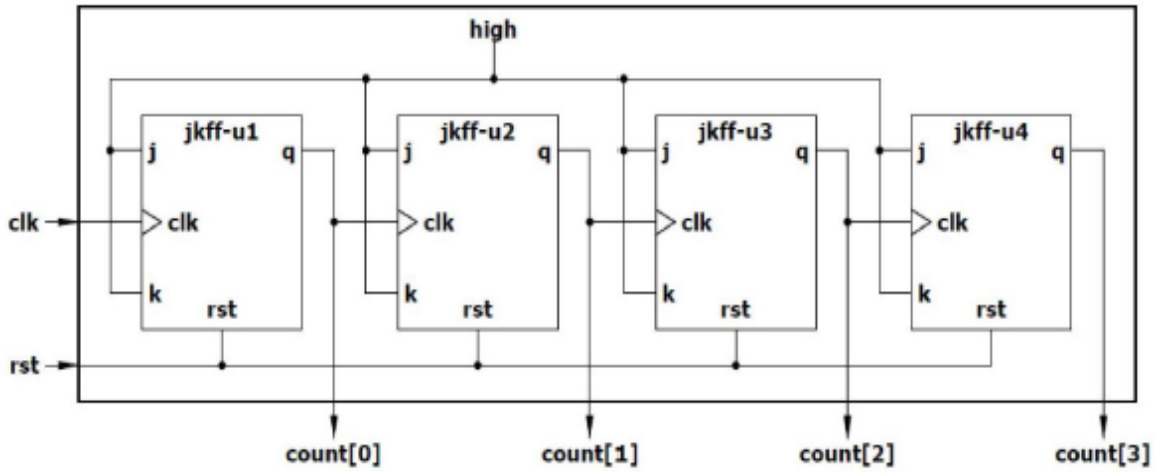


RESULT:

Verilog code for the Synchronous counter is written, verification for the same is done using test bench and the waveform is observed

Aim: To write Verilog code for **Asynchronous Counter** and their Test Bench for verification

Module:



Truth table:

INPUTS			OUTPUTS		
clk	rst	count[3]	count[2]	count[1]	count[0]
x	0	0	0	0	0
↑	1	0	0	0	1
↑	1	0	0	1	0
↑	1	0	0	1	1
↑	1	0	1	0	0
↑	1	0	1	0	1
↑	1	0	1	1	0
↑	1	0	1	1	1
↑	1	1	0	0	0
↑	1	1	0	0	1
↑	1	1	0	1	0
↑	1	1	0	1	1
↑	1	1	1	0	0
↑	1	1	1	0	1
↑	1	1	1	1	0
↑	1	1	1	1	1
↑	1	0	0	0	0

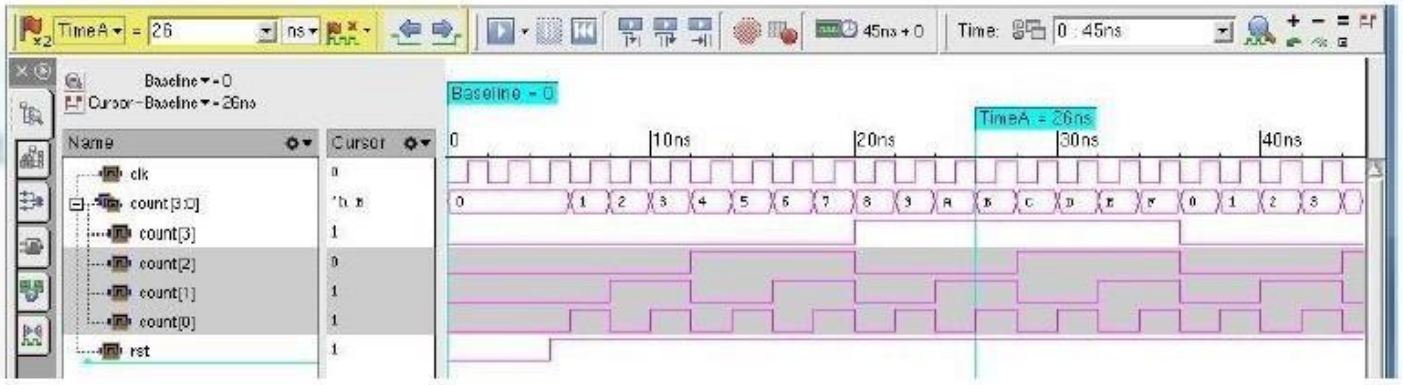
async_cntr.v:

```
module asyn_cntr (clk, rst, count);
input clk, rst;
output [3:0] count;
wire high;
assign high = 1'b1;
jkff u1 (clk, rst, high, high, count[0]);
jkff u2 (count[0], rst, high, high, count[1]);
jkff u3 (count[1], rst, high, high, count[2]);
jkff u4 (count[2], rst, high, high, count[3]);
endmodule
module jkff(clk, rst, j,k,q);
input j,k,clk,rst;
output q;
reg q;
always @(negedge clk or negedge rst)
begin
if (!rst)
q <= 1'b0;
else
begin
case ({j,k})
2'b00 : q <= q;
2'b01 : q <= 1'b0;
2'b10 : q <= 1'b1;
2'b11 : q <= ~q;
endcase
end
end
endmodule
```

async_cntr_tb.v:

```
module asyn_cntr_tb;
reg clk, rst;
wire [3:0] count;
asyn_cntr u1 (clk, rst, count);
initial begin
clk = 1'b0;
forever #1 clk = ~clk;
end
initial begin
rst = 1'b0; #5
rst = 1'b1; #40
$finish;
end
endmodul
```

Simulation:



RESULT:

Verilog code for the Asynchronous counter is written, verification for the same is done using test bench and the waveform is observed

II Cycle of experiments
Analog Design

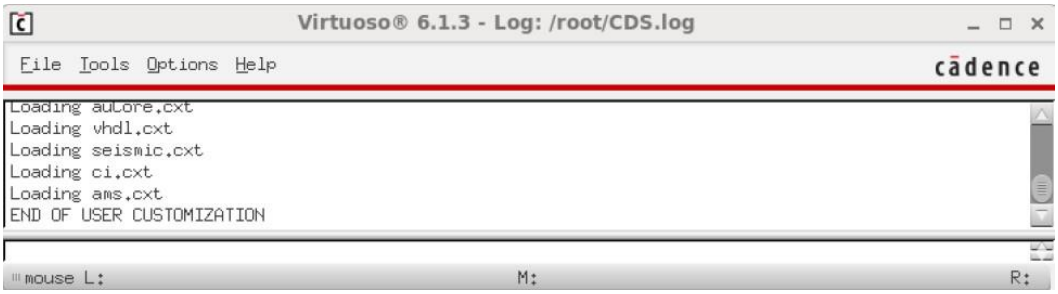
Analog Design using Cadence Software

6. Inverter
7. Common Source Amplifier
8. Common Drain Amplifier
9. Differential Amplifier
10. Operational Amplifier
11. R-2R DAC

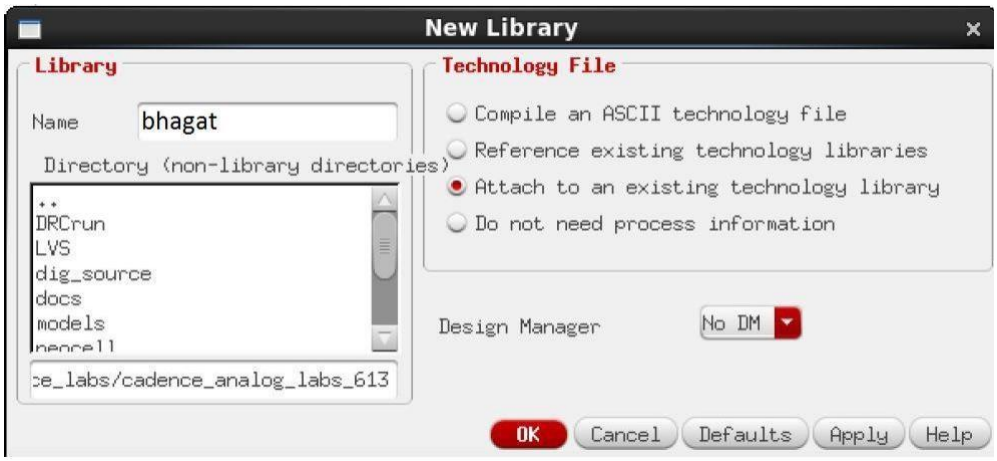
Experiment No. : 6	**Date : / / .**
Design an Inverter with given specifications	

Aim: To simulate the schematic of the CMOS inverter, and then to perform the physical verification for the layout of the same.

1. Start Virtuoso
[user4@vlsiserver cadence_analog_labs_613]$ virtuoso &



2.Create a new library, if already created proceed to next step
From Virtuoso window: **File > New > Library**



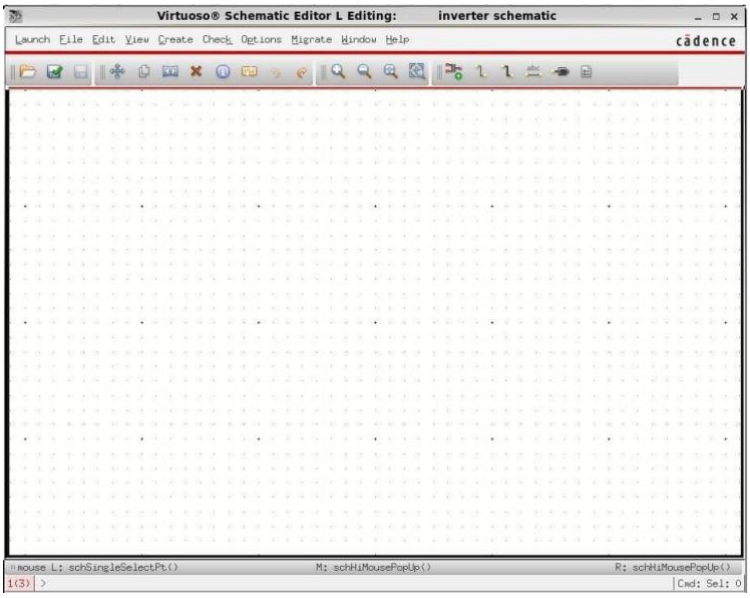
In the New Library window,
Give the name of the library: **bhagat (your_name)** Select : **Attach to an existing technology library** and Click **OK**

In the Attach library to Technology Library window
Select Technology Library: **gpdk180**
Click**: OK**

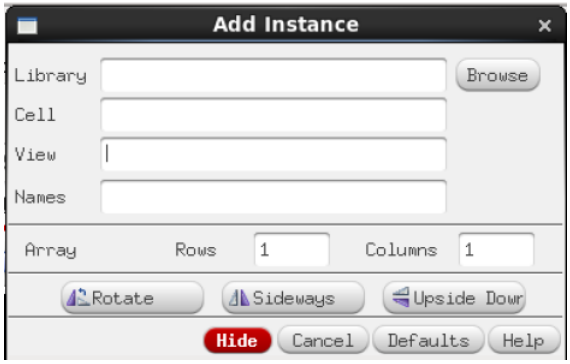
7. Create new Cellview
From Virtuoso window: **File > New > CellView**

Select the Library: **bhagat**
Name of the Cell: **inverter**
View: **schematic**
Type: **schematic**
Click: **OK**

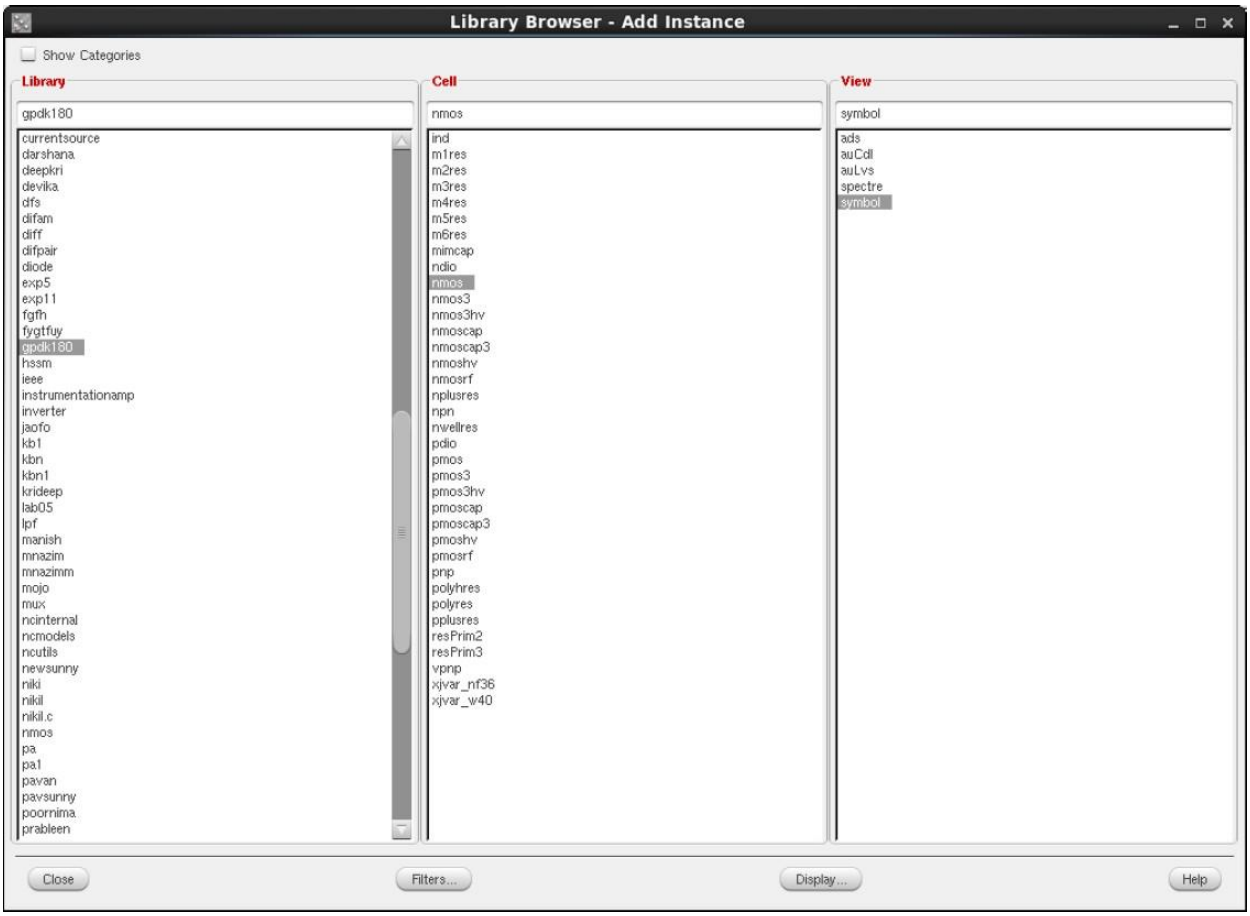
8. The Virtuoso Schematic Editor is opened
In the Virtuoso Schematic Editor required circuit can be drawn using transistors, wires, ports etc.



9. Insert the transistor symbol using instance
In Virtuoso Schematic Editor: **Create > Instance**



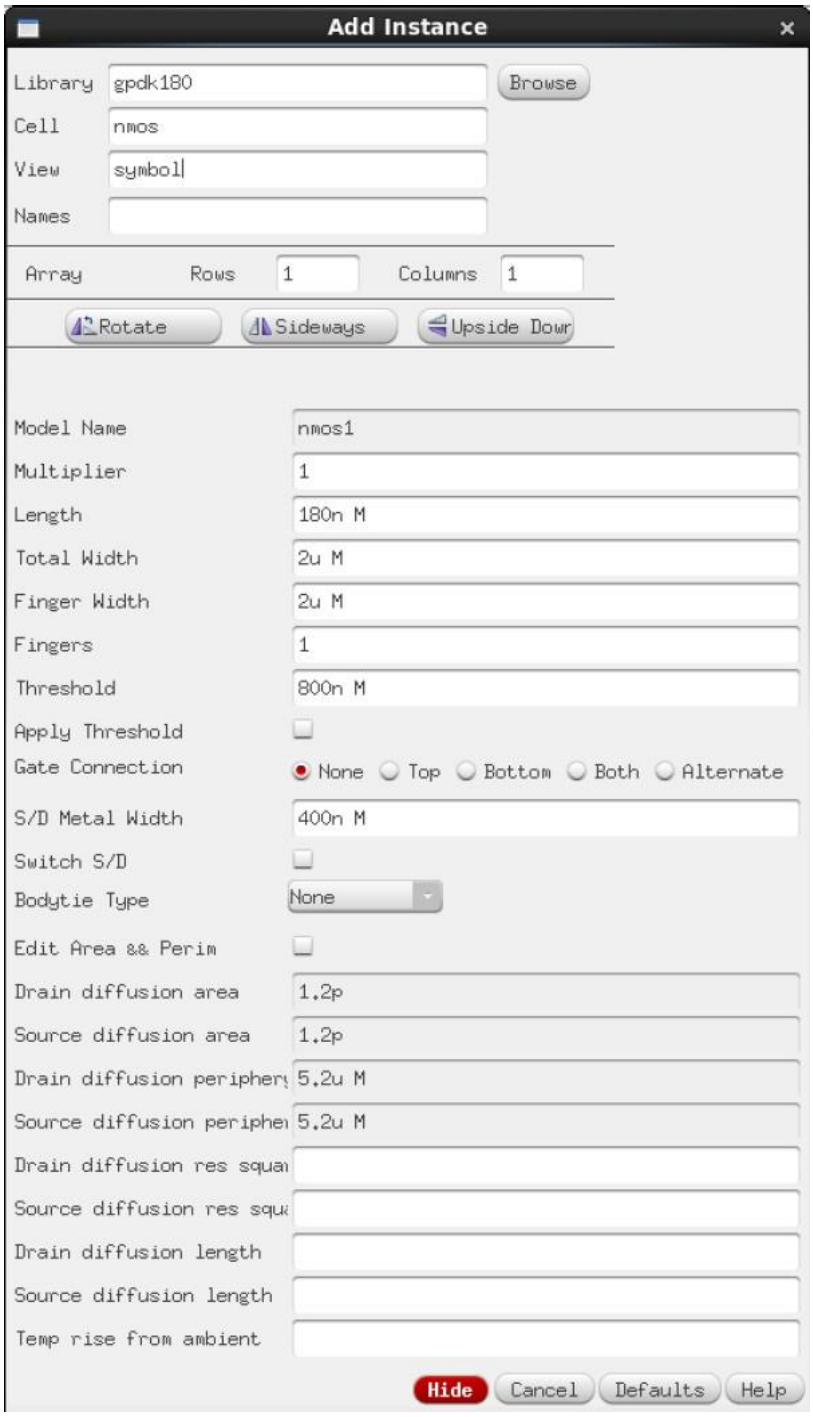
In Add Instance Window Click **Browse** to open library browser



In the Library Browser Window

Select Library: **gpdk180**
Cell: **nmos** (the required instance)
View: **symbol**
Click: **Close** to return to add instance window

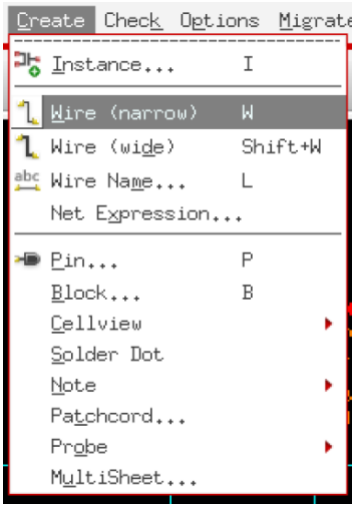
After selecting the instance from Library Browser the Add Instance Window displays the parameters applicable for the selected instance. Any parameters required for this instance can be specified.

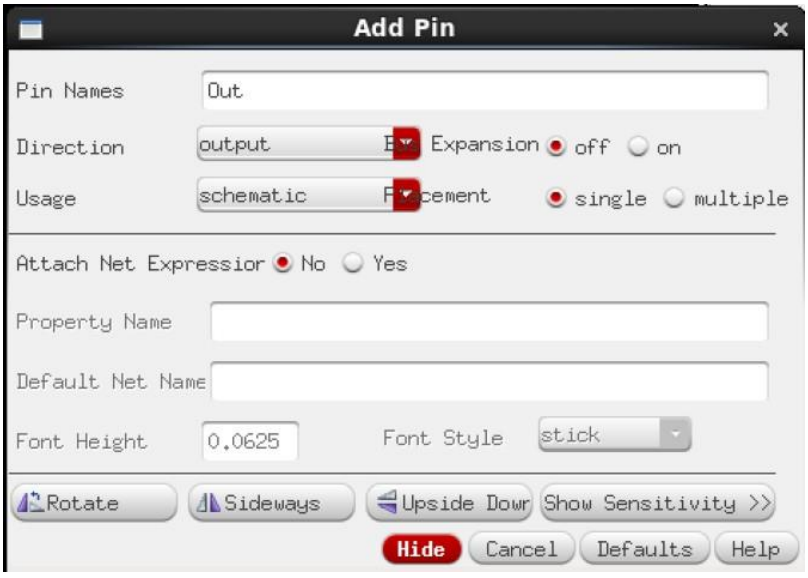


Click **Hide** to hide Add Instance Window and to start placing the components on Schematic Editor.
Place the required number components on the Schematic Editor
By following the same step insert all the components required for the circuit

10. Add Pins for circuit inputs, outputs, Vdd, Vss

In Virtuoso Schematic Editor: **Create > Pin**
You can also create Pin directly from toolbar



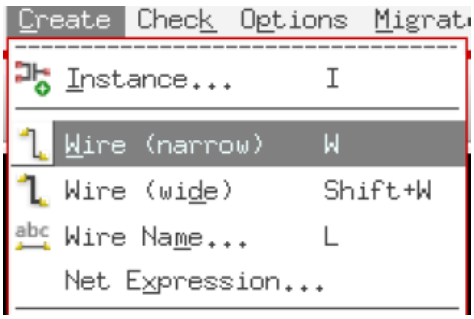


In the Add Pin Window enter the pin name and direction

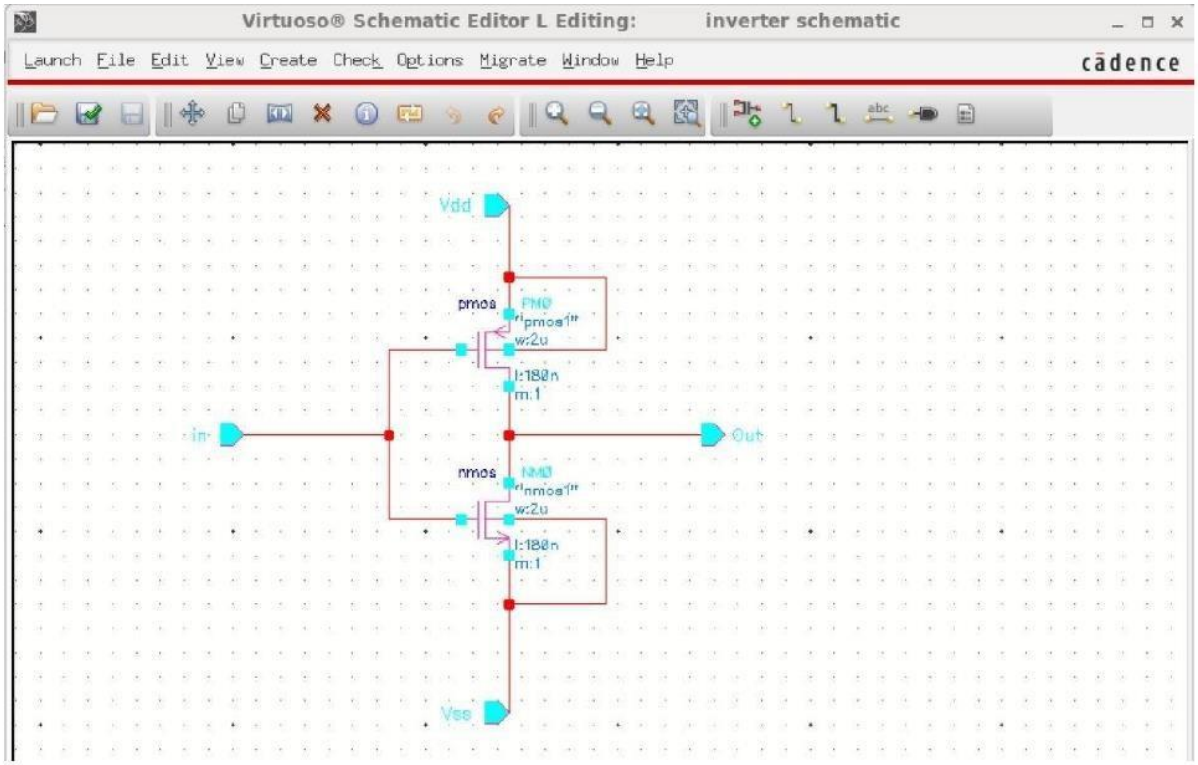
Give Pin Name: **Out / In / Vdd / Vss**
Give the Direction: **output / input / inputoutput**
Click **Hide** to return to schematic and place the pin.
Repeat this step for all the Pins required for the circuit taking care of direction for each pin.

11. Inter Connect the components using wires
In Virtuoso Schematic Editor: **Create > Wire (Narrow)**

You can also create Wire directly from toolbar



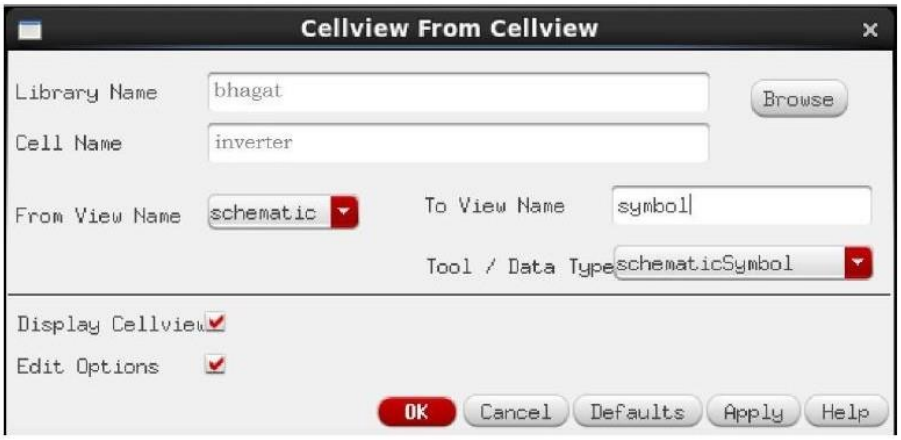
12. Check and save the circuit **File > Check and Save**
There should not be any errors and warnings in Virtuoso Log window.



In case of errors and warning check from virtuoso window the error type and correct them before proceeding to next step

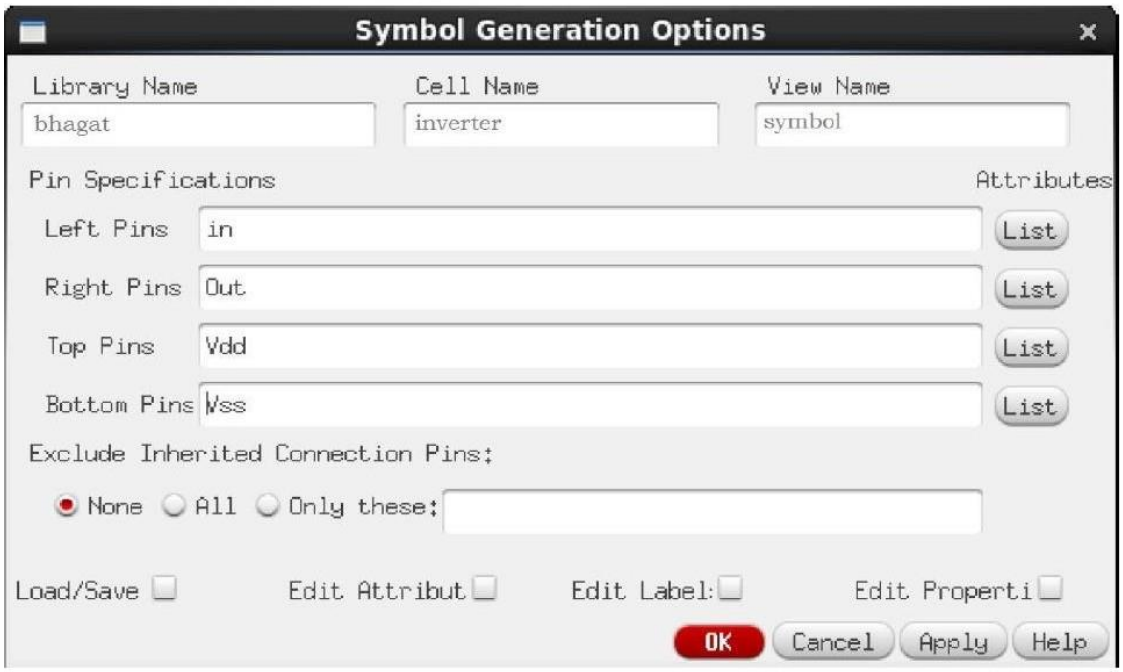
13. Once the schematic is complete, a symbol has to be created for this circuit to facilitate placing this circuit in another schematic.

14. Create Cellview from Cellview
    In Virtuoso Schematic Editor: **Create > Cellview > From cellview**



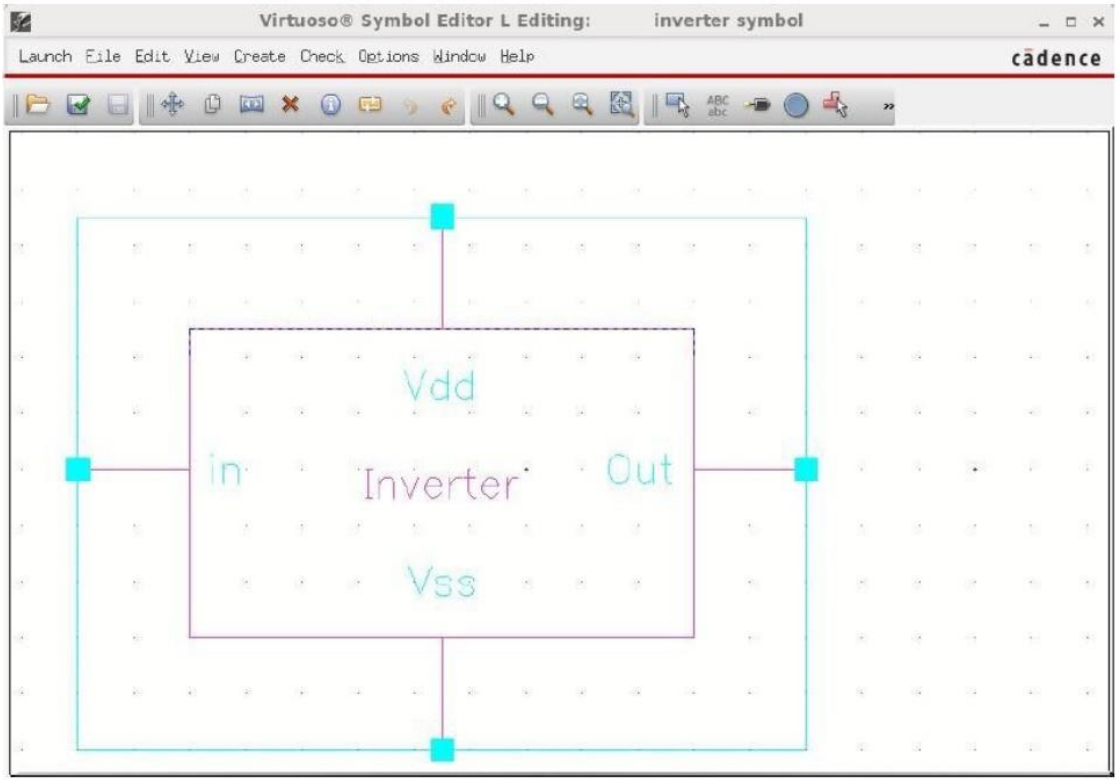
Click **OK** to open Symbol Generation Options window

15. In Symbol Generation Options window select Locations of pins you would like to appear on the symbol.



Group the pins that need to appear on Left side of symbol in **Left Pins** such as **input1, input2** etc. Similarly classify the pins depending on the location Right / Top/ Bottom Location. Multiple Pins can be specified for each orientation by separating them with a space.
Click **OK** to open Virtuoso Symbol Editor

16. Edit the symbol In the Symbol Editor Window then check and save (**File > Check and Save**)



After saving the symbol with no errors close the symbol editor.
This completes creation of a new component (**inverter** in this example) with its schematics and symbol. It will be available in the library (**bhagat** in this example) and can be used in other schematics.

Functioning of this circuit can be checked by creating a new test cell and calling this component from the test cell.

17. For testing circuit, create new Cellview
From Virtuoso window:
File > New > CellView

Select the Library: **bhagat**
Name of the Cell: **inverter_test**
View: **schematic**
Type: **schematic**
Click **OK**

18. Insert the Inverter instance on the schematic editor (Refer Step 11)

In Virtuoso Schematic Editor: **Create > Instance**

Library : **bhagat**

Cell : **inverter (cell to be tested)**

View : **symbol**

After placing the inverter symbol on the schematic, insert all required instances for the testing from the library.

For this example (**inverter_test**) the following instances are used

1. DC Ground

Library : **analogLib**

Cell : **gnd**

View : **symbol**

2. DC Supply Voltage

Library : **analogLib**

Cell : **vdc**

View : **symbol**

DC Voltage : **1.8 V**

3. Pulse Generator

Library : **analogLib**

Cell : **vpulse**

View : **symbol**

Voltage 1 : **0 V**

Voltage 2 : **1.8 V**

Period : **20n s**

Delay time : **0 s**

Rise time : **1n s**

Fall time : **1n s**

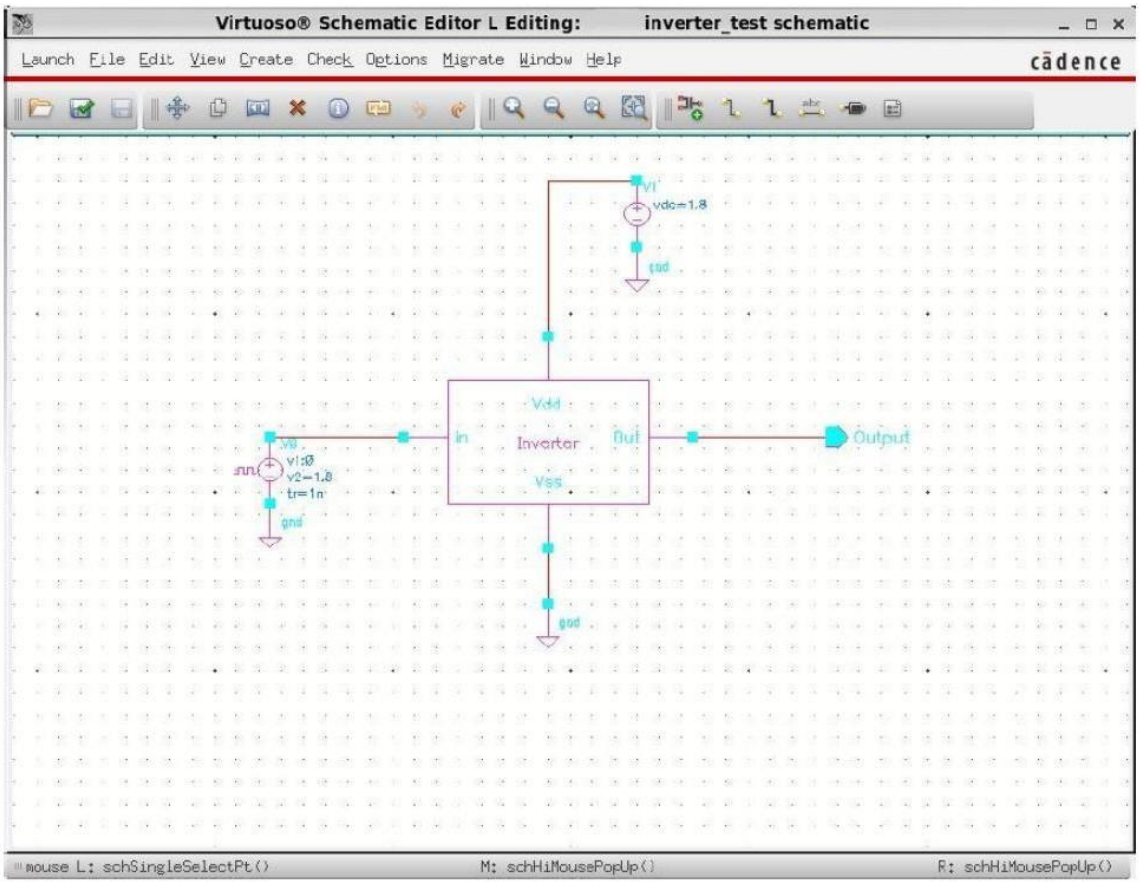
Pulse width : **10n s**

Edit Object Properties of instances either at the time of creation or after insertion

(**Select Instance** > **Right Click** > **Properties**)

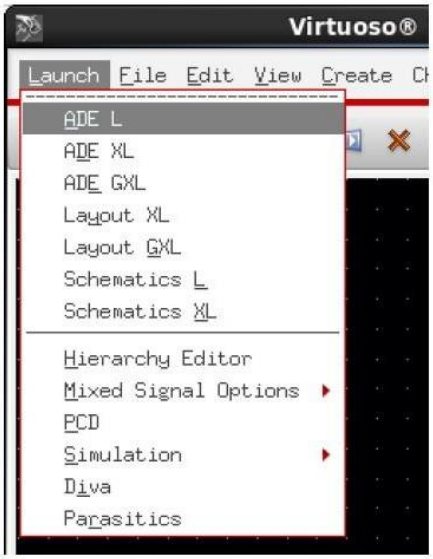
Interconnect the schematic using wires and Pins; Check and save the circuit

(**File > Check and Save**) Correct if any errors found



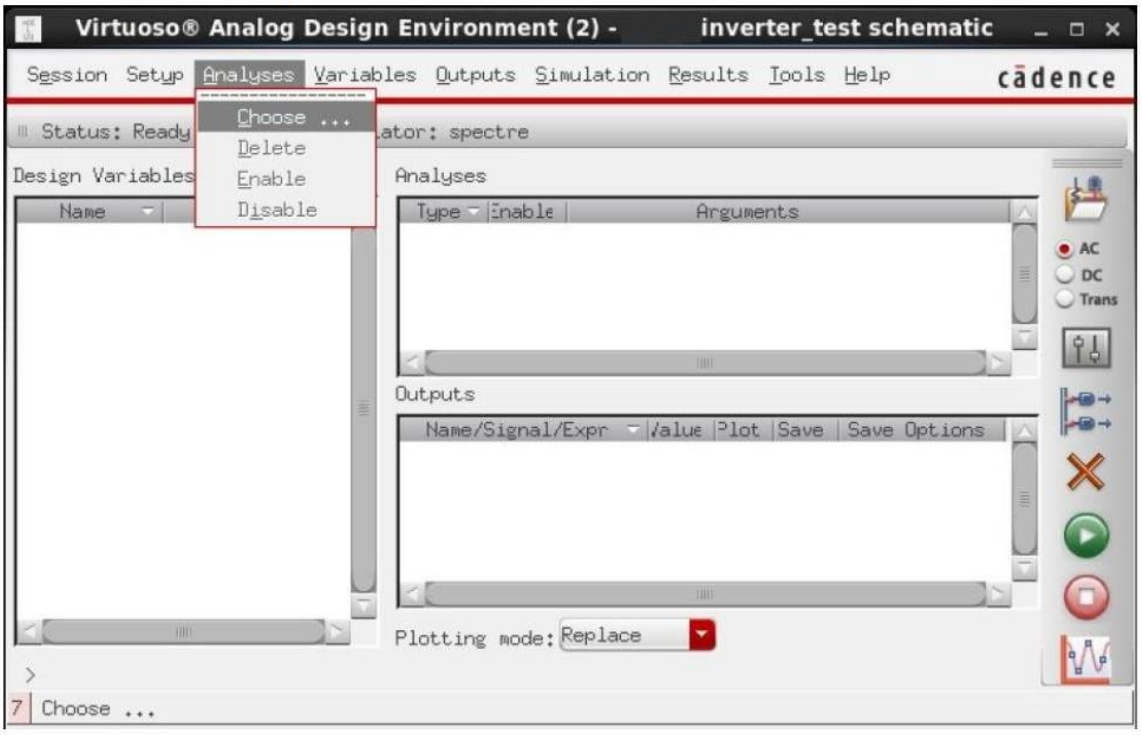
19. Launch Analog Design Environment

In Virtuoso Schematic Editor: **Launch > ADE L**



20.In Analog Design Environment window

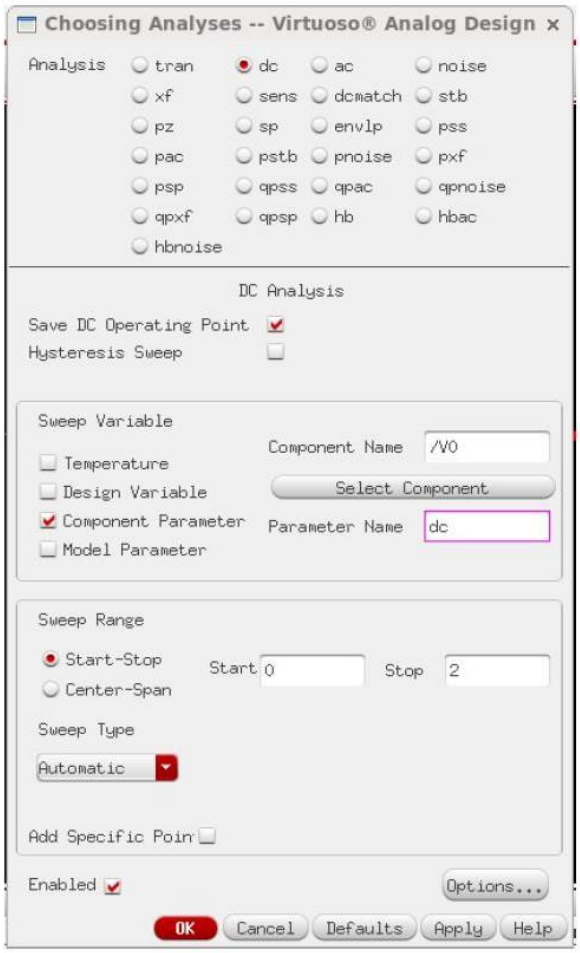
Select **Analyses > Choose**



21.In Choosing Analyses window for Transient Analysis

Select Analysis:	**tran**
Stop Time:	**200n**
Accuracy Defaults:	**moderate**
Enabled:	**Checked**
Click:	**OK**

22.Next for DC Analysis in Analog Design Environment window
Select **Analyses > Choose**



Analysis: **dc**
Select: **Save DC Operating Point**
Select: **Sweep Range**
Start: **0**
Stop: **2**

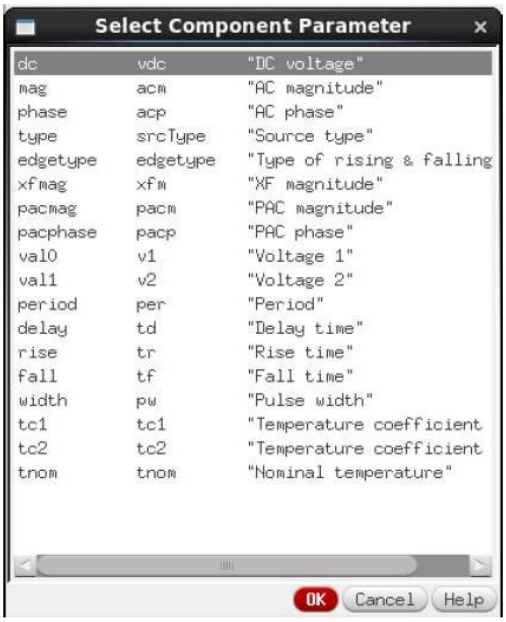
Click: **Select Component**

Select the pulse generator (**V0** in this case) from the schematic editor

In the Select Component Parameter window

Select: **dc**
Click: **OK**

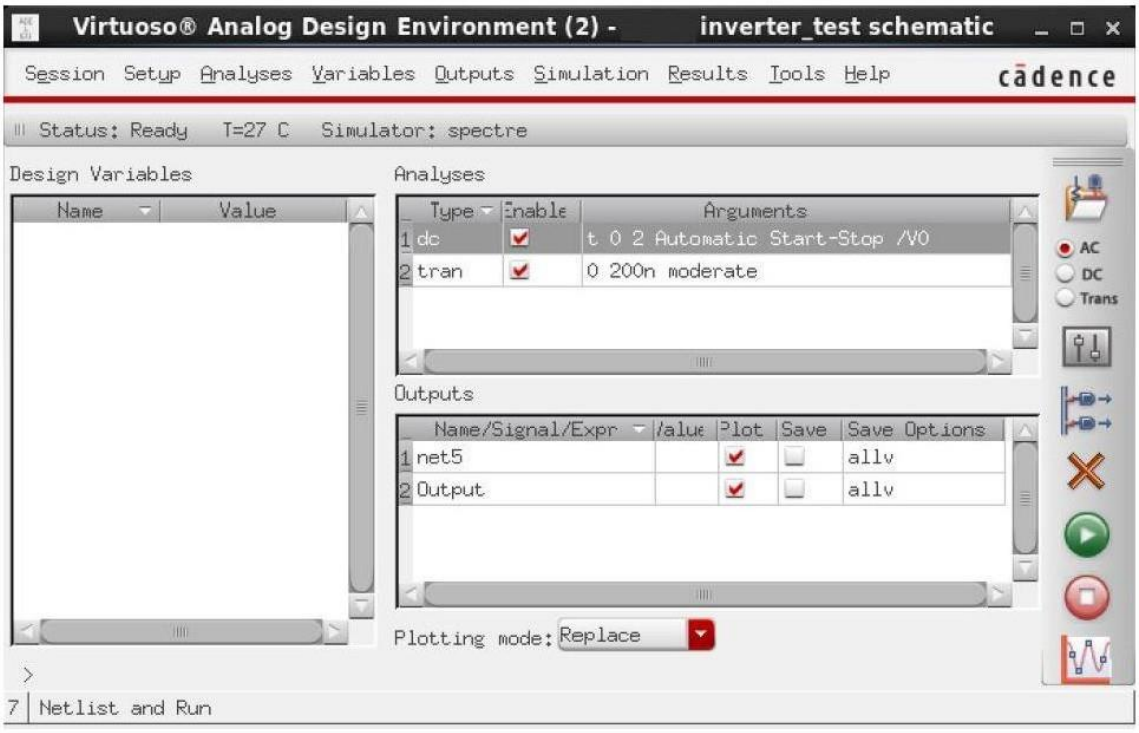
In Choosing Analyses window
Click: **OK**



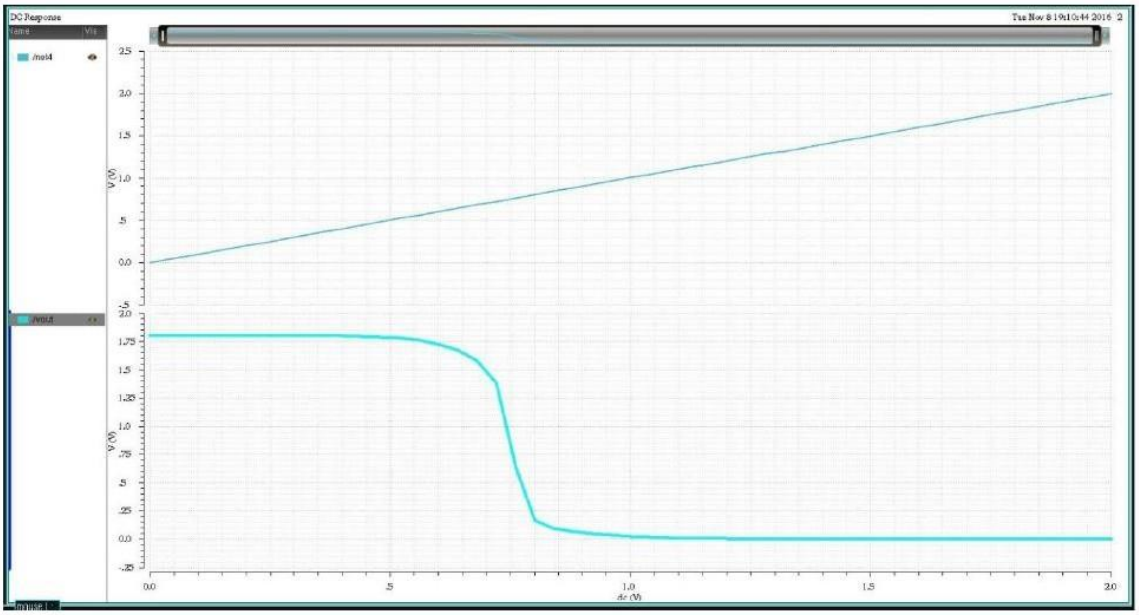
23. Select outputs to be plotted in Analog Design Environment window

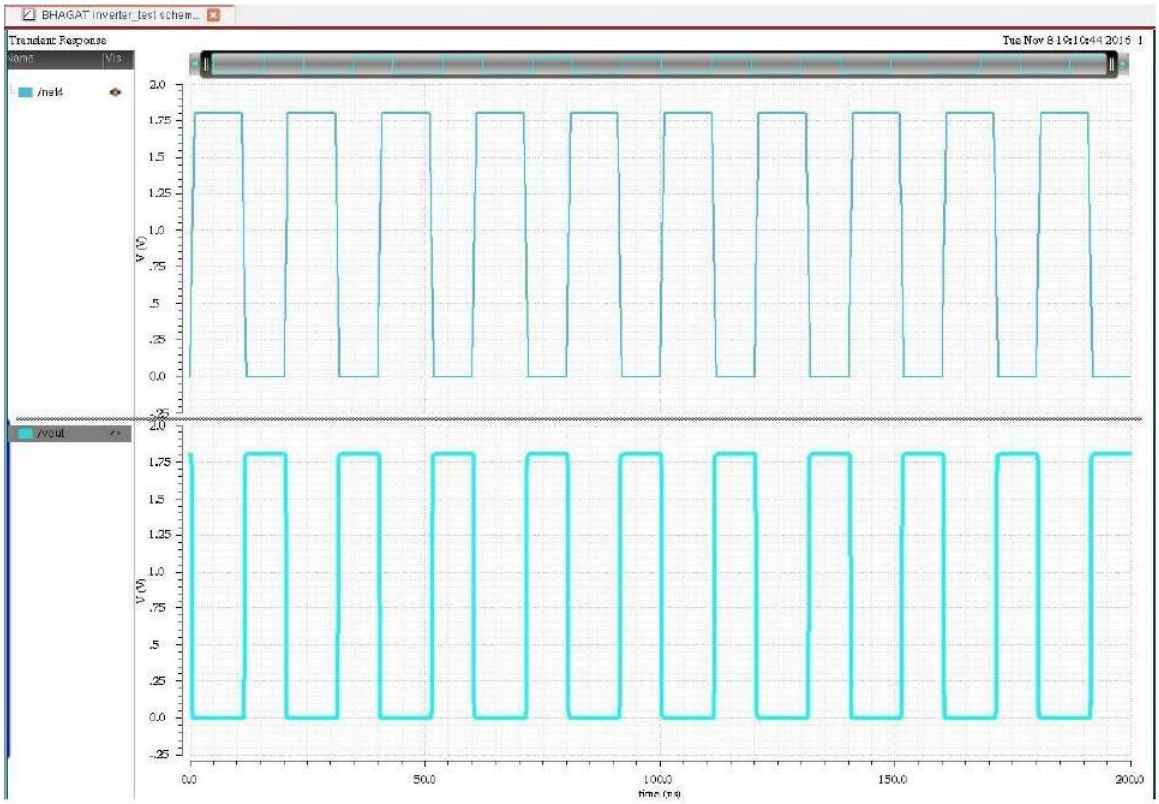
Select **Outputs > To Be Plotted > Select On Schematic**

Select input and output wires from the schematic



24. Run the Analysis from **Simulation > Run** and observe the waveforms





Creating Layout View of Inverter

1. From the **Inverter** schematic window menu execute
Launch – Layout XL. A **Startup Option** form appears.

2. Select ***Create New*** option. This gives a New Cell View Form

3. Check the Cellname **(Inverter)**, Viewname **(layout).**

4. Click **OK** from the New Cellview form.
LSW and a blank layout window appear along with schematic window.

Adding Components to Layout

1. Execute **Connectivity – Generate – All from Source** or click the icon in the layout editor window, **Generate Layout** form appears. Click **OK** which imports the schematic components in to the Layout window automatically.

2. Re arrange the components with in PR-Boundary as shown in the next page.

3. To rotate a component, Select the component and execute **Edit –Properties**. Now select the degree of rotation from the property edit form.

4. To Move a component, Select the component and execute **Edit -Move** command.

Making interconnection

1. Execute **Connectivity –Nets – Show/Hide selected Incomplete Nets** or click the icon in the Layout Menu.

2. Move the mouse pointer over the device and click **LMB** to get the connectivity information, which shows the guide lines (or flight lines) for the inter connections of the components.

3. From the layout window execute **Create – Shape – Path/ Create wire** or **Create – Shape – Rectangle (**for vdd and gnd bar**)** and select the appropriate Layers from the **LSW** window and Vias for making the inter connections

Creating Contacts/Vias

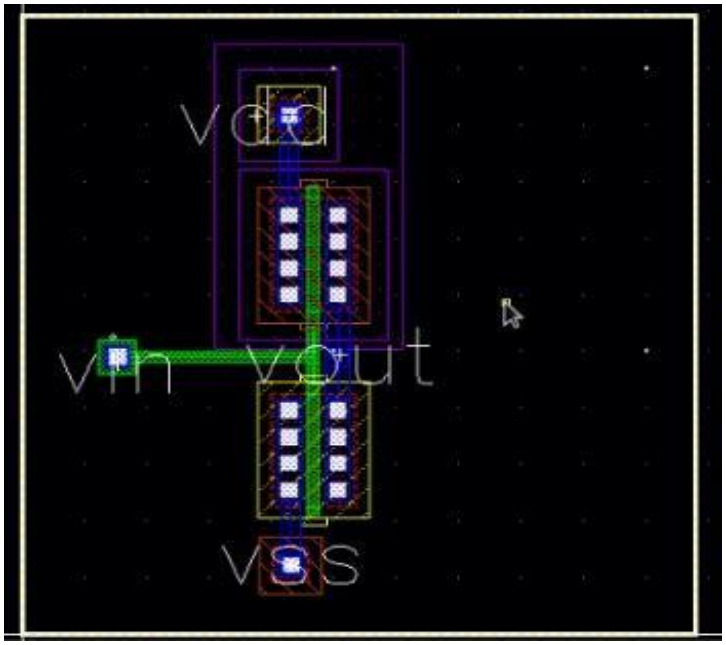
You will use the contacts or vias to make connections between two different layers.

1. Execute **Create — Via** or select command to place different Contacts, as given in below table.

Connection	Contact Type
For Metal1- Poly Connection	Metal1-Poly
For Metal1- Psubstrate Connection	Metal1-Psub
For Metal1- Nwell Connection	Metal1-Nwell

Saving the design

1. Save your design by selecting **File — Save** or click to save the layout, and layout should appear as below.



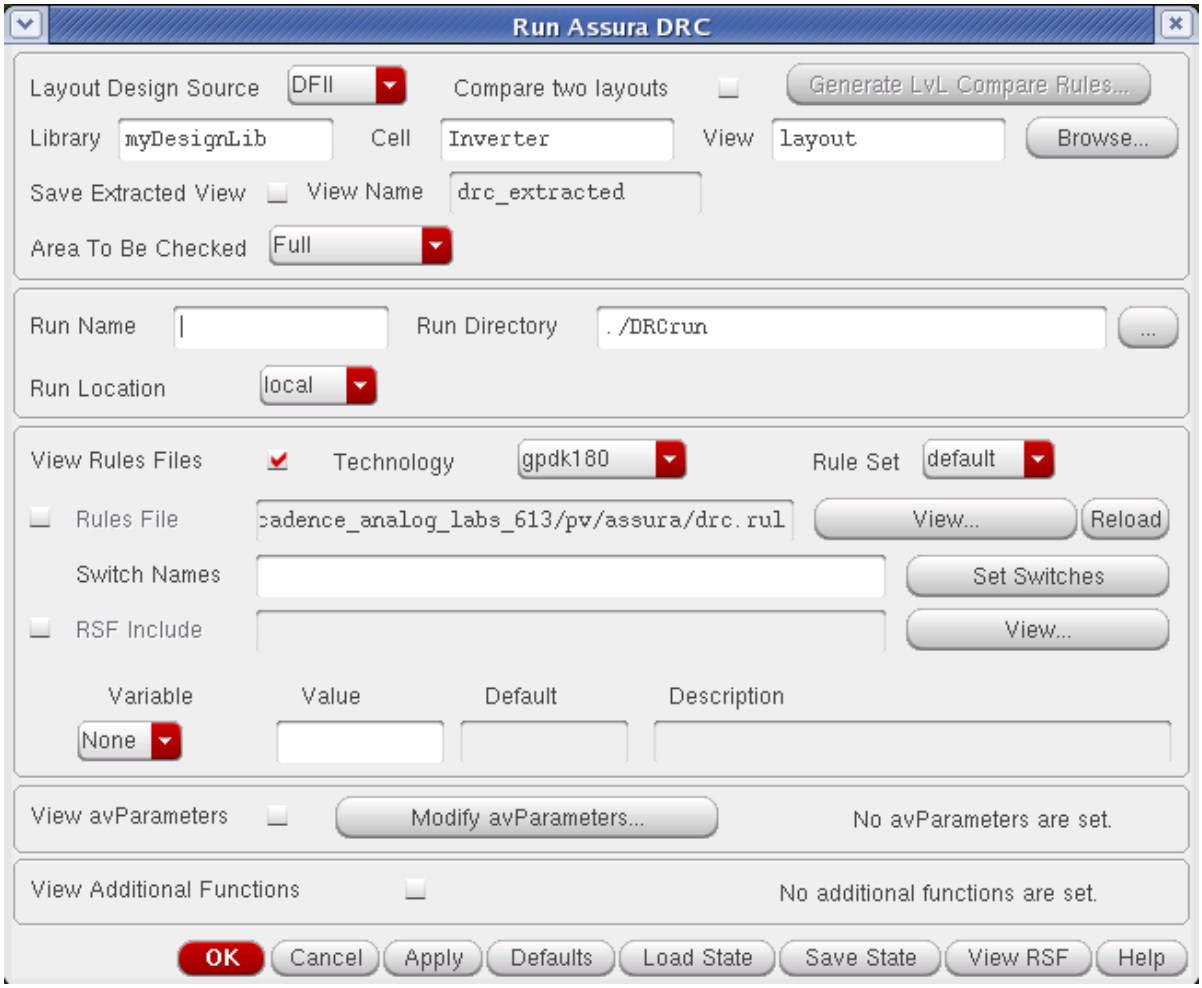
Physical Verification

Assura DRC

Running a DRC

1. Open the Inverter layout form the CIW or library manger if you have closed that. Press **shift – f** in the layout window to display all the levels.

2. Select **Assura - Run DRC** from layout window.
The DRC form appears. The Library and Cellname are taken from the current design window, but rule file may be missing. Select the Technology as **gpdk180.** This automatically loads the rule file.Your DRC form should appear like this



3. Click **OK** to start DRC.

4. A Progress form will appears. You can click on the watch log file to see the log file.

5. When DRC finishes, a dialog box appears asking you if you want to view your DRC results, and then click **Yes** to view the results of this run.

6. If there any DRC error exists in the design **View Layer Window** (VLW) and **Error Layer Window** (ELW) appears. Also the errors highlight in the design itself.

7. Click **View – Summary** in the ELW to find the details of errors.

8. You can refer to rule file also for more information, correct all the DRC errors and **Re – run** the DRC.

9. If there are no errors in the layout then a dialog box appears with **No DRC errors found** written in it, click on **close** to terminate the DRC run.

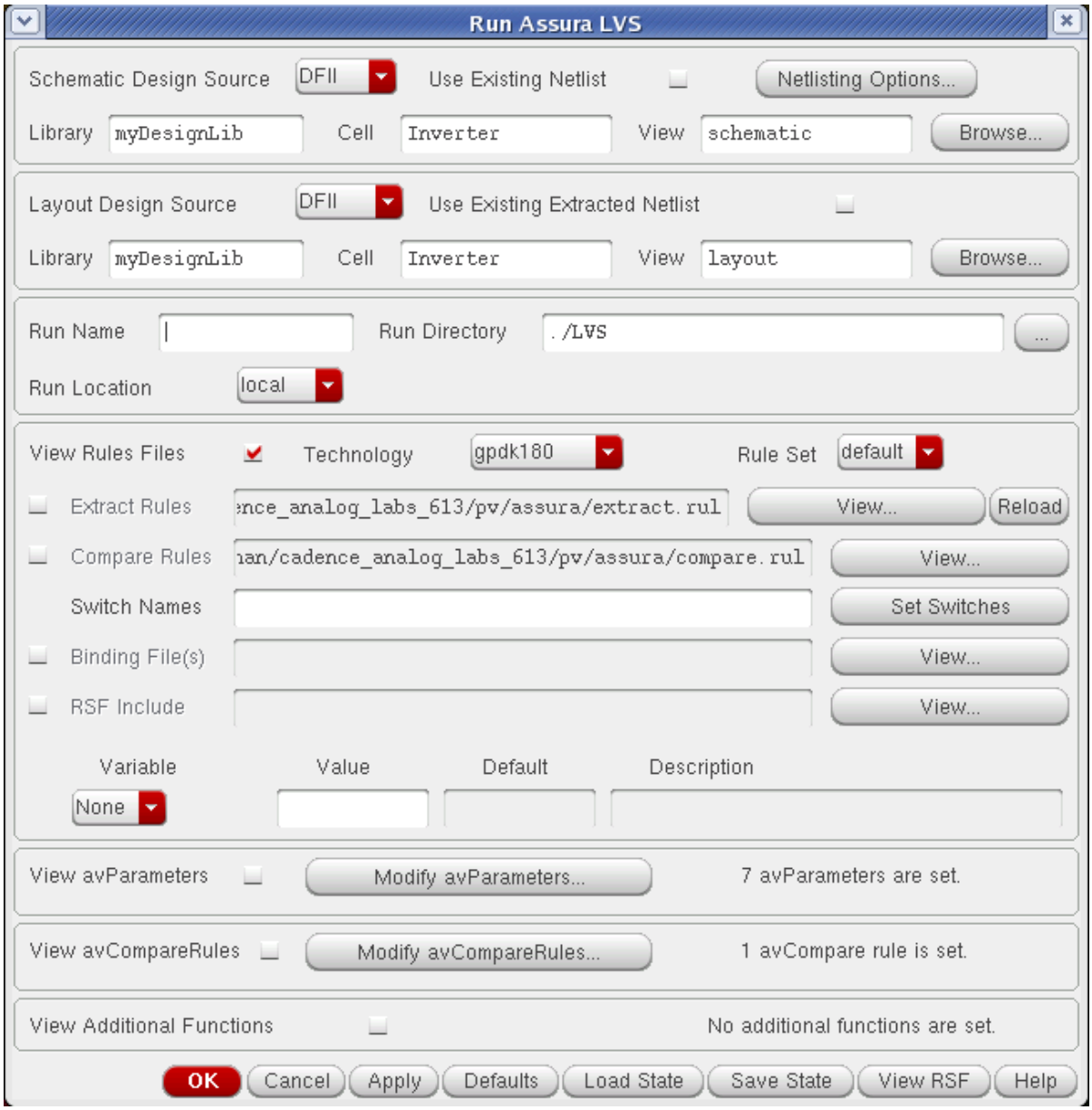
ASSURA LVS

In this section we will perform the LVS check that will compare the schematic netlist and the layout netlist.

Running LVS

1. Select **Assura – Run LVS** from the layout window.
The Assura Run LVS form appears. It will automatically load both the schematic and layout view of the cell.

2. Change the following in the form and click **OK**.



3. The LVS begins and a Progress form appears.

4. If the schematic and layout matches completely, you will get the form displaying **Schematic and Layout Match**.

5. If the schematic and layout do not matches, a form informs that the LVS completed successfully and asks if you want to see the results of this run.

6. Click **Yes** in the form.
LVS debug form appears, and you are directed into LVS debug environment.

7. In the **LVS debug form** you can find the details of mismatches and you need to correct all those mismatches and **Re – run** the LVS till you will be able to match the schematic with layout.

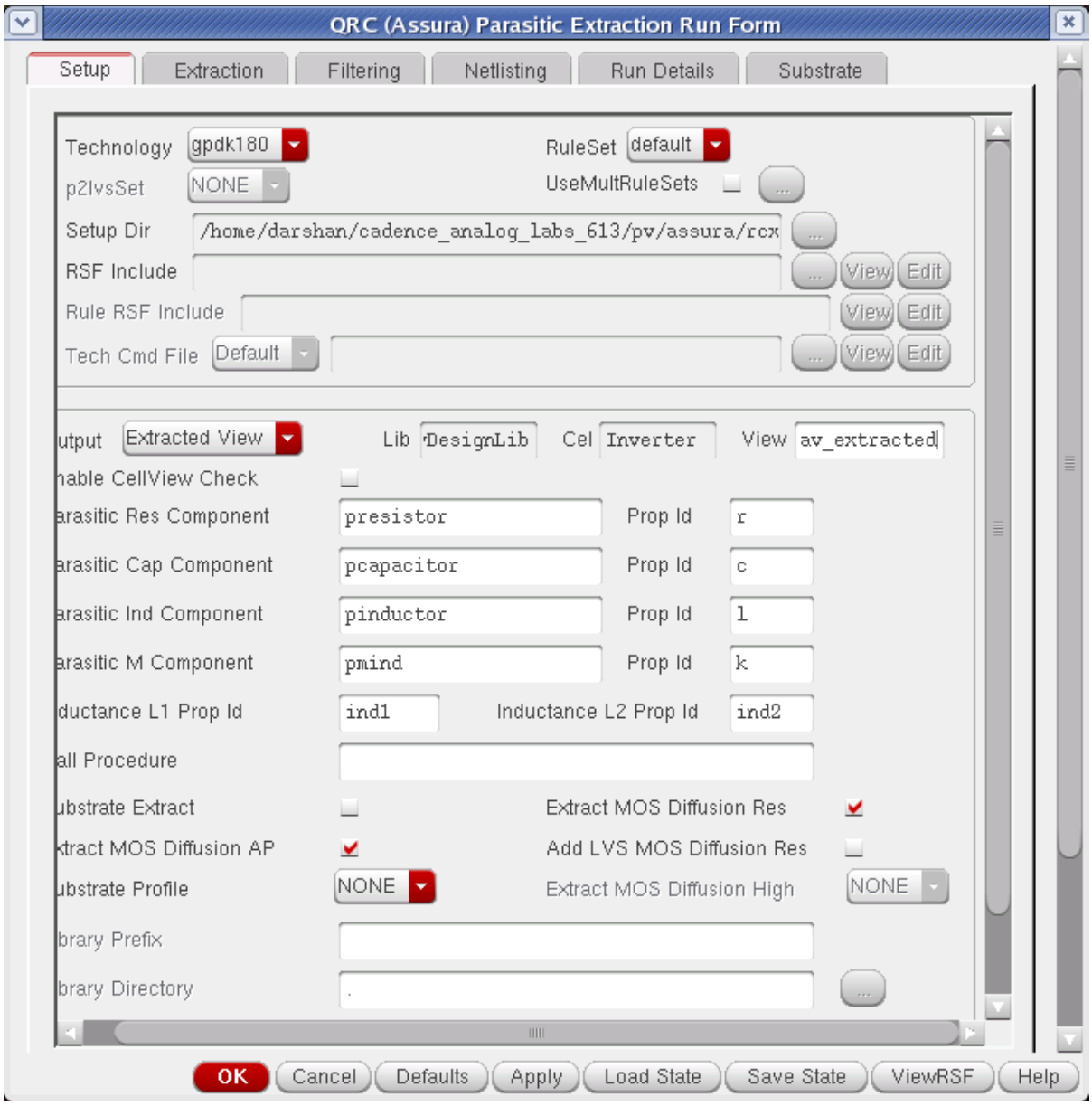
Assura RCX

In this section we will extract the RC values from the layout and perform analog circuit simulation on the designs extracted with RCX.
Before using RCX to extract parasitic devices for simulation, the layout should match with schematic completely to ensure that all parasites will be backannoted to the correct schematic nets.

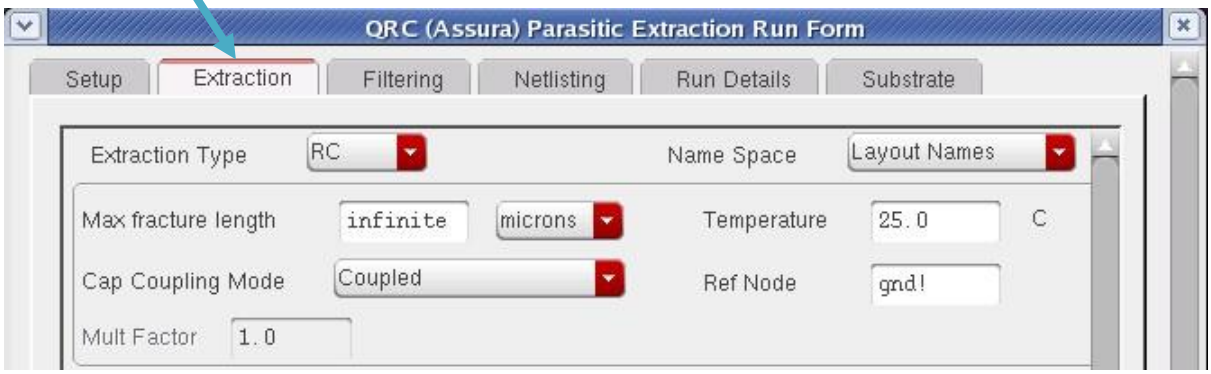
Running RCX

1. From the layout window execute **Assura – Run RCX**.

2. Change the following in the Assura parasitic extraction form. Select **output** type under **Setup** tab of the form.



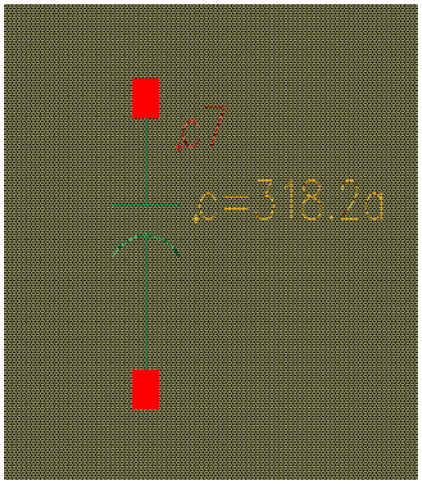
3. In the **Extraction** tab of the form, choose Extraction type, Cap Coupling Mode and specify the Reference node for extraction.

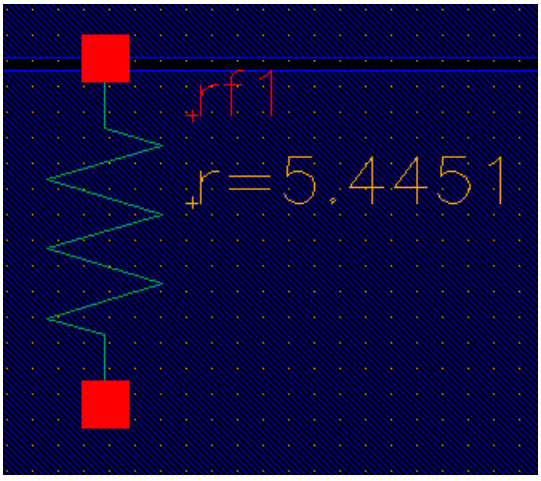


4. Click **OK** in the Assura parasitic extraction form when done.
The RCX progress form appears, in the progress form click **Watch log file** to see the output log file.

5. When RCX completes, a dialog box appears, informs you that **Assura RCX run Completed successfully.**

6. You can open the **av_extracted** view from the library manager and view the parasitic.



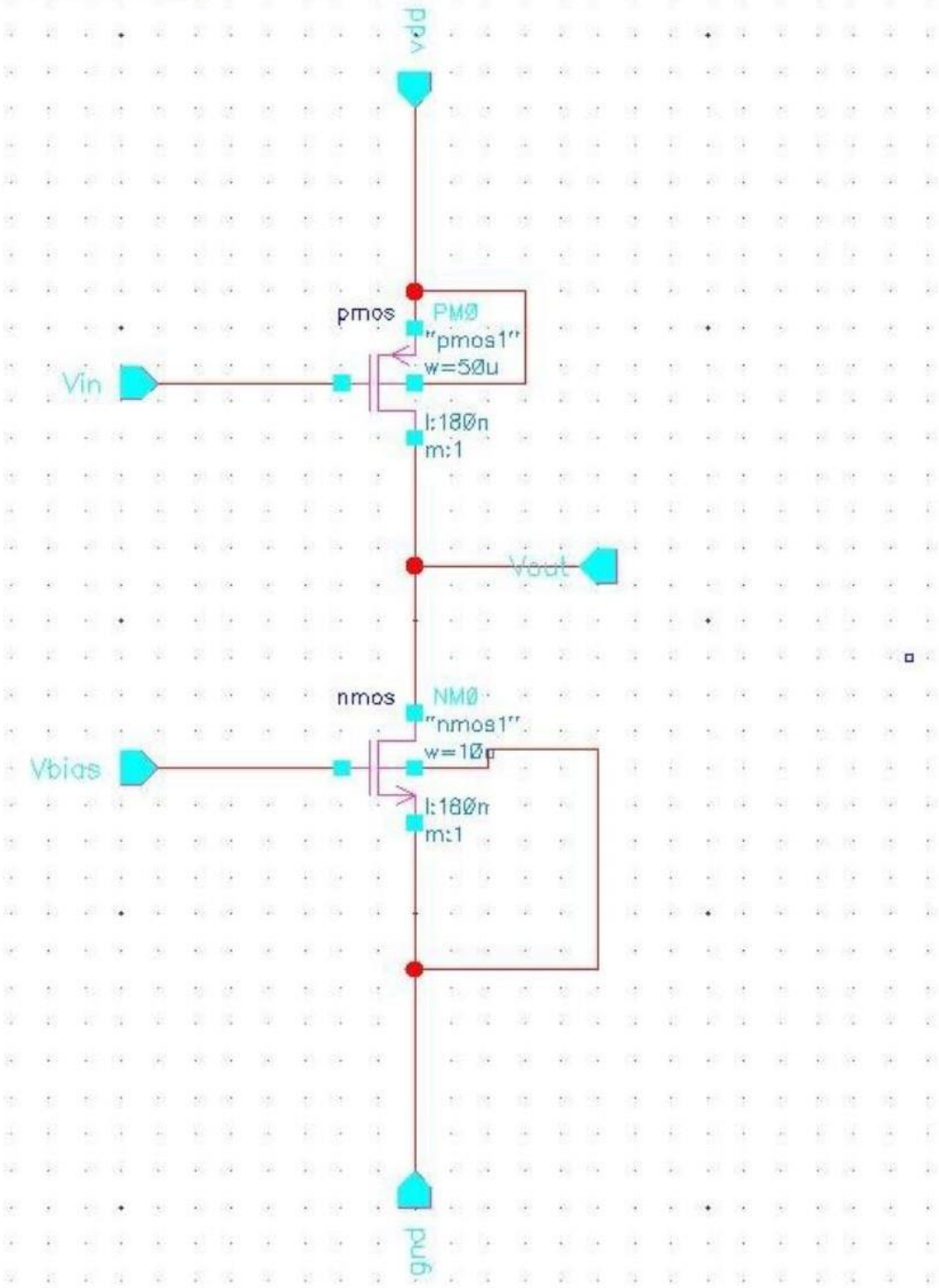


RESULT:
a. The schematic for the Inverter is drawn and verified the DC and Transient Analysis.
b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.

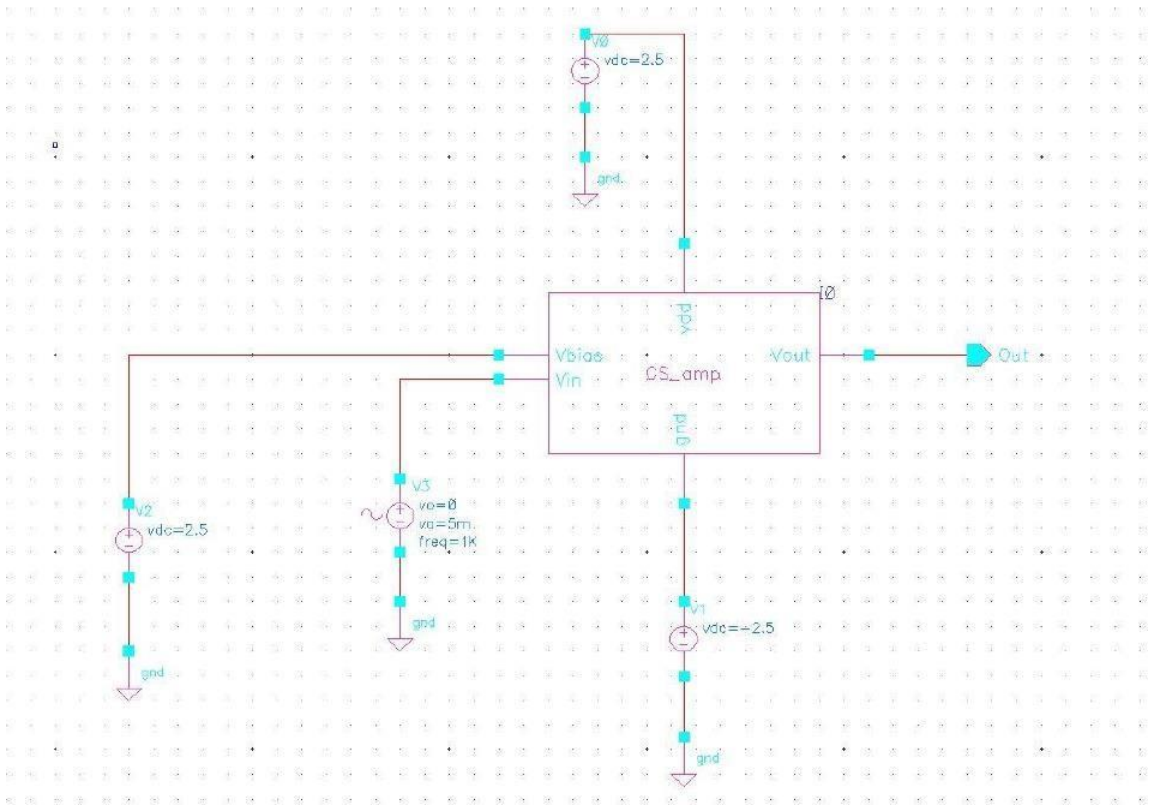
Experiment No. : 7	Date : / / .
Design of Common source amplifier	

Aim: To simulate the schematic of the common source amplifier, and then to perform the physical verification for the layout of the same.

Schematic csamp:



Test circuit csamp_test:



Properties:

Vdd	: vdc (analogLib)	– V0 DC Voltage: 2.5 V
Gnd	: vdc (analogLib)	– V1 DC Voltage: -2.5 V
Vbias	: vdc (analogLib)	– V2 DC Voltage: 2.5 V
Vin	: vsin (analogLib)	– V3 DC Voltage: 0 V
AC Magnitude		: 1 V
Offset Voltage		: 0 V
Amplitude		: 5m V
Frequency		: 1K Hz

Analysis Parameters:

Analysis: tran

Stop Time	: 5m
Accuracy Defaults	: moderate
Enabled	: checked

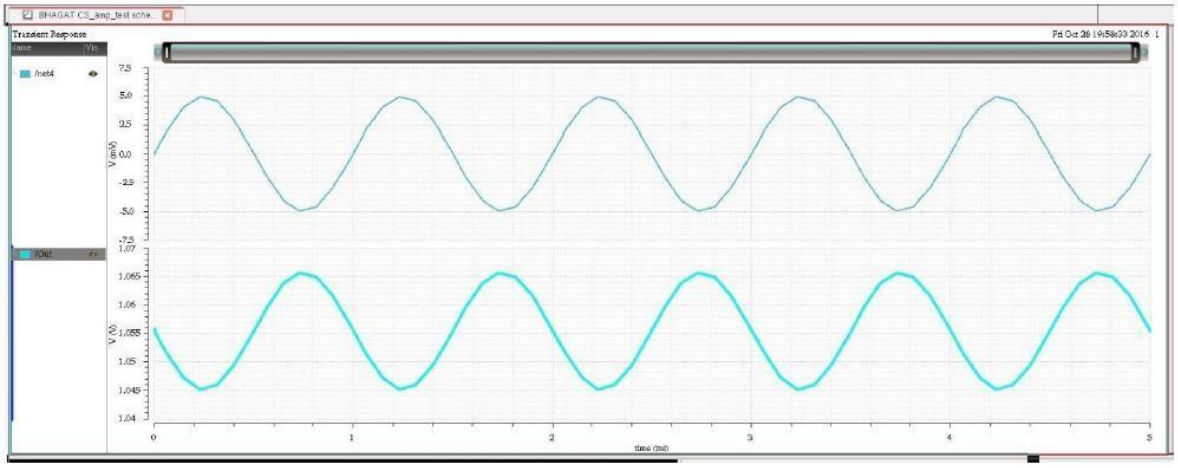
Analysis: dc

Save DC Operating Point	: checked
Component	: /V3 (select **vsin**)
Component Parameter	: dc
Sweep Range	: Start-Stop
Start	: -5
Stop	5
Sweep Type	: Automatic
Enabled	: checked

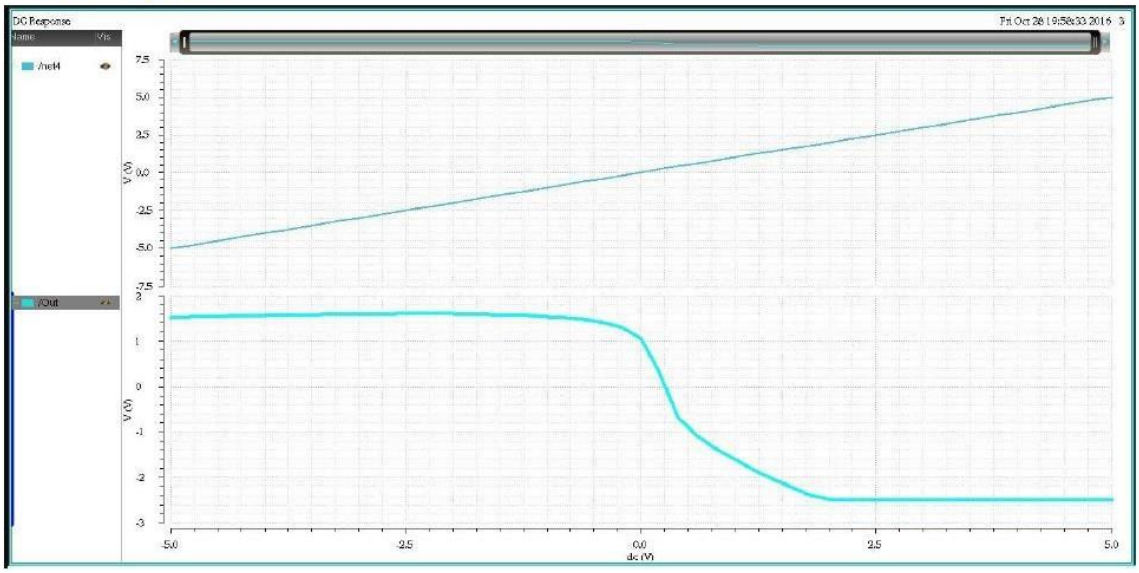
Analysis: ac

Sweep Variable	: Frequency
Sweep Range	: Start-Stop
Start	100
Stop	: 100M
Sweep Type	: Logarithmic
Points Per Decade	20
Enabled	: checked

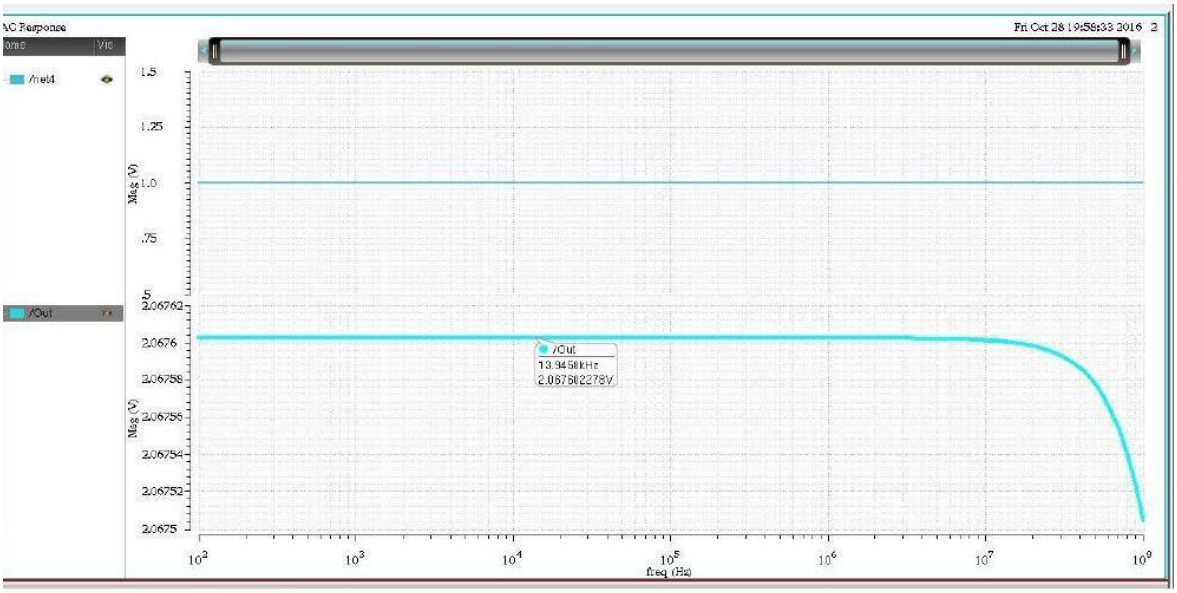
Transient Response:



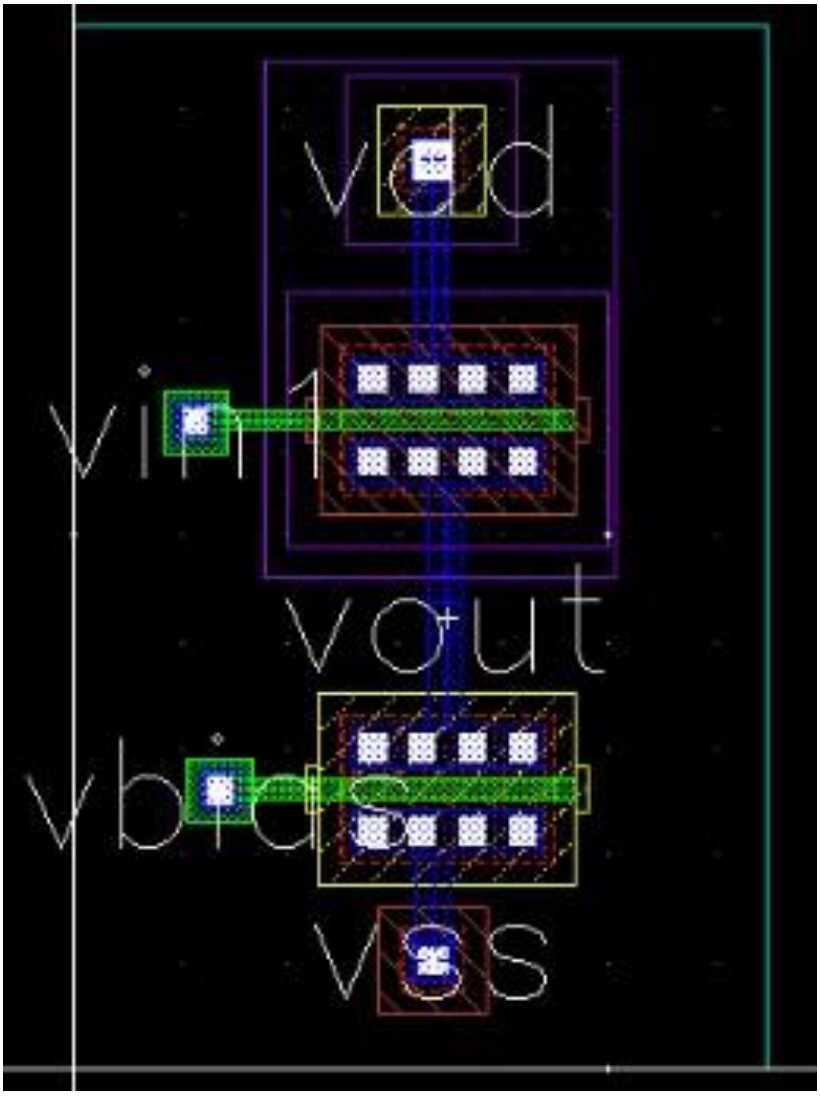
DC Response:



AC Response:



Layout View



RESULT:

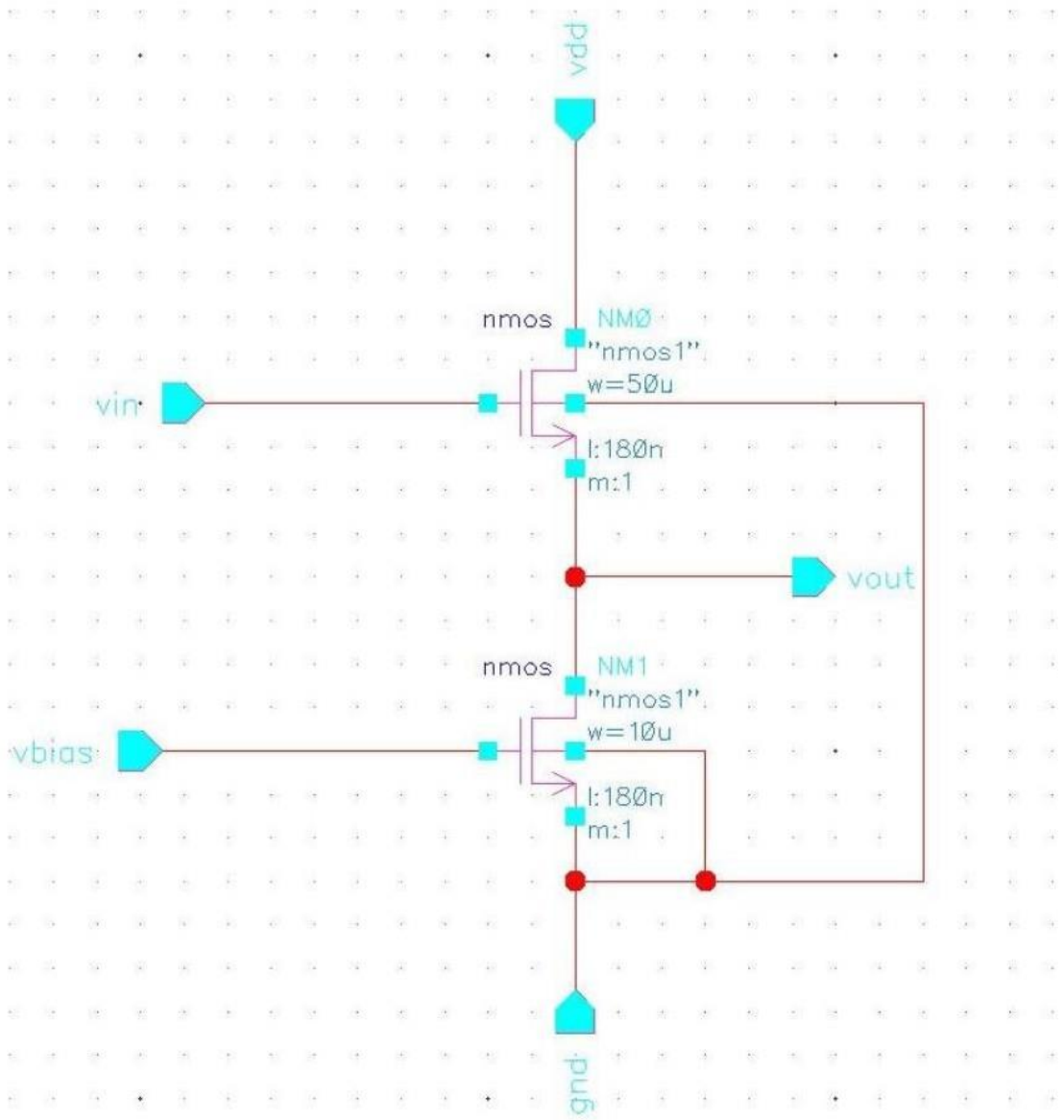
a. The schematic for the Common Source amplifier is drawn and verified the following: DC Analysis, AC Analysis and Transient Analysis.

b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.

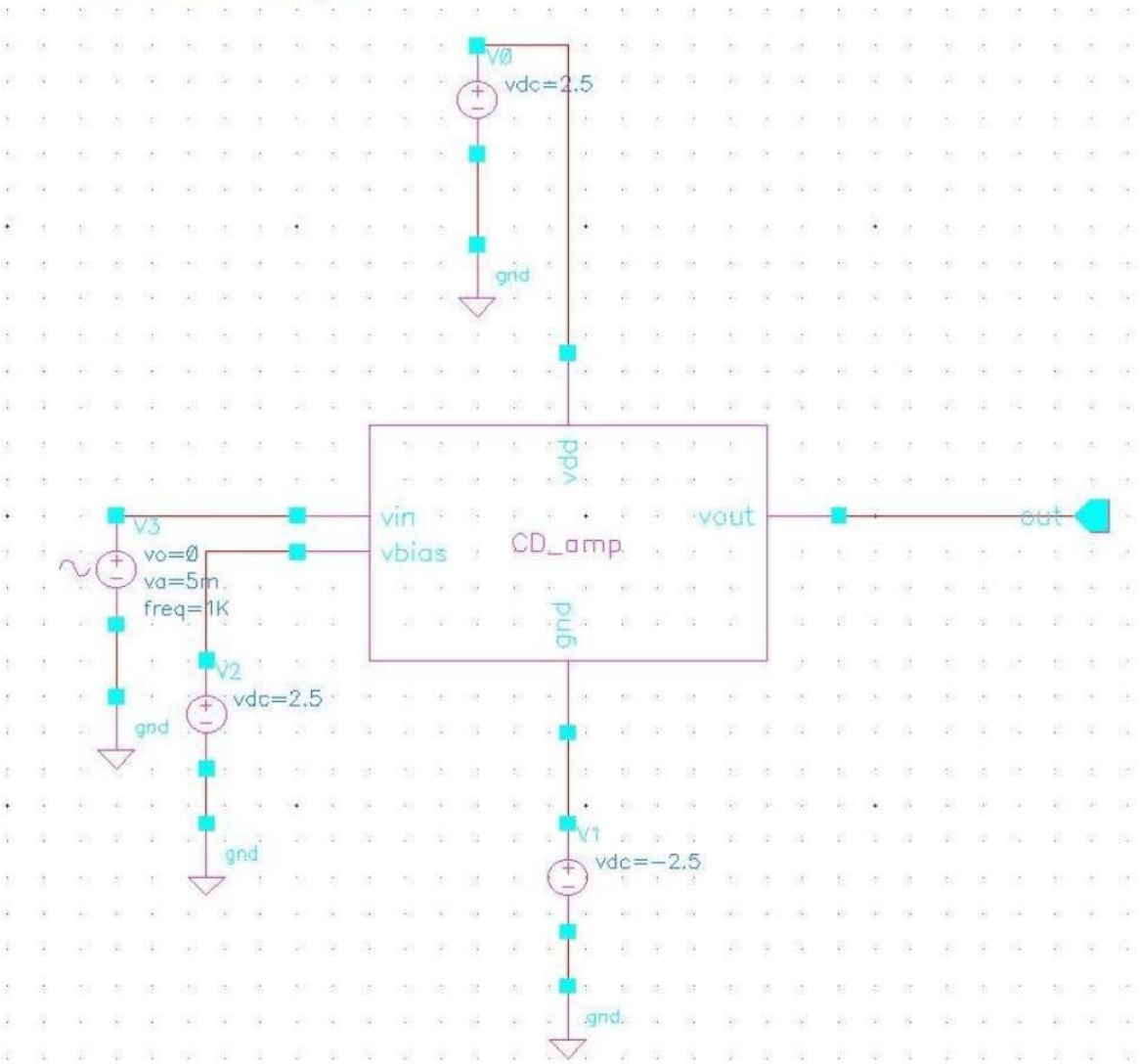
Experiment No. : 8	Date : / / .
Design of Common Drain amplifier	

Aim: To simulate the schematic of the common drain amplifier, and then to perform the physical verification for the layout of the same.

Schematic CD-amp:



Test circuit CDamp_test:



Properties:

Vdd	: vdc (analogLib)	– V0 DC Voltage: 2.5 V
Gnd	: vdc (analogLib)	– V1 DC Voltage: -2.5 V
Vbias	: vdc (analogLib)	– V2 DC Voltage: 2.5 V
Vin	: vsin (analogLib)	– V3 DC Voltage: 0 V
AC Magnitude		: 1 V
Offset Voltage		: 0 V
Amplitude		: 5m V
Frequency		: 1K Hz

Analysis Parameters:

Analysis: tran

Stop Time	: 5m
Accuracy Defaults	: moderate
Enabled	: checked

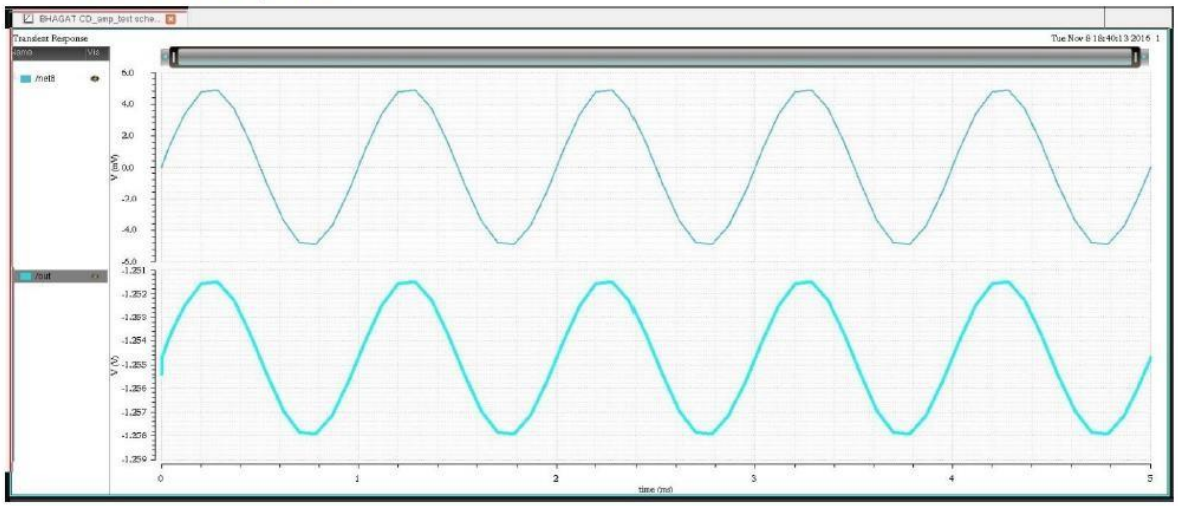
Analysis: dc

Save DC Operating Point	: checked
Component	: /V3 (select **vsin**)
Component Parameter	: dc
Sweep Range	: Start-Stop
Start	: -5
Stop	5
Sweep Type	: Automatic
Enabled	: checked

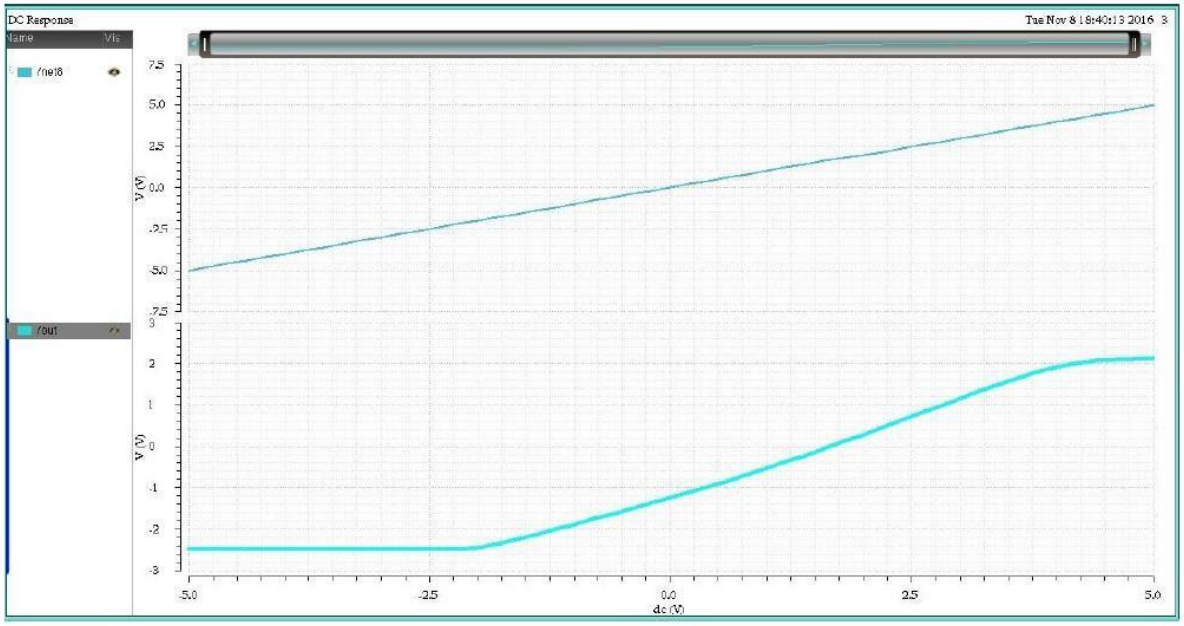
Analysis: ac

Sweep Variable	: Frequency
Sweep Range	: Start-Stop
Start	100
Stop	: 100M
Sweep Type	: Logarithmic
Points Per Decade	20
Enabled	: checked

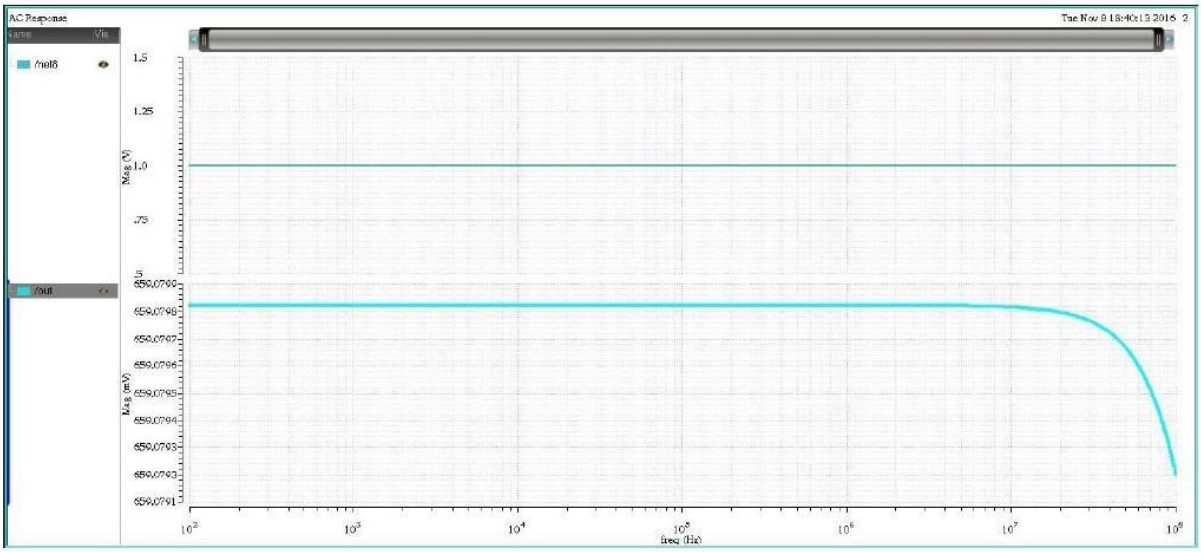
Transient Response:



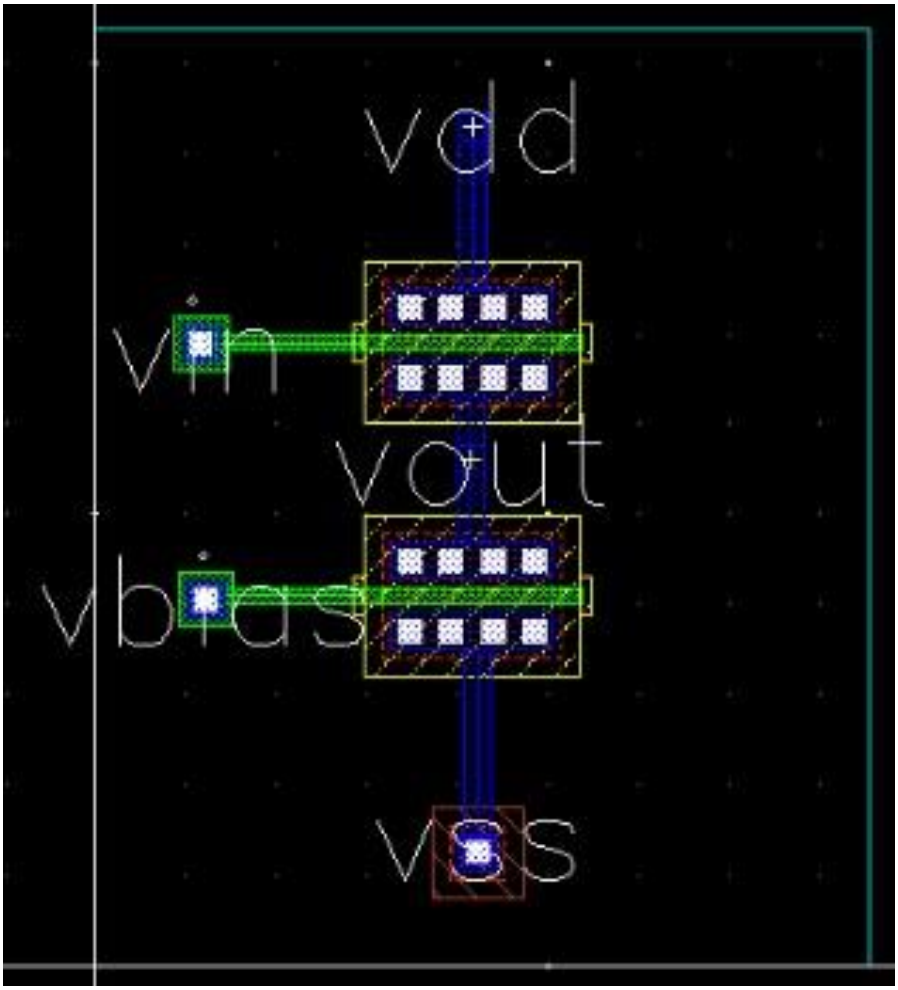
DC Response:



AC Response:



Layout View



RESULT:

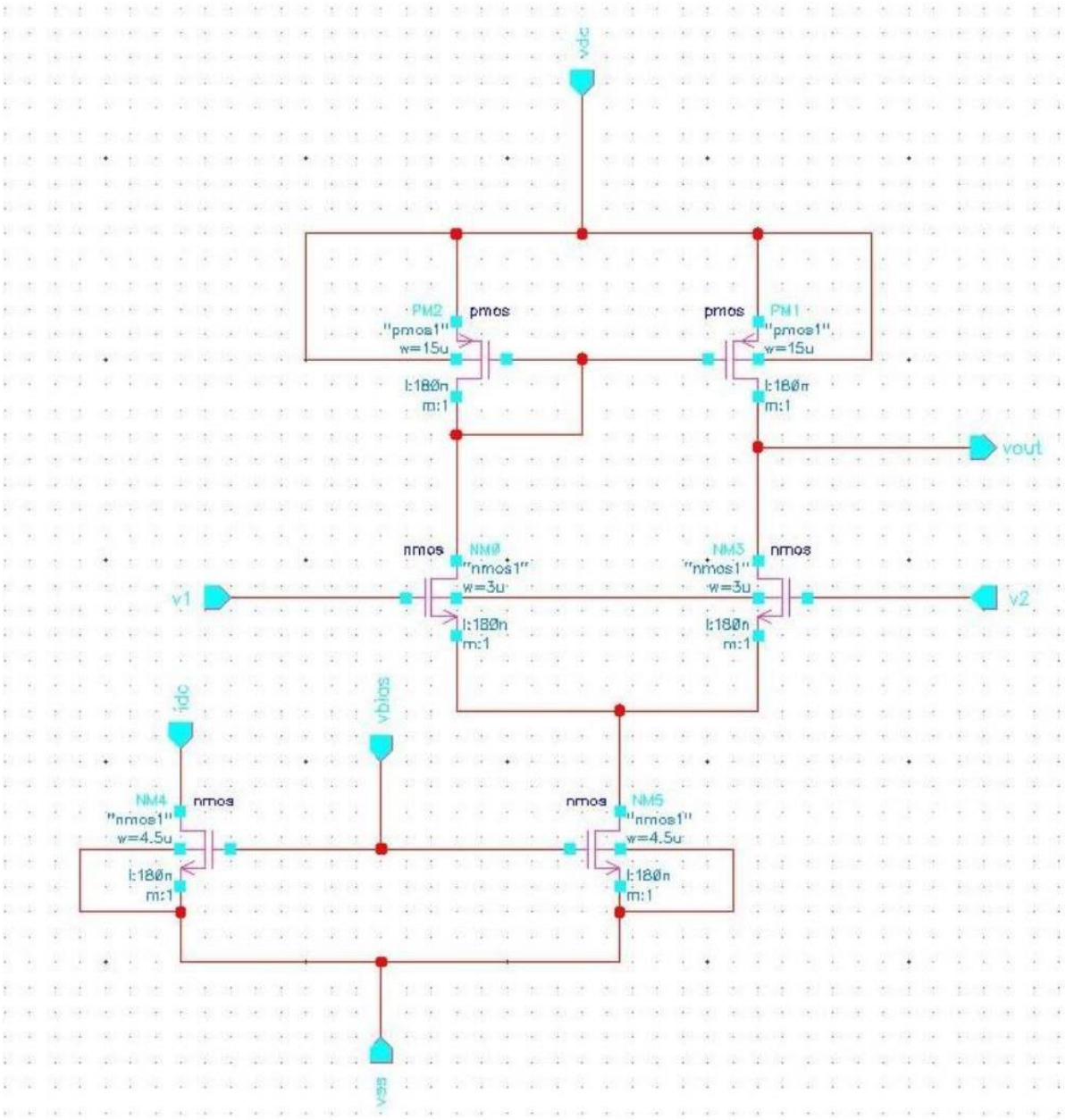
a. The schematic for the Common Drain amplifier is drawn and verified the following: DC Analysis, AC Analysis and, Transient Analysis.

b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.

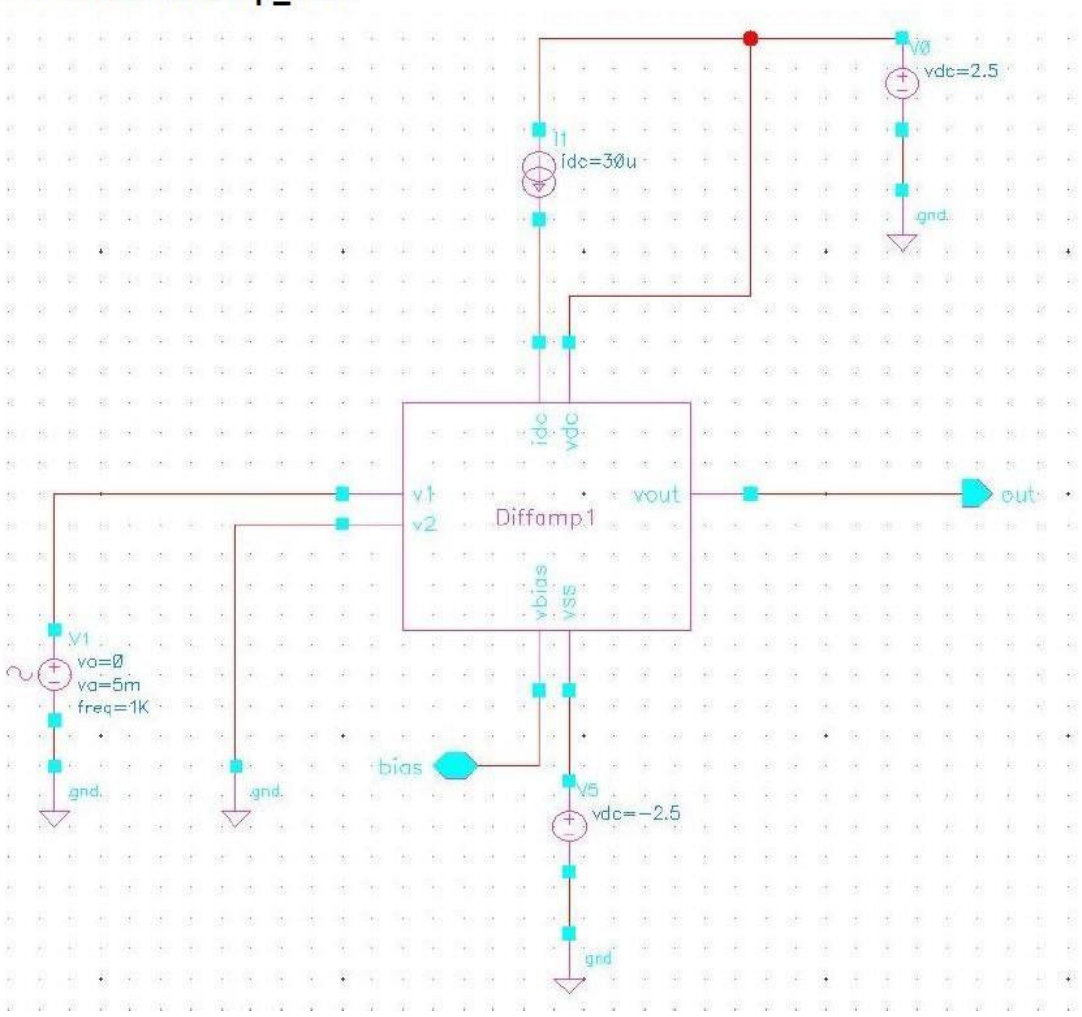
Experiment No. : 9	Date : / / .
Design of single stage differential amplifier	

Aim: To simulate the schematic of the differential amplifier, and then to perform the physical verification for the layout of the same.

Schematic Diffamp:



Test Circuit Diffamp_test:



Properties:

Vdd	: vdc (analogLib)	– V0 DC Voltage: 2.5 V
Gnd	: vdc (analogLib)	– V5 DC Voltage: -2.5 V
Vbias		: no connection
v2		: gnd (analogLib)
v1	: vsin (analogLib)	– V1 DC Voltage: 0 V
AC Magnitude		: 1 V
Offset Voltage		: 0 V
Amplitude		: 5m V
Frequency		: 1K Hz
Idc	: idc (analogLib)	– I1 DC Current: 30u A

Analysis Parameters:

Analysis: tran

Stop Time	: 5m
Accuracy Defaults	: moderate
Enabled	: checked

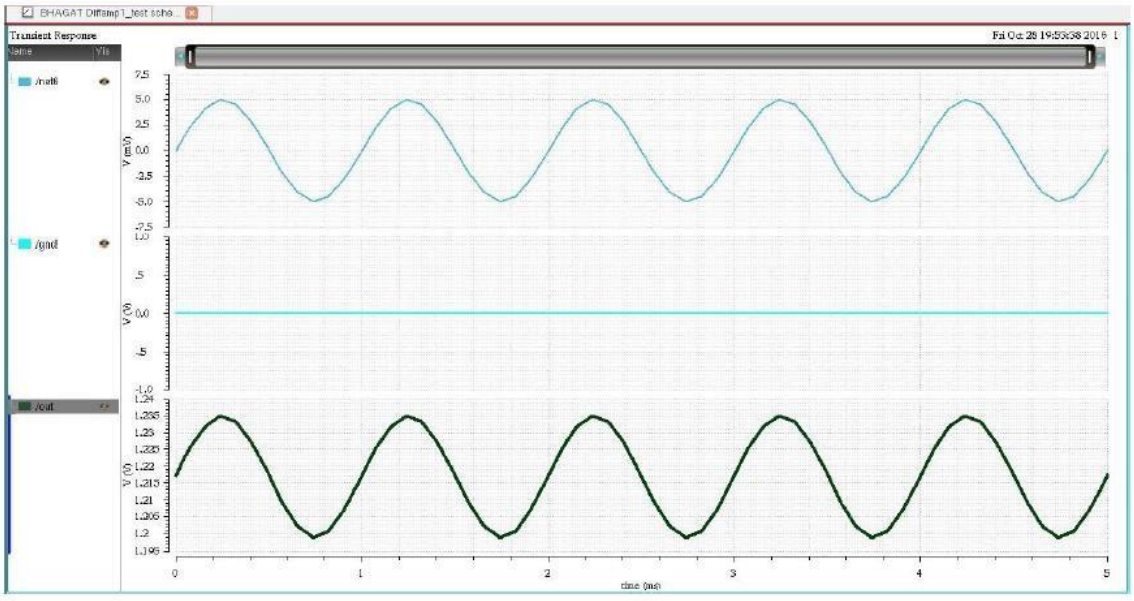
Analysis: dc

Save DC Operating Point	: checked
Component	: /V1 (select **vsin**)
Component Parameter	: dc
Sweep Range	: Start-Stop
Start	: -5
Stop	5
Sweep Type	: Automatic
Enabled	: checked

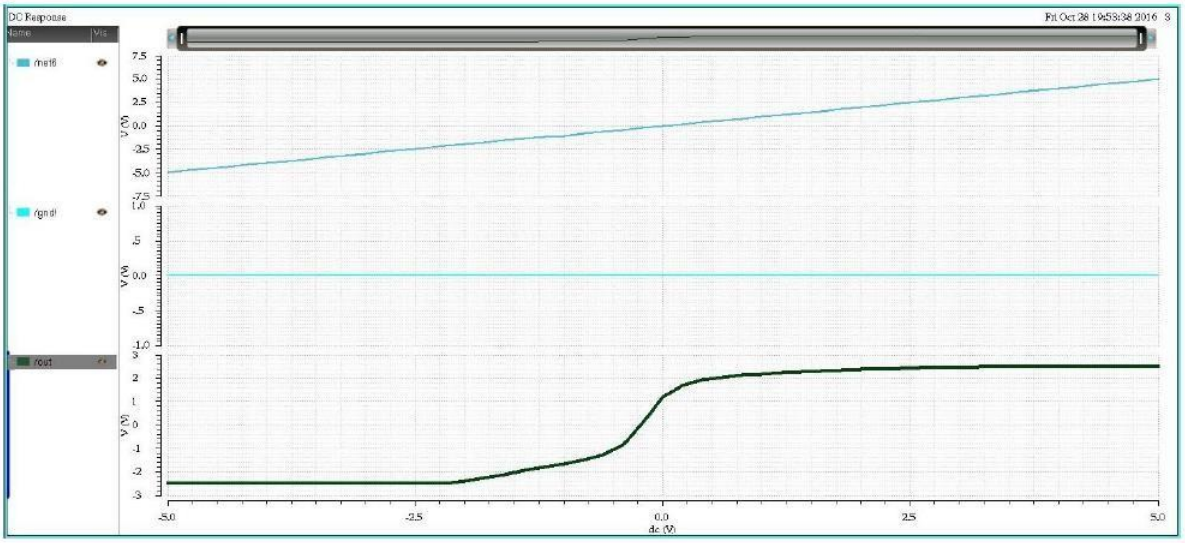
Analysis: ac

Sweep Variable	: Frequency
Sweep Range	: Start-Stop
Start	: 0.1
Stop	100
Sweep Type	: Logarithmic
Points Per Decade	20
Enabled	: checked

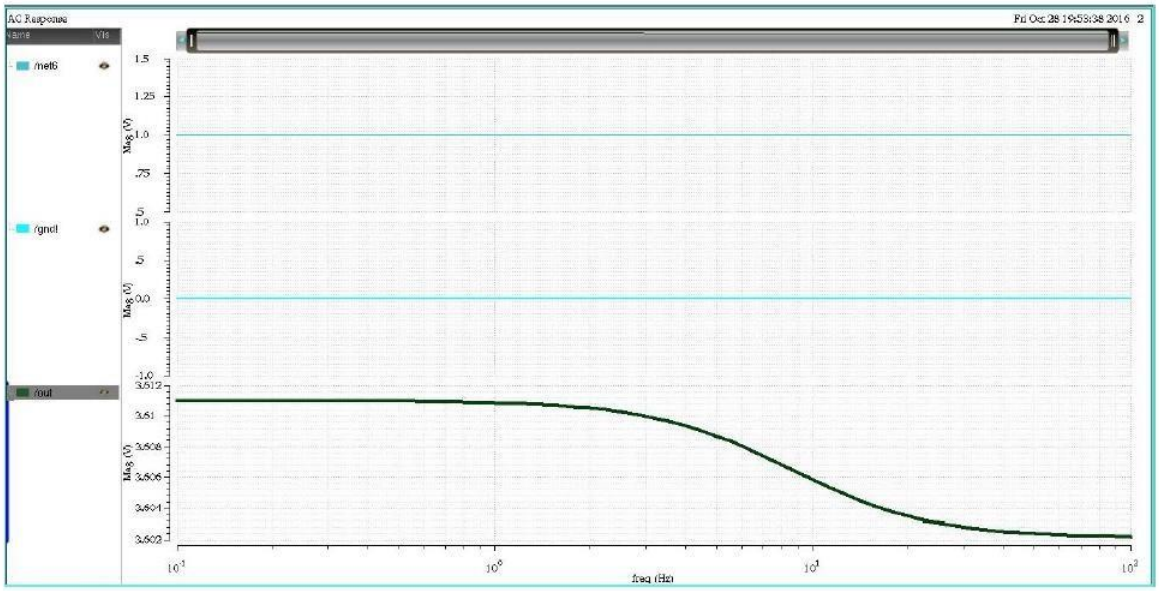
Transient Response:



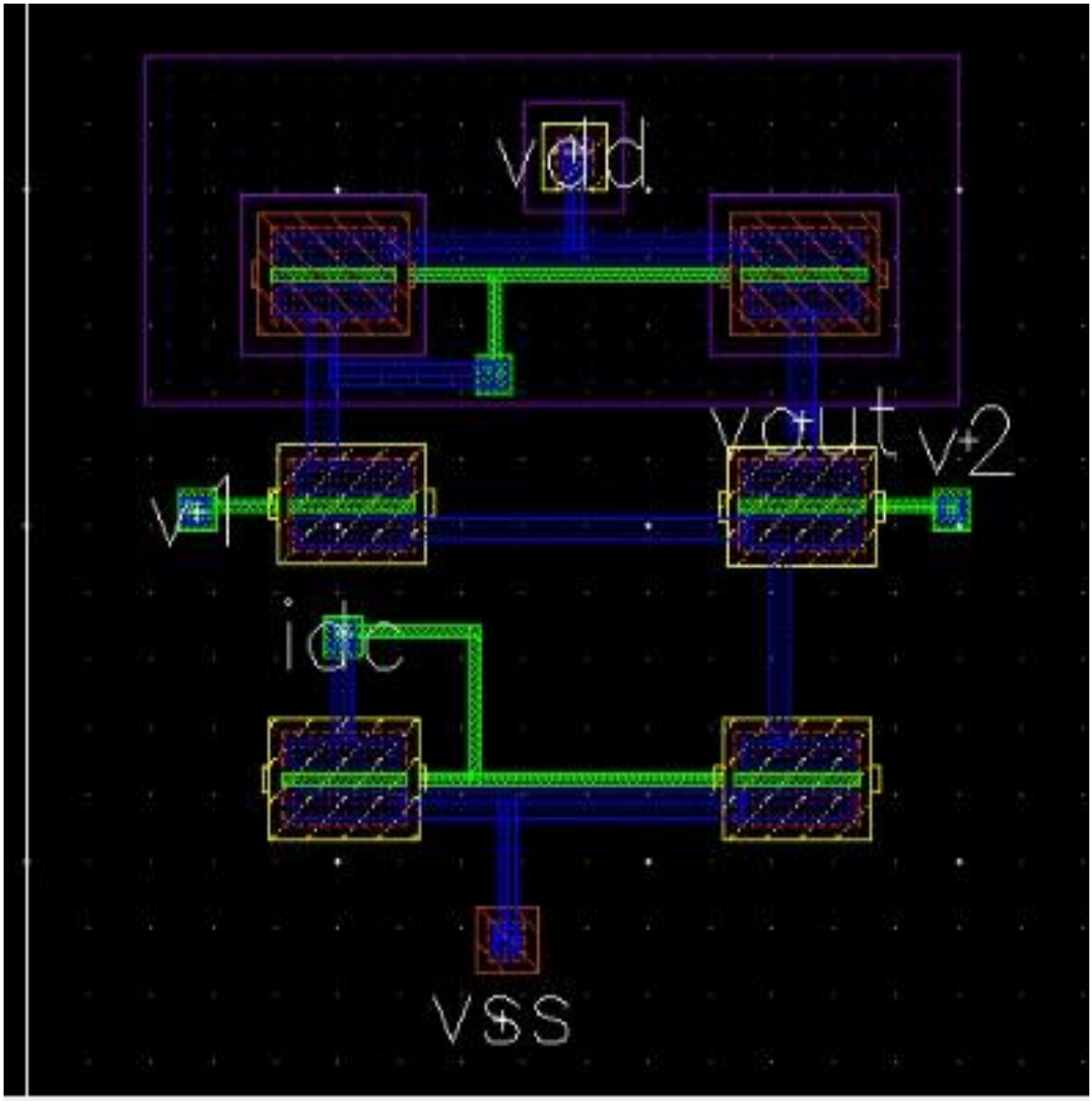
DC Response:



AC Response:



Layout View



RESULT:

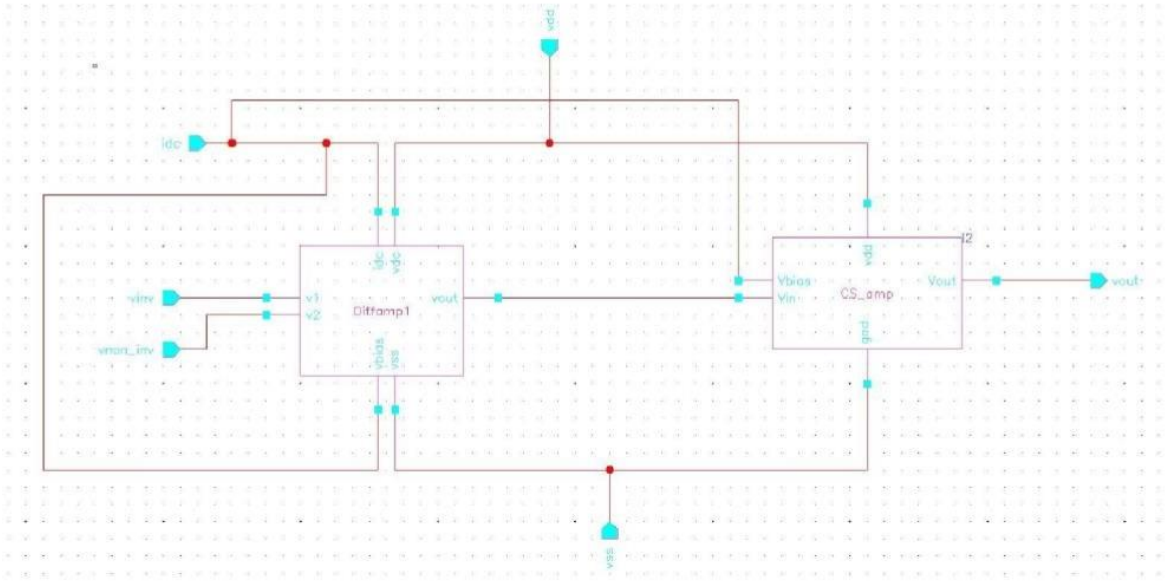
a. The schematic for the Differential amplifier is drawn and verified the following: DC Analysis, AC Analysis and Transient Analysis.

b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.

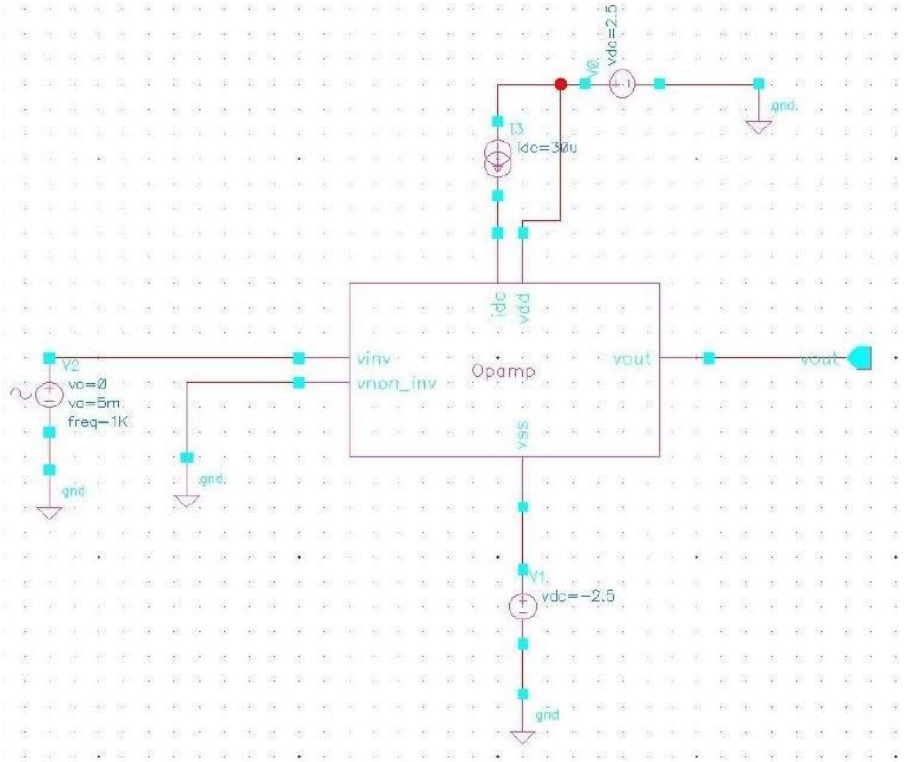
Experiment No. : 10	Date : / / .
Design of an op-amp with given specification	

Aim: To simulate the schematic of the operational amplifier, and then to perform the physical verification for the layout of the same.

Schematic Op-amp:



Test Circuit Opamp_test:



Properties:

Vdd	: vdc (analogLib)	– V0 DC Voltage: 2.5 V
Vss	: vdc (analogLib)	– V1 DC Voltage: -2.5 V
vnon_inv	: gnd (analogLib)	
vinv	: vsin (analogLib)	– V2 DC Voltage: 0 V
AC Magnitude		: 1 V
Offset Voltage		: 0 V
Amplitude		: 5m V
Frequency		: 1K Hz
Idc	: idc (analogLib)	– I3 DC Current: 30u A

Analysis Parameters:

Analysis: tran

Stop Time	: 5m
Accuracy Defaults	: moderate
Enabled	: checked

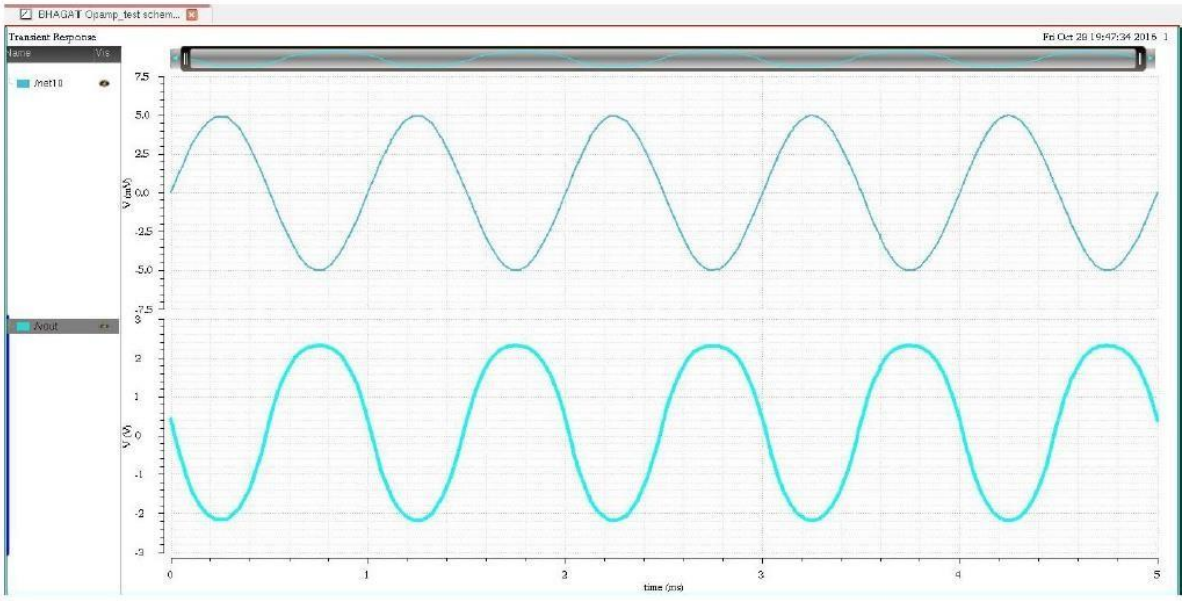
Analysis: dc

Save DC Operating Point	: checked
Component	: /V2 (select **vsin**)
Component Parameter	: dc
Sweep Range	: Start-Stop
Start	: -5
Stop	5
Sweep Type	: Automatic
Enabled	: checked

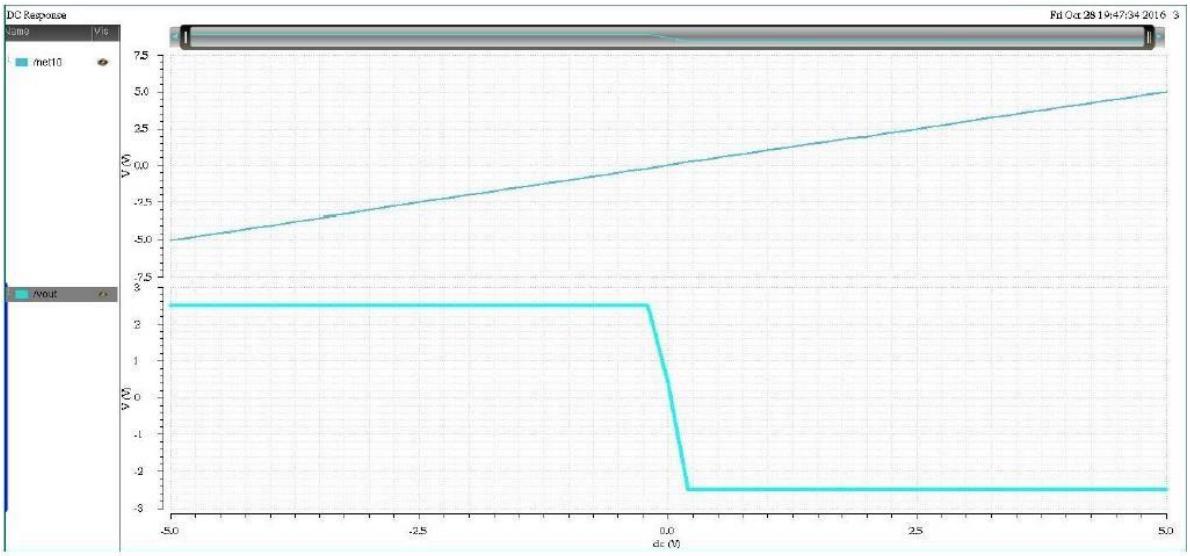
Analysis: ac

Sweep Variable	: Frequency
Sweep Range	: Start-Stop
Start	150
Stop	: 100M
Sweep Type	: Logarithmic
Points Per Decade	20
Enabled	: checked

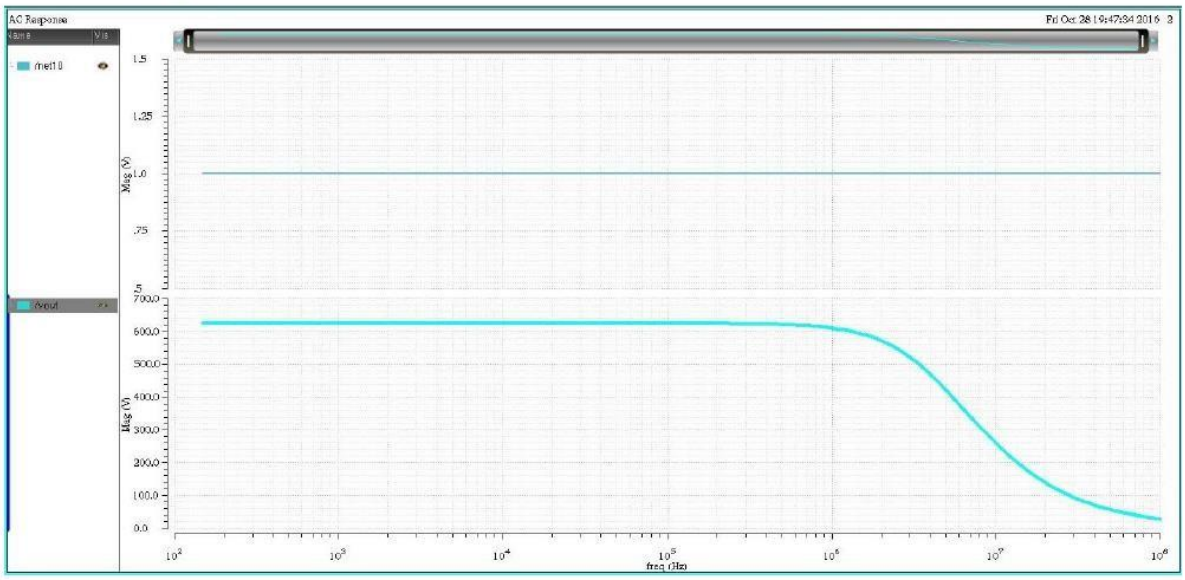
Transient response:



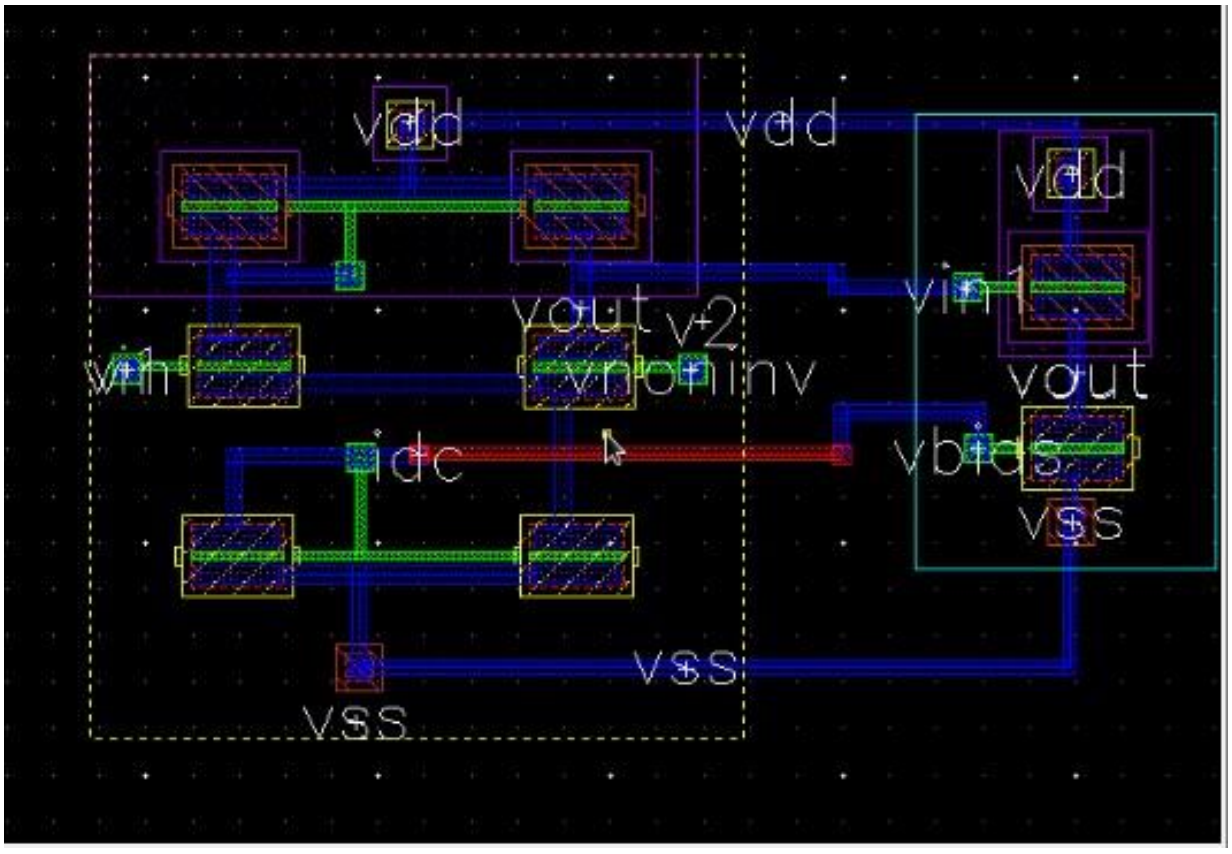
DC response:



AC Response:



Layout View



RESULT:

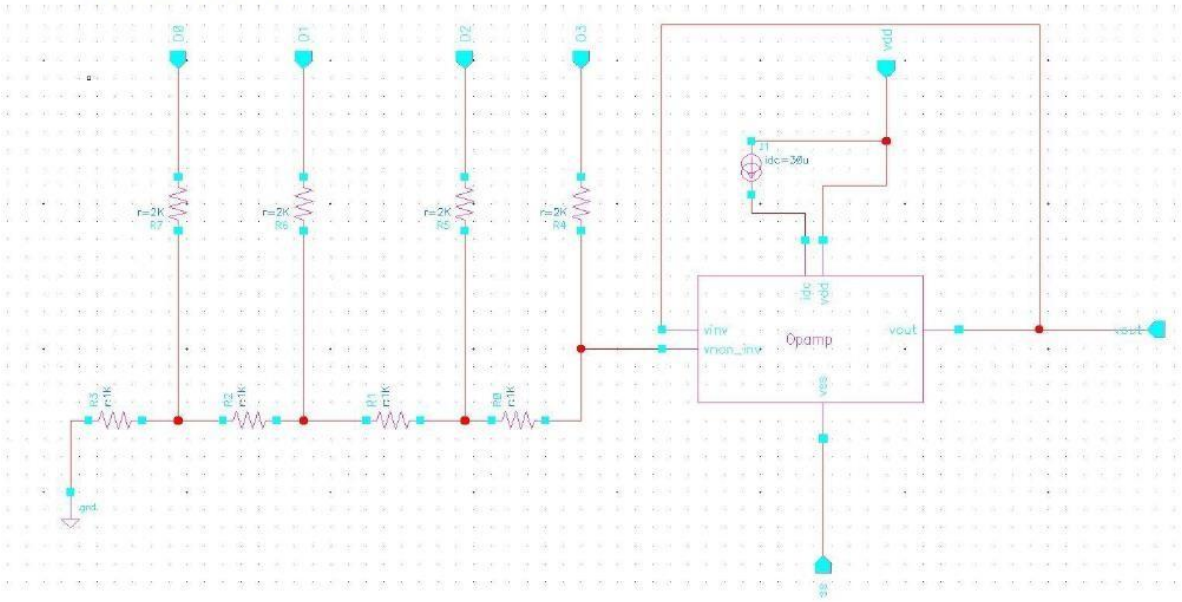
a. The schematic for the Operational amplifier is drawn and verified the following: DC Analysis, AC Analysis and Transient Analysis.

b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.

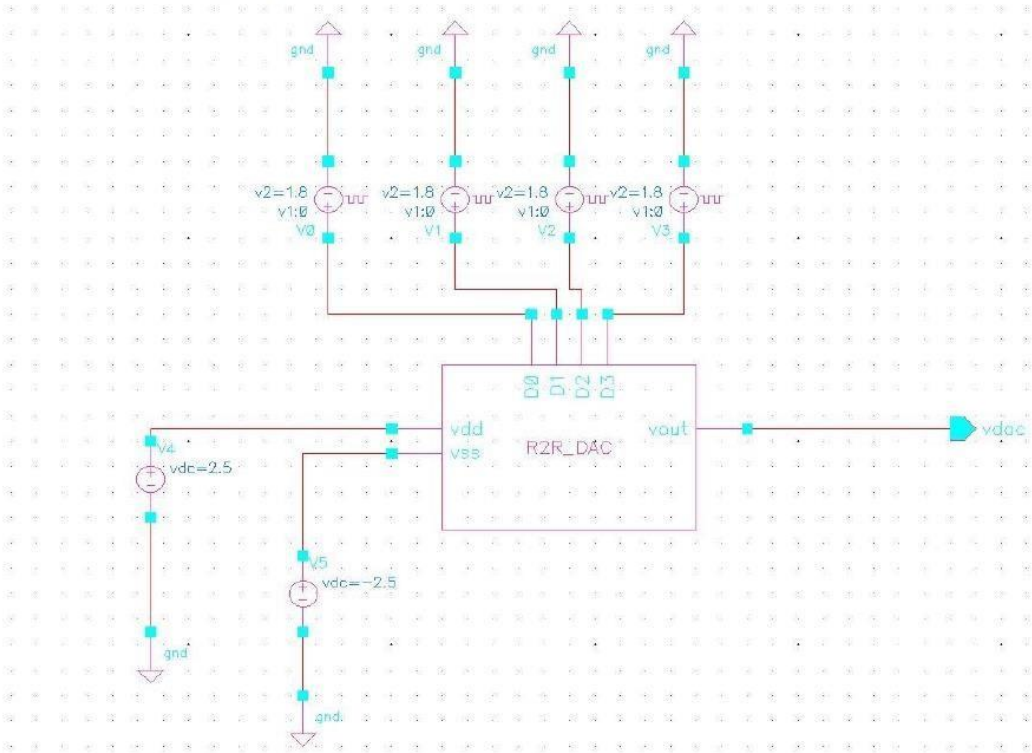
Experiment No. : 11	Date : / / .
Design a 4 bit R-2R based DAC for the given specification	

Aim: To simulate the schematic of the 4 bit R-2R based DAC, and then to perform the physical verification for the layout of the same.

Schematic R2R-DAC:



Test circuit R2R_DAC_test:



Properties:

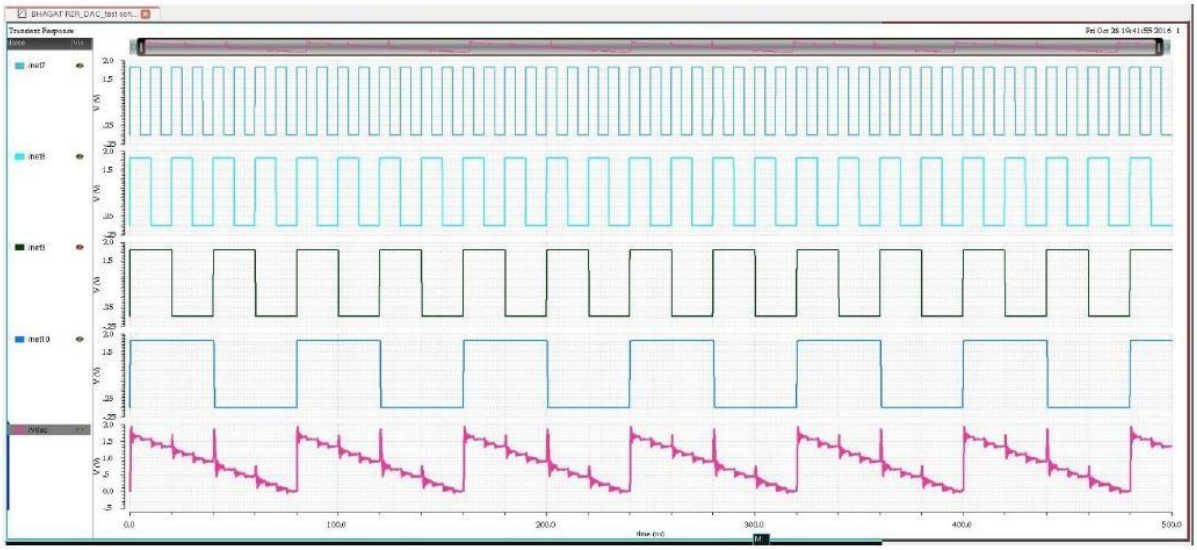
Vdd : vdc (analogLib) – V4 DC Voltage: 2.5 V
Vss : vdc (analogLib) – V5 DC Voltage: -2.5 V
D0 : vpulse(analogLib) - V0 Pulse gen
V1 = 0v, v2 = 1.8v, Ton = 5ns, T = 10ns
D1 : vpulse(analogLib) - V1 PuV1 = 0v, v2 = 1.8v, Ton = 10ns, T = 20ns
D2 : vpulse(analogLib) - V2 Pulse gen
V1 = 0v, v2 = 1.8v, Ton = 20ns, T = 40ns
D3: vpulse(analogLib) - V3 Pulse gen
V1 = 0v, v2 = 1.8v, Ton = 40ns, T = 80ns

Analysis Parameters:

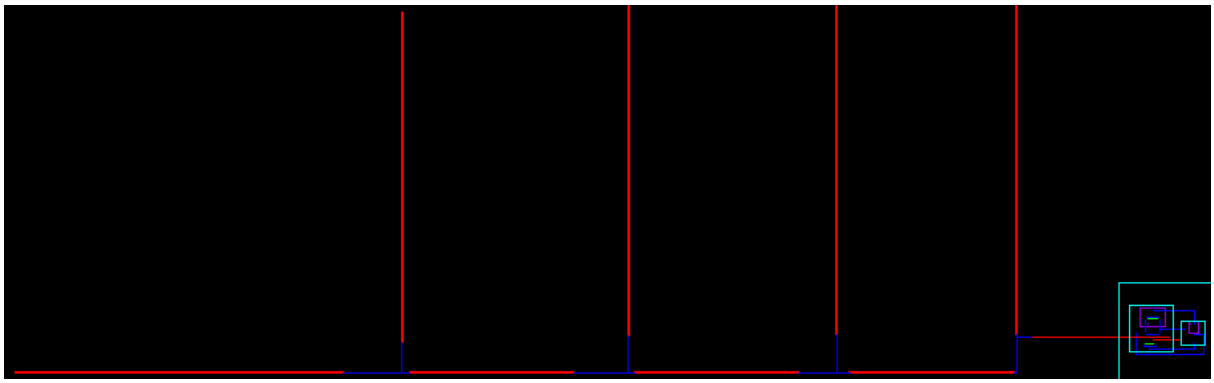
Analysis: tran

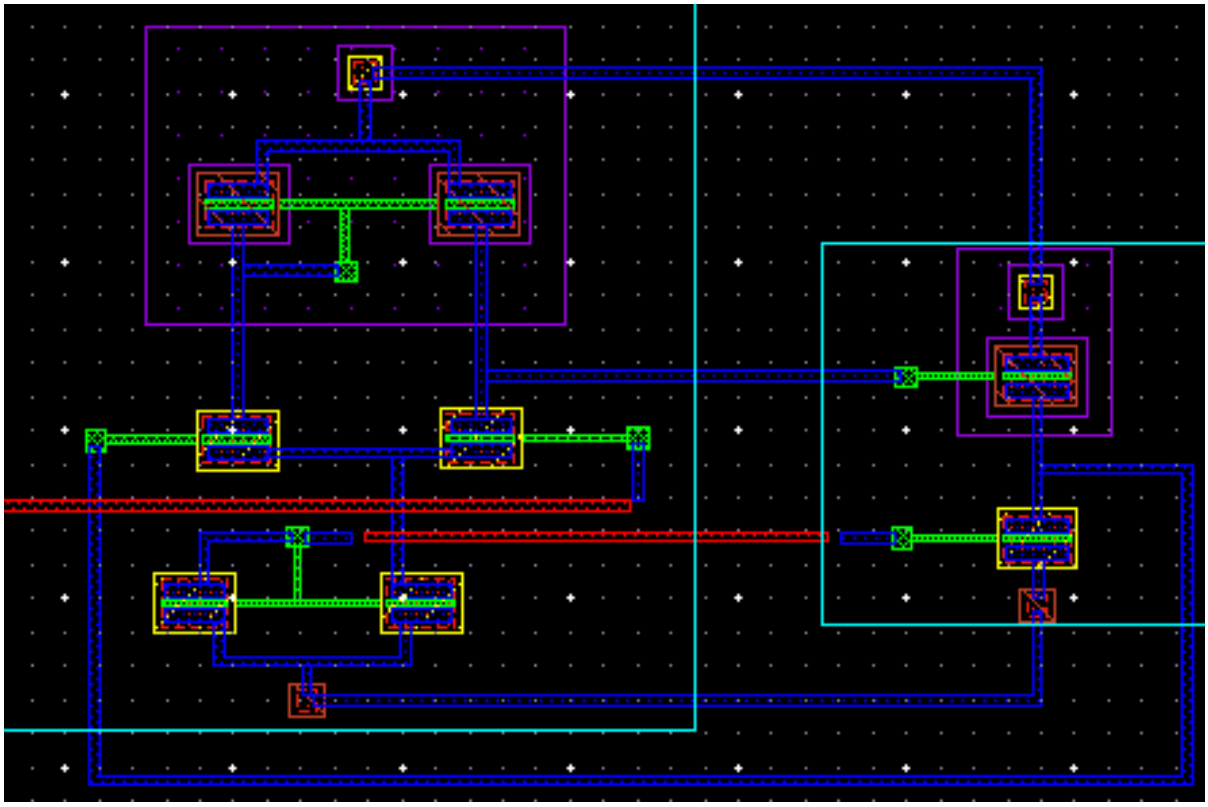
Stop Time : 5m
Accuracy Defaults : moderate
Enabled : checked

Transient response:



Layout View





RESULT:

a. The schematic for the the 4 bit R-2R based DAC is drawn and verified the Transient Analysis.

b. The Layout for the same is drawn and verified the DRC, LVS, RC Extraction.