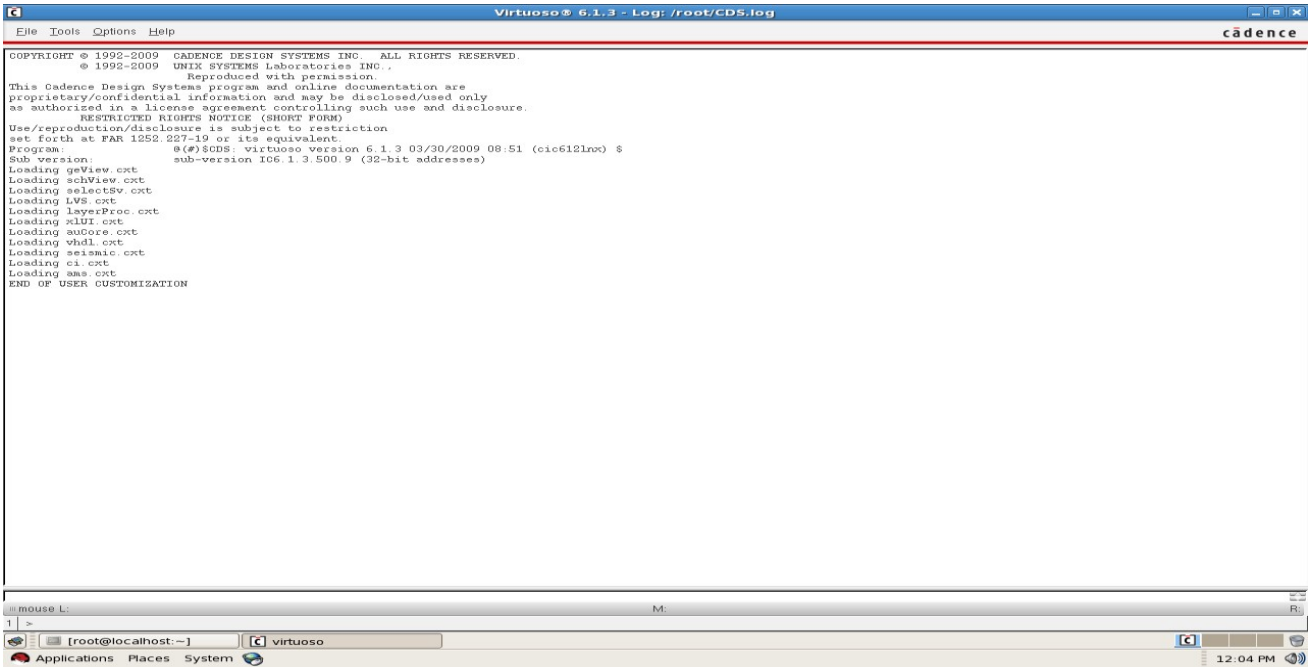
ANALOG DESIGN (SIMULATION & LAYOUT)

CUSTOM IC DESIGN FLOW

Note: Schematic entry can be done either through direct circuit entry, or through VerilogA, or through SPICE netlist. For simulation, a test circuit along with the symbol of the schematic is required. DRC, LVS and RCX form parts of the physical verification process.

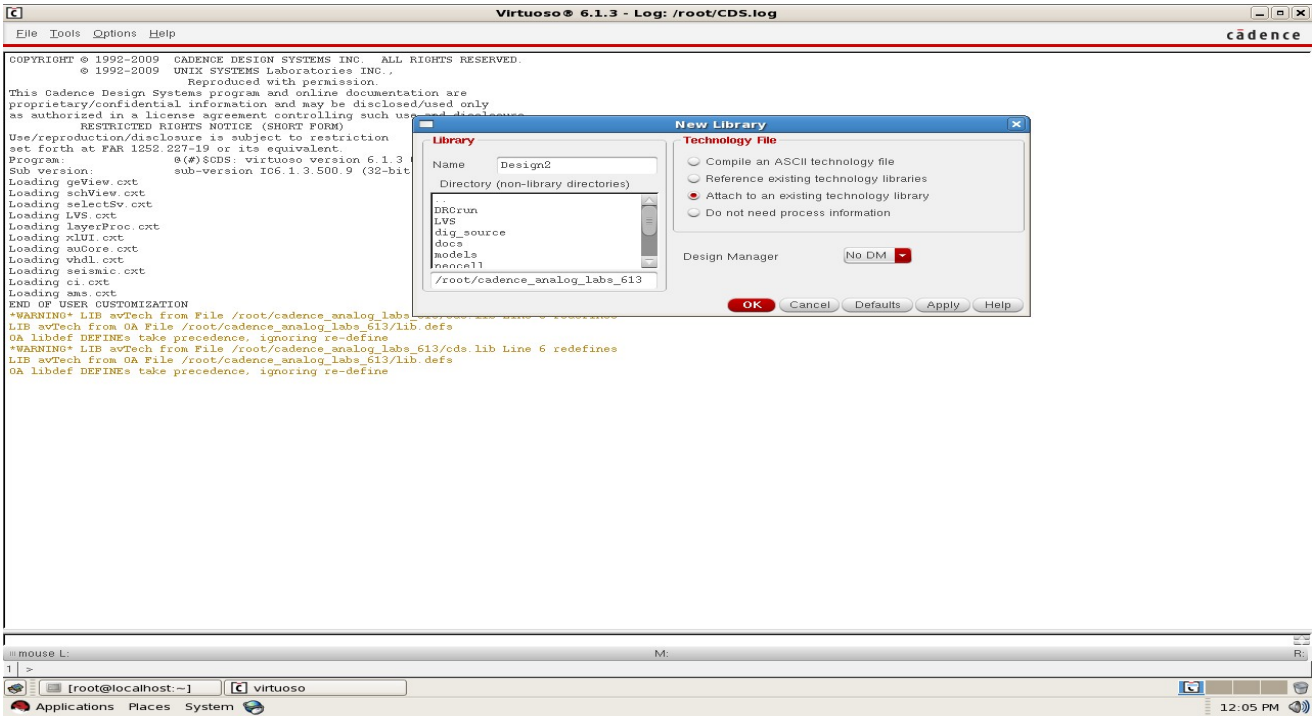
INITIAL PROCEDURES:

1. After logging in, *right click* and *open terminal*.
2. Get into the *c shell* by typing the command – **csh**
3. Run the shellscript by typing the command – **source /home/install/cshrc**
2. Invoke the analog design tool by using the command – **virtuoso &**
   After the virtuoso console opens up, maximize it. The linux terminal can be minimized.
3. In the virtuoso console, create your own library by following the steps –
   File → New → Library

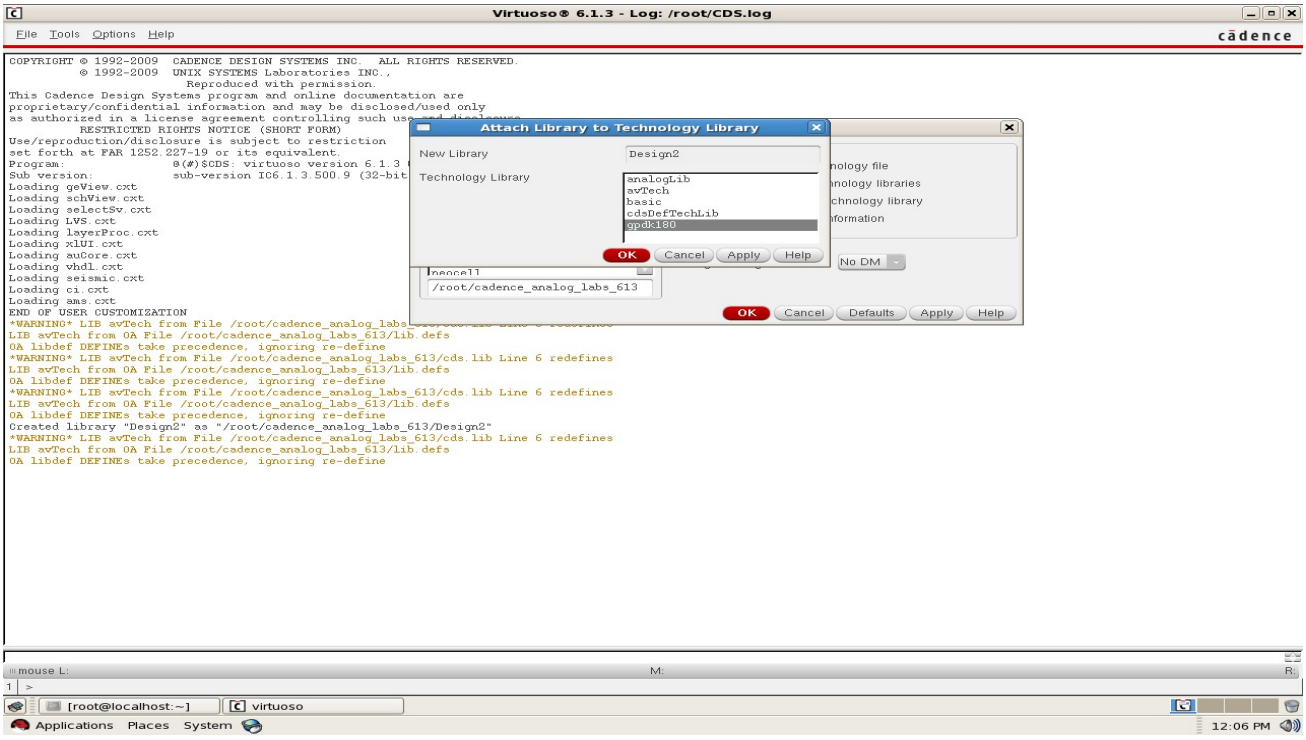


4. In the "New Library" window that opens up, fill in your library name (e.g.: Design2), and then click on the option –

   Attach to an existing technology library



5. A selection box named "Attach Library to Technology Library" will open. Select "**gpdk180**" and click on *OK*.



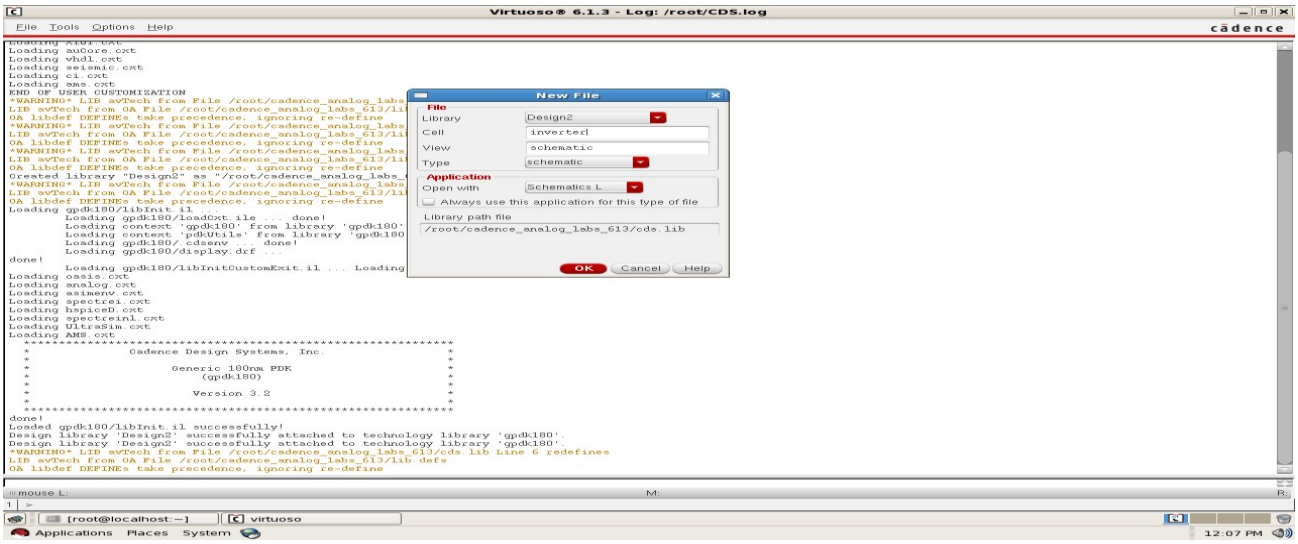
I. STEPS FOR DESIGN ENTRY:

1. In the virtuoso console, select the following –

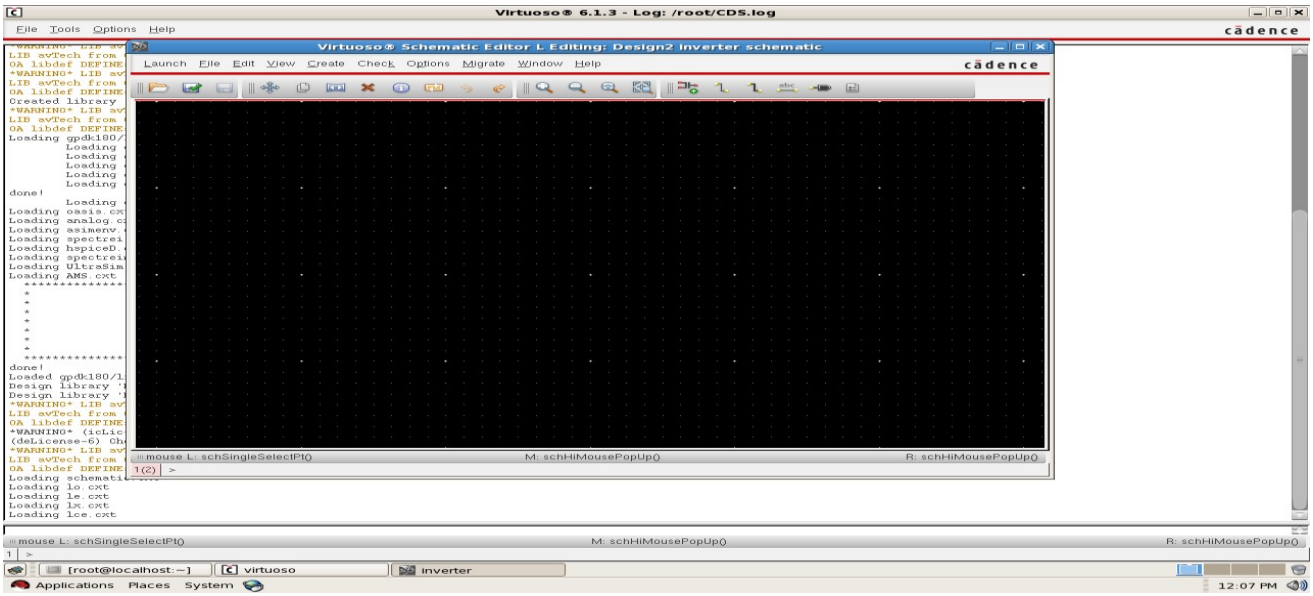
File → New → Cellview

2. In the "New File" form that opens, browse for your library name, and in front of the *Cell*, fill

in the name of the schematic that is going to be entered (e.g.: inverter). Later, click on *OK*.



3. A new design window opens, in which the schematic has to be entered.



4. After the schematic entry, a symbol for the schematic has to be created in the schematic editor window, by selecting the following –

Create → Cellview → From Cellview

During these steps, one more editor window will open up for the symbol entry. The symbol generation procedures will be elaborated while describing the experiments.

5. After symbol creation, both the editor windows can be closed. Now for the test circuit, a new *Cellview* has to be created in the virtuoso console. Again, the detailed procedures for the test circuit are elaborated while discussing the respective experiments.

II. STEPS FOR SIMULATION AND LAYOUT:

The test circuit has to be simulated by launching ADE-L in the schematic editor window, and then by choosing the respective analyses in ADE. The three main analyses that are performed are *transient*, *dc* and *ac*. With output plotted in Y-axis, the details of these analyses are summarized below –

Type of analysis	X-axis	Observation
Transient	Time	Waveform
DC	Input	Transfer characteristics
AC	Frequency	Bandwidth

After the circuit verification, the layout for the schematic has to be prepared using Layout-XL, and the same has to be physically verified. The detailed procedures are explained with the experiments.

NOTES

1. Abbreviations:

ADE Analog Design Environment

ELW Error Layer Window

LSW Layer Selection Window

RCX RC extraction

av assura verification

DRC Design Rule Check

gpdk general process design kit

LVS Layout Versus Schematic

VLW View Layer Window

***2.* Cadence tools used**:

Virtuoso ADE - for analyses and plots

Virtuoso layout suite

Virtuoso schematic editor

Spectre - for circuit simulation

Assura – DRC, LVS, RCX

Experiment No: 1

Draw the schematic of i) CMOS Inverter for the given specifications, and verify using Transient and DC Analyses. Draw the layout of CMOS Inverter and perform physical verification using DRC, ERC and LVS and Extract RC.

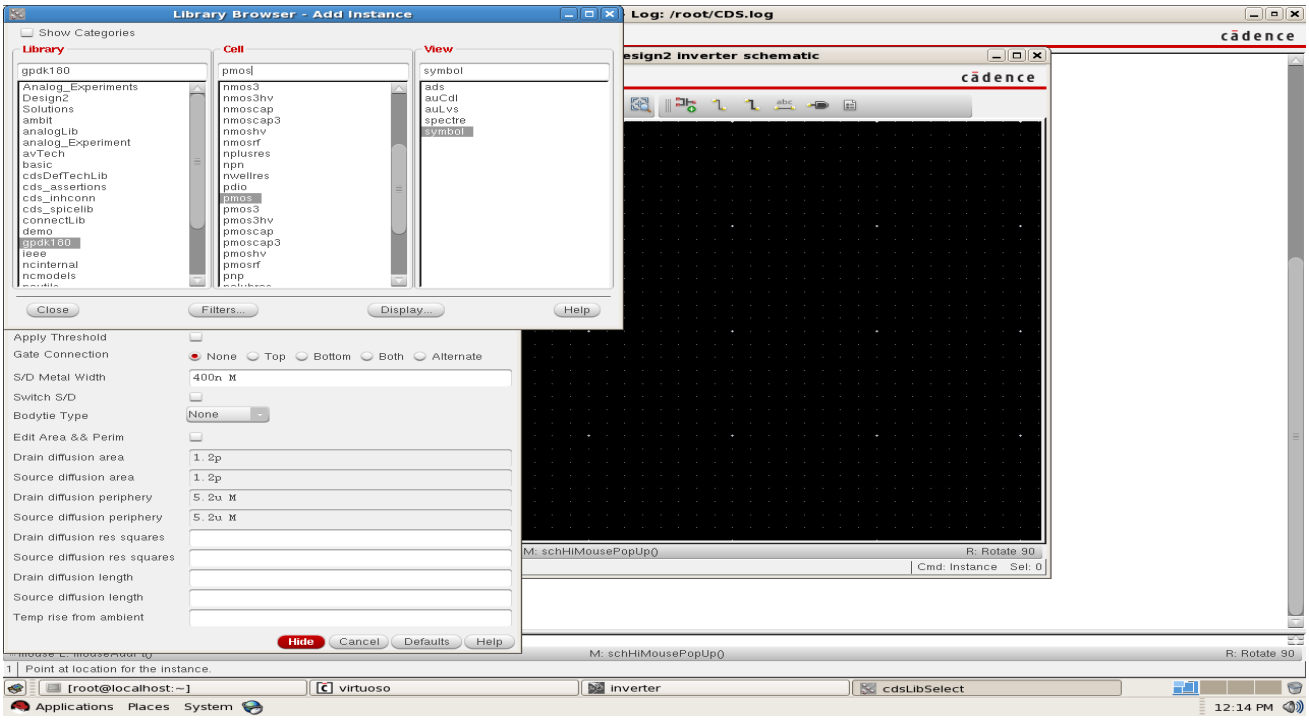
Aim: To simulate the schematic of the CMOS inverter, and then to perform the physical verification for the layout of the same.

Theory: Complementary MOSFET (CMOS) technology is widely used today to form circuits in numerous and varied applications. Today's computers CPUs and cell phones make use of CMOS due to several key advantages. CMOS offers low power dissipation, relatively high speed, high noise margins in both states, and will operate over a wide range of source and input voltages. CMOS circuit is composed of two MOSFETs. The top FET (MP) is a PMOS type device while the bottom FET (MN) is an NMOS type. Both gates are connected to the input line. The output line connects to the drains of both FETs. The CMOS circuit functions as an inverter by noting that when VIN is five volts, VOUT is zero, and vice versa. Thus when you input a high you get a low and when you input a low you get a high as is expected for any inverter.

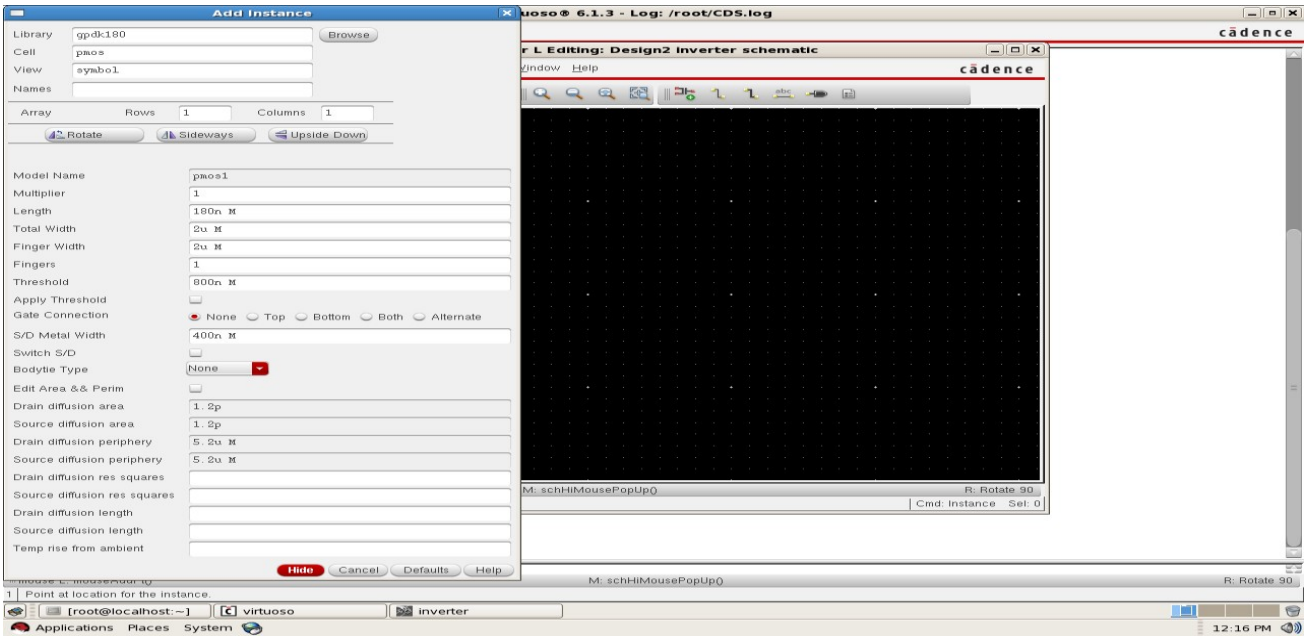
Procedure: The three initial steps before simulation are: schematic entry, symbol entry and test circuit entry. The procedures are as detailed below –

I. DESIGN ENTRY:

1. In the schematic editor window, for the addition of instances, press "i". This will open the "Add Instance" window. In that window, browse for the library *gpdk180*, select the cell *pmos* and then select the view *symbol*. Click on *close*.

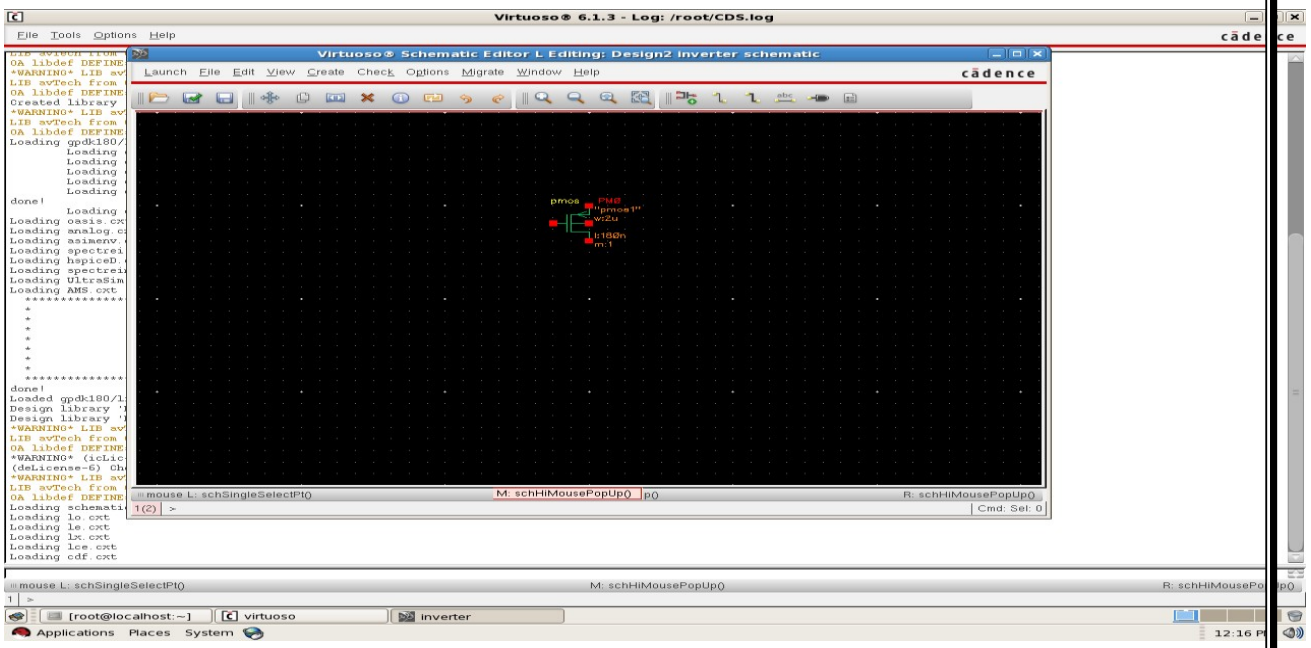


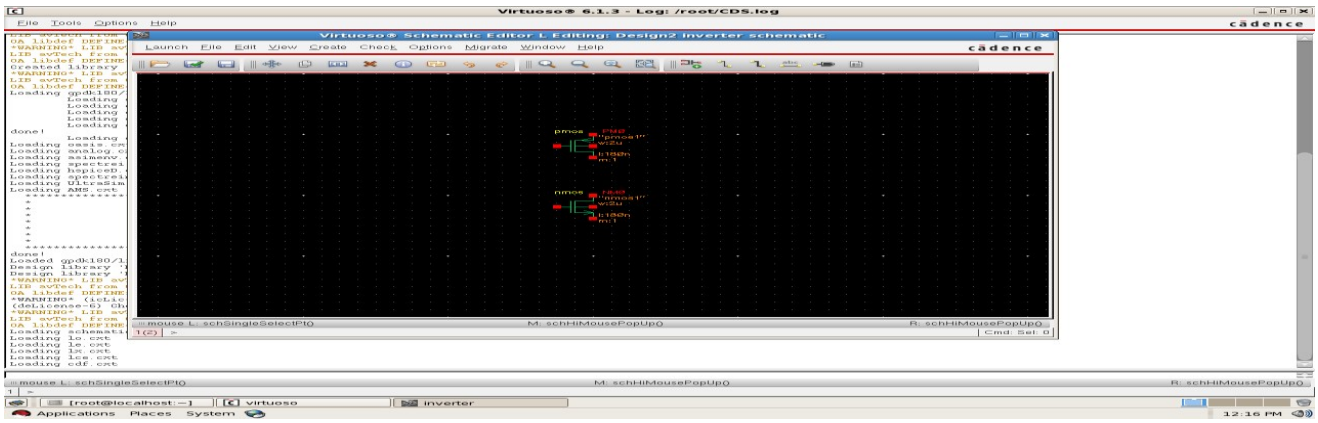
2. The properties of the selected instance are displayed in the “Add Instance” window. There is no need to modify any properties for this particular experiment. Click on *Hide*.



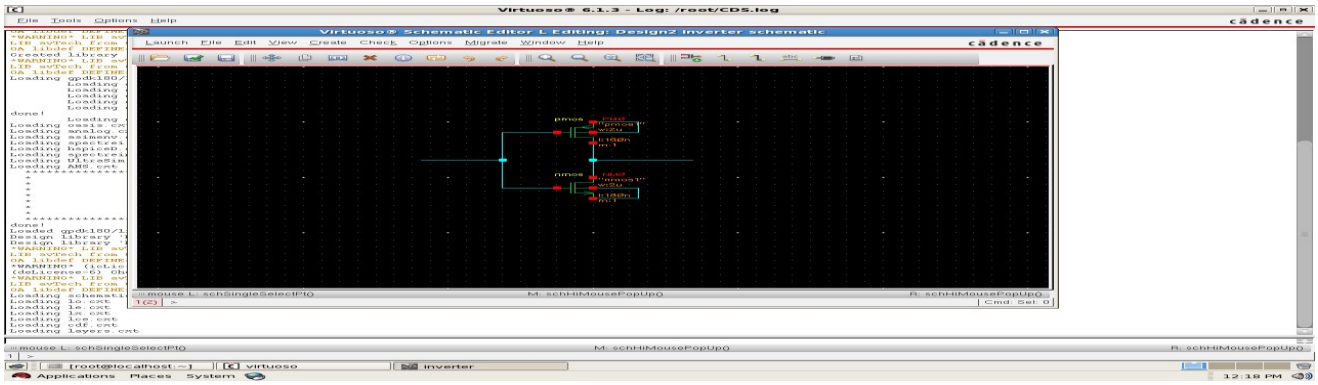
3. The *pmos* symbol will move along with the cursor. Place it in the top mid-position, left-click and then press *Esc*.

4. Similarly place *nmos* device, and then press *Esc*.

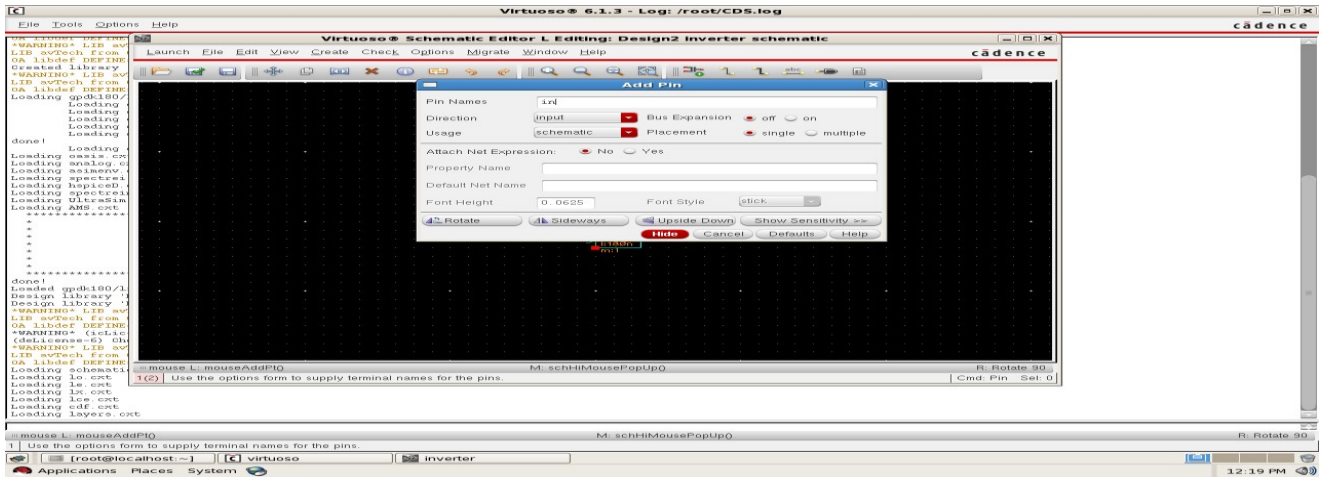




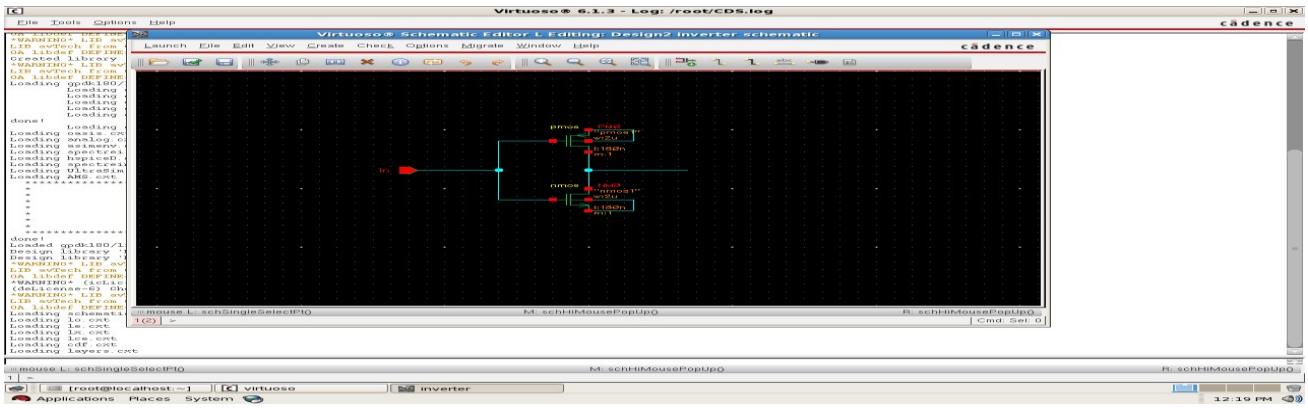
5. Now, press “w” for placing wire, click on the respective nodes and connect them through wire. Place the input and output wires as well. Complete the substrate connections also.



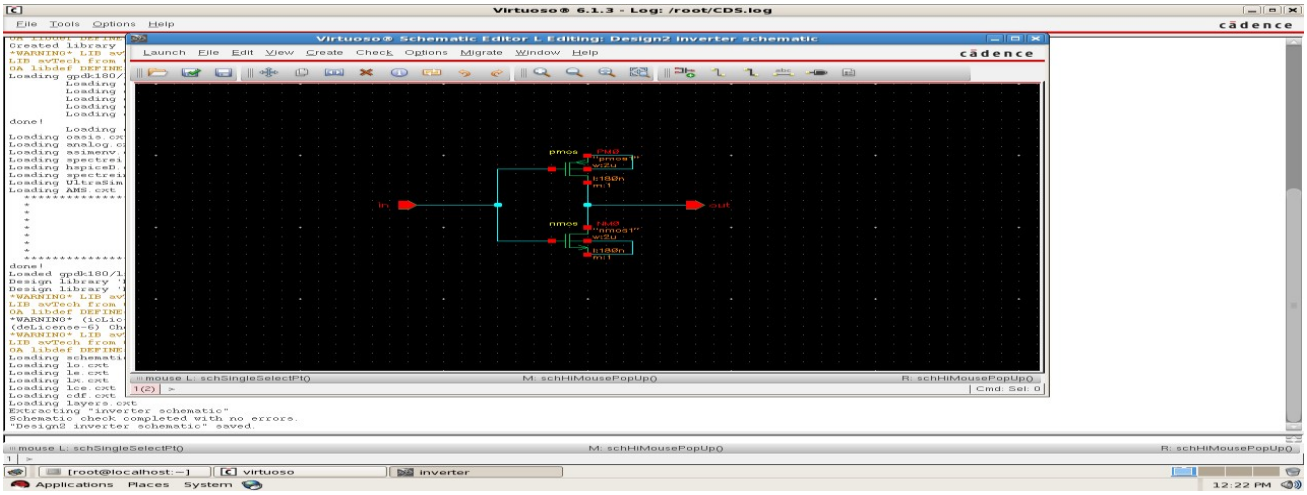
6. After pressing *Esc*, press “p” for adding pins to the schematic diagram. In the “Add Pin” window, enter the name of the pin (e.g.: in), and ensure its direction as input.



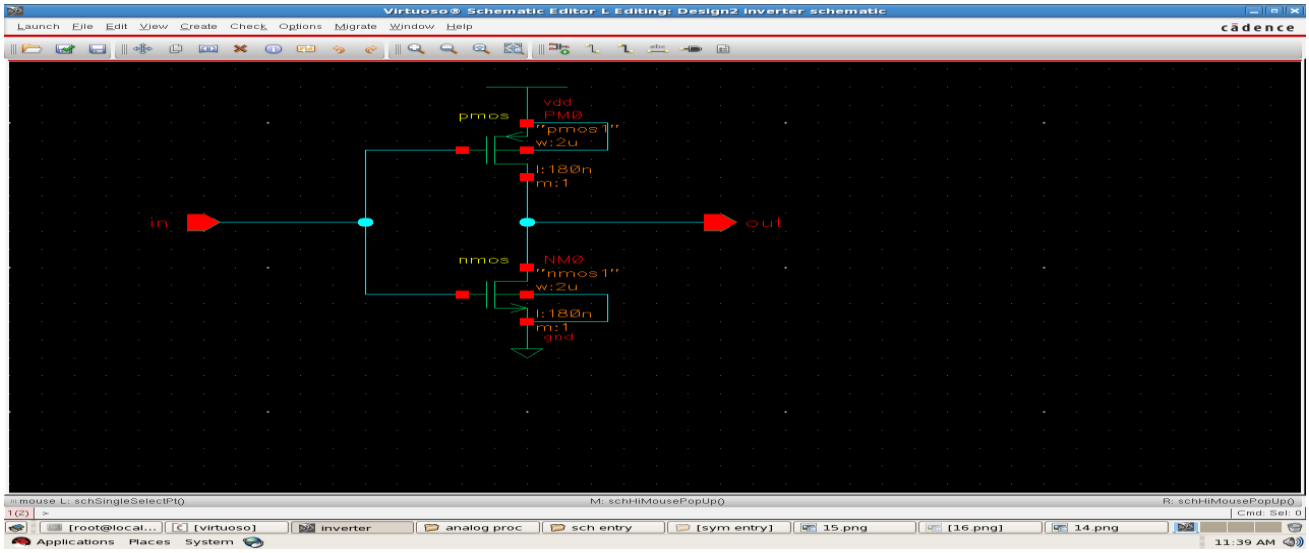
7. Click on *Hide*, and place the pin at the input. Later press *Esc*.



8. Similarly, place the output pin with name “out” and direction *output*.

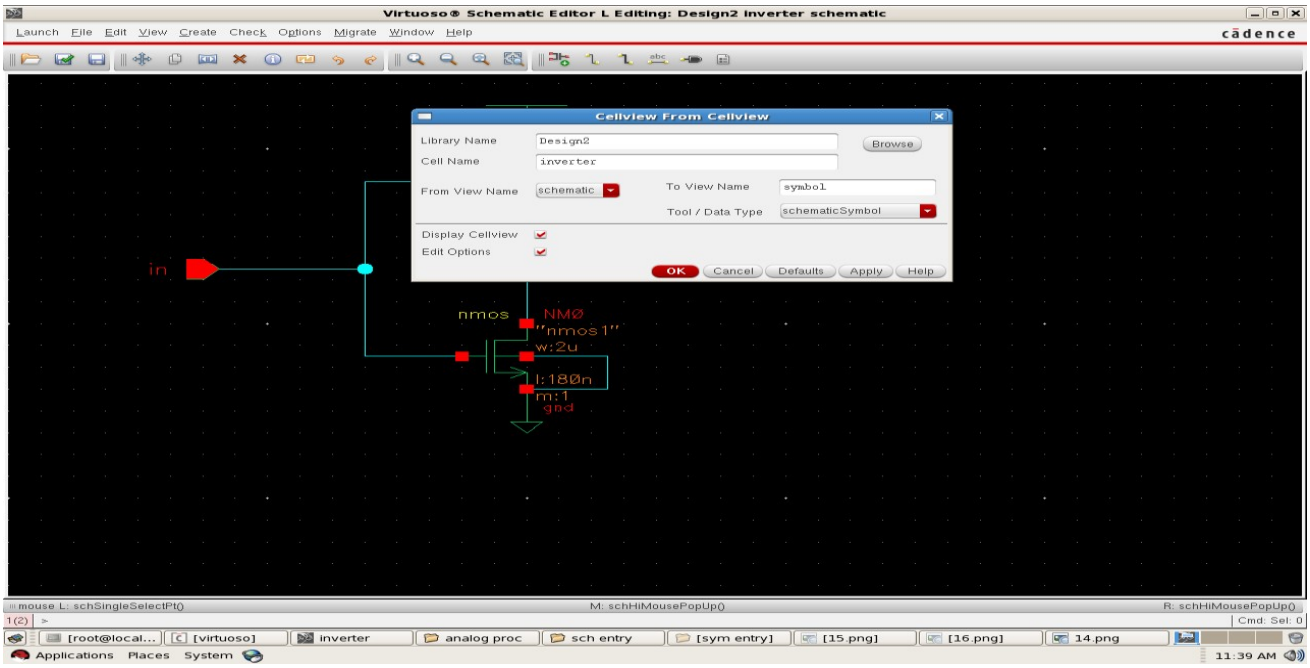


9. Complete the schematic by placing the instances “vdd” and “gnd”, which are in the *analogLib* library. Finally, click on “check & save” icon and observe the errors in the virtuoso console. In the schematic window, the errors will be highlighted with yellow boxes. Move those boxes, correct the errors, and click on “check & save”. Correct all the errors that are reported.



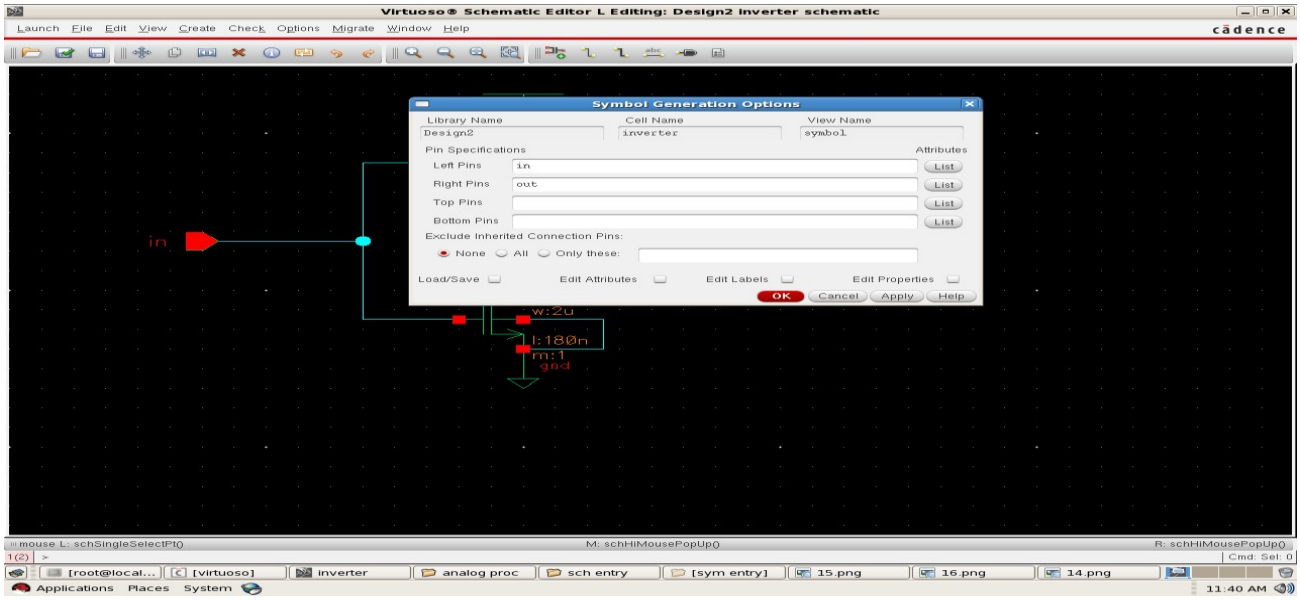
10. After the schematic entry is finished, a symbol for the design has to be created. For this purpose, click on *Create* and follow the procedure –

Create → Cellview → From Cellview

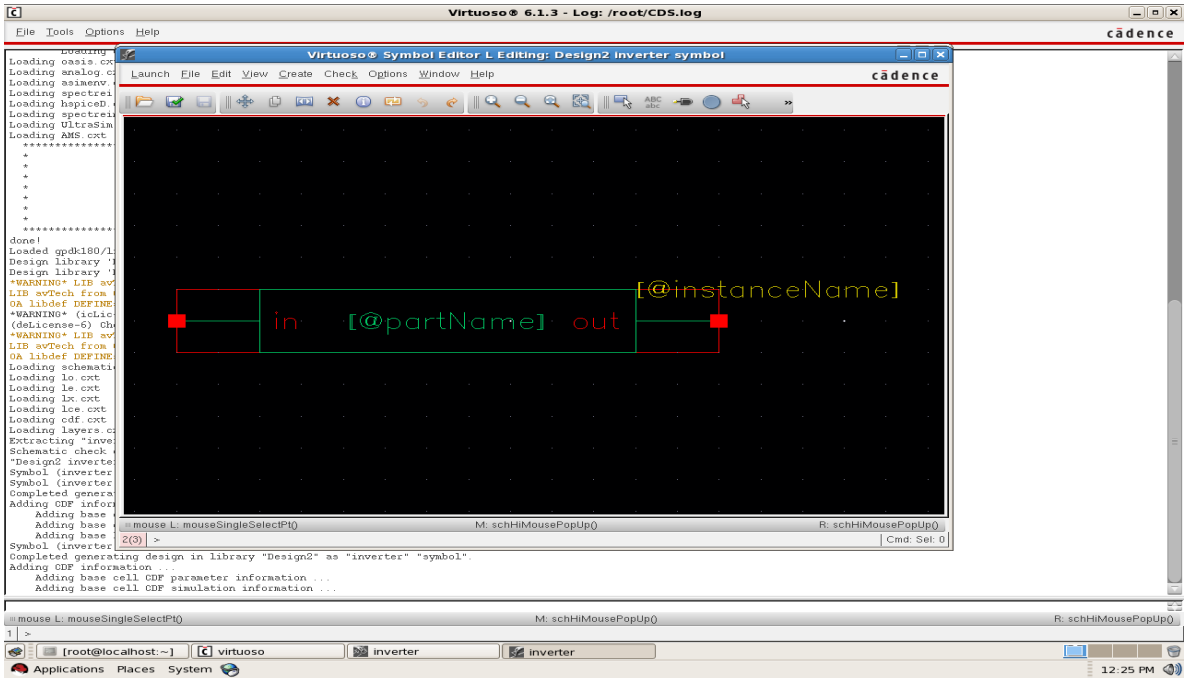


The name comes by default, along with the other options. Click on *OK*.

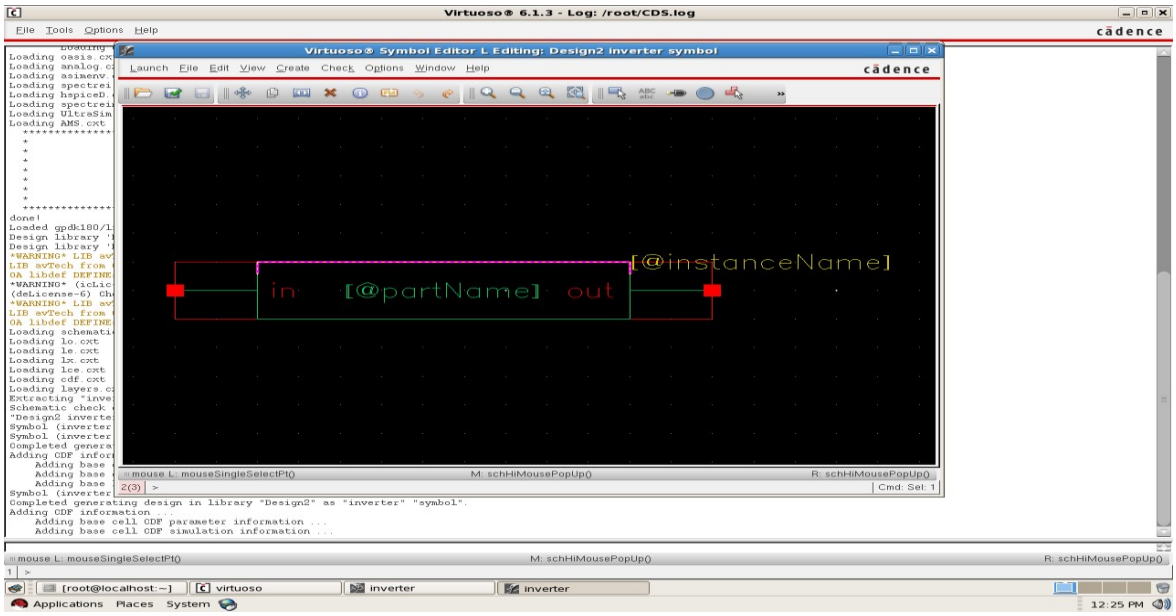
11. Another window opens, which shows the input and output pin configurations. Click on *OK*.



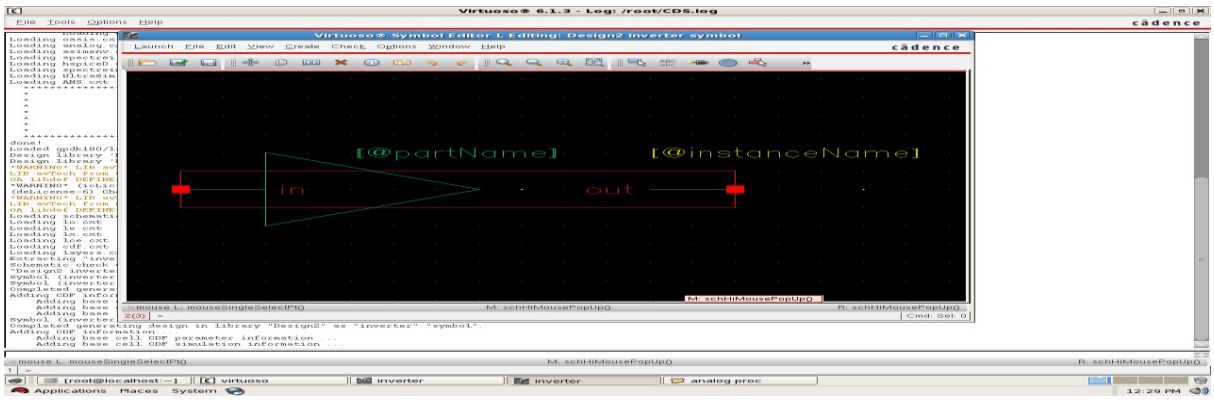
12. A symbol editor window opens up with the default symbol.



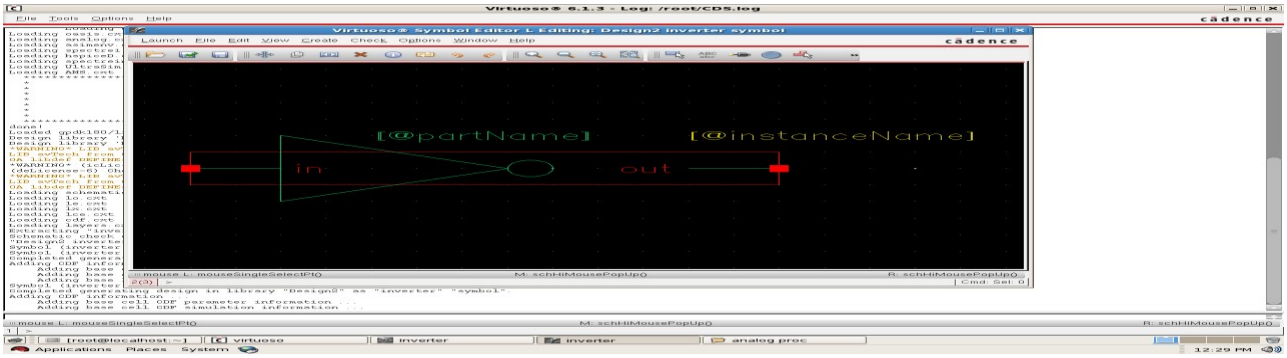
13. The default symbol can be deleted and the symbol as per our convention can be created. For this purpose, click on the green colored edge of the symbol. Now the color of the selected part changes into magenta. Press *Delete* and click again on the edge. The default green box gets deleted, and a new symbol can be created. Now press *Esc* to come out of the delete mode.



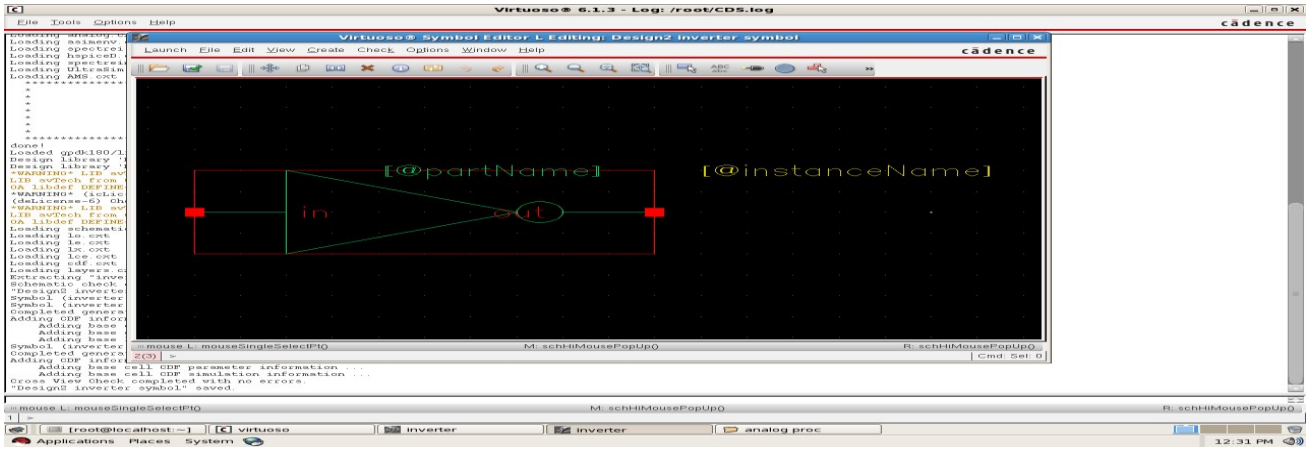
14. On the pulldown menu, Select *Create* → *Shape* → *Line* and click on the editor window. Use the mouse to create the shape of a triangle. Alternatively, you can select *Create* → *Shape* → *Polygon* as well. Use "right click" for making the line slant.



15. After the triangle is complete, select *Create* → *Shape* → *Circle,* and then bring the cursor in front of the triangle, and click once. Release the finger and move the mouse. After the circle is generated, click once again to place it.

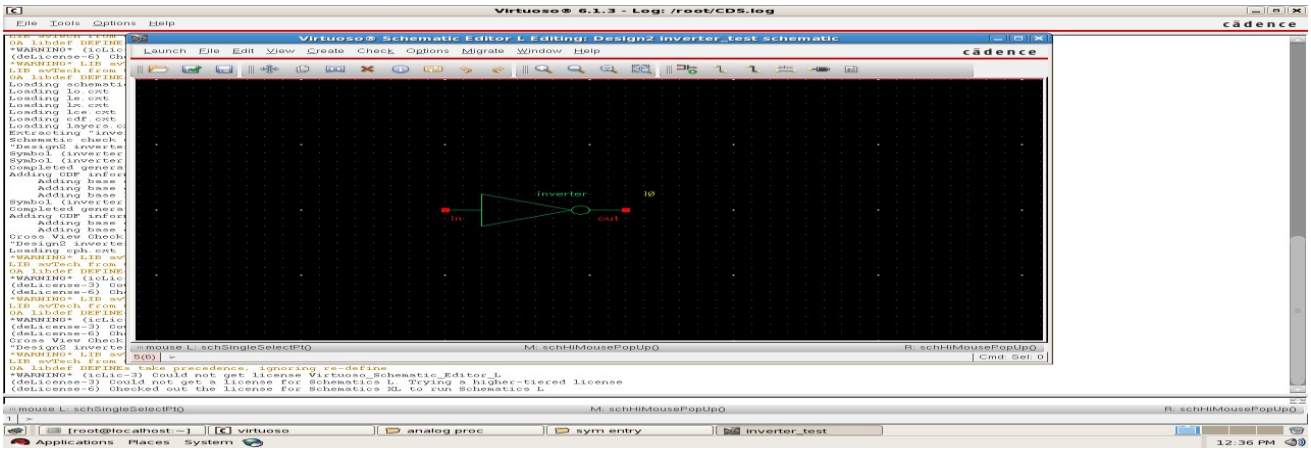


16. Now, click on the output wire and drag it in front of the circle. Similarly, join the output pin to the wire. Later, select *Create* → *Selection box* and click on *Automatic.* The tool will adjust the selection box around the created symbol automatically. You can similarly select and drag the port names to the respective places. Finally, “check & save” and observe the errors.

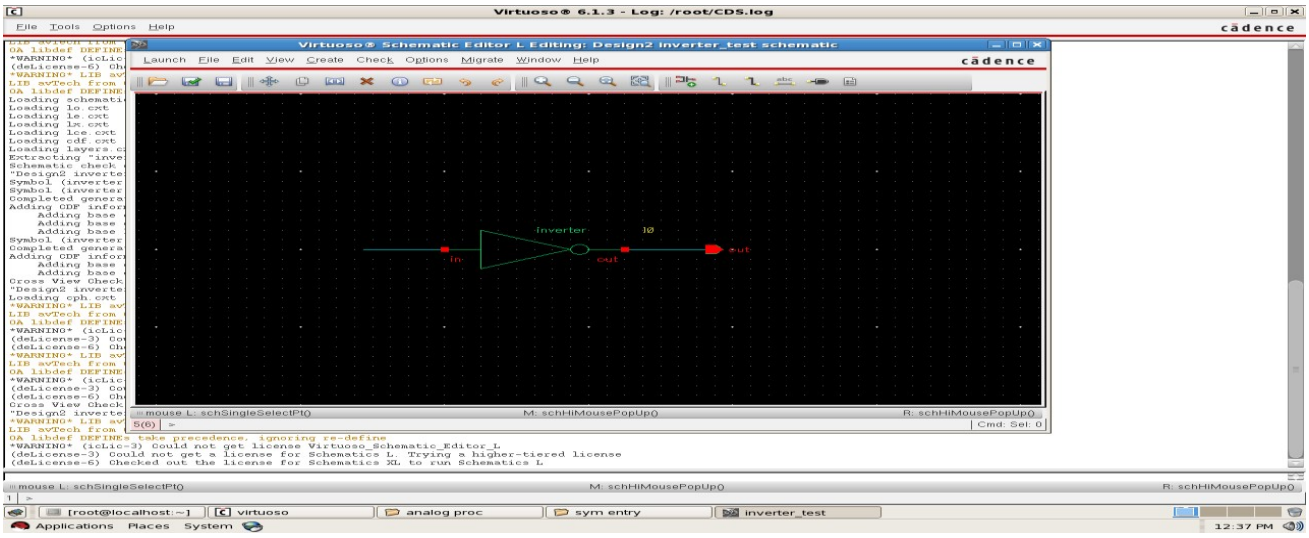


17. After the schematic and symbol entries, both the editor windows can be closed. Now for creating a test circuit, create a new cellview from the virtuoso console, and give the name as "inverter_test". When the editor window opens, press "i", browse for your library and select the inverter symbol which was created earlier. Place it in the middle of the screen.

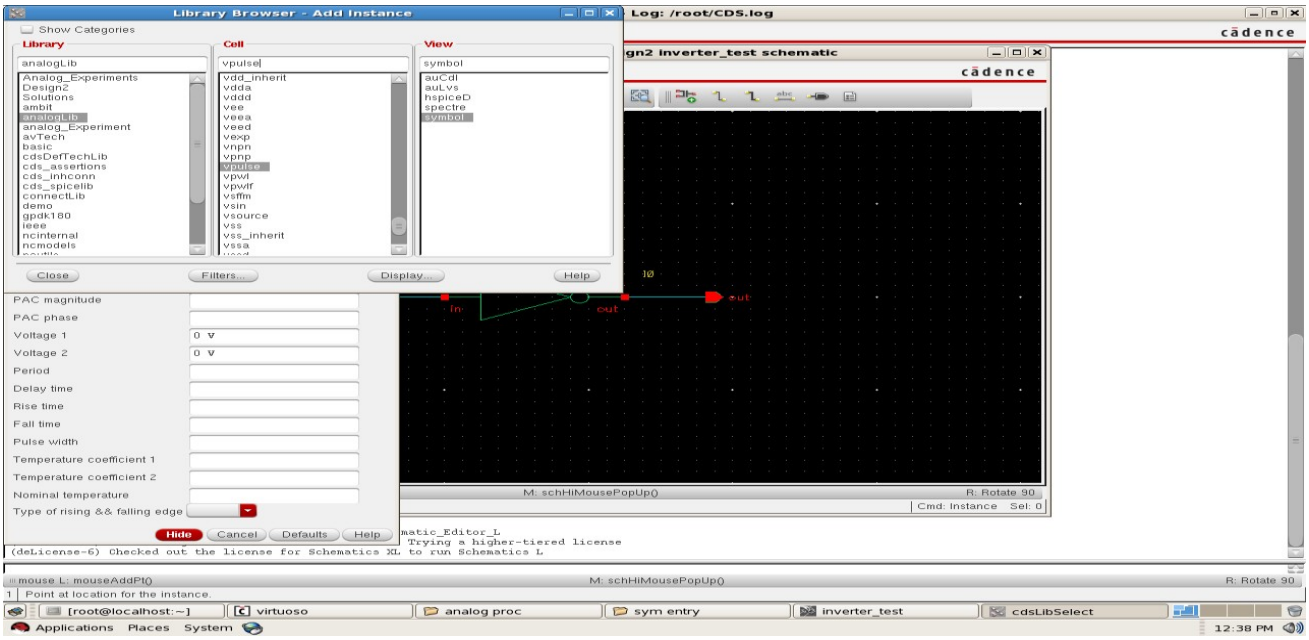
    To ensure that the symbol is loaded correctly, you can click on the symbol and then press "Shift E". The schematic editor will move one level down, and the inverter's circuit entered earlier will be displayed on the screen. To come back to the symbol, press "Ctrl E"; the symbol will be displayed back. Press *Esc* to unselect the symbol.



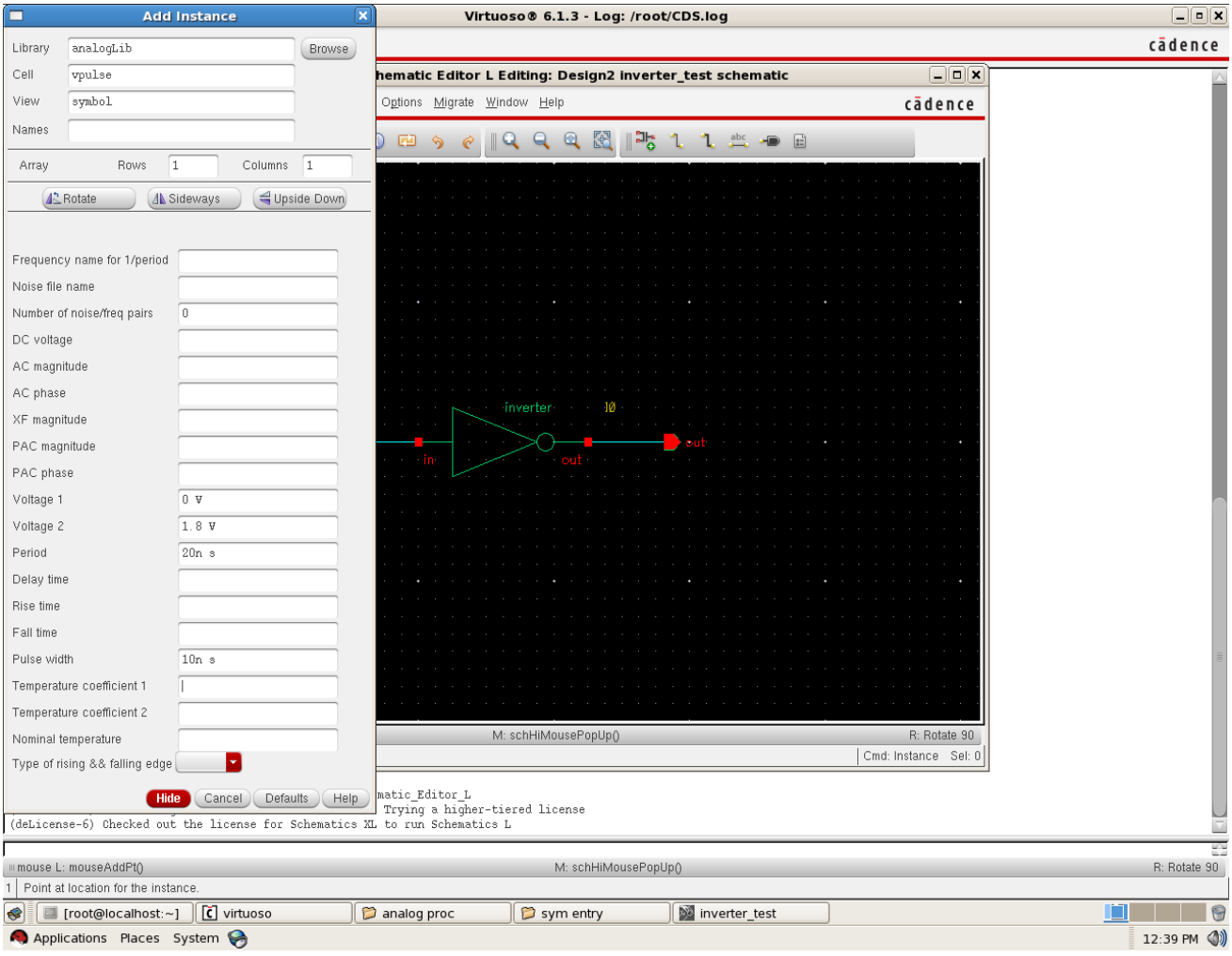
18. Place wires at the input and output, and place an output pin as well. These wires are needed during simulation, to plot the voltage waveforms



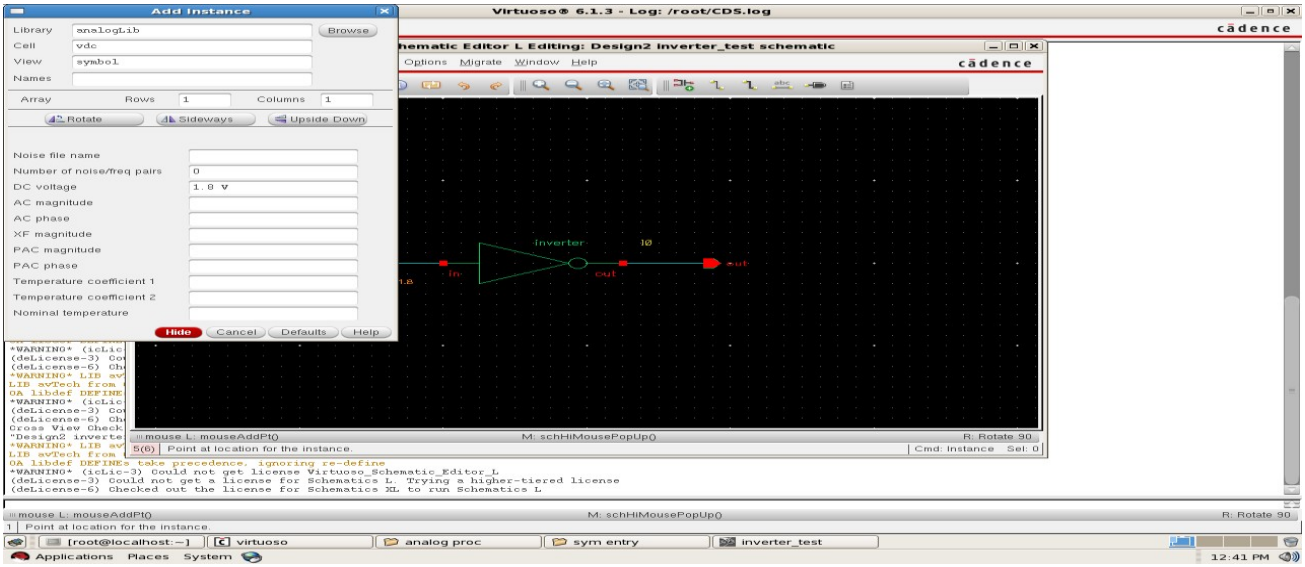
19. Press "i", browse *analogLib* library and select "vpulse" and its symbol.



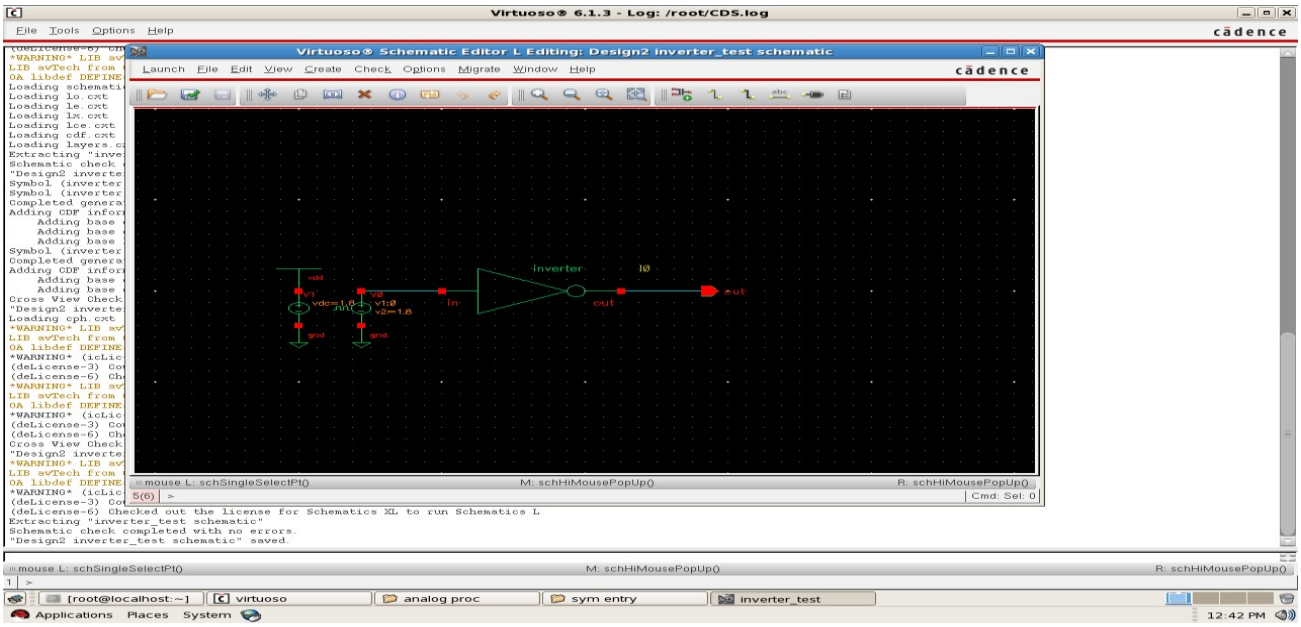
20. In the property window, enter *Voltage1* as **0** and *Voltage2* as **1.8**. Similarly enter *Period* as **20n** and *Pulse width* as **10n**, without the space in between. No need to enter the units; they appear automatically. Place the "vpulse" at the input wire. Connect "gnd" at the other end.



21. Similarly, browse for the instance "vdc", and enter its *DC voltage* as **1.8**.



22. Place “vdc” at the front, and connect “vdd” and “gnd” accordingly. Finally “check & save”. The test circuit is complete now, and ready to be simulated.



Note: The library *gpdk180* contains the technology dependent components (180nm), and the library *analogLib* contains the technology independent components.

Whenever a component needs to be selected, place the cursor on the component and click on it. The selected component’s boundary turns into magenta color. Now the properties of the component can be verified by pressing “q”, after which the property window opens.

To zoom a particular portion of the screen, right click, hold, and move the mouse. A yellow colored boundary will be drawn on the screen. When the finger is released, the highlighted portion gets zoomed. To come back to the original screen, press “f”. Alternatively, “Ctrl Z” and “Shift Z” can be used, to zoom in and zoom out.

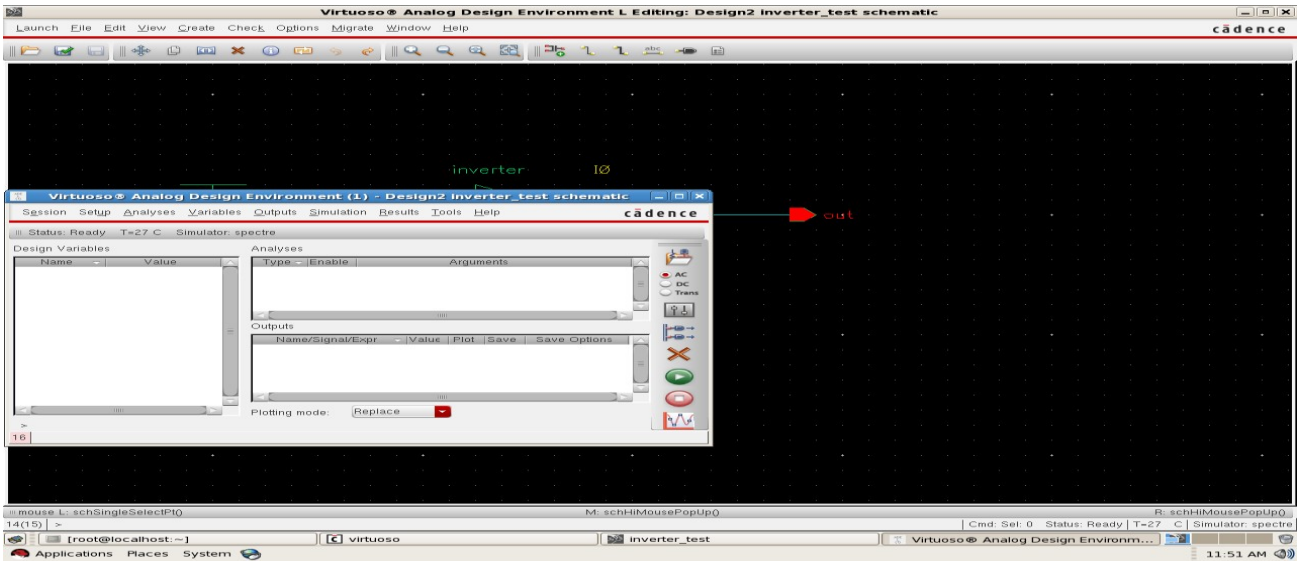
After the symbol is entered, *Shift* and *E* can be used together to move one level down, to view the schematic diagram. Later, *Ctrl* and *E* can be used together to move one level up, to the symbol.

The hot key functions that are used during design entry are summarized as follows –

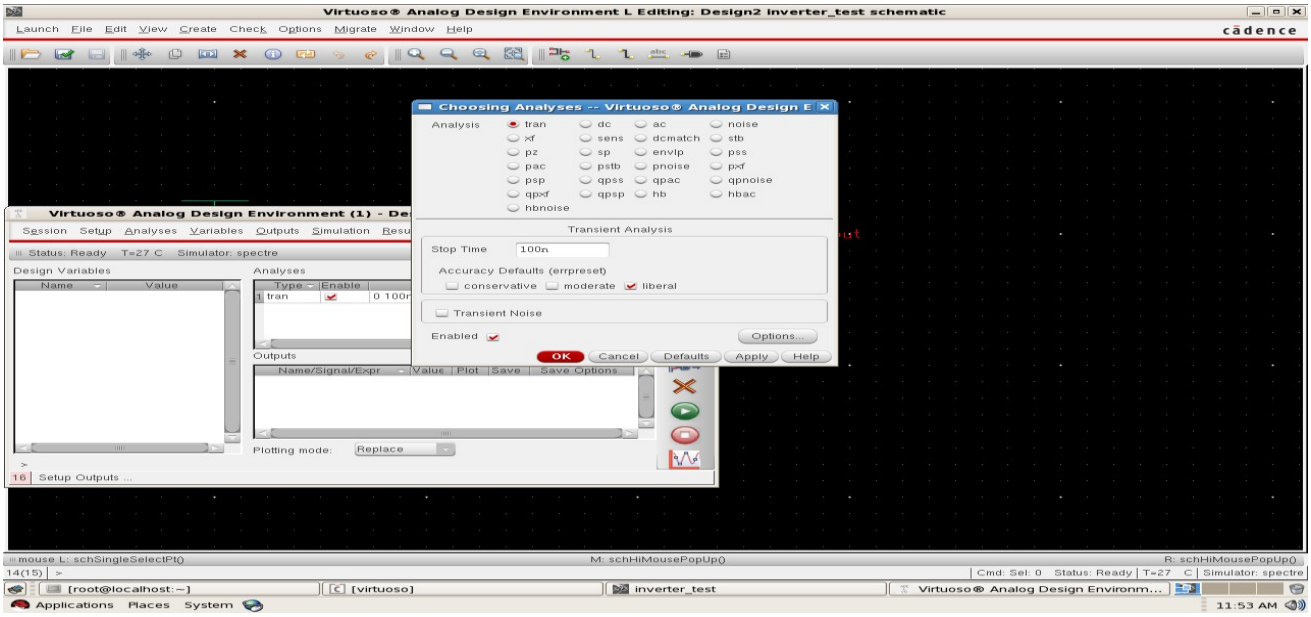
Hot key	Function
i	Instance
w	Wire
p	Pin
m	Stretch
q	Property
r	Rotate
u	Undo
c	Copy
f	Fit to screen
Esc	Exit

II. SIMULATION:

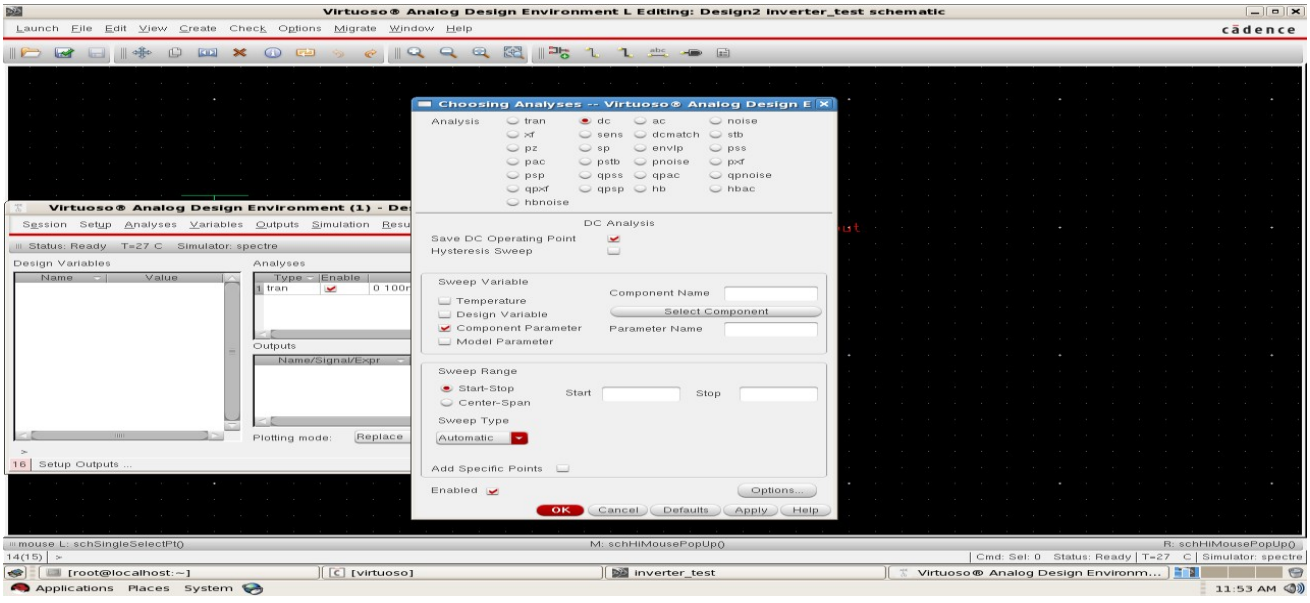
1. In the test circuit's editor window, click on **Launch → ADE L**. A new window will open.



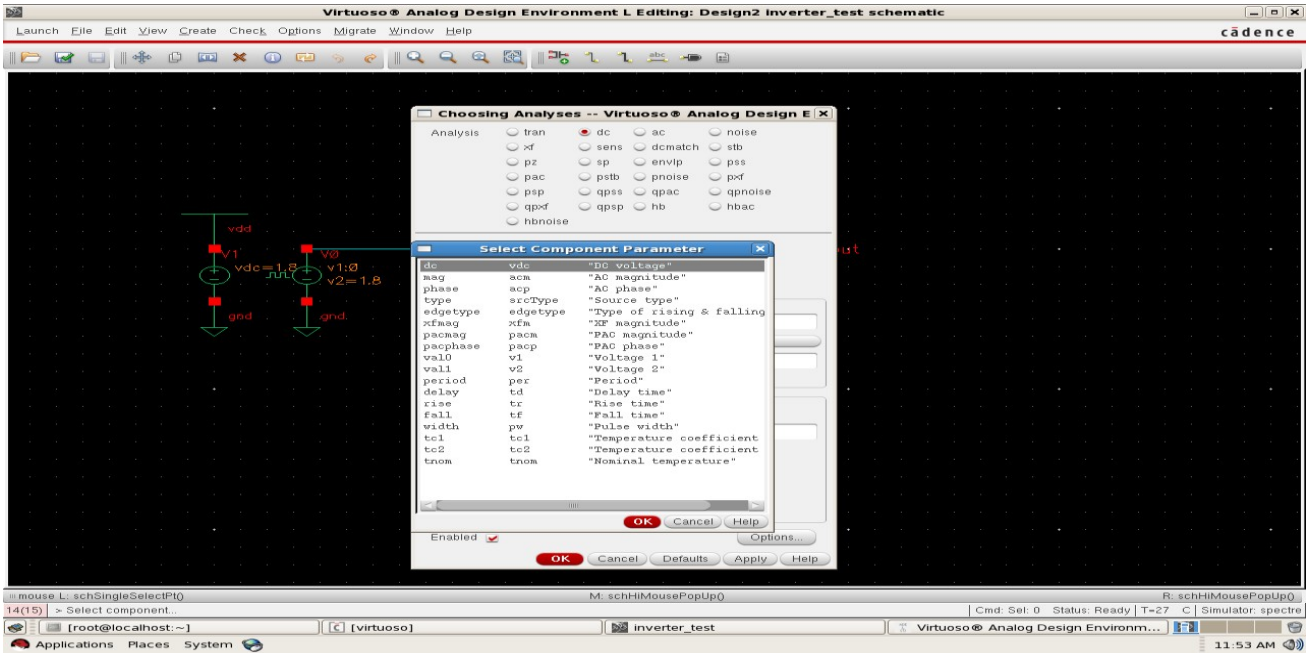
2. Click on **Analyses → Choose**. A new window will open, in which select “tran”. Fill the stop time as **100n**, and select *liberal*. Later, click on *Apply*.



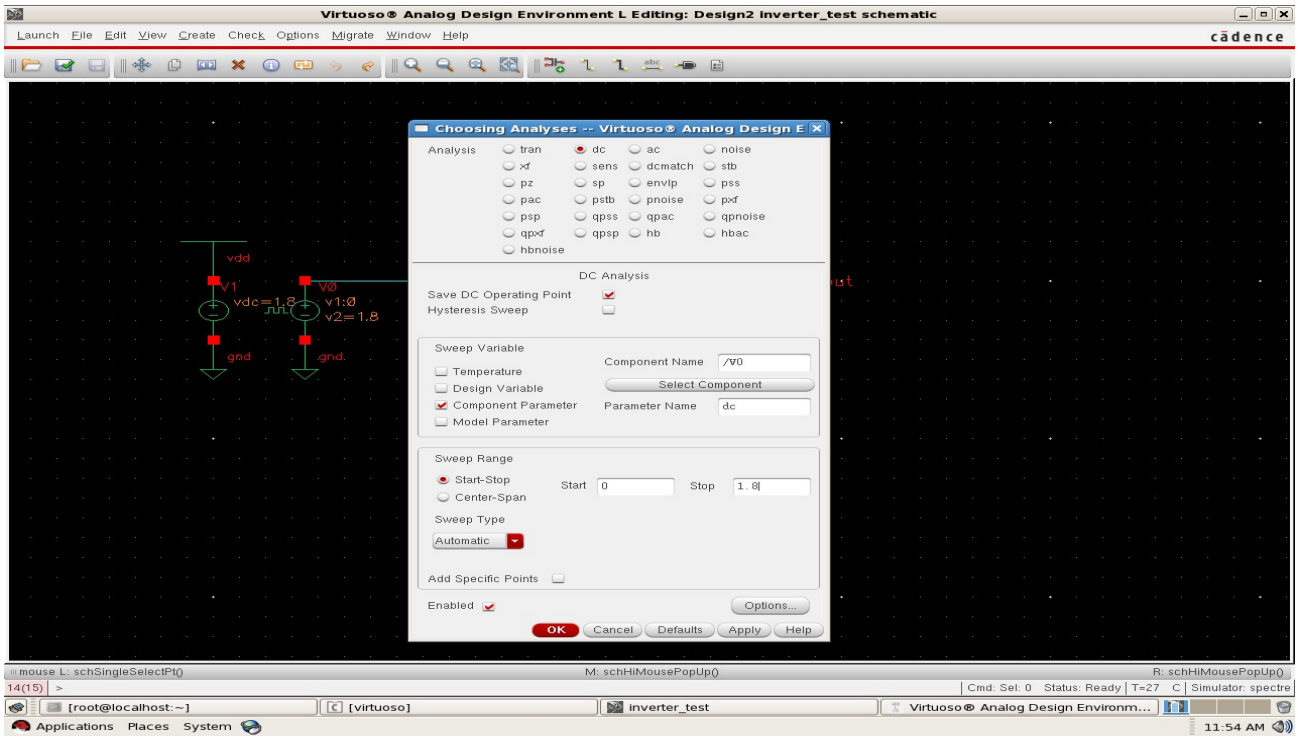
3. Now select “dc” on the *Choosing Analyses* window, and click on *Save DC Operating Point*. Click on the *Component Parameter*. A *Select Component* option will pop up.



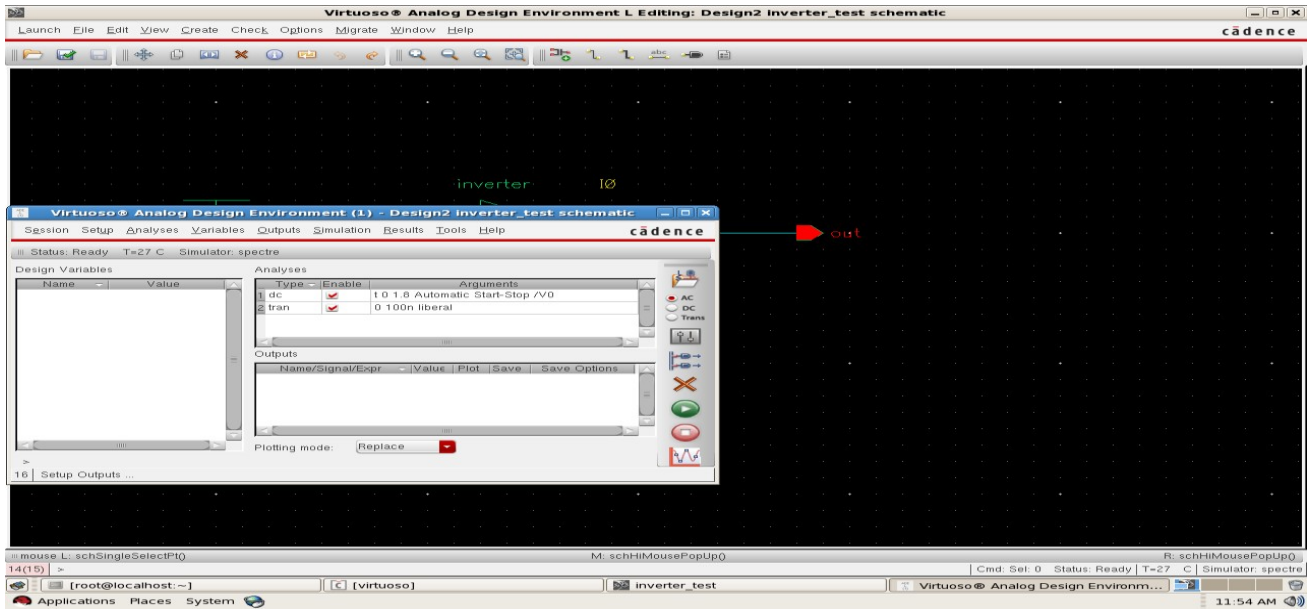
4. Double click on *Select Component*. The ADE window gets minimized, and the schematic is shown. Click on the component “vpulse”. In the new window that opens up, click on the top most parameter *dc* and then click on *OK*.



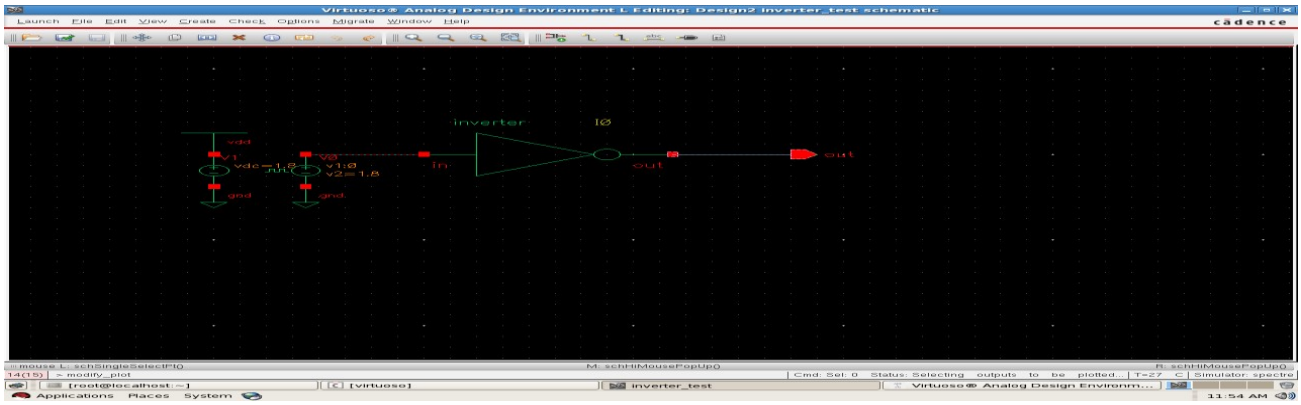
5. Ensure that the component name and parameter name are updated. Now in the *Sweep range*, enter **0** and **1.8** in the *Start* and *Stop* options respectively. Later, click on *Apply*.



6. In the ADE window, ensure that the Analyses fields are updated for **tran** and **dc**.

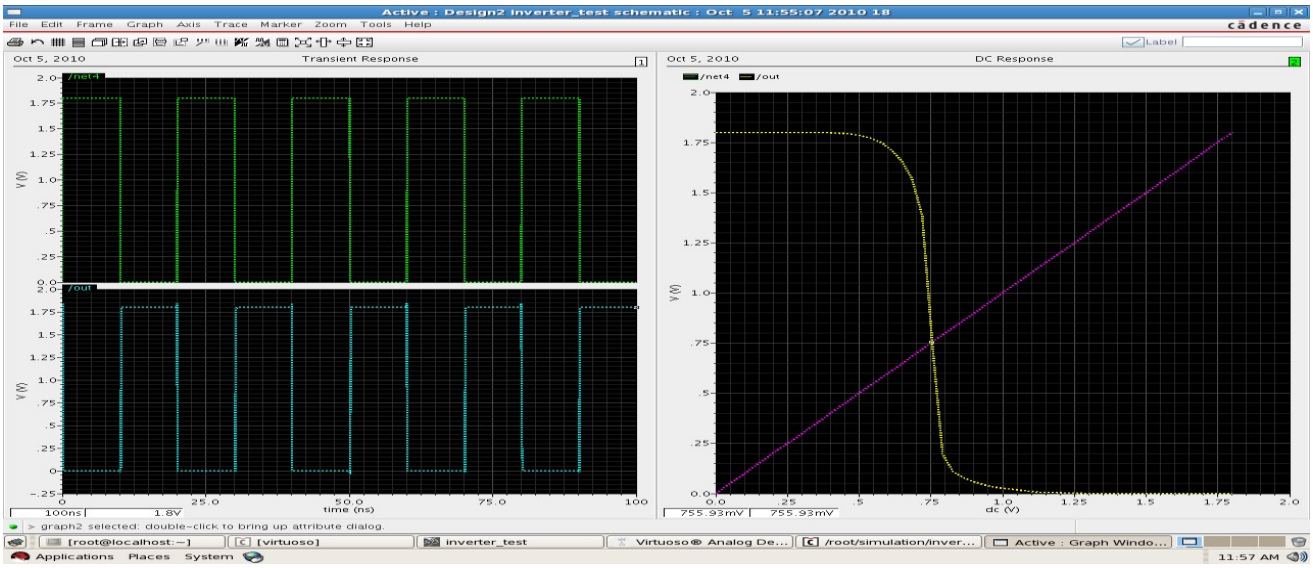


7. Now to select the stimulus and response points, in the ADE window, click on **Outputs → To be plotted → Select on Schematic**. In the schematic window, click on input and output wires. These wires will become dotted lines when selected. Later, press *Esc*.



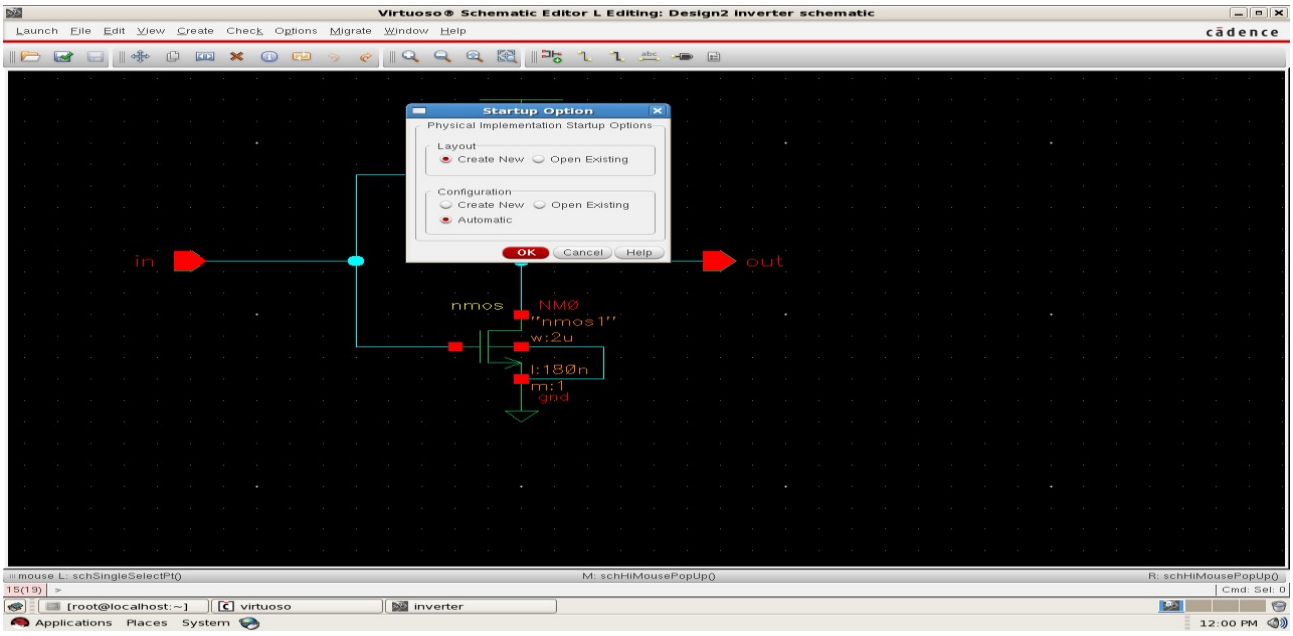
8. In the ADE window, check that the Outputs fields are updated. Now click on **Simulation → Netlist and Run**. The waveform window will open and the simulation results are displayed. Transient response is displayed on the left side and DC response on the right.

   Click on the transient response waveform and then click on the fourth icon at the top (Strip chart mode). The input and output waveforms are displayed separately. You can “right click” on each waveform, and then edit the properties of the display such as color and appearance. Similarly, the transfer characteristics can be observed at the right hand side.

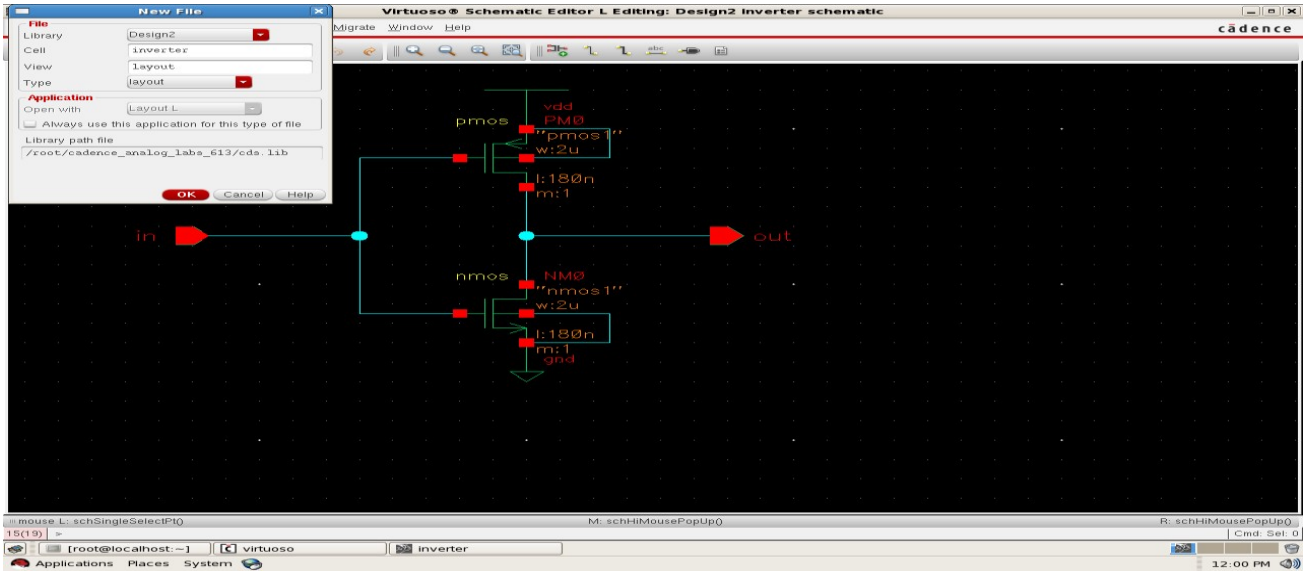


III. LAYOUT:

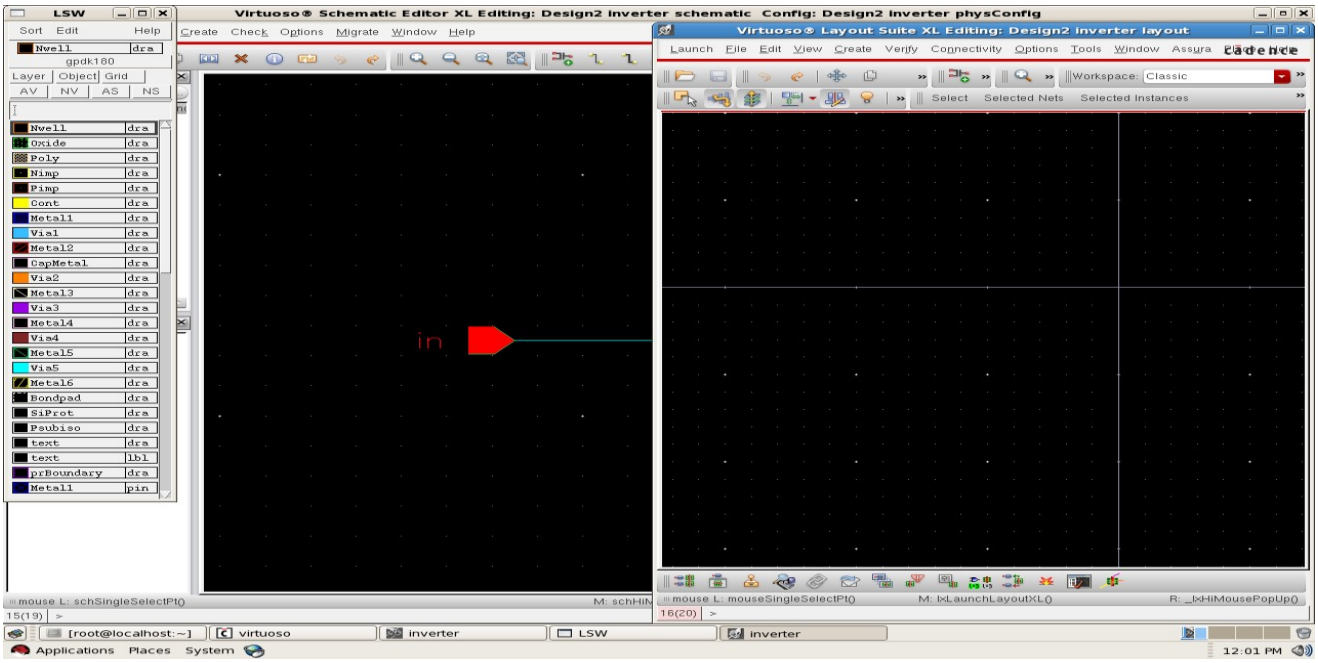
1. For preparing the layout of the inverter, all the other windows can be closed, except for the virtuoso console. In the console, open the schematic of the inverter and click on **Launch → Layout XL**. In the *Startup Option*, click on *OK*.



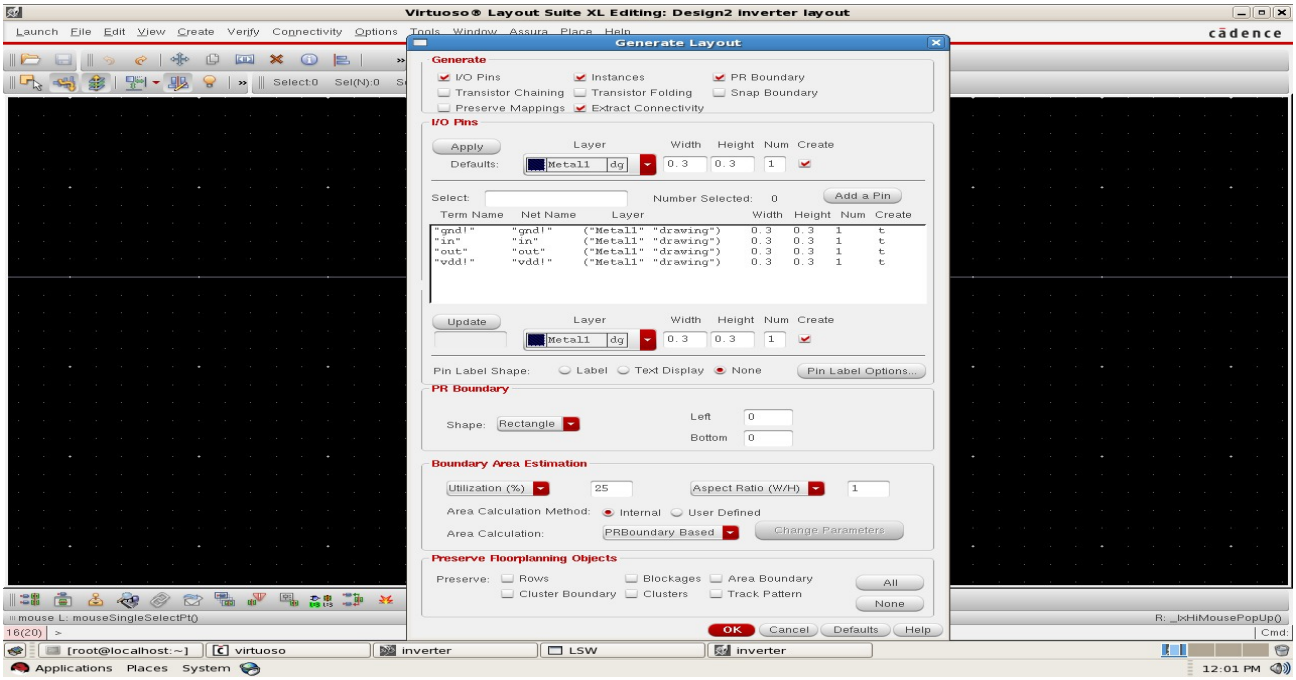
2. In the *New File* option, the tool selects the view as *layout* by default. Click on *OK*.



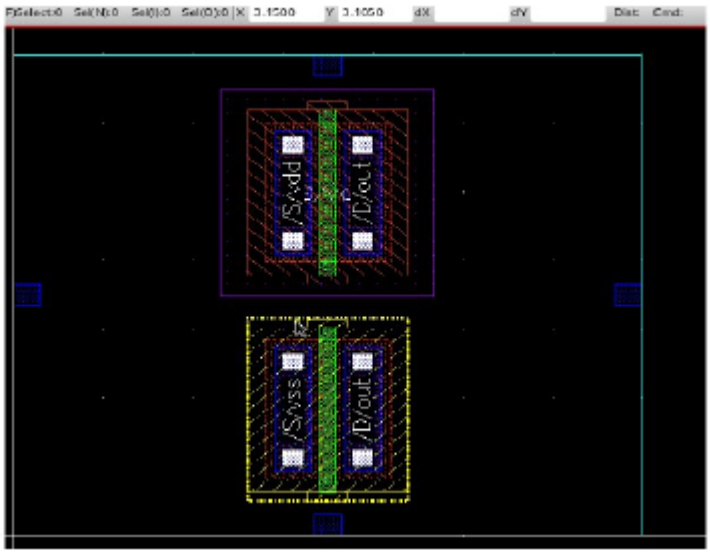
3. The tool opens the LSW and the Layout suite.



4. Maximize the layout suite and click on **Connectivity → Generate → All from Source**. A *Generate Layout* window will open, with default attributes. Click on *OK*.



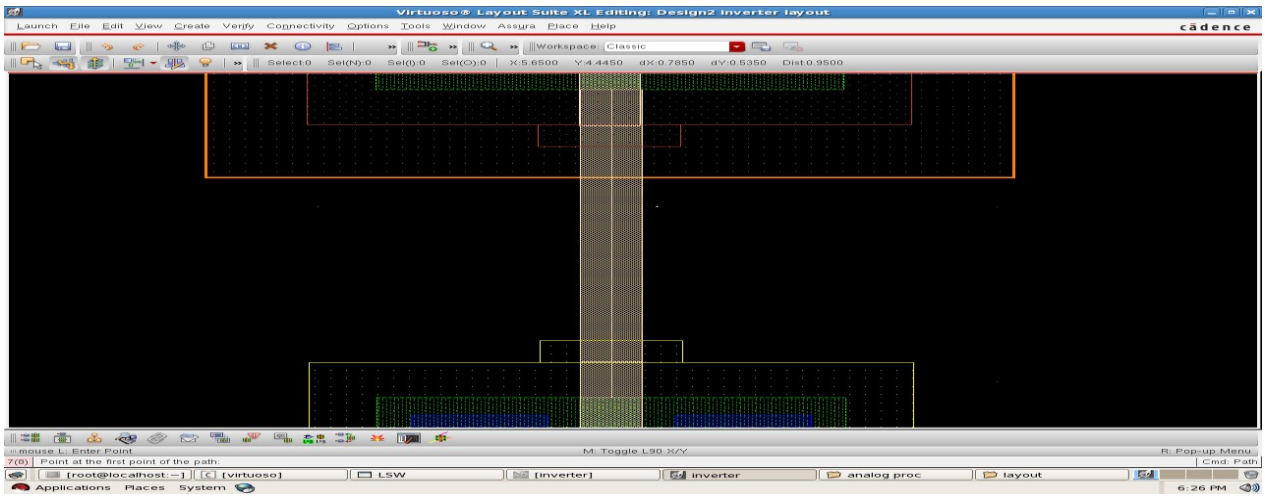
5. Press “Shift F” to see all the layers within the default layout. Hold the “right click” and move the mouse to zoom a selected portion, and observe the layout carefully. The color details are – **Orange border**: *n-well*, **Red border**: *p-diffusion’s boundary*, **Yellow border**: *n-diffusion’s boundary*, **Green**: *diffusion*, **Rose**: *polysilicon*, **Yellow square**: *contact cut*, **Blue**: *metal1*.



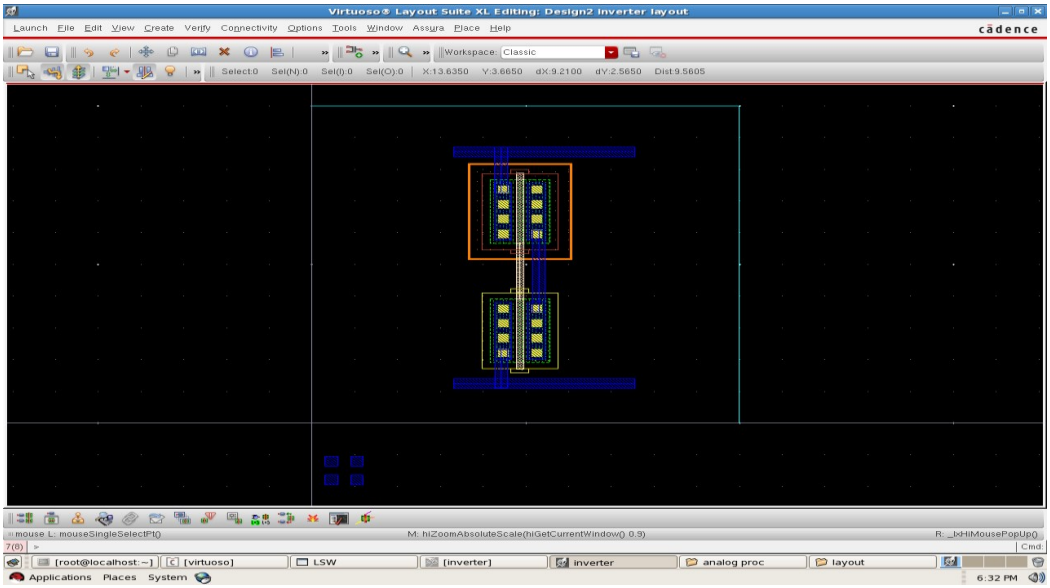
6. The layout can be moved either vertically or horizontally, not diagonally.(“s” is for stretch, in the layout suite). The selected edge will turn into magenta color. Now release the finger and move the mouse till the desired area, and click again. Later, press *Esc*.

7. After placing the devices, zoom the space in between the transistors. In the LSW, select *Poly*. Now in the layout suite, press “p”, place the mouse at the middle of the gate’s lower contact of pmos device, and click once. (“p” is for path, in the layout suite). Release the finger and move the mouse downwards. The poly path will move along with the mouse. Move the mouse until the gate area of the nmos device gets overlapped. Bring the cursor exactly to the middle

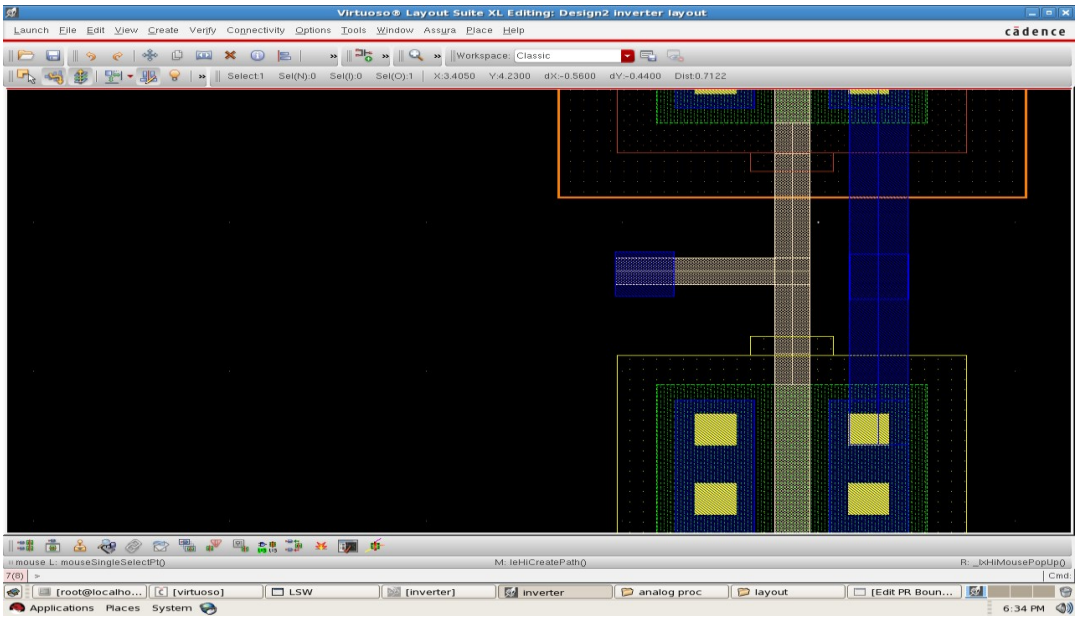
of the path and double click. The poly path between the gates gets realized. The area can be zoomed further, and the devices can be moved, for the exact overlapping of the poly layers.



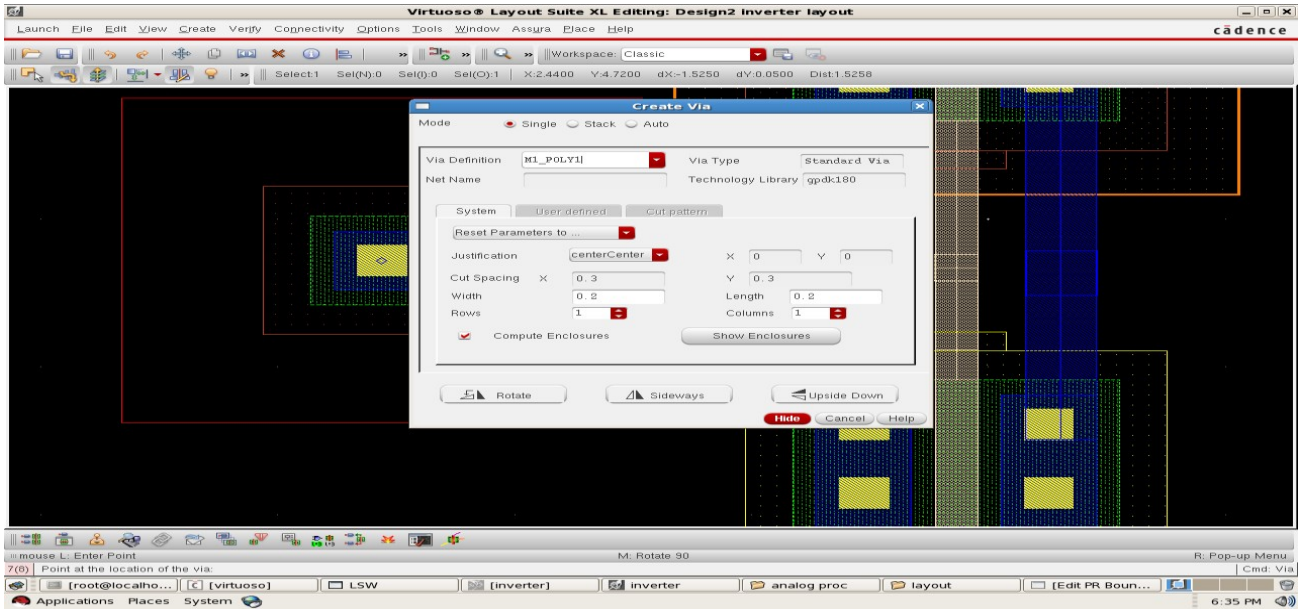
8. In the LSW, select *Metal1*. Using the same procedure, draw the paths for “vdd” at the top and “gnd” at the bottom. Later, using the same metal path, connect the source of pmos device to “vdd” and that of nmos device to “gnd”. Finally, connect both the drains for the output path.



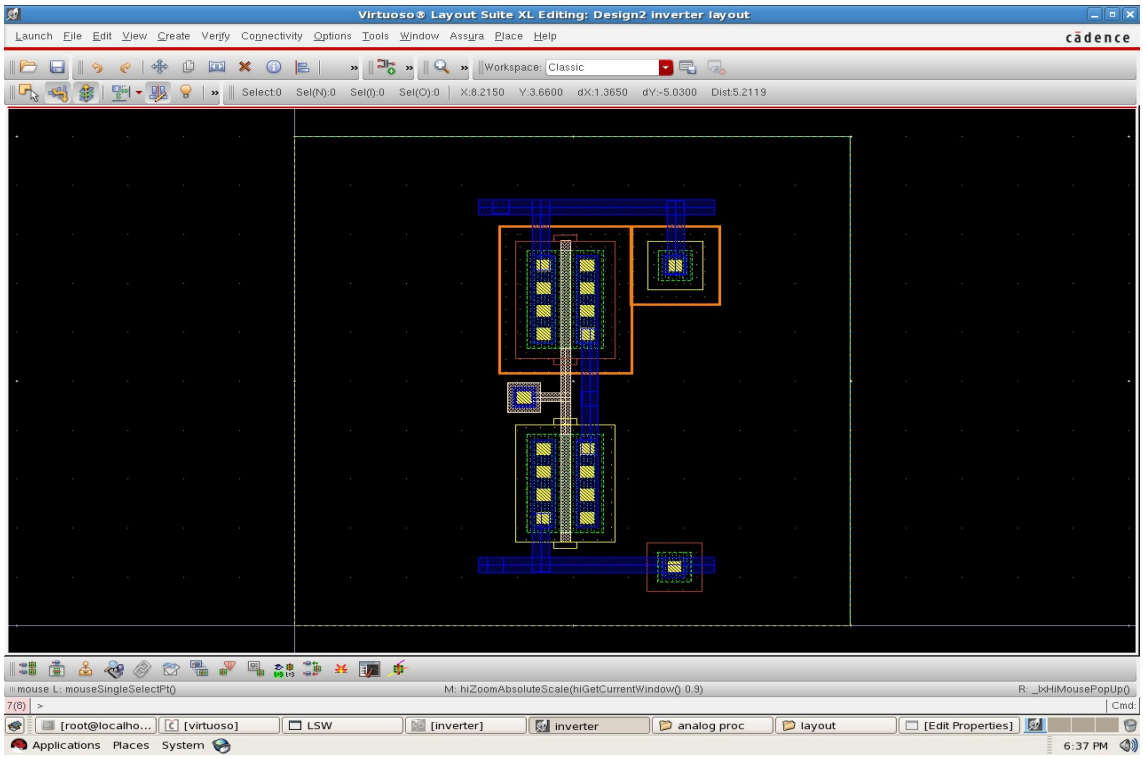
9. place the input pin in front of the poly and connect through a poly path.



10. Now, for connecting the input metal pin to the poly path, a via needs to be placed. Hence, in the layout suite click **Create → Via**. In the Via Definition pull-down menu, select the via M1_POLY1. Click on *Hide*, and place the via on the input pin. Press *Esc*.



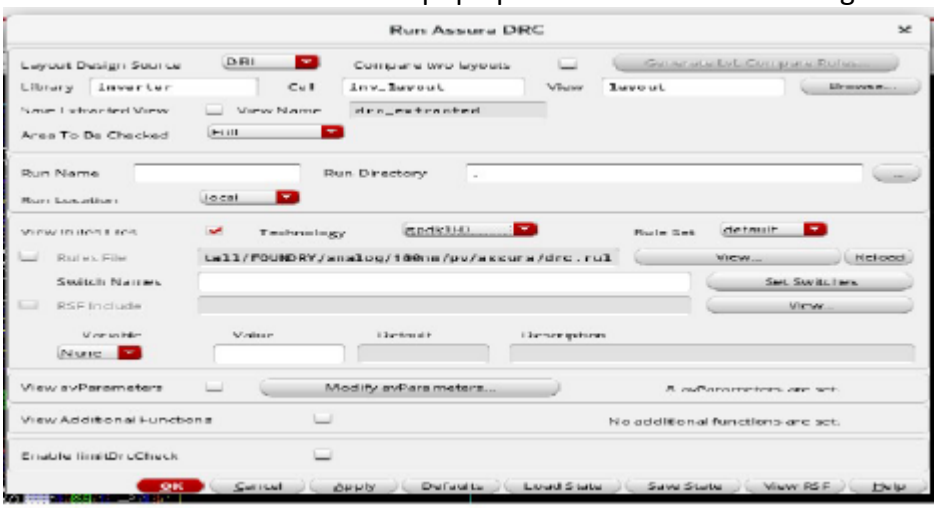
11. Similarly, for the substrate connections, select the via M1_NWELL and place it touching the n-well, and connect it to “vdd” through a metal path. Later, place the via M1_PSUB on the “vss” path, for the substrate connection of nmos device; the **Black** background itself indicates the p-substrate. (p-device resides on n-well and n-device resides directly on p-substrate).



Physical Verification:

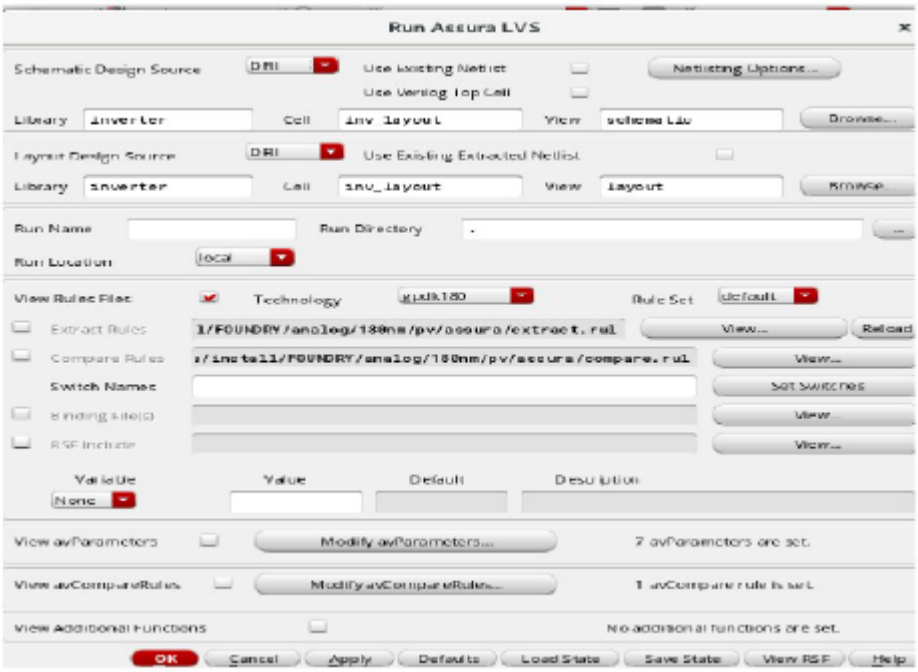
DRC Check

1. As the layout is now complete, its verification can be performed. In the layout suite, click on **Assura → Run DRC**.
2. The **Run Assura DRC window** will pop up. Now enable **View Rule files**, select the **technology** like gpdk180, 90 or 45 and also provide the **Run Name** while running the DRC as shown in the below figure
3. Click on Ok and new **Progress File** Window pop up so click on W**atch log file** so log file will appear.
4. If there are any DRC errors exist in the design **View Layer Window (VLW)** the **Error Layer Window (ELW) appears** by double clicking on the errors in ELW the errors will be highlighted in the layout window.Re-correct all the errors and re-run the DRC again.
5. If there are no DRC errors it will pop up with the window showing no **DRC errors.**

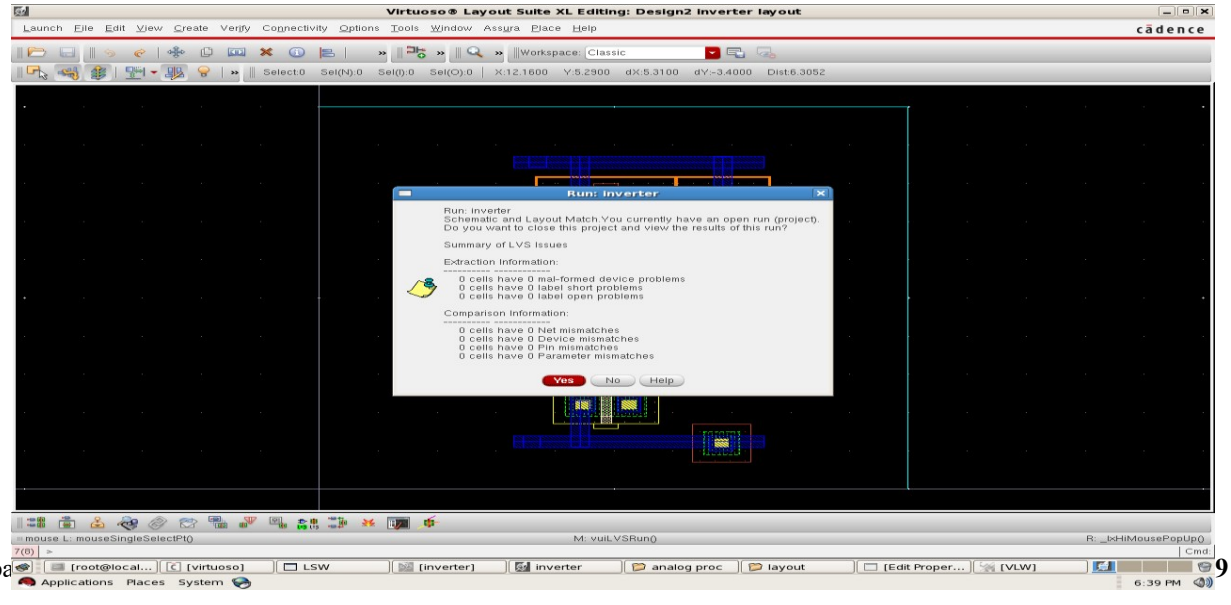


LVS Check

After the DRC check, click on **Assura → Run LVS**, Below window will pop up Select the **Technology File** and specify the run name make all the necessary changes as shown in the figure and hit on OK.and verify the output. Correct the errors.



If Layout and schematic matches one window will pop and notify schematic and layout match as shown in the figure.



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.

Click **Yes** in the form.

LVS debug form appears, and you are redirected to LVS debug environment.

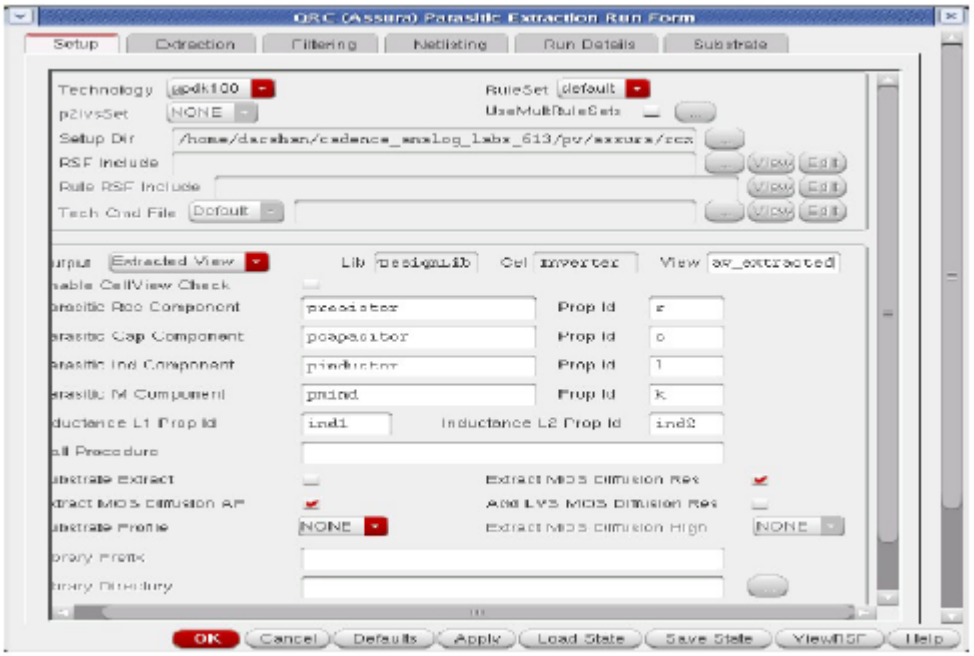
In the LVS debug form you can find the details of mismatches and you need to correct all those mismatches and Re – run the LVS till you will be able to match the schematic and layout.

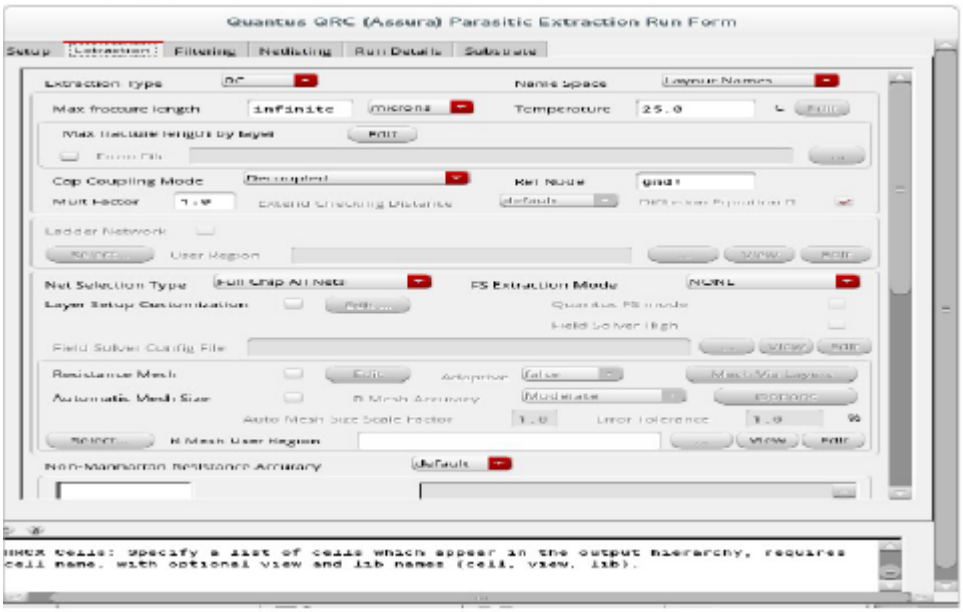
Running Quantus QRC:

1. After the LVS check, click on **Assura →Run Quantus QRC** for RC extraction.

2. Click on **setup** and set the **output** to **Extracted View** and select the **Technology** as shown in the below image**.**

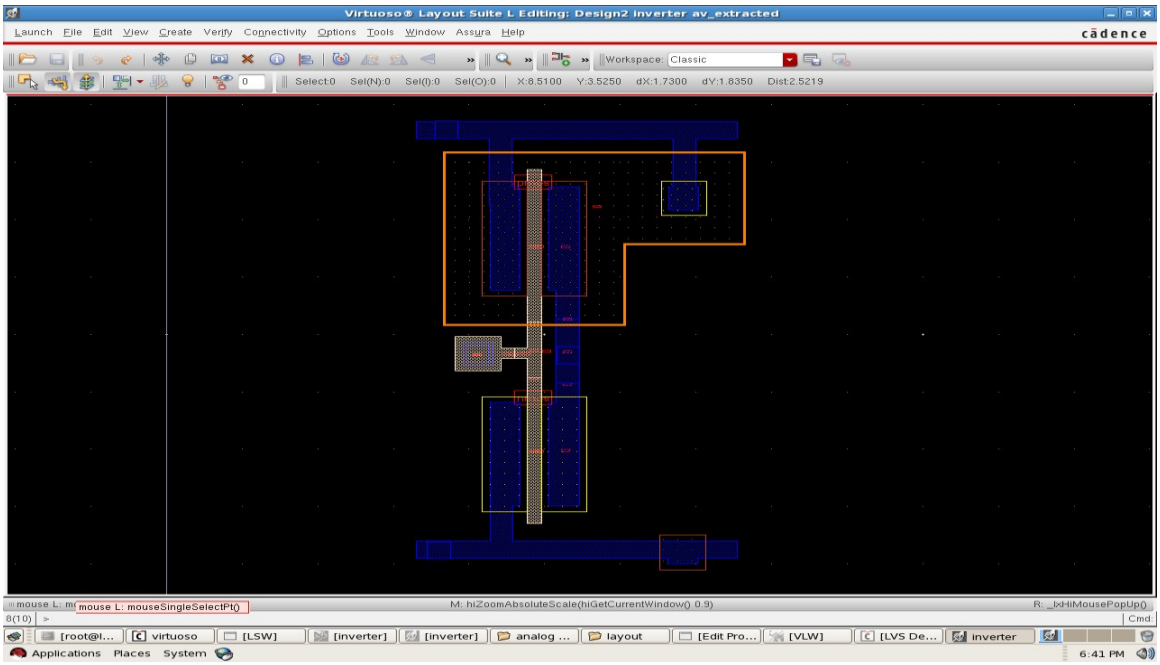
3. Click on **Extraction** from the top of the window and give the **Ref Node** as **gnd!** And also select the **Extraction Type** to **RC .**





After the RCX is run, informs you that Quantus QRC run

1. completed successfully or not.the output is saved in your library as **av_extracted**. In the virtuoso console, open the "inverter" file with view as *av_extracted*, and observe the output. The layout can be enlarged and the parasitic components can be observed. Each components value can be checked, by selecting the component and pressing "q".



If the parasitic component values are beyond the limits, then the layout can be optimized in the layout suite, for the reduction of the parasitic component values; later on, the layout can be back-annotated with the existing parasitic components, and simulation can be performed, for verifying the output.

Result

The CMOS inverter simulation and layout verification has been done for the given specifications.

Experiment No: 2

Draw the schematic of i) 2-input CMOS NAND gate, ii) 2-input CMOS NOR gate for the given specifications, and verify using Transient and DC Analyses. Also draw the layout of i) 2-input CMOS NAND gate, ii) 2-input CMOS NOR gate, and perform physical verification using DRC, ERC and LVS. And Extract RC.

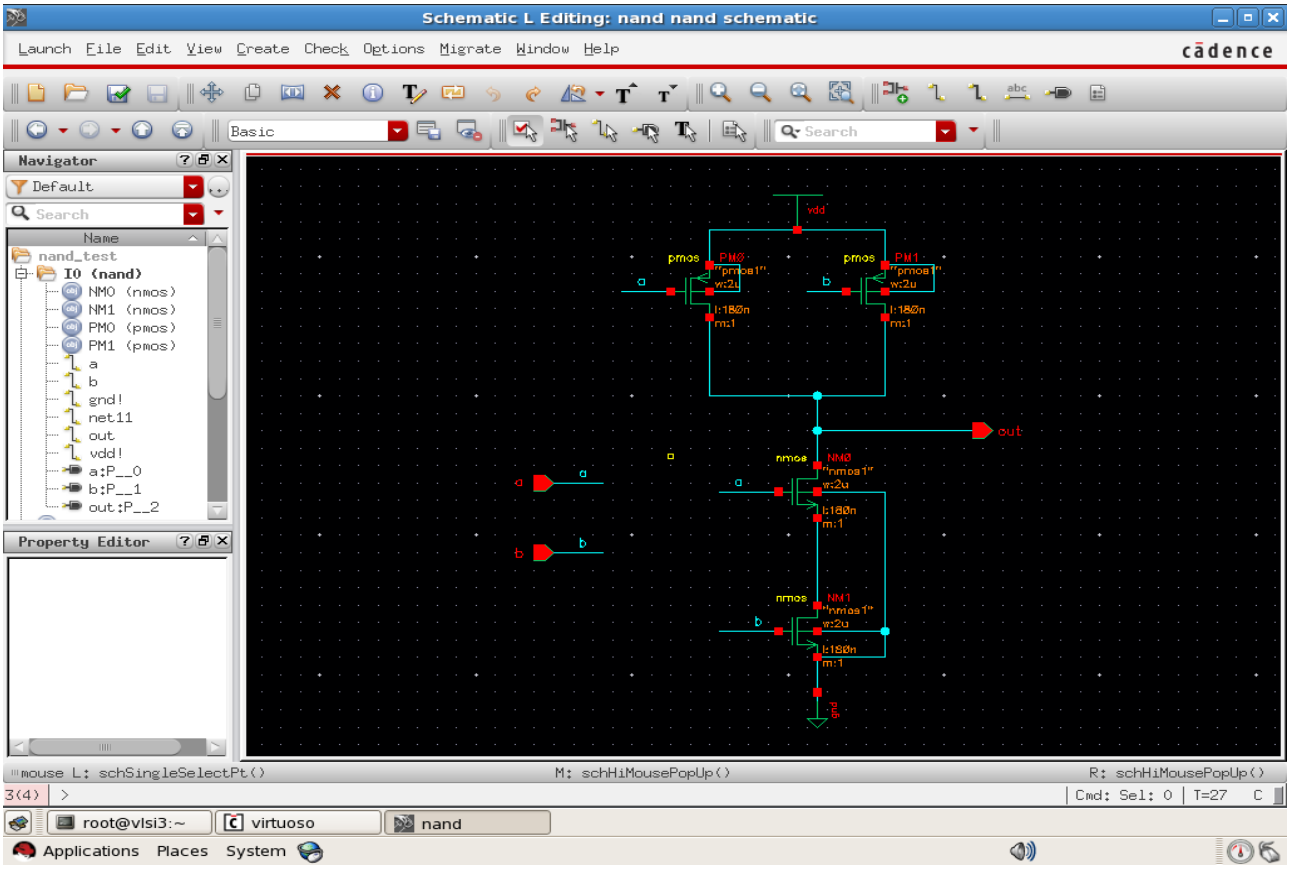
2.1 2- INPUT CMOS NAND GATE

Aim : To simulate the schematic diagrams of the NAND gate, and then to perform the physical verification for the layouts of the same.

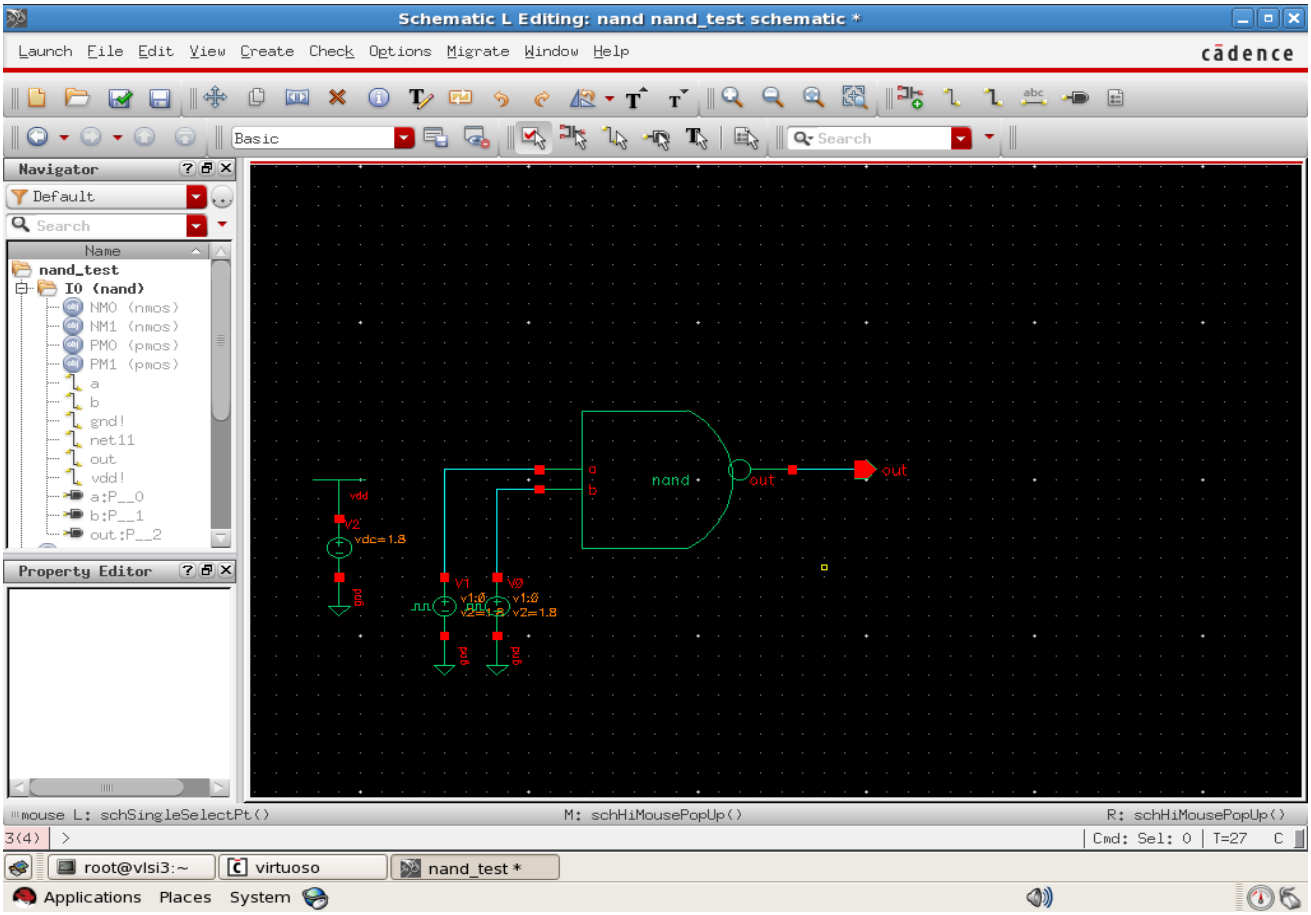
Theory: A CMOS NAND gate circuit uses four MOSFETs just like the NOR gate, except that its transistors are differently arranged. The NAND gate uses two series-connected sinking transistors and two parallel-connected sourcing transistors.

Circuits Diagrams:

1. For the schematic entry, select pmos device from *gpdk180* library, Place the wires for connection.
2. Place “vdd” and “gnd”. Later, place the input pins for A and B, and output pin for Vout. Click on “check & save” and correct the errors, if any.



3. Create the symbol for the schematic, using **create cellview from cellview** , the default symbol itself can be used.



Close the editor windows, and create a new schematic in the virtuoso console, for the test circuit. In the test circuit, retrieve the symbol from your library, and then place “vpulse” from the *analogLib* library for the input signal, with the attributes entered as –

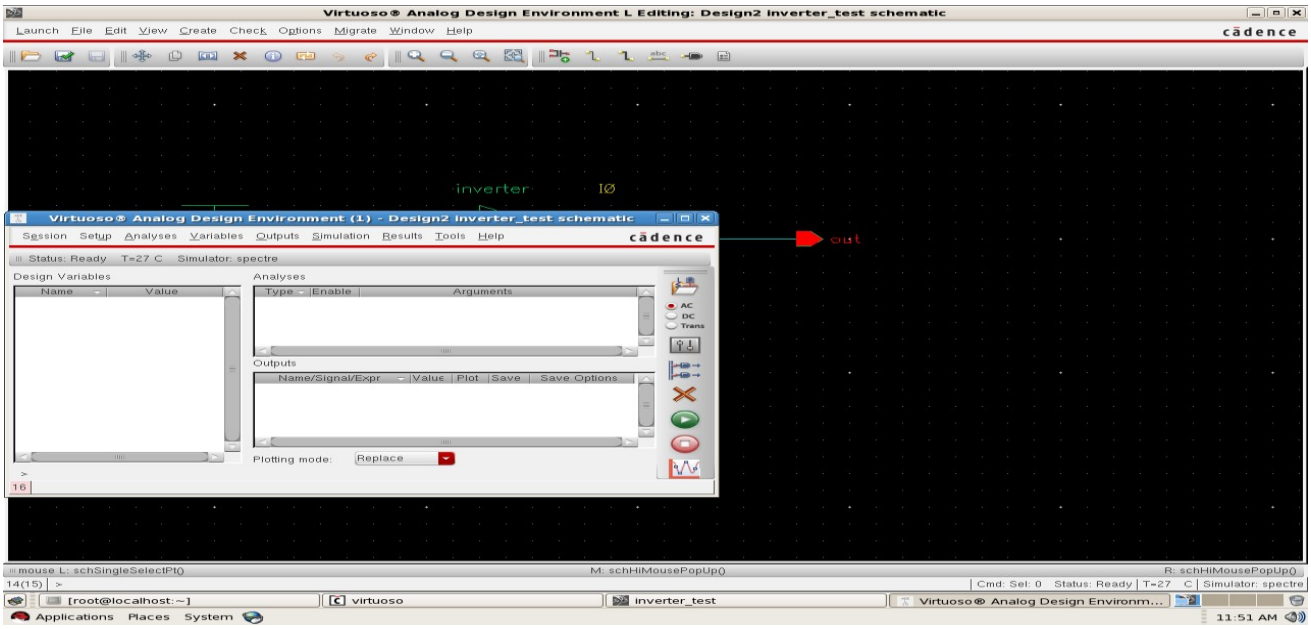
4. In the property window, enter *Voltage1* as **0** and *Voltage2* as **1.8**. Similarly enter *Period* as **40n** and *Pulse width* as **20n**, without the space in between. No need to enter the units; they appear automatically. Place the “vpulse” at the input wire. Connect “gnd” at the other end

5. Same step follow for other one where enter *Voltage1* as **0** and *Voltage2* as **1.8**. Similarly enter *Period* as **20n** and *Pulse width* as **10n**, then change **vdc** parameter as **voltage 1.8** place vdd and gnd at end

Press q for check the parameters of the components.

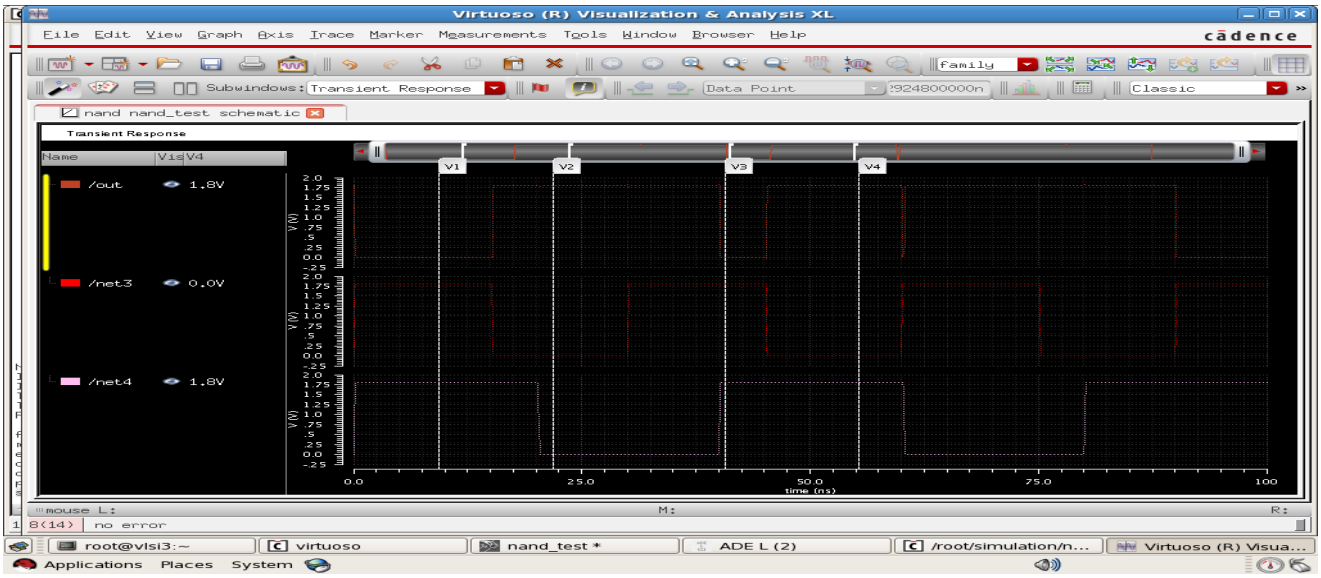
SIMULATION:

In the test circuit’s editor window, click on **Launch → ADE L**. A new window will open.



Click on **Analyses → Choose**. A new window will open, in which select "tran". Fill the stop time as **100n**, and select *liberal*. Later, click on *Apply*. Ensure analysis window updated.

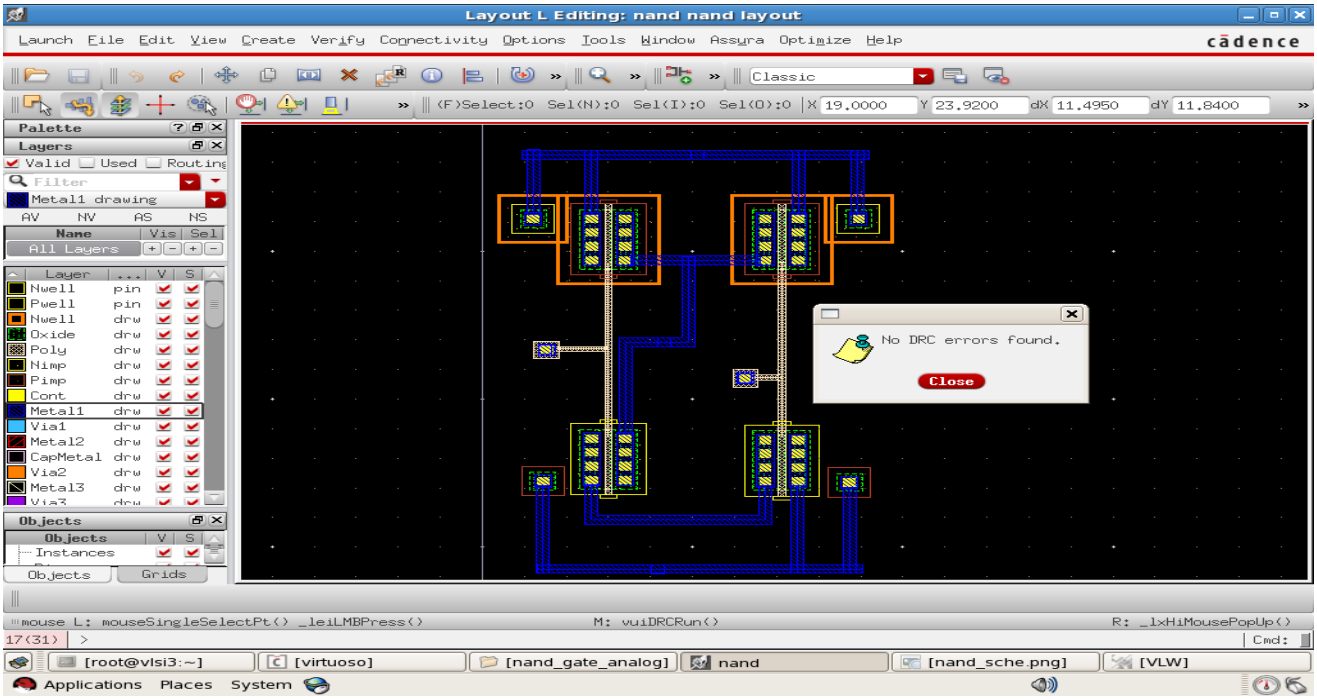
Now to select the stimulus and response points, in the ADE window, click on **Outputs → to be plotted → Select on Schematic**. In the schematic window, click on input and output wires. These wires will become dotted lines when selected. Later, press *Esc*.



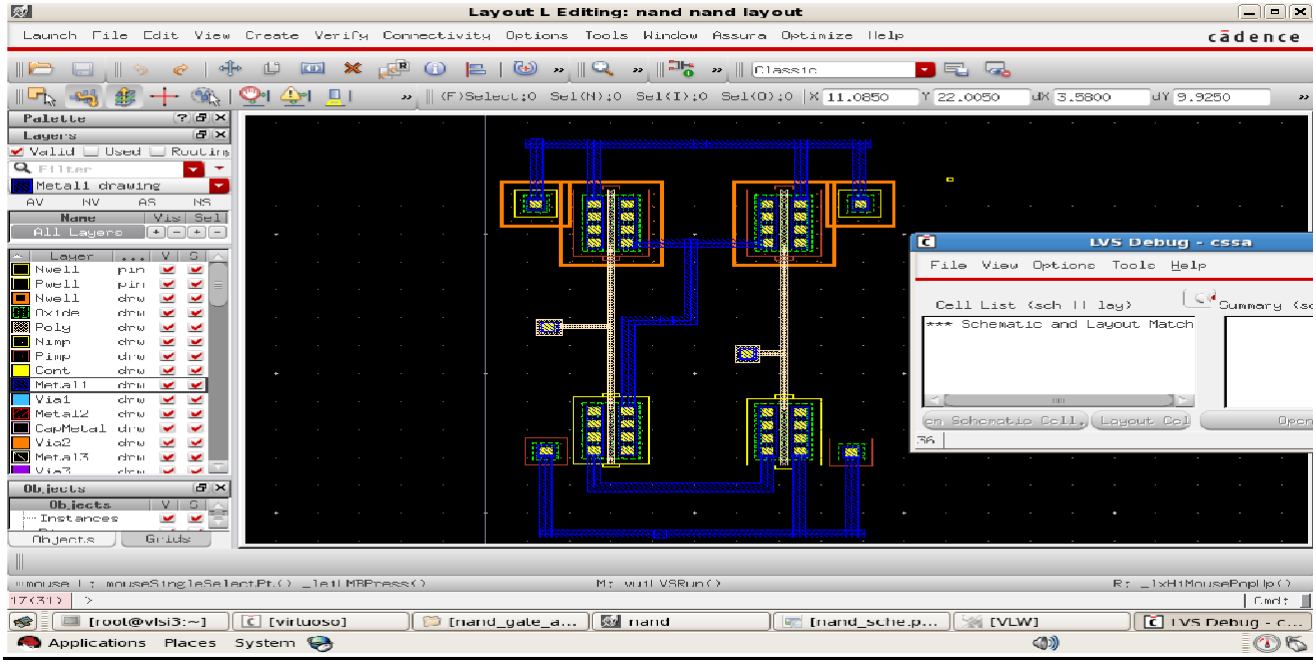
LAYOUT: NAND GATE

1. For preparing the layout of the inverter, all the other windows can be closed, except for the virtuoso console. In the console, open the schematic of the inverter and click on **Launch → Layout XL**. In the *Startup Option*, click on *OK*.
2. In the *New File* option, the tool selects the view as *layout* by default. Click on *OK*.
3. The tool opens the LSW and the Layout suite.
4. Maximize the layout suite and click on **Connectivity → Generate → All from Source**. A *Generate Layout* window will open, with default attributes. Click on *OK*.

5. The layout suite displays a cyan colored box in the first quadrant, which is the Photo-Resist boundary. In addition, in the fourth quadrant, the default layouts of pmos and nmos transistors are displayed, along with four blue squares, which are the nodes - vdd, gnd, input & output.
6. Start doing layout design,
7. As the layout is now complete, its verification can be performed. In the layout suite, click on **Assura → Run DRC**. Verify the output. If there are errors, the tool will highlight those areas in **White** color. The errors will be displayed in the ELW, and the location of each error can be known, by selecting the error in ELW, and then clicking on the arrow mark available in ELW. Correct those errors and rerun DRC.

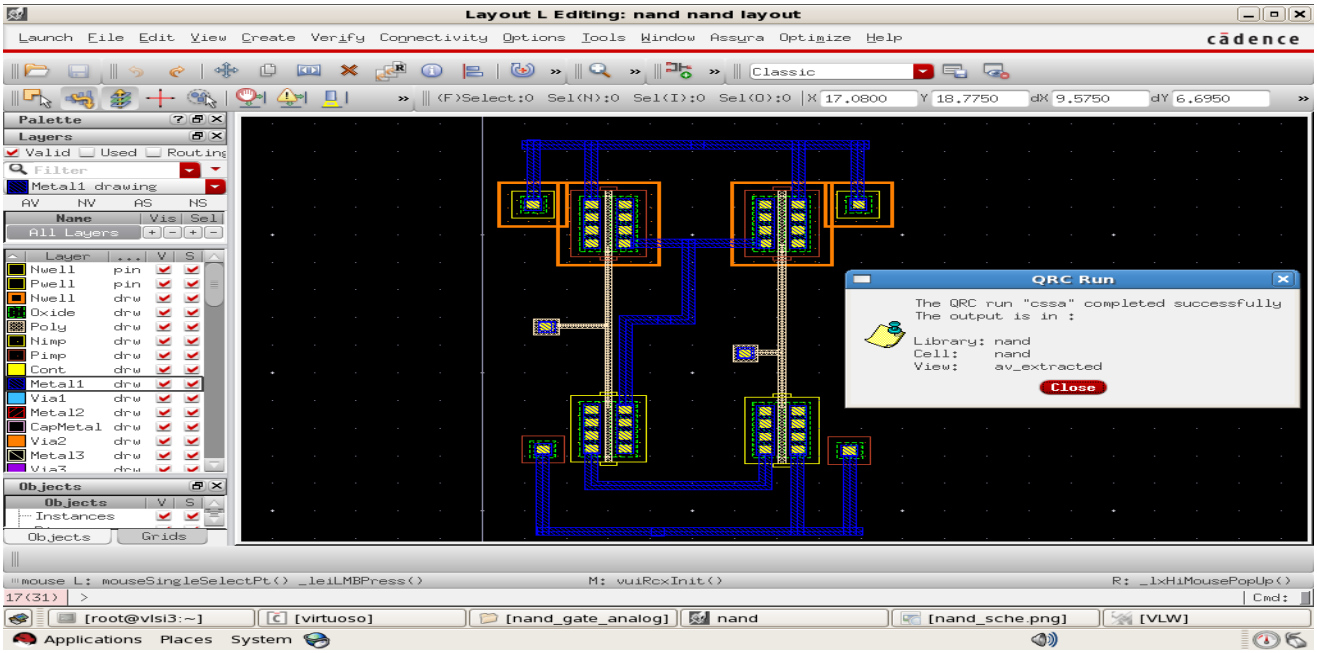


8. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.

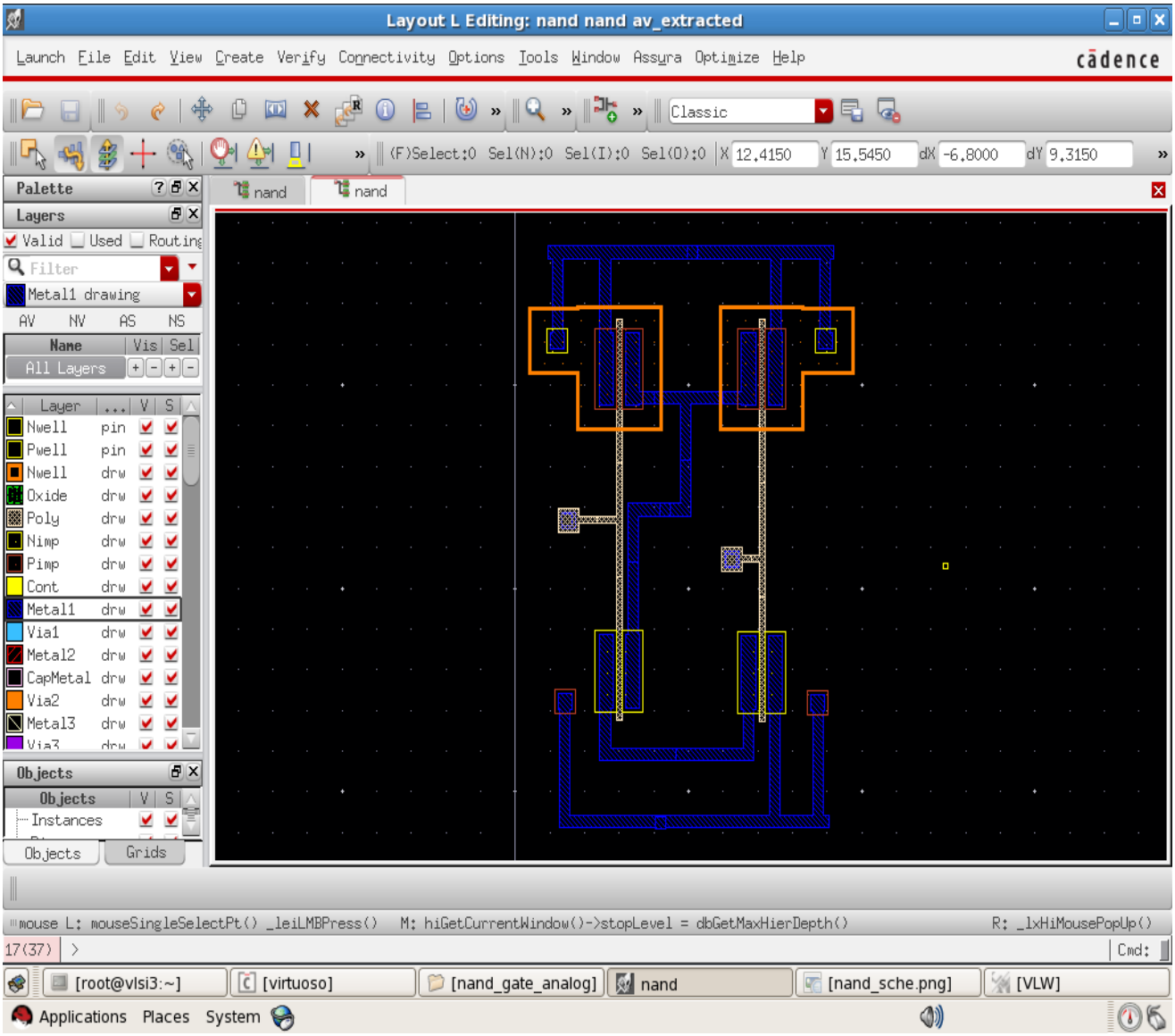


9. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.

10. After the LVS check, click on **Assura → Run RCX**. Click *OK* on the form that appears.



11. After the RCX is run, the output is saved in your library as **av_extracted**. In the virtuoso console, open the "inverter" file with view as *av_extracted*, and observe the output. The layout can be enlarged and the parasitic components can be observed. Each components value can be checked, by selecting the component and pressing "q".



12. If the parasitic component values are beyond the limits, then the layout can be optimized in the layout suite, for the reduction of the parasitic component values; later on, the layout can be back-annotated with the existing parasitic components, and simulation can be performed, for verifying the output.

2.2 . 2- I/P NOR GATE:

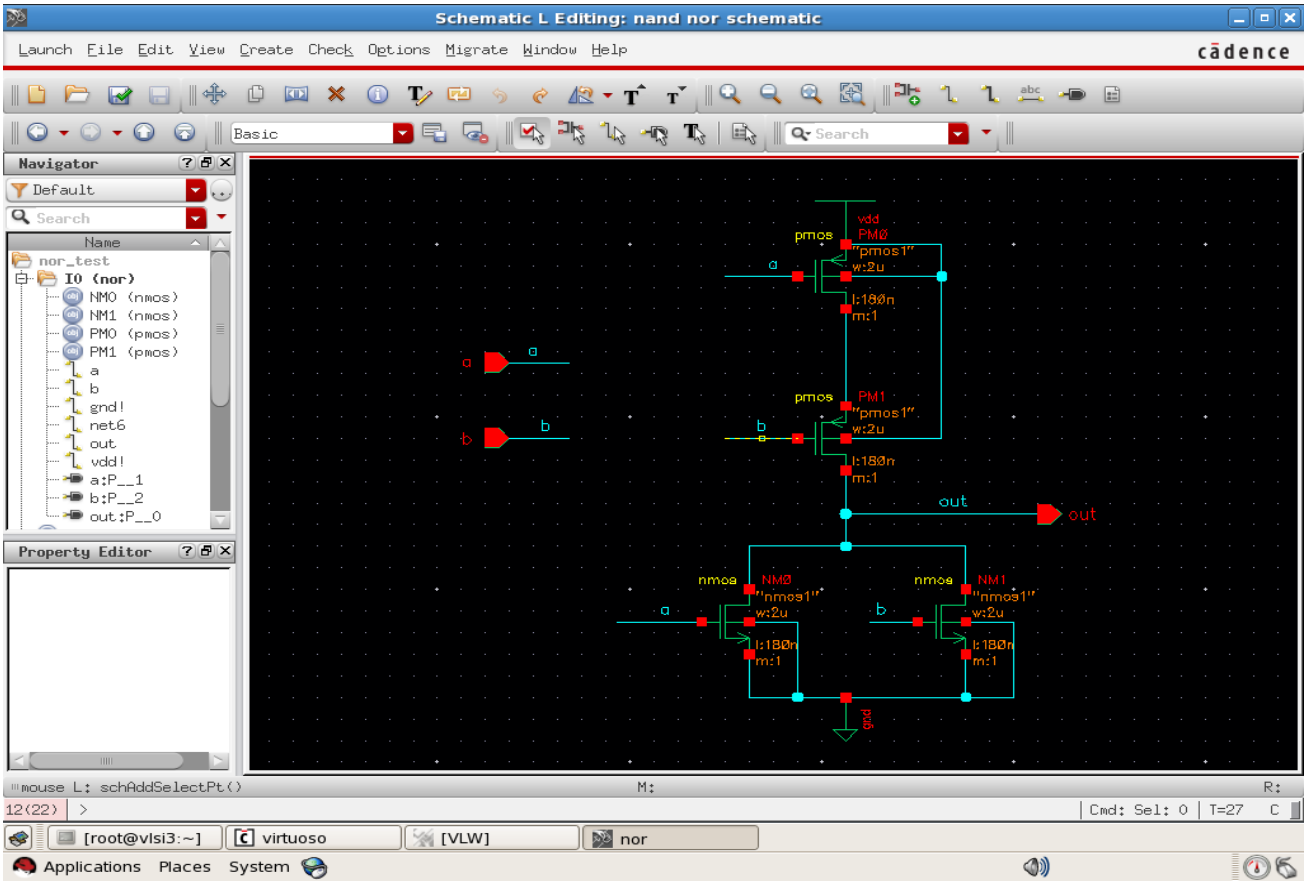
Aim : To simulate the schematic diagrams of the NOR gate, and then to perform the physical verification for the layouts of the same.

Theory:A CMOS NOR gate circuit uses four MOSFETs just like the NAND gate, except that its transistors are differently arranged. Instead of two paralleled sourcing (upper) transistors connected to Vdd and two series-connected sinking (lower) transistors connected to ground, the NOR gate uses two series-connected sourcing transistors and two parallel-connected sinking transistors.

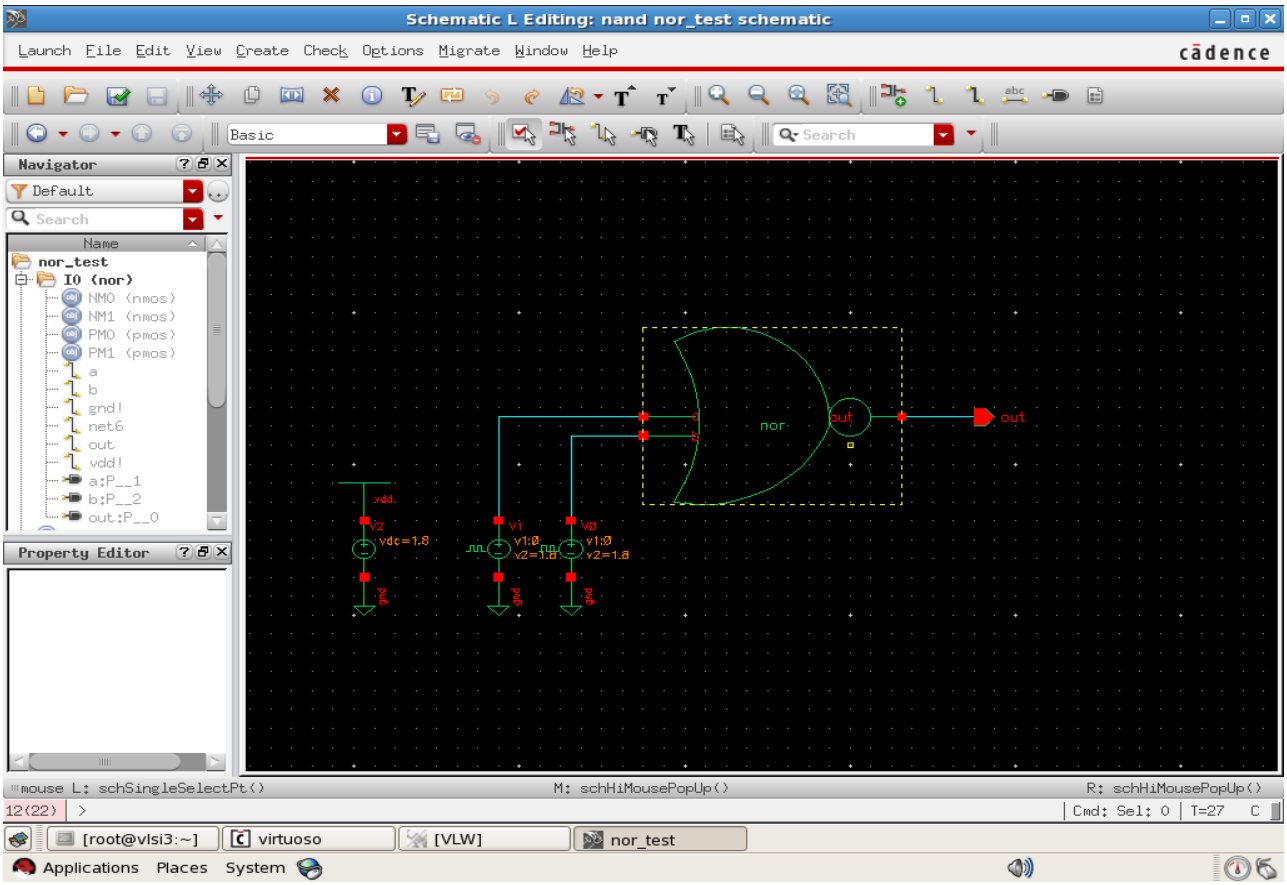
Circuits Diagrams:

1. For the schematic entry, select pmos device from *gpdk180* library, Place the wires for connection.

2. Place “vdd” and “gnd”. Later, place the input pins for A and B, and output pin for Vout. Click on “check & save” and correct the errors, if any.



Create the symbol for the schematic, using **create cellview from cellview** , the default symbol itself can be used.



Close the editor windows, and create a new schematic in the virtuoso console, for the test circuit. In the test circuit, retrieve the symbol from your library, and then place "vpulse" from the *analogLib* library for the input signal, with the attributes entered as –

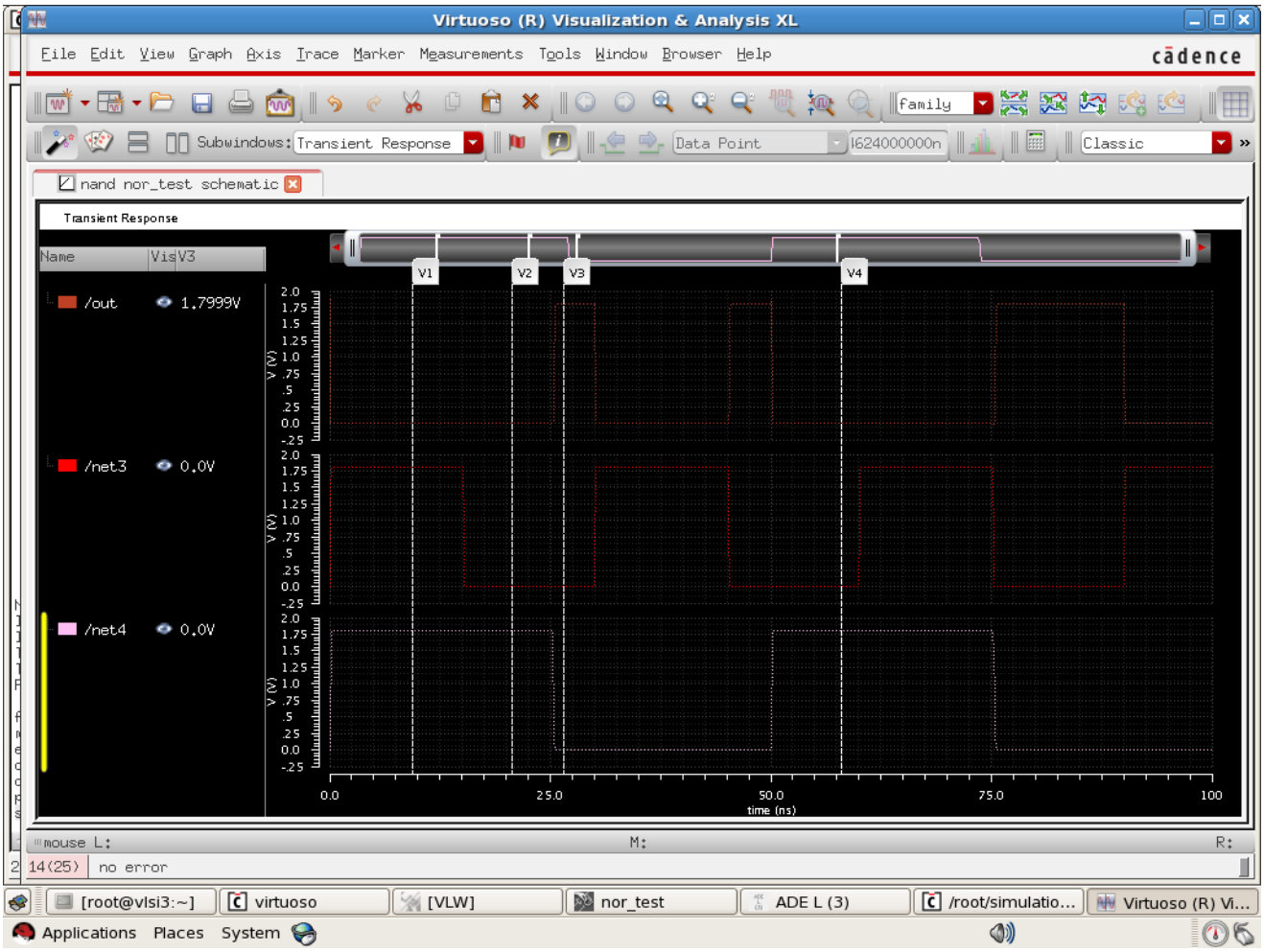
1. In the property window, enter *Voltage1* as **0** and *Voltage2* as **1.8**. Similarly enter *Period* as **40n** and *Pulse width* as **20n**, without the space in between. No need to enter the units; they appear automatically. Place the "vpulse" at the input wire. Connect "gnd" at the other end

Same step follow for other one where enter *Voltage1* as **0** and *Voltage2* as **1.8**. Similarly enter *Period* as **20n** and *Pulse width* as **10n**, then change **vdc** parameter as **voltage 1.8** place vdd and gnd at end

Press q for check the parameters of the components.

SIMULATION:

In the test circuit's editor window, click on **Launch → ADE L**. A new window will open.



Click on **Analyses → Choose**. A new window will open, in which select "tran". Fill the stop time as **100n**, and select *liberal*. Later, click on *Apply*. Ensure analysis window updated.

Now to select the stimulus and response points, in the ADE window, click on **Outputs → to be plotted → Select on Schematic**. In the schematic window, click on input and output wires. These wires will become dotted lines when selected. Later, press *Esc*.

LAYOUT:

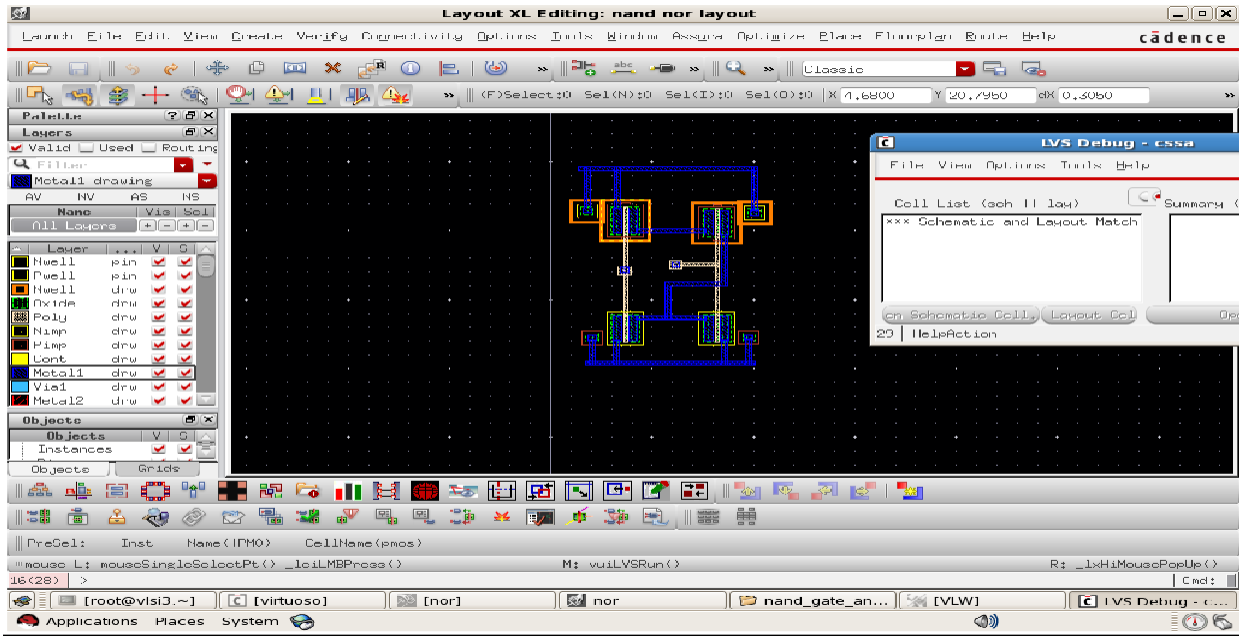
1. For preparing the layout of the inverter, all the other windows can be closed, except for the virtuoso console. In the console, open the schematic of the inverter and click on **Launch → Layout XL**. In the *Startup Option*, click on *OK*.
2. In the *New File* option, the tool selects the view as *layout* by default. Click on *OK*.
3. The tool opens the LSW and the Layout suite.
4. Maximize the layout suite and click on **Connectivity → Generate → All from Source**. A *Generate Layout* window will open, with default attributes. Click on *OK*.

5. The layout suite displays a cyan colored box in the first quadrant, which is the Photo-Resist boundary. In addition, in the fourth quadrant, the default layouts of pmos and nmos transistors are displayed, along with four blue squares, which are the nodes - vdd, gnd, input & output.

Start doing layout design,

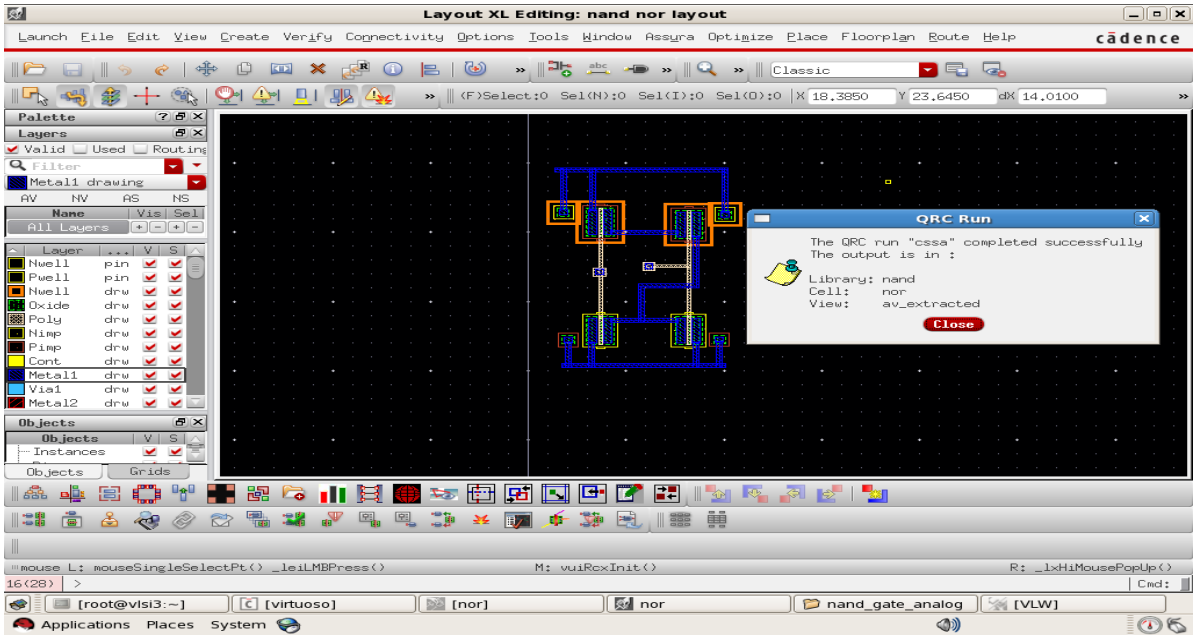
6. As the layout is now complete, its verification can be performed. In the layout suite, click on **Assura → Run DRC**. Verify the output. If there are errors, the tool will highlight those areas in **White** color. The errors will be displayed in the ELW, and the location of each error can be known, by selecting the error in ELW, and then clicking on the arrow mark available in ELW. Correct those errors and rerun DRC.

7. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.

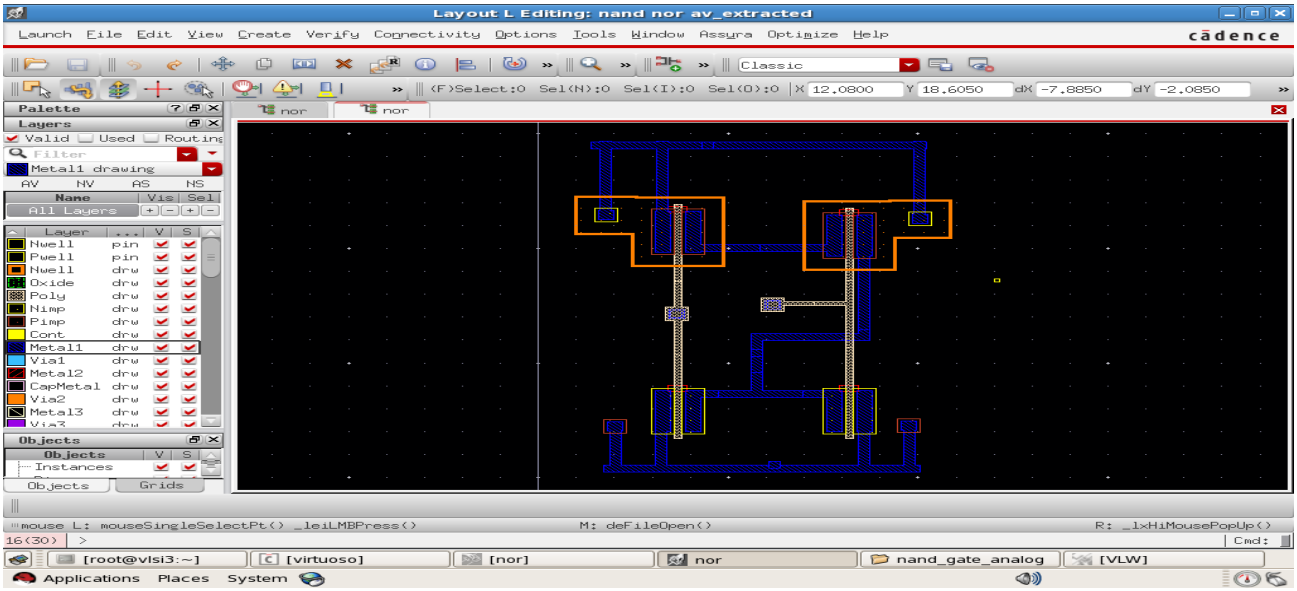


8. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.

9. After the LVS check, click on **Assura → Run RCX**. Click *OK* on the form that appears.



10. After the RCX is run, the output is saved in your library as **av_extracted**. In the virtuoso console, open the "inverter" file with view as *av_extracted*, and observe the output. The layout can be enlarged and the parasitic components can be observed. Each components value can be checked, by selecting the component and pressing "q".



11. If the parasitic component values are beyond the limits, then the layout can be optimized in the layout suite, for the reduction of the parasitic component values; later on, the layout can be back-annotated with the existing parasitic components, and simulation can be performed, for verifying the output.

Result

The NAND and NOR gate simulation and layout verification has been done successfully.

Experiment No: 3

Draw the schematic of Transmission Gate for the given specifications, and verify using Transient and DC Analyses. Draw the layout of Transmission Gate and perform physical verification using DRC, ERC and LVS and Extract RC.

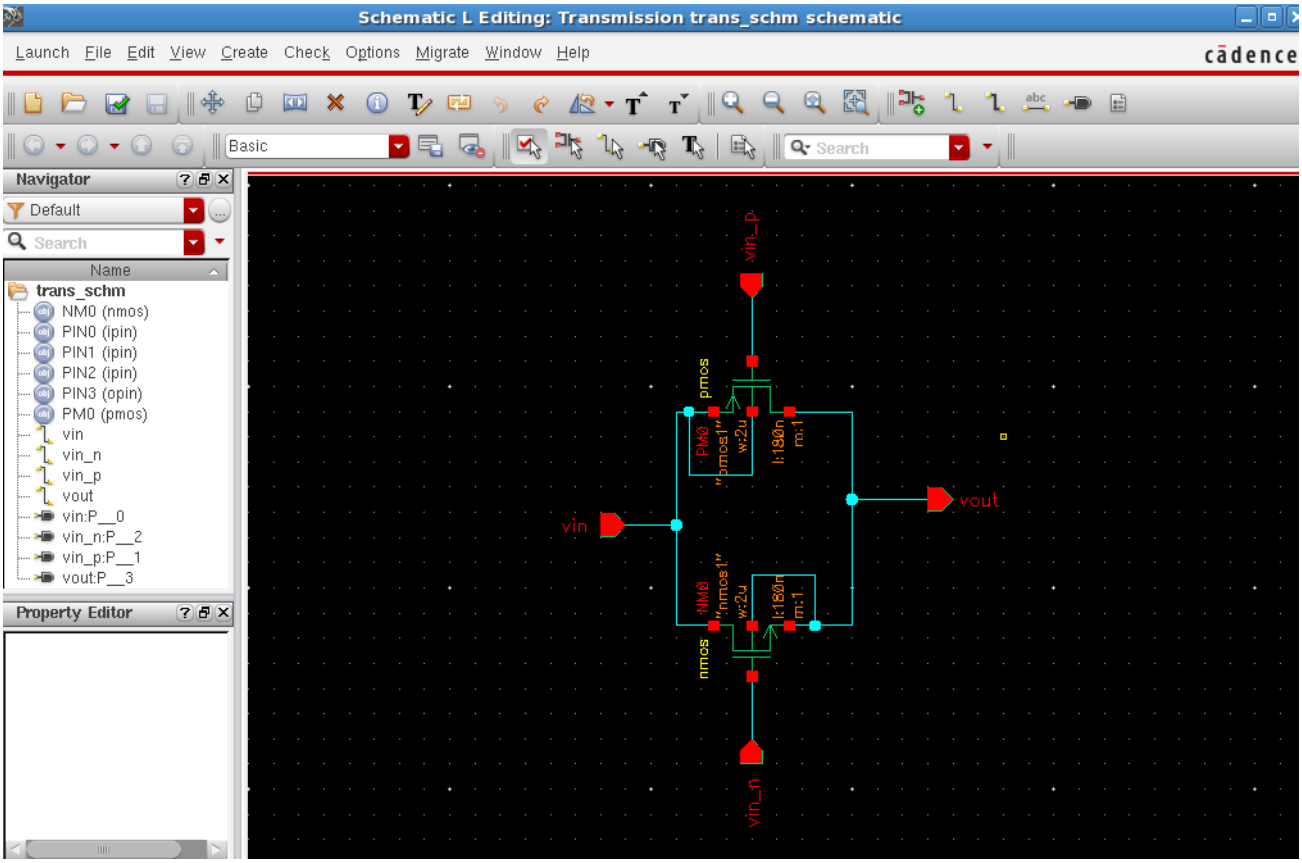
Aim: To simulate the schematic of the Transmission Gate, and then to perform the physical verification for the layout of the same.

Theory: A transmission gate is made up of two field-effect transistors. The two transistors, an n-channel MOSFET and a p-channel MOSFET, are connected in parallel with this, however, only the drain and source terminals of the two transistors are connected together. Their gate terminals are connected to each other by a NOT gate (inverter), to form the control terminal. Transmission gates are used in order to implement electronic switches and analog multiplexers.

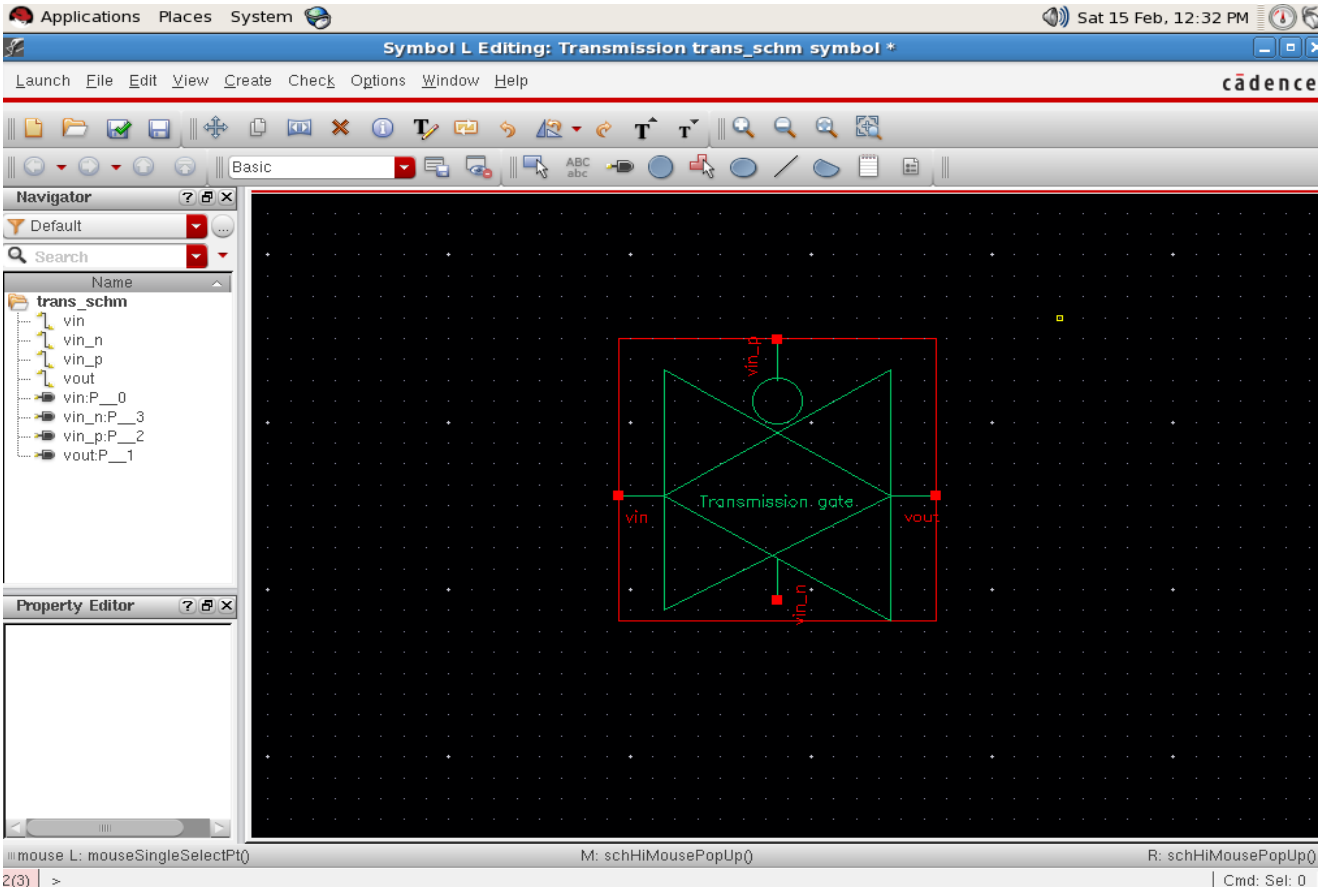
Circuits Diagrams:

1. For the schematic entry, select pmos and nmos device from *gpdk180* library, Place the wires for connection.

2. Later, place the input pins for A and B, and output pin for Vout. Click on "check & save" and correct the errors, if any.



Create the symbol for the schematic, using **create cellview from cellview** , the default symbol itself can be used.

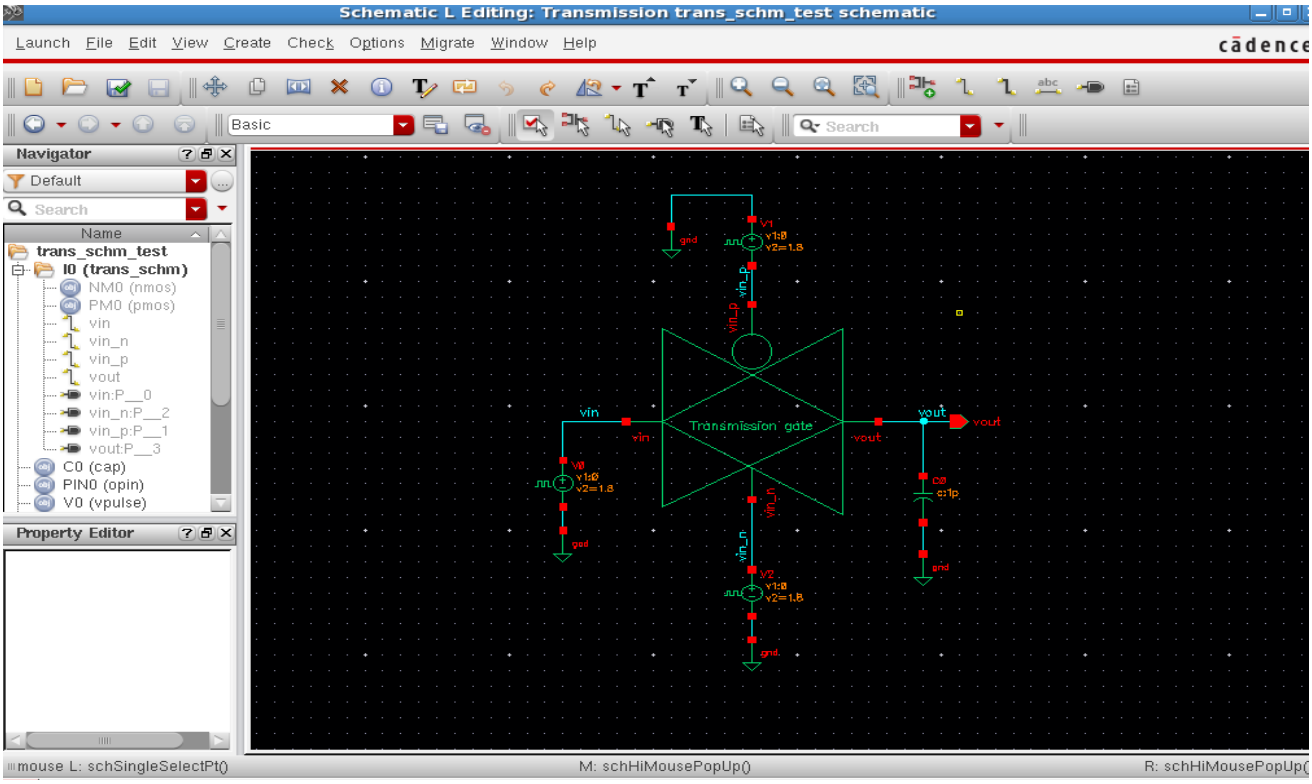


Close the editor windows, and create a new schematic in the virtuoso console, for the test circuit. In the test circuit, retrieve the symbol from your library, and then place "vpulse" from the *analogLib* library for the input signal, with the attributes entered appropriately.

Press q for check the parameters of the components.

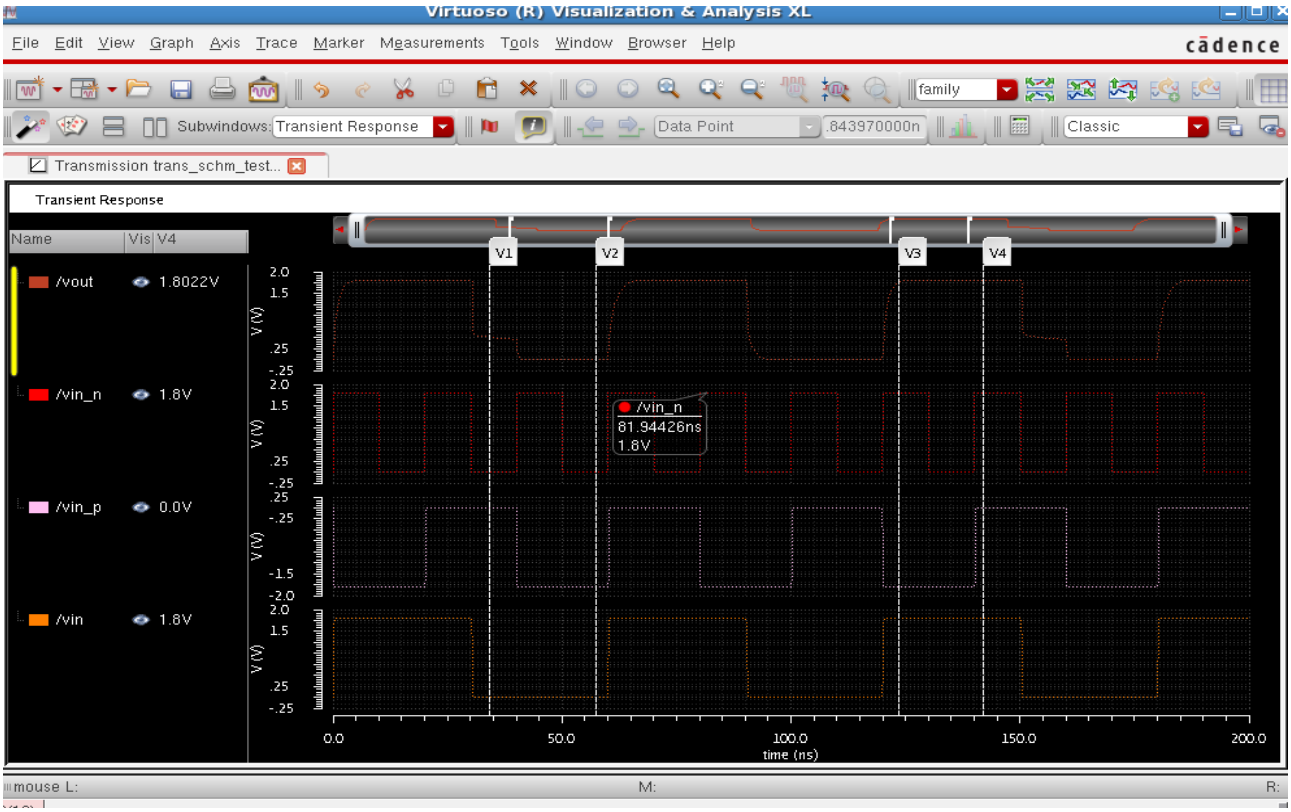
SIMULATION:

In the test circuit's editor window, click on **Launch → ADE L**. A new window will open.



Click on **Analyses → Choose**. A new window will open, in which select "tran". Fill the stop time as **100n**, and select *liberal*. Later, click on *Apply*. Ensure analysis window updated.

Now to select the stimulus and response points, in the ADE window, click on **Outputs → to be plotted → Select on Schematic**. In the schematic window, click on input and output wires. These wires will become dotted lines when selected. Later, press *Esc*.



LAYOUT:

12. For preparing the layout , all the other windows can be closed, except for the virtuoso console. In the console, open the schematic of the inverter and click on **Launch → Layout XL**. In the *Startup Option*, click on *OK*.

13. In the *New File* option, the tool selects the view as *layout* by default. Click on *OK*.

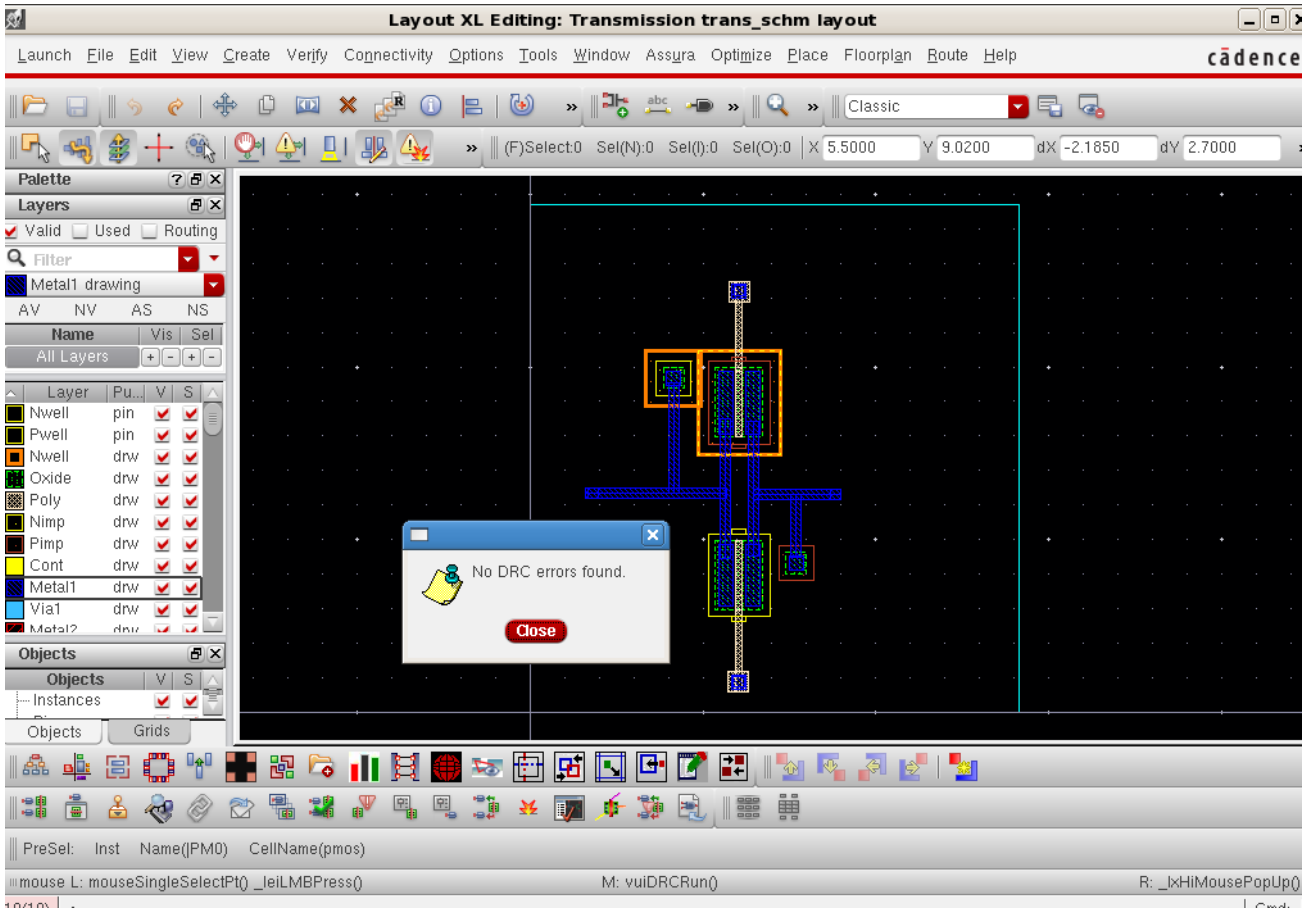
14. The tool opens the LSW and the Layout suite.

15. Maximize the layout suite and click on **Connectivity → Generate → All from Source**. A *Generate Layout* window will open, with default attributes. Click on *OK*.

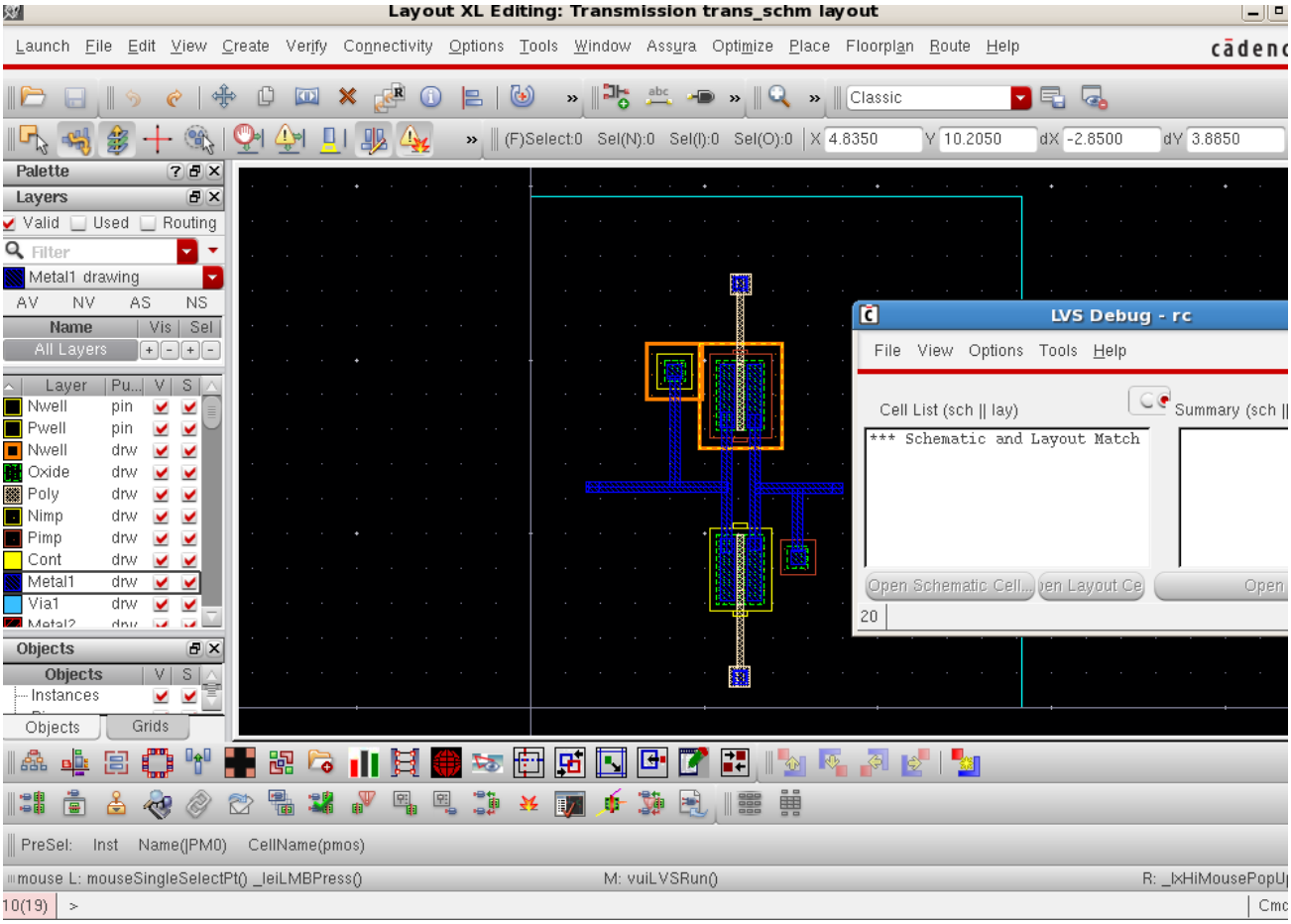
16. The layout suite displays a cyan colored box in the first quadrant, which is the Photo-Resist boundary. In addition, in the fourth quadrant, the default layouts of pmos and nmos transistors are displayed, along with four blue squares, which are the nodes - vdd, gnd, input & output.

Start doing layout design,

17. As the layout is now complete, its verification can be performed. In the layout suite, click on **Assura → Run DRC**. Verify the output. If there are errors, the tool will highlight those areas in **White** color. The errors will be displayed in the ELW, and the location of each error can be known, by selecting the error in ELW, and then clicking on the arrow mark available in ELW. Correct those errors and rerun DRC.

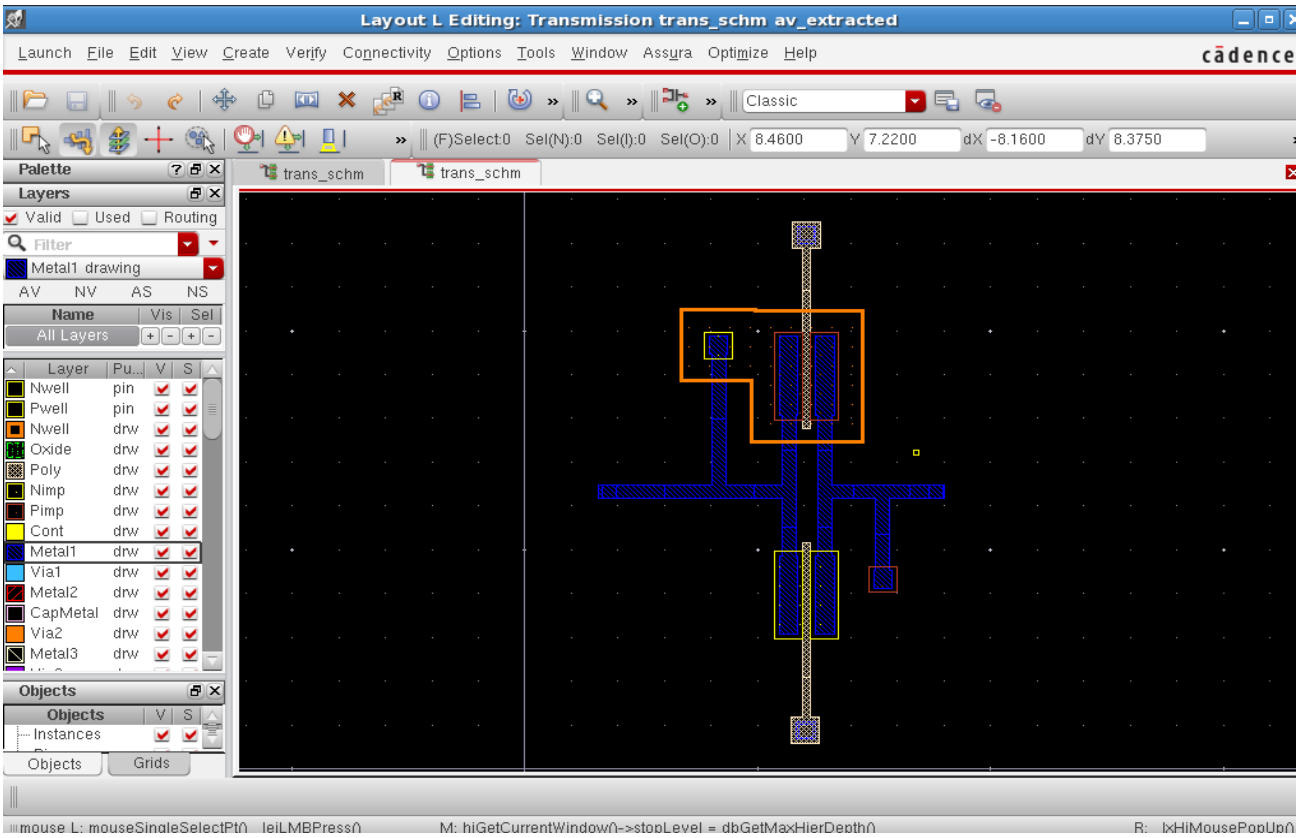


18. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.



19. After the LVS check, click on **Assura → Run RCX**. Click *OK* on the form that appears.

20. After the RCX is run, the output is saved in your library as **av_extracted**. In the virtuoso console, open the "inverter" file with view as *av_extracted*, and observe the output. The layout can be enlarged and the parasitic components can be observed. Each components value can be checked, by selecting the component and pressing "q".



21. If the parasitic component values are beyond the limits, then the layout can be optimized in the layout suite, for the reduction of the parasitic component values; later on, the layout can be back-annotated with the existing parasitic components, and simulation can be performed, for verifying the output.

Result

The transmission gate circuit simulation and layout verification has been done for the given specifications.

Experiment No: 4

Draw the schematic of Common source amplifier and Common drain amplifier for the given specifications, and verify using Transient, DC and AC Analyses. Draw the layout these amplifiers, and perform physical verification using DRC, ERC and LVS and Extract RC .

Aim: To simulate the schematic diagrams of the common source and common drain amplifiers, and then to perform the physical verification for the layouts of the same.

Theory:

Common source: This FET configuration is probably the most widely used. The common source circuit provides a medium input and output impedance levels. Both current and voltage gain can be described as medium, but the output is the inverse of the input, i.e. 180° phase change. This provides a good overall performance and as such it is often thought of as the most widely used configuration.

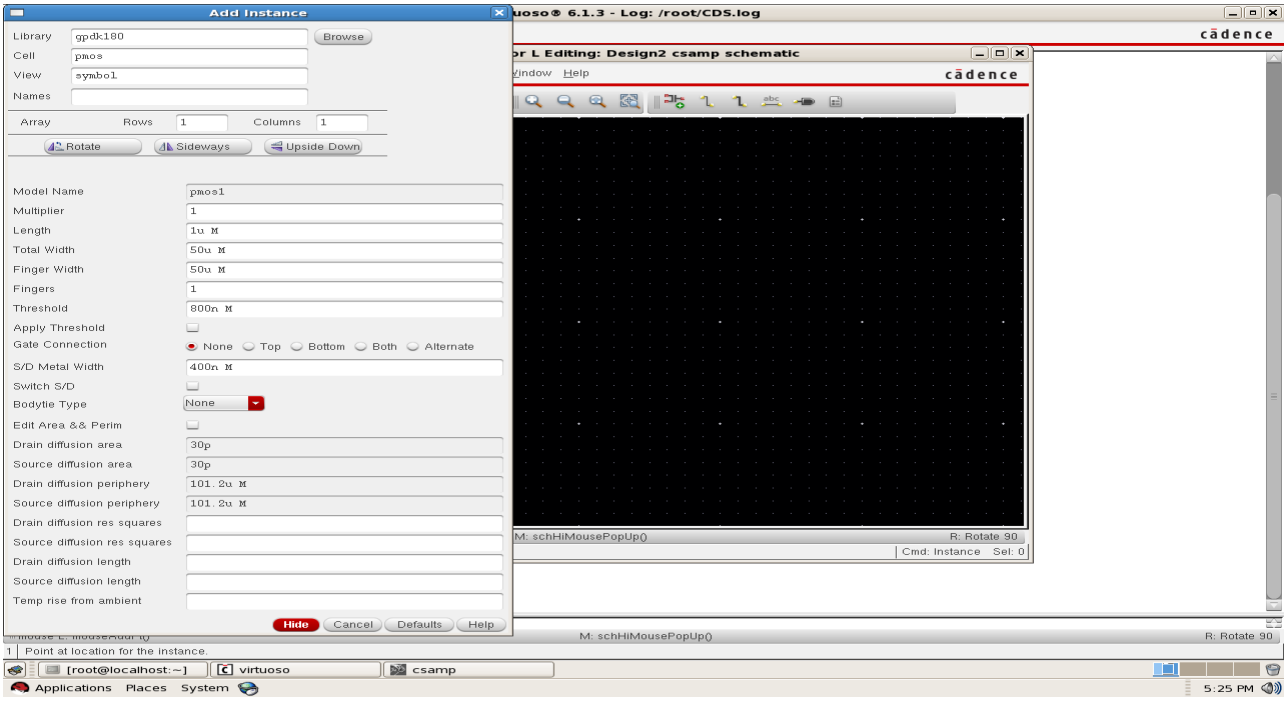
Common drain: This FET configuration is also known as the source follower. The reason for this is that the source voltage follows that of the gate. Offering a high input impedance and a low output impedance it is widely used as a buffer. The voltage gain is unity, although current gain is high. The input and output signals are in phase.

Circuit diagrams:

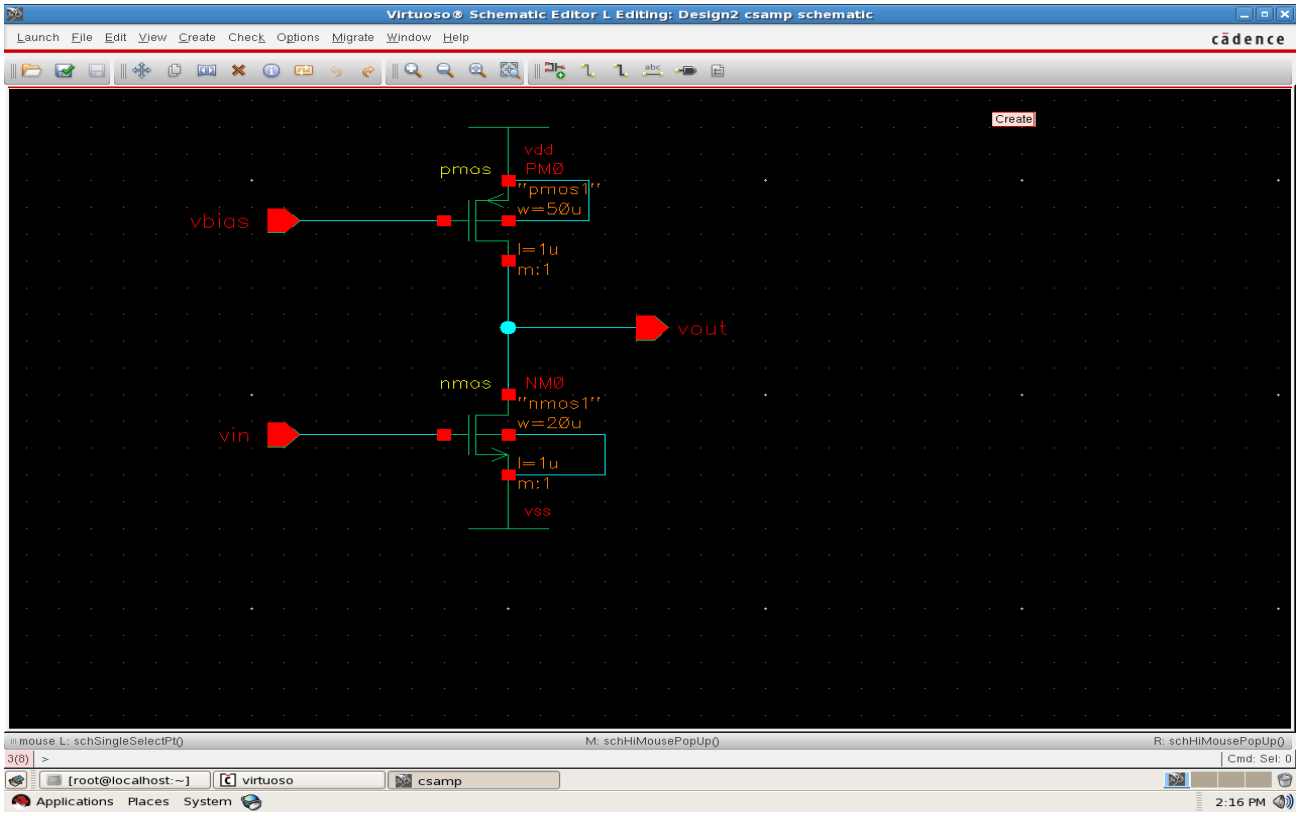
Procedure:

a) COMMON SOURCE AMPLIFIER:

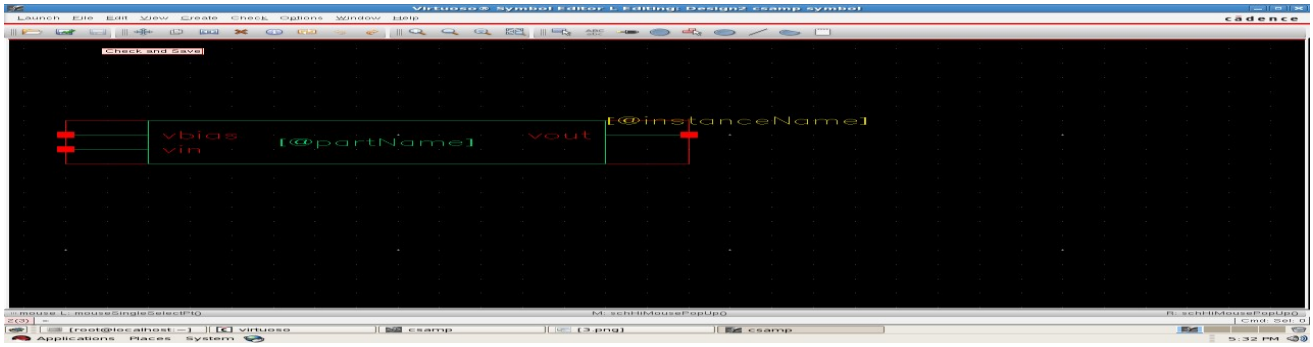
1. For the schematic entry, select pmos device from *gpdk180* library, and edit the properties as Length = **1** micron and Width = **50** microns. (type "u" for micron). Similarly, place the nmos device with Length = **1** micron and Width = **20** microns. Place the wires for connection.



2. Place “vdd” and “vss”. Later, place the input pins for Vbias and Vin, and output pin for Vout. Click on “check & save” and correct the errors, if any.



3. Create the symbol for the schematic; the default symbol itself can be used.

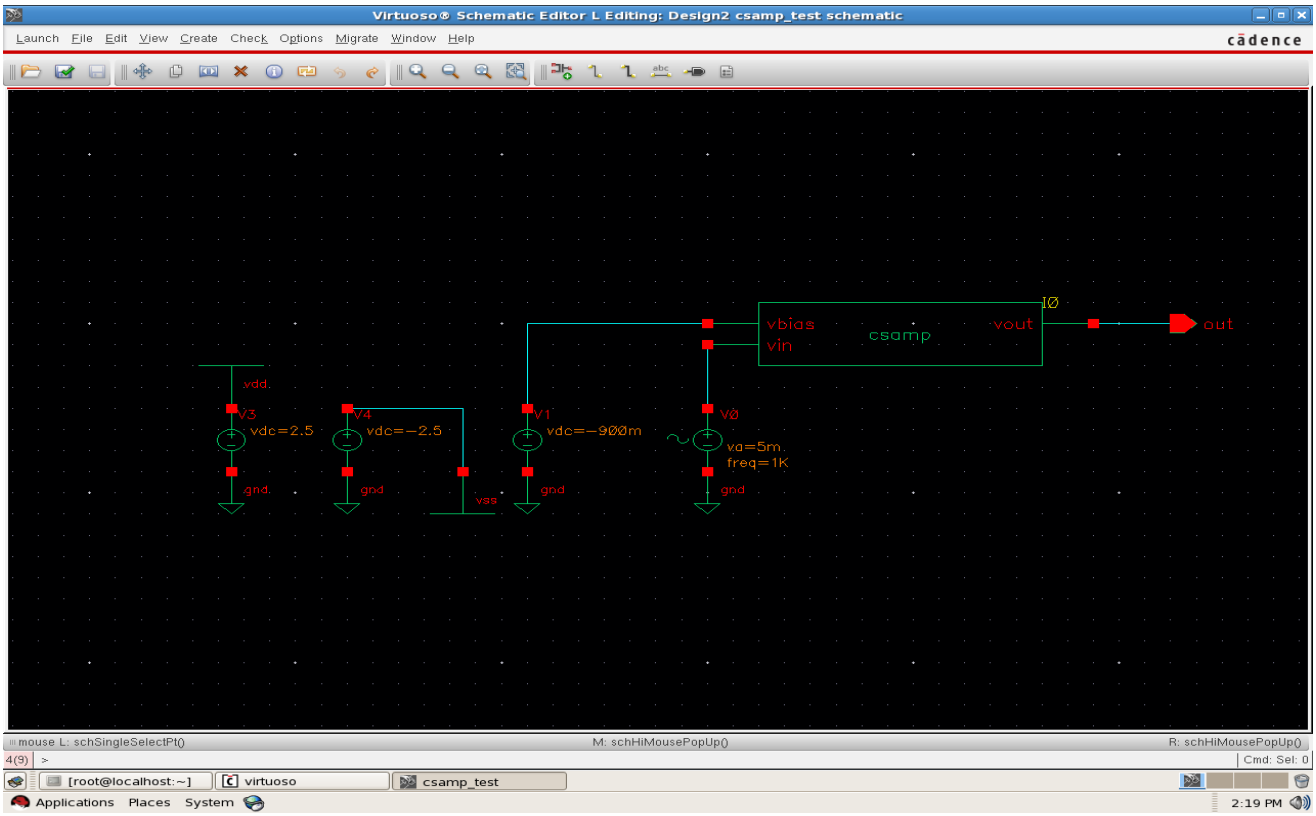


4. Close the editor windows, and create a new schematic in the virtuoso console, for the test circuit. In the test circuit, retrieve the symbol from your library, and then place “vsin” from the *analogLib* library for the input signal, with the attributes entered as –

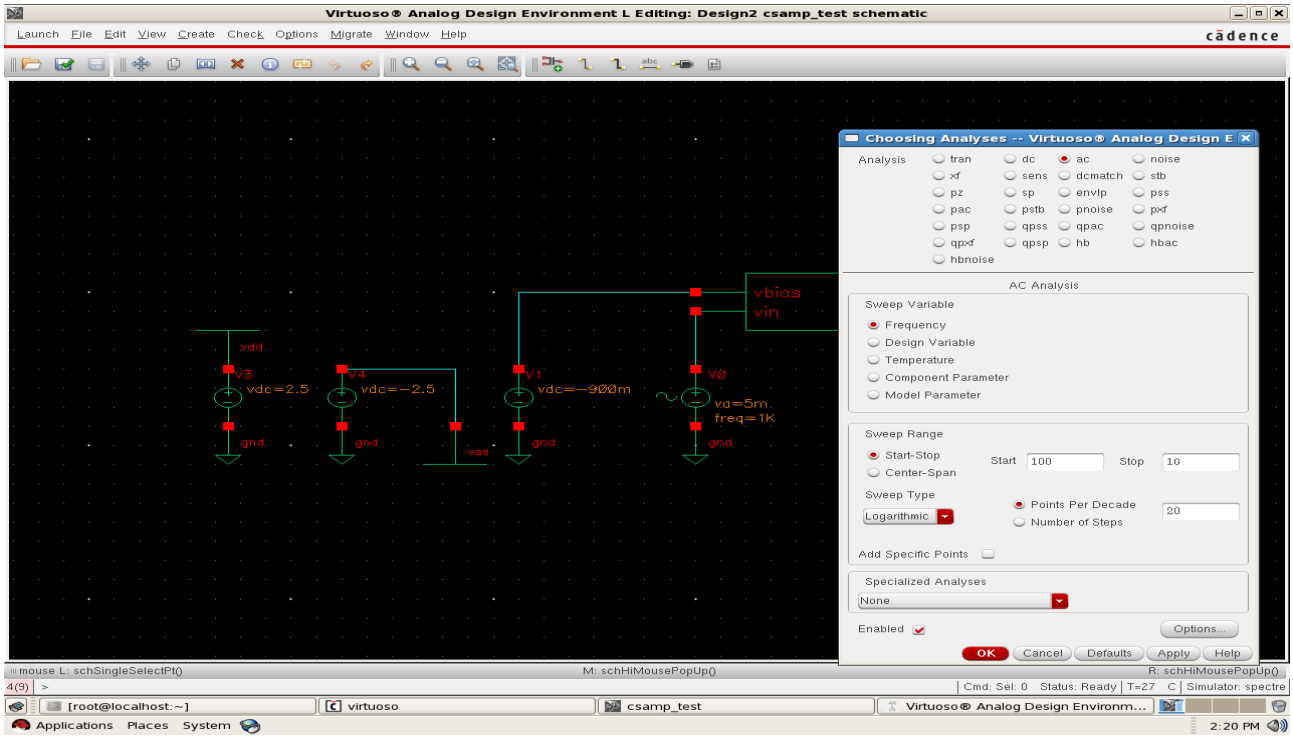
   AC magnitude = **1m**V, Amplitude = **5m**V and Frequency = **1K**Hz. (Don’t enter the units).

   The *AC magnitude* is used for ac analysis, and the *Amplitude* is used for transient analysis. Place a wire in between *vsin* and *vin*.

5. Now, place “vdc” for the the biasing voltage, with *DC voltage* as **-0.9**V. Place two more instances of “vdc”, to be utilized for the attributes of *vdd* and *vss*, and edit their values as **2.5**V and **-2.5**V respectively. Connect the respective *vdd* and *vss* symbols, along with *gnd*. Finally, place a wire and pin for the output.



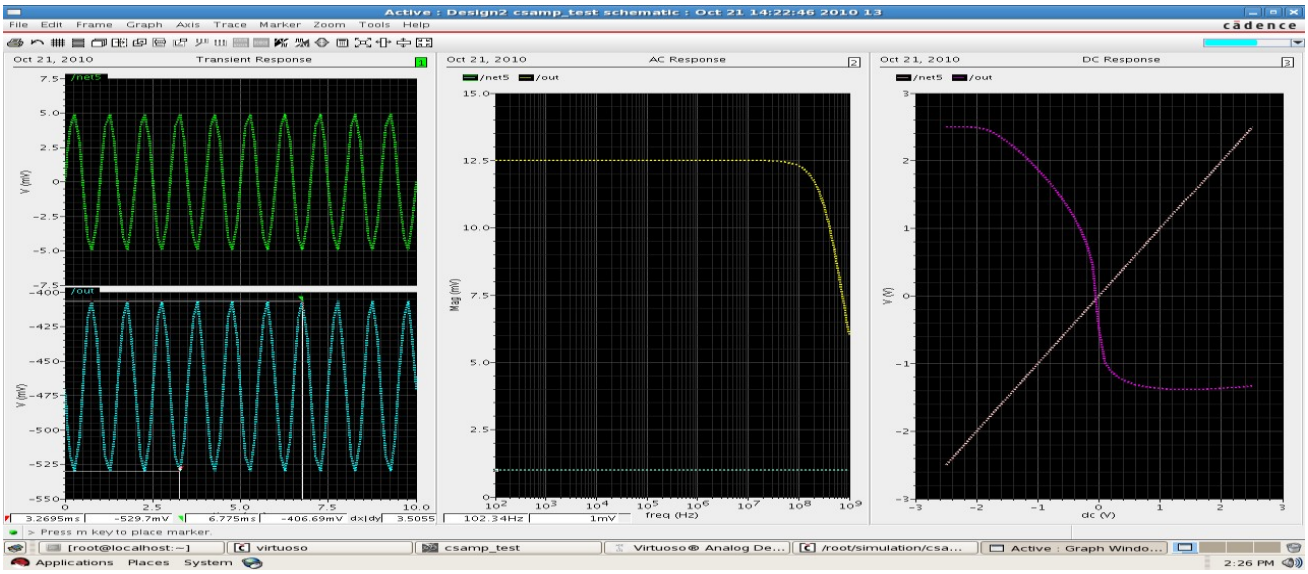
6. After saving, launch ADE-L and enter the analyses requirements, as per the procedures given in the previous experiment. For transient analysis, as the input signal is of period 1ms, enter the Stop Time as **10m**. For dc analysis, Select Component "vsin", select *dc*, and enter the *Sweep Range* as *Start* **-2.5** and *Stop* **2.5**. (Units will appear automatically). As this particular experiment is on amplifiers, ac analysis is also required for finding the bandwidth. Hence, after *tran* and *dc*, click on *ac*, and enter the *Sweep Range* of *Frequency* as **100** Hz till **1GHz**. Select the *Sweep Type* as *Logarithmic* and enter the *Points per Decade* as **20**.



7. Finally, select the wires on the schematic for plotting the outputs, and click on the PLAY icon in the ADE window, for *netlist & run*. The output will appear, and the waveforms can be edited in different colors, for better viewing.

   In the *Transient Response*, for measuring the amplitude, press "d". Now two delta cursors are made available on the screen, one with red pointer and the other with green pointer. Move one cursor to the positive peak of the waveform and the other cursor to the negative peak. The values (time & amplitude) corresponding to the cursor positions are displayed at the bottom, red one first and green one next. The difference in y-values gives the peak-to-peak amplitude.

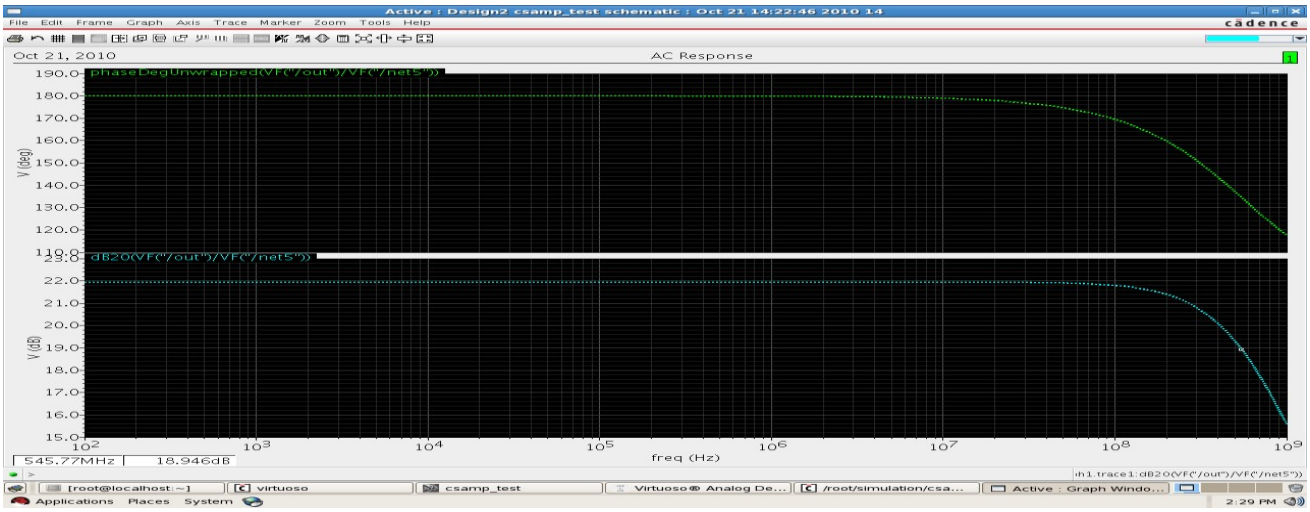
   (The input amplitude in this case is 10 mVpp, as the amplitude given was 5mV). The *dx | dy* is also displayed at the right side, and the dy value represents the amplitude.



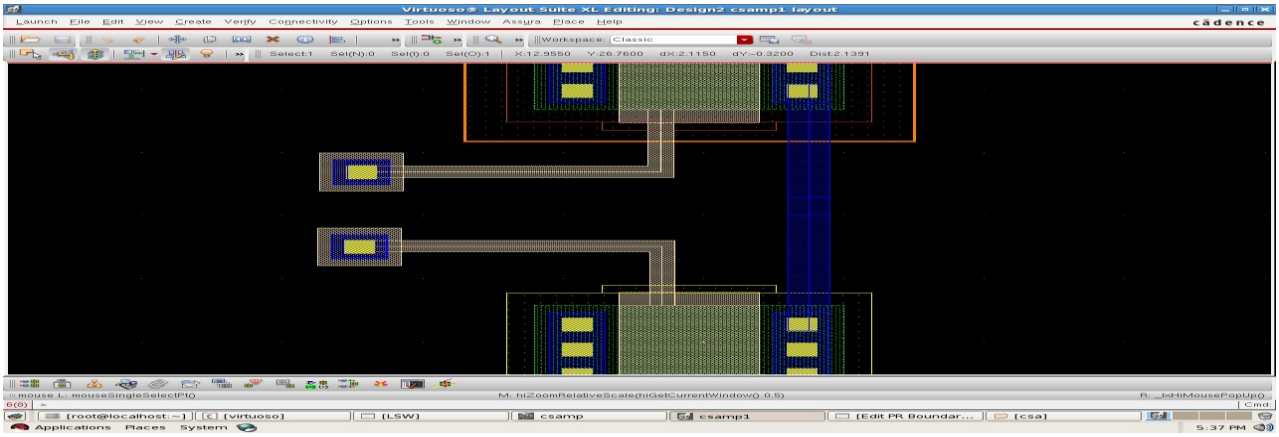
   For measuring the bandwidth, move the trace cursor in the *AC Response* to the point which corresponds to 0.707 V_{max}, and the display indicates the frequency, which is the bandwidth corresponding to the -3dB gain.

   The *DC Response* can be observed for the quiescent operation of the amplifier.

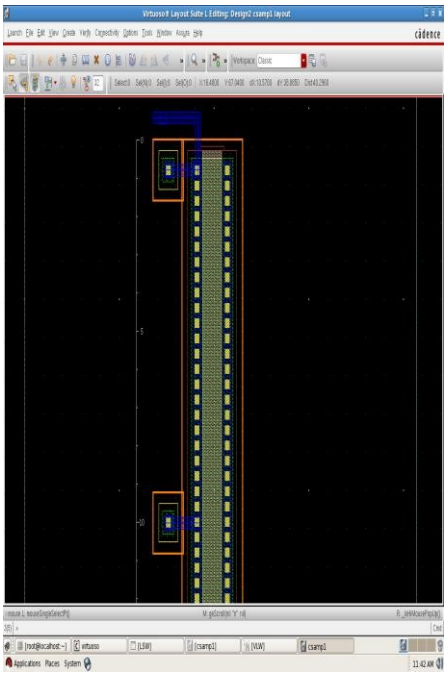
8. The gain and phase response can be obtained separately, by following the procedure in the ADE window as: **Results → Direct plot → AC Gain & Phase**. In the schematic window, click on the output wire first and then on the input wire. Later, press *Esc*. The phase and gain curves are displayed in a separate window. Click on the output and then on *Strip chart mode*, and observe the output. The gain is displayed in dB, and hence, to check the bandwidth of the amplifier, the trace cursor can be moved to the -3dB position on the gain curve.



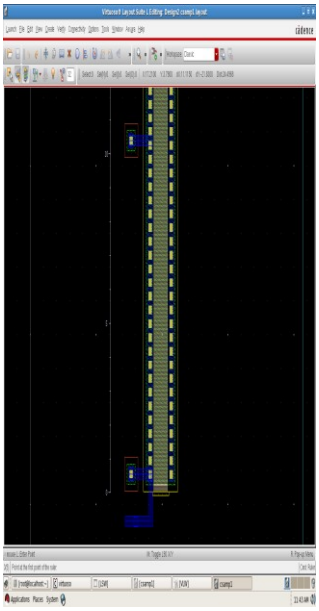
9. After the completion of the simulation, close all the windows except the virtuoso console. In the console, open the schematic file of the amplifier and launch the layout suite. Proceed with the steps as mentioned in the previous experiment, and place the respective paths, pins and the vias. For change of direction of any path, single click at the center, and then change direction.



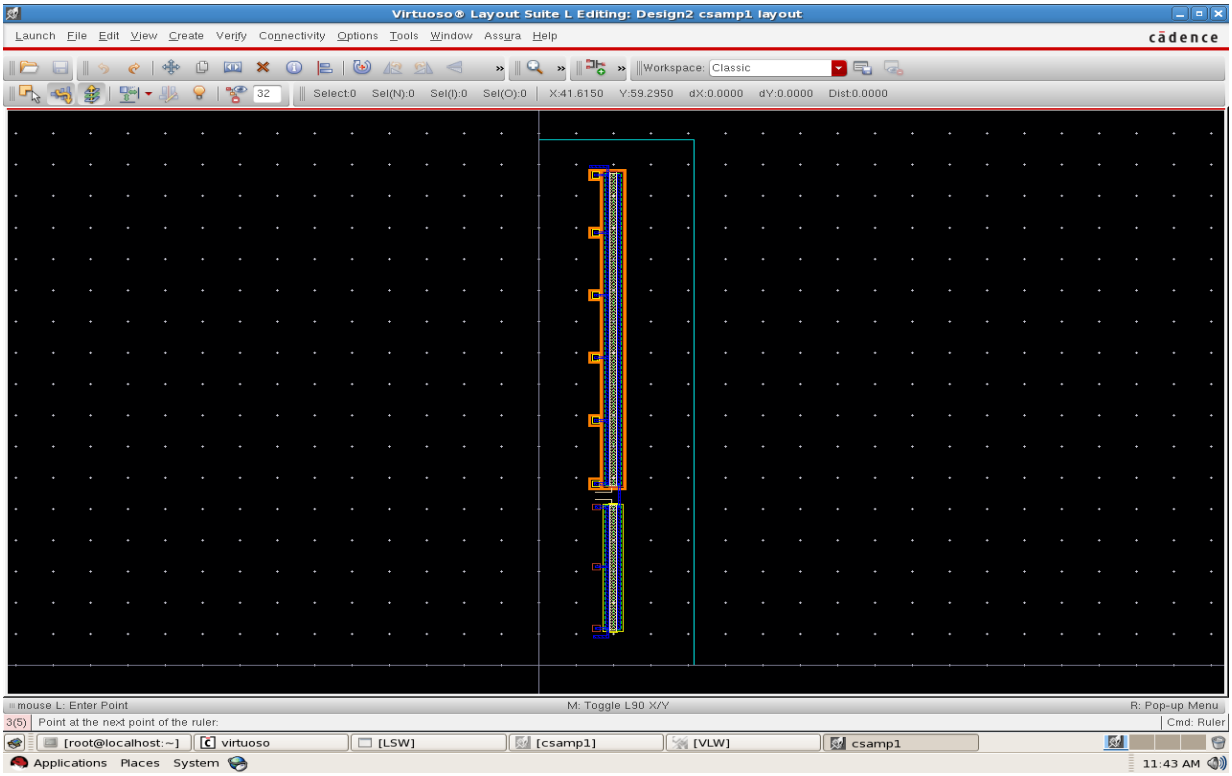
10. As the PMOS device is of 50 microns length, and as substrate connections of the device are required at every 10 microns (as per the design rules), get a ruler on the screen by pressing "K". Click the mouse adjacent to the top of the device, and move the cursor till the lower end of the device. Click again to place the ruler. Now, place the via M1_NWELL at every 10 micron distance. Connect all of these vias to vdd by means of the metal path.



11. Similarly, place the via M1_PSUB near the NMOS device, and connect it to vss. Place one more via at another 10 micron distance.



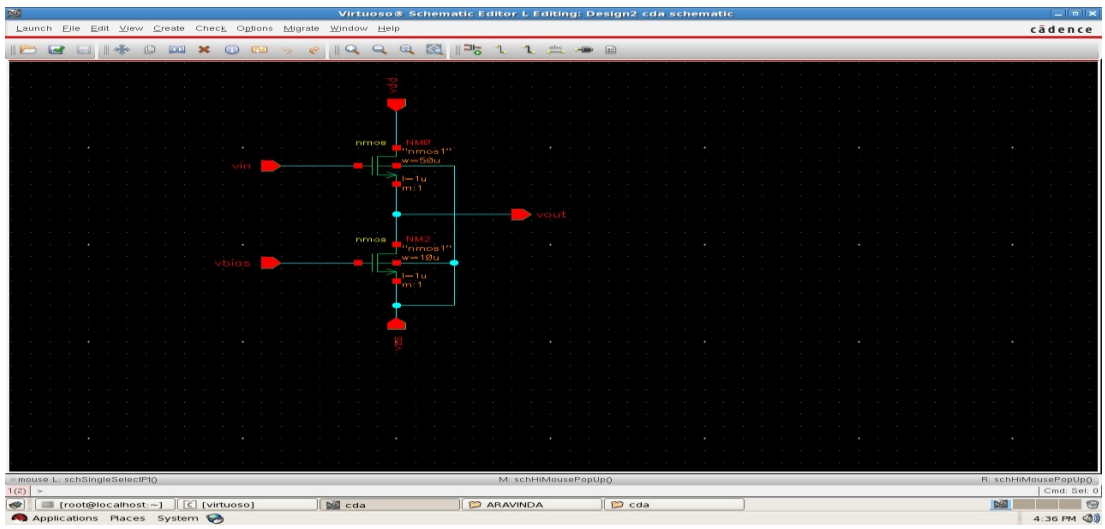
12. Press “Shift K” to delete all the rulers. Save the layout, and run the DRC, LVS and RCX tests.



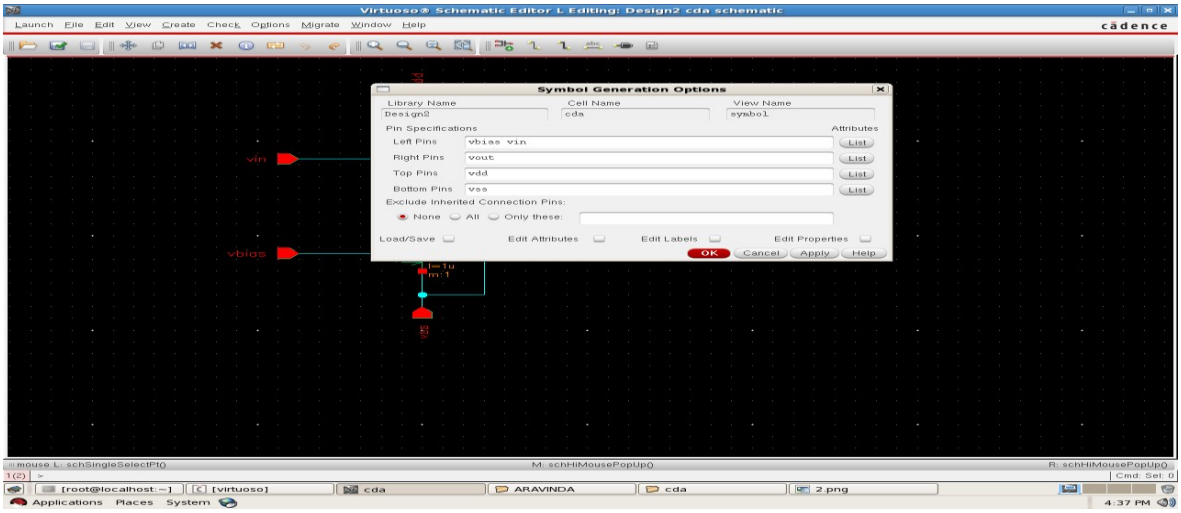
b) COMMON DRAIN AMPLIFIER:

1. The procedures remain similar to the previous experiment. Enter the schematic diagram with the upper NMOS device dimensions as Length = **1** micron & Width = **50** microns, and the lower NMOS device dimensions as Length = **1** micron & Width = **10** microns.

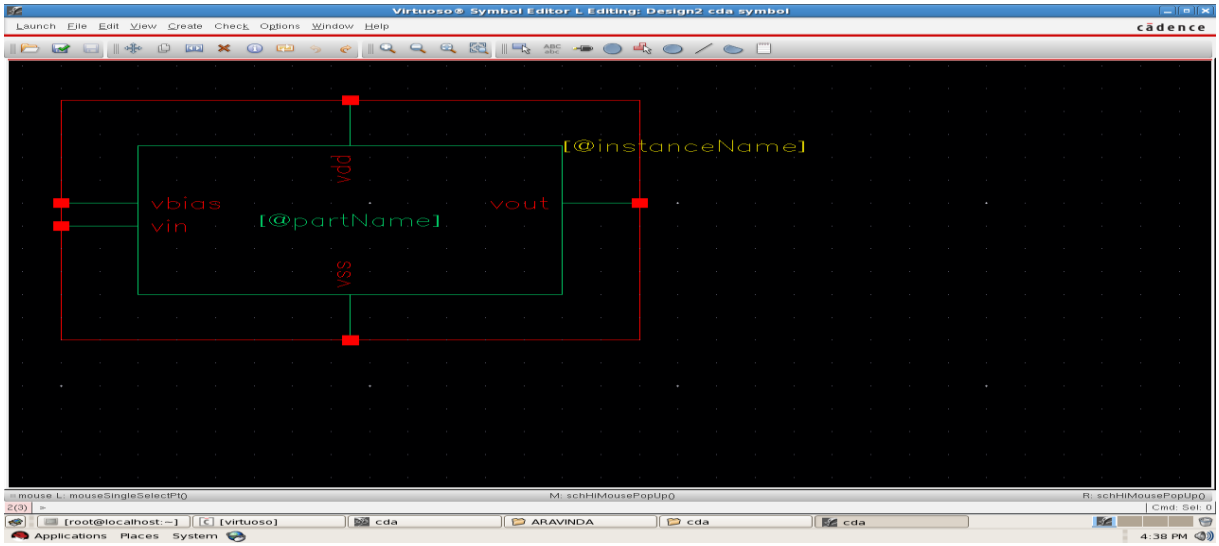
   It is not mandatory that the "vdd" and "vss" symbols from the *analogLi*b library have to be used directly. They can be declared as pins, and their voltages can be directly specified in the test circuit. This alternative method is followed in this experiment, as shown in the circuit diagram below. Complete the other connections as per the circuit diagram, and click on "check & save" and correct the errors, if any.



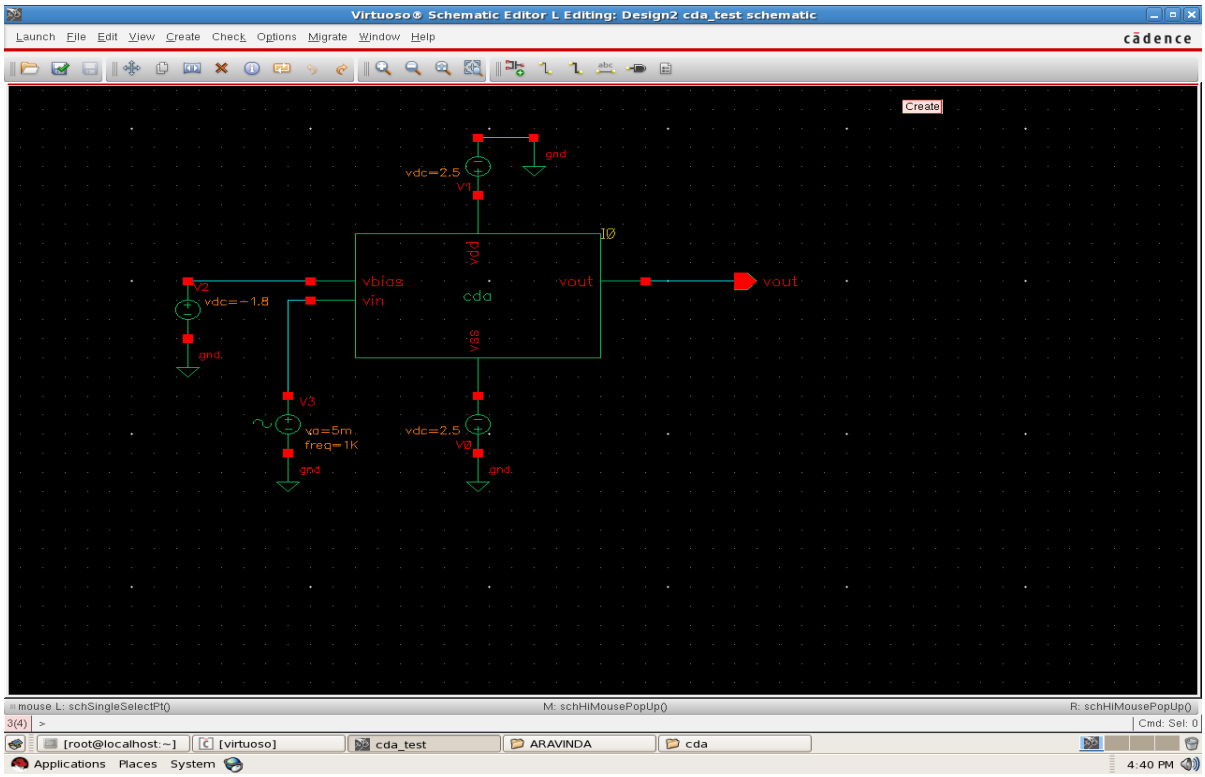
2. During the symbol generation, the *vdd* pin can be placed as the top pin and the *vss* pin can be placed as the bottom pin.



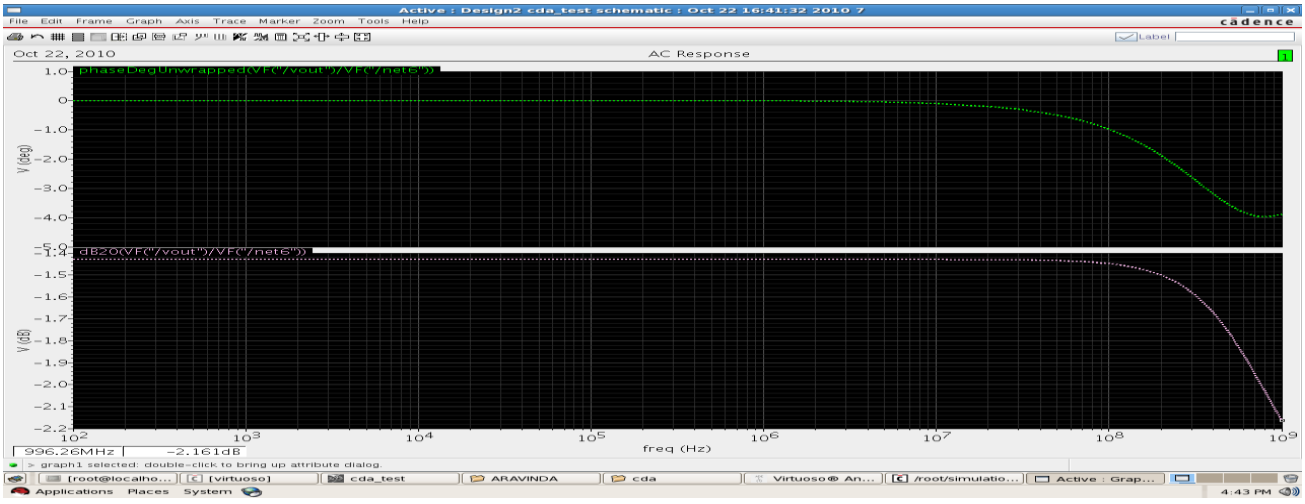
3. The symbol that is generated can be used directly. Click on “check & save”.



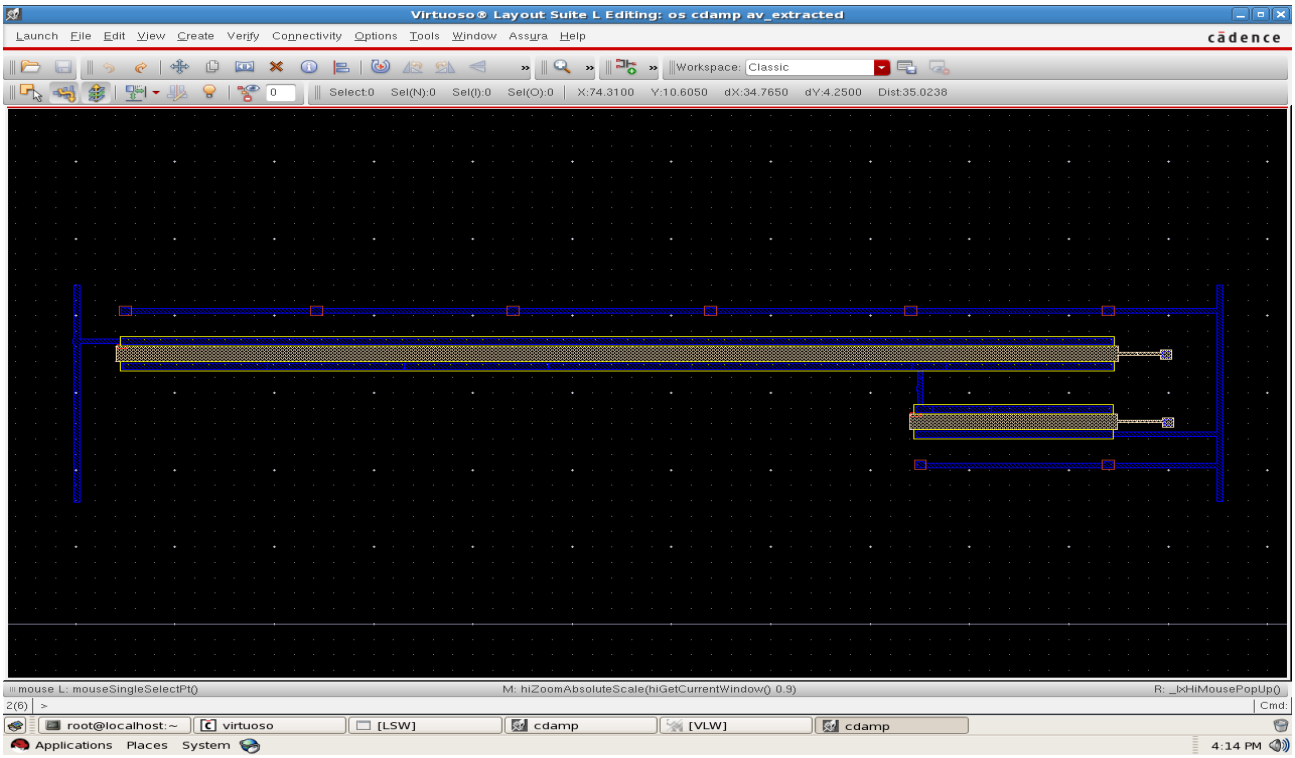
4. The test circuit is similar to the common source amplifier except that “vdc” can be directly connected to the *vdd* and *vss* pins, and their voltages can be edited respectively. The biasing voltage for the common drain amplifier is **-1.8V**.



5. Perform the simulation on the test circuit, and verify the gain and phase. As the circuit is similar to the Emitter follower circuit of BJT, there will be no voltage gain as such.



6. As the next step, complete the layout and perform the physical verification. The assura verification_extracted layout is as shown below –



Result

The common source and common drain amplifier simulation and layout verification has been done for the given specifications.

Experiment No: 5

Draw the schematic of i) Differential Amplifier, ii) Op-amp, for the given specifications, and verify using Transient, DC and AC Analyses.
Draw the layout of i) Differential Amplifier, ii) Op-amp, and perform physical verification using DRC, ERC and LVS. Extract RC and back annotate the same and verify the Design.

i) Differential Amplifier

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

Theory: A differential amplifier is a type of electronic amplifier that amplifies the difference between two input voltages but suppresses any voltage common to the two inputs. It is an analog circuit with two inputs V+ and V- and one output vout , in which the output is ideally proportional to the difference between the two voltages.

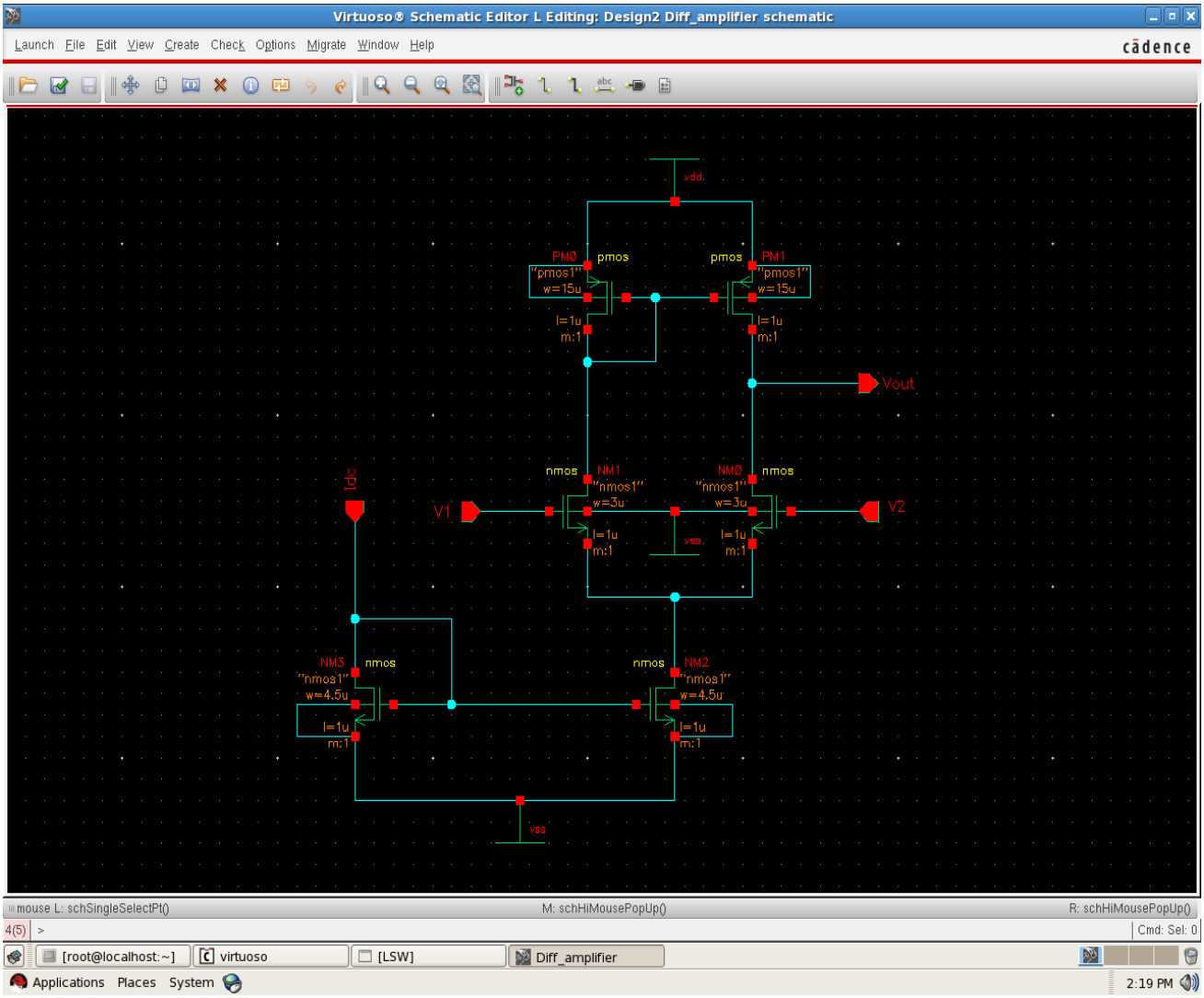
$Vout = A (V^+ - V^-)$

where A is the gain of the amplifier.

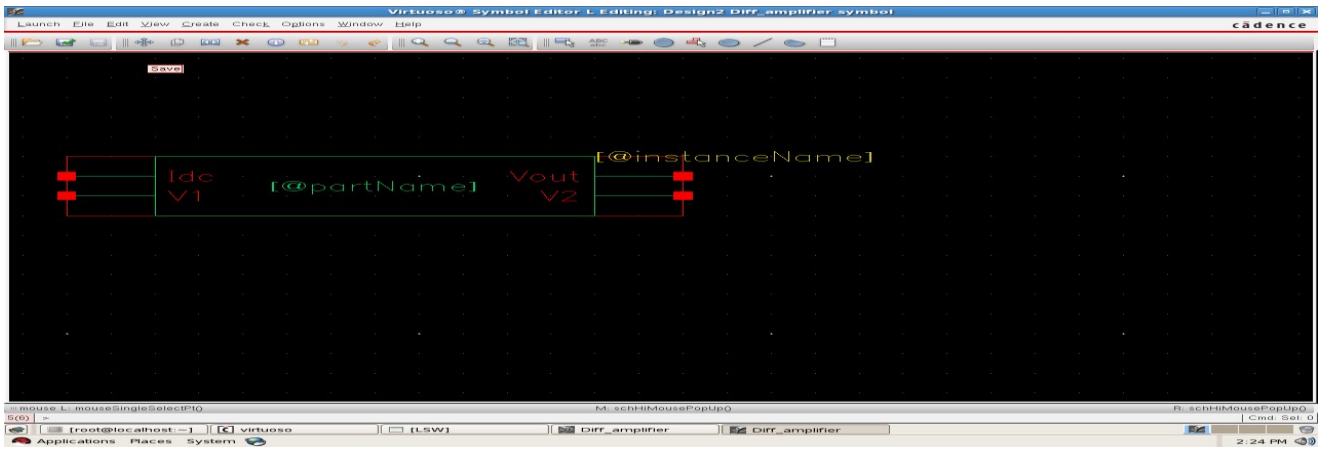
Procedure:

1. Perform the schematic entry as per the following specs: All of the transistors will have the length = **1** micron; the PMOS load transistors with width = **15** microns; the NMOS input transistors with width = **3** microns; and finally, the NMOS biasing transistors with width =

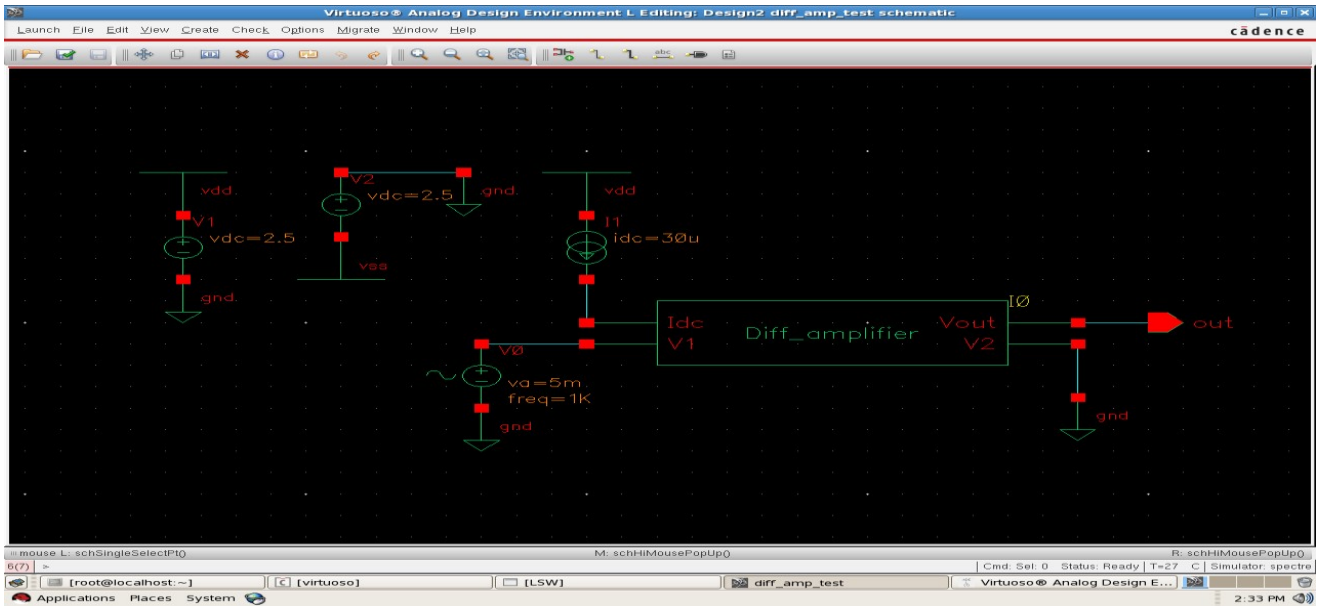
4.5 microns. While placing the PMOS transistors, the "Sideways" option can be used, for having their position as shown in the schematic diagram.



2. Create a symbol for the schematic diagram, and place the input V2 as the right pin.



3. Complete the test circuit by placing the current source, the input signal and the power supplies. Initially, V1 can be used as the input, and V2 can be grounded at that time. The current source is available as “idc” in the analogLib library; set its DC current as **30 µA**. Connect its positive end to *vdd* and negative end to the *Idc* input.

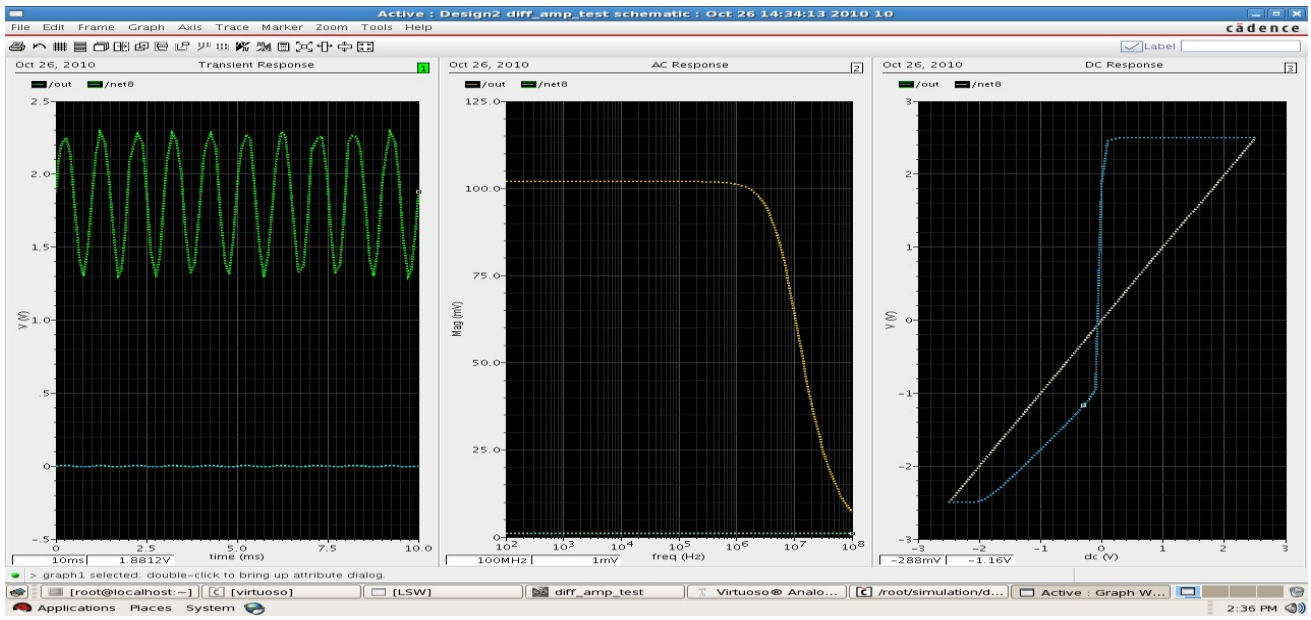


4. For simulation, the *transient* and *dc* analyses details remain the same as that of the previous experiment. In the *ac* analysis, the frequency range from 100 Hz till 100 MHz.

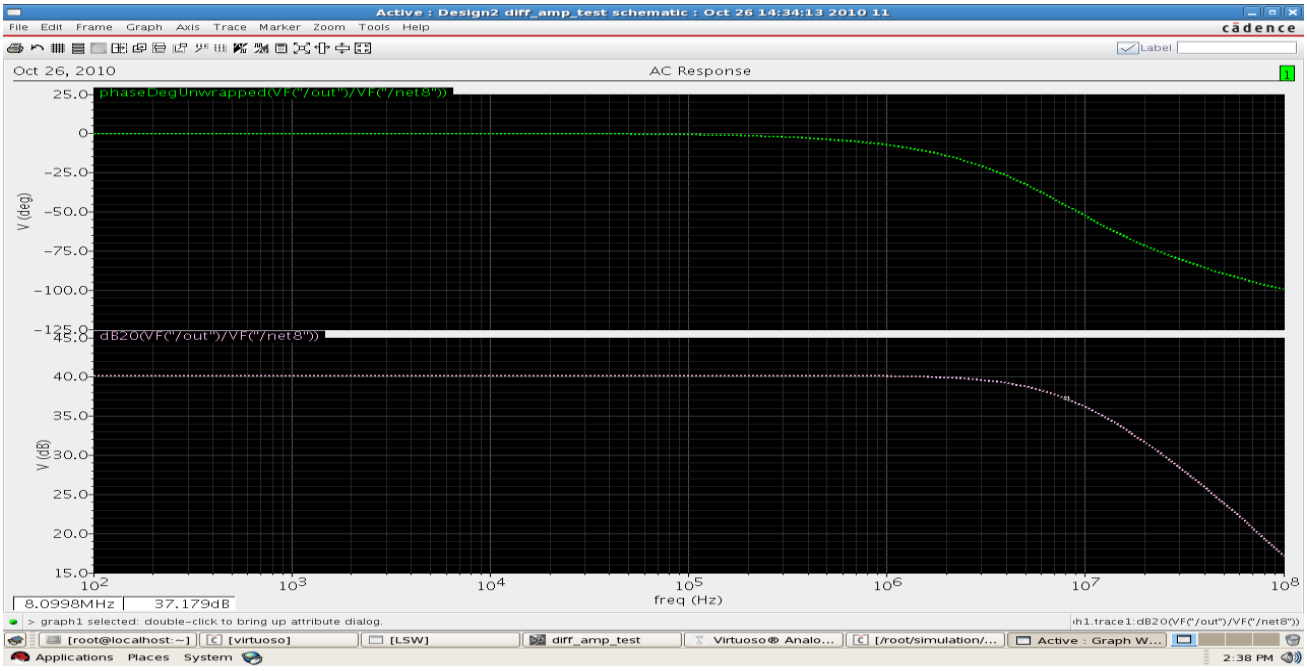
As V1 is the non-inverting input, the output will be in phase with the input, and the circuit will provide around 40 dB gain. This can be verified initially, and then the test circuit can be altered with grounded V1, and input provided to V2. The inverting operation of the circuit can be verified. The gain and phase responses can also be verified.

The following screenshots indicate both of the functional operations

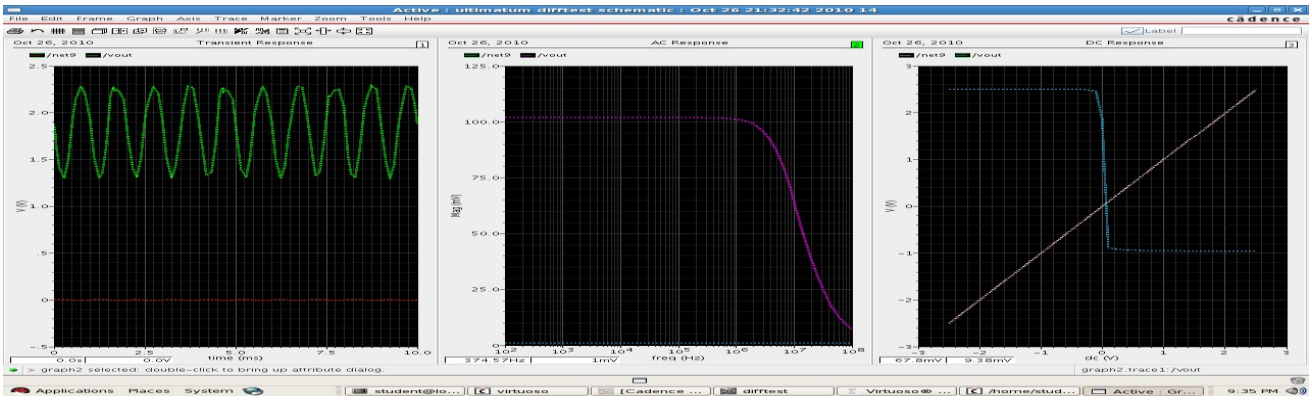
<u>Non-inverting amplifier</u>: (V1 as input)



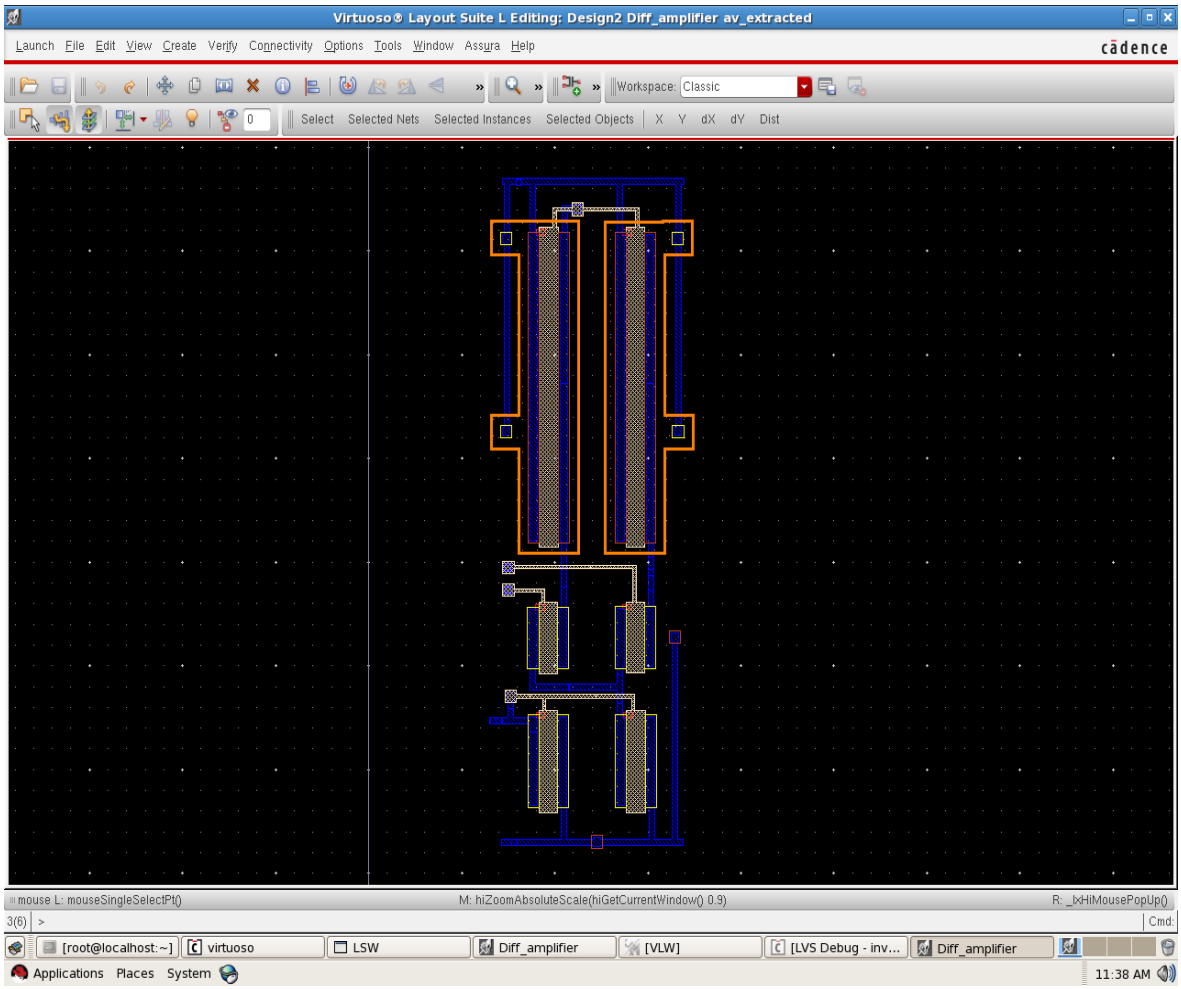
<u>Its phase and gain response</u>:



Inverting amplifier: (V2 as input)



5. After the simulation, complete the layout and perform the physical verification. The assura verification_extracted layout is shown below –



b) OPERATIONAL AMPLIFIER

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

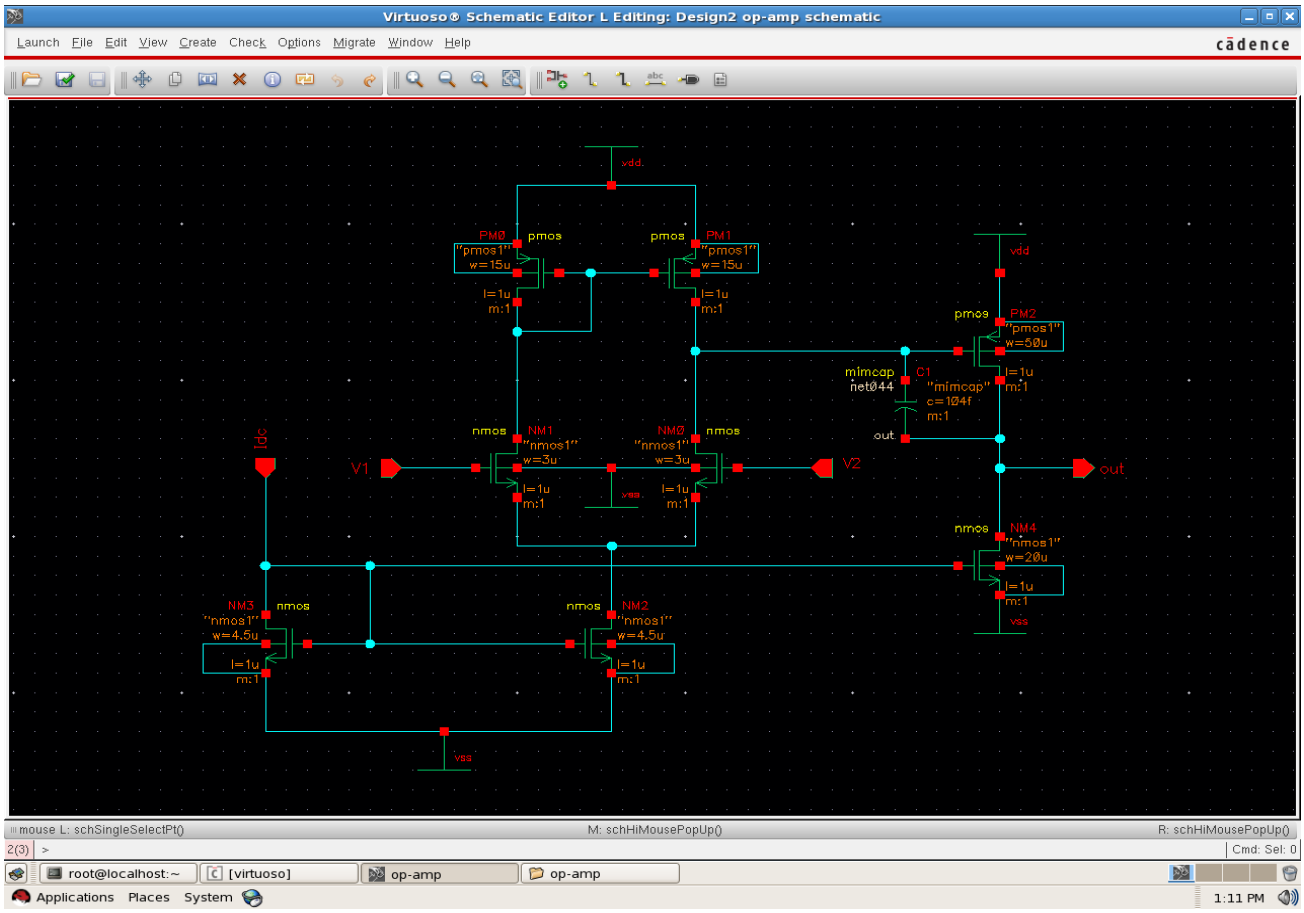
Theory: An operational amplifier (often op-amp or opamp) is a DC-coupled high-gain electronic voltage amplifier with a differential input and, usually, a single-ended output. In this configuration, an op-amp produces an output potential (relative to circuit ground) that is typically hundreds of thousands of times larger than the potential difference between its input terminals. Operational amplifiers had their origins in analog computers, where they were used to perform mathematical operations in many linear, non-linear, and frequency-dependent circuits. The popularity of the op-amp as a building block in analog circuits is due to its versatility. By using negative feedback, the characteristics of an op-amp circuit, its gain, input and output impedance, bandwidth etc

Procedure:

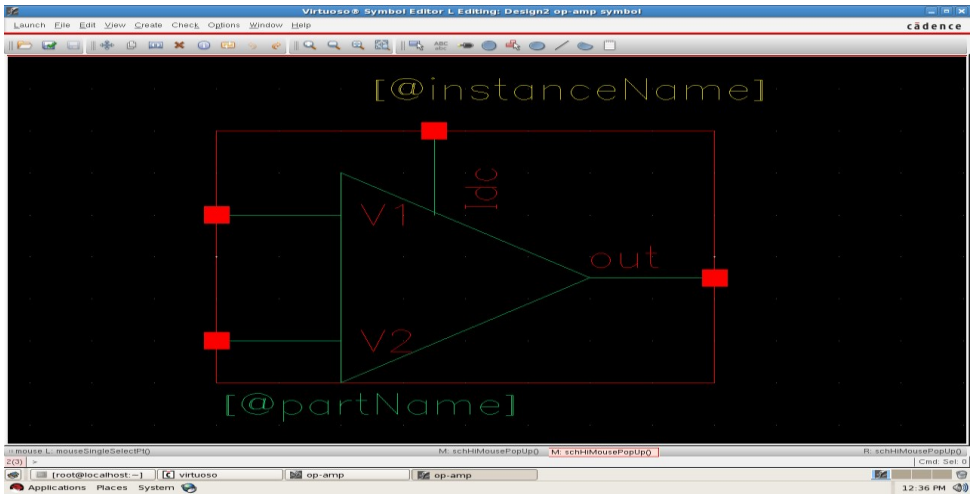
1. To obtain the operational amplifier's circuitry, in this particular design, a common source amplifier with PMOS driver is cascaded with the differential amplifier. Due to the addition of this high-gain stage, V1 now becomes the inverting input and V2 becomes the non-inverting input. Select PMOS device width as **50** microns and NMOS device width as **20** microns, the

length of both the transistors remaining at **1** micron. Connect the gate of the NMOS device to Idc, and complete the substrate and power supply connections. Connect an output pin.

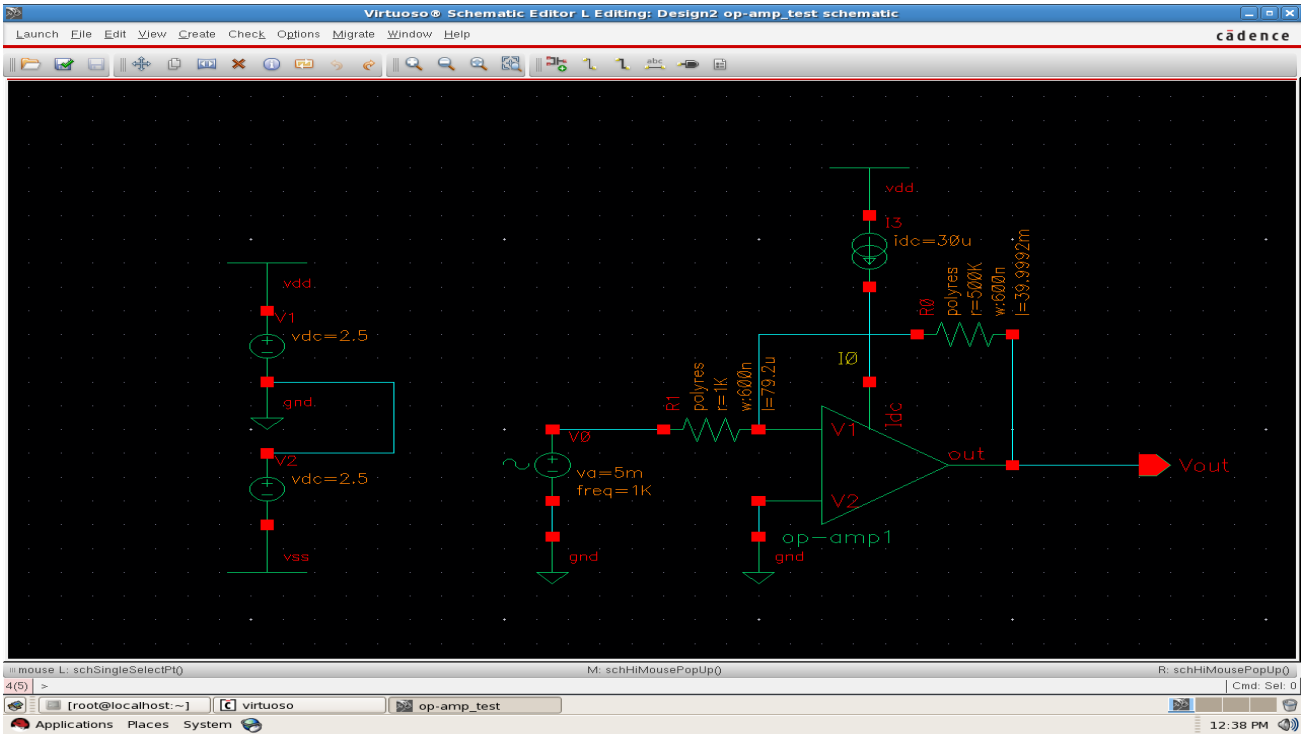
In addition, to ensure the safer phase margin, a compensating capacitor is added between the gate and drain of the PMOS driver. Select the component “mimcap” from *gpdk180* library, with its width and length set at **10** microns, and its value will become 104 fF. (*mim* stands for metal-insulator-metal; this capacitor is formed in between the two metal layers).



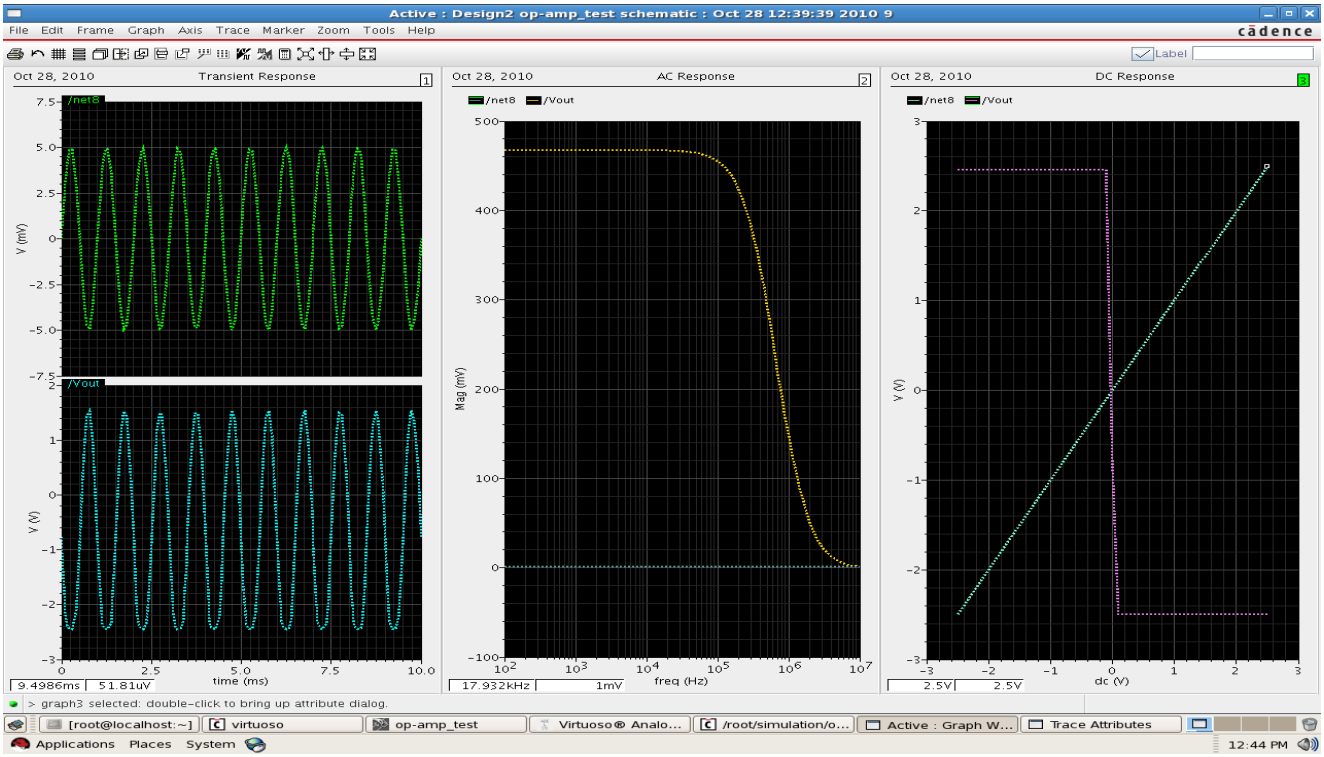
2. Create a symbol for the op-amp, as shown. Take *Idc* pin at the top, and widen the gap between the input pins.



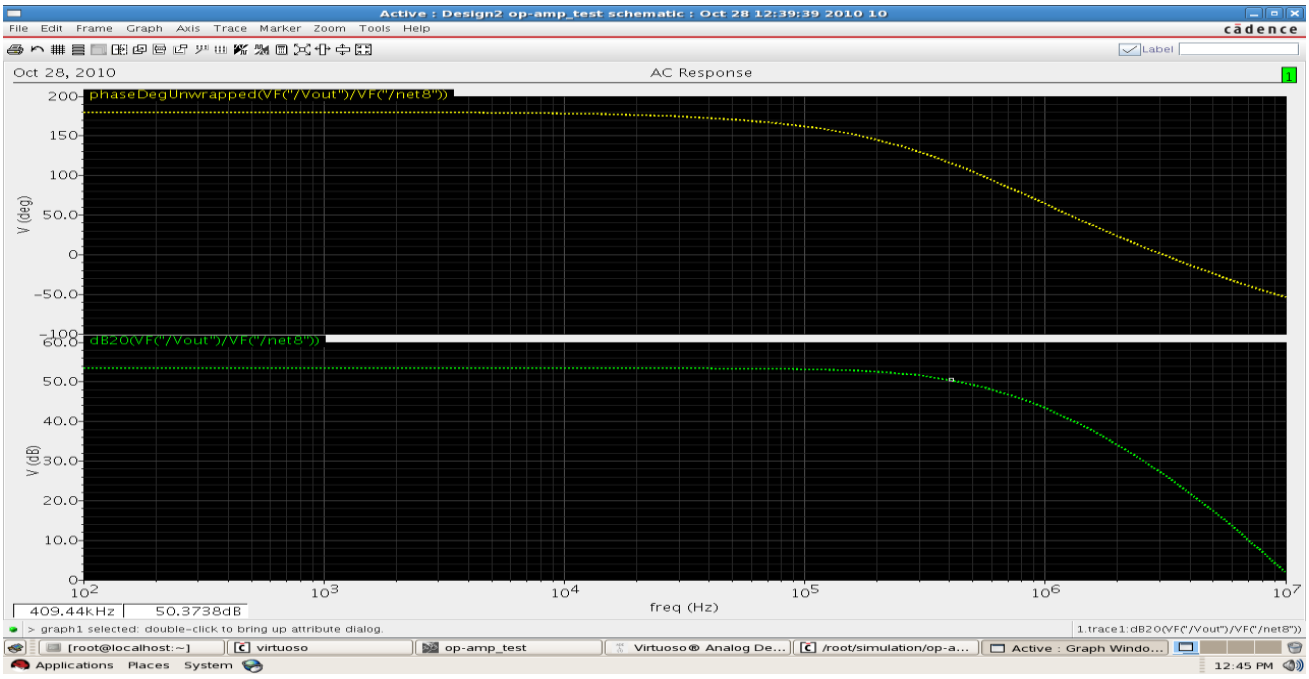
3. Close both the editor windows, and then create the test circuit, initially for the inverting amplifier, as shown. For the resistors, use “polyres” from *gpdk180* library, with the values 1k ohms and 500k ohms. Connect the 30 µA current source for *Idc*.



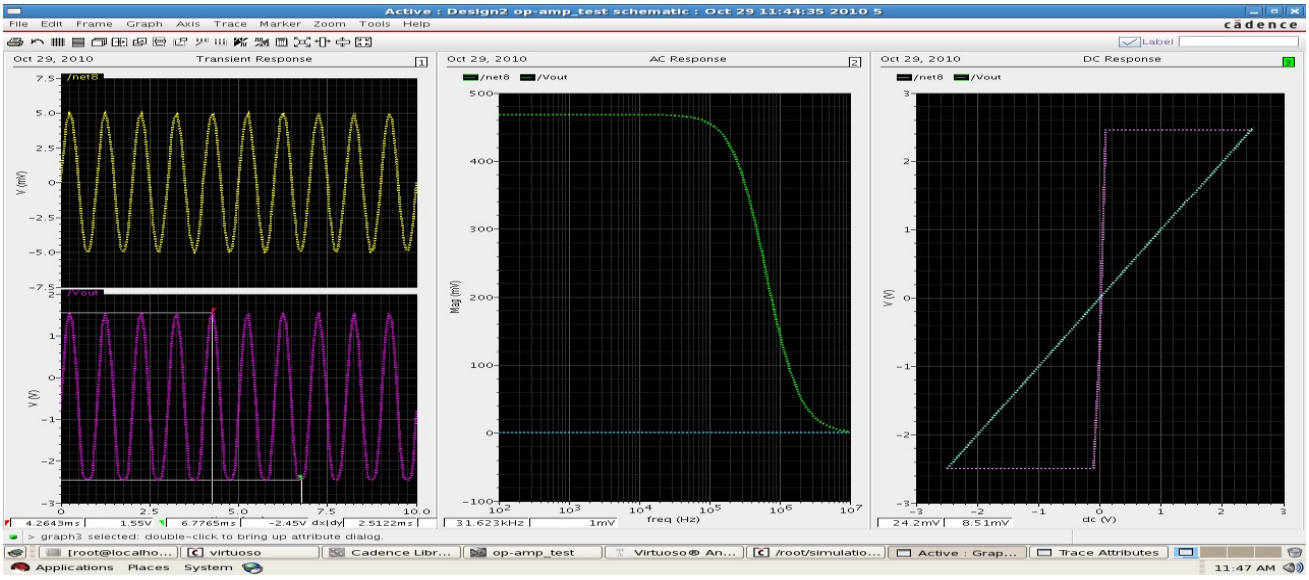
4. Run the simulation with the same details as that of the previous experiment, except that – choose the frequency range of 100 Hz to 10 MHz for the *ac* analysis. Verify the output.



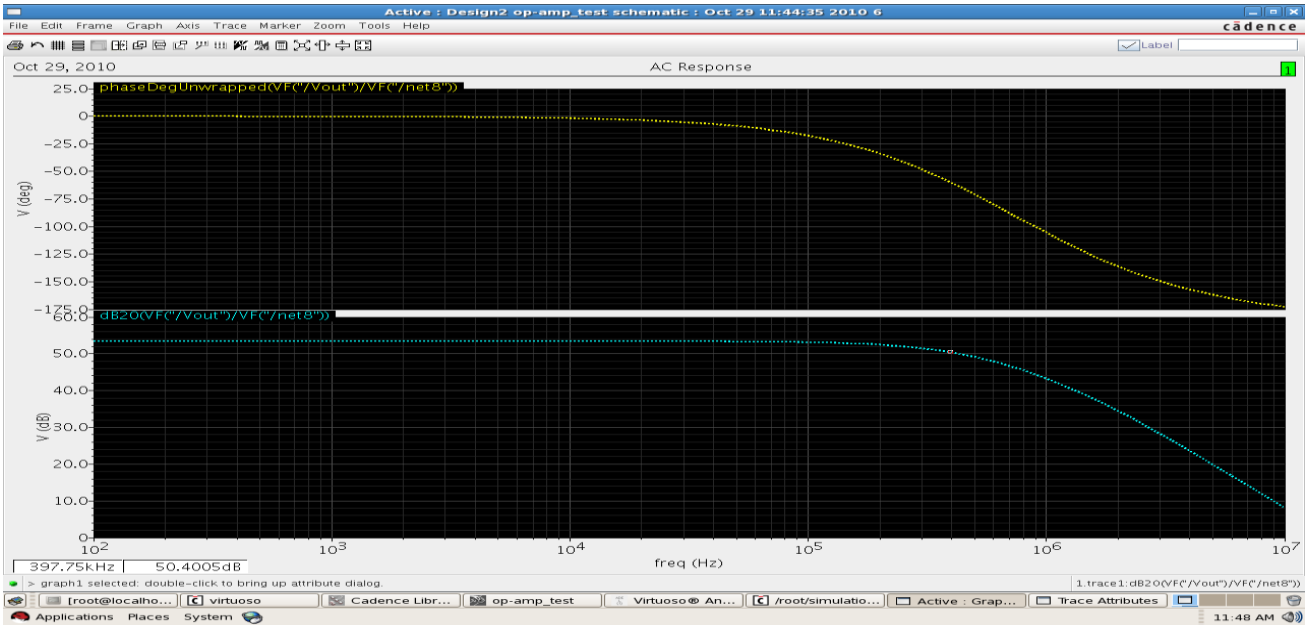
5. Verify the phase and gain responses as well, and find out the bandwidth of the op-amp. Vary the value of the feedback resistor, for obtaining different gains, and verify the bandwidth.



6. Later, change the input to V2, and check the performance in the non-inverting mode.



7. Verify its phase and gain responses as well. Check the bandwidth for different gains, by varying the value of the feedback resistor.



8. After the simulation, complete the layout and perform the physical verification.

Result

The differential amplifier and op-amp simulation and layout verification has been done for the given specifications.

CYCLE 2 EXPERIMENTS

DIGITAL DESIGN (SIMULATION & SYNTHESIZE)

INITIAL PROCEDURES:

1. After logging in, *right click* and *open terminal*.
2. Get into the *c shell* by typing the command – **csh**
3. Run the shell script by typing the command – **source home/install/cshrc**

I. STEPS FOR DESIGN ENTRY:

1. Create your own design directory (e.g.: Design1) using the *mkdir* command –

   mkdir Design1

2. Write the verilog program for your design (e.g.: Codefile1.v) inside *Design1* directory.
3. Write the verilog test bench program for your design (e.g.: Codefile1_tb.v) inside *Design1* directory. Now, the design entry using HDL gets finished.

1. STEPS FOR SIMULATION:

1. Initially, both of your verilog programs have to be compiled using the command –

   ncvlog *filename* -mess

   For this example: **ncvlog Codefile1.v -mess**

   ncvlog Codefile1_tb.v -mess

2. After compilation, you have to elaborate the top module using the command –

   ncelab *modulename* -access +rwc -mess

   The process of elaboration builds up the instances and then connects them. Care must be taken that you elaborate only the top module – namely the test bench; the attribute for the command should be the test bench module name.

3. Next step is to simulate the design using the command –

   ncsim *modulename* (or) **ncsim *modulename* -gui**

   Here also, the simulation has to be performed using the test bench module name. When *gui* is used, the Simvision tool will be invoked and the waveforms can be seen.

2. STEPS FOR SYNTHESIS:

1. This is required because the synthesis is performed only on the design file, and not on the test bench. Now, edit the program file and delete all of the compiler directives which are present in the program file (the commands preceded by `).
2. As the next step, if switch primitives are present in the program, then they have to be replaced by the RTL description, because of the fact that, the switch primitives are not synthesizable. The output of synthesis is a schematic using logic gates. This is called as "logic synthesis", which is technology independent.

3. Create a file named *Contraints_top.sdc*. The timing constraints are defined in this file. Example of one such file is as shown –

```
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 1.0 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "A"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "B"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "sum"] -clock [get_clocks "clk"]
```

There are three different parts in the constraint file:

a. Clock definition and clock constraints –

*create_clock -name **clk** -period 10 -waveform {0 5} [get_ports "clock"]* → Clock definition
*set_clock_transition -rise 0.1 [get_clocks **"clk"**]* → Clock rise time
*set_clock_transition -fall 0.1 [get_clocks **"clk"**]* → Clock fall time
*set_clock_uncertainty 1.0 [get_ports **"clk"**]* → Uncertainties of Clock

b. Input port timing constraints –

*set_input_delay -max 1.0 [get_ports **"A"**] -clock [get_clocks **"clk"**]* → Input port delay
*set_input_delay -max 1.0 [get_ports **"B"**] -clock [get_clocks **"clk"**]* → Input port delay

c. Output port timing constraints –
*set_output_delay -max 1.0 [get_ports **"sum"**] -clock [get_clocks **"clk"**]* → Output port delay

The port names that are used in the constraint file (**bolded**) must match with the names that are used in the Verilog program of the main design module. The constraints are defined for all the ports in the design.

4. Now, invoke the *GENUS* tool using the command –

genus -f –gui

The tool will be invoked and you will get the *rc* prompt in the terminal, along with a synthesis window. Next, the following commands are to be typed in the *rc* prompt

1. **set_db lib_search_path ./home/install.FOUNDRY/digital/90nm/dig.lib7**
2. **set_db hdl_search_path ./**
3. **set_db library slow.lib**
4. **read_hdl inverter.v**
5. **elaborate**
6. **read_sdc./constraints_top.sdc**
7. **synthesize -to_mapped -effort medium**
8. **report timing**
9. **report gates**
10. **report area**
11. **report power**
12. **write_hdl > inverter_netlist.v**

Script file is explained below as shown in Figure 19 :

1. Give the path of the library w.r.t to the directory you are using the command : **set_db lib_search_path**
2. Give the path of the RTL files w.r.t to the directory you are using the command : **set_db 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) is the name of the library file in the directory.
4. Read the RTL files from the directory specified in the second line. The RTL files are in the directory name : **read_hdl inverter.v**
5. Now Elaborate the design using the command : **elaborate**

   *If you are having constraint file then you can include the constraint file like this
6. Give the standard delay constraints using: **read_sdc./constraints_top.sdc**
7. Synthesize the circuit using the commands :**synthesize -to_mapped -effort medium**
8. Timing could be checking using: **report timing**.
9. Similarly for Gates: **report gates**.
10. Check area using: **report area**.
11. Check Power dissipation using: **report power**.
12. It will generate the reports
13. Write the hdl code in terms of library components for the synthesized circuit using the command: **write_hdl > inverter_netlist.v**

The tool will execute each command as and when it is entered. To come out of the synthesis environment, *exit* or *quit* command in the *genus* prompt is used.

The commands of step-10 can be saved in a rc_script.*tcl* file in the *work* directory, and that script file can be invoked by

genus –f rc_script.tcl -gui

NOTES

1. Linux commands:

Ls	list files	cp	copy files
mv	move files	rm	remove files
cd	change directory	pwd	print working directory
mkdir	make directory	rmdir	remove directory

2. Command options:

~	for root directory
.	for present directory
..	for parent directory
-mess	for displaying messages
&	for making the process a background one

3. Abbreviations:

GDS	Generic Data Stream
IUS	Incisive Unified Simulator
NC	Native compiler
RC	RTL compiler
RTL	Register Transfer Level
TCL	Tool Command Language

4. Cadence tools used:

IUS and ***Simvision*** - to compile, elaborate and simulate

genus and ***Encounter*** - for RTL Compilation & Synthesis

Experiment No: 6

For the following circuits, write the switch level Verilog Code, and verify using Test Bench:

i) CMOS inverter, ii) 2-input CMOS NAND and NOR gates

iii) 2-input EXOR gate using CMOS logic, iv) 2-input EXOR gate using PTL

AIM: To write the switch level Verilog Code for inverter, NAND, NOR and XOR gate and verify using Test Bench.

Theory: transistor level modeling is referred to model in hardware structures using transistor models with analog input and output signal values. At this level, a hardware component is described at the transistor level, but transistors only exhibit digital behavior and their input, and output signal values are only limited to digital values. At the switch level, transistors behave as on-off switches- Verilog uses a 4 value logic value system, so Verilog switch input and output signals can take any of the four 0, 1, Z, and X logic values.

In Verilog nmos and pmos switches are instantiated as shown in below,

nmos n1(out, data, control); // instantiate a nmos switch

pmos p1(out, data, control); // instantiate a pmos switch

Design Files: Main design module, Test bench module and Constraints file.

i) Main Design Module: Inverter.v

```
`timescale 1 ns / 1 ns
//Define our own Inverter
module inverter (out, in);
// Declarations of I/O, Power and Ground Lines
        output out;
        input in;
        supply1 pwr;
        supply0 gnd;
        // Instantiate pmos and nmos switches
        pmos (out, pwr, in);
        nmos (out, gnd, in);
endmodule
```

II Test Bench Module: Inverter_test.v

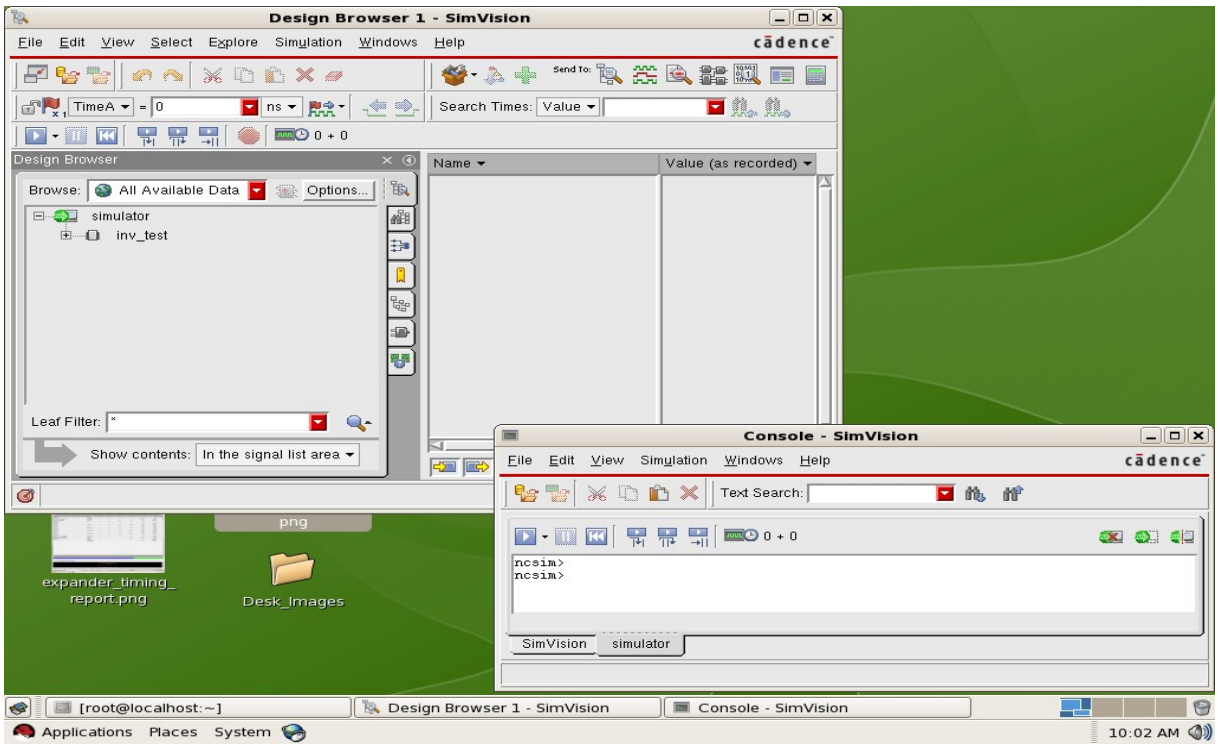
```
`timescale 1 ns / 1 ns
// Testbench for Inverter Module

module inv_test;
        wire out;
        reg in;
```

```
        // Instantiate inverter Module
        inverter i1 (out, in);
        // Apply Stimulus
        initial
                begin
                        in = 1'b0; #10;
                        in = 1'b1; #10;
                        in = 1'bx; #10;
                        in = 1'bz; #10;
                end
endmodule
```

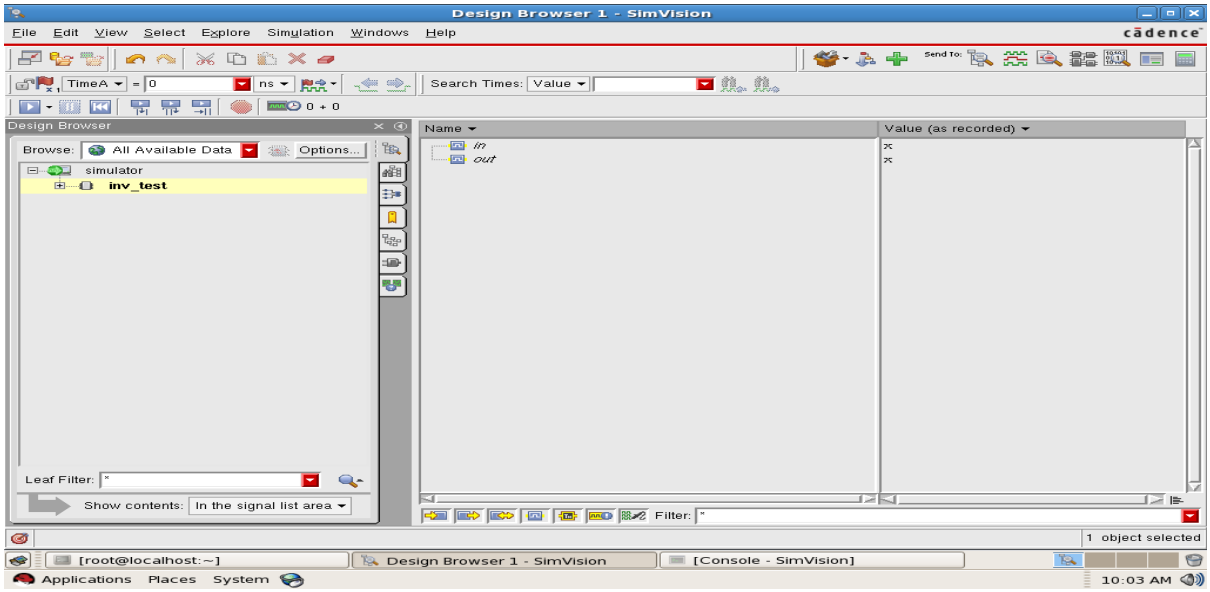
Procedure:

Initially, follow all the steps mentioned in the previous section. The rest of the steps are as follows –

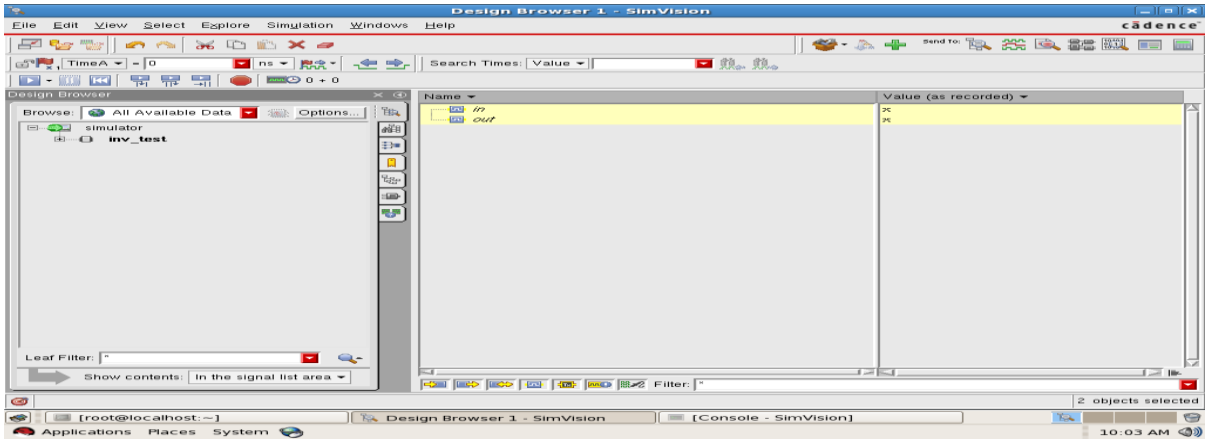
1. After invoking the **Simvision** tool with the command *ncsim modulename –gui*, two windows will be opened: one is the *Simvision Console* and the other is the *Design Browser window* of the tool, as shown in the snap shot –



2. As the next step, you need to click on the *module name* on the left side of the Design Browser window. As soon as you click, you will see all the port names displayed in the right side of the Design Browser window.

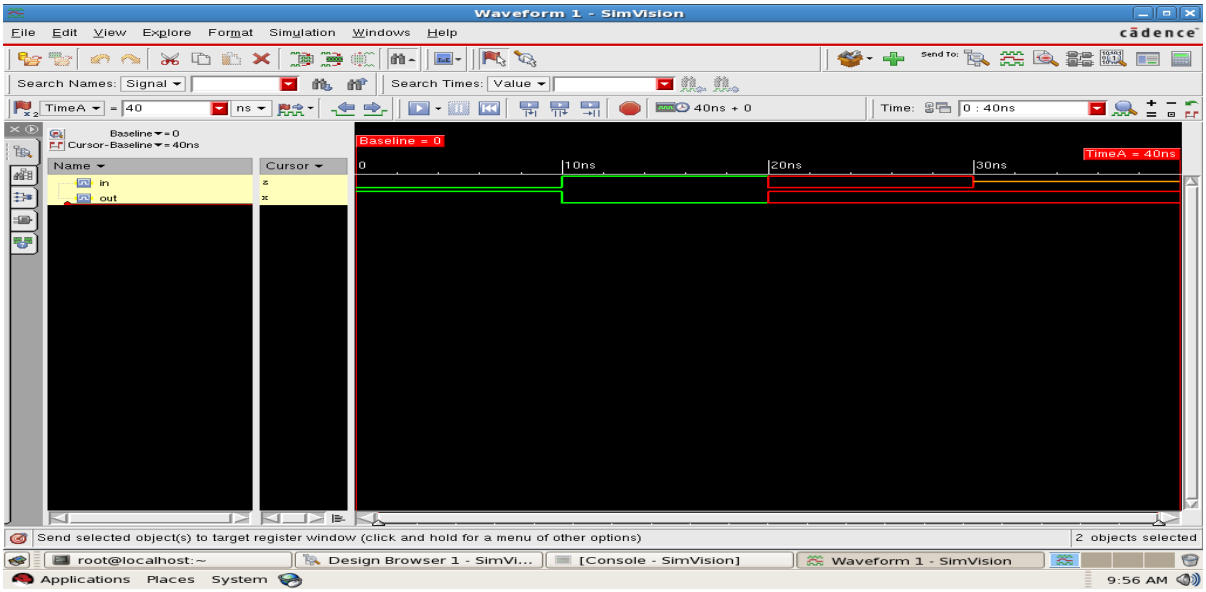


3. Then select all the ports for which you want to see the waveforms.

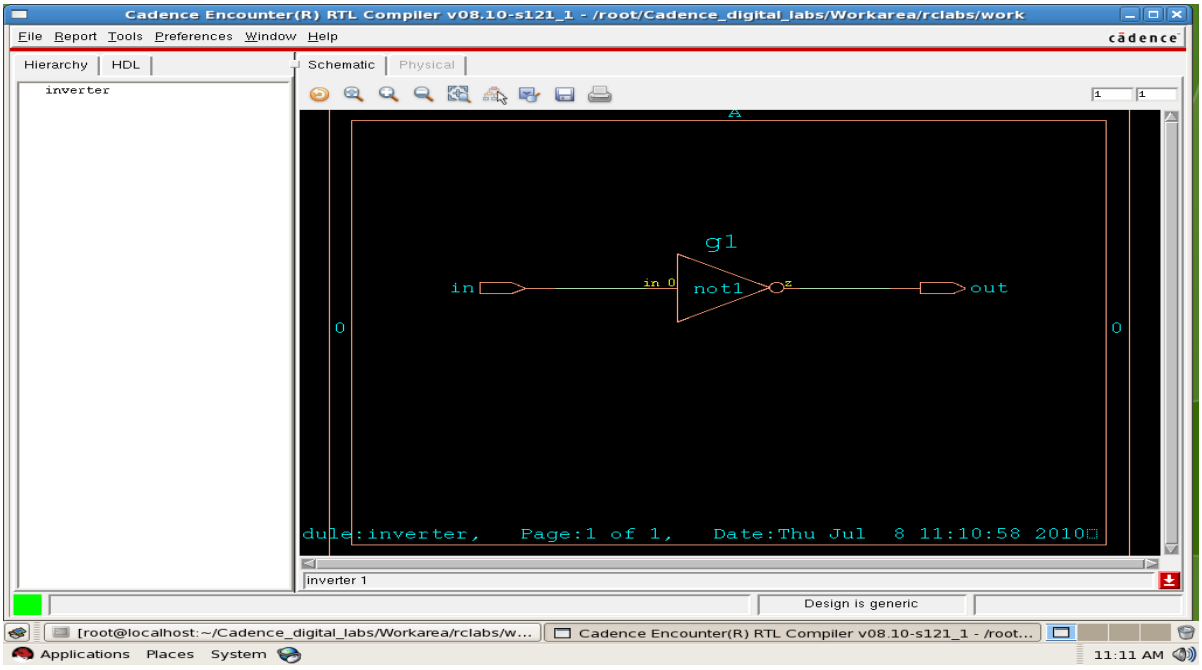


4. Next, click the *waveform icon* on the right hand side corner of the Design Browser window. Now the waveform window will open.

5. Click the *play* button on the waveform window to run the simulation. The waveforms will be generated as shown –



6. For synthesis, follow the steps that are mentioned previously. After synthesis, the synthesis window will be shown as follows –



ii) NAND GATE:

Main Design Module: Nandg.v

```
`timescale 1 ns / 1 ns
//Define our own Nand Gate
 module nandgate (out, in1, in2);
        // Declarations of I/O, Power and Ground Lines
```

```
        output out;
        input in1, in2;
        supply1 pwr;
        supply0 gnd;
        wire contact;

        // Instantiate pmos and nmos switches

        pmos (out, pwr, in1);
        pmos (out, pwr, in2);
        nmos (out, contact, in1);
        nmos (contact, gnd, in2);

endmodule
```

II <u>Test Bench Module</u>: Nand_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Nand Gate Module

module nand_test;

        wire out;
        reg in1, in2;
        `uselib view = vlog

        // Instantiate Nand Gate Module

        nandgate n1 (out, in1, in2);
        `nouselib

        // Apply Stimulus

        initial
                begin
                        in1 = 1'b0; in2 = 1'b0; #10;
                        in1 = 1'b0; in2 = 1'b1; #10;
                        in1 = 1'b1; in2 = 1'b0; #10;
                        in1 = 1'b1; in2 = 1'b1; #10;
                end
endmodule
```

iii) NOR GATE

Main Design Module: Norg.v

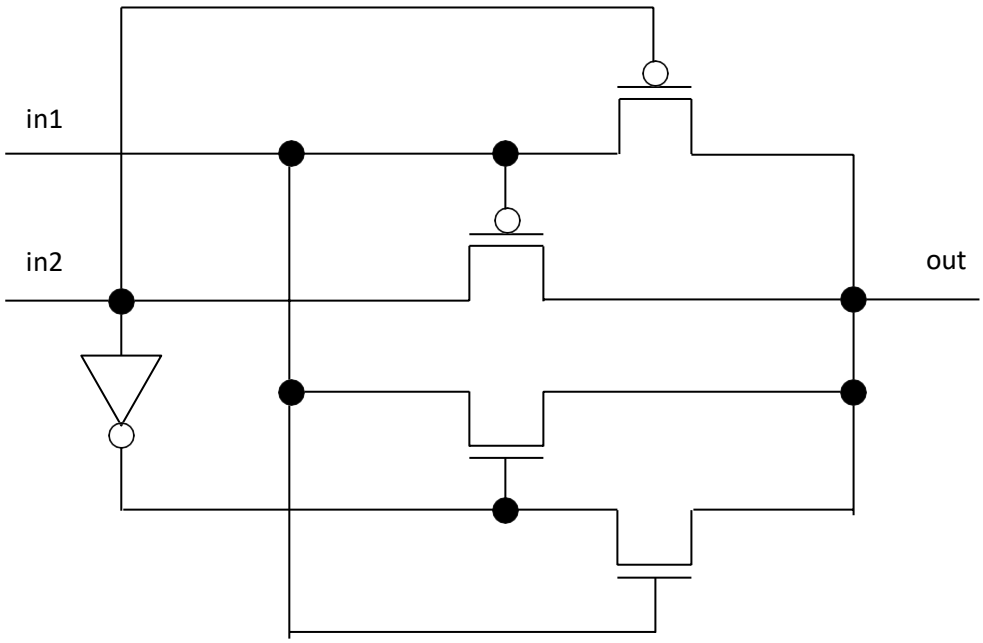
```
`timescale 1 ns / 1 ns
//Define our own Nor Gate
 module norgate (out, in1, in2);
        // Declarations of I/O, Power and Ground Lines
        output out;
        input in1, in2;
        supply1 pwr;
        supply0 gnd;

        // Declaration of Wire

        wire contact;

        // Instantiate pmos and nmos switches

        pmos (contact, pwr, in1);
        pmos (out, contact, in2);
        nmos (out, gnd, in1);
        nmos (out, gnd, in2);

 endmodule
```

II Test Bench Module: Nor_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Nor Gate Module
module nor_test;
        wire out;
        reg in1, in2;

        // Instantiate Nor Gate Module
        norgate n1 (out, in1, in2);
        // Apply Stimulus
        initial
                begin
                        in1 = 1'b0; in2 = 1'b0; #10;
                        in1 = 1'b0; in2 = 1'b1; #10;
                        in1 = 1'b1; in2 = 1'b0; #10;
                        in1 = 1'b1; in2 = 1'b1; #10;
                end
endmodule
```

iv) EXOR GATE:

Schematic diagram (using pass transistor logic)



Main Design Module: Exorg.v

```
`timescale 1 ns / 1 ns
//Define our own XOR Gate
module xorgate (out, in1, in2);
        output out;
        input in1, in2;
        wire in2bar;
        assign in2bar = ~ in2;
        pmos (out, in2, in1);
        nmos (out, in1, in2bar);
        pmos (out, in1, in2);
        nmos (out, in2bar, in1);

endmodule
```

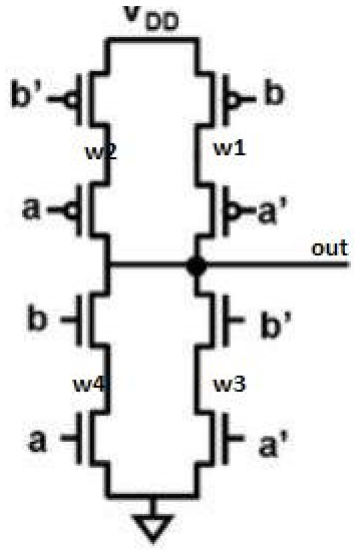
Test Bench Module: Exor_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Xor Module
module xor_test;
        wire out;
        reg in1, in2;
        // Instantiate Xorgate Module
```

```
        xorgate x1 (out, in1, in2);

        // Apply Stimulus

        initial
                begin
                        in1 = 1'b0; in2 = 1'b0; #10;
                        in1 = 1'b0; in2 = 1'b1; #10;
                        in1 = 1'b1; in2 = 1'b0; #10;
                        in1 = 1'b1; in2 = 1'b1; #10;
                end
endmodule
```

v) EXOR GATE

Schematic diagram (using CMOS logic)



Main Design Module: Exorg.v

```
`timescale 1 ns / 1 ns
module xorgate (out, a,b);
        output out;
        input a, b;
        supply1 pwr;
        supply0 gnd;

        // Declaration of Wire

        wire w1,w2,w3,w4;

pmos p1(w2, pwr, ~b);
```

```
pmos p2(out,w2,,a);
pmos p3(w2, pwr, ~b);
pmos  p4(out,w2,,a);

        nmos n1(out, w4, b);
        nmos n2 (out, w3, ~b);
        nmos n3 (w4, gnd,a);
        nmos n4 (w3,gnd,~a);


endmodule
```

Test Bench Module: Exor_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Xor Module
module xor_test;
        wire out;
        reg  a,b;
        xorgate x1 (out,a,b);
        initial
                begin
                        a = 1'b0; b = 1'b0; #10;
                        a = 1'b0; b = 1'b1; #10;
                        a = 1'b1; b = 1'b0; #10;
                        a = 1'b1; b = 1'b1; #10;
                end
endmodule
```

Result

The switch level Verilog Code was written and verified using Test Bench for CMOS inverter, i2-input CMOS NAND and NOR gates, 2-input EXOR gate using CMOS logic, 2-input EXOR gate using PTL.

Experiment No: 7

For the following circuits, write the Verilog Code, verify using Test Bench, and then Synthesize the following circuits using the gate level Verilog Code, with the given Constraints:

i) CMOS inverter, ii) 2-input CMOS NAND and NOR gates

Aim: To compile and to simulate the Verilog code for an inverter, and then to synthesize the same for the given constraints.

Theory: In Gate level model, the module is implemented in terms of logic gates and interconnections between these gates. Designer should know the gate-level diagram of the design. In general, gate-level modeling is used for implementing lowest level modules in a design like, full-adder, multiplexers, etc. Verilog HDL has gate primitives for all basic gates. Gate primitives are predefined in Verilog, which are ready to use. They are instantiated like modules. There are two classes of gate primitives: Multiple input gate primitives and Single input gate primitives. Multiple input gate primitives include and, nand, or, nor, xor, and xnor. These can have multiple inputs and a single output. They are instantiated as follows:

```
// Two input AND gate.

and and_1 (out, in0, in1);

// Three input NAND gate.

nand nand_1 (out, in0, in1, in2);
```

Design Files: Main design module, Test bench module and Constraints file.

I Main Design Module: Inverter.v

```
`timescale 1 ns / 1 ns

//Define our own Inverter

module inverter (out, in);

// Declarations of I/O, Power and Ground Lines

        output out;

input in;

        not n1(out,in);

endmodule
```

II Test Bench Module: Inverter_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Inverter Module
module inv_test;
wire out;
reg in;// Instantiate inverter Module
inverter i1 (out, in);
// Apply Stimulus
initial
        begin
                in = 1'b0; #10;
                in = 1'b1; #10;
                in = 1'bx; #10;
                in = 1'bz; #10;
        end
endmodule
```

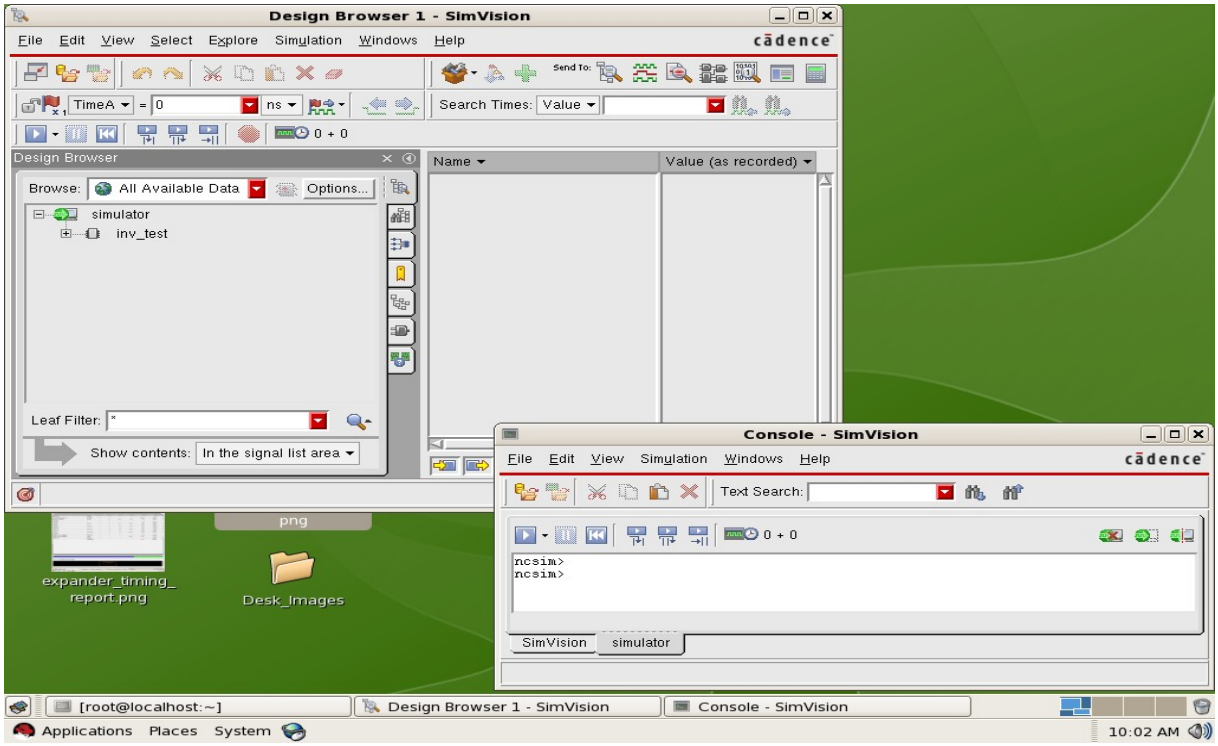
III <u>Constraints file for Synthesis</u>: Constraints_Inverter.g

```
set_input_delay -max 1.0 [get_ports "in"]
set_output_delay -max 1.0 [get_ports "out"]
```

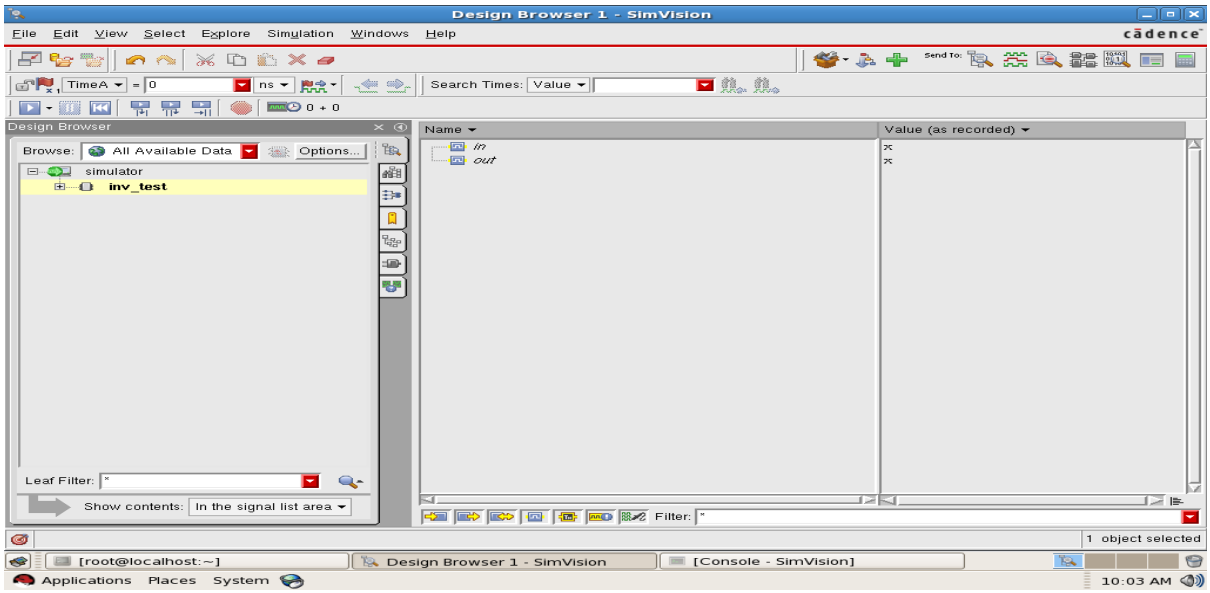
<u>Procedure</u>:

Initially, follow all the steps mentioned in the previous section. The rest of the steps are as follows –

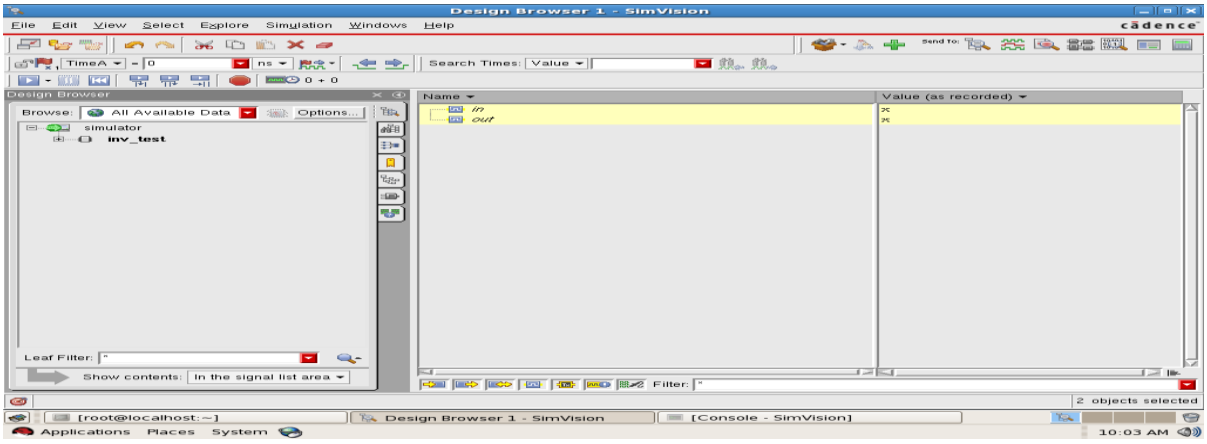
1. After invoking the **Simvision** tool with the command *ncsim modulename –gui*, two windows will be opened: one is the *Simvision Console* and the other is the *Design Browser window* of the tool, as shown in the snap shot –



2. As the next step, you need to click on the *module name* on the left side of the Design Browser window. As soon as you click, you will see all the port names displayed in the right side of the Design Browser window.

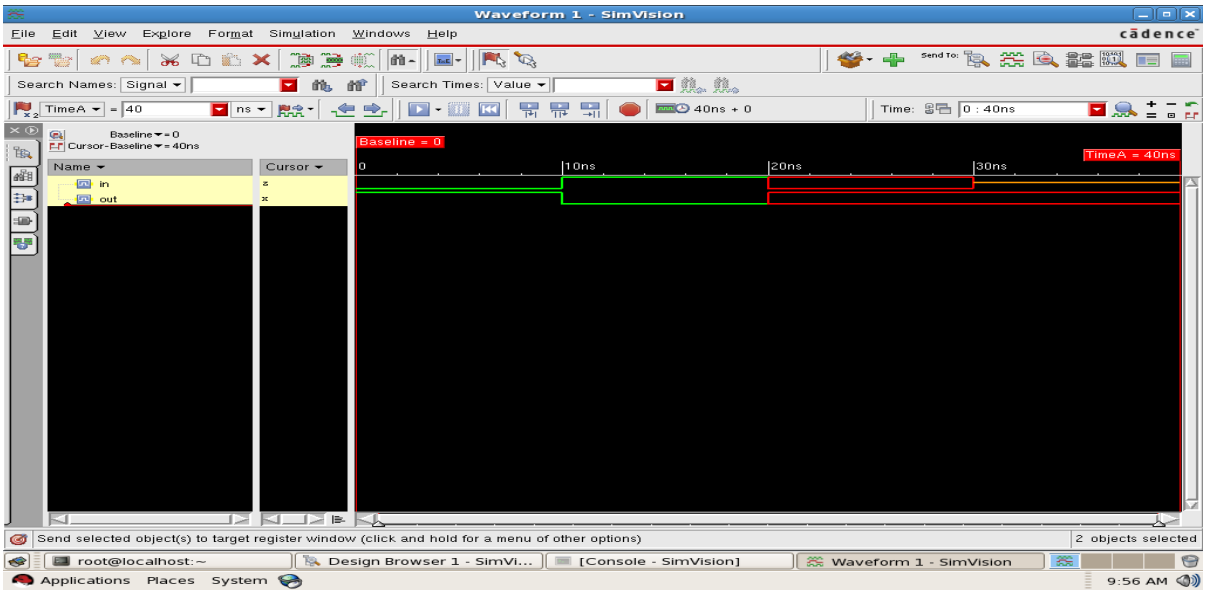


3. Then select all the ports for which you want to see the waveforms.

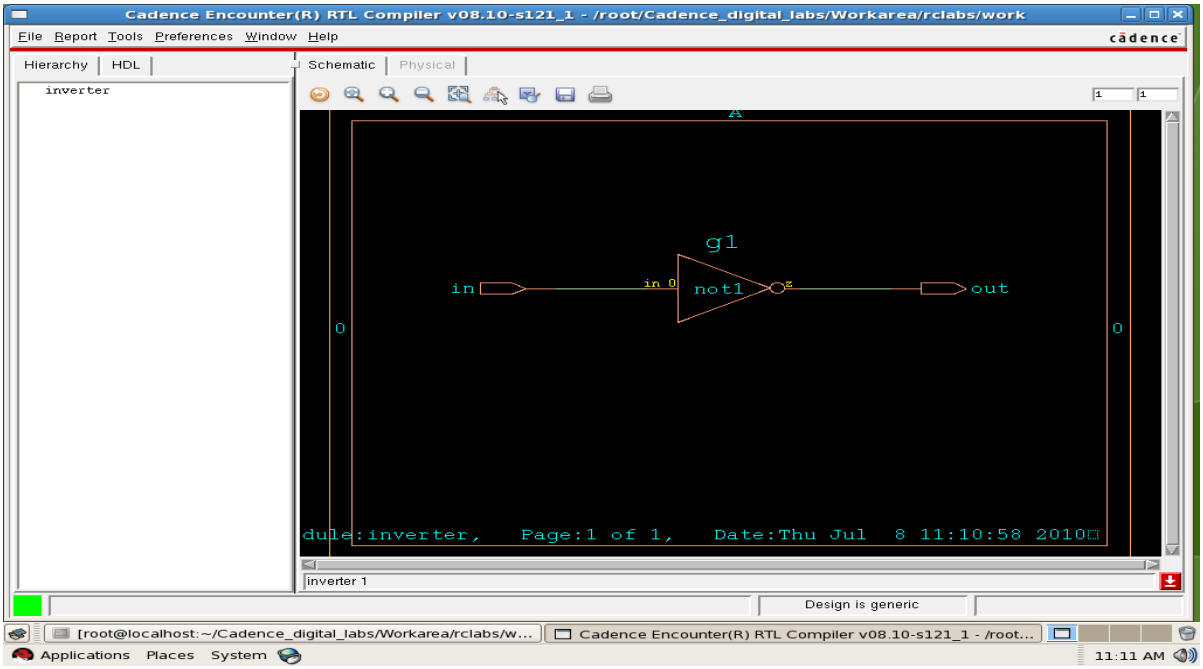


4. Next, click the *waveform icon* on the right hand side corner of the Design Browser window. Now the waveform window will open.

5. Click the *play* button on the waveform window to run the simulation. The waveforms will be generated as shown –



6. For synthesis, follow the steps that are mentioned previously. After synthesis, the synthesis window will be shown as follows –



ii) 2 input NAND and NOR gates

Design Files: The main design modules and the respective test bench modules are given as follows, in the order of – NAND, NOR

Note: The words *nand*, *nor*, *xor* and *xnor* are the gate primitives in Verilog, and hence, care must be taken not to use these keywords as module names or file names.

I Main Design Module: Nandg.v

```
`timescale 1 ns / 1 ns

//Define our own Nand Gate

 module nandgate (out, in1, in2);

// Declarations of I/O, Power and Ground Lines

        output out;

        input in1, in2;

        nand n1(out,in1,in2);

endmodule
```

Test Bench Module: Nand_test.v

```
`timescale 1 ns / 1 ns

// Testbench for Nand Gate Module
```

```
module nand_test;
        wire out;
        reg in1, in2;
        // Instantiate Nand Gate Module
        nandgate n1 (out, in1, in2);
// Apply Stimulus
        initial
                begin
                        in1 = 1'b0; in2 = 1'b0; #10;
                        in1 = 1'b0; in2 = 1'b1; #10;
                        in1 = 1'b1; in2 = 1'b0; #10;
                        in1 = 1'b1; in2 = 1'b1; #10;
                end
endmodule
```

III Constraints file for Synthesis: Constraints_nand.g

```
set_input_delay -max 1.0 [get_ports "in1"]
set_input_delay -max 1.0 [get_ports "in2"]
set_output_delay -max 1.0 [get_ports "out"]
```

NOR GATE

Main Design Module: Norg.v

```
`timescale 1 ns / 1 ns
//Define our own Nor Gate
 module norgate (out, in1, in2);
// Declarations of I/O, Power and Ground Lines
        output out;
        input in1, in2;
         nor n1(out,in1,in2);
```

```
endmodule
```

II Test Bench Module: Nor_test.v

```
`timescale 1 ns / 1 ns
// Testbench for Nor Gate Module
module nor_test;
    wire out;
    reg in1, in2;
    // Instantiate Nor Gate Module
    norgate n1 (out, in1, in2);
// Apply Stimulus
    initial
        begin
            in1 = 1'b0; in2 = 1'b0; #10;
            in1 = 1'b0; in2 = 1'b1; #10;
            in1 = 1'b1; in2 = 1'b0; #10;
            in1 = 1'b1; in2 = 1'b1; #10;
        end
endmodule
```

III Constraints file for Synthesis: Constraints_nor.g

```
set_input_delay -max 1.0 [get_ports "in1"]
set_input_delay -max 1.0 [get_ports "in2"]
set_output_delay -max 1.0 [get_ports "out"]
```

Result

Following the similar procedure for inverter, the simulation and synthesis can be done and verify using truth table.

Experiment No: 8

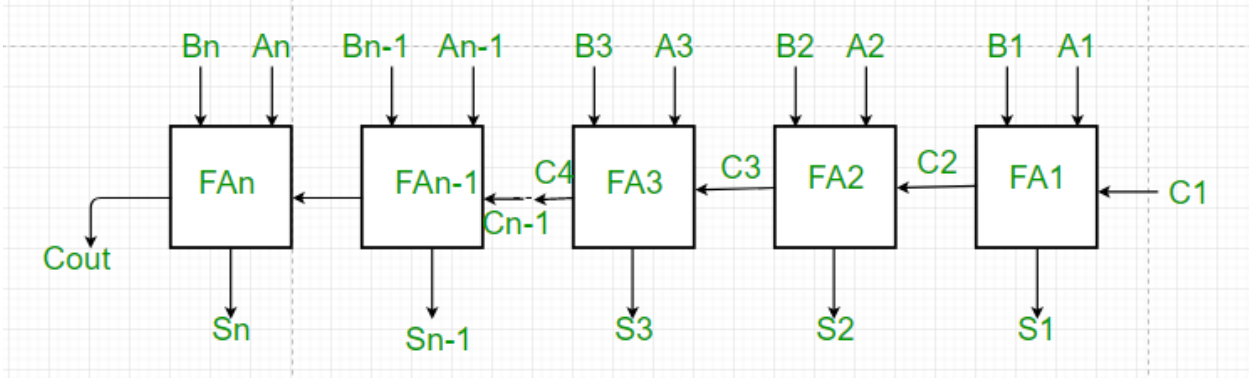
For the following circuits, write the Verilog Code, verify using Test Bench, and then synthesize with the given Constraints:

i) 4-bit Parallel adder, ii) D Flip-flop, iii) T Flip-flop, iv)4-bit Synchronous counter

i) 4-BIT PARALLEL ADDERS:

Aim: To compile and to simulate the Verilog code for parallel adders, and then to synthesize these designs for the given constraints.

Theory: A n bit parallel adder requires n full adders to perform the operation. So for the two-bit number, two adders are needed while for four bit number, four adders are needed and so on. Parallel adders normally incorporate carry lookahead logic to ensure that carry propagation between subsequent stages of addition does not limit addition speed.



Design Files: The main design modules and the respective test bench modules are given as follow 4 bit parallel adder.

I Main Design Module: Parallel_adder.v

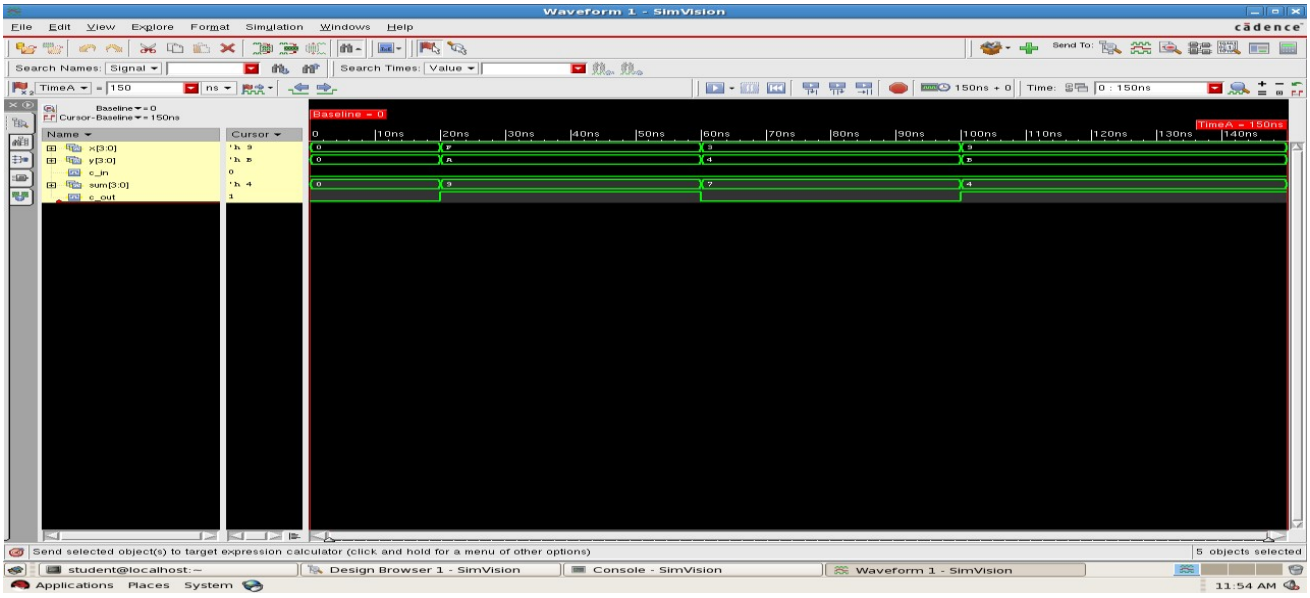
```
module padder ( x, y, c_in, sum, c_out);
        input [3:0] x, y;
        input c_in;
        output [3:0] sum;
        output c_out;
        wire c1, c2, c3;

        fulladd stage0 (x[0], y[0], c_in, sum[0], c1);
        fulladd stage1 (x[1], y[1], c1, sum[1], c2);
        fulladd stage2 (x[2], y[2], c2, sum[2], c3);
        fulladd stage3 (x[3], y[3], c3, sum[3], c_out);
endmodule

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

```
        output s, cout;

        assign s = a^b^cin;
        assign cout =(a & b)|(b & cin)|(a & cin);
endmodule
```

II <u>Test Bench Module</u>: Parallel_adder_test.v

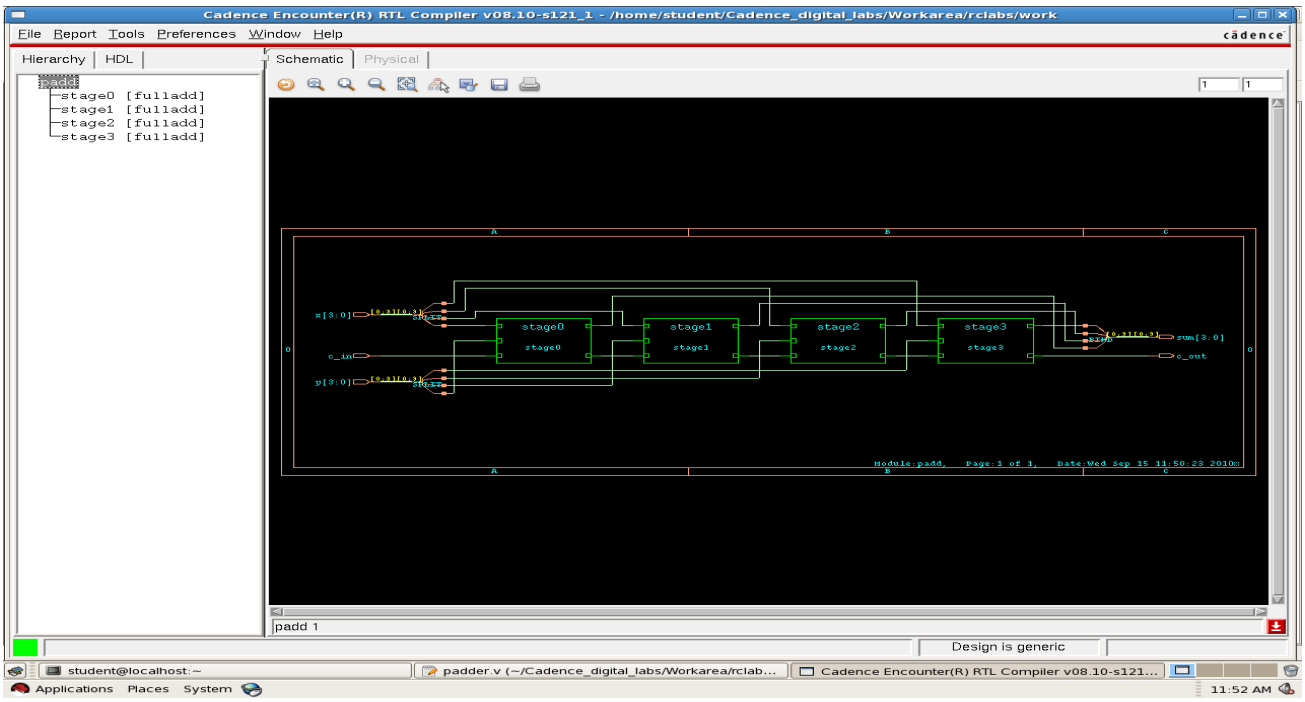
```
module padder_test ;
        reg [3:0] x, y;
        reg c_in;
        wire [3:0] sum;
        wire c_out;

        padder a1 (x, y, c_in, sum, c_out);

        initial
         begin
                  x = 4'b0000; y= 4'b0000; c_in = 1'b0;
                  #20 x = 4'b1111; y = 4'b1010;
                  #40 x = 4'b0011; y = 4'b0100;
                  #40 x = 4'b1001; y = 4'b1011;
                 #50 $finish;
        end
endmodule
```

Following is the simulation result snapshot for the parallel adder –

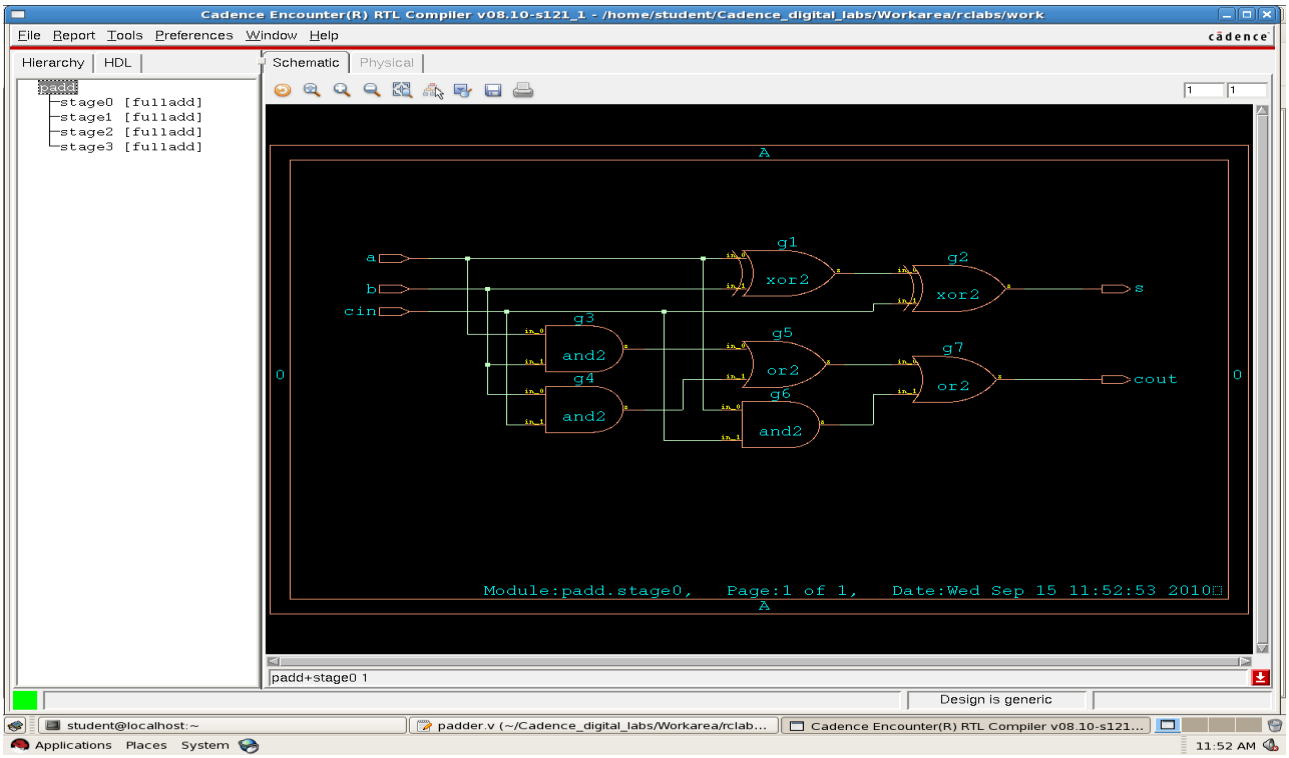


III <u>Constraints file for Synthesis</u>: Constraints_pa.g

set_input_delay -max 1.0 [get_ports "x"]
set_input_delay -max 1.0 [get_ports "y"]
set_input_delay -max 1.0 [get_ports "c_in"]
set_output_delay -max 1.0 [get_ports "sum"]
set_output_delay -max 1.0 [get_ports "c_out"]
Following is the synthesis result snapshot for the parallel adder –



When double-clicked inside the blocks shown, the circuit diagram of the full adder gets displayed –



ii) D Flip-flop

Aim: To compile and to simulate the Verilog code for the common flip-flops, and then to synthesize those designs for the given constraints.

Theory : The D flip-flop is widely used. It is also known as a "data" or "delay" flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line.

In T flip flop, If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value.

Design Files: The main design modules and the respective test bench modules are given as follows, in the order of – D, T. The compiler directives are not used in these modules, as the primitives are not used in the design modules, and the timing is specified in the test bench modules.

D FLIP-FLOP

I Main Design Module: D_ff.v

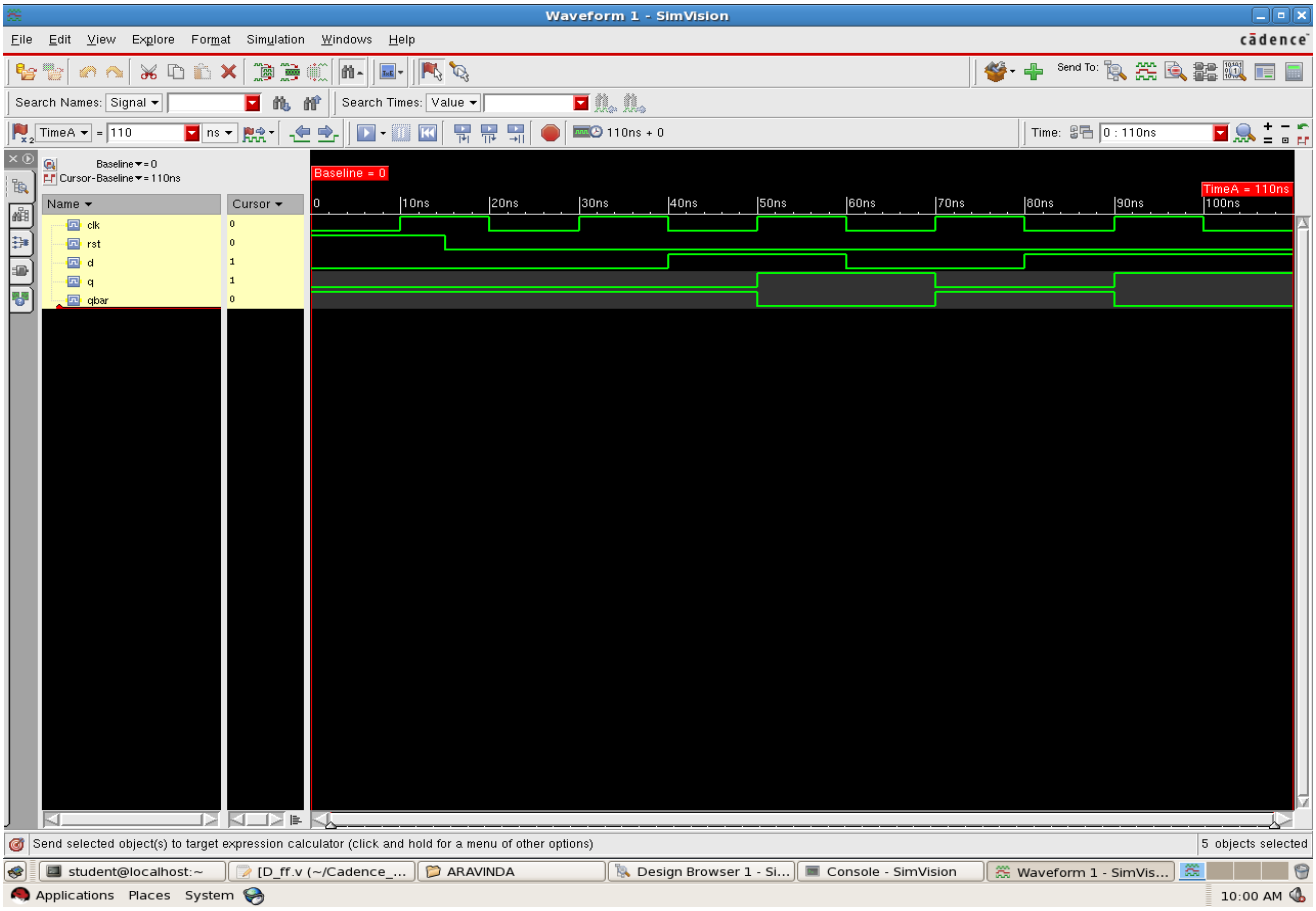
```
module dff(q, qbar, d, clk, rst);
        output q, qbar;
        input clk, d, rst;
        reg tq;

        always @(posedge clk or posedge rst)
        begin
                if (rst)
                        tq <= 1'b0;
                else
                tq <= d;
        end
        assign q = tq;
        assign qbar = ~ tq;
endmodule
```

II Test Bench Module: D_ff_test.v

```
module dff_test;
        reg clk, d, rst;
        wire q, qbar;
        dff d1(q, qbar, d, clk, rst);

        initial
        clk = 1'b0
```

```
        always
        #10 clk = ~ clk;

        initial
        begin
                rst = 1'b1;
                d = 1'b0;
                #15 rst = 1'b0;
                #25 d = 1'b1;
                #20 d = 1'b0;
                #20 d = 1'b1;
        end
        initial
        #110 $finish;
endmodule
```

Following is the simulation result snapshot for the D flip-flop –

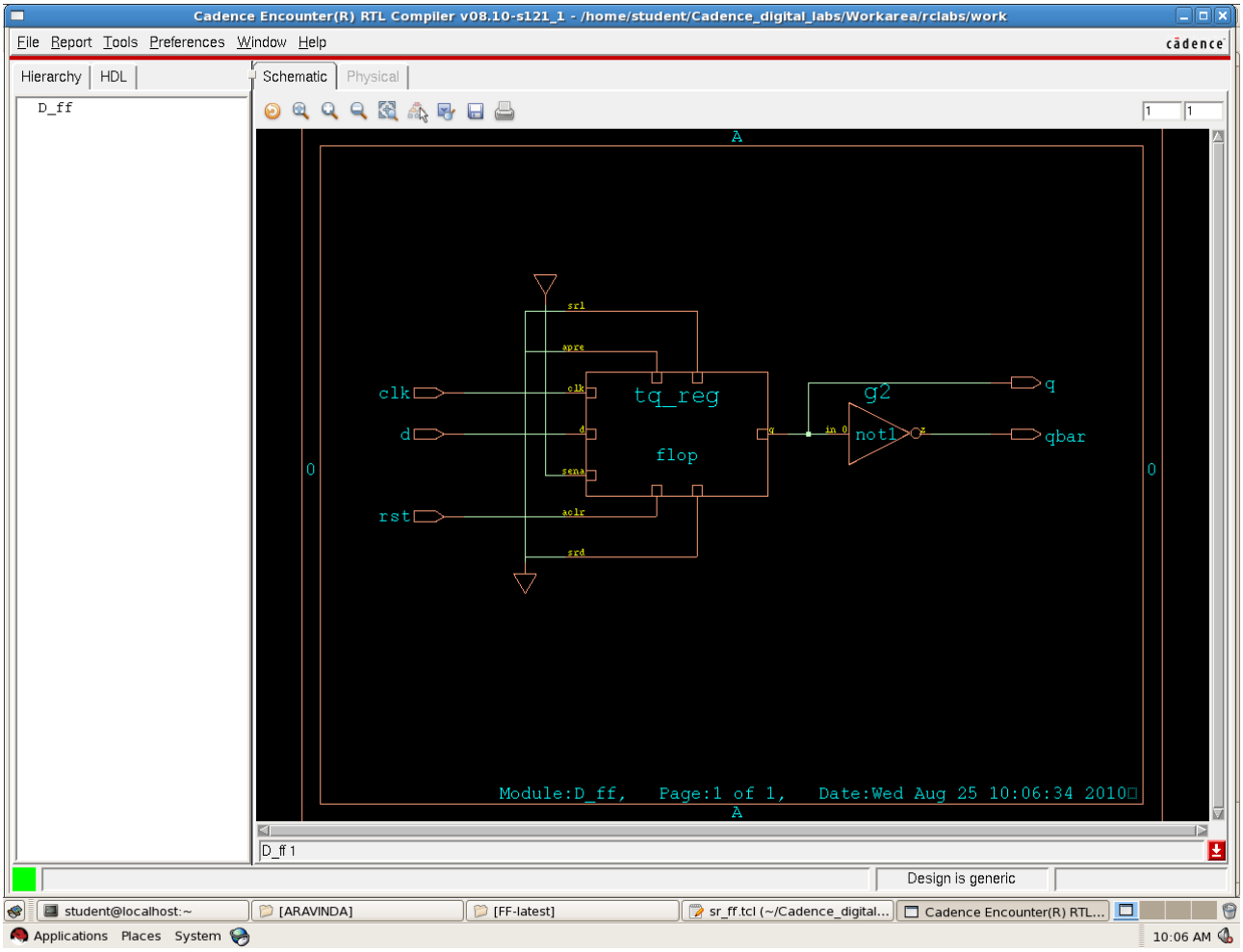


III Constraints file for Synthesis: Constraints_D_ff.g

```
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
```

```
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 1.0 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "rst"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "d"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "q"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "qbar"] -clock [get_clocks "clk"]
```

Following is the synthesis result snapshot for the D flip-flop –



iii) T FLIP-FLOP:

Main Design Module: T_ff.v

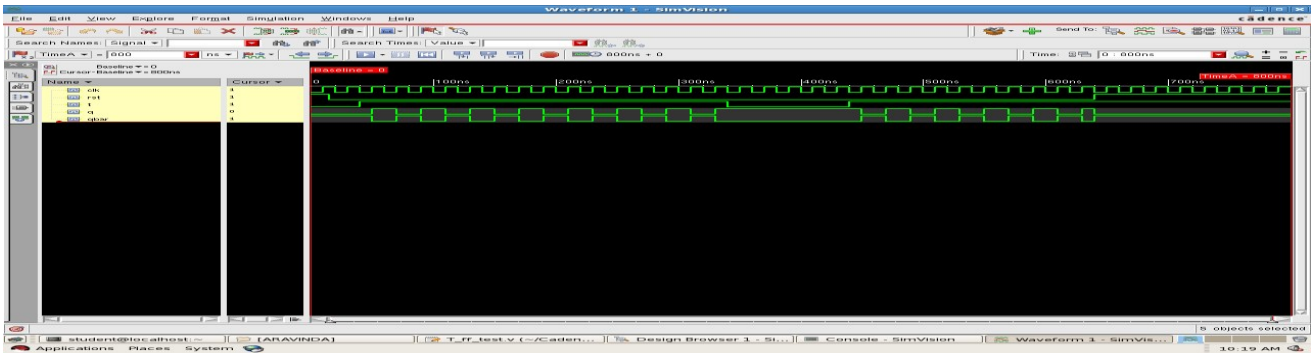
```
module tff(q, qbar, t, clk, rst);
        output q, qbar;
        input clk, t, rst;
        reg tq;
        always @(posedge clk or posedge rst)
        begin
```

```
            if (rst)
                  tq <= 1'b0;
            else
            begin
                  if (t)
                        tq <= ~ tq;
            end
      end
      assign q = tq;
      assign qbar = ~ tq;
endmodule
```

II <u>Test Bench Module</u>: T_ff_test.v

```
module T_ff_test;
      reg clk, t, rst;
      wire q, qbar;
      tff t1(q, qbar, t, clk, rst);
      initial
      clk = 1'b0;
      always
      #10 clk = ~ clk;
      initial
      begin
            rst = 1'b1; t = 1'b0;
            #15 rst = 1'b0;
            #25 t = 1'b1;
            #300 t = 1'b0;
            #100 t = 1'b1;
            #200 rst = 1'b1;
      end
      initial
      #800 $finish;
endmodule
```

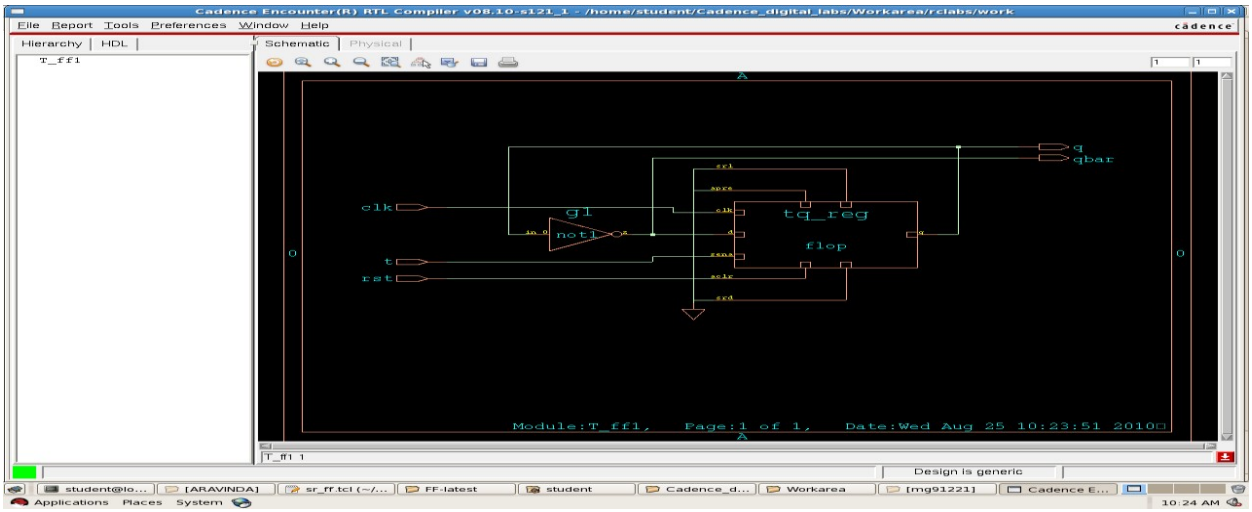
Following is the simulation result snapshot for the T flip-flop –



III Constraints file for Synthesis: Constraints_T_ff.g

```
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 1.0 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "rst"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "t"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "q"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "qbar"] -clock [get_clocks "clk"]
```

Following is the synthesis result snapshot for the T flip-flop –



iii) 4-BIT COUNTERS

Aim: To compile and to simulate the Verilog code for the 4-bit synchronous counters, and then to synthesize those designs for the given constraints.

Theory: Synchronous Counter, the external clock signal is connected to the clock input of EVERY individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously (in parallel) at the same time giving a fixed time relationship. In other words, changes in the output occur in "synchronisation" with the clock signal. A 4 bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 (0000) to 15 (1111).

Design Files: The main design modules and the respective test bench modules are given as follows, in the order of synchronous counter, and then asynchronous counter

SYNCHRONOUS COUNTER

I Main Design Module: Sync_counter.v

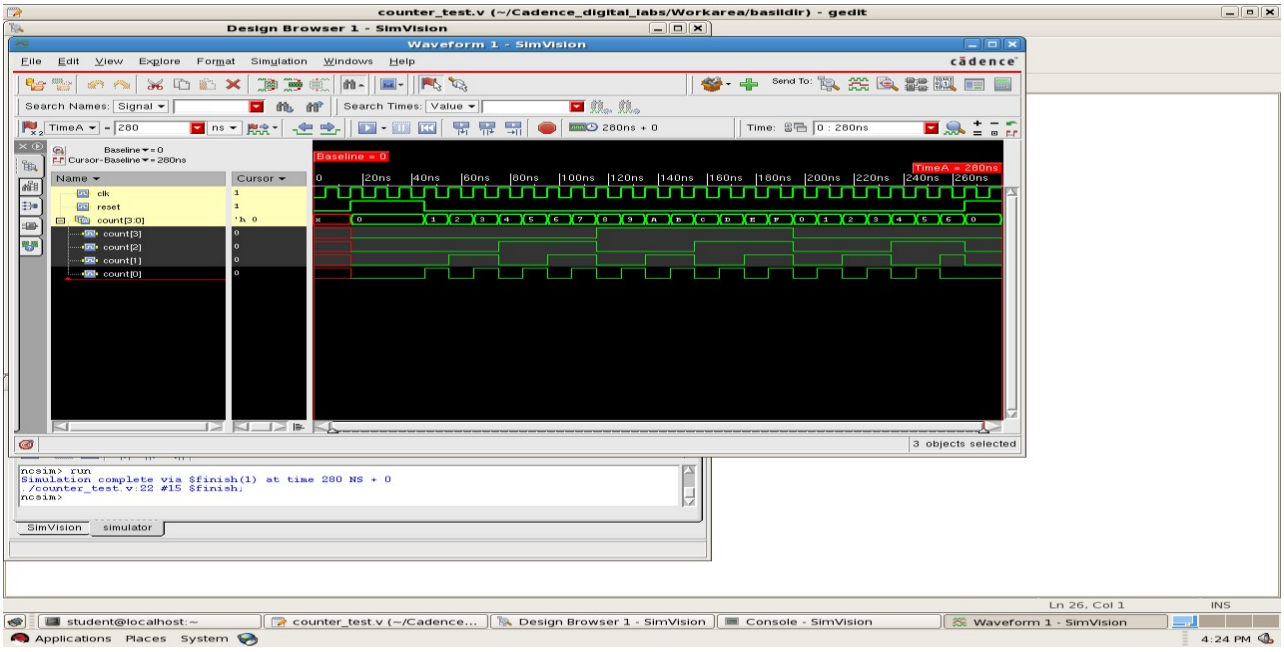
```
module s_counter (clk, reset, count);
        input wire clk, reset;
       output reg [3:0] count;
```

```
        always @(posedge clk)
        if (reset)
                count <= 4'b0000;
        else
                count <= count + 4'b0001;
 endmodule
```

II <u>Test Bench Module</u>: Sync_counter_test.v

```
module s_counter_test ;
        reg clk, reset;
        wire [3:0] count;
        initial
        clk = 1'b0;
        always
        #5 clk = ~ clk;
        s_counter m1 (clk, reset, count);
        initial
        begin
                reset = 1'b0 ;
                #15 reset =1'b1;
                #30 reset =1'b0;
                #220 reset =1'b1;
                #15 $finish;
        end
endmodule
```

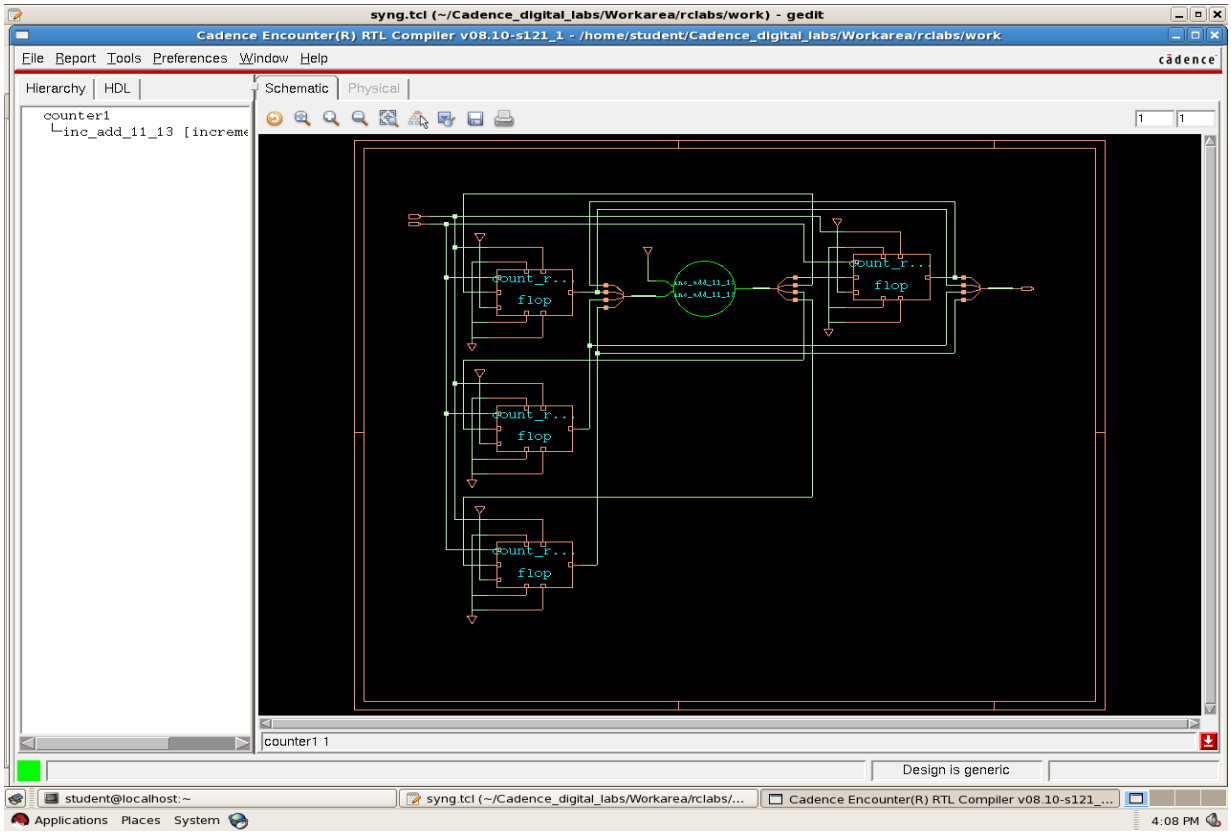
Following is the simulation result snapshot for the synchronous counter –



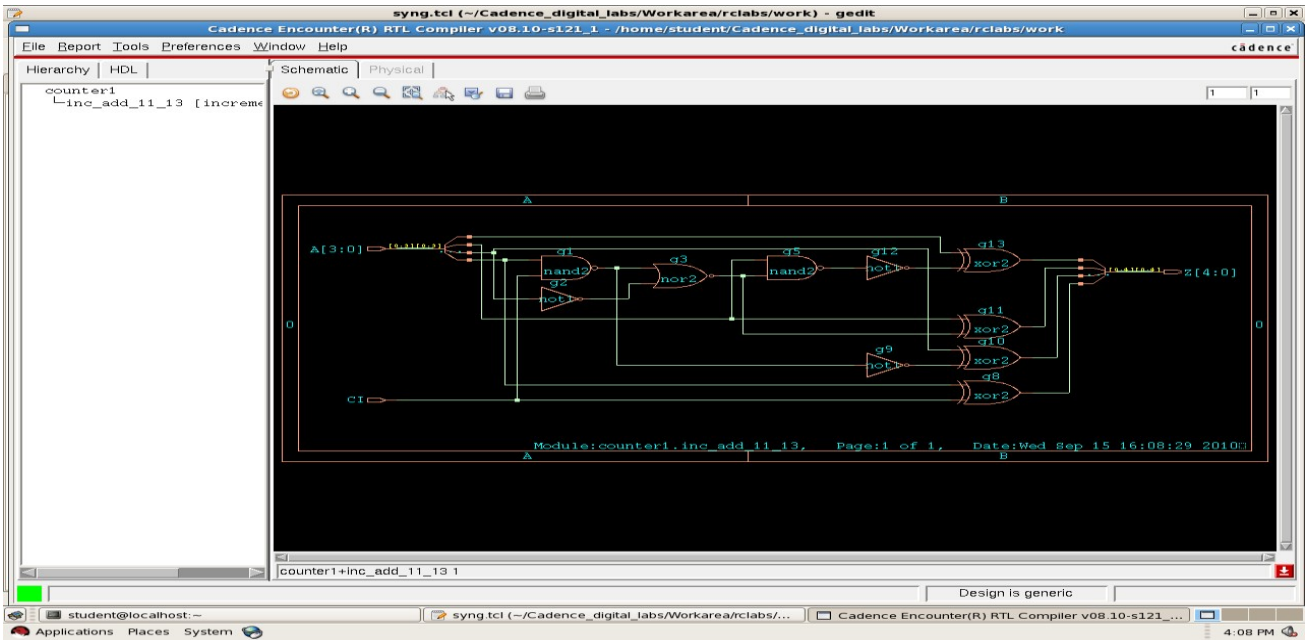
III Constraints file for Synthesis: Constraints_sc.g

```
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 1.0 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "reset"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "count"] -clock [get_clocks "clk"]
```

Following is the synthesis result snapshot for the synchronous counter –



When double-clicked inside the circular block, the logics get displayed as shown –



Result

The Verilog Code for 4-bit Parallel adder, D Flip-flop, T Flip-flop, 4-bit Synchronous counter has been written and verified using test Bench, and synthesized with the given Constraints

FREQUENTLY ASKED VIVA QUESTIONS

1. **Why N-MOS is called good logic 0 and bad logic 1 switch?**

ANS: N-MOS cannot pass full swing V_{DD} when input is VDD with N-MOS On, as

Vgs> Vt is not satisfied . This is called voltage degradation.

2. **What is the switching threshold of a inverter and how to obtain switching threshold = Vdd/2 for inverter?**

ANS : Switching threshold is the input of the inverter at which input is equal to output . Denoted by Vm.

By proper sizing of NMOS and PMOS transistor the switching threshold can be made equal to Vdd/2.

3. **Which are called universal gates and universal element?**

ANS : NAND and NOR are called universal gates

MUX is called a universal elements.

4. **Which are the basic MOS amplifier configurations?**

ANS: Common drain

Common source

Common gate.

5. **Which MOS configuration is used as voltage follower and current amplifier?**

ANS: Common drain

Common gate

6. **What is the advantages of differential amplifier over single stage amplifier?**

ANS: Noise cancellation

Improved voltage swing

7. **What is the use of capacitor in a 2 stage opamp?**

ANS: The capacitor acts as miller capacitor which improves the stability of the opamp by increasing the phase margin.

8. **List the properties of ideal OPAMP**

ANS: Infinite input impedance

Infinite gain

Infinite CMRR

Infinite Bandwidth

Zero output impedance

9. What Is Body Effect?

Answer :

In general multiple MOS devices are made on a common substrate. As a result, the substrate voltage of all devices is normally equal. However while connecting the devices serially this may result in an increase in source-to-substrate voltage as we proceed vertically along the series chain (Vsb1=0, Vsb2≠ 0).Which results Vth2>Vth1.

10. Why Is The Substrate In NMOS Connected To Ground And In PMOS To Vdd?

Answer :

To reverse bias the source bulk pn junction diode.

11. Explain why present VLSI circuits use MOSFETs instead of BJTs?

In comparison to BJT, MOSFETS can be made very compact as they occupy very small silicon area on IC chip and also in term of manufacturing they are relatively simple. Moreover, digital and memory ICs can be employed with circuits that use only MOSFETs, i.e., diodes, resistors, etc.

12. What Is The Fundamental Difference Between A Mosfet And Bjt?

Answer :

In MOSFET, current flow is either due to electrons(n-channel MOS) or due to holes(p-channel MOS) - In BJT, we see current due to both the carriers.. electrons and holes. BJT is a current controlled device and MOSFET is a voltage controlled device.

13. In Cmos Technology, In Digital Design, Why Do We Design The Size Of PMOS To Be Higher Than The NMOS. What Determines The Size Of Pmos Wrt Nmos.

Answer :

In PMOS the carriers are holes whose mobility is less[aprrox half] than the electrons, the carriers in NMOS. That means PMOS is slower than an NMOS. In CMOS technology, nmos helps in pulling down the output to ground and PMOS helps in pulling up the output to Vdd. If the sizes of PMOS and NMOS are the same, then PMOS takes long time to charge up the output node. If we have a larger PMOS than there will be more carriers to charge the node quickly and overcome the slow nature of PMOS . Basically we do all this to get equal rise and fall times for the output node.

14. All Of Us Know How An Inverter Works. What Happens When The Pmos And Nmos Are Interchanged With One Another In An Inverter?

Answer :

I have seen similar Qs in some of the discussions. If the source & drain also connected properly...it acts as a buffer. But suppose input is logic 1 O/P will be degraded 1 Similarly degraded 0;

15. What is the latch up problem that arises in bulk CMOS technology?

Ans: The latch-up is an inherent problem in both n-well as well as pwell based CMOS circuits. The phenomenon is caused by the parasitic bipolar transistors formed in the bulk of silicon. Latch-up can be defined as the formation of a low-impedance path between the power supply and ground rails through the parasitic npn and pnp bipolar transistors. Leakage current through the parasitic resistors can cause one transistor to turn on, which in turn turns on the other transistor due to positive feedback and leading to heavy current flow and consequent device failure.

16. Explain the three modes of operation of a MOS transistors

Ans: The three modes are: (a) Accumulation mode when Vgs is much less than is much less than Vt (b) Depletion mode when mode when Vgs is equal to is equal to Vt (c) Inversion mode when mode when Vgs is greater than is greater than Vt.

17. What are the three regions of operation of a MOS transistor?

Ans: The three regions are: Cut-off region: This is essentially the accumulation mode, where there is no effective flow of current between the source and drain.

Non-saturated region saturated region: This is the active, active, linear or week inversion region, where the drain current is dependent on both the gate

Saturated region: The drain current is independent of the drain-to-source voltage but depends on the gate voltage.

18. What is the threshold voltage of a MOS transistor? How it varies with the body bias?

Ans: One of the parameters that characterizes the switching behavior of a MOS transistor is its threshold voltage Vt. This can be defined as the gate voltage at which a MOS transistor begins to conduct.

19. Explain what is Verilog?

Verilog is an HDL (Hardware Description Language) for describing electronic circuits and systems. In Verilog, circuit components are prepared inside a Module. It contains both behavioral and structural statements. Structural statements signify circuit components like logic gates, counters and micro-processors. Behavioral statements represent programming aspects like loops, if-then statements and stimulus vectors.

20. Mention what are the two types of procedural blocks in Verilog?

The two types of procedural blocks in Verilog are

Initial: Initial blocks runs only once at time zero

Always: This block loop to execute over and again and executes always, as the name suggests

21. Define synthesis?

Synthesis is the process of transforming your HDL design into a gate-level netlist.

22. What are the Cadence tools used in the CMOS VLSI Design

- Virtuoso : Analog Design
- Spectre : Analog simulation
- Assura : Analog layout and RC extraction
- NCSIM : Native Complier Simulator for digital simulation
- RC Encounter : Digital synthesis

23. Acronyms

- DRC: Design Rule Check
- LVS : Layout Vs Schematic
- RCX: RC extraction
- ERC: Electric Rule check
- ADE :Analog Design Enviornment

24. Linux commands

- csh: C Shell
- cshrc: C Shell Run Commands
- ls: list
- cd : change directory
- cp :copy