

ECE484 Laboratory Manual - Fall 2020
Version 2.1
(Compatible with AMS HitKit)

Contents

1	Introduction	2
2	LINUX Tutorial	4
3	Cadence Setup	5
4	Schematic Entry	8
5	Symbol Creation	16
6	Electrical Simulation	19
7	Physical Layout, DRC, and LVS	29
7.1	Layout	29
7.2	Design Rule Check (DRC)	38
7.3	Layout Versus Schematic (LVS) Check	41
8	Space Based Router	46
8.1	4-Bit Synchronous Binary Counter	46
8.2	Schematic Entry and Symbol Creation	47
8.3	Electrical Simulation	57
8.4	Physical Layout, DRC, and LVS	64
8.4.1	Layout	64
8.4.2	Design Rule Check (DRC)	81
8.4.3	Layout Versus Schematic (LVS) Check	82
9	Verilog Simulation	85
10	Logic Synthesis	90
11	Place and Route	97
12	Parasitic Extraction User Guide	103
13	Cadence Hot Keys	114

1 Introduction

This document is to serve as the lab manual for the ECE484 (Digital IC Design) class which is offered each Fall by the SIUE ECE Department. The manual was prepared by Professor G. Engel (August 2018). The purpose of the manual is to train students in the use of the Cadence EDA (Electronic Design Automation) tools which we currently use here in the ECE Department. Specifically, the manual describes how to use the custom IC design tools which comprise the Cadence IC6 software suite.

Prior to Fall 2018, the version 6 (IC6) toolset has been used in the IC design courses offered by the Department but Cadence is urging customers to adopt the IC6 toolset. Moreover, starting in the 2018 Fall semester, we will be using the AMS 0.35 micron Process Design Kit (PDK) which is generally referred to as the AMS HitKit. This has made it necessary for us to update the tutorials used by students in ECE484. The tutorials in this new manual are based on the Washington University in Saint Louis (WUSTL) Cadence Tutorials. The WUSTL tutorials were prepared by Michael Hall (a former SIUE student and graduate assistant of Dr. Engel's). The WUSTL tutorials are a good place to look for additional information on how to use the Cadence design software.

This manual will walk you through all the necessary steps for designing and testing an inverter. First, we are going to create a schematic for the inverter. We, then will create a symbol for the inverter and test the transient characteristics of this inverter using Analog Artist Simulator. Lastly, we are going to create a layout for the inverter and test the transient and DC characteristics using a netlist extracted from the layout of the inverter. This final simulation will include parasitic capacitances which will more accurately reflect the true performance of the inverter.

Labs for the ECE484 class will be carried out in EB3009. EB3009 will serve as the new Student Computer Aided Design (SCAD) Laboratory. The room is home to about a dozen computers running Centos 6.9 (Linux). The Cadence IC design tools reside on each of the computers, but student files are stored on a server (vlsi.ece.siu.edu) located in Dr. Engel's research laboratory.

The machines in EB3009 mount `vlsi`'s home directory and the campus server provides user authentication. Therefore, provided you have an SIUE computer account (and you remember the password!), you can get an account on `vlsi.siu.edu` and all of the other computers in the SCAD lab. Since your Linux home directory resides on "`vlsi`", feel free to use any of the computers in the lab. **NOTE: Before using the machines in the SCAD lab, one must have an account on the vlsi server. These accounts will be created with the help of the ECE484 teaching assistant during the first week of lab in ECE484.** The TA will also create an `ece484` project directory for you by running a setup script (namely, `clone-ece484`).

The computers in the SCAD lab have been assigned the following hostnames:

- scad0.ece.siue.edu (NOT WORKING)
- scad1.ece.siue.edu
- scad2.ece.siue.edu
- scad3.ece.siue.edu (NOT WORKING)
- scad4.ece.siue.edu
- scad5.ece.siue.edu
- scad6.ece.siue.edu
- scad7.ece.siue.edu
- scad8.ece.siue.edu
- scad9.ece.siue.edu (USED BY TA)
- scad10.ece.siue.edu
- scad11.ece.siue.edu
- scad12.ece.siue.edu

The combination to the room will be given to you during the first lab meeting. Please **DO NOT** share the information with others!.

Finally, Cadence asks that any tutorials prepared by faculty come with following disclaimer:

Information is provided "as is" without warranty or guarantee of any kind. No statement is made and no attempt has been made to examine the information, either with respect to operability, origin, authorship, or otherwise.

Please use this information at your own risk –and any attempt to use this information is at your own risk – we recommend using it on a copy of your data to be sure you understand what it does and under what conditions. Keep your master intact until you are personally satisfied with the use of this information within your environment."

Cadence® is a trademark of Cadence Design Systems, Inc. 555 River Oaks Parkway, San Jose, CA 95134.

2 LINUX Tutorial

Login to the machines in the SCAD lab by using your university e-id and password. Since many of you may have never used the Linux operating system before, I will provide a brief introduction to Linux here. Begin by double-clicking on the terminal icon. This will launch an xterm window. Here is a link to a Linux Tutorial for beginners. I suggest you become familiar with the following commands: **cd**, **ls**, **cat**, **pwd**, **less**, **mv**, **rm**, **mkdir**, **rmdir**, **which**, **find**. Give the following commands and see do what you would expect based on the tutorials. The TA will explain what the sequence of commands do in more detail.

```
cd ~
```

```
pwd
```

```
ls
```

```
ls -al
```

```
cat ~/.alias
```

```
less ~/.alias
```

```
mkdir tmpdir
```

```
cd tmpdir
```

```
pwd
```

```
gedit tmpfile
```

```
ls
```

```
cd ..
```

```
pwd
```

```
rm -rf ./tmpdir
```

```
ls
```

```
go ./
```

Pretty cool? Linux is a very nice operating system!.

3 Cadence Setup

You can check to make sure that the ece484 project directory was created correctly by typing

```
cd ~/cds/ece484
```

```
ls
```

You should see several subdirectories including a **Lib** and a **Libtest** directory.

Each time you log in, you should open up a terminal window. In order to use the Cadence Version 6 tools you must type the following command

```
cds_ams
```

This will configure the environment for you. You should make sure that the programs will use the ece484 project directory. To check this, type the command

```
p
```

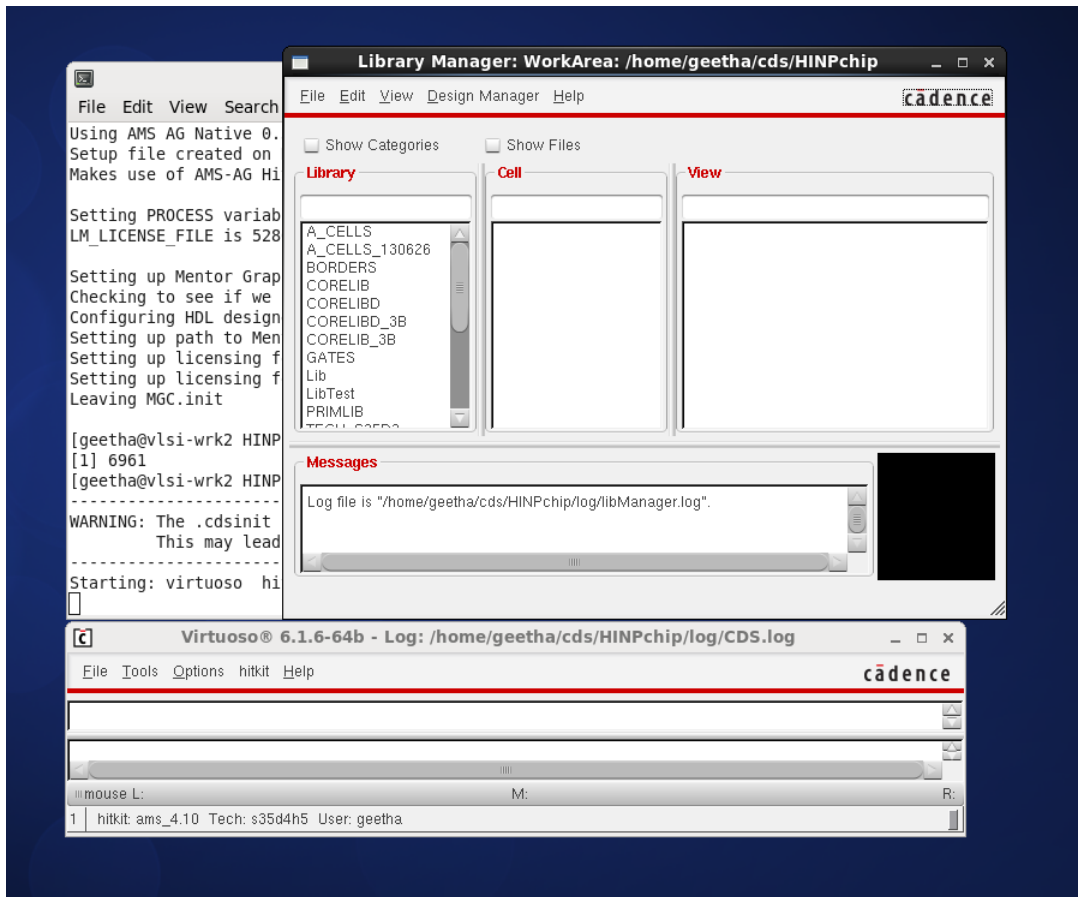
The command should return the string

```
ece484
```

Then to actually launch the tools, you should type

```
icd_ams
```

This launches Cadence's Virtuoso program. The following window will pop up.

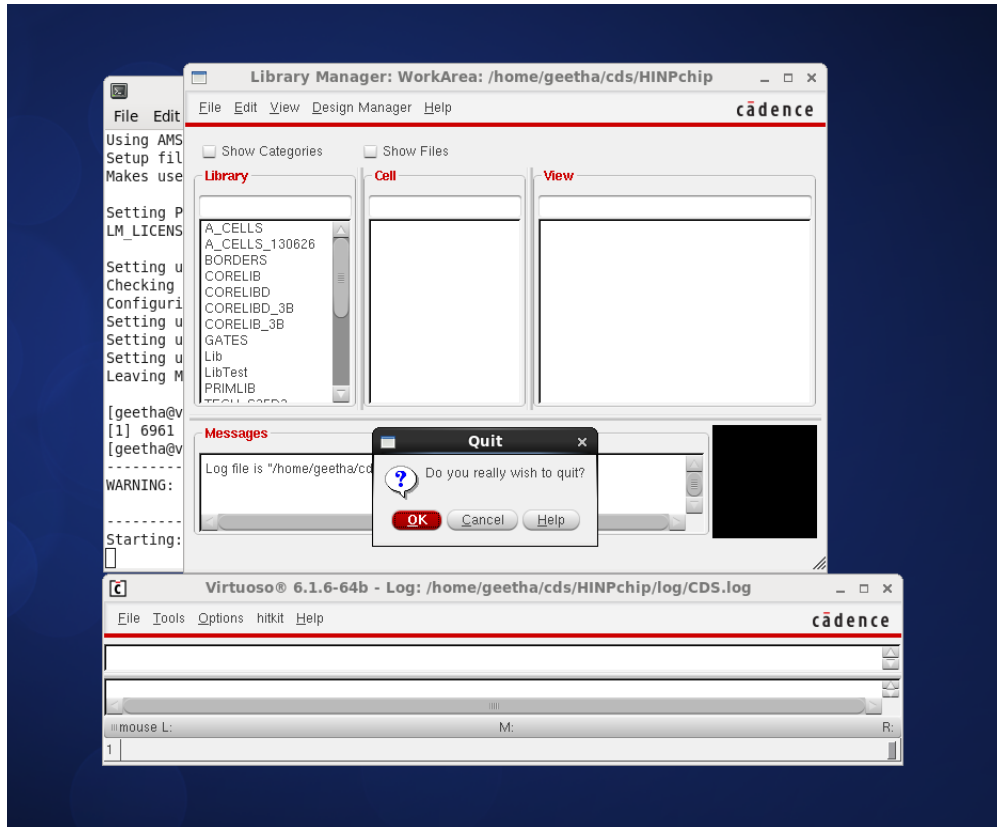


Click the **YES** button.

In the next section of this manual, we will actually begin designing a circuit (specifically, we will perform a task known as schematic entry). All I ask that you do now, is exit from the program. Here is how one exits from the Virtuoso program.

Left click CIW: File → Exit...

A dialog box should appear, and ask to confirm the request. Left click **yes** to exit Cadence, or left click **no** to cancel and resume. If you have any unsaved work, the **Save All** form appears with the library, cell, and cell view names. All Cadence windows will now close, and the design session will end.



If you want to use the Cadence tools in a different class, for example, ece585 then issue the command

sp ece585

If you would like ece585 to be your default project then issue the commands

ulp

sp ece585

lp ece585

The **sp** command "sets the current working project". The **ulp** command "unlocks the default project" and the **lp** command "locks the current project down as the default project".

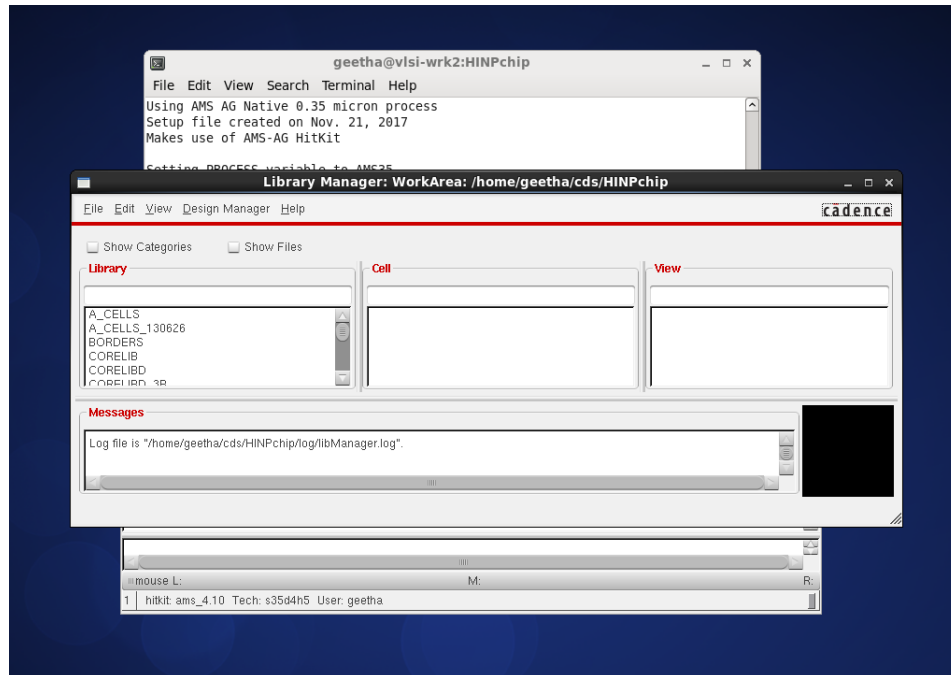
LASTLY, PLEASE MAKE SURE THAT YOU LOG OUT OF THE COMPUTER BEFORE LEAVING THE LAB SO THAT OTHERS CAN USE IT.

4 Schematic Entry

Launch the Cadence tools as described in the previous section of this manual.

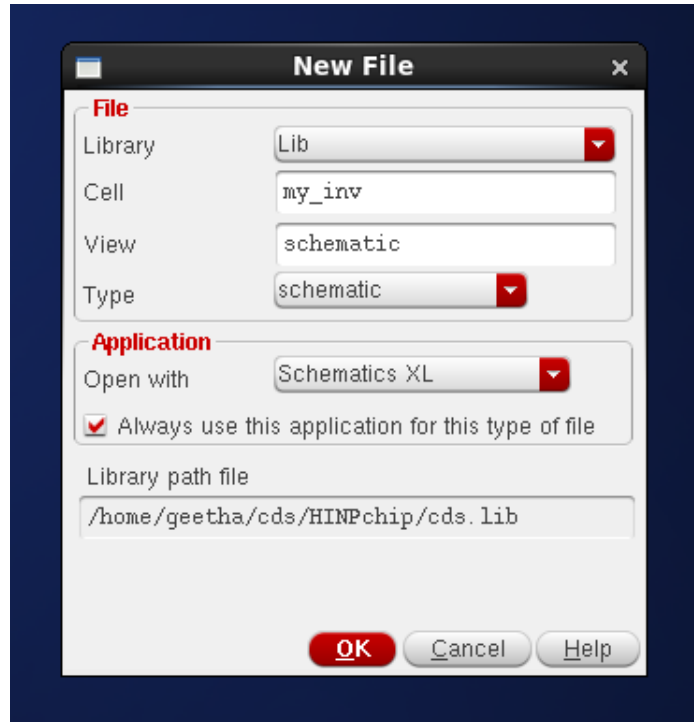
From the CIW window please select **Tools: Library Manager**

You should see the following window pop up. It is what we call the "Library Manager". We will use the Library Manager each time we create a new design. We can also use it to copy and/or delete designs.

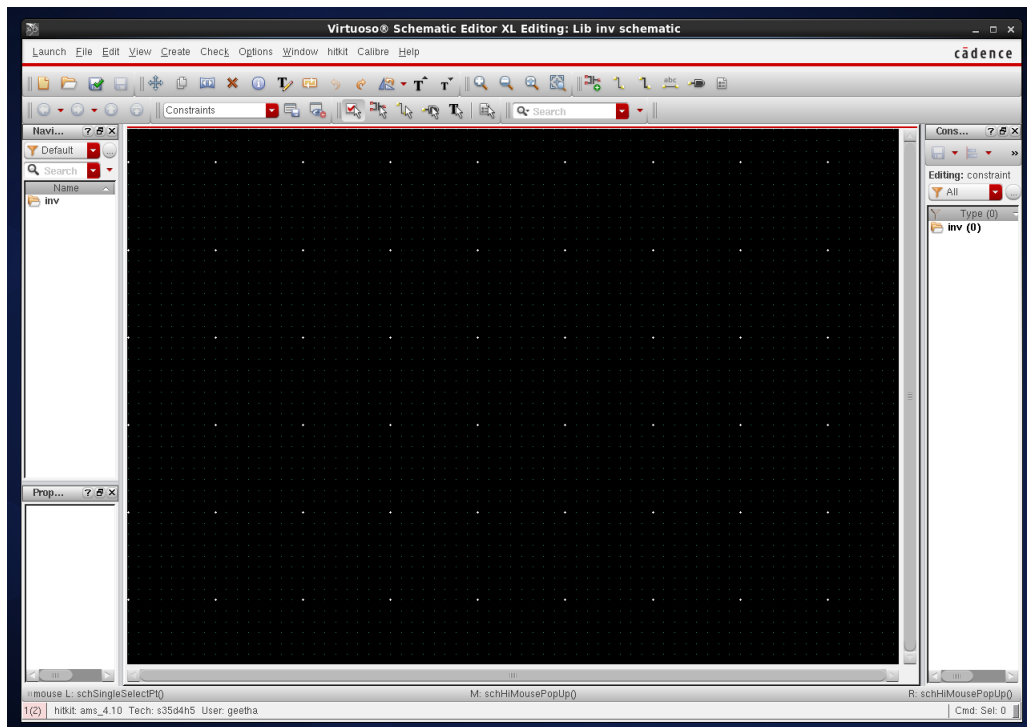


Whenever we create a schematic for a logic cell in ECE484, it should be saved in the **Lib** directory. When we create a schematic of a "testbench" (something we will use to test the cell), we will want to save the testbench in **LibTest**. We will find things like nfets, pfets, etc. in the **PRIMLIB** library. Items such as ideal resistors, ideal capacitors, voltage sources, *etc* can be found in **analogLib**.

First select Lib and then **File: New** → **Cell View**. The **New File** window will pop up. Let's make a digital inverter cell which we will call **my_inv** so type **my_inv** in the the **Cell** field. In the the **View** field type **schematic** or from the **type** pull-down menu choose **schematic** and the **View** fiels will be automatiially filled in. In the **Application** section, select the following: Open with **Schematic XL** and check the box that says *Always use this application for this type of file*.



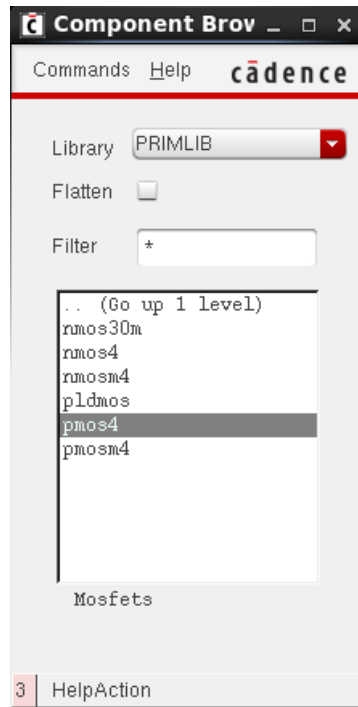
Left click the **OK** button. The **Virtuoso Schematic Editor** window should pop up as shown in the figure below.



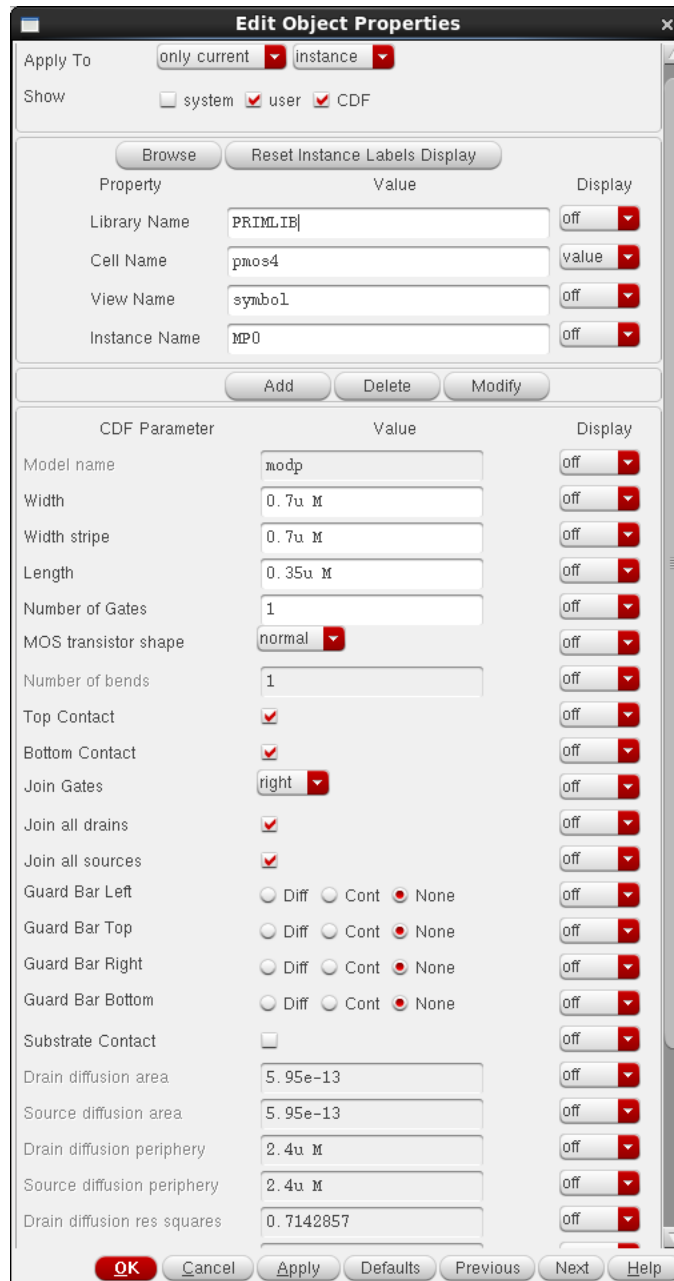
Left click: **Schematic Editor: Create → Instance**

Shortcuts: You may notice letters by some of these menu choices. Rather than searching through the menus, you can just hit that key on the keyboard to have the same effect. These are called **bind-keys**. In the future you can press **i** in order to insert an instance.

A **Command Browser** window appears. In this window select **PRIMLIB** under the **Library** pull-down menu. Next click on **Mosfets** → **pmos4**. The window should look like the one in the figure below.



You can edit the parameters of the **pmos4** cell such as width, length, etc. For this exercise, keep the default parameters. The **Add Instance** window is shown below.



Move the cursor into the editing window. Notice that there is an PMOS transistor there instead of the normal cursor. Position it where you want to put the transistor, and **left click** to place it. You can **right click** to rotate the transistor if you want it to face a different direction (this is especially useful with pins). While placing, stretching, etc., you can press **F3** to show the options form for the command if it is currently hidden.

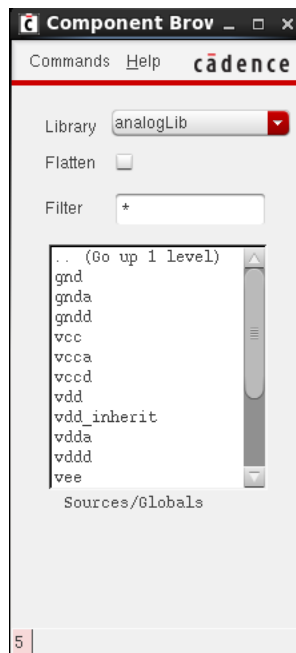
Press **Esc** to return to a normal cursor after you have finished placing all the transistors you want. For this inverter example, place it on the bottom half of the screen on the right side of the center-line. To rotate press **r**.

Click **Schematic Editor: Create** → **Instance**

Follow the same steps as before, but choose a **nmos4** transistor. Use the default values for length and width. Place the NMOS transistor somewhere below the PMOS transistor.

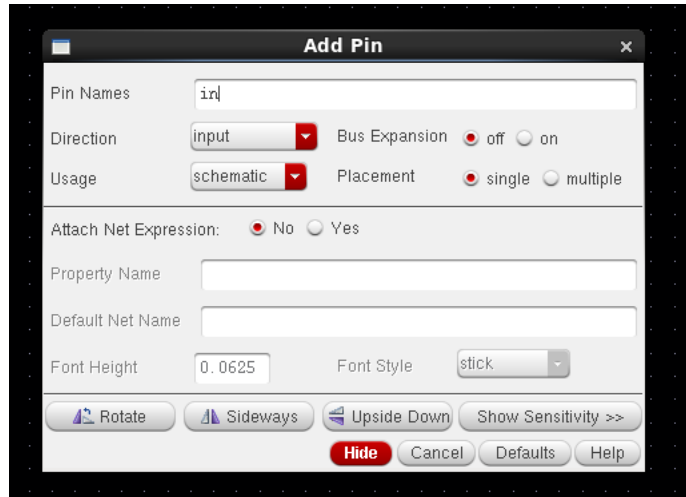
To make life easier: Before trying to place a component, **left click** the **Hide** button on the **Add Component** window. This will move it into the background so it's out of your way.

Repeat the same procedure as above and add both **gnd** and **vdd** symbols (from **analogLib**). Place the **gnd** symbol below the **nmos4** transistor and place the **vdd** symbol above the **pmos4** transistor. Refer to the final figure at the end of this section to see the placement of all components. When adding the **gnd** and **vdd** symbols, the **Add Instance** window appears and it should look like the ones shown below.



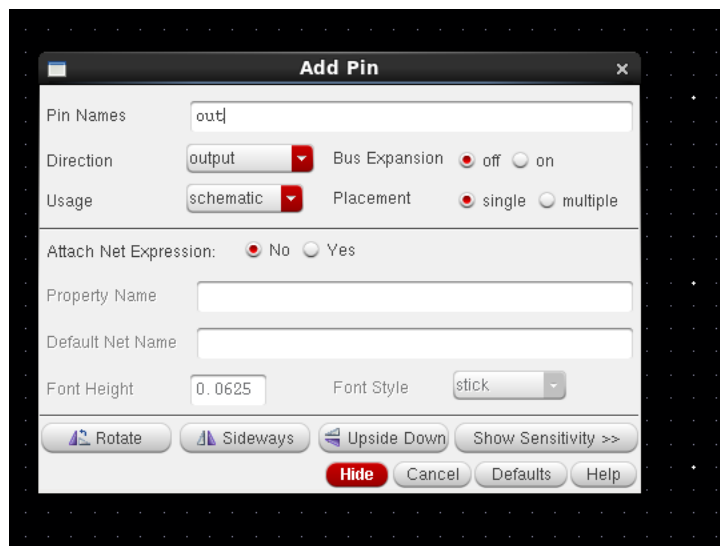
Now, we'll add the pins for the inverter. Click **Schematic Editor:** → **Create Pin** (bind key **p**).

The **Add Pin** dialog box comes up. In the **Pin Names** field enter the pin name in (note, multiple pin names can be given, separated by a space) and ensure that Direction is set to input. The **Add Pin** window should look as follows:



Place the **in** pin somewhere towards the left side of the editing window, between the two transistors.

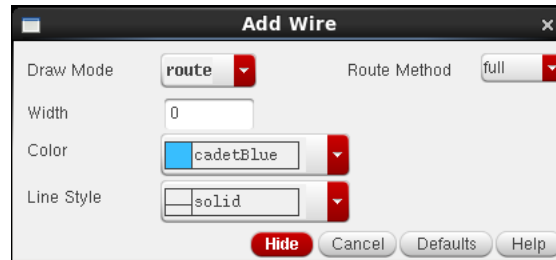
Click **Schematic Editor: → Create Pin** again. The **Add Pin** dialog box comes up. In the Pin Names field enter the pin name out and ensure that Direction is set to output. The **Add Pin** window should look as follows:



Place the **out** pin somewhere towards the right side of the editing window, between the two transistors. Now, we'll add all the wires to make the inverter work. **Click Schematic Editor: Create Wire** (bind key **w**). The **Add Wire** form should appear. Just click its **Hide** button. You can refer to the figure at the end of this section to see how everything is connected together.

Notice that as you get closer to one pin than another (including those on devices), a small diamond will show up either inside of or around that pin. That is where you want to click to

connect a wire. Also, when wiring the schematic, leave the wire width at 0, Route method at full, and (usually) Draw Mode as route. This tells the software to auto-route the wires for you.

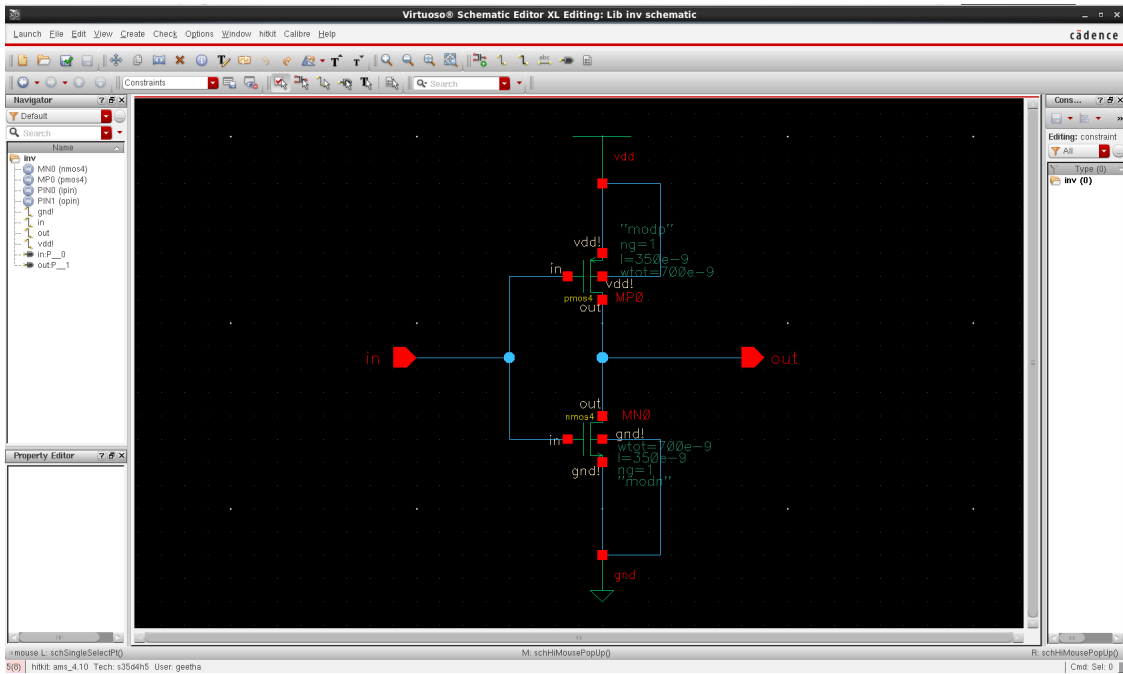


If you put a wire where you don't want it to go, you can delete the wire by left clicking **Schematic Editor: Edit Delete** (bind key **del**) and then left click on the object you want to delete (wire, pin, component, etc.).

Make all of the following wire connections:

- Gates - For the first wire, left click on the gate terminal of the **pmos4** transistor, and left click again on the gate terminal of the **nmos4** transistor. You have just connected the gates.
- Input - Now, move the mouse until the little square is inside the diamond on the **in** pin. Left click in the diamond. Move the cursor over to the wire you connected the two gates together with. A diamond will form around the cursor, as long as it's on the wire. Left click on the wire. You have just connected the input to the gates of both transistors.
- Body - The body of the **nmos4** and **pmos4** transistors are the center pins. On the **pmos4** transistor, connect this pin to **vdd**. On the **nmos4** transistor, connect the body to **gnd**.
- Sources - The source of the **nmos4** transistor is the bottom terminal with an arrow pointing out and the source of the **pmos4** transistor is the top terminal with an arrow pointing in. Connect the source of the **nmos4** transistor to **gnd** and the source of the **pmos4** transistor to **vdd**.
- Drains - There should now only be one pin left on each transistor (the drains of both transistors). Connect these two pins together.
- Output - Finally, connect the drains of the FETs to the **out** pin.

A picture of what this should all look like when completed is shown on the next page.



Once you are done editing, left click the "check mark" icon on the left side of the screen. This will check your work for connection errors and will save your work in the library. You can accomplish the same thing by left clicking **Schematic Editor: File → Check and Save**.

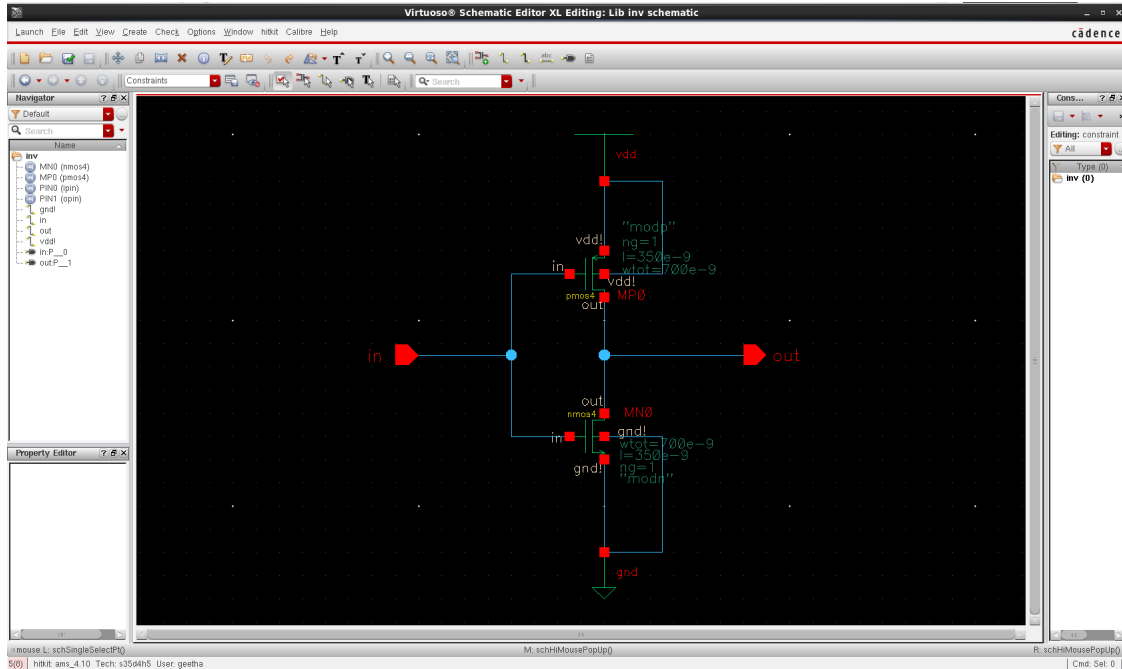
You can now close the Schematic Editor by left clicking on the "X" in the upper right most corner of the Schematic Editor Window.

5 Symbol Creation

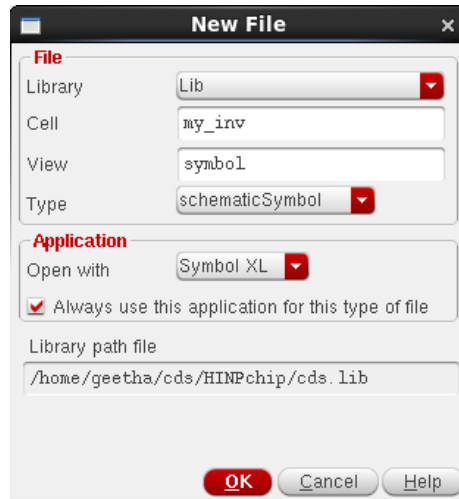
The symbol editor allows one to create a "black box" description of a cell using labels, pins, shapes, notes, and a selection box. Symbols make your design more readable, as you can use them in more complex designs, instead of individual transistors.

There are two different ways you can create a symbol: create a new symbol cell view using the Library Manager, or generate a symbol cell view from the schematic. I will explain the latter which is easier and should be used throughout the semester.

In the Library Manager click once on the **Lib** library to select this library. Then left click the **my_inv** cell and finally the schematic cell view. Right click on the schematic cell view and click Open or double click on schematic to open the schematic editor. You should see the circuit which you previously. It should look something like the circuit shown below.

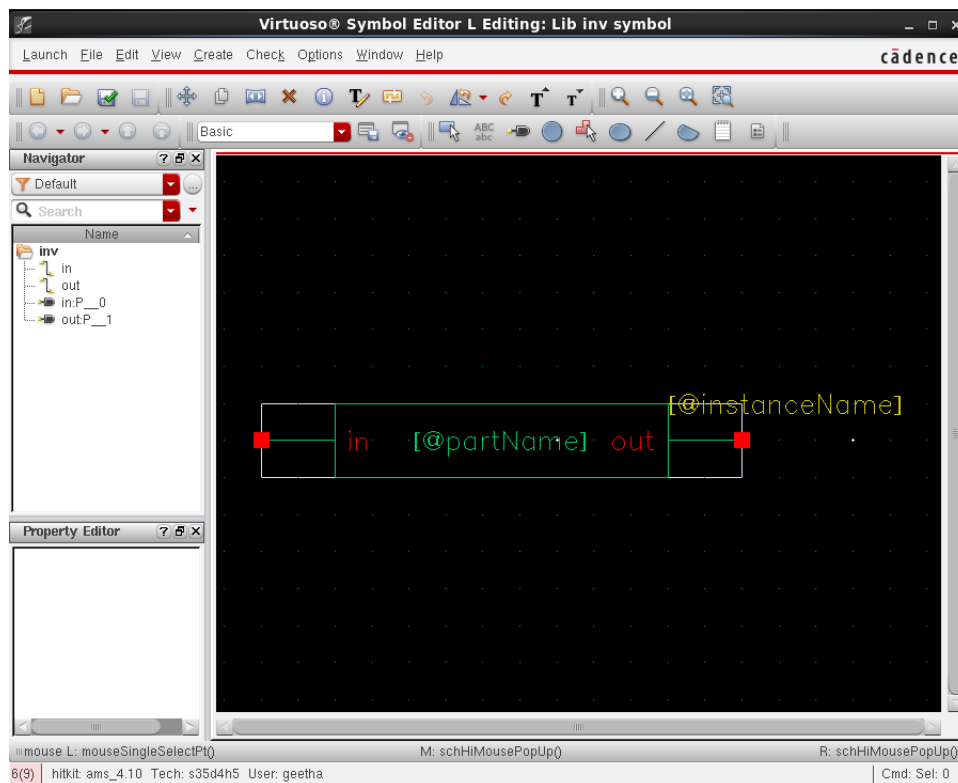


Left click **Schematic Editor: Create** → **Cellview** → **From Cellview...** This will then show the Cellview From Cellview window shown on the next page. This window allows a symbol to be generated from the schematic view. It does this by identifying the pins that are already defined in the schematic and automatically generating a symbol with the same pins.



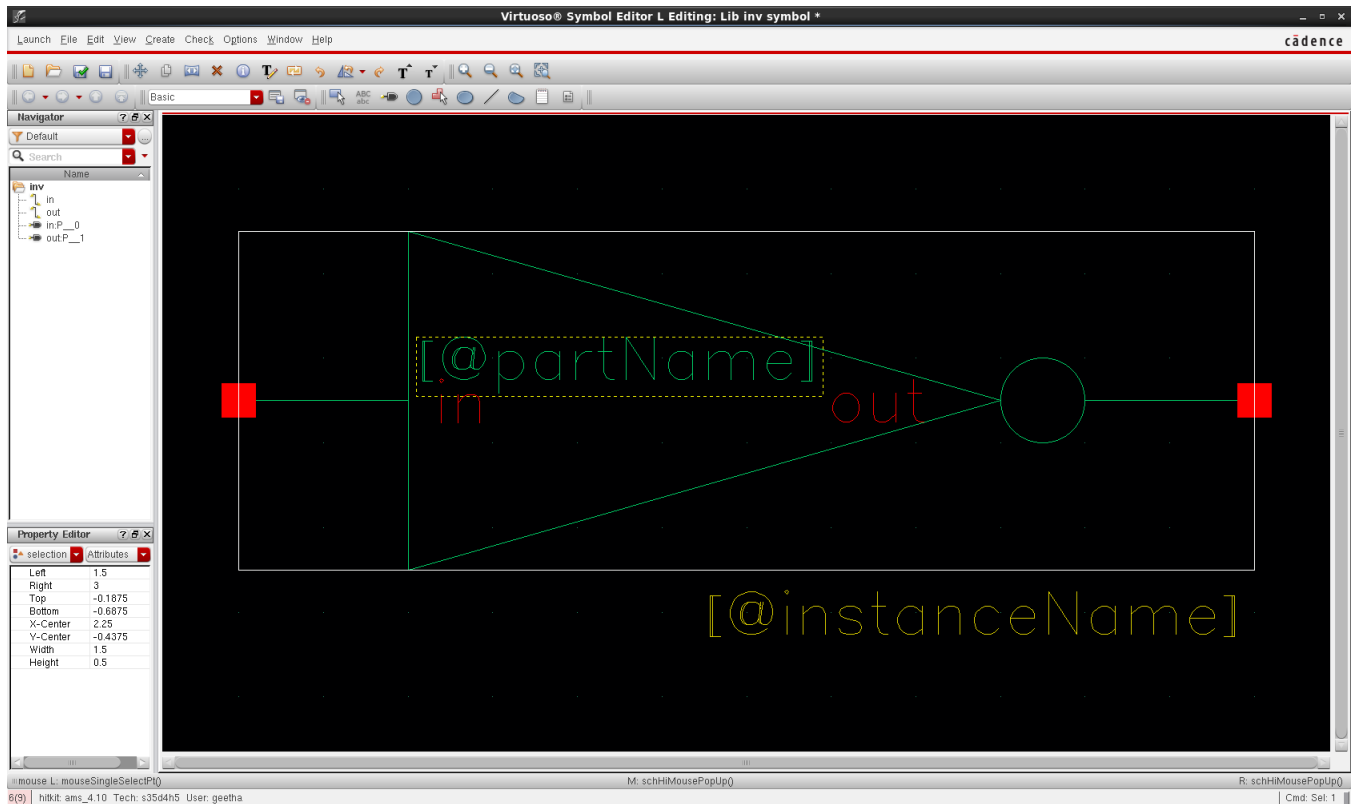
Click OK.

Next, the Symbol Generation Options window will appear which will list the pin specifications of the new symbol. Cadence has already identified the input and output pins and has automatically added the input pin, in, to the left pins and the output pin, out, to the right pins. Cadence has now generated a default symbol using the pin specifications previously given. The symbol should look like the one shown below.



If you wish you can use the drawing tools in the "Symbol Editor" to modify the schematic and make it look like the more traditional symbol used for inverters. The lab TA can help

you do this, if you ask. When finished your symbol might look like the one below.



You should now close the Symbol Editor by left clicking on the "X" in the upper right most corner of the Symbol Editor Window. You should also close the Schematic Editor Window.

Congratulations! You have successfully created your first schematic and associated symbol. But how do we know that the inverter design is correct? Well, we will have to run electrical simulations to prove that indeed the design is correct. We will do that in the next section of this manual.

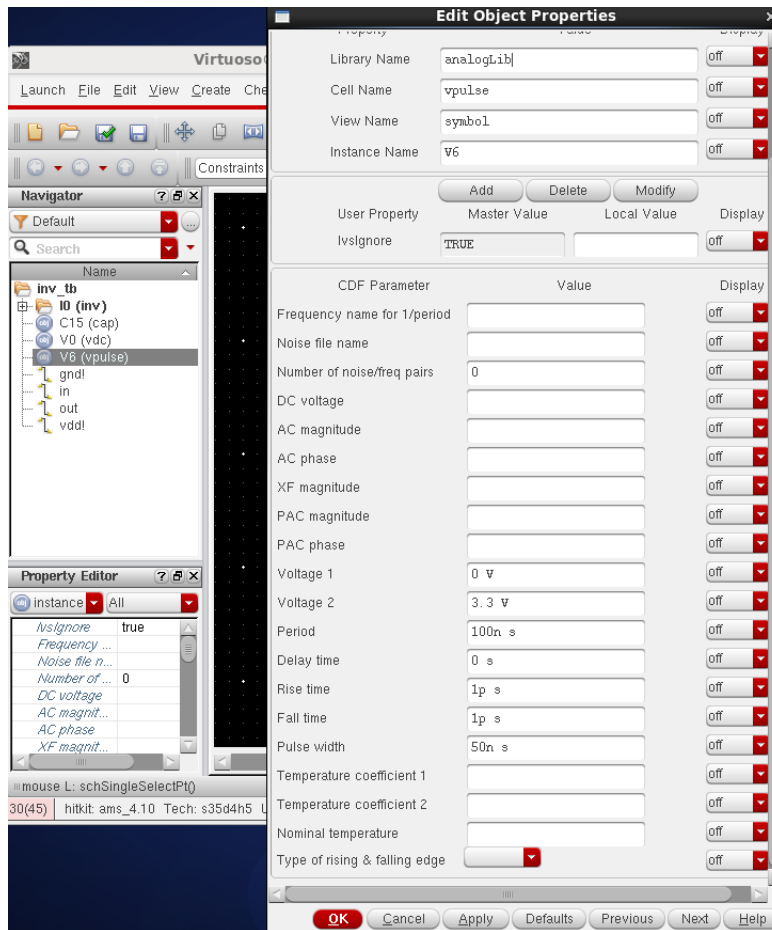
6 Electrical Simulation

Before we can simulate the inverter, we must create *another* schematic. This schematic is what engineers call a *testbench*. Use the Library Manager to create a new schematic in the **LibTest** folder. The name of the testbench should be **my_inv_tb**.

Begin by instantiating your inverter symbol, **my_inv**. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **my_inv** from **Library Name: Lib** and **View Name: symbol**. Or you could also use the shortcut key **i**.

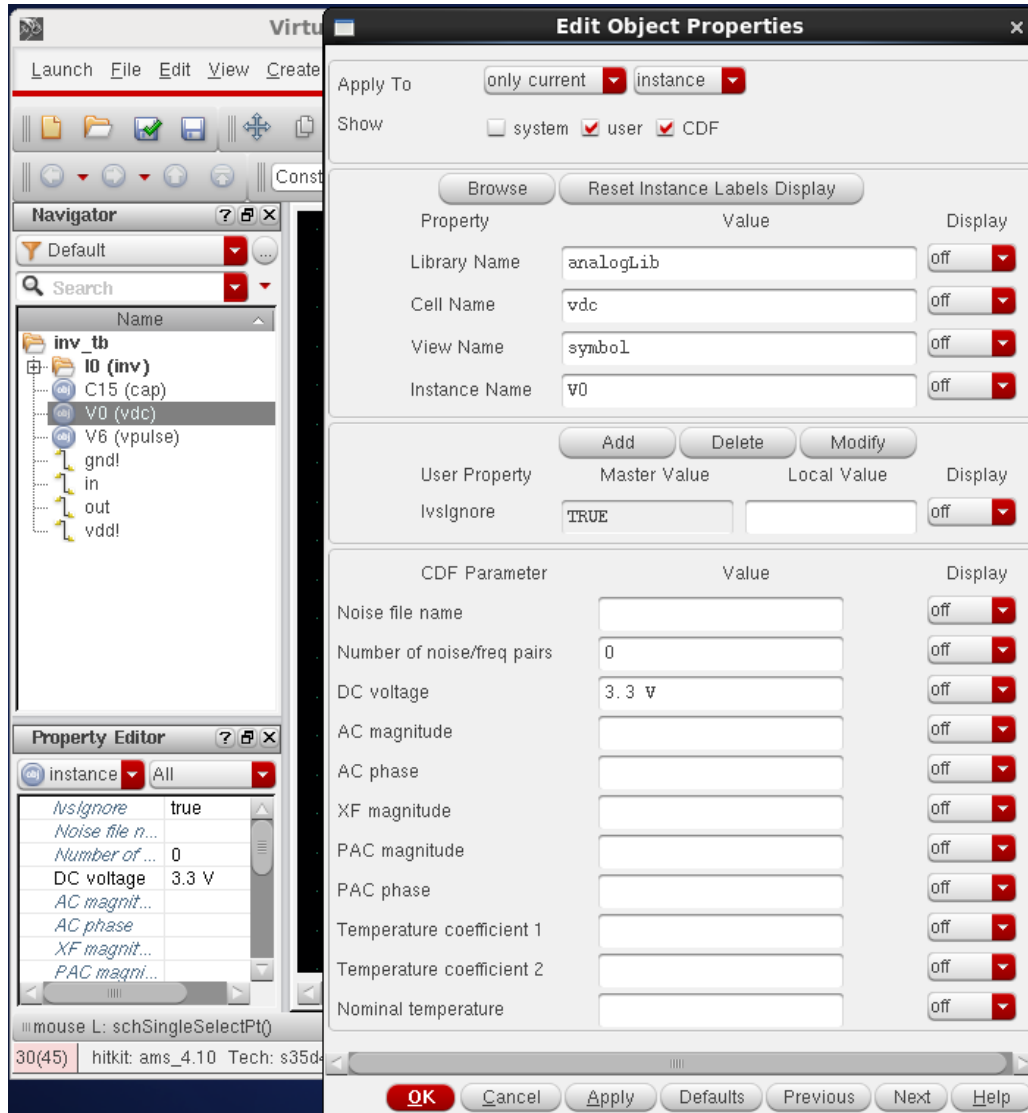
We then need to add a voltage source to drive the input of the inverter. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **vpulse** from **Library Name: analogLib** → **Sources** → **Independent** and **View Name: symbol**.

Fill in the properties form as shown in the figure below. The TA will explain the meaning of the parameters. Place this symbol such that the positive end is connected to the input of the inverter. The negative end should be connected to a **gnd** net which can also be found in the analogLib library.

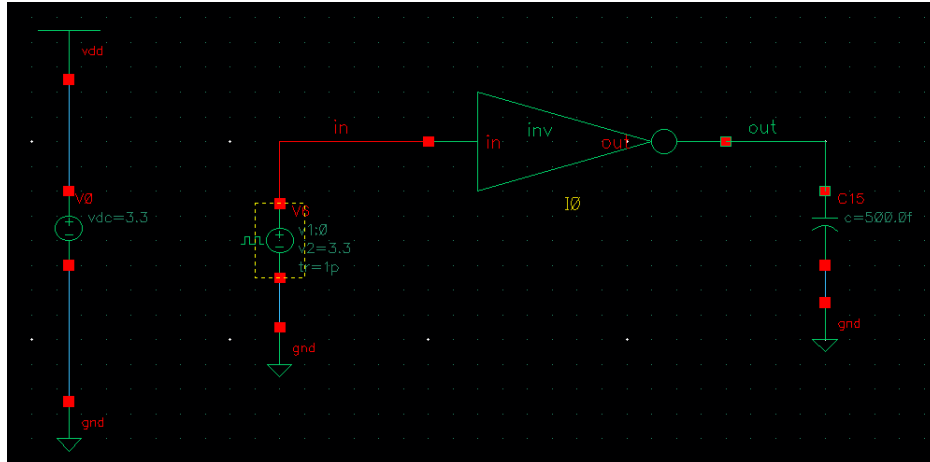


We also will need to add a DC voltage source which we will connect up to the **vdd** net. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **vdc** from **Library**

Name: analogLib and View Name: symbol.

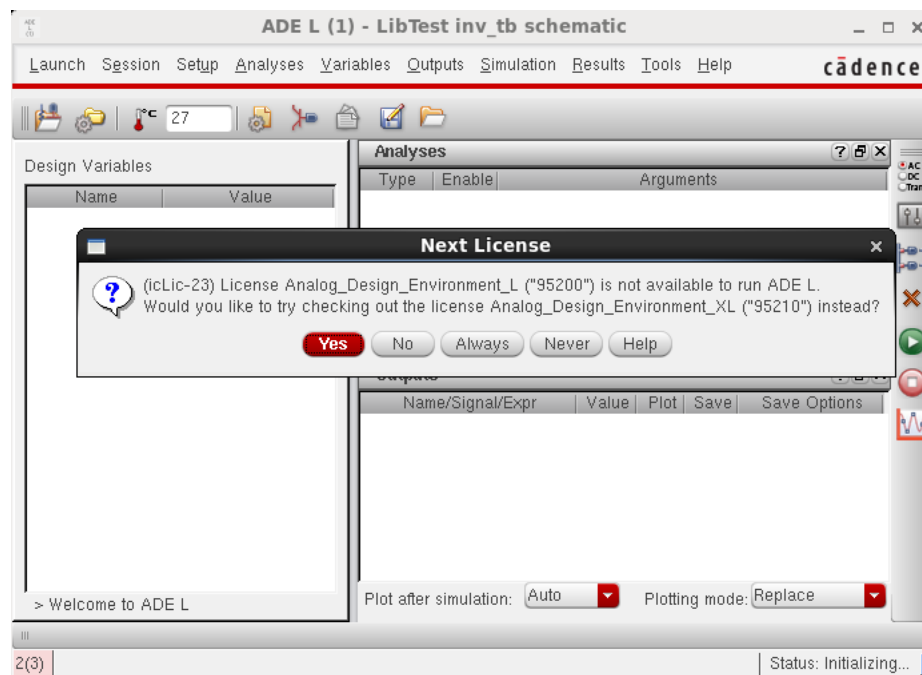


In the same manner as just described, add a capacitor from the inverter output to the **gnd** net. Wire up everything as you did in the previous section of this manual. Also, you will want to use the shortcut **I** to label the input and output nodes. Just click on the node you want to label before pressing the **I** key. When completed your testbench schematic should look like the circuit shown on the next page.



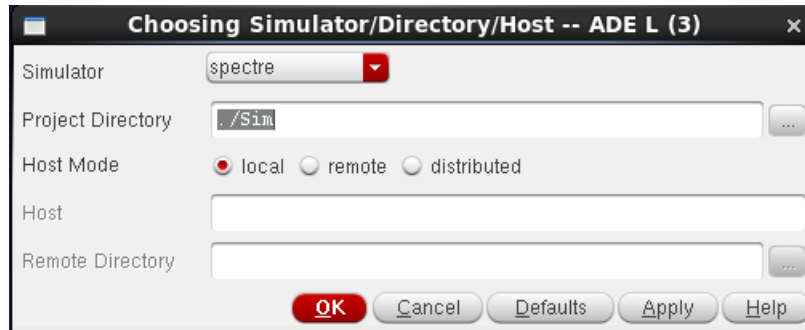
Save the schematic. In the schematic window menu you should: Left Click **Editing:File** → **Save**. We are now ready to run a transient analysis on the inverter. This time you should: Left Click **Editing: Launch** → **ADE L**. Click YES.

You should see something similar to that pictured below.

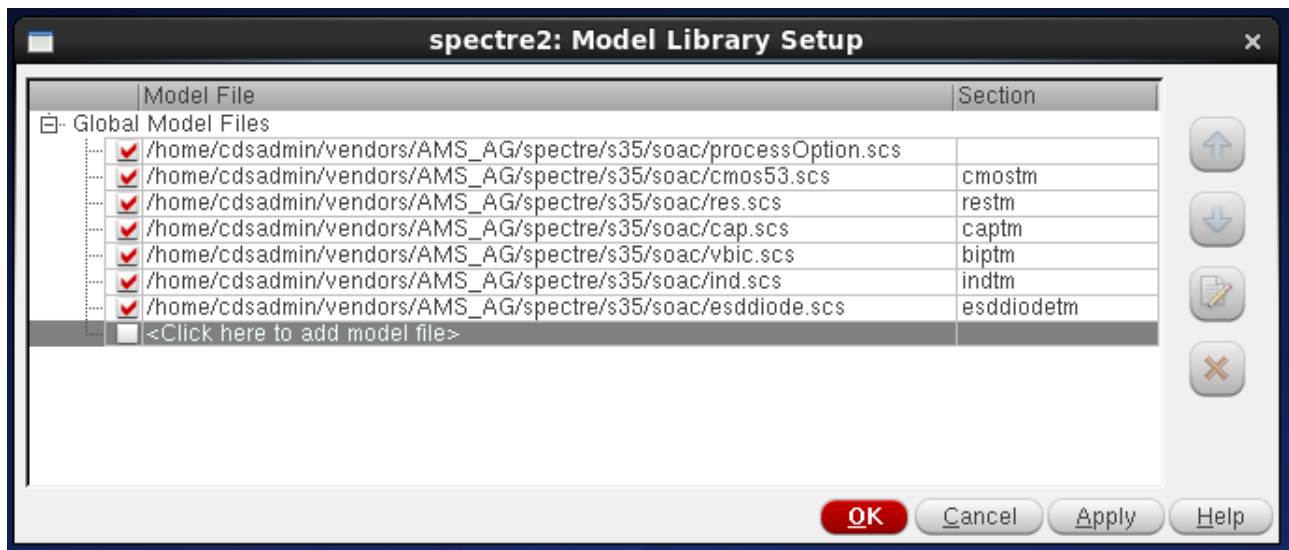


In the Analog Design Environment (ADE) window go to **Setup** → **Simulator** ... You should select the **spectre** simulator. The project directory should be **./Sim**.

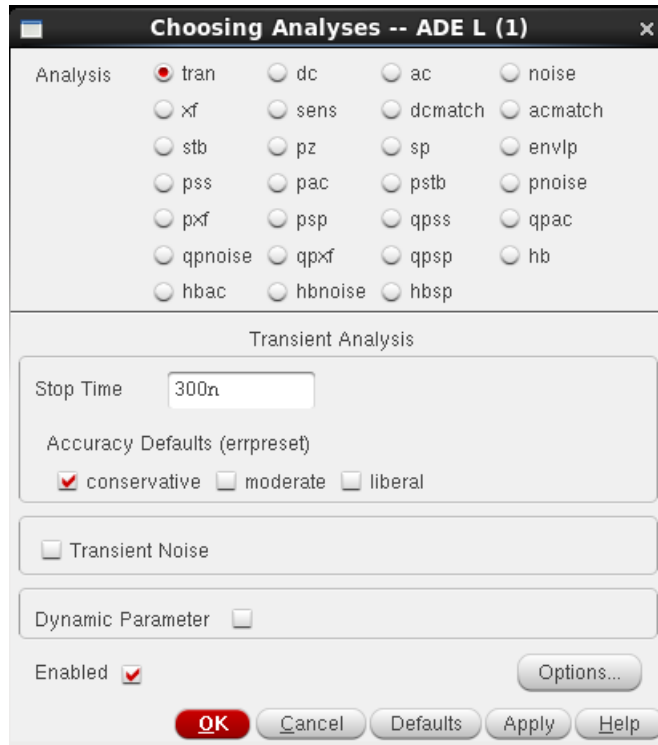
You should see something like what is shown in the figure below.



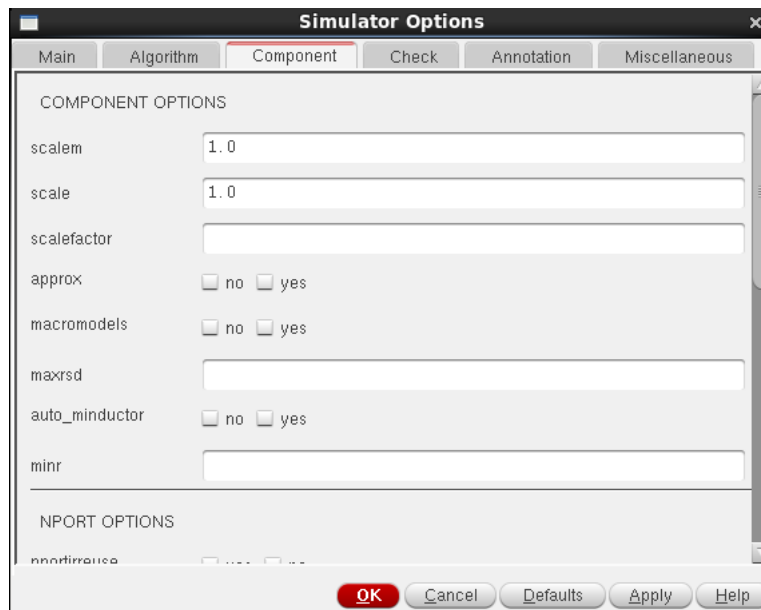
Now we must tell the simulator where it can find the electrical models for the **nmos4** and **pmos4** devices. In the ADE window go to **Setup** → **Model Libraries** ... You should complete the form as shown below.



In the ADE window menu go to **Analysis** → **Choose** ... Set the transient analysis to have Stop Time as 300n. We have specified a transient analysis from 0 to 300ns. Make sure the **conservative** box is checked.

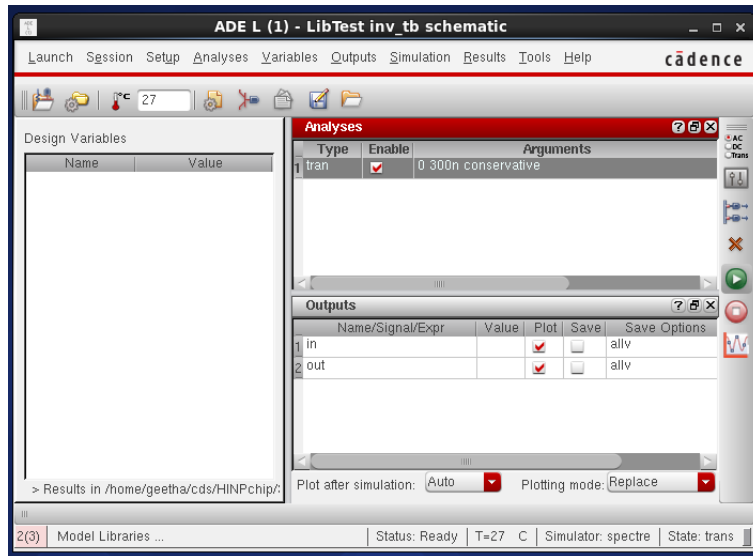


We need to make sure the analog options are correct. You select from the ADE menu **Simulation** → **Options** → **Analog ...**. The following menu with Simulator Options should pop up (Component Tab). You must make sure that the scale is set to 1.0 and *NOT* 1e-6.



Now we need to select some nodes that we would like to display. Left Click **Outputs** → **To be Plotted** → **Select on Schematic**. Select the input and output of the inverter. Use the **ESC** button to exit this mode. The Analog Design Environment window should look

like the one shown below.



Note that the "Plot" buttons are checked for the input and output nodes of the inverter. Run the simulation by pressing on the green traffic light icon. After a few seconds you should see the following report.

```

/home/geetha/cds/HINPchip/Sim/inv_tb/spectre/schema _
File Edit View Help cadence

abstol(I) = 1 pA
temp = 27 C
trnom = 27 C
tempeffects = all
errpreset = conservative
method = gear2only
iteratio = 10
relref = alllocal
cmin = 0 F
gmin = 1 pS

Output and IO/nodeset summary:
      save 2      (current)
      save 5      (voltage)

tran: time = 7.642 ns (2.55 %), step = 273.3 ps (91.1 m%)
tran: time = 25.29 ns (8.43 %), step = 3 ns (1 %)
tran: time = 40.29 ns (13.4 %), step = 3 ns (1 %)
tran: time = 53.62 ns (17.9 %), step = 1.307 ns (436 m%)
tran: time = 67.78 ns (22.6 %), step = 734 ps (245 m%)
tran: time = 83.78 ns (27.9 %), step = 1.645 ns (548 m%)
tran: time = 97.78 ns (32.6 %), step = 2.225 ns (742 m%)
tran: time = 112.6 ns (37.5 %), step = 724.3 ps (241 m%)
tran: time = 128.5 ns (42.8 %), step = 3 ns (1 %)
tran: time = 143.5 ns (47.8 %), step = 3 ns (1 %)
tran: time = 157.8 ns (52.6 %), step = 587.2 ps (196 m%)
tran: time = 172.7 ns (57.6 %), step = 897.5 ps (299 m%)
tran: time = 187.6 ns (62.5 %), step = 2.03 ns (677 m%)
tran: time = 202.5 ns (67.5 %), step = 330.6 ps (110 m%)
tran: time = 219.5 ns (73.2 %), step = 2.199 ns (733 m%)
tran: time = 234.5 ns (78.2 %), step = 3 ns (1 %)
tran: time = 248.3 ns (82.8 %), step = 1.74 ns (580 m%)
tran: time = 263.3 ns (87.8 %), step = 1.126 ns (375 m%)
tran: time = 278 ns (92.7 %), step = 1.19 ns (397 m%)
tran: time = 292.6 ns (97.5 %), step = 2.631 ns (877 m%)

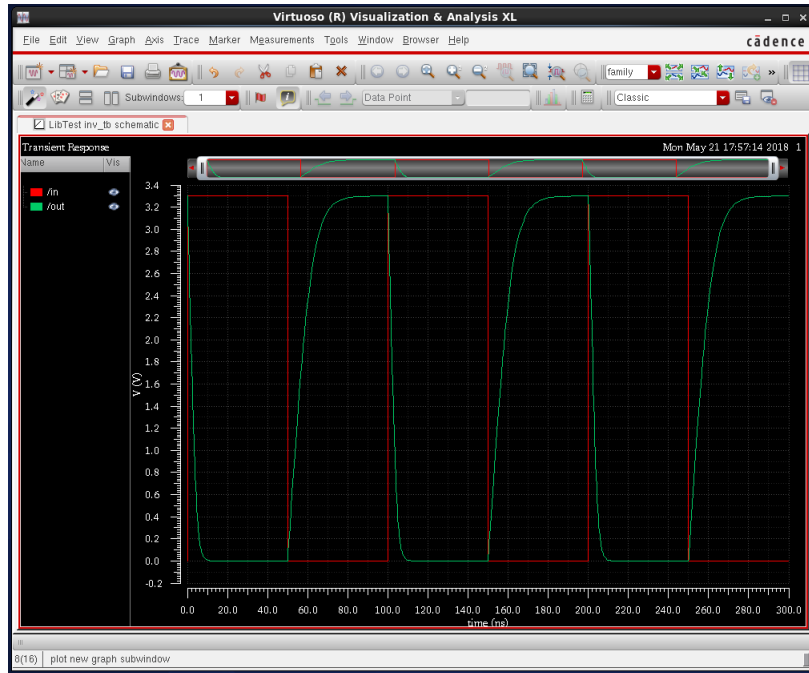
Number of accepted tran steps = 1071

*****
Post-Transient Simulation Summary
*****
- To further speed up simulation, consider
  add ++aps on command line
- Non-default settings that could significantly slow down simulat:
  errpreset = conservative, default moderate
  reltol = 10e-06, default 100e-06 (conservative)
*****

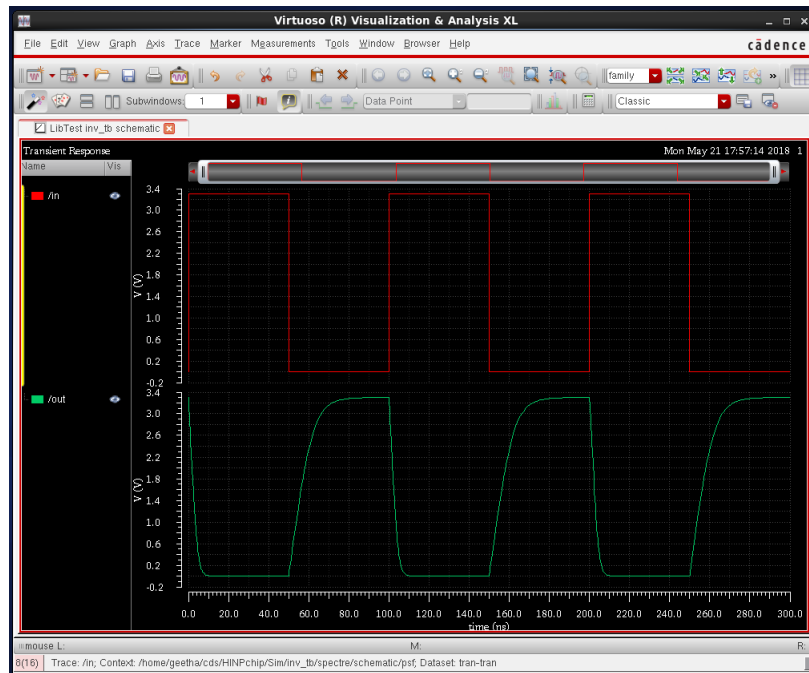
18 | Trace: /out; Context: /home/geetha/cds/HINPchip/Sim/inv_tb/spectre | L1 | C1

```

You will also see the wave window appear as shown on the next page.



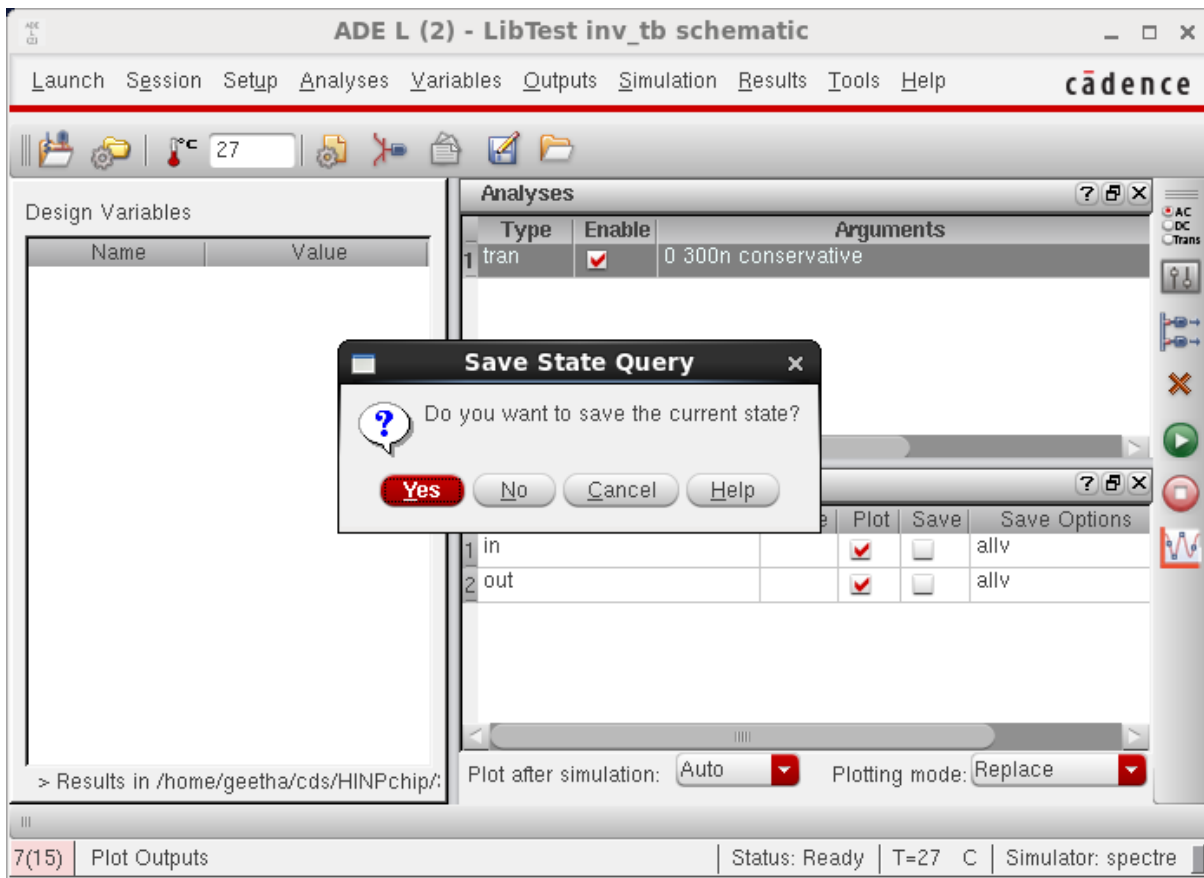
Left click **Axes** → **Strip** in order to separate the different curves. The result should look like the one on the next page.



Notice that the output node voltage is delayed from the input and that the output rise time is longer than the output fall time. The TA can show you how to actually make measure-

ments on the waveform.

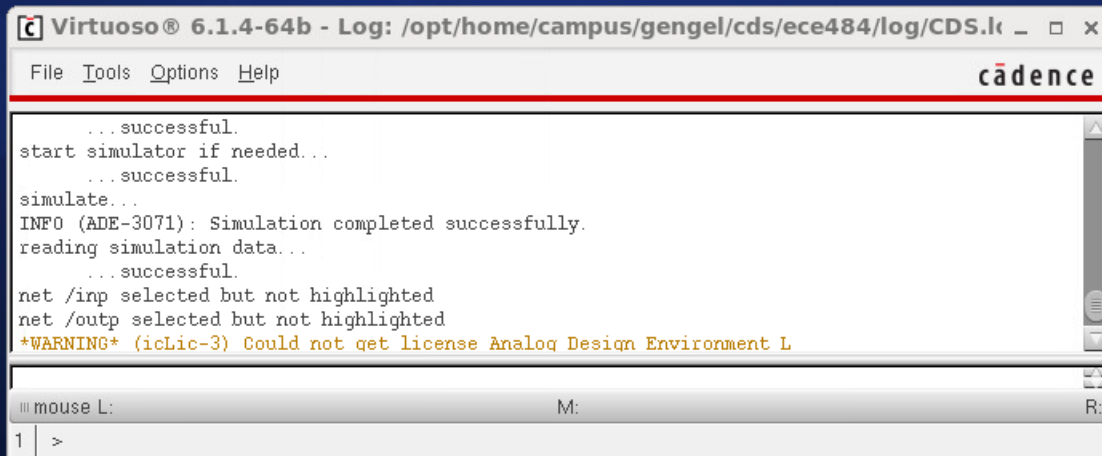
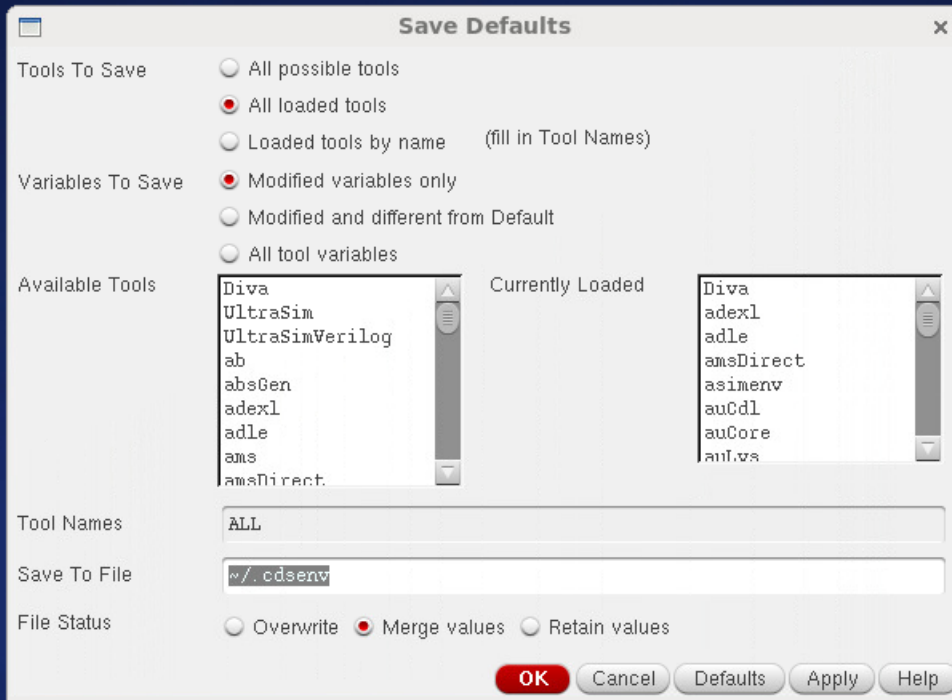
Now go around and close up all of the windows. Left click on the "X" in the upper right most corner of the window. When you get to the ADE window, it will ask you if you wish to save the state. Respond by hitting the YES button. If you save the state then when you want to re-run the simulation all you have to do is to load the state. You will not have to enter the simulator, the model files, etc.



You should see window pictured below pop up. Enter a state name or take the default "state1" name and then click OK. I called my state "tran".



Before you close up the CIW (Command Interface Window), you should left click **Options** → **Save Defaults**. When the window pictured below pops up just click **OK**. This is something you only have to do once. From now on the simulator type should default to "Spectre" and the scale should always be set to 1.0 but it never hurts to check to make sure that this is the case! Now go ahead and close up the CIW. Congratulations, you have successfully simulated your first IC design.



7 Physical Layout, DRC, and LVS

In this section of the manual we will describe how to physically lay out a cell, how to check to make sure that the layout conforms to all of the manufacturer's rules (DRC), and finally how to make sure that the layout yields a netlist which is equivalent to the netlist produced by the schematic editor (LVS).

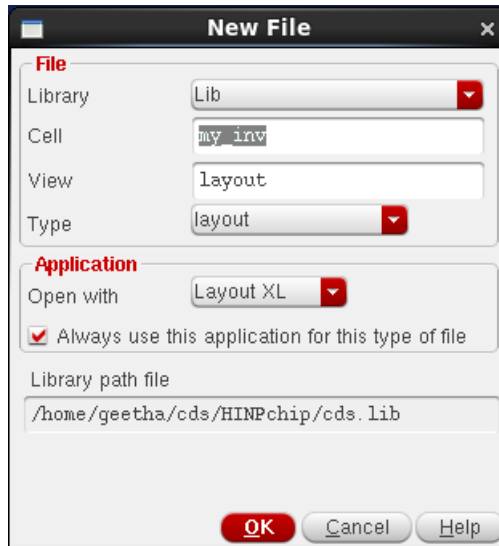
7.1 Layout

By now, you should know how to enter a schematic, how to produce a corresponding symbol for the cell, and finally how to simulate your design to make sure it functions correctly. The next step in the process of making an integrated circuit (IC) is to perform the physical layout of the cell. What is a layout? A layout is basically a drawing of the masks from which your design will be fabricated. Therefore, layout is as critical as specifying the parameters of your devices because it determines whether you will have a working design or a flop!

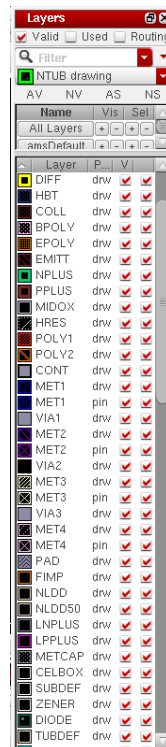
There are two approaches to doing layout: *manual* and automated. Manual layout usually enables the designer to pack one's devices in a smaller area compared to the automated process but it is much more tedious. The automated process, on the other hand, is done using standard cells and usually takes more real estate, but it is much faster. In this tutorial, you will learn how to perform **MANUAL LAYOUTS ONLY** and, specifically, the layout of a simple inverter will be described. You will then be asked to lay out other cells for which you have already completed the schematic entry process. You should know that for the purposes of this course, you are primarily required to know how to create manual layouts, although the Cadence tools can support either manual or automated layouts. In fact, later in the semester we *will* experiment with the standard cell approach to layout.

Before we start the layout of our inverter, one needs to understand the importance of design rules. Design rules give designers guidelines for generating layouts. They dictate the spacings between wells, sizes of contacts, minimum spacing between a poly and a metal layer, and many other similar things. Design rules are essential to any successful layout, since they account for the various allowances that need to be given during actual fabrication and to account for the sizes and the steps involved in generating masks for the final layout. Note that layout is very much process-dependent since every process has a different fixed number of available masks for layout and fabrication. In this tutorial, we will be using the AMS 0.35 micron CMOS process, which is a N WELL process. It supports 2 poly and 4 metal layers.

We will now create a layout for the **my_inv** cell we created a schematic for and then later simulated. You should launch the Library Manager just like you did in early sections of this tutorial. In the Library Manager window, select the **my_inv** cell in **Lib**. You should then left click on **File** → **New** → **CellView**. A "New File" menu will pop-up. You should set the View filed to **layout** and the the Application to open with should be **Layout XL**. Click the "Always use this application for this type of file: button. The form should be completed as shown in the figure on the next page.



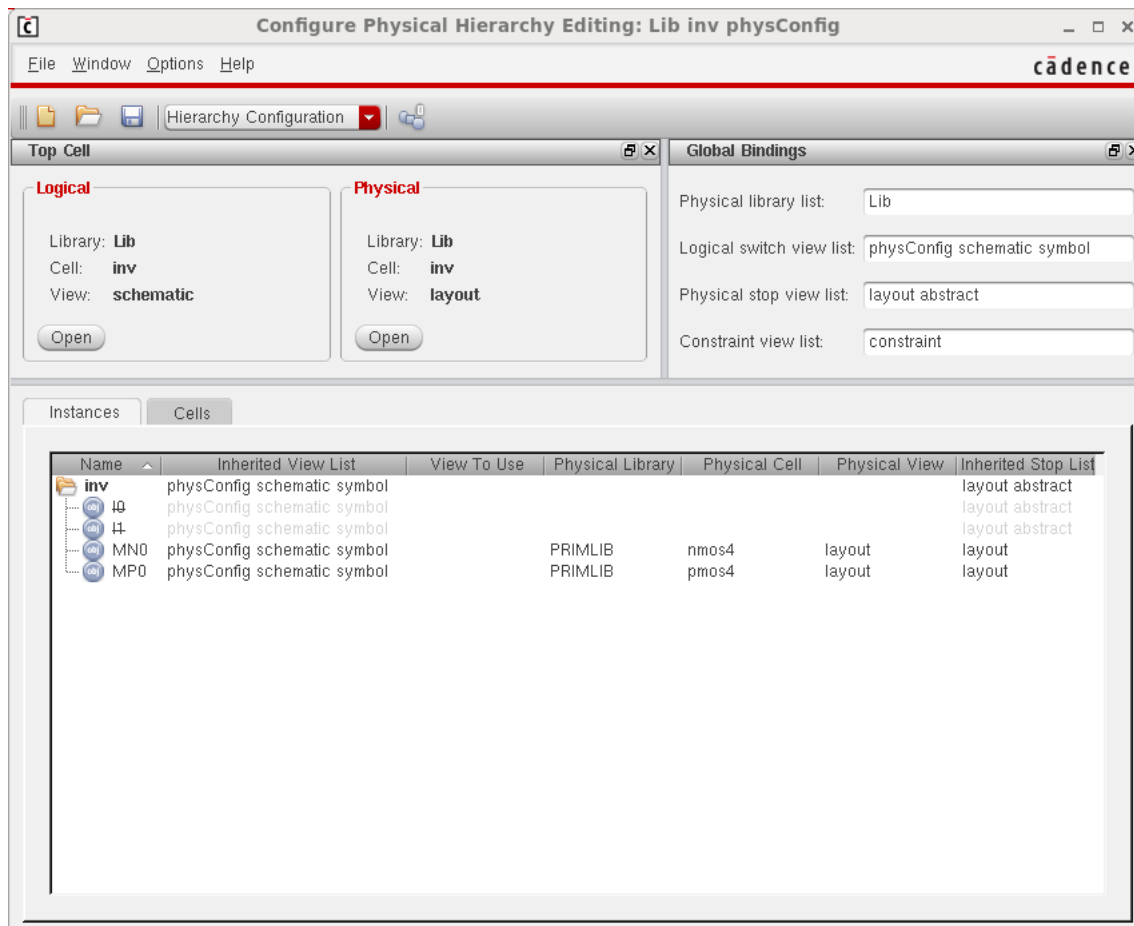
Then click on the **OK** button. An empty layout editor window and the schematic editor window will pop-up along with a Layers window. In fact, the schematic window may cover up the Layers window. You may need to bring the Layers window to the forefront. If you want minimize the schematic window. The Layers window will show all the layers such as nwell, pwell, active, etc. for the given process. If the Layers window is blank, then there is an error! The Layers window should look something like the pallette shown below:



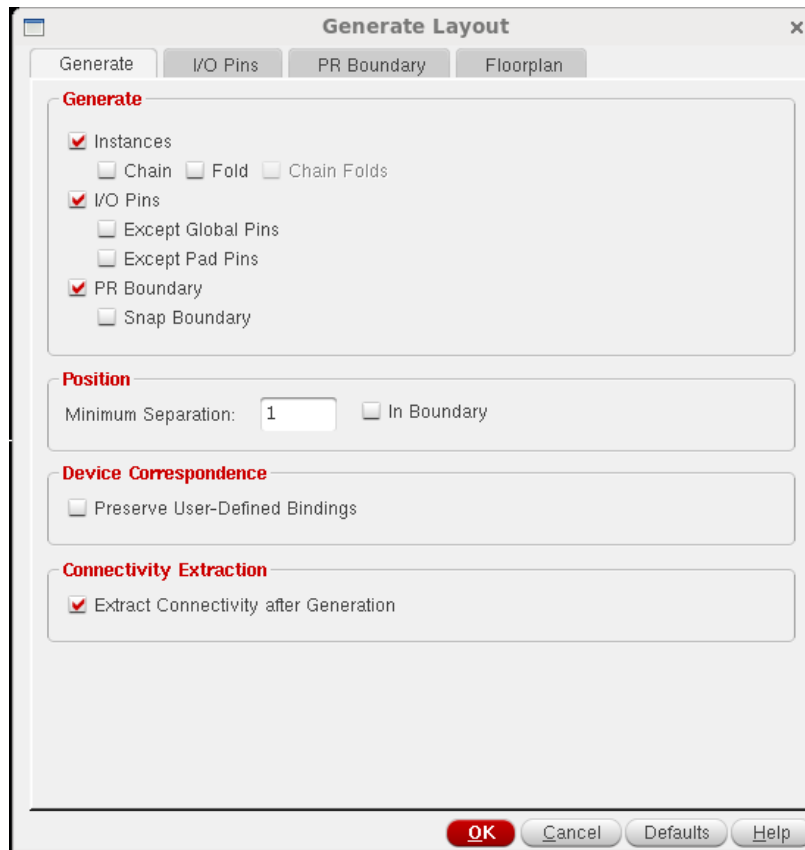
You should now go to the Virtuoso Layout Editor Window (NOT the Virtuoso Schematic Window!) and from the main menu at the top select **Launch** → **Configure Physical Hierarchy** We will use prepared scripts (called PCELL for "parameterizable cell" generators)

to automatically draw our transistors but first we must tell the tool which script to use for each of the devices which appear in our schematic. We need to fill in the form correctly as described below.

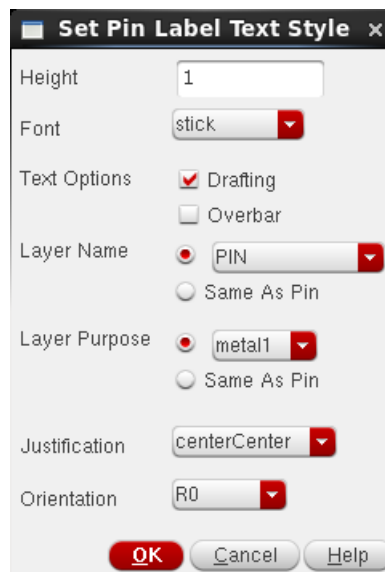
You need to select **PRIMLIB** as the Physical Library for both P0 and for N0. The TA will help you do this. For MP0 the physical cell should be **pmos4** and for MN0 the physical cell should be **nmos4**. Your form should look like as the one that is shown below. Save the information by clicking on the **Disk** symbol or by going to the File menu item and selecting Save. Then close the "Configure Physical Hierarchy Editor Window"



Now that the tool knows which PCELL generators to use for the FETS, we can have the tool use the schematic to draw all of the FETS and to place them into the layout window. It will also place the ports for us. In the Virtuoso Layout Window (NOT the Virtuoso Schematic Window) from the main menu, you should select **Connectivity** → **Generate** → **All From Source ...**. The form should be filled out like the one shown below.

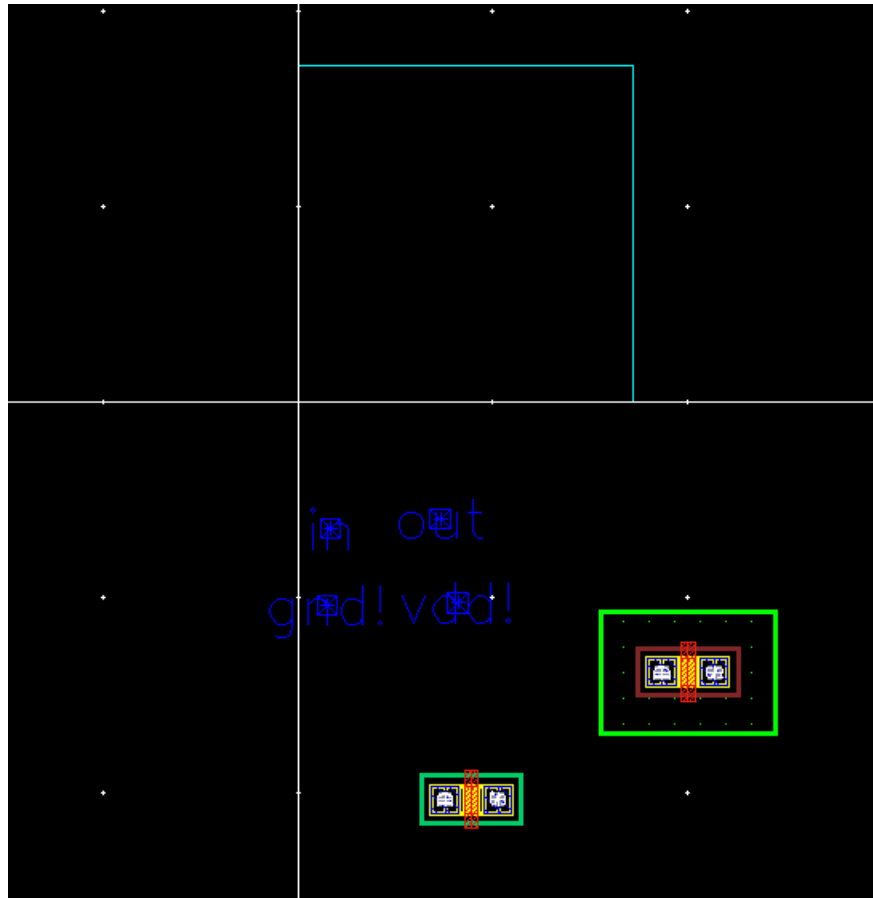


And now switch tab to **I/O Pins** then under **Pin Label** select radio button called **Label** click on **Options**. In the form shown below, make sure to select the Layer Name:**PIN** and Layer Purpose:**metal1**. Then click **OK**

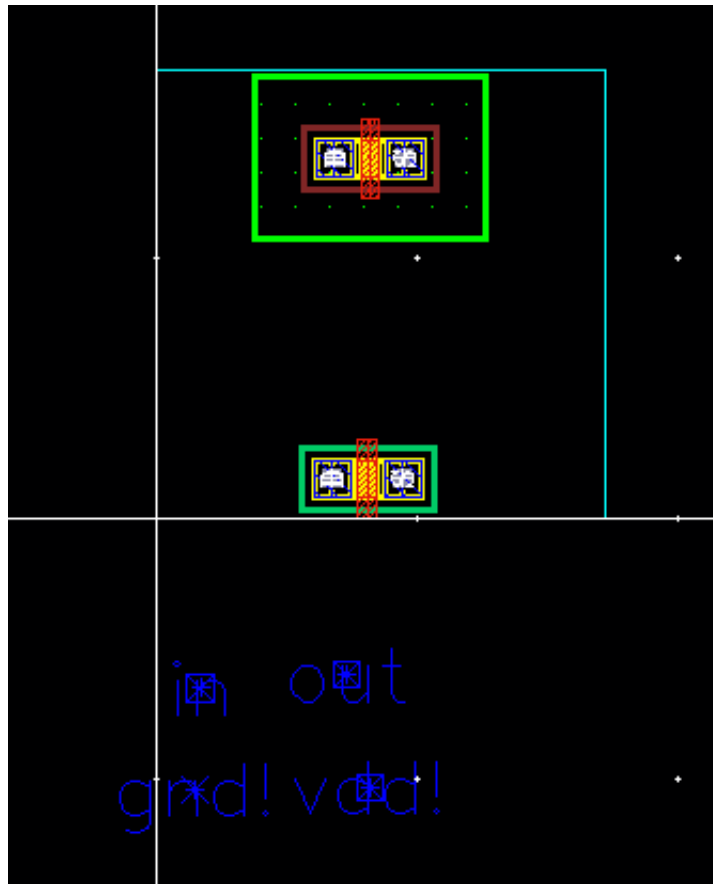


After clicking on the **OK** button, you should see the layout for the NFET and PFET (the one with the big green box!) appear in the layout window. You will also see 4 "blue" squares

(metal-1 i.e. M1). These are the 4 ports: **vdd!**, **gnd!**, **in**, and **out**.



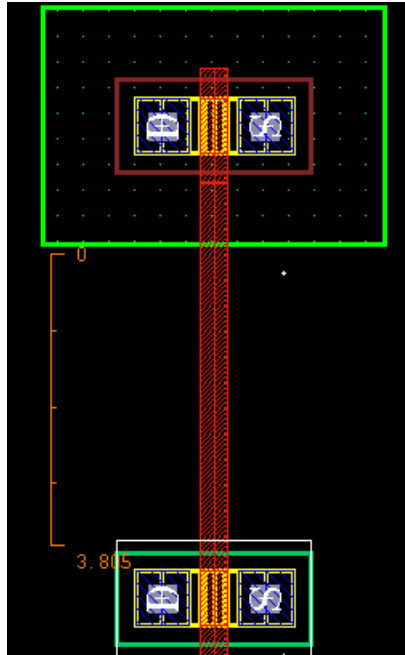
Select the PFET and use the "move" command (short-cut key **m**) to move it to the top of the bounding box. Notice that you can only move in one direction. Place the FET and then repeat the move command; this time moving in the opposite direction. Do the same for the NFET (but place it at the bottom of the bounding box). Your layout should be similar to the one shown on the next page.



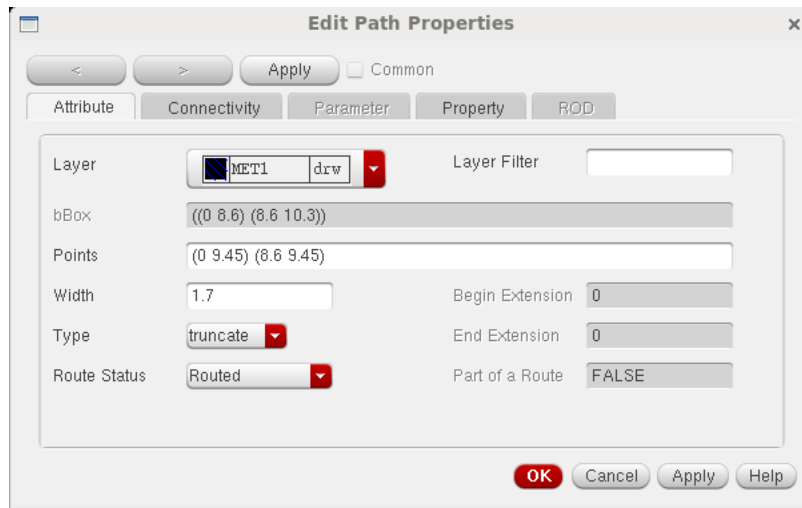
We now need to connect the two poly gates. We will do this by using the "path" command (short-cut key **p**) but before you press the **p** key, we need to tell the tool that we want to route using the "poly1" layer. We do this by selecting *poly1* layer in the Layers palette. We then position our cursor on the top of the NFET poly stub. One then hits the **p** key.

You can then route the poly from the NFET to the poly stub on the PFET. To complete the route, you should *double left-click*. If you make a mistake you can undo by using the short-cut key **u**. Your layout should look similar to what is shown on the next page. You can cancel the "Create Path" menu. Also, I recommend frequent saving of your layout by pressing the "Disk" icon in the Layout Window menu bar.

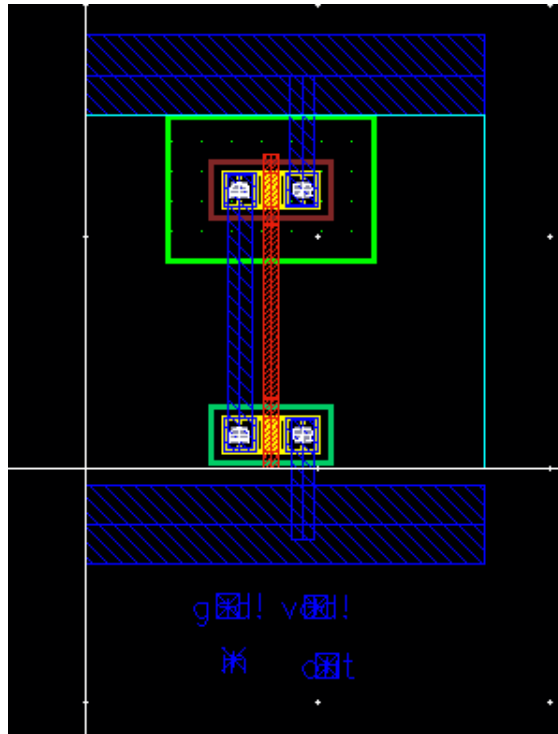
Note: one can measure distances by using the "Ruler" command. Click in the Layout Window where you want to start measuring and press the shortcut key **k**. Drag the ruler and then double left click to terminate the ruler command. Rulers can be removed by using the shortcut-key **K**.



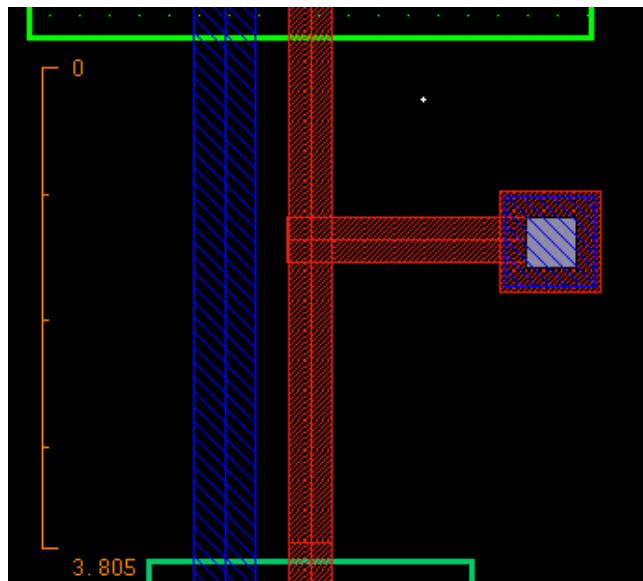
We will now connect the drains of the FETs together using metal1 (M1 for short). Tap the metal1 tag in the LSW palette. Then go to the Layout Window and use the path command to connect up the drains. Also put in the two power supply rails using M1. Select the "gnd" rail and hit the shortcut-key **q**. This is the "query" command. Change the **Width** parameter to 1.7 and hit the **OK** button. The following menu shown below should pop up.



Repeat for the "vdd" rail. Move the rails using the shortcut-key **m**. Also, hook up the source terminals of the two FETs using M1. Don't forget to "Save"! Your layout should be similar to the one shown below.

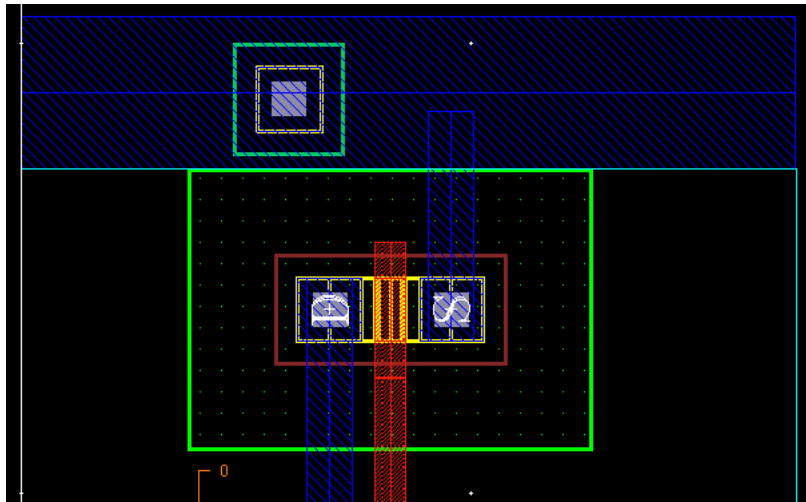


Poly1 is not a layer which one generally likes to route in so we will want to make the gates of the FETs accessible through the M1 layer. We will do this by placing a M1-poly1 say **P1_C** contact. Go to the Layout Editor menu bar and select **Create** *rightarrow* **Via**. Change the "Via Definition" from **ND_C** to **P1_C**. Place the contact as shown below and hit the **ESC** key.

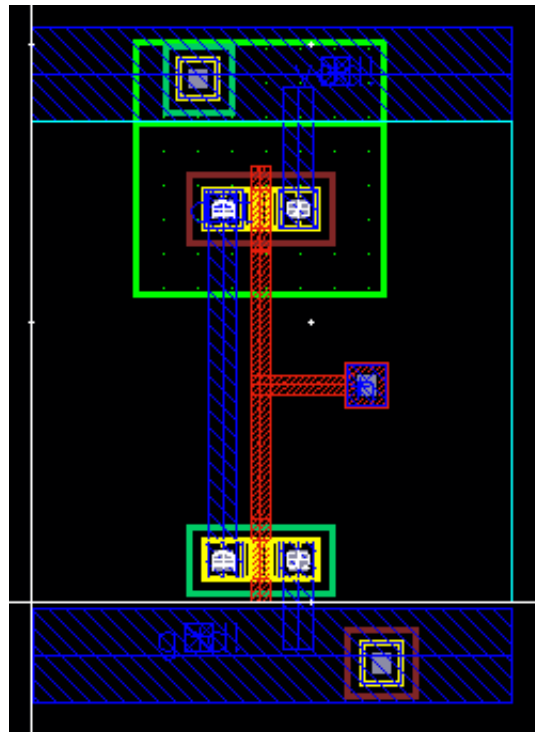


We are almost finished, but we still need to add substrate contacts. You will need to use the "Create Via" command just like you did for the poly contact but this time we will want

to use the **ND_C** via definition. We will want to place the **ND_C** onto the **vdd** rail as is shown below.



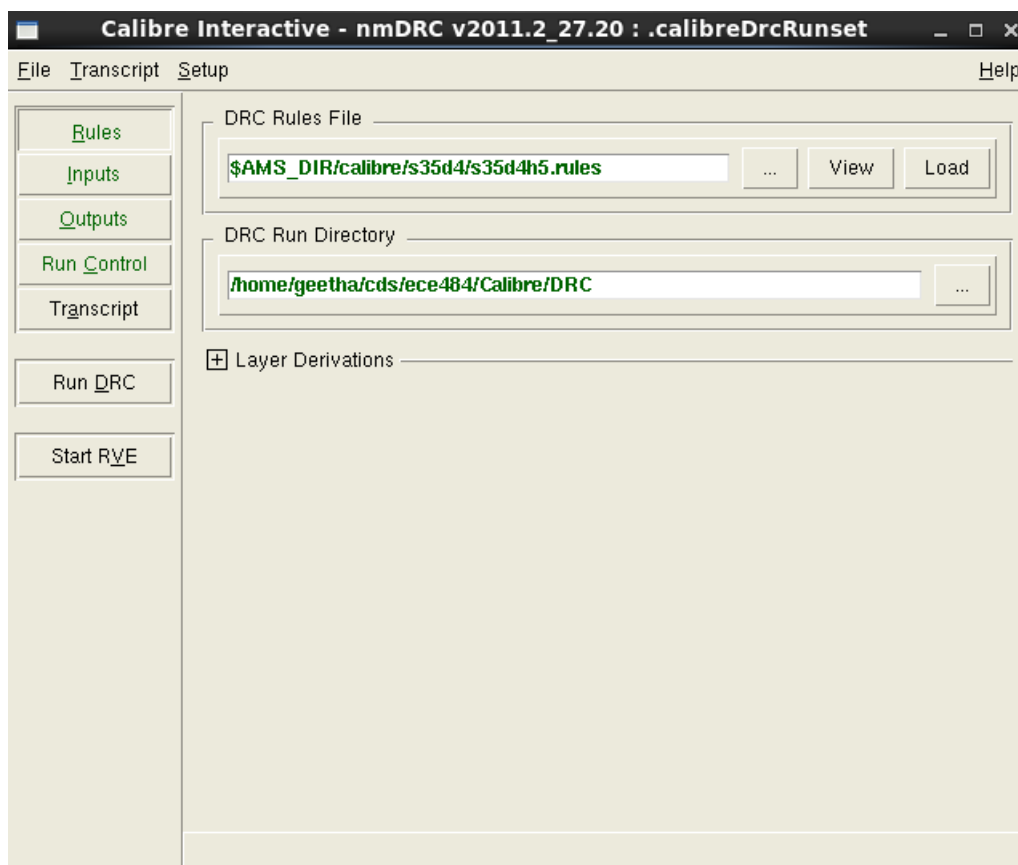
We also need to place a substrate contact on the **gnd** rail. This time when you use the "Create Via" command, choose the **PD_C** via definition. Then add a **NWELL** by selecting **NTUB drawing** in the Layers as shown below in the vdd railing using **r** shortcut-key. Finally move the ports (blue boxes at the bottom of the layout onto the appropriate nets in the layout. You can identify the port by selecting it and then using the **q** shortcut-key. The finished layout should be similar to what is shown below. Don't forget to "Save" your layout



7.2 Design Rule Check (DRC)

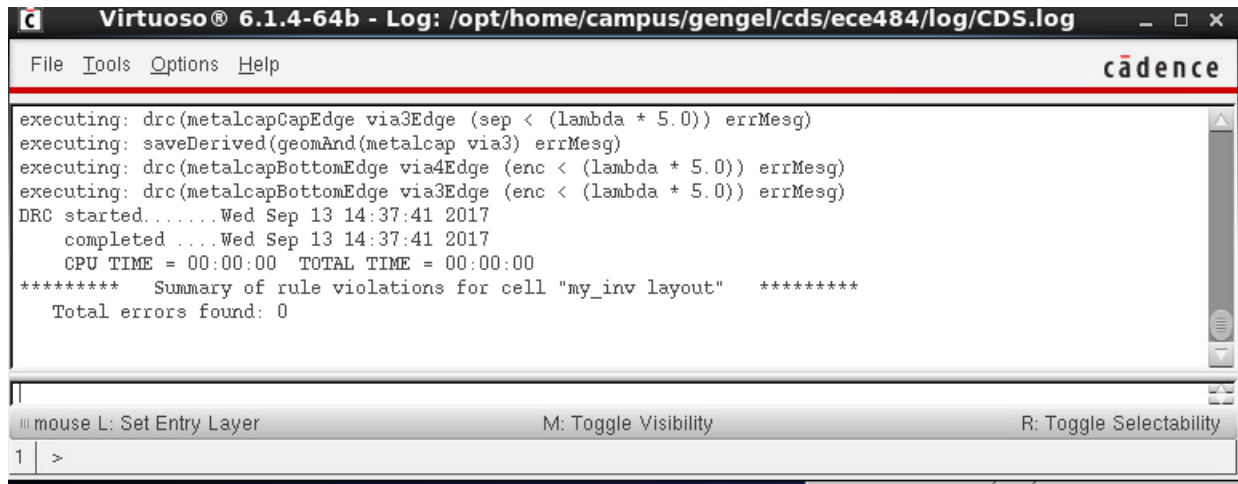
Our next step in the design process is to perform a Design Rule Check, more commonly known as DRC, on the layout. Although designers might be conscious of the design rules when performing the layout, there is a possibility of overlooking and thus violating the design rules. So, the DRC is a step taken to alert us to any violations. This step is important because the violation of any design rule would result in a higher probability, and in some cases an absolute certainty, that the fabricated chip does not work as desired.

To run the DRC, choose Run DRC... from the Calibre menu in the layout view window. A pop-up menu, similar to one shown below will appear. In the below form select **Rules** then change the DRC Run Directory as `/home/X/cds/ece484/Calibre/DRC` where X is your **Username**



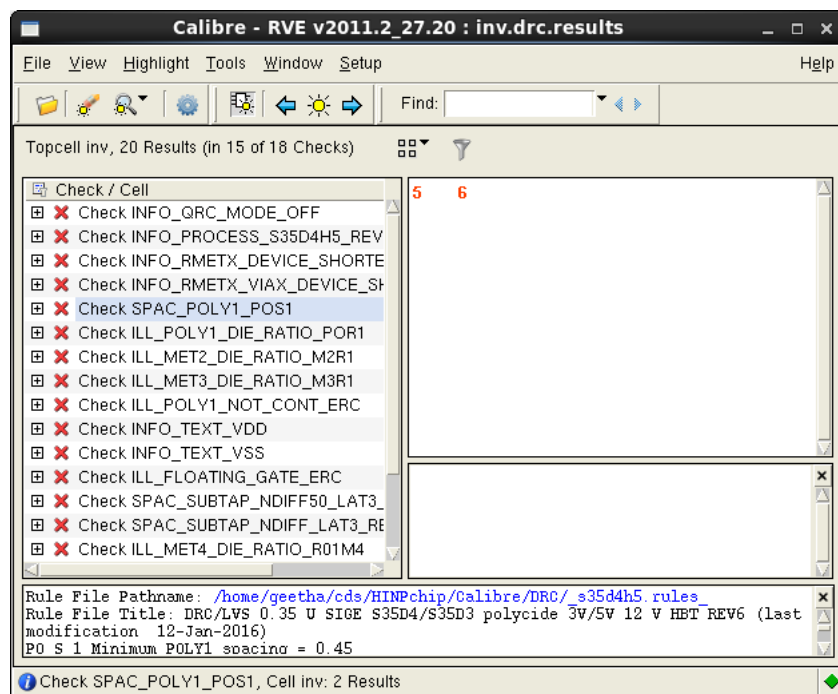
You need to make sure that you're in edit mode for your layout that you want to run DRC on. For huge layouts, DRC might take a bit of time to perform. You can shorten this time by deselecting the Echo Commands option. Click on the **OK** button.

Cadence then runs the DRC and reports the errors or warnings, if any, in the CIW window.

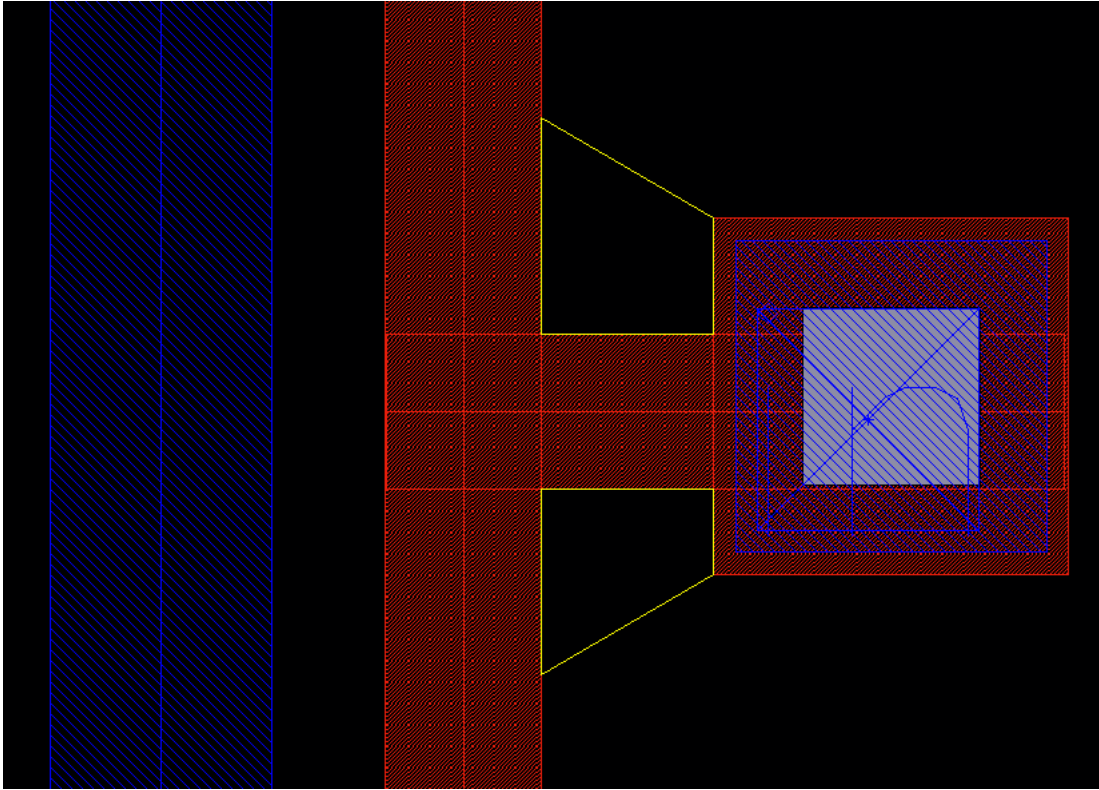


The CIW window above shows that there are no errors or warnings found in the DRC process.

Let's perform a DRC on a layout that has errors. For the purposes of this tutorial, this step is strongly recommended so that you can gain some experience in dealing with layout DRC violations. You can purposely violate a design rule by stretching any of the mask layers to some ridiculously large dimension or by moving/shifting some layer as shown below. In this case I moved the poly contact to the right so that the spacing requirement between poly1 wires was VIOLATED. You will see the spacing error between poly 1 layers as shown in the below figures.



Right click on **Check SPAC_POLY1_POS1** error and click on highlight, this highlights the error in the layout.

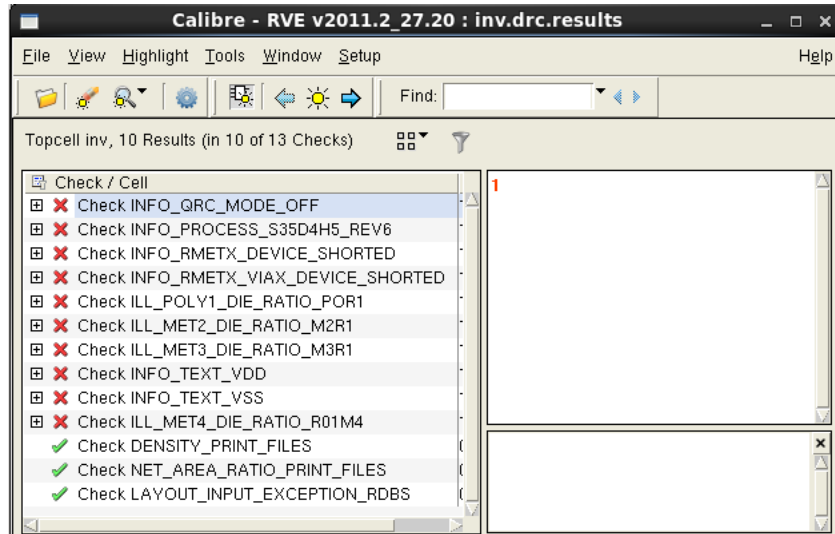


The layout above leads to the results of the DRC. Errors are indicated by markers (white as shown above) but in your layout, these markers will blink. The errors are also reported in the CIW as shown below. You may then proceed to correct the errors according to the design rules.

When performing huge layouts, the blinking markers might not be easily located at times. Fortunately, Cadence has an easy search tool. Under the Verify menu in the layout window, choose Markers → Find...

A pop-up menu will appear. Click on the Zoom to Markers box.

Click on the **Apply** button and Cadence will zoom in to the errors or warnings as desired. If there are more than one errors/warnings, as is almost always the case, you can view each one of them by clicking on **Next**.

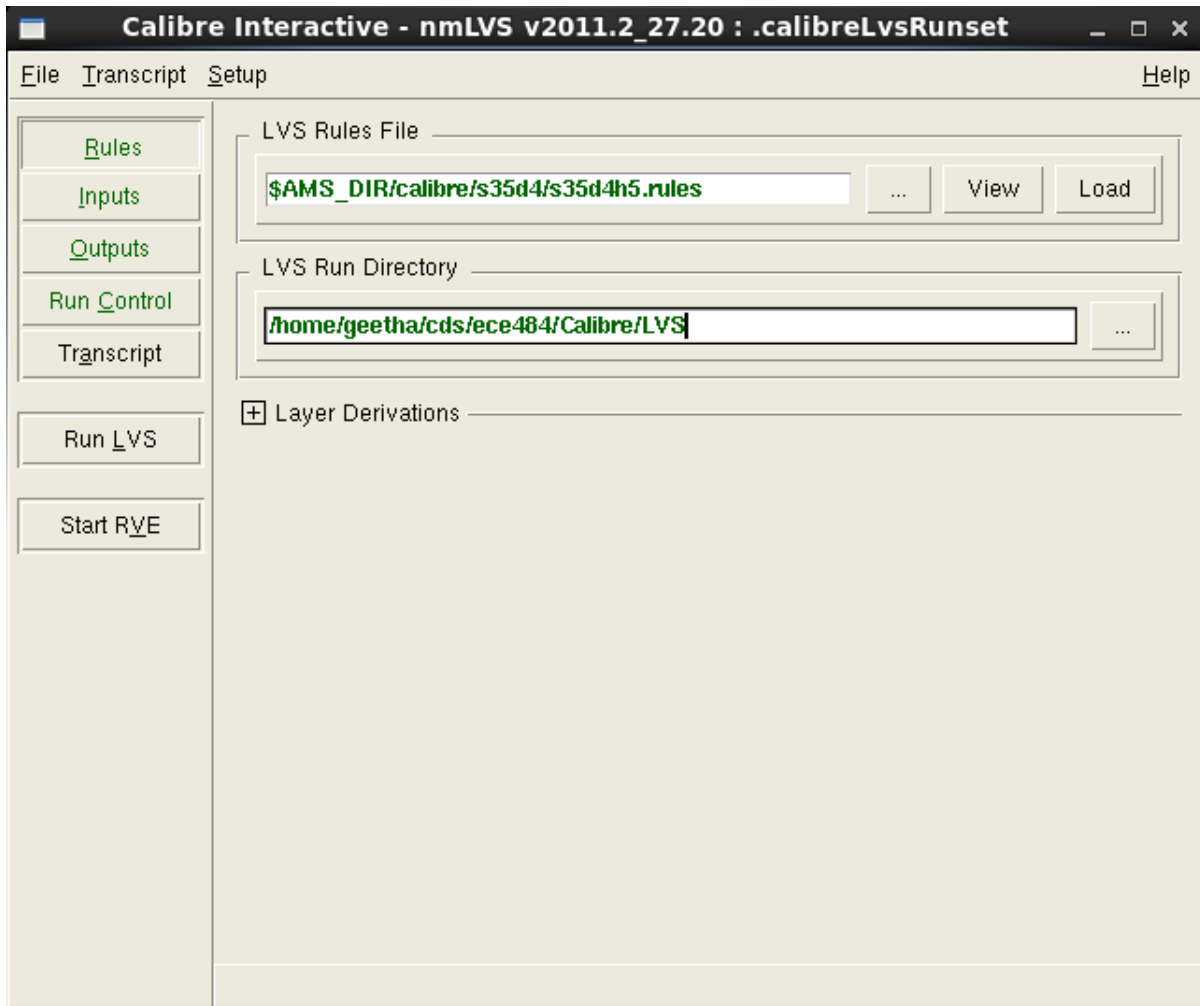


The Calibre DRC results window above shows the list of benign drc errors which can be ignored if noticed during the DRC check.

7.3 Layout Versus Schematic (LVS) Check

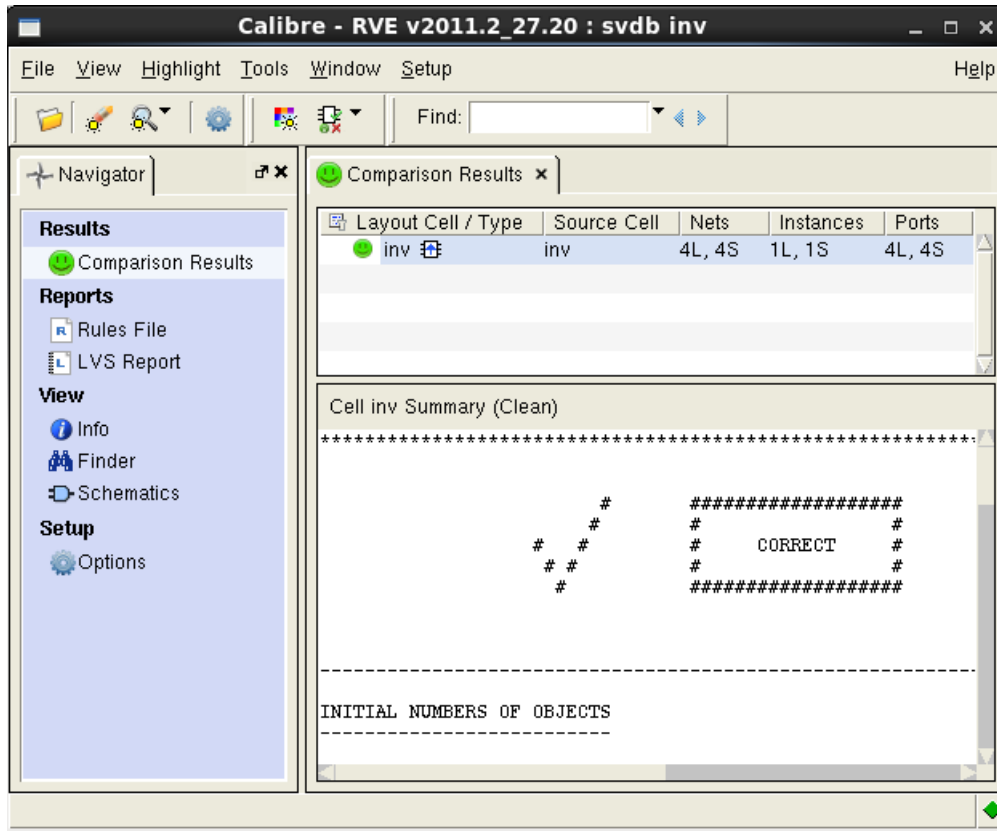
The next step is to perform LVS. Since we generated a layout with certain a W and L for the transistors (for the case discussed here, I had the NFET $W = 0.7u$ and $L = 0.35u$ and the PFET $W = 0.7u$ and $L = 0.35u$), the layout versus schematic operation (discussed below) will give you an error if the schematic against which the layout is compared has a different W and L for its pmos4 and nmos4 transistors. So, make sure that the nmos4 and pmos4 have correct entries in their properties field for W and L.

Now from the layout window, choose Run LVS... under the Calibre menu. A pop-up menu will appear, similar to one shown below. Select **Rules** button in the below form then change the LVS Run Directory as `/home/X/cds/ece484/Calibre/LVS` where X is your **Username**

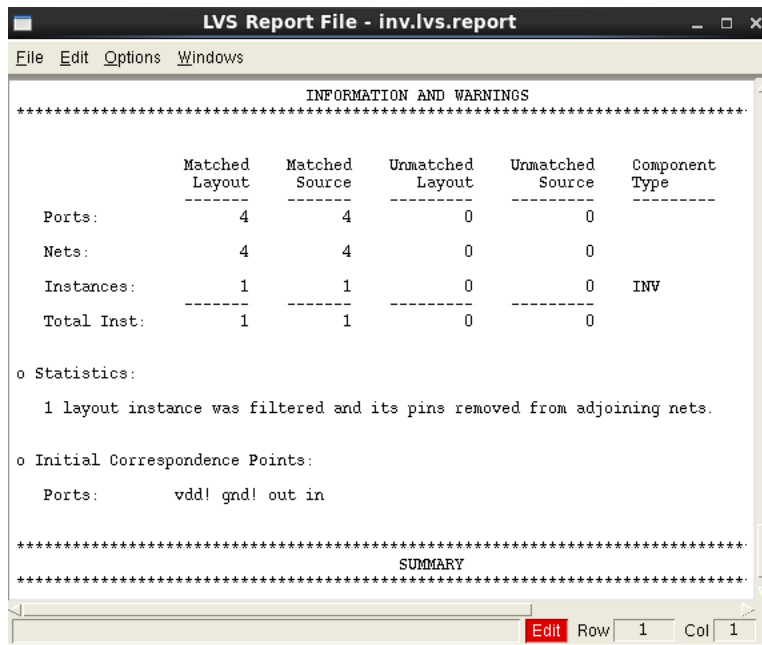


Click on the Run LVS button and wait. It may take a few seconds. Be patient!

A pop-up menu will then appear notifying you of the successful completion or failure of the LVS job. If successfully completed the job, you will see the following dialog box with a smiling face.

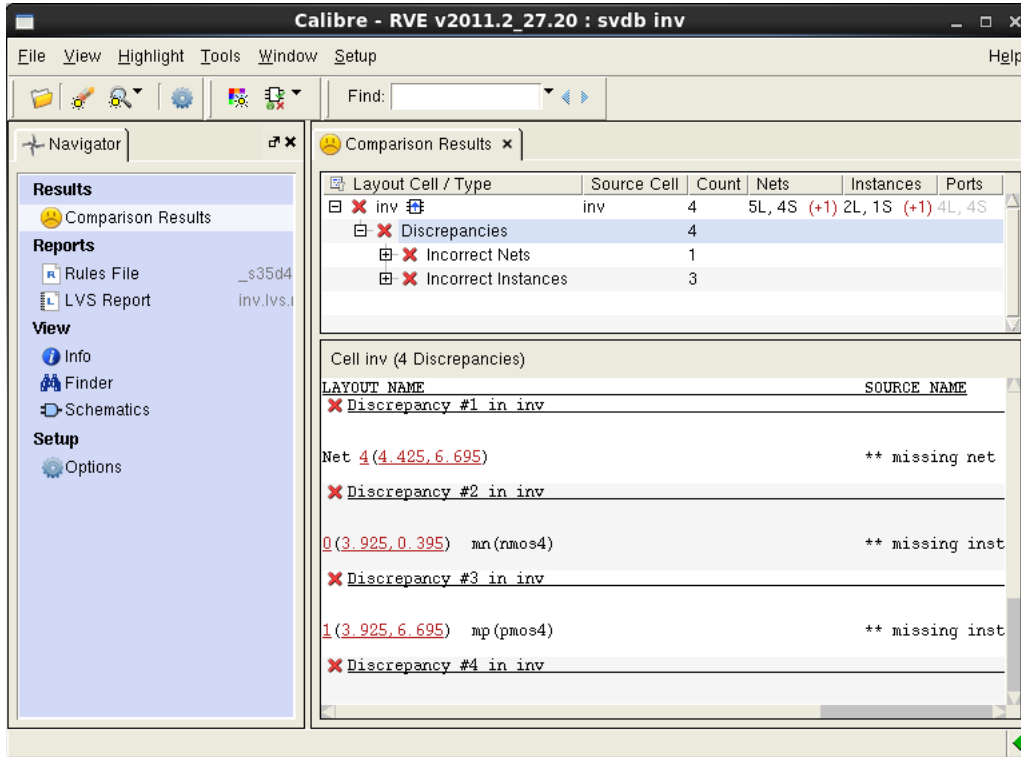


You should see the following message in the LVS report file as shown below:



We see from the dialog box that there are no errors in the LVS comparison. However, there could have been errors if, for example, the W and L values of the transistors in the schematic window did not match with the W and L values of the transistors in the layout. If there

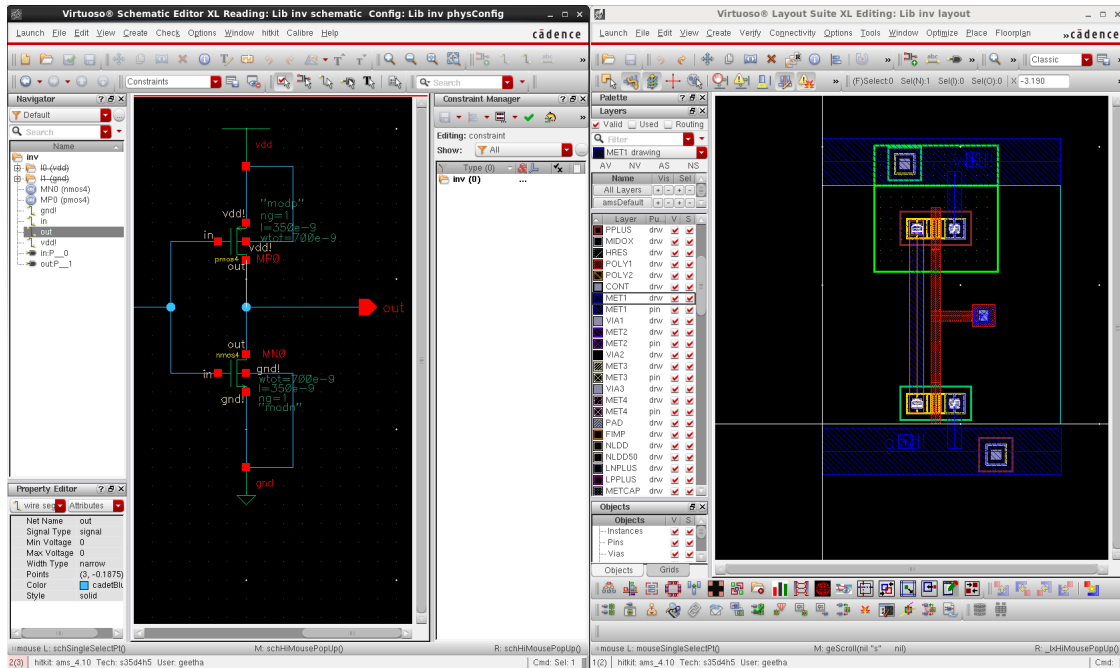
are any errors, you will see the errors noticed in the calibre LVS form shown below. Click on Error in the form to view what went wrong. It will explain each of the terms in the above window in great detail. The calibre form explains to you all the errors that it detected in both the schematic and layout views during the LVS comparison.



The following points illustrate a situation if we had an error in the previous step.

1. By clicking on Errors displayed in the LVS calibre window. Select the **my_inv** then **Discrepancies** that further breaks down to specific errors so that the errors that will be shown are only those that you want to see at this point. In our case, the only errors we encounter is the missing one of the nets in the layout that inturn fixes the Incorrect instances error.
2. To zoom into each error, select the error and then right click to select the highlight option. This helps to highlight the missing or additional nets and also helps to highlight the devices.
3. Modify the layout or schematic appropriately and rerun the LVS check till your layout design is perfectly matched to the schematic view.
4. There is also a feature that can help you in debugging the layout; especially in checking the connections of your nets. Move your cursor to the net (in the Schematic Window) you want to highlight and click on the left mouse button. Anything that is connected

to that net will be highlighted. You are then able to determine which connection or connections should not have been made to that nets based on what has been highlighted. An example of the highlight is shown on the next page.



One should realize that almost no one designs a perfect layout on the first attempt so do not expect to pass the LVS check on your first try. In most cases, there will be many errors reported in the lvs report and calibre form. You should not be intimidated by all these errors. Many of these are, in fact, related to each other. Hence, once you fix one of these errors, many of the other errors should disappear. The idea is to concentrate on one error at a time, change the layout design accordingly and repeat LVS steps until the layout and schematic views match perfectly with each other.

8 Space Based Router

Cadence's space based router tool is used to generate optimized layouts automatically based on the requirements if, for example, for Device level, ASIC, or Chip Assembly. It is a routing solution integrated into the Virtuoso Layout Suite and provides a comprehensive set of routing features for various layout tasks. It helps make huge and complex physical layouts of custom digital, analog, and mixed designs at the device, cell and block levels in a small amount of time. You can run automatic routing using the **Wire Assistant**. It supports some of the following routing topologies like: **Minimum Spanning Tree** used for most connectivities where the algorithm is based on minimizing wire length, **Pin to Trunk** is used for connecting power rails where it uses spine style routing and **Pin to Aligned Pin** used to quickly route straight connections between aligned pins. In addition, the **Wire Assistant** has three pre-defined topologies say Device level, ASIC and Chip Assembly as discussed before.

We will walk you through all the necessary steps for designing and testing a 4-bit synchronous counter which is functionally equivalent to a 74LS163. We will create a schematic of the 4-bit synchronous counter before that we create a 1-bit synchronous counter cell using standard cells from the **CORELIB**. We will then will create a symbol for the 1-bit and the 4-bit synchronous counter and test the transient characteristic for both 1-bit and 4-bit synchronous counters using the Analog Artist Simulator. Lastly, we will create a layout for the 4-bit synchronous counter using the space based routing tool automatically and run the DRC and LVS checks for the 4-bit synchronous counter. A final simulation will include parasitic capacitances which will more accurately reflect the true performance of the 4-bit synchronous counter.

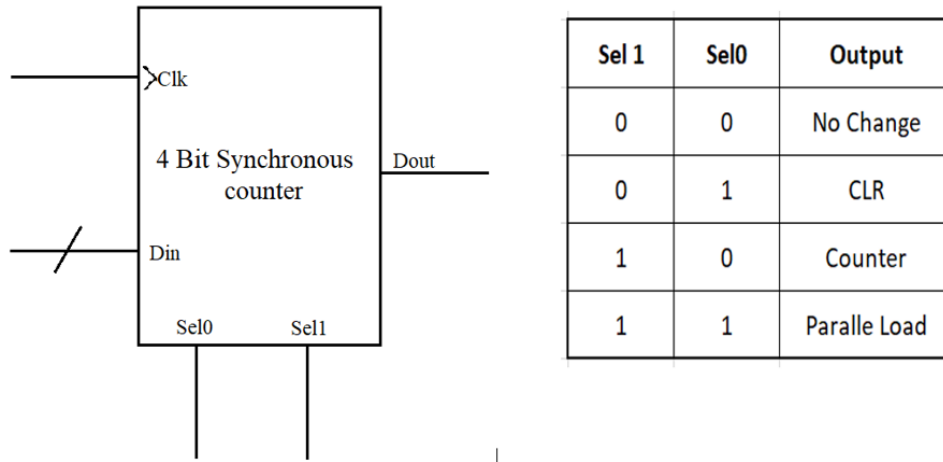
8.1 4-Bit Synchronous Binary Counter

We will design, simulate, and layout a 4-bit synchronous counter which is functionally equivalent to 74LS163. A 74LS163 is a 4-bit synchronous counter that can count up from 0(0000) to 15 (1111). Instead of implementing the complex circuit shown in the datasheet of the 74LS163 with specific pins like CLEAR, NO CHANGE, PARALLEL LOAD and EN, we will design some simple logic using the select controls (Sel0 and Sel1) to achieve the same functionality.

Initially, we will design a 1-bit synchronous counter using a positive-edge triggered D-flipflop and a 4 to 1 multiplexer with a 2-bit select bus to implement the CLEAR, NO CHANGE, COUNT UP and PARALLEL LOAD function. A half-adder with one input set to 0 will be used to implement the COUNT UP function. The synchronous counter we will design is able to maintain its previous state when the select bits Sel[1:0] are "00", clear its output when the select bits Sel[1:0] are "01", count up by one when the select bits are "10" and finally it does a parallel load when the select bits are "11".

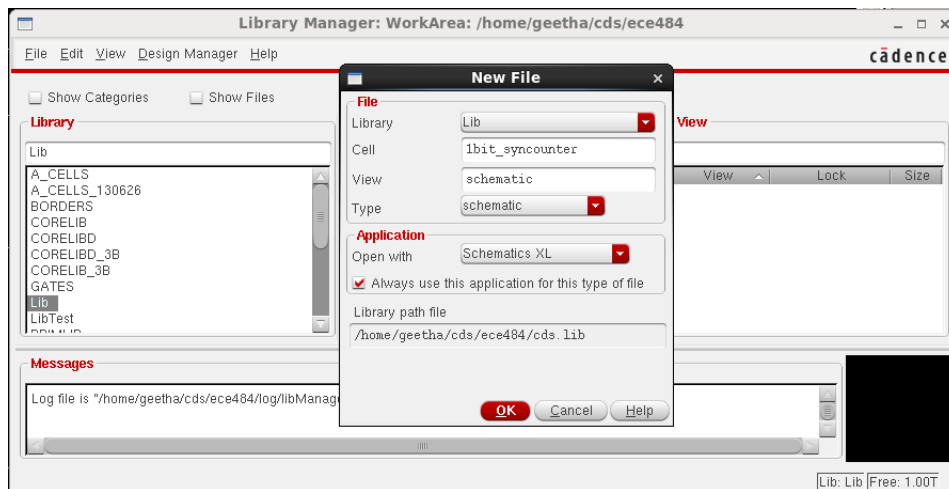
We can either design the primitives cells, for example, a D flipflop, a half adder and a 4x1

MUX using transistors and gates or we can use the standard cells available in CORELIB. In this document we will simply use the standard cells from CORELIB. Once we have a 1-bit counter, we can easily construct a 4-bit synchronous counter by connecting the four 1-bit cells together by connecting the carry out of the preceding section to the next stage input. You can look at the block diagram of the final 4-bit synchronous counter and its functions listed as a truth table in the figure shown below.

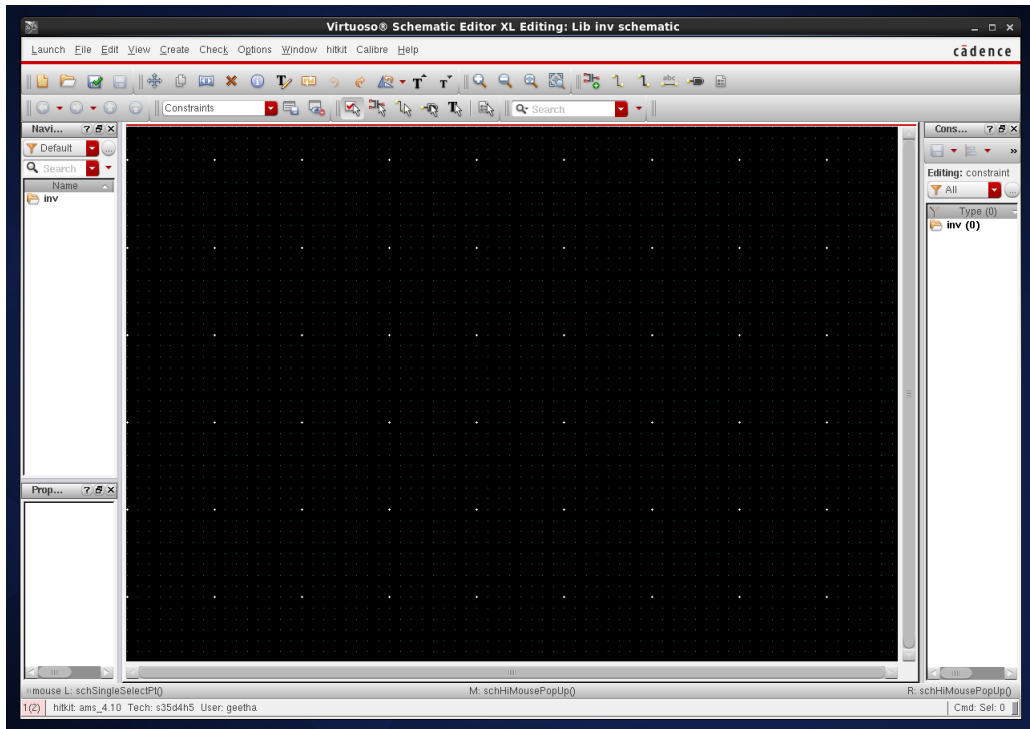


8.2 Schematic Entry and Symbol Creation

Launch the Cadence tools and create a schematic for a 1-bit synchronous counter with **Schematic XL** as shown in the form below. Just make sure that whenever you create a schematic for a logic cell, it should be saved in the **Lib** directory. Also, when we create a schematic of a "testbench" (something we will use to test the cell), we will want to save the testbench in **LibTest**.

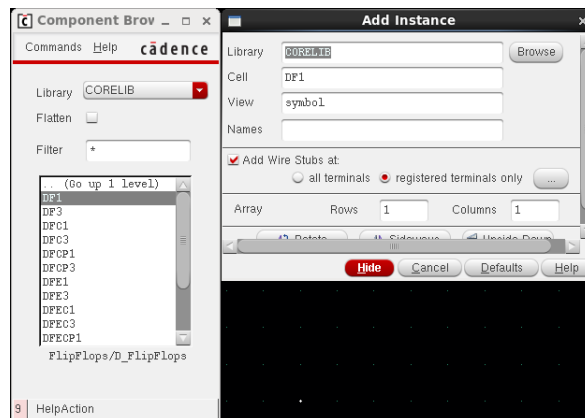


Left click the **OK** button. The **Virtuoso Schematic Editor** window should pop up as shown in the figure below.



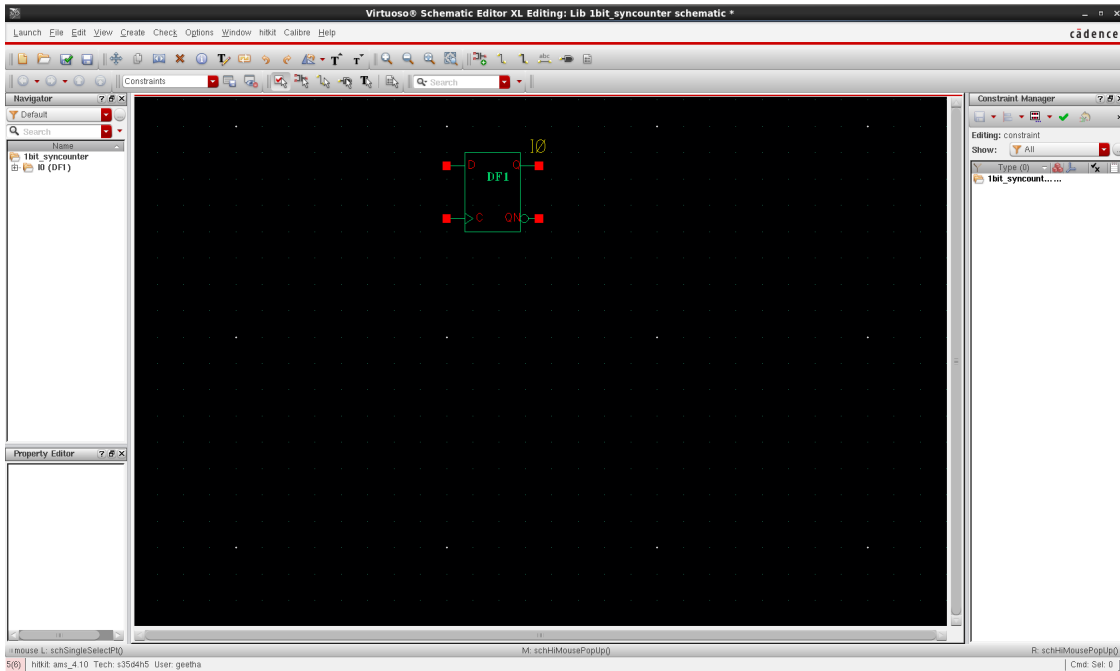
Left click: **Schematic Editor: Create** → **Instance** or you can press **i** in order to insert an instance.

A **Command Browser** window appears. In this window select **CORELIB** under the **Library** pull-down menu. Next click on **FlipFlops** → **D_FlipFlops** → **DF1**. The window should look like the one in the figure below.

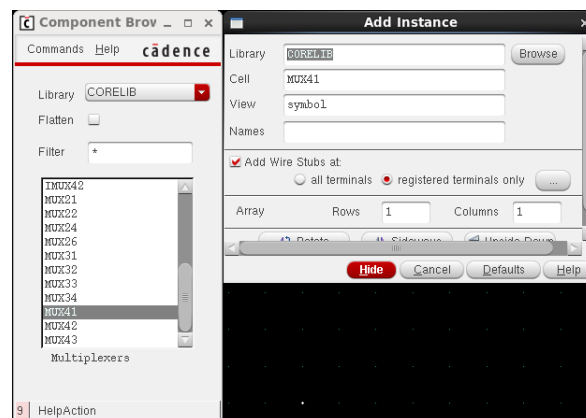


Move the cursor into the editing window. Notice that there is a D-flipflop there instead of the normal cursor. Position it where you want to put the transistor, and **left click** to place

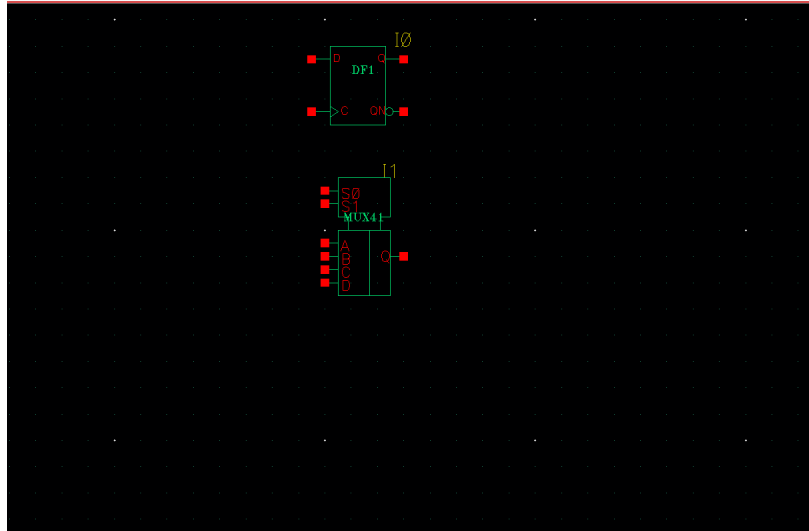
it. You can **right click** to rotate the D Flipflop if you want it to face a different direction (this is especially useful with pins). While placing, stretching, etc., you can press **F3** to show the options form for the command if it is currently hidden. Once you place it, the window should look like the one in the figure below.



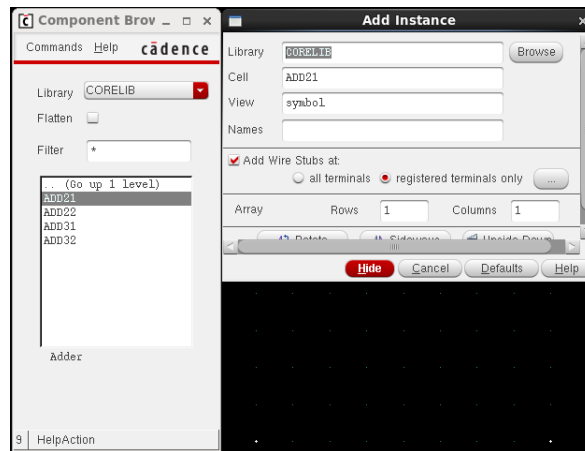
Add another instance **MUX41** under **Multiplexers** from **CORELIB**.



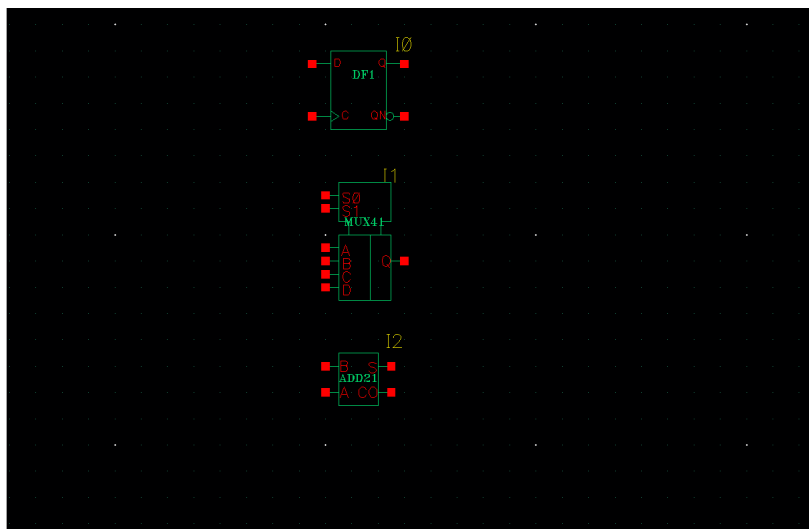
Now the window should look like the one in the figure below.



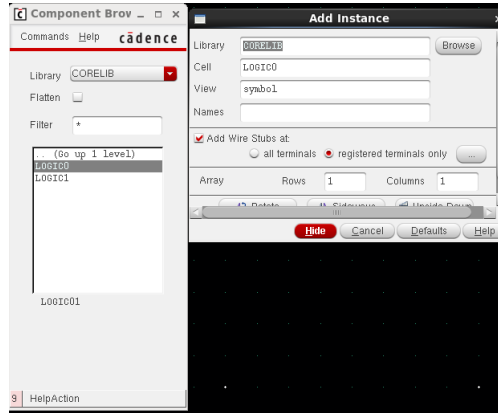
Then add another instance **ADD21** under **Adders** from **CORELIB**.



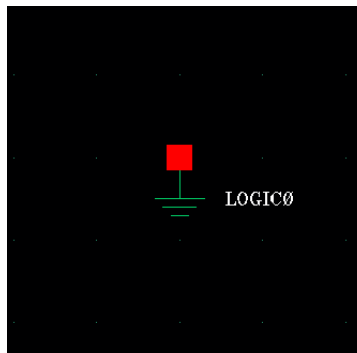
And the window should look like the one in the figure below.



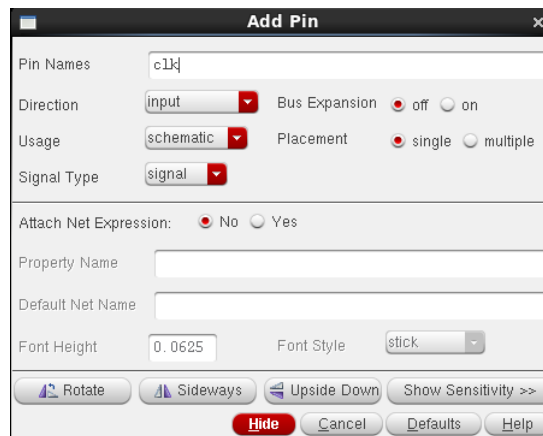
We will also use LOGIC0 symbol to implement the CLEAR function, so add the LOGIC0 symbol as well, following the same steps as before. We get this LOGIC0 instance from LOGIC01 under CORELIB.



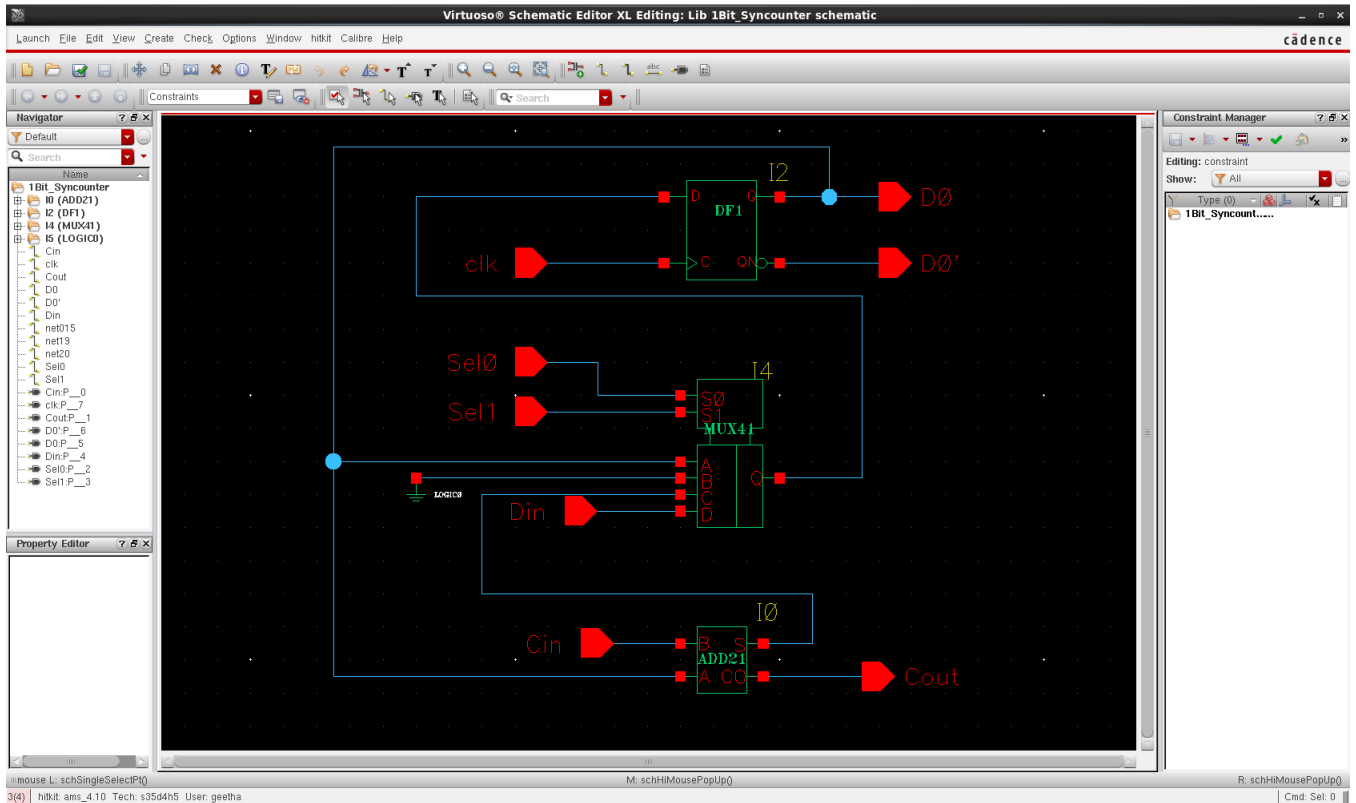
LOGIC0 looks like the one in the figure below.



Now, we'll add the pins using the shortcut key:p, by stating if they need to be input or output in the form as shown below.

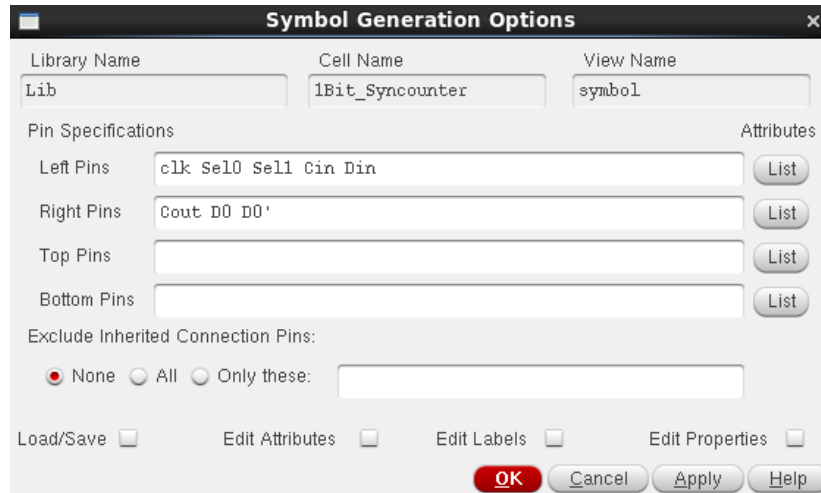


Once you add all the required components: D Flipflop, 4-to-1 Multiplexer, Half-Adder, LOGIC 0, input and output pins that are needed, use the shortcut key:w to create wires. You have to make connections between the components and pins in the same way as shown in the figure below.



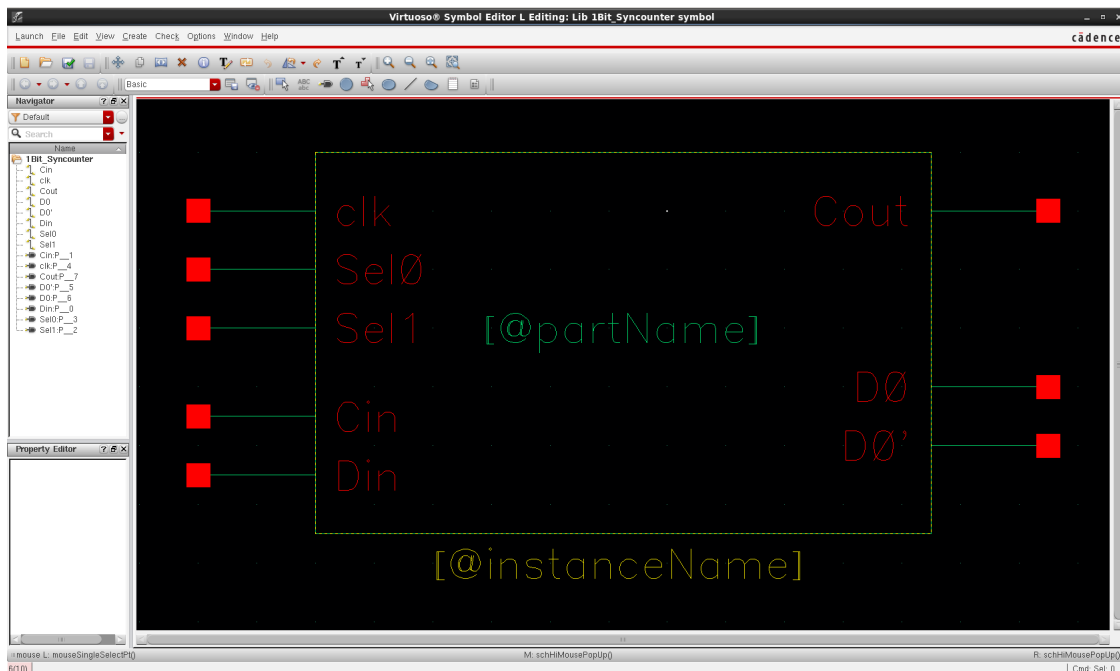
Once you are done editing, left click the "check mark" icon on the left side of the screen. This will check your work for connection errors and will save your work in the library. You can accomplish the same thing by left clicking **Schematic Editor: File → Check and Save**.

You have created a schematic for the 1-bit synchronous counter. Now it's time to create a symbol for it. Using the symbol editor, you can create the symbol either by using a create new cell view or left click **Schematic Editor: Create → Cellview → From Cellview...** where as creating symbol through schematic identifies the pins that are already defined in the schematic and automatically generates a symbol with the same pins as shown below.



Click OK.

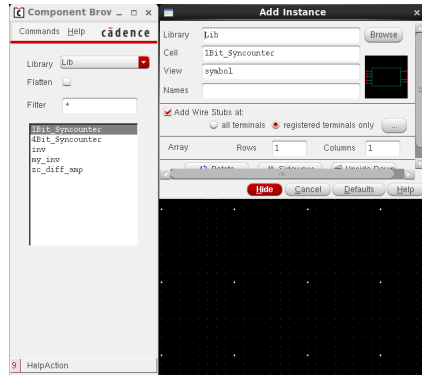
The 1-bit synchronous counter symbol should look like the one shown below.



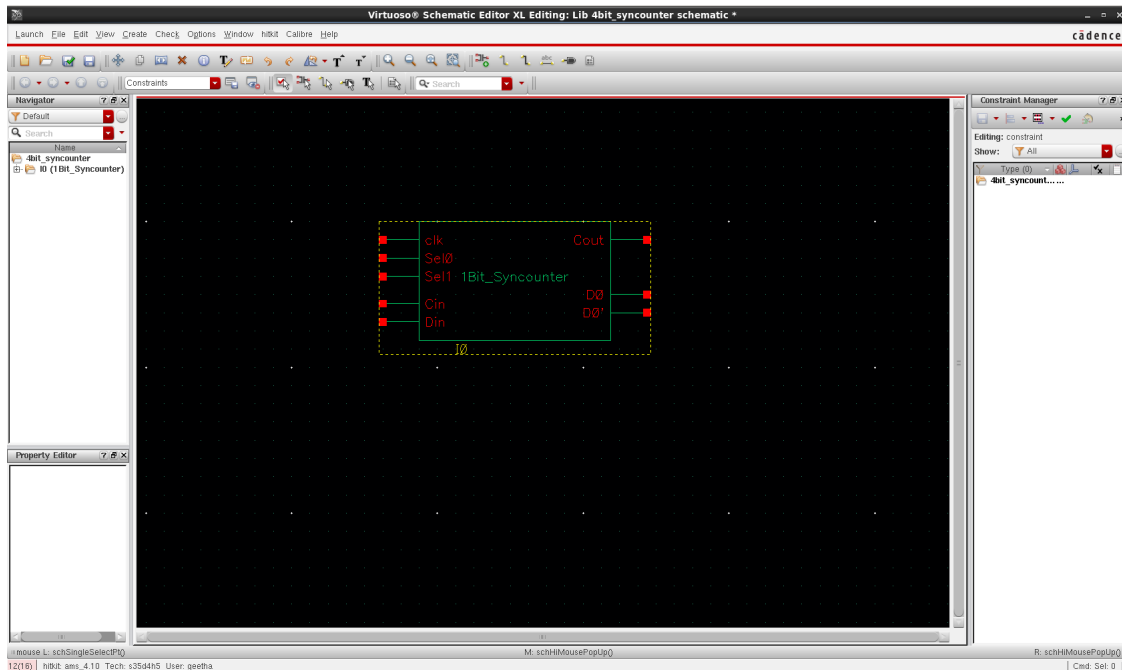
By now, You have successfully created a 1-bit counter cell.

Now, we use four of these 1-bit counter cells to build a 4-bit synchronous counter just by connecting the carry-out of a lesser bit to the carry-in of a more significant bit. The carry-in of the first bit will be set to 0.

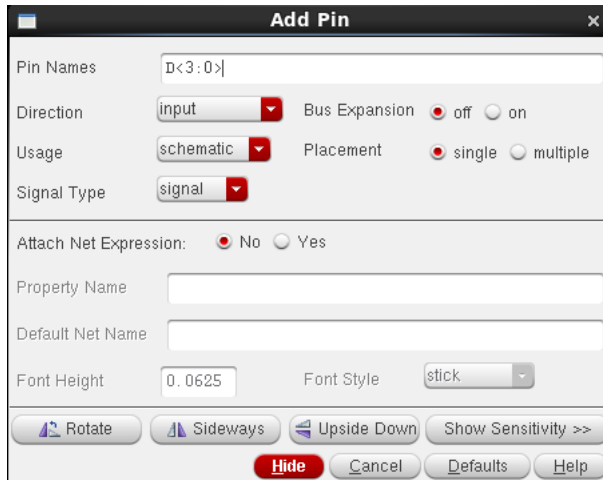
Create a schematic for a 4-bit synchronous counter using **Schematic XL** in the same way as we did for 1-bit synchronous counter. The **Virtuoso Schematic Editor** window should pop up. Then add the instance **1Bit_Syncounter** from **Lib** which you have created before.



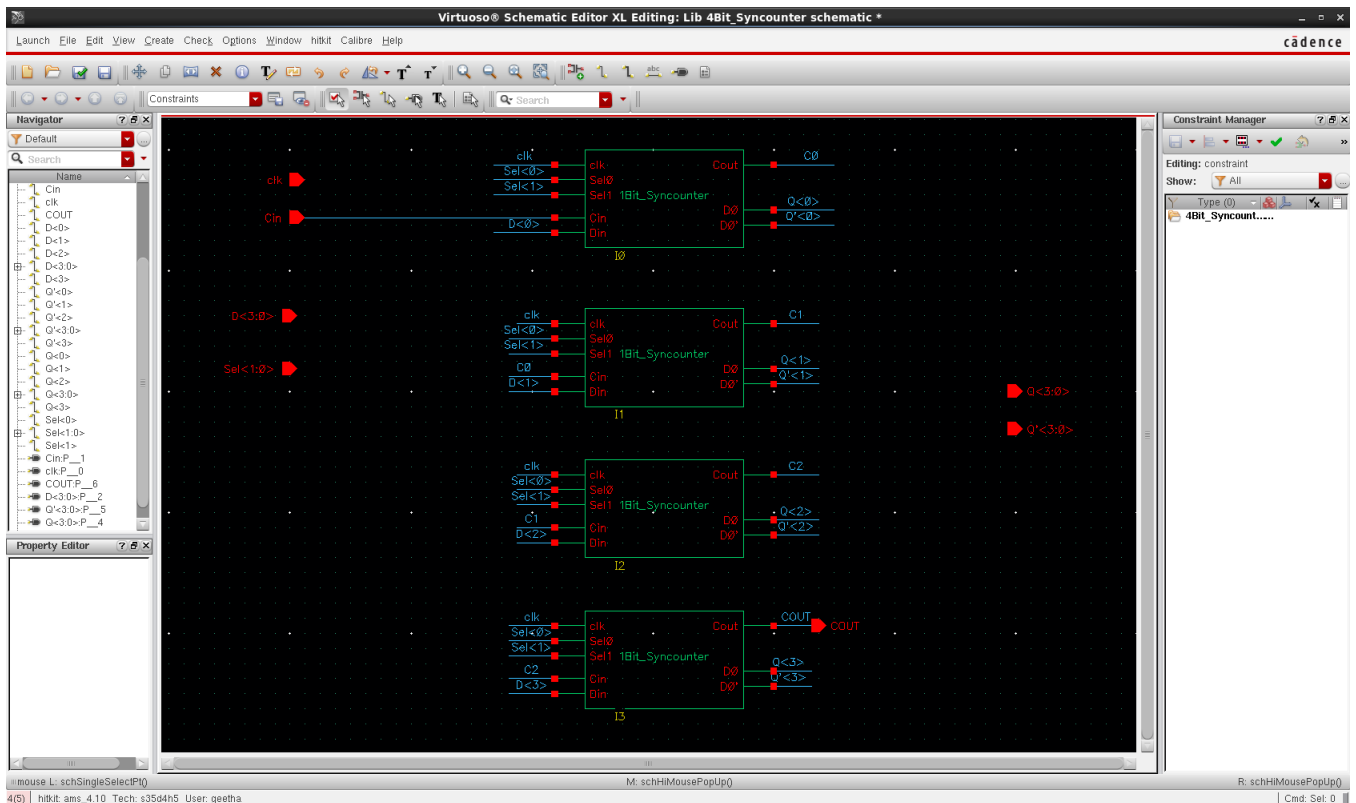
After you place it, your schematic editor window should look like the one shown below.



Proceed further placing four of the 1-bit synchronous counters, either one below the other vertically or one beside the other horizontally and connect them up. Since you are creating a 4-bit counter you can use compound data input, output and select pins say **D[3:0]**, **Q[3:0]**, **Qbar[3:0]** and **Sel[1:0]** along with the clk and carry bit pins. You can create compound pins as shown below.



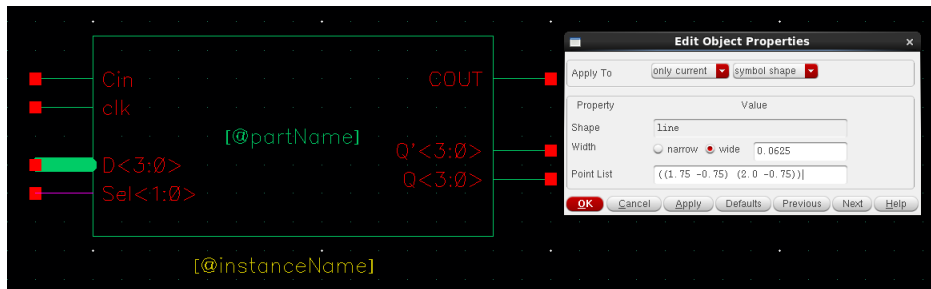
Once you add four 1-bit synchronous counters, a clock pin, select pins, along with data input and output pins, use the shortcut key:**w** to create wires and make connections between the components and pins. Also, use the shortcut key:**l** to label the wires in a way similar to that shown in the figure below.



Once you are done editing, left click the "check mark" icon on the left side of the screen. This will check your work for connection errors and will save your design.

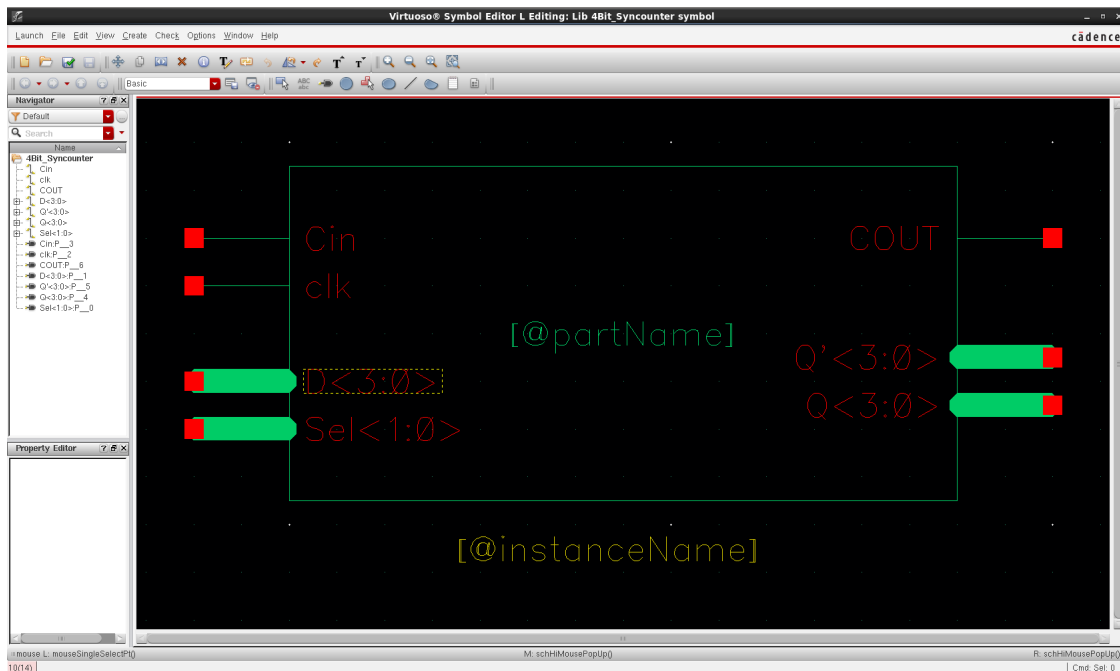
You can now create a symbol for the 4-bit synchronous counter from the schematic you made.

Create a symbol for the 4-bit synchronous counter from its schematic in the same way you did for 1-bit synchronous counter and make the 4-bit signal lines say D[3:0],Q[3:0],Qbar[3:0] and the 2-bit select bus say Sel[1:0]. To do this select the D[3:0] line and use the shortcut key **q** and select the width to be wide as shown in the below form.



Click OK.

Repeat the same for other signal wires. Your 4-bit synchronous counter symbol should look like the one below when finished.



You should now close the Symbol Editor and also the Schematic Editor Window.

Congratulations! You have successfully created your schematic and associated symbol. But we need to test if your 1-bit and 4-bit synchronous counters designs are correct. To do this we have to run electrical simulations to prove that indeed the design is correct.

8.3 Electrical Simulation

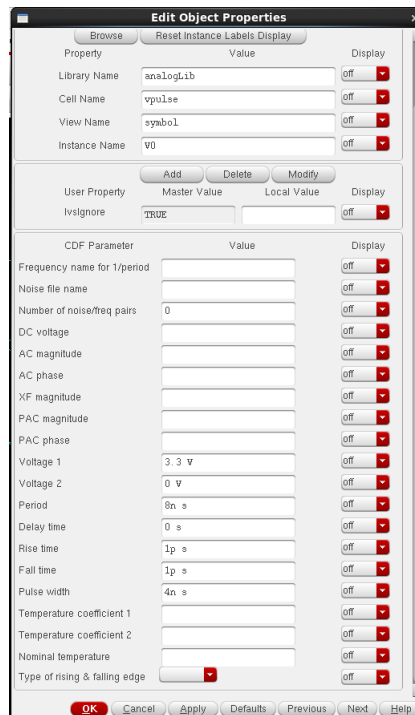
Before we can simulate the 1-bit and 4-bit synchronous counter designs, we must create *another* schematic, what engineers call a *testbench*. We start with creating a test environment for the 1-Bit Synchronous Counter first and then proceed further with 4-Bit Synchronous Counter.

Use the Library Manger to create a new schematic in the **LibTest** folder. The name of the testbench should be **1Bit_Syncounter_tb**.

Begin by instantiating your 1-Bit Synchronous Counter symbol, **1Bit_Syncounter**. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **1Bit_Syncounter** from **Library Name: Lib** and **View Name: symbol**. Or you could also use the shortcut key **i**.

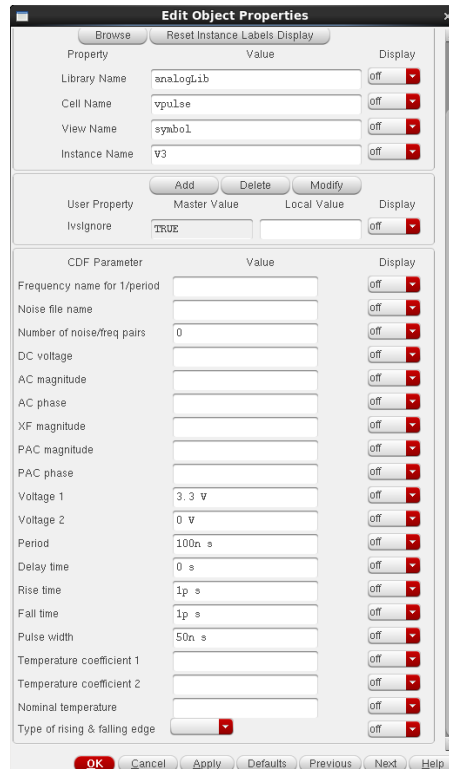
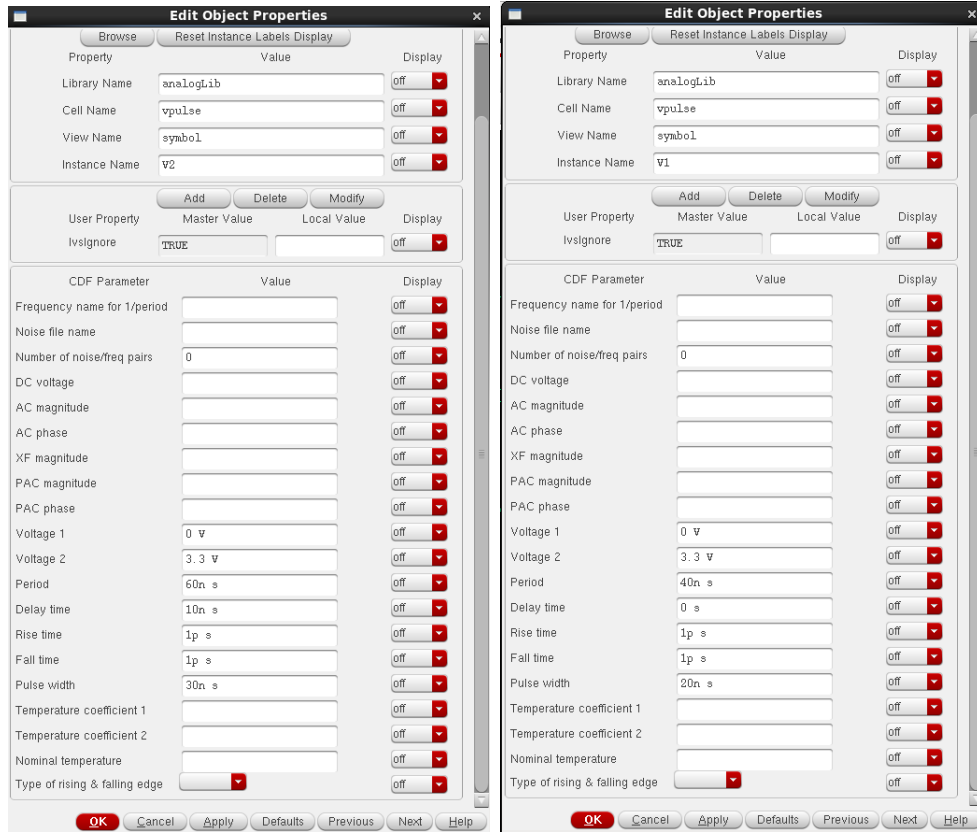
We then need to add a clock say voltage source to drive the input of the counter. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **vpulse** from **Library Name: analogLib** → **Sources** → **Independent** and **View Name: symbol**.

Fill in the properties form as shown in the figure below. The TA will explain the meaning of the parameters. Place this symbol such that the positive end is connected to the input of the 1-bit synchronous counter. The negative end should be connected to a **gnd** net which can also be found in the analogLib library.

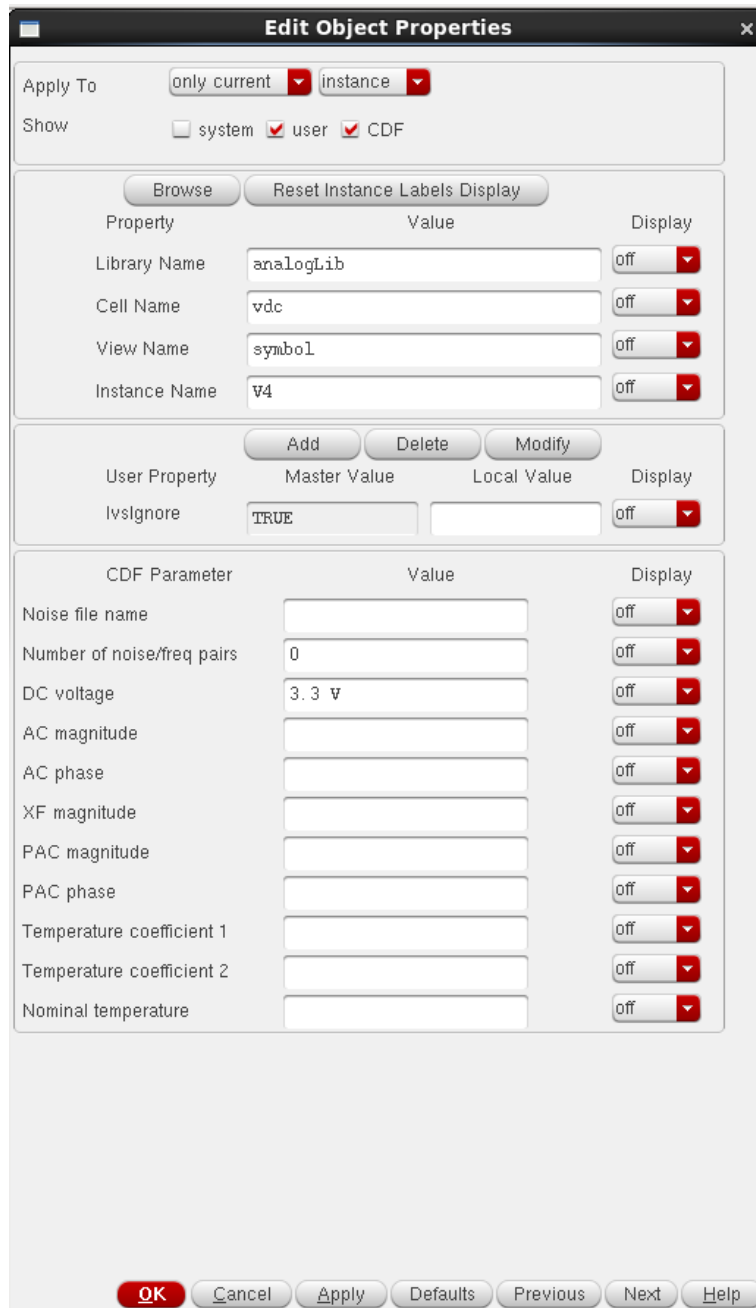


In the same way add a Select controls and Data line say Sel[1], Sel[0], Din as inputs of the counter using a **vpulse**. You can use the vpulse properties for Sel[1], Sel[0] and Din in order

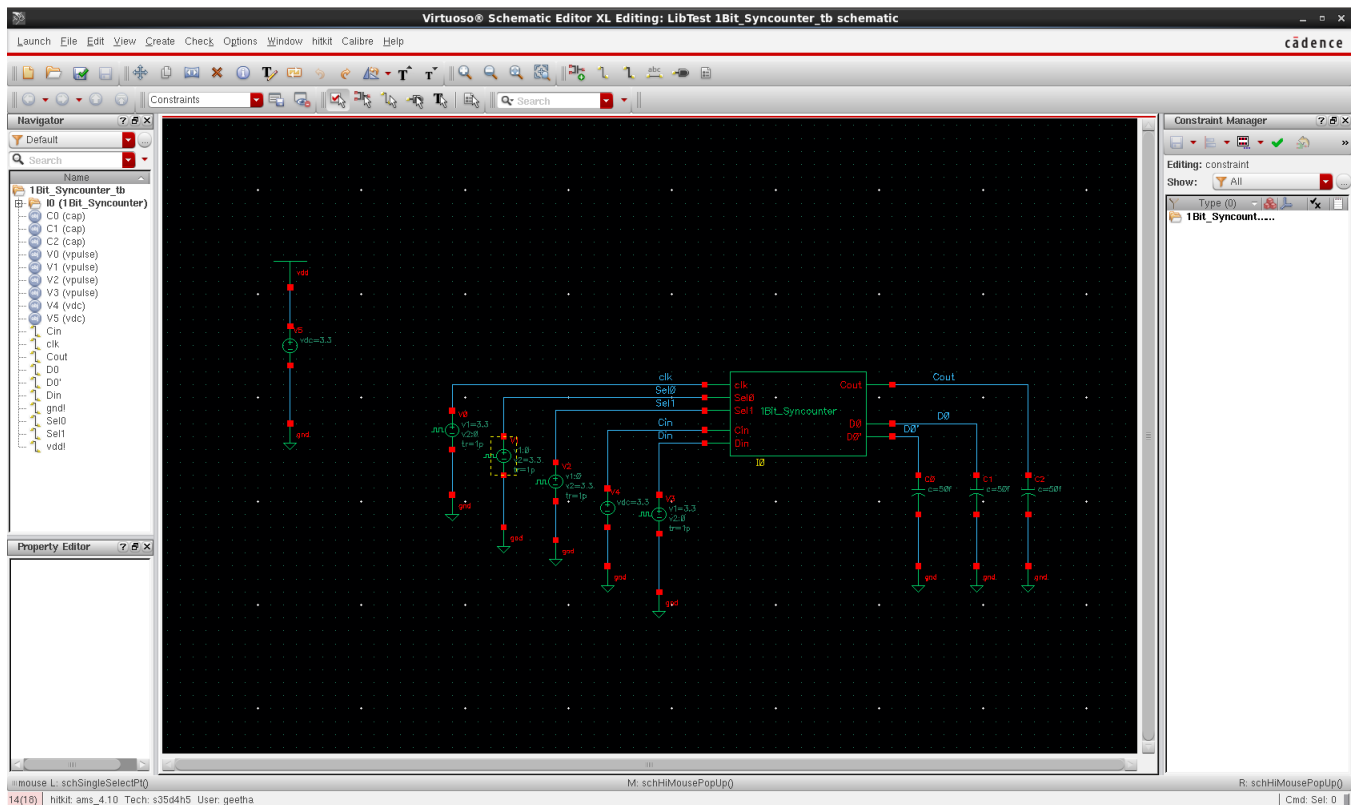
as shown below as reference or you can come up with your own test cases to see if your 1-bit synchronous counter design is working correctly.



We also will need to add a DC voltage source which we will connect up to the **vdd** net. This can be done if you Left Click **Add** → **Instance** and select Cell Name: **vdc** from **Library Name: analogLib** and **View Name: symbol**.



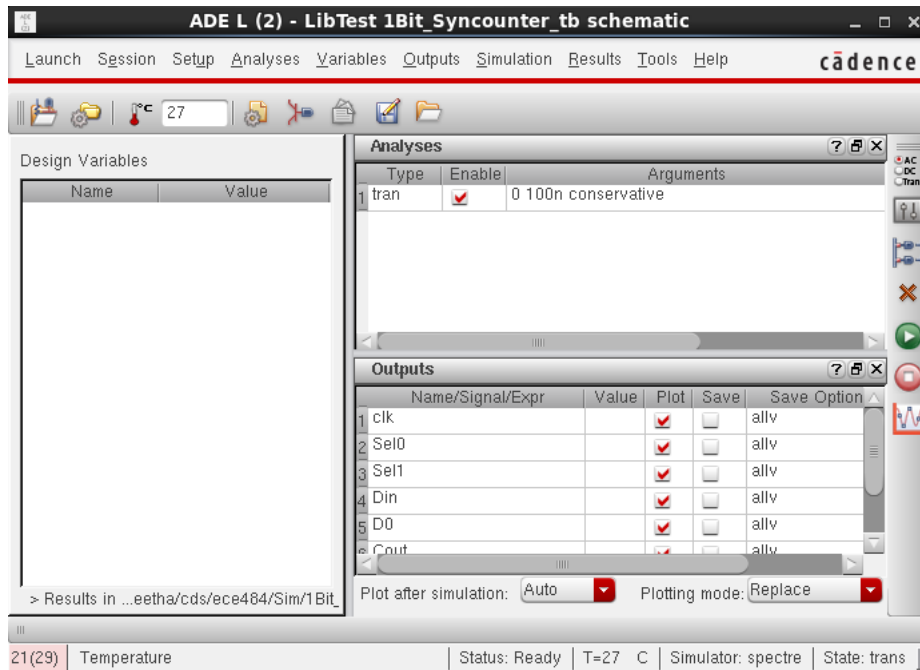
In the same manner as just described, add capacitors from the 1-bit synchronous counter output lines say **D0**, **D0bar**, **Cout** to the **gnd** net. Wire up everything as you did in the previous section of this document. Also, you will want to use the shortcut **I** to label the input and output nodes. Just click on the node you want to label before pressing the **I** key. When completed your testbench schematic should look like the circuit shown on the next page.



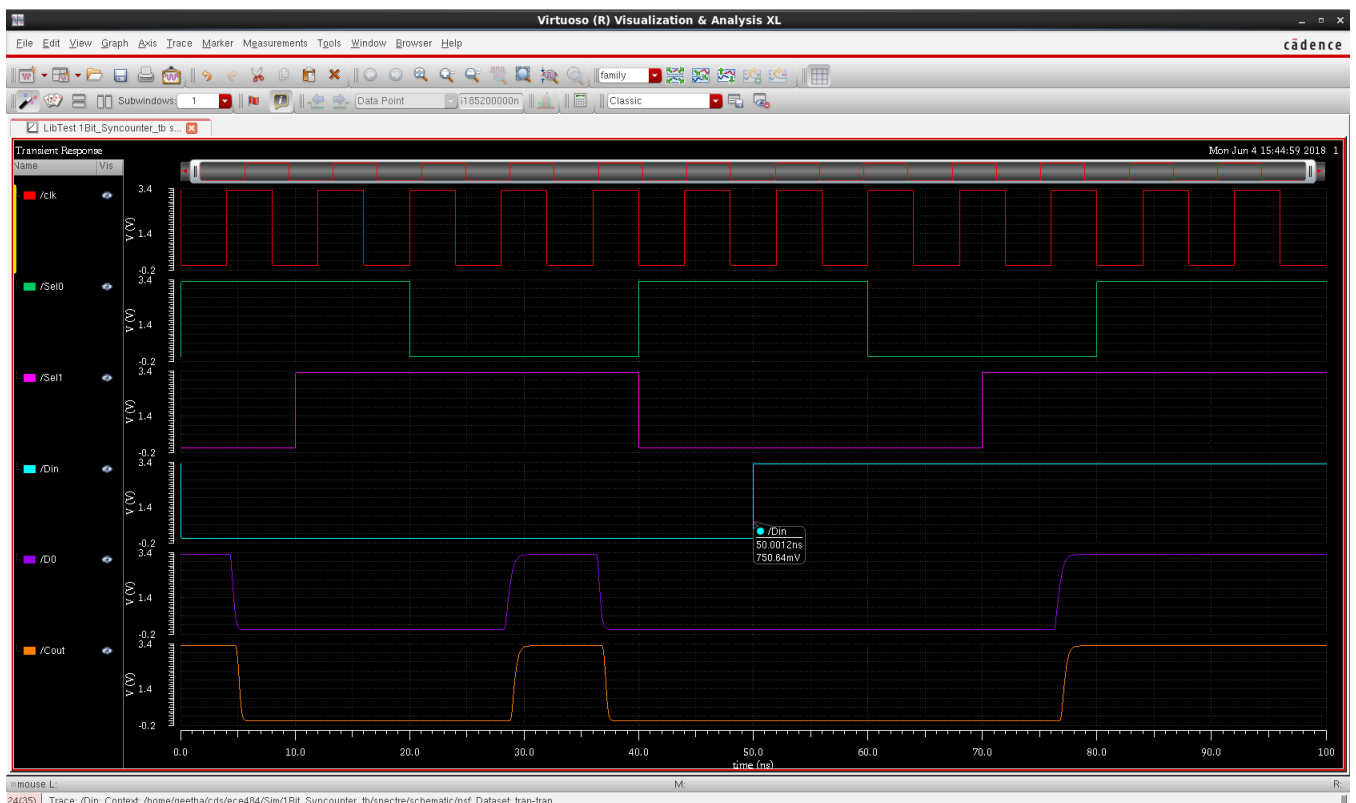
Save the schematic. In the schematic window menu you should: Left Click **Editing:File** → **Save**. We are now ready to run a transient analysis on the 1-bit synchronous counter. This time you should: Left Click **Editing: Launch** → **ADE L**. Click YES. Make sure that the ADE window settings are configured properly as discussed in your first lab.

In the ADE window menu go to **Analysis** → **Choose** ... Set the transient analysis to have Stop Time as 100n. We have specified a transient analysis from 0 to 100ns. Make sure the **conservative** box is checked.

Now select the input and output nodes of the 1-bit synchronous counter, you would like to display. Left Click **Outputs** → **To be Plotted** → **Select on Schematic**. Use the **ESC** button to exit this mode. Make sure that the "Plot" buttons are checked for the input and output nodes that need to be plotted. The Analog Design Environment window should look like the one shown below.



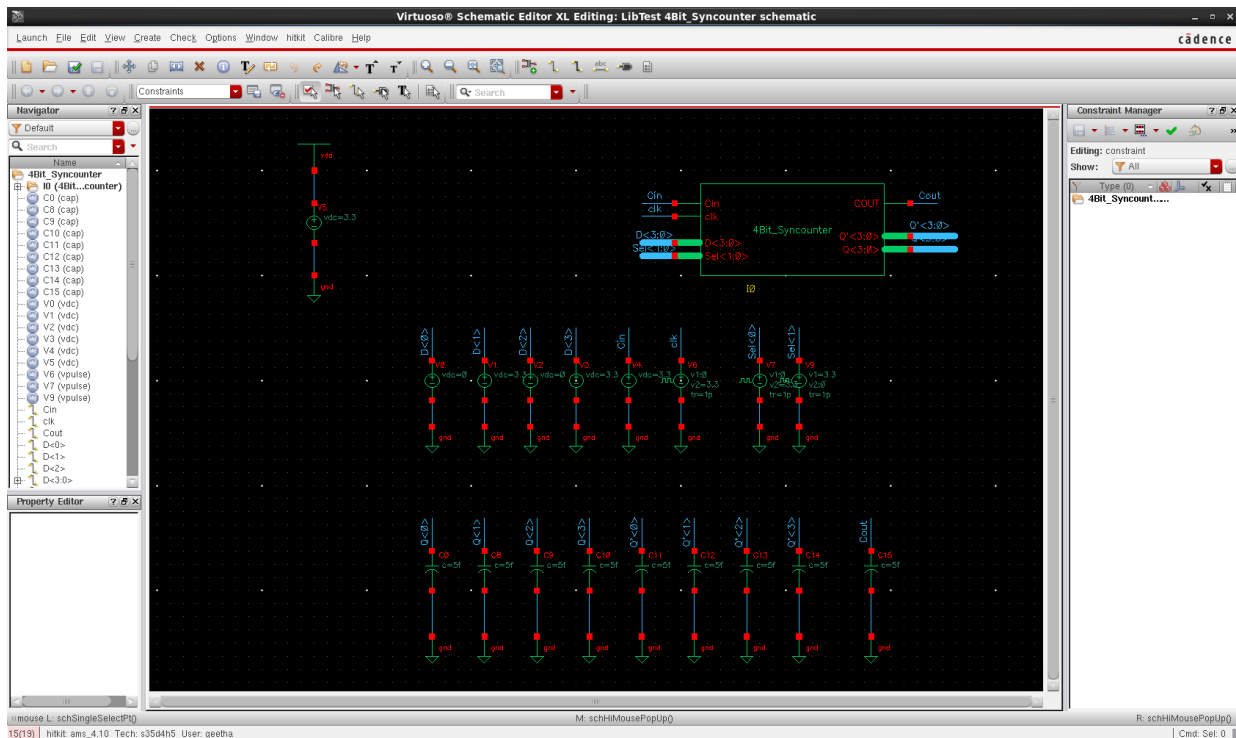
Run the simulation by pressing on the green traffic light icon. After a few seconds you should see the report showing the simulation summary and you will also see the wave window appears with all the waveforms overlapped. Left click **Axes** → **Strip** in order to separate the different curves. The result should look like the one on shown below.



Notice that the output node voltage is delayed from the input and that the output rise time is longer than the output fall time. You have to verify if 1-bit synchronous counter is working as expected by comparing the waveform with the truth table we discussed. You can even save the state, when you want to re-run the simulation all you have to do is to load the state.

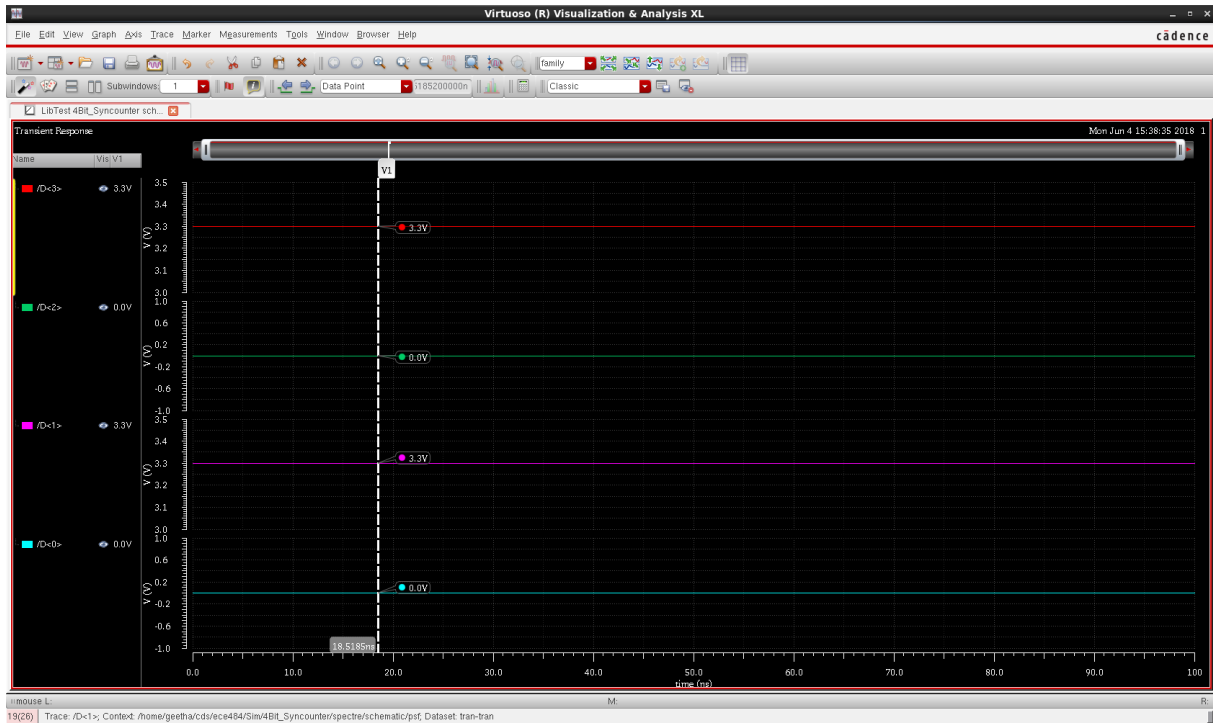
Once your 1-bit synchronous counter is working correctly, you are good to proceed further and create a testbench for the 4-bit synchronous counter.

Create a test bench for the 4-bit synchronous counter in the following the same procedure you followed while creating 1-bit synchronous counter testbench. Input appropriate voltage pulse properties say clock, D[3:0], Sel[1:0] to check the operation of the 4-bit synchronous counter that counts sequentially for every clock pulse. Also, add capacitors from the 4-bit synchronous counter output lines say Q[3:0], Qbar[3:0], Cout to the gnd net. Wire up everything and label the input and output nodes. Your testbench schematic should look like the figure shown below.



Save the schematic. We are now ready to run a transient analysis on the 4-bit synchronous counter. you should: Left Click **Editing: Launch** → **ADE L**. Click YES. And configure the ADE window parameters, then set the transient analysis to have Stop Time as 100n. We have specified a transient analysis from 0 to 100ns. Make sure the **conservative** box is checked.

Also select the input and output nodes of the 4-bit synchronous counter, you would like to display as shown in the figures in the next page.



Its time to verify if your 4-bit synchronous counter is working as expected by comparing the waveform with the truth table we discussed. You can even save the state, when you want to re-run the simulation all you have to do is to load the state. Your TA will help you analyze the waveforms.

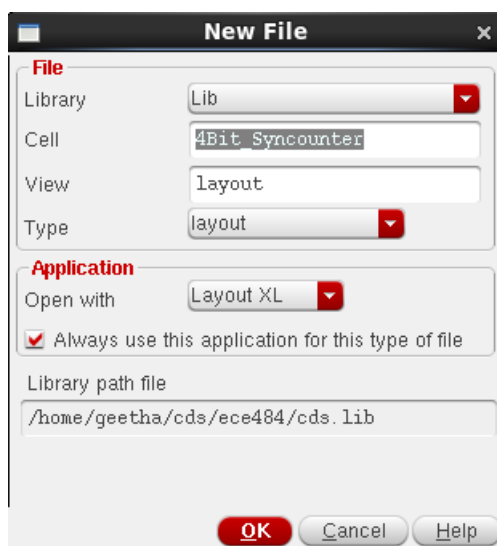
8.4 Physical Layout, DRC, and LVS

In this section of the document we will walk through how to physically lay out the final cell which is 4-bit synchronous counter using space based automatic routing tool. We will then check to make sure that the layout confirms to all of the manufacturer's rules (DRC), and finally we will make sure that the layout yields a netlist which is equivalent to the netlist produced by the schematic editor (LVS).

8.4.1 Layout

By now, you should know how to enter a schematic, how to produce a corresponding symbol for the cell, and finally how to simulate your design to make sure it functions correctly. The next step in the process of making an integrated circuit (IC) is to perform the physical layout of the cell.

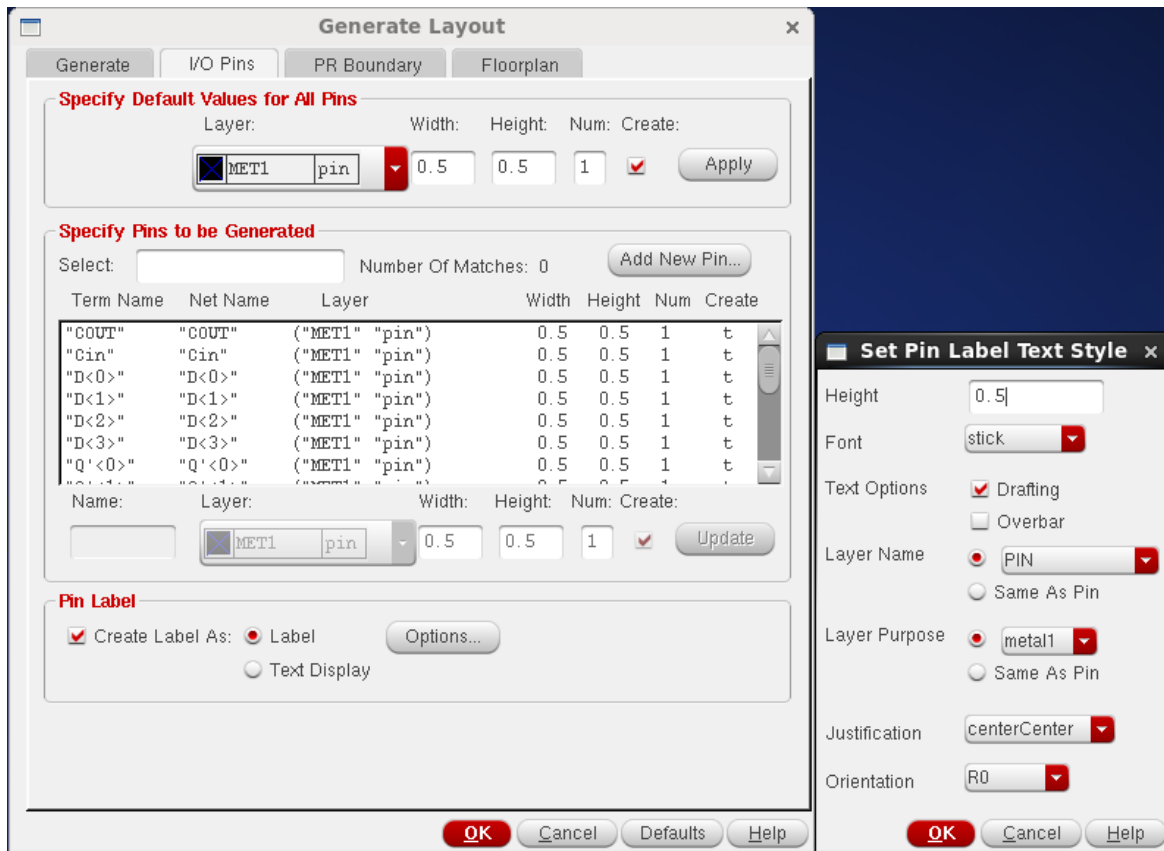
We will create a layout automatically using the space based routing tool for the **4Bit_Syncounter** cell. You should launch the Library Manager just like you did in earlier sections of this tutorial. In the Library Manager window, select the **4Bit_Syncounter** cell in **Lib**. You should then left click on **File** → **New** → **CellView**. A "New File" menu will pop-up. The form should be completed as shown in the figure on the next page.



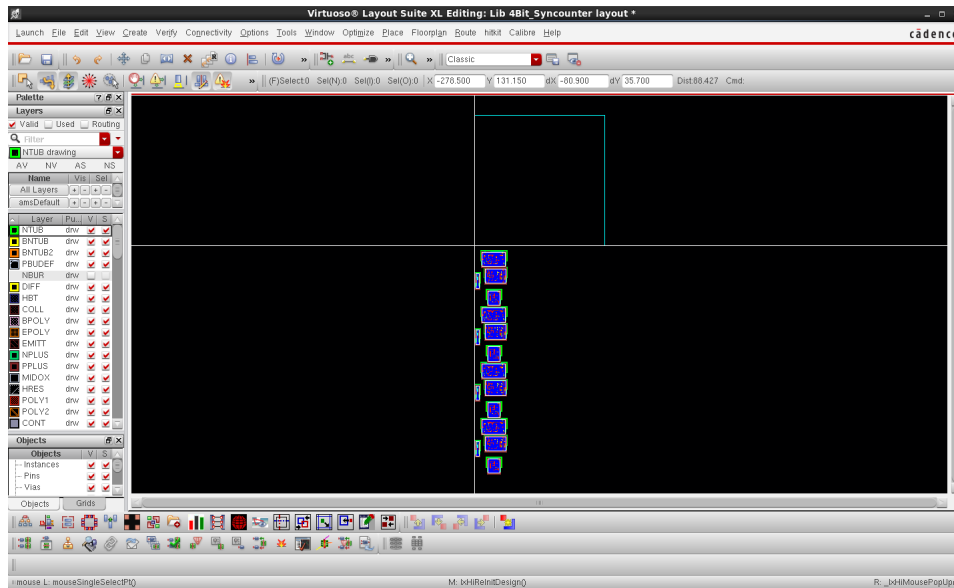
Then click on the **OK** button. An empty layout editor window and the schematic editor window will pop-up along with a Layers window. In the Virtuoso Layout Window (NOT the Virtuoso Schematic Window) from the main menu, you should select **Connectivity** → **Generate** → **All From Source**

You will see a **Generate Layout** window with Generate, I/O pins, PR Boundary, and Floorplan tabs. Go to the I/O Pins tab and choose the **Label** radio button. Then click on the **options** under Pin Label. You will now see "Set Pin Label Text Style" window. Choose

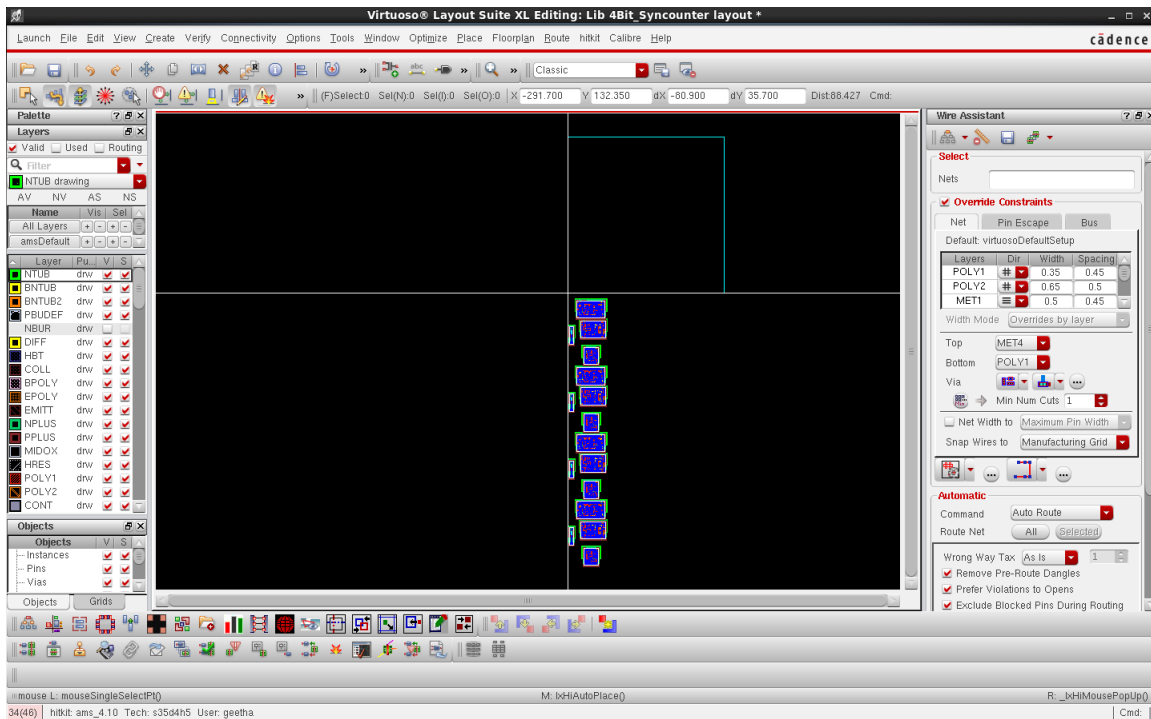
the Layer Name to be **PIN** and Layer Purpose to be **metal1** as shown in the below form.



Go ahead and click "OK" to generate all the Instances and I/O pins. You should see that the layout has D flipflops ,4-to1 multiplexers, and half-adders. You will also see some "blue" squares (metal-1 i.e. M1). These are the I/O ports: **vdd!**, **gnd!**, **Din** ,**Q**, **Qbar** and **clock**. Your Layout Editor Window should look like the one shown on the next page.

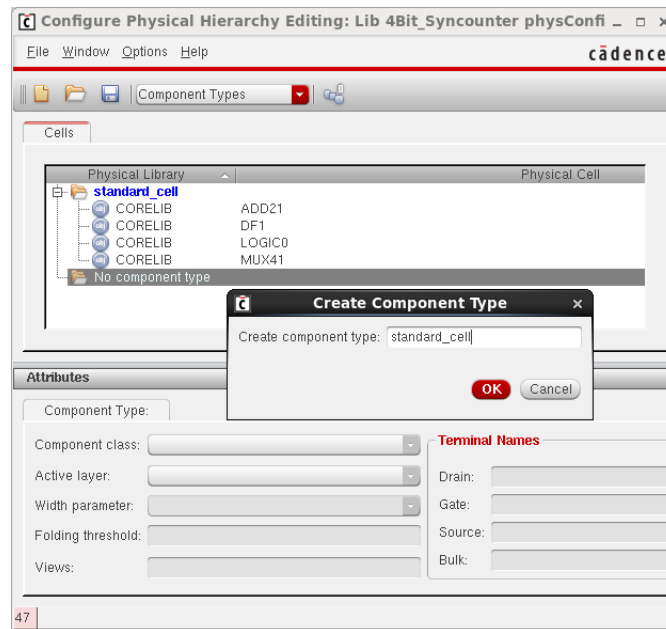


Add the **Wire Assistant** toolbar to your Layout Editor by choosing **Window** → **Assistants** → **Wire Assistant** from the menu bar, useful for performing space based routing. Your Layout Editor Window should look like the one shown below with the assistant tool bar.

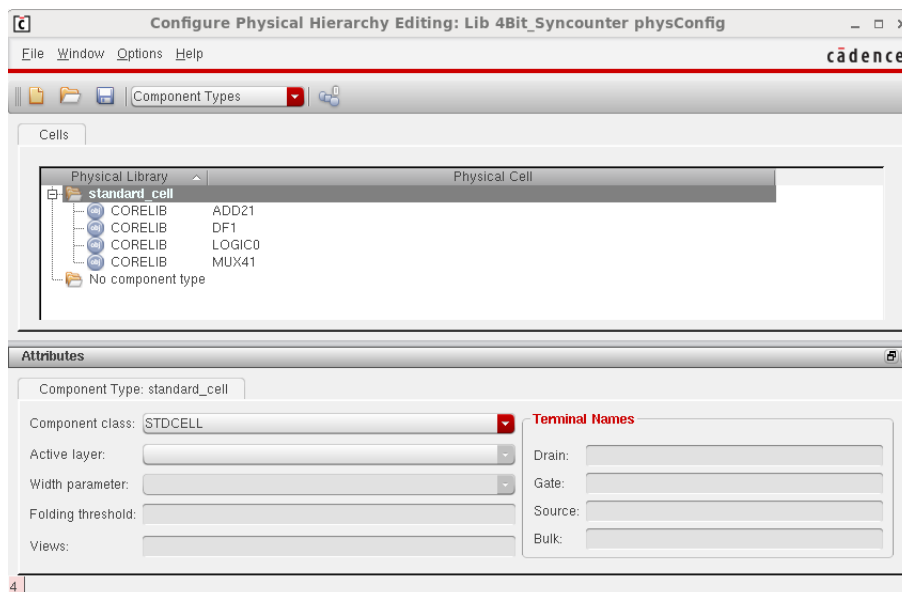


Click on **Edit** → **Component Type** to make the components/instances present in the Layout editor categorized as **No Component type** to be **standard_cells** as shown in the form on the next page. To do this, we need to create a component type named: **standard_cells** and move the components under No Component type into standard_cells component type

you created.



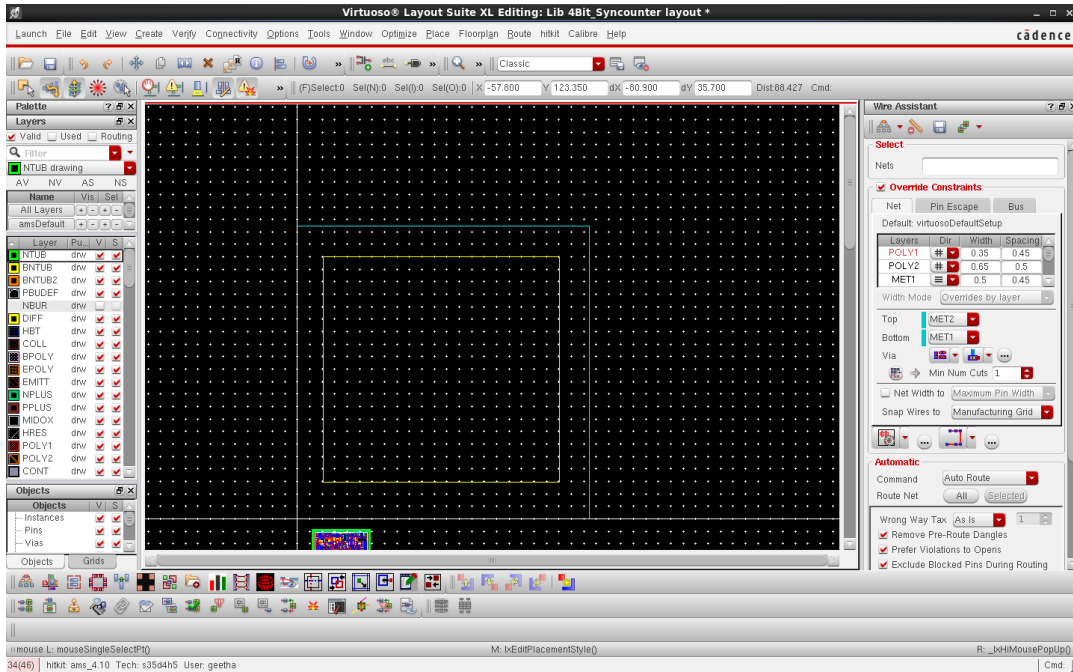
Once you move the components into standard_cell component type, select **standard_cell** and choose **component class** to be **STDCELL** under the Attributes section as shown in the form below.



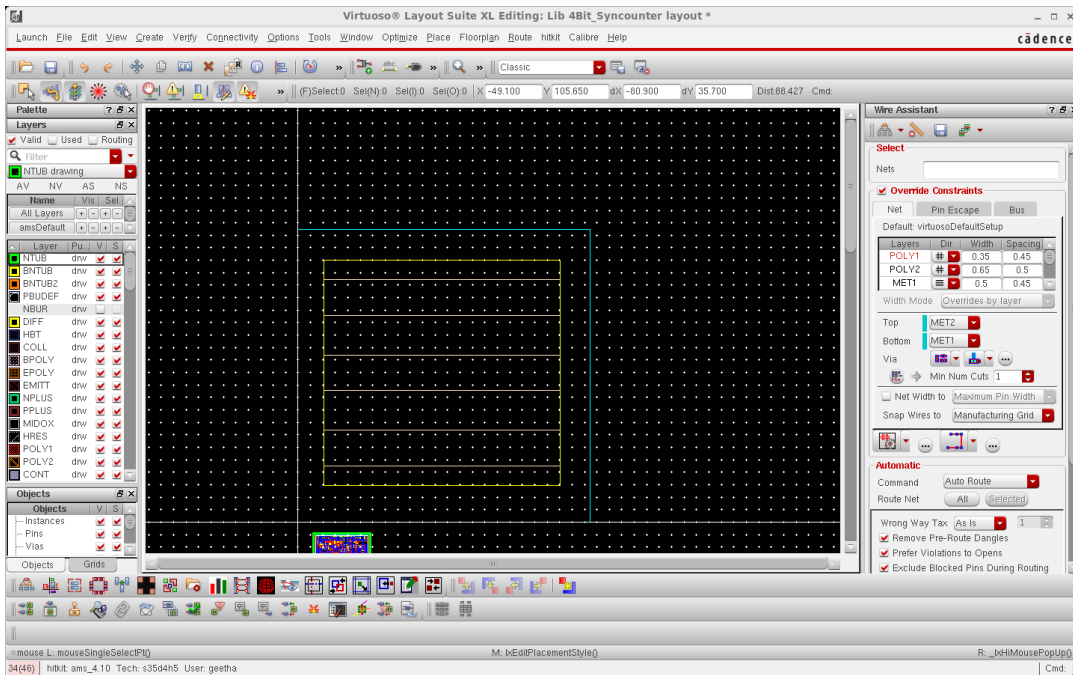
Go to the Layout Editor menu bar and select **Place** → **Custom Digital** → **Placement Planning**. You should see a pop up menu like that shown in the figure below.



In the above **Placement Planning** form under the *Regions* tab of *Style Parameters*, click on the button **Draw**. Then you should select a point on the Layout Editor and draw the square box by clicking on the other end to place your complete layout of the 4-bit synchronous counter inside it. The Layout Editor window should look like the one shown below.

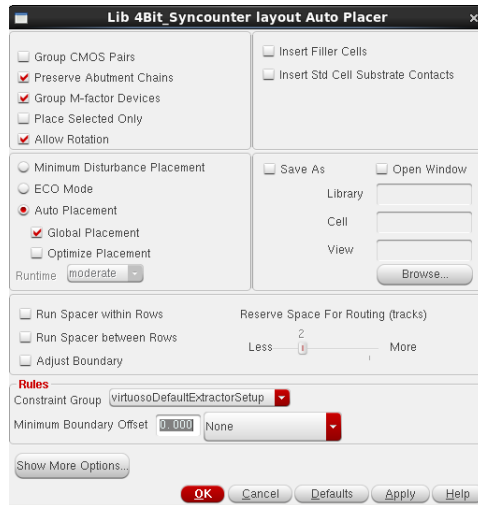


In the Placement Planning form, click on the **Calculate Rows** button at the bottom of the form. Once you do this, you should see rows in the Layout Editor form as shown below.

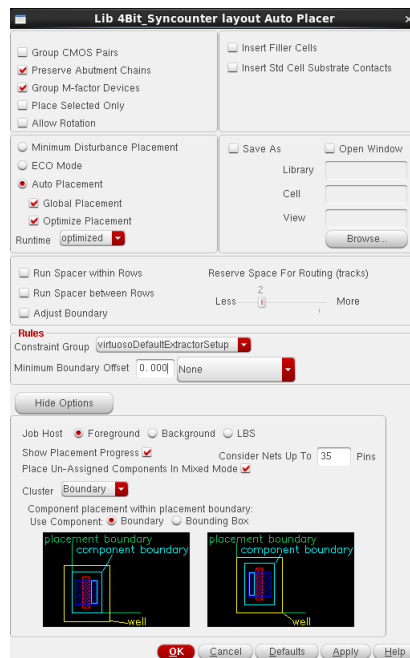


Close the **Placement Planning** form. Use the placer and automatically place the components inside the rows you just created.

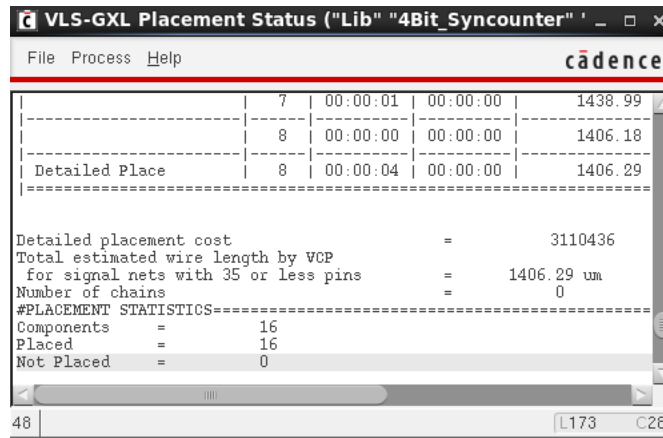
Go to the Layout Editor menu bar and select **Place** → **Custom Digital** → **Placer**. You should see a pop up menu. Your form should look like the one shown below.



In the above **Auto Placer** form, uncheck **Allow Rotation** then check **Optimize Placement** and Runtime to be **Optimized** under **Auto Placement**. Upon clicking on **Show More Options** button you will look some more details about the placer tool. Now click on the **Show Placement Progress** to see the progress while the placer tool automatically routes the components inside the rows. Your Placer form should look like the one shown below.

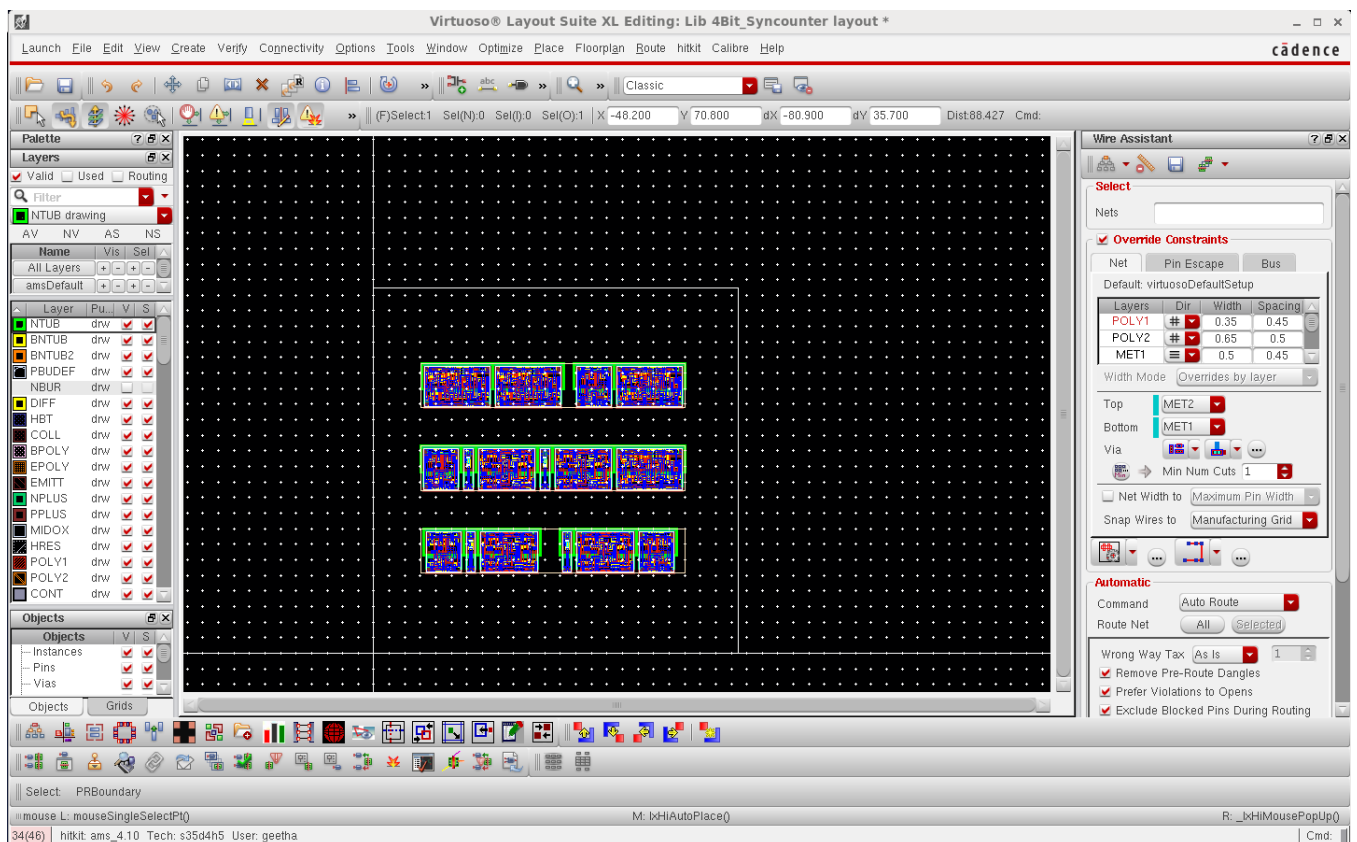


Click on "OK". You will see a pop menu called Placement Status like the one shown below.



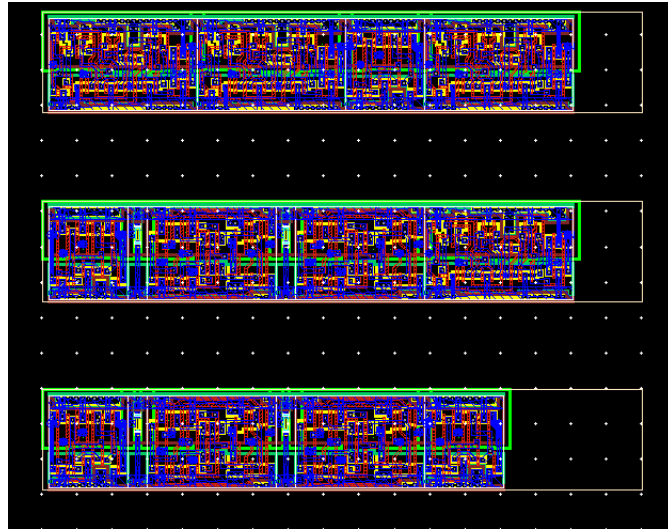
Wait for few seconds. Please be patient! This might take some time.

The Placer tool works on arranging the components and pins of the layout around the perimeter of the square box. The Placer tries all the possibilities and produces an optimized layout. Your Layout Editor window should look like the one shown below.



Select the standard cells one at a time and use the "move" command (short-cut key **m**) to move them close to one another in order to overlap or join their **vdd!** and **gnd!** rails. The

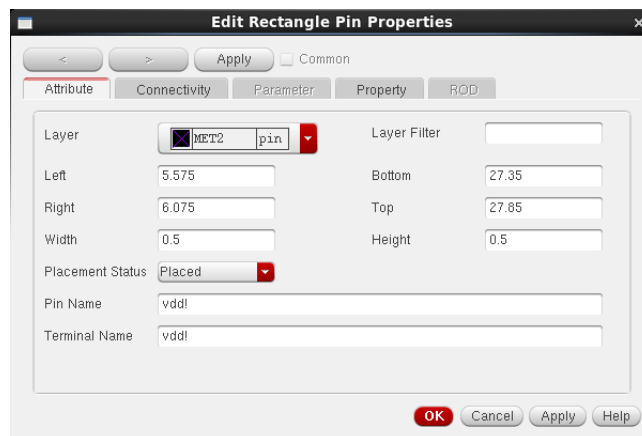
standard cells after auto placement should look like the one shown below.



We need to perform two different forms of routing: one is Pin to Trunk routing and the other is the Minimum Spanning Tree routing. As discussed in our introduction, the Minimum Spanning Tree routing is used for most of the connections. The algorithm is based on minimizing wire length. Pin to Trunk routing used for connecting supply rails using spine style routing.

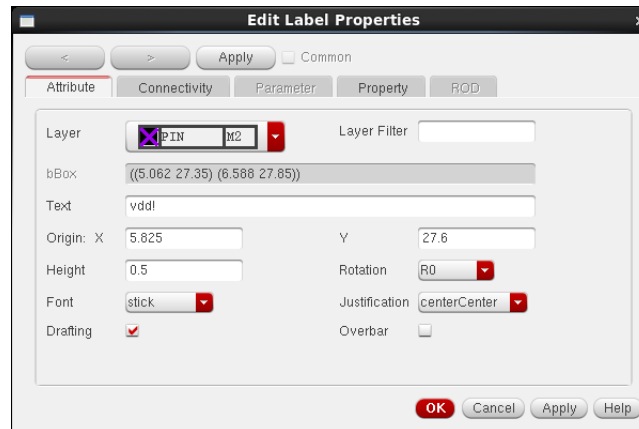
We start with Pin to Trunk routing. You can see that the **vdd!** pin is **metal1** when we generated our instances and pins as we choose to be. Now, we need to make the **vdd!** pin to be **metal2** pin.

To do this we need to edit the **vdd!** pin properties by using the short cut key:**q**. You will see a pop-up window of **vdd!** pin properties. Choose the Layer to be **MET2 | Pin** as shown in the form below.



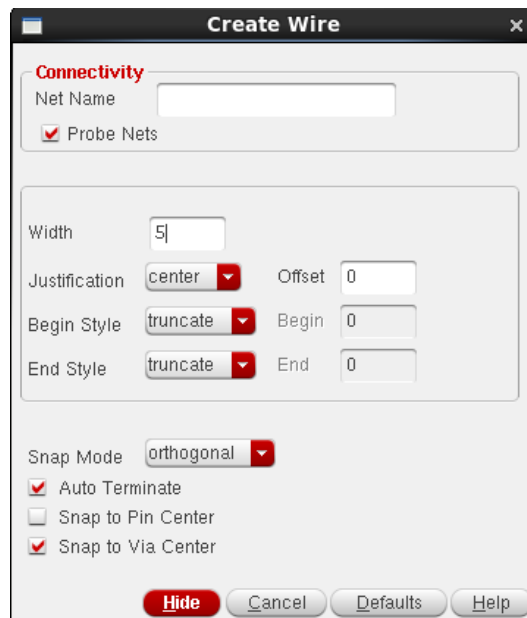
Click "OK".

Also, make it a **metal2** label. To do this we need to edit the **vdd!** label properties by selecting the label using the short cut key:**q**. You will see a pop-up menu of **vdd!** label properties. Then choose the Layer to be **PIN | M2** as shown in the form on the next page.



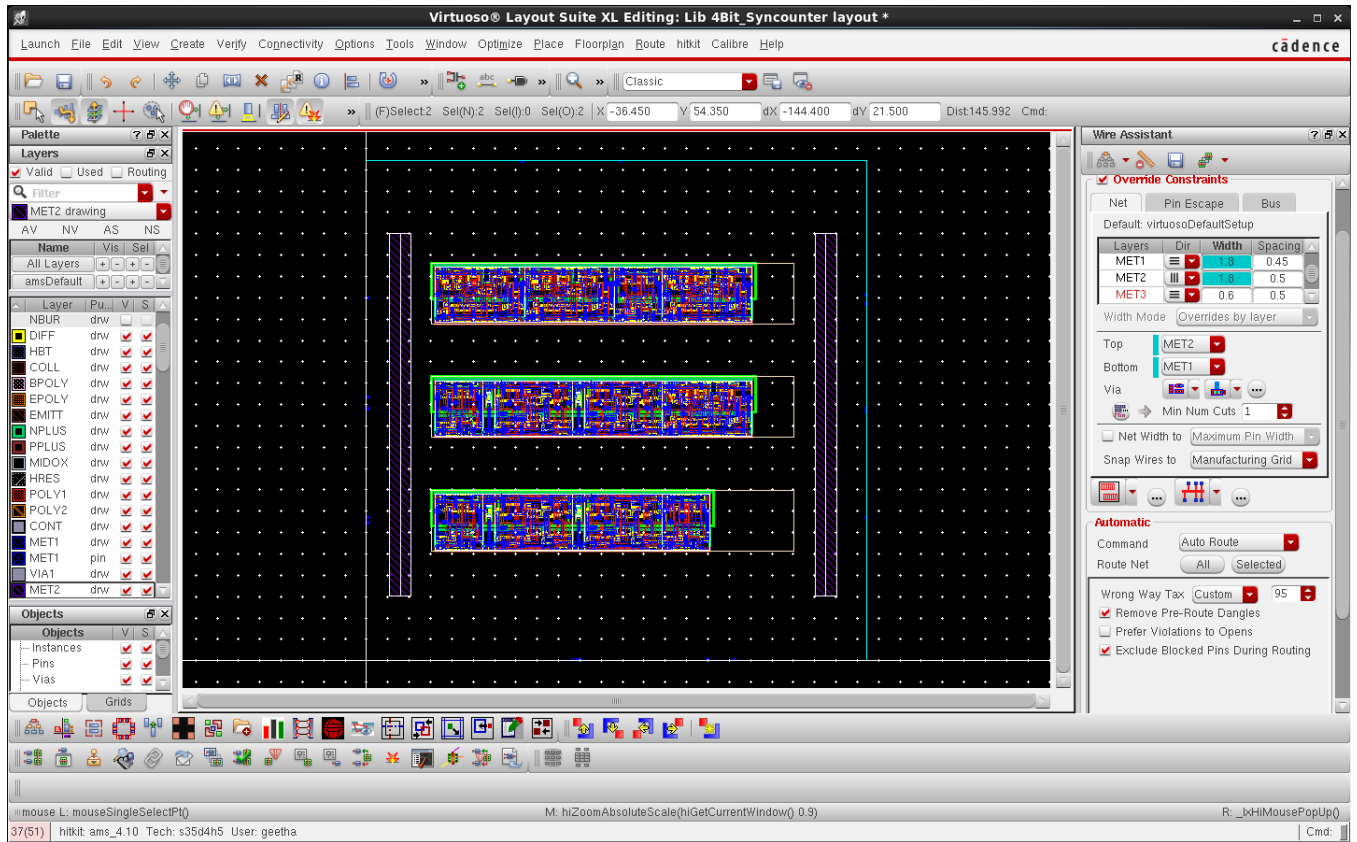
Click "OK".

We now need to create a wire to create the **vdd!** rail. Go to the Layout Editor menu bar and select **Create** → **Wiring** → **Wire**. You will see a pop up window of Create Wire and choose the width to be 5 i.e, 5um as shown in the below form.

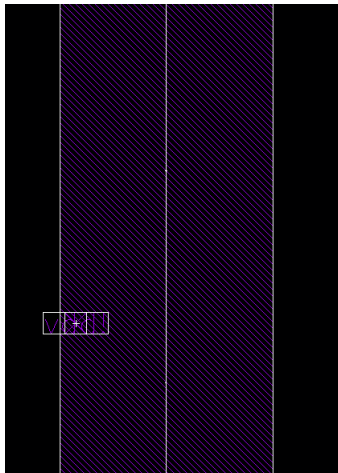


Then click on the edge of the Metal2 **vdd!** pin at the bottom of the layout and draw a wire.

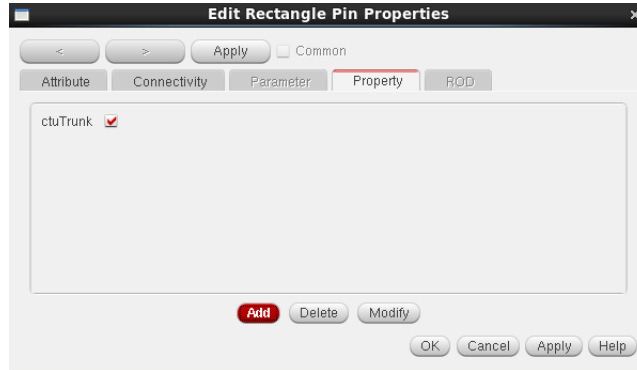
Repeat the steps as discussed above to make the **gnd!** pin and label **metal2** and also create a wire to form the **gnd!** rail. Once you do this your Layout Editor window should look like the one shown below.



Now, we need to make the **vdd!** pin, label, and the wire into a "trunk". Select the **vdd!** pin, label and the wire you created by holding shift key and click on the **RMB** (right mouse button) → **Compose Trunk** from the pulldown menu bar.



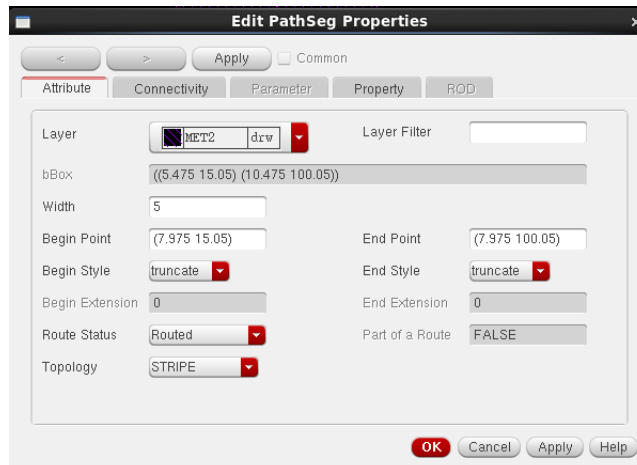
Once you do this, just cross-check the **vdd!** pin, label and the wire properties to make sure that they are trunks. Use the short cut key:**q** to look if the **vdd!** pin has the *ctuTrunk* property checked under the property tab as shown in the below form.



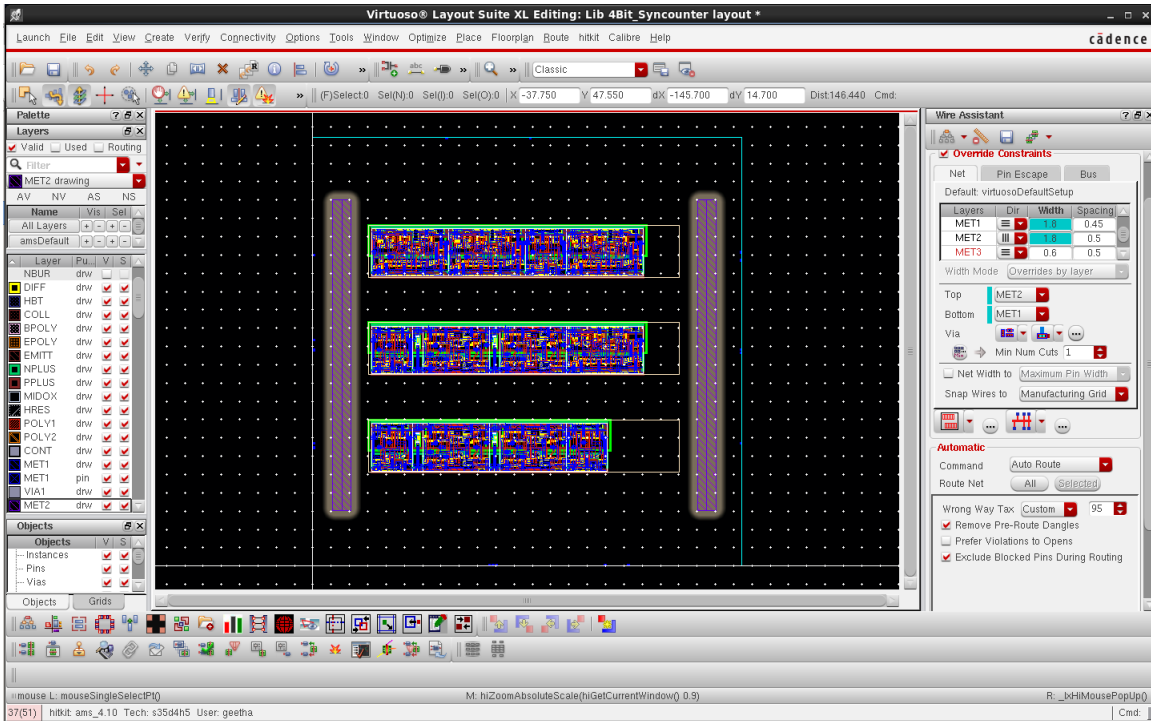
Select wire and use the short cut key:**q** to check if the wire drawn for vdd! has *1xStickyNet* property checked under the Property tab as shown below.



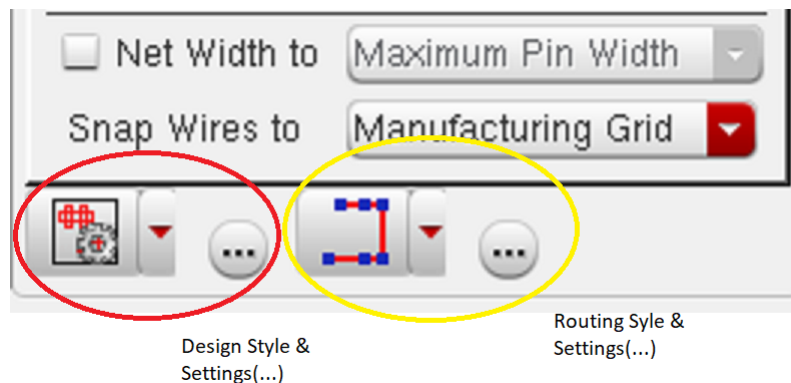
And you will also see that the topology is *STRIPE* under the Attribute tab as shown in the below form.



Repeat the same process discussed above to make the **gnd!** pin, label and wire a trunk. Once you do this you can test if your **vdd!** and **gnd!** rails, pins and labels are trunks, just select on the layout outside the Bounding Box and click on the **RMB (right mouse button)** → **Highligh All Trunks**. You will see that the trunks are highlighted on the Layout Editor window as shown below.



In your Layout Editor Window, we now configure the settings of the Pin to Trunk topology settings in the Wire Assistant. Edit the width of the MET1 and MET2 to 1.8 i.e 1.8 μm and then choose layers Top to be **MET2** and Bottom to be **MET1**. Then, Set Design Style to "ASIC", choose the routing style to be "Pin to Trunk" in the Wire Assistant. Choose the Wrong Way Tax to be *Custom* and give 95 for it. Your Wire Assistant form should look like the one shown on the next page.

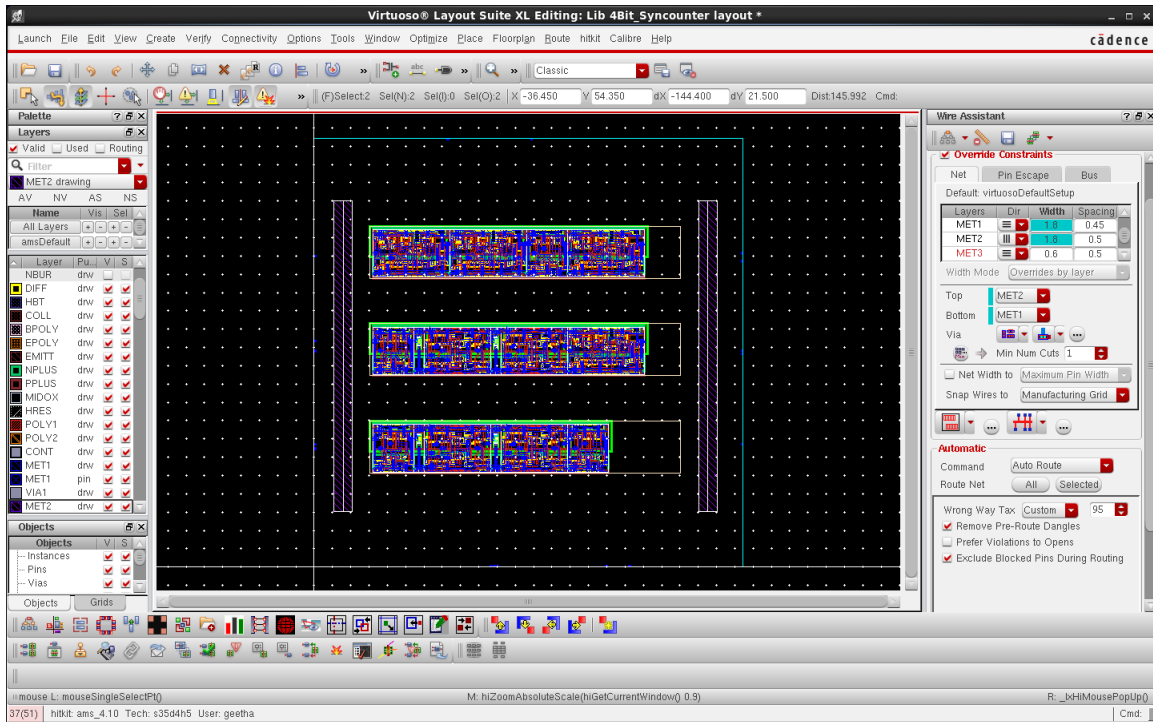


We need to do the topology setup in order to do pin to trunk routing. Check the *Pin to Trunks* option under Routing Steps. Choose Pins to be *All* and Ortho Pin Select Mode to be *All* under Pin Selection. You have to make sure that your Topology Setup form looks the same as the one shown below.

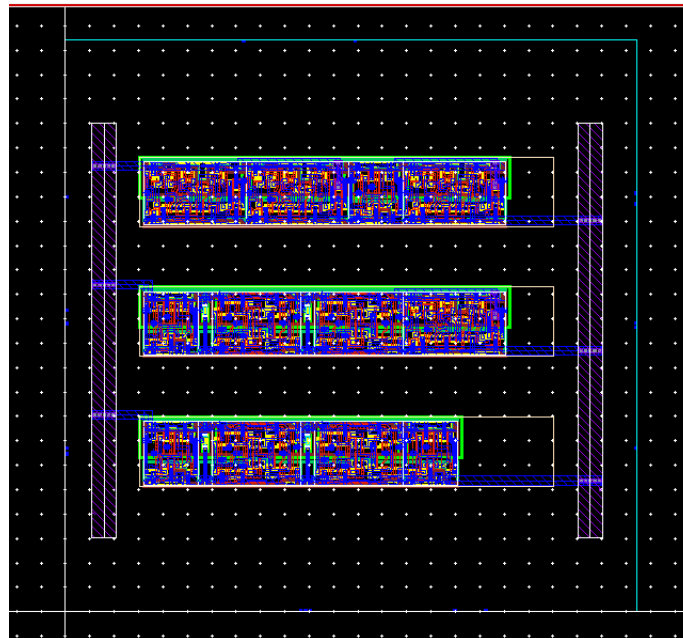


Click "OK".

Now, Select the **vdd!** and **gnd!** trunks and click on the **selected** button of Route Net in the Wire Assistant Window as shown below.



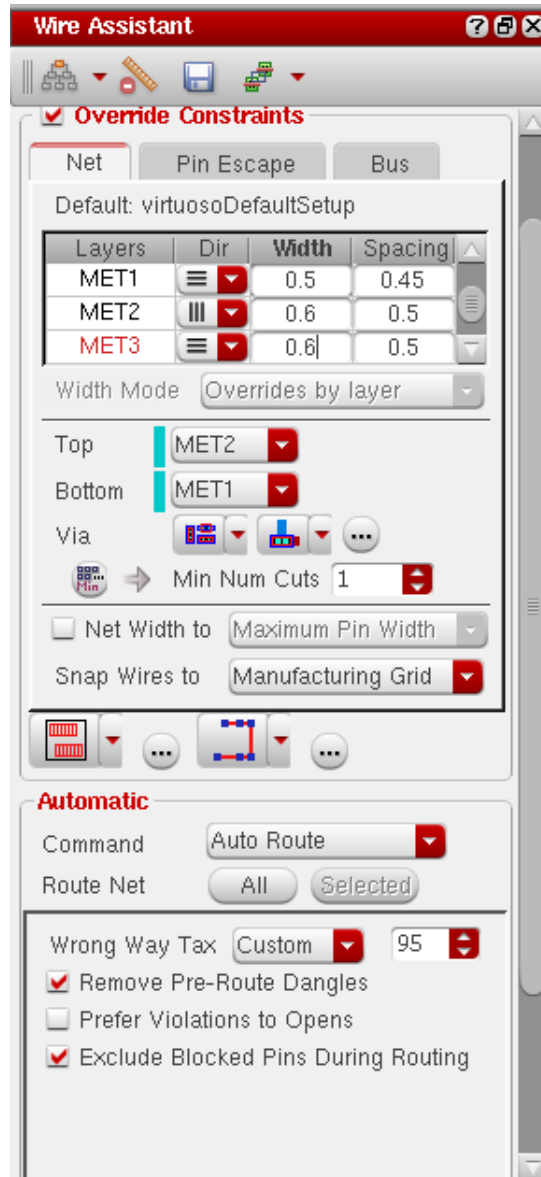
Once you do this you will see that the connections are made from the components to the selected trunks automatically as shown below.



So far, we have just made connections to the **vdd!** and **gnd!** trunks. Now we proceed further to make other connections between the components, as well as to the Input and Output pins. We make these connections automatically using the Minimum Spanning Tree topology. Before this you can choose to place/arrange your pins on all the four sides or just on

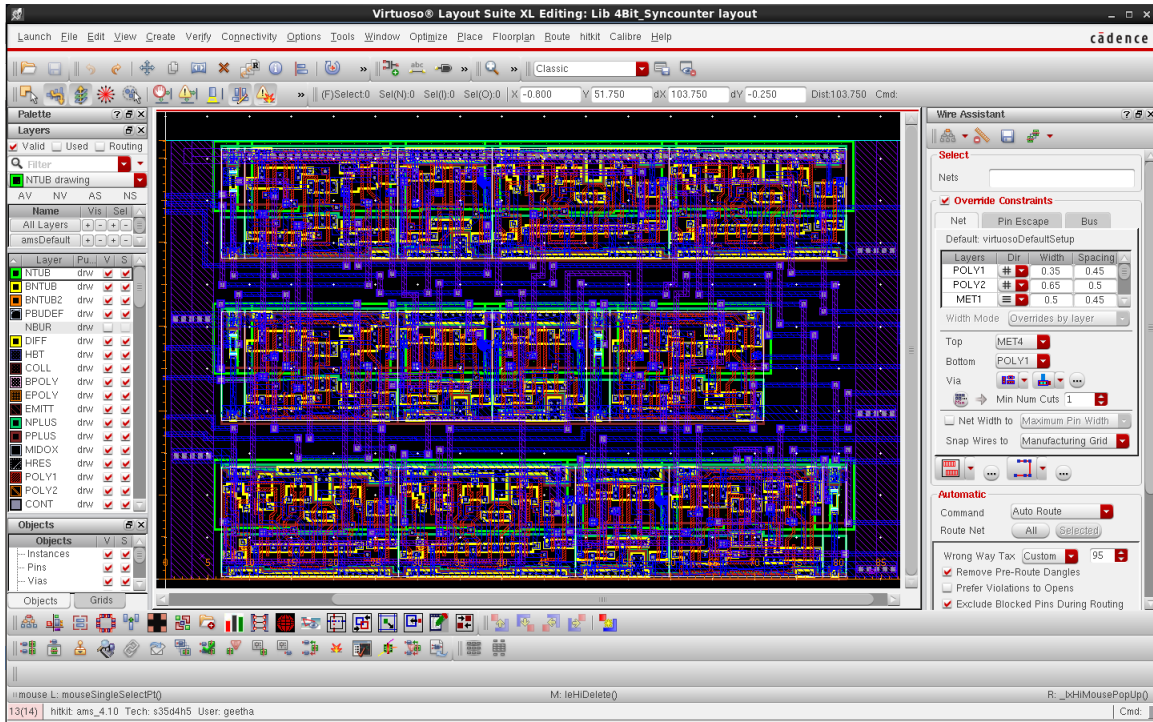
two sides. By default the automatic routing tool places pins on all four sides around your bounding box. In the layout shown below, pins are placed only on two sides besides the rails.

Edit the width of the MET1 and MET2 wires to default values i.e. MET1 to be $0.5 \mu m$ and MET2 to be $0.6 \mu m$. Then choose layers Top to be **MET2** and Bottom to be **MET1**. Set the Design Style to "ASIC" and choose the routing style to "Minimum Spanning Tree" in the Wire Assistant. Your Wire Assistant form should look like the one shown on the next page.

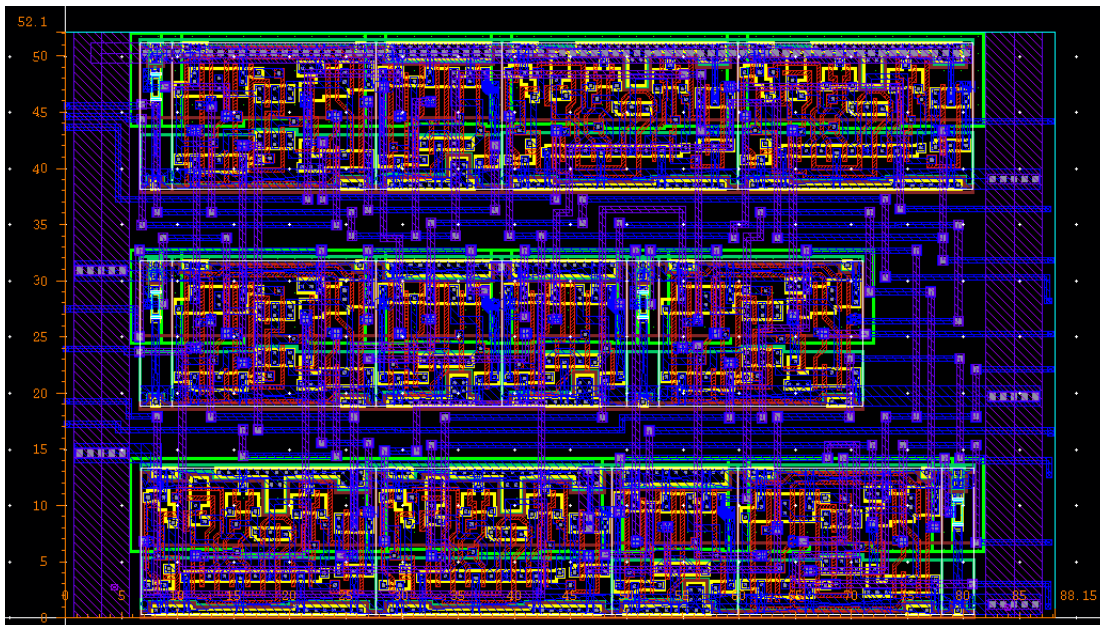


Now, click on the *All* button of Route Net to route the signal nets. Wait for a few seconds and you will see that the connections are made automatically based on the minimum spanning tree algorithm. Connections are spread through out the perimeter of the Bounding Box. One can constrain the connections closer (compact layout) by reducing the size of the

bounding box. Your Layout Editor window should look like the one in the figure shown below once all the connections in the layout are made.



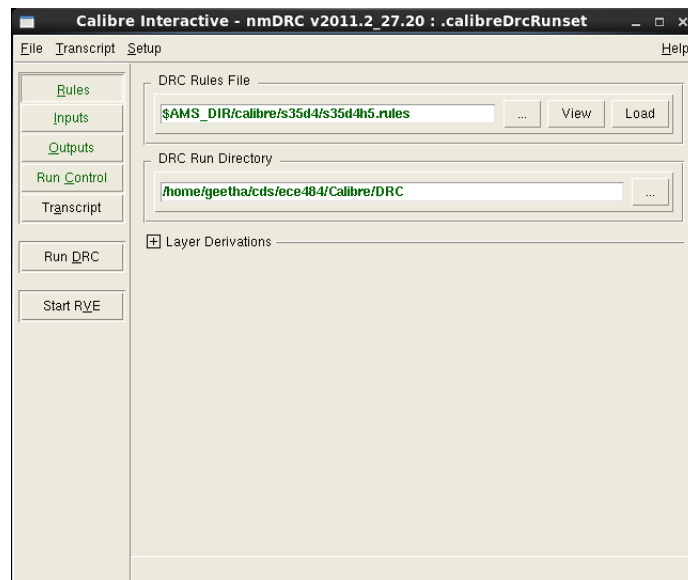
The finished layout will be similar to what is shown below. Don't forget to "Save" your layout



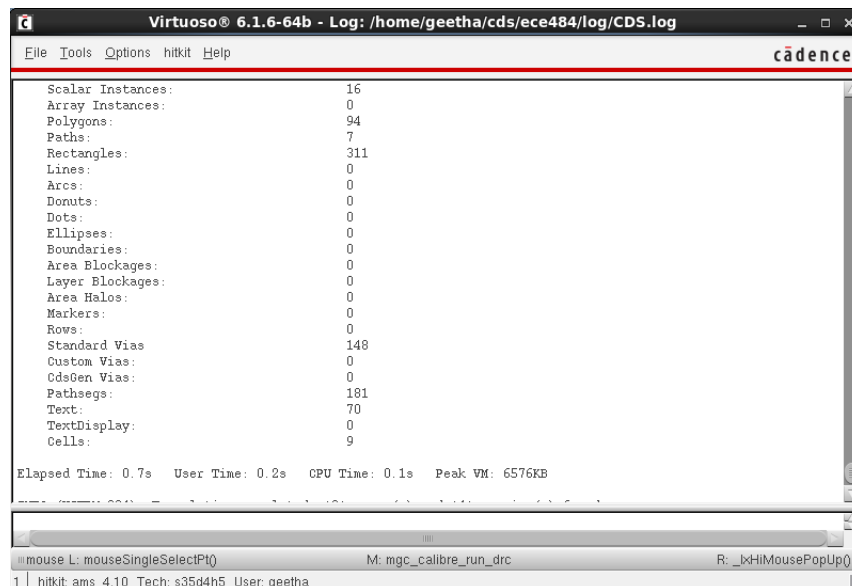
8.4.2 Design Rule Check (DRC)

Our next step is to perform a Design Rule Check, more commonly known as DRC, on the layout. DRC checks the layout against a set of defined design rules that must be satisfied. Despite of the fact that the designers might be aware of the design rules when performing the layout, there is a possibility of overlooking and thus violating the design rules. So, DRC is the step taken to ensure that the design can be manufactured correctly even if there may be misalignments during the fabrication.

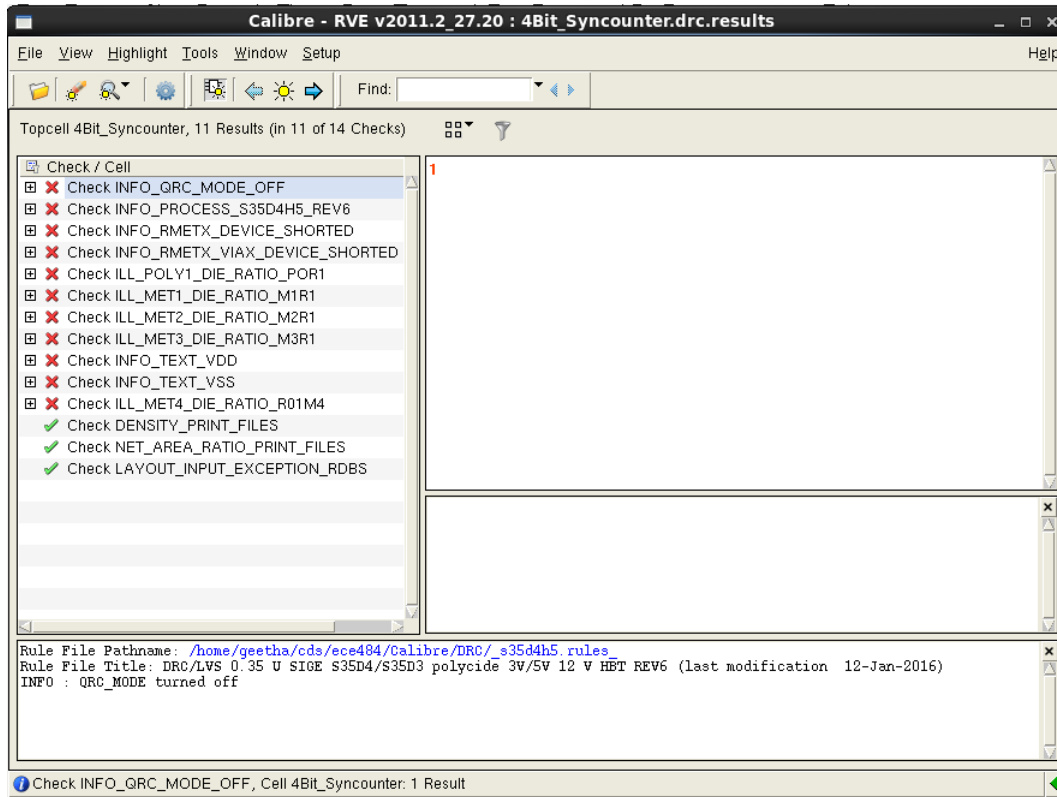
To run the DRC, choose Run DRC... from the Calibre menu in the layout view window. A pop-up menu, similar to one shown below will appear.



You need to make sure that you are in edit mode. Calibre then runs the DRC and reports the errors or warnings, if any, in the CIW window.



The CIW window above shows that there are no errors or warnings found in the DRC process.

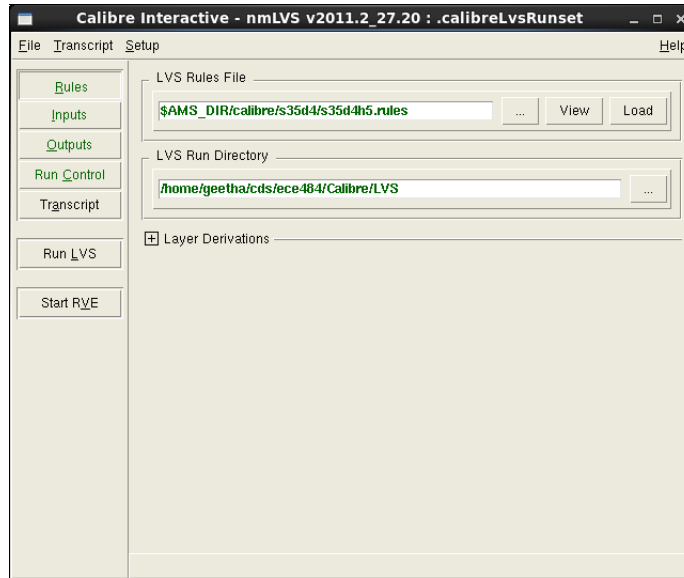


The Calibre DRC results window above shows the list of benign DRC errors which can be ignored if noticed during the DRC check. In other cases there can be errors, for example, like violating the defined spacing rules between the metal or poly layers. You can identify the errors indicated by markers on the layout. When performing huge layouts, the blinking markers might not be easily located at times. Fortunately, Cadence has an easy search tool. Under the Verify menu in the layout window, choose Markers → Find...

8.4.3 Layout Versus Schematic (LVS) Check

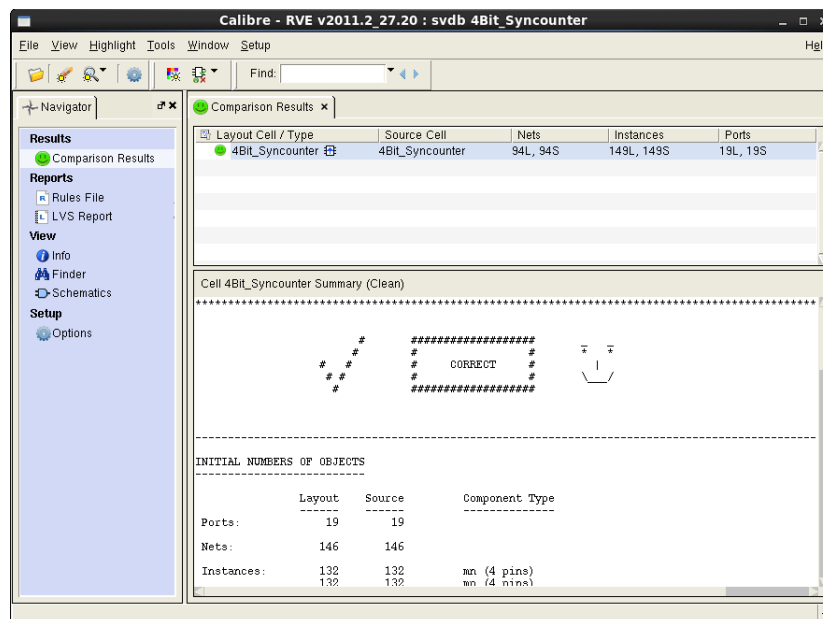
By now, we have generated an optimized layout automatically using the space based routing tool. The last step in our design process is LVS. LVS is a tool that compares the netlist extracted from the layout with the netlist generated from the schematic.

Choose Run LVS from the layout window... under the Calibre menu. A pop-up menu will appear, similar to one shown below.

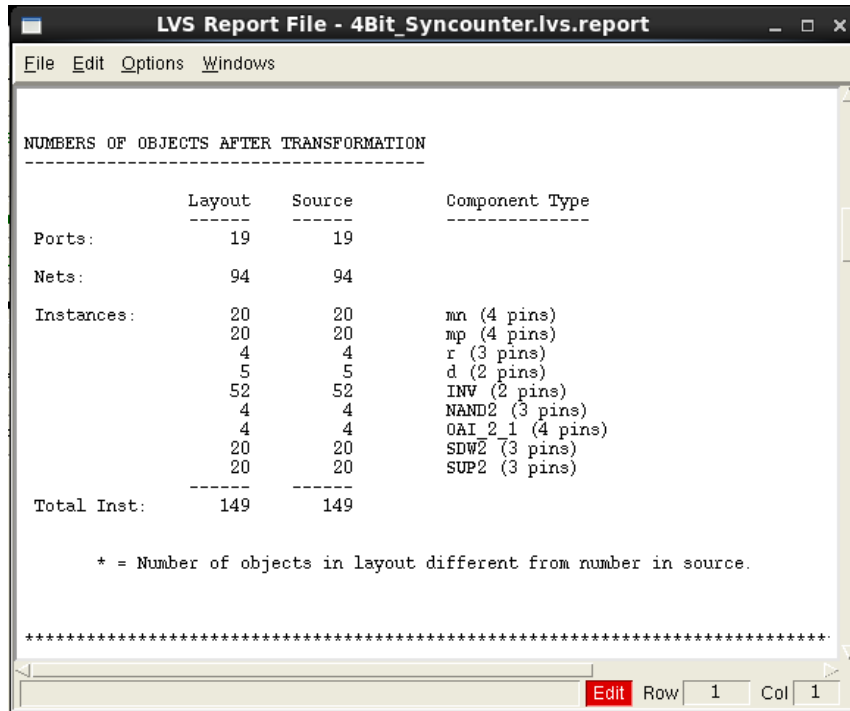


Click on the Run LVS button and wait. You will be shown the transcript file that is produced. Hit OK to "Overwrite" it. It may take a few seconds. Be patient!

A pop-up menu will then appear notifying you of the successful completion or failure of the LVS job. If the job is successfully completed, you will see the following RVE dialog box with a smiling face. If there are any errors, you will see a list of all LVS mismatches in a tree format. Click on the error to see what went wrong.



You should see the following message in the LVS report file showing the comparison between the layout and the cell schematic connectivities as shown below:



We see from the dialog box that there are no errors in the LVS comparison. If there are any errors, it will explain each of the terms in the above window in great detail. In most cases, there will be numerous errors reported in the lvs report and calibre form. Many of these are, in fact, related to each other. Fixing one of these errors, fixes many of the other errors. Focus on one error at a time, change the layout design appropriately, and repeat LVS steps until the layout and schematic perfectly match with each other.

9 Verilog Simulation

HDL/Verilog Simulation Tutorial

This is a revised tutorial for instructing students on how to carry out standard cell design flows.

The following Cadence CAD tools will be used in this tutorial:

NC-Sim for simulation.

Sim Vision for visualization.

Running the Cadence Simulation tools

First you need to log into a computer in the SCAD lab. Open up a terminal window and type

- `cds_ams`
- `cd $PHOME`
- `setup_edi`

We now need select the accumulator design. To do this, we type

- `sb accu`

Use the editor, `gedit`, to open the `env.tcl` file. Make sure that the simulation mode is set to **"rtl"**. If not, please change to mode to **"rtl"**. To do this, please type the command

- `gedit env.tcl`

We should now take a look at the Verilog description of our accumulator module. Please type the below commands now

- `cd $SRC/accu/design`
- `gedit accu.v`

You should see the following verilog code

```
// this is a simple 8-bit accumulator
```

```
`timescale 1ns/10ps
```

```
module accu (in, acc, clk, reset);
```

```
    input [7:0] in;
    input clk, reset;
    output [7:0] acc;
```

```
    reg [7:0] acc;
```

```
    always@(posedge clk) begin
        if(reset) acc<= 0;
        else acc<=acc+in;
    end
```

```
endmodule
```

Close the file and now let's go look at the testbench.

- cd \$SRC/accu/testbench
- gedit accu_tb.v

You should see the following testbench code...

```
// This is the testbench for our 8-bit accumulator
```

```
`timescale 1ns/10ps
```

```
module accu_tb;
```

```
    reg clk, reset;
    reg [7:0] in;
```

```
    wire [7:0] out;
```

```
    accu accu1(in, out, clk, reset);
```

```
    initial begin
        clk = 1'b0;
        forever begin
            #50 clk = ~clk;
        end
    end
```

```
end
```

```
initial begin
```

```
    #20000 $finish;
```

```
end
```

```
// Simulate the input signals
```

```
initial begin
```

```
    #0 reset<=1;
```

```
    in<= 8'd1;
```

```
    #1000 reset<=0;
```

```
end
```

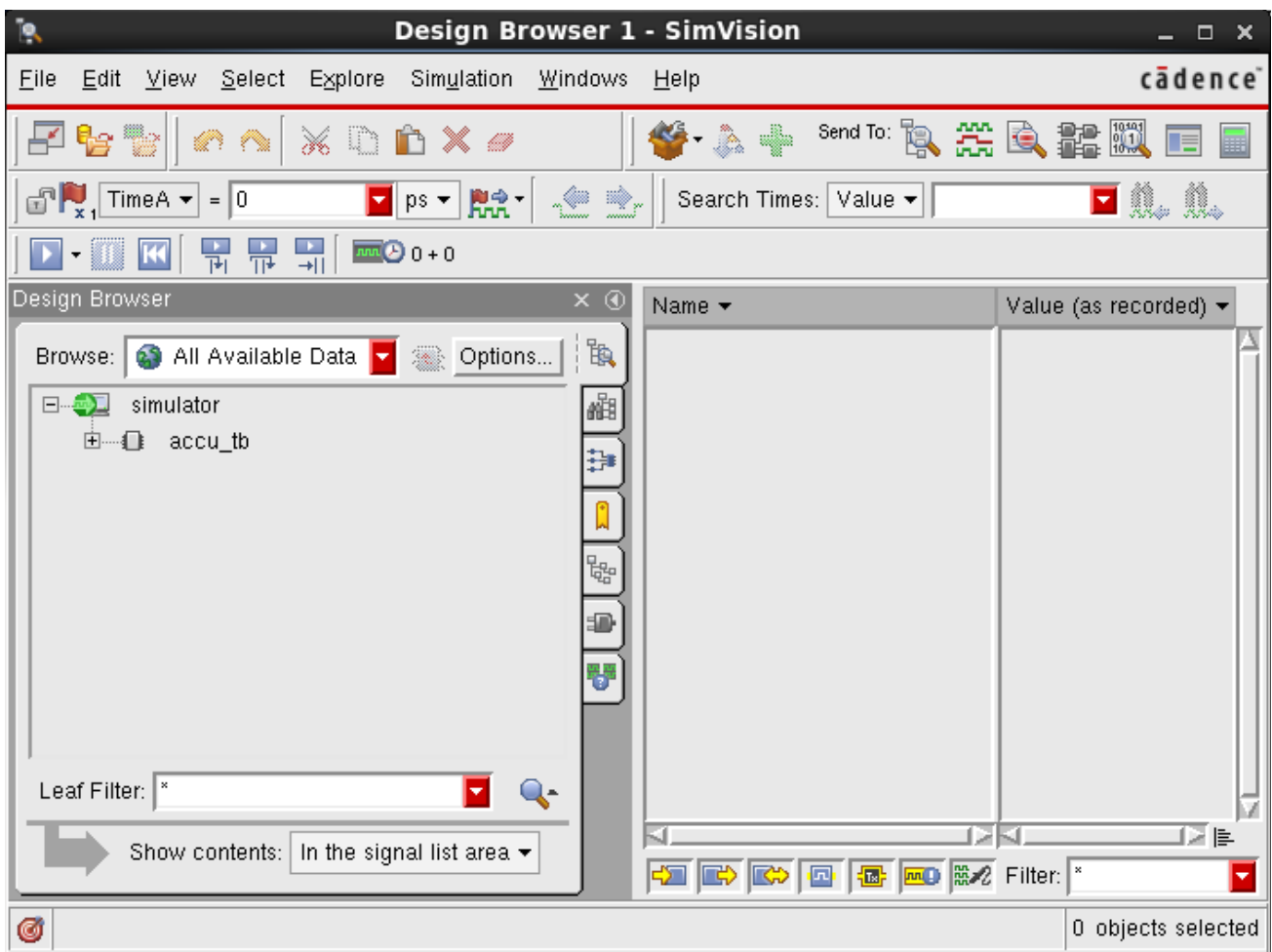
```
endmodule
```

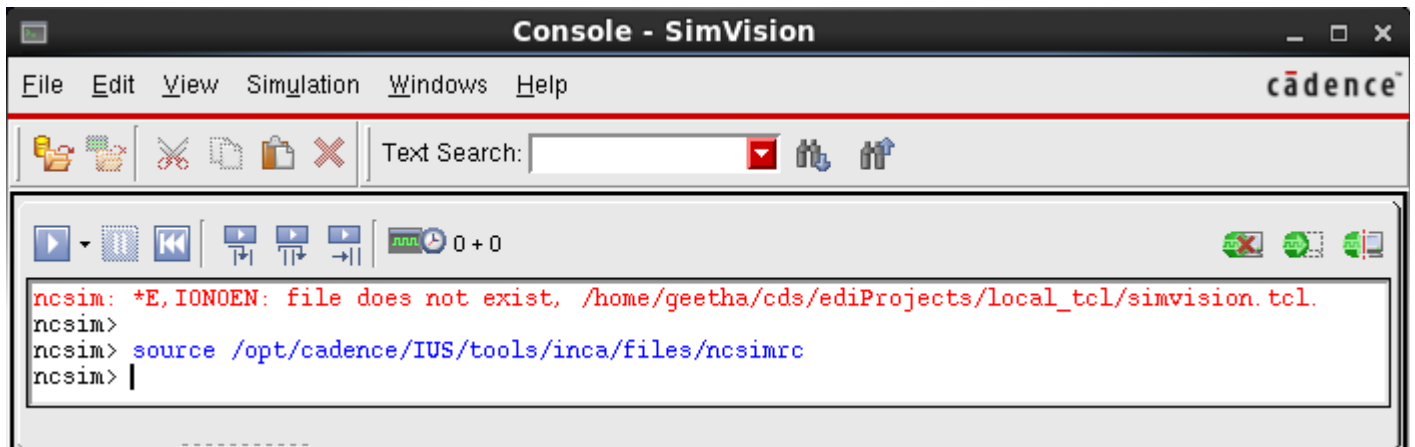
Close the file. We will now simulate the behavioral description of our accumulator design.

To run the simulator, we simply type

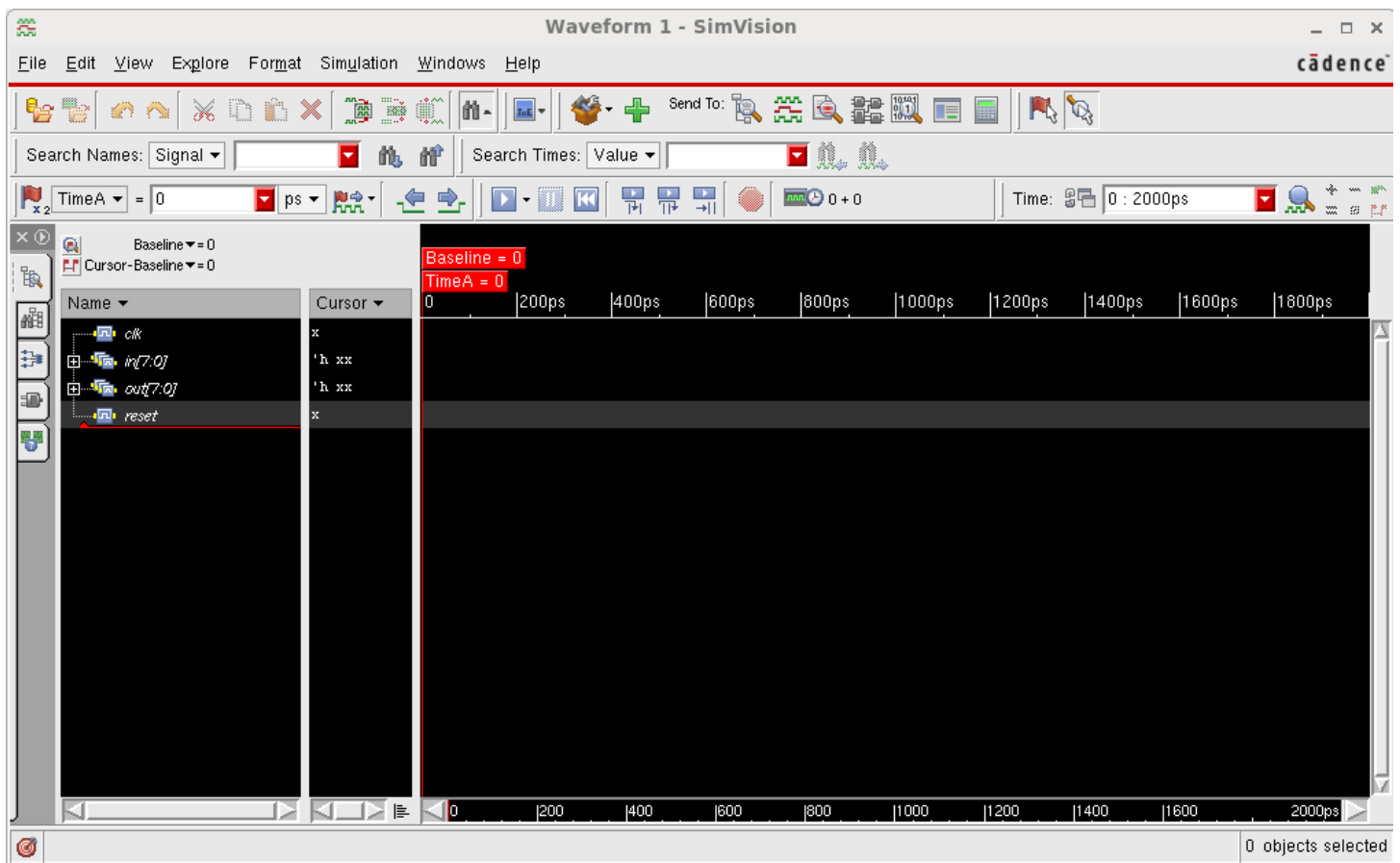
- cd \$PHOME
- sim

The following SimVision & Console windows should appear.

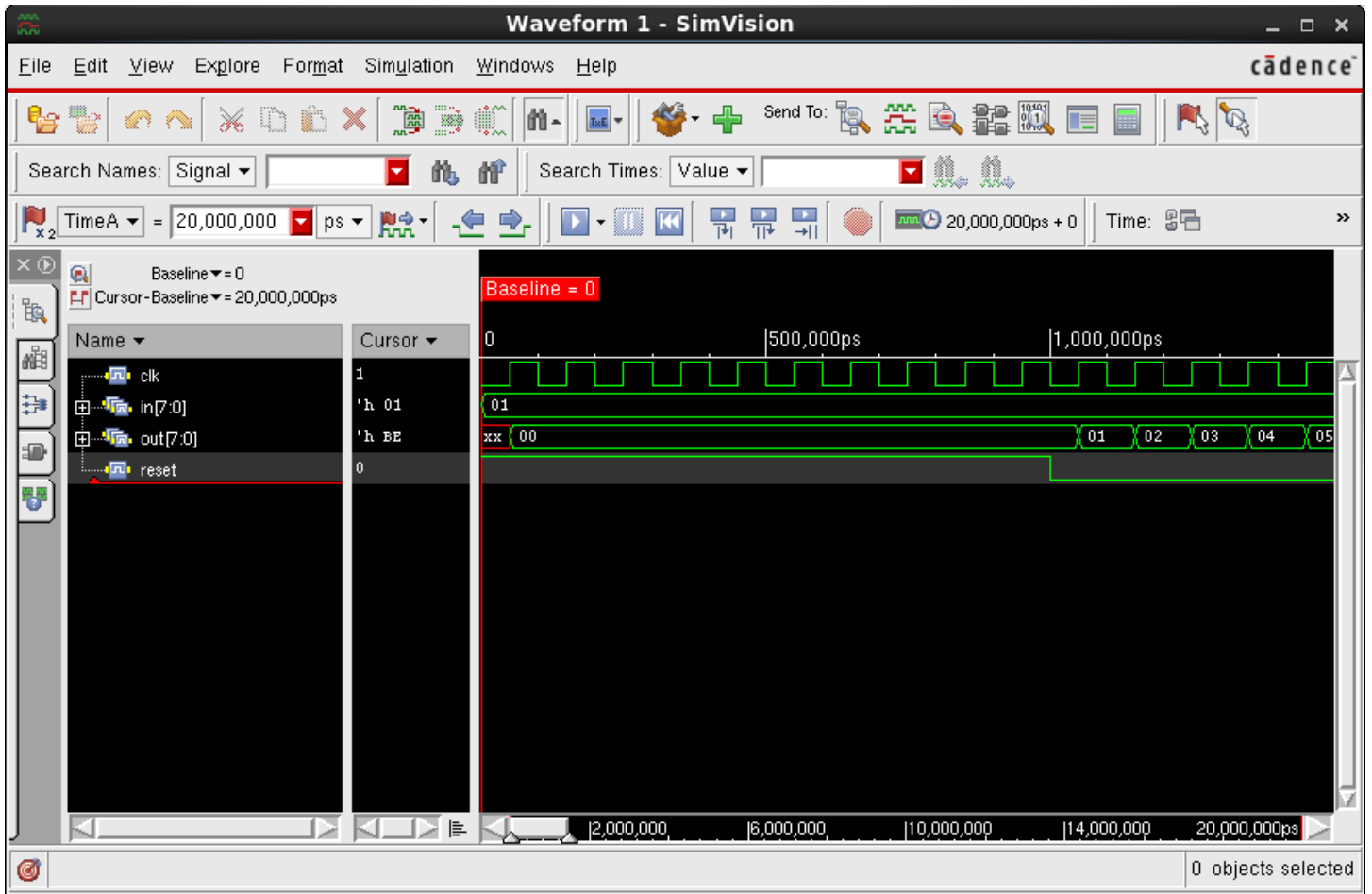




Click on the stimulus (accu_tb ... in our case) in the Simvision window. Now click on the Waveform button (the one that looks like a set of white digital waveforms on a black background, sixth from the right). This should open a new Waveform window as shown below. Now press on the Run button on the Design Browser window (Simulation → Run) or press F2 key (a short cut).



Then Zoom into the waveform if need, this should now display the signal waveforms that validate correct functionality for the accumulator (increment by one for every clock cycle). And when you zoom more to look at the output transition, you should see that the transition of the output from one value to another occurs at the positive edge of the clock, showing ideal behavior.



Congratulations! You have successfully simulated the behavioral description of our simple 8-bit accumulator. You may quit the Sim Vision now and proceed to synthesis and then on to place and route.

10 Logic Synthesis

RTL Logic Synthesis Tutorial

The following Cadence CAD tools will be used in this tutorial:

RTL Compiler Ultra for logic synthesis.

You must complete the Simulation Tutorial before doing this new tutorial.

Running the Cadence logic synthesis tools

First you need to login to a computer in the SCAD lab. We then open up a terminal window and issue the following commands

- `cds_ams`
- `cd $PHOME`
- `sb accu`

Before we can synthesize our design we need to prepare a SDC file. SDC stands for Synopsys Design Constraints. We need, for example, to make sure that the tool knows how fast the design is to operate. Let's view the SDC file that Dr. Engel wrote.

- `gedit $SRC/sdc/accu.sdc`

Notice that in the sdc file we tell the tool that the clock has a 50% duty cycle and a period of 100 ns.

In addition to the SDC file we also need an I/O pin assignments before we can place and route the design. Let's view env file.

- `cd $PHOME`
- `gedit env.tcl`

Notice that we are instructing the tool to place the input pins on the "North" side of the chip while the outputs should be available on the "South" side of the chip. Where the "clk" on the "East" side and the "reset" on the "West" side. We can also choose the Aspect ratio, metal layers in addition to the pin assignments as shown in the figure on the next page.

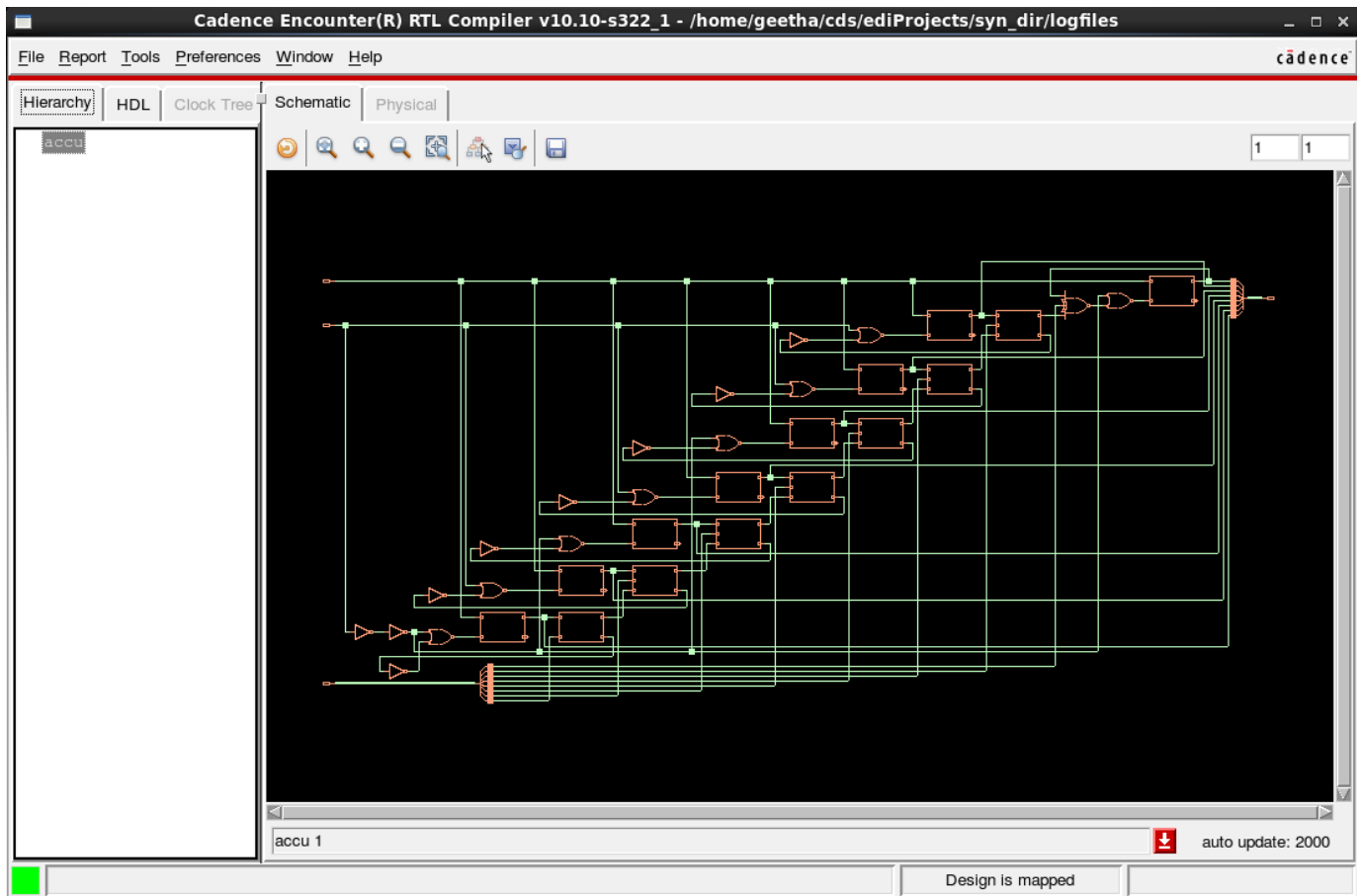
To launch the **RTL Compiler** simulator, type the command:

- `syn`

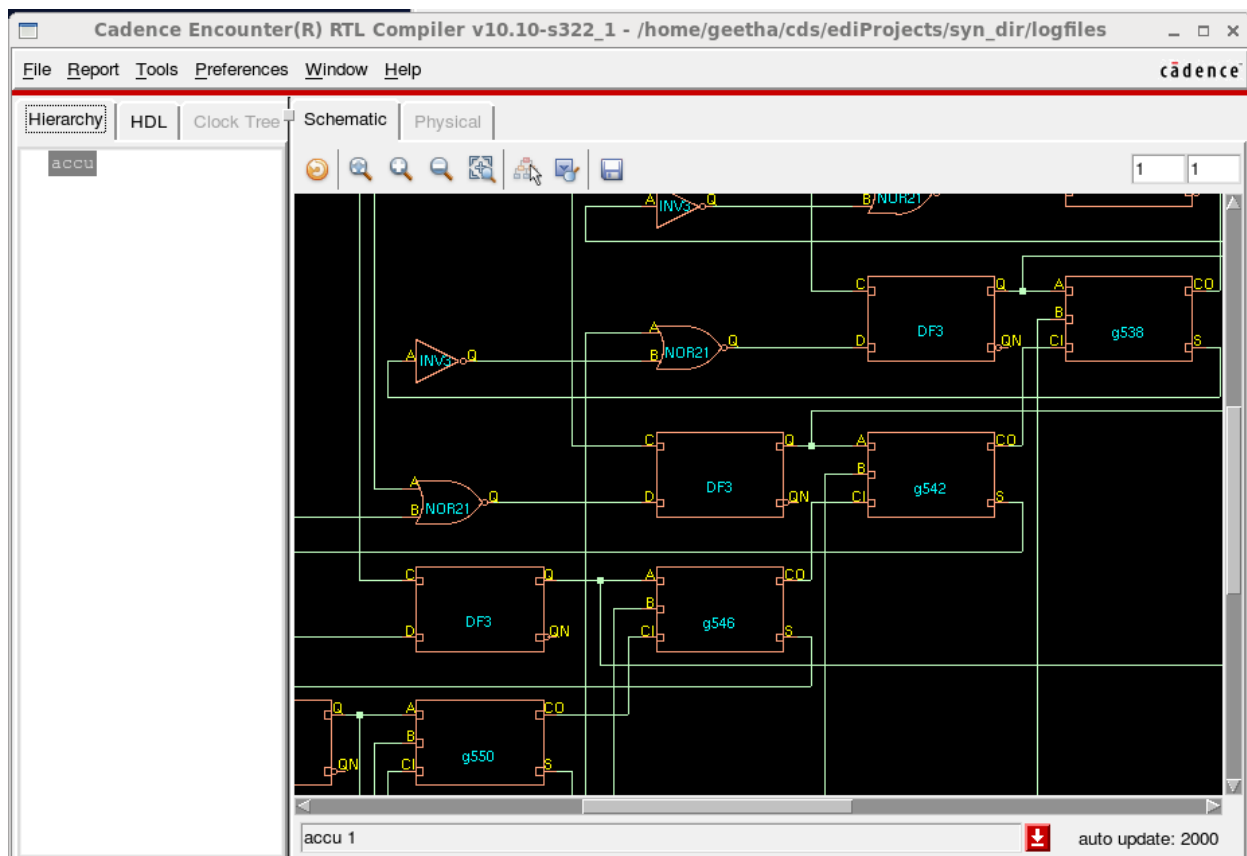
```
env.tcl x
78 # ~~~~~
79 # Floorplanning
80 # ~~~~~
81
82 # Provide X and Y dimensions of the core
83
84 set CORE_X 1000
85 set CORE_Y 2000
86
87 # Set the aspect ratio for the layout
88 # A values less than 1.0 means wide and not so high!
89
90 set ASPECT 0.25
91
92 # Establish a boundary outside of the core area
93
94 set CORE_TO_BOUND 15
95
96 # Utilization
97
98 set UTILIZATION 0.6
99
100 # Pin assignments
101
102 set N_PINS {acc[0] acc[1] acc[2] acc[3] acc[4] acc[5] acc[6] acc[7]}
103 set S_PINS {in[0] in[1] in[2] in[3] in[4] in[5] in[6] in[7]}
104 set E_PINS {clk}
105 set W_PINS {reset}
106
107 # Spacing in microns between the pins
108
109 set N_SPACING 10
110 set S_SPACING 10
111 set E_SPACING 10
112 set W_SPACING 10
113
114 # Metal layer that should be used
115
116 set N_LAYER 2
117 set S_LAYER 2
118 set E_LAYER 3
119 set W_LAYER 2
```

The command `syn` starts RTL Compiler and runs a synthesis script written by Dr. Engel and his former graduate student, James Ziebold. The script will pause to allow the user to make sure that the tool understands all of the SDCs. Make sure there are 0 failures. If so, then type “resume” to continue as the script instructs.

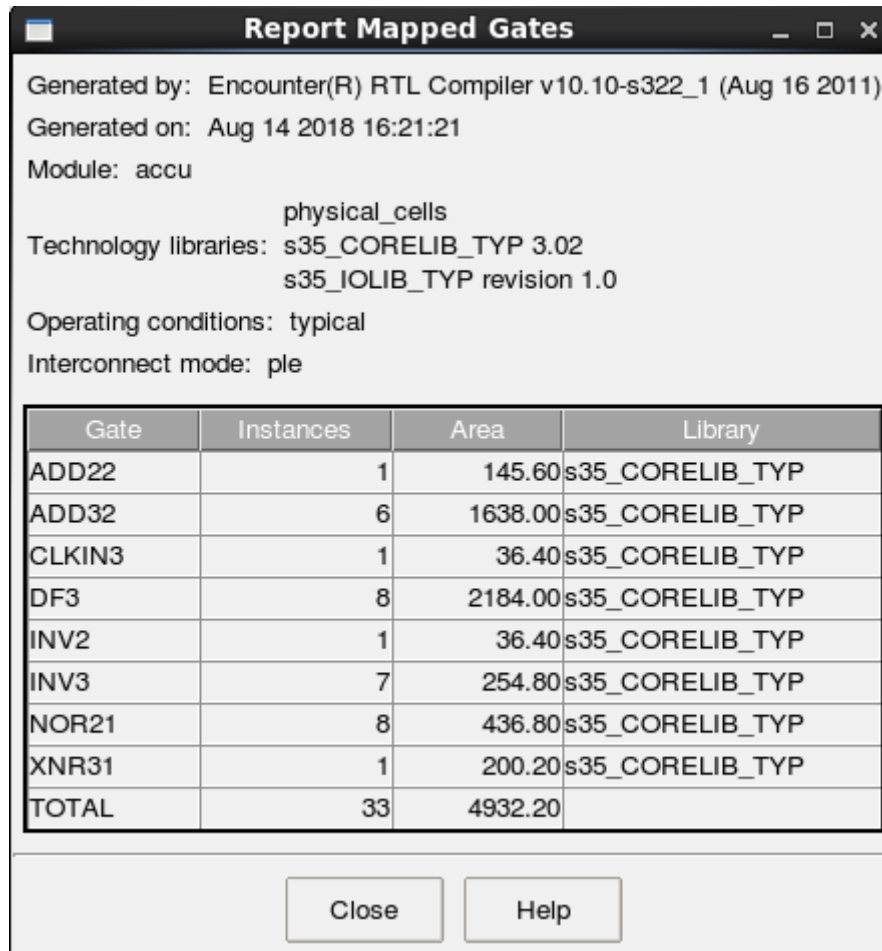
After some time, a schematic window will pop up and you can double click on the `accu` entry to view the schematic. You should see the following:



If you zoom in, then one might see something like



Try some of the menu options, for example you can see what library standard cells were used in the synthesized netlist by going to **Report -> Netlist -> Mapped Gates...**:



Report Mapped Gates

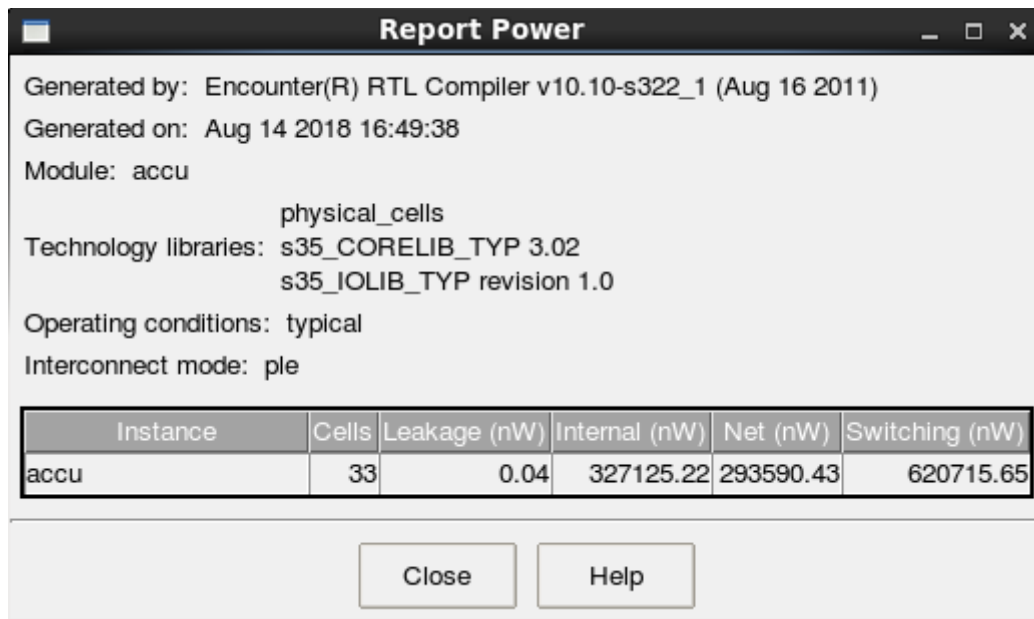
Generated by: Encounter(R) RTL Compiler v10.10-s322_1 (Aug 16 2011)
Generated on: Aug 14 2018 16:21:21
Module: accu

physical_cells
Technology libraries: s35_CORELIB_TYP 3.02
s35_IOLIB_TYP revision 1.0
Operating conditions: typical
Interconnect mode: ple

Gate	Instances	Area	Library
ADD22	1	145.60	s35_CORELIB_TYP
ADD32	6	1638.00	s35_CORELIB_TYP
CLKIN3	1	36.40	s35_CORELIB_TYP
DF3	8	2184.00	s35_CORELIB_TYP
INV2	1	36.40	s35_CORELIB_TYP
INV3	7	254.80	s35_CORELIB_TYP
NOR21	8	436.80	s35_CORELIB_TYP
XNR31	1	200.20	s35_CORELIB_TYP
TOTAL	33	4932.20	

Close Help

In a similar manner, you can view the power report: **Report -> Power -> Detailed Report**. You will see a window popped up asking for Depth and Minimum Count, choose depth to be 100 and minimum count to be 0 then click OK. “Leakage” power is what we call static power and “Switching” power is what we call dynamic power.



Report Power

Generated by: Encounter(R) RTL Compiler v10.10-s322_1 (Aug 16 2011)
Generated on: Aug 14 2018 16:49:38
Module: accu

physical_cells
Technology libraries: s35_CORELIB_TYP 3.02
s35_IOLIB_TYP revision 1.0
Operating conditions: typical
Interconnect mode: ple

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
accu	33	0.04	327125.22	293590.43	620715.65

Close Help

You can also view the timing report: **Report -> Timing -> Worst Path**. The tool is showing you the slowest path. Every other delay path is shorter. The important value is the 91541 ps “Slack” value you see in the report. This means that the delay associated with this path could have been 91541 ps longer and the design would still function correctly! Or this means that we could operate the accumulator at a significantly higher clock frequency. As long as you have positive slack the design will function correctly. Negative slack is VERY BAD. The design will fail.

The screenshot shows a 'Detailed Timing Report' window. At the top, the endpoint is 'acc_reg[7]/D'. The table below shows the following data:

Endpoint	Slack (ps)	Rise Slew (ps)	Fall Slew (ps)
acc_reg[7]/D	91541	521	375

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
(clock clk)	launch					50000.00	R
(accu.sdc_line_28_7_1)	ext delay				5000.00	55000.00	F
in[0]	in port	1	40.30	0.00	0.00	55000.00	F
g554/B					0.60	55000.60	
g554/CO	ADD22	1	50.30	317.40	365.50	55366.10	F
g550/CI					0.80	55366.90	
g550/CO	ADD32	1	50.30	325.40	411.30	55778.20	F
g546/CI					0.80	55779.00	
g546/CO	ADD32	1	50.30	325.60	412.60	56191.60	F

Below the table is a circuit diagram showing a chain of logic gates: in[0] -> g554 -> g550 -> g546 -> g542 -> g538 -> g534 -> g530 -> g528 -> g526 -> acc_reg[7]. A progress bar above the diagram shows the current path highlighted in blue and green.

Congratulations, this is the end of the Logic Synthesis Tutorial. Exit the Synthesis tools.

Assignment: Post-synthesis simulation

Now that you are done with the synthesis, it's time to simulate the accumulator using the design as implemented using cells from the 0.35um standard cell library.

- `gedit env.tcl`

Change the simulation mode from "rtl" to "syn". Save and quit the editor. Then type

- `sdf`

Dr. Engel's "sdf" script will create another testbench, similar to the one we looked at before but will attach the SDF (Standard Delay Format) file which the RTL Compiler created. The `$sdf_annotate()` system task attaches the delay information.

In order to view the new testbench, type

- `gedit $SRC/accu/testbench/accu_syn_tb.v`

To view the netlist type

- `gedit $PHOME/syn_dir/netlists/accu_syn.v`

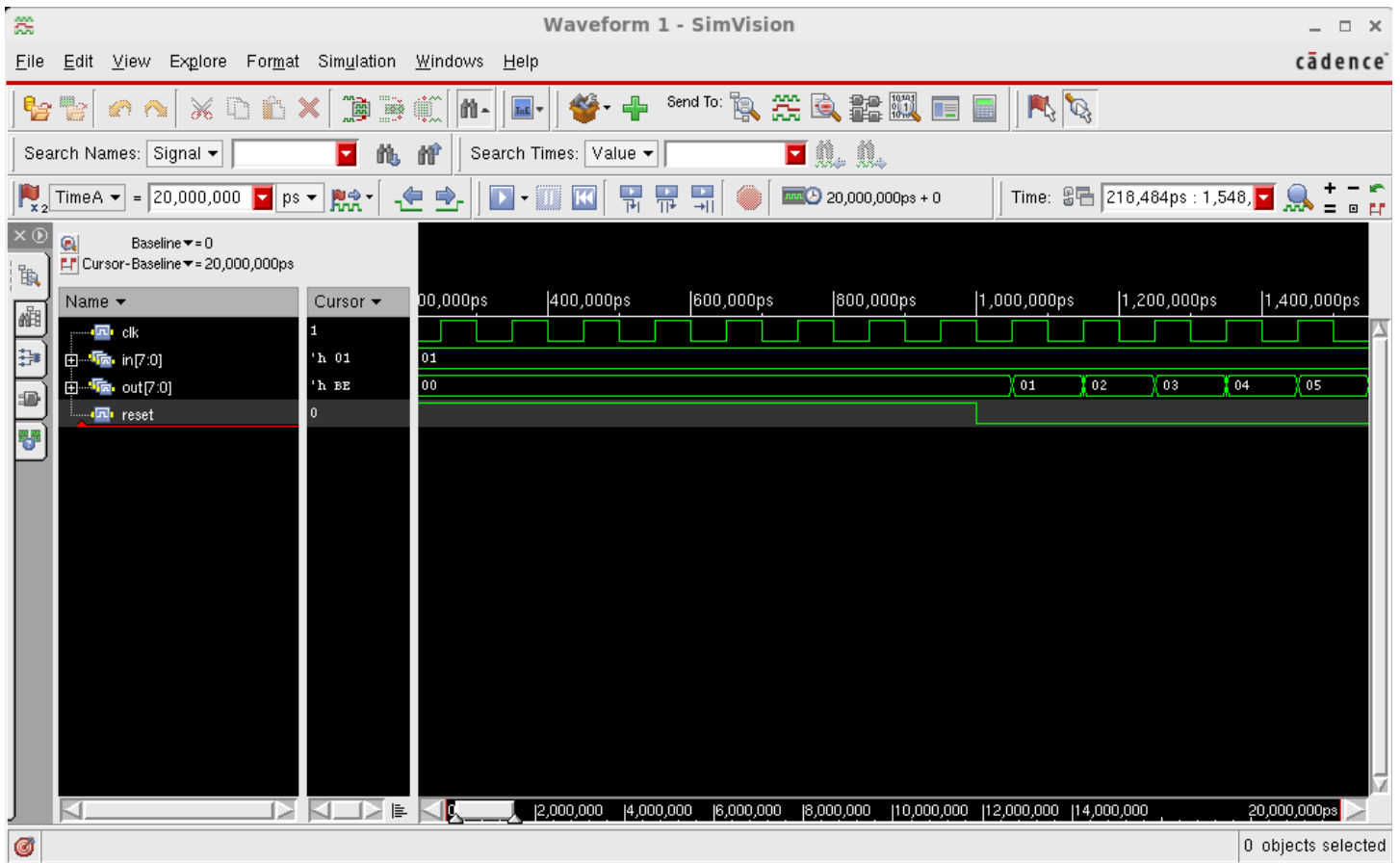
To view the sdf file

- `gedit $PHOME/sim_dir/sdf/accu_syn.sdf`

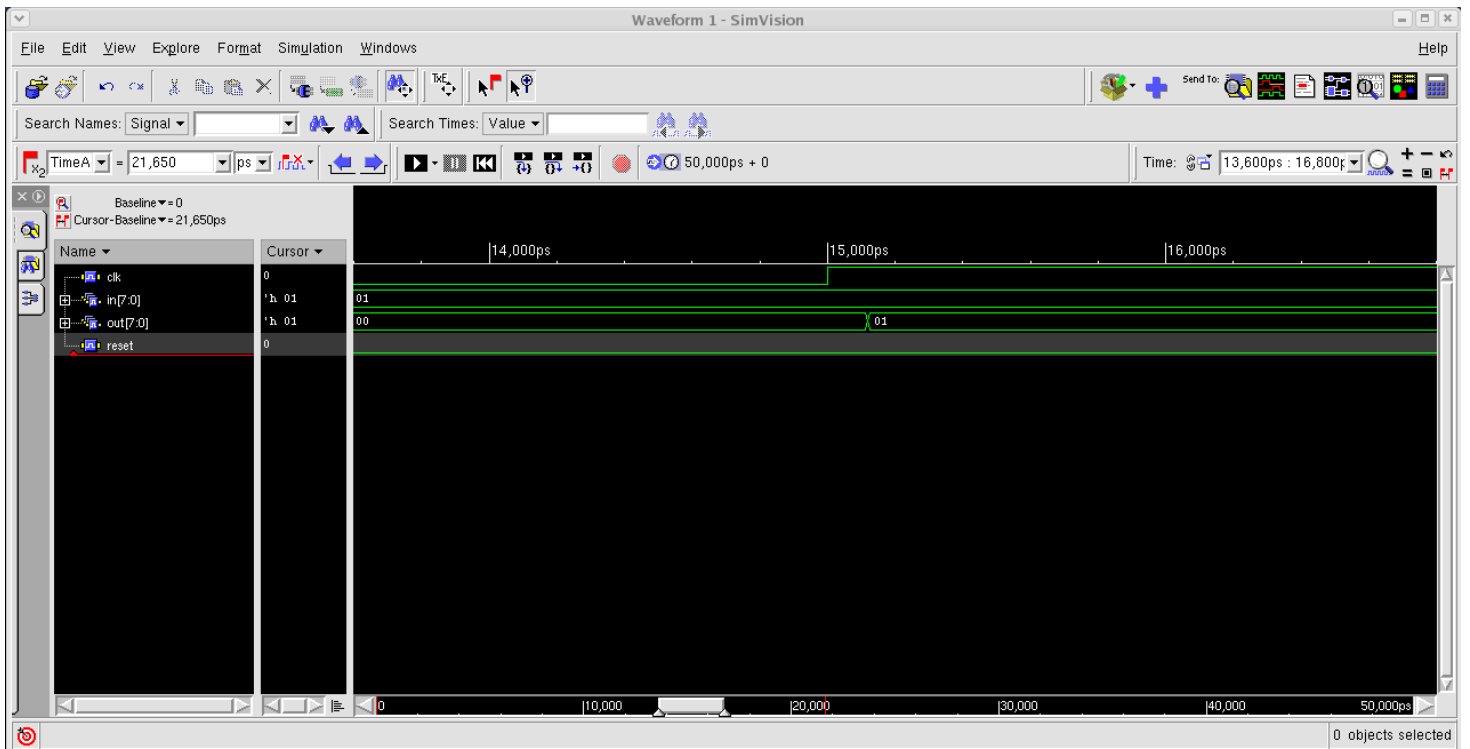
Now when we simulate the design, we no longer are simulating a Verilog RTL description but a Verilog netlist that is describing the connection of standard cells we just observed in the schematic. Moreover, the delay of these gates is accurately modeled in the SDF file which will also be used. To re-run the simulation using the netlist and SDF file we type again (but remember we set the mode to "syn")

- `sim`

The signal waveforms display should look like this:



Notice that if you zoom in the waveforms, you will see that the transition of the output from one value to another doesn't occur at the positive edge of the clock, this delay is introduced by the cells.



11 Place and Route

Standard Cell Place and Route Tutorial

This tutorial instructs students on how to use the Cadence Standard Cell Place and Route tool. **The following Cadence CAD tools will be used in this tutorial:**

SOC Encounter

You may want to revisit the Simulation Tutorial and the Logic Synthesis Tutorial before doing this new tutorial.

Running the Cadence place and route tools

First you need to log into a computer in the SCAD lab. Issue the following commands

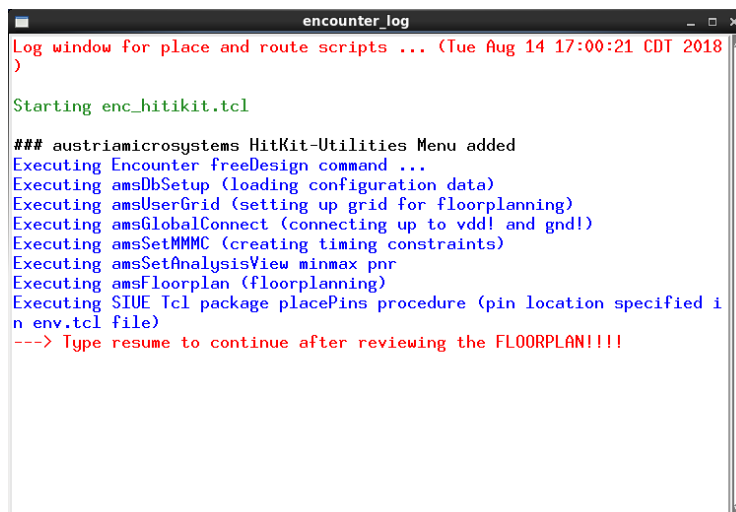
- cds_ams
- cd \$PHOME
- sb accu

To launch SoC (System on Chip) Encounter, type

- pnr

The “pnr” script was written by Dr. Engel and his graduate students. It calls other “tcl” scripts. For each step in the place and route flow described in class, there is a script which performs that specific task. The “encounter_log” window will keep you informed as to exactly what the tool is doing. Pay careful attention to what is being printed to this window. As the pnr script calls multiple steps while placing and routing, please pay careful attention to encounter_log window to learn step by step approach about how the script does place and route ?

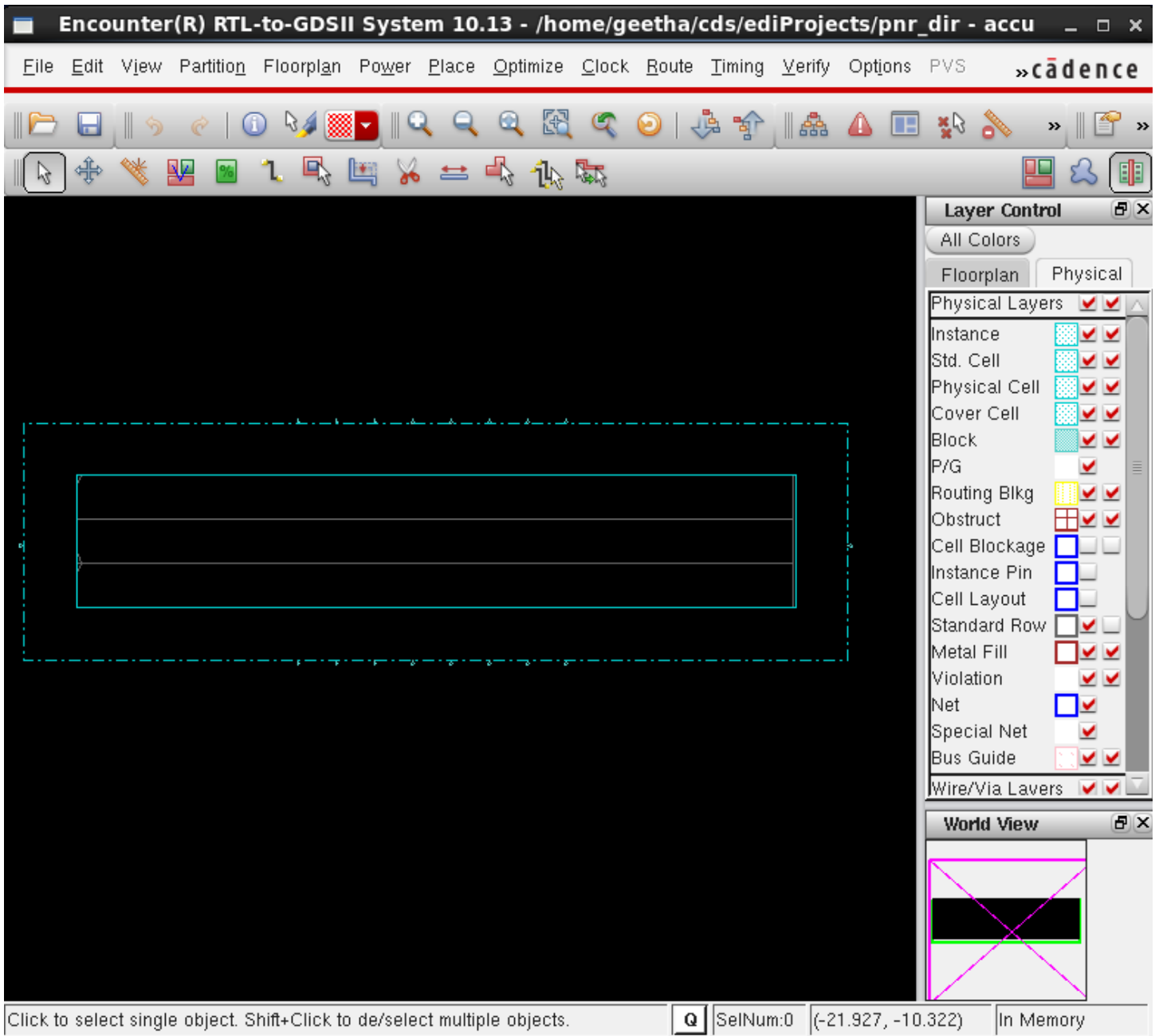
The task is to create a “floorplan” for the design.



```
encounter_log
Log window for place and route scripts ... (Tue Aug 14 17:00:21 CDT 2018)
)
Starting enc_hitikit.tcl

### austriamicrosystems HitKit-Utilities Menu added
Executing Encounter freeDesign command ...
Executing amsDbSetup (loading configuration data)
Executing amsUserGrid (setting up grid for floorplanning)
Executing amsGlobalConnect (connecting up to vdd! and gnd!)
Executing amsSetMMMC (creating timing constraints)
Executing amsSetAnalysisView minmax pnr
Executing amsFloorplan (floorplanning)
Executing SIUE Tcl package placePins procedure (pin location specified in env.tcl file)
--> Type resume to continue after reviewing the FLOORPLAN!!!!
```

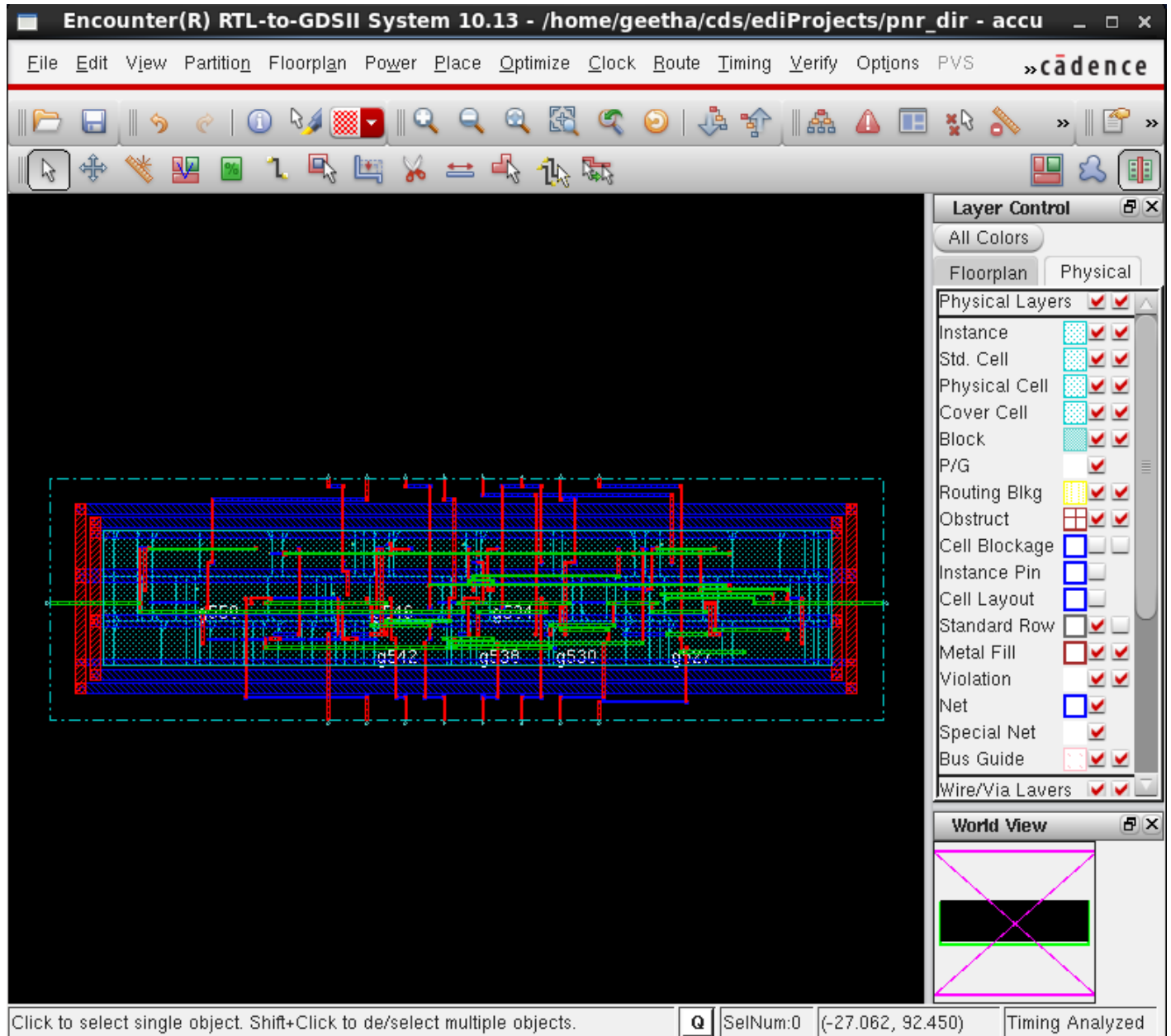
The script pauses so that you can get a good look at the floorplan. You should see the floorplan as shown below.

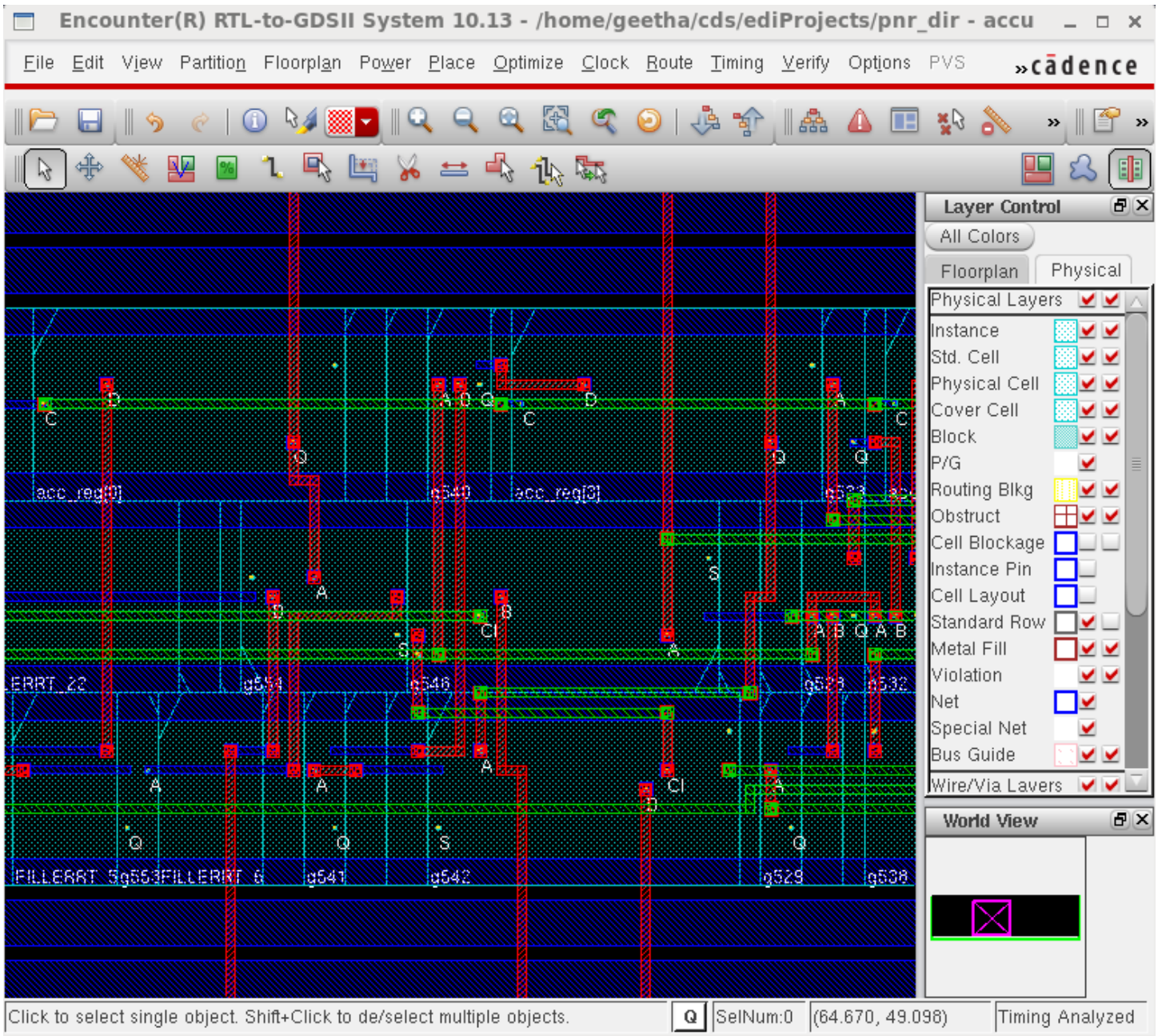


After viewing the floorplan, please type “resume” as instructed by the script. The tools then go on to place the standard cells, create a clock tree, route the cells, etc. Verify that all of the steps discussed in class are actually performed. When the script finishes, you should see the placed and routed design. Zoom in, if you wish to get a better look. Notice that transistors cannot be seen. There is no need for the transistors. The LEF file tells the tool where the inputs and outputs are for each of the standard cells. The LIBERTY file has complete timing information for each of the cells. Nothing more is needed for the standard cell design flow.

Recall that in the previous tutorial (synthesis), we viewed the env.tcl file where the place and route tool was told on which side of the chip the various inputs and outputs should be placed. Did the place and route tool follow our directions? Please check to make sure that the inputs and outputs were indeed placed on the side of the chip which we specified.

In the figure below, notice the huge power and ground rails which ring the core of our design. The width of these rails, the distance between core and boundary, etc. can all be specified in the env.tcl file which we viewed in earlier tutorials.





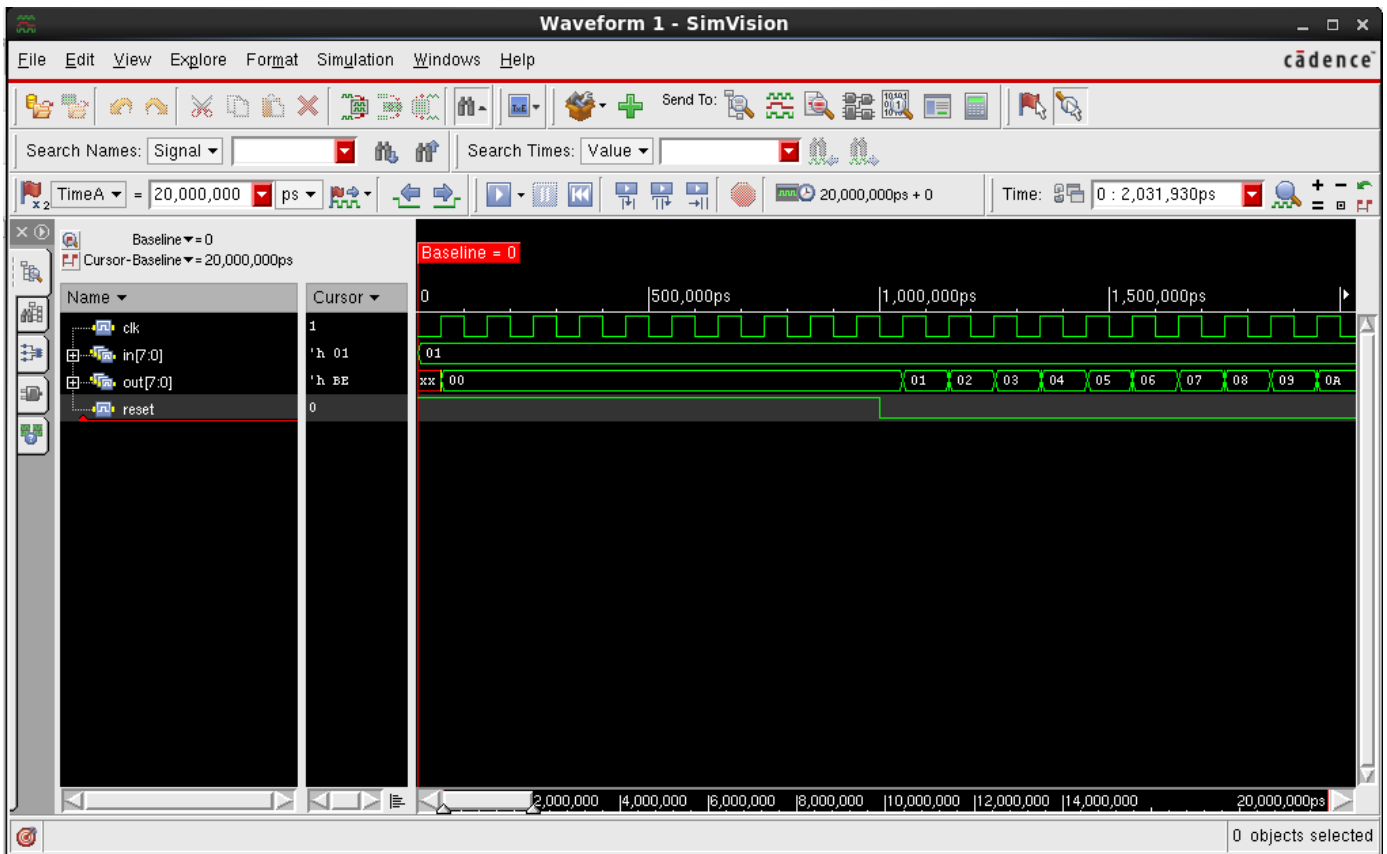
You can go ahead and EXIT the Place & route tool. All we need to do now is to simulate the design. The simulation will include wire delay as well as gate delay. Give the following commands

- cds_ams
- cd \$PHOME
- gedit env.tcl

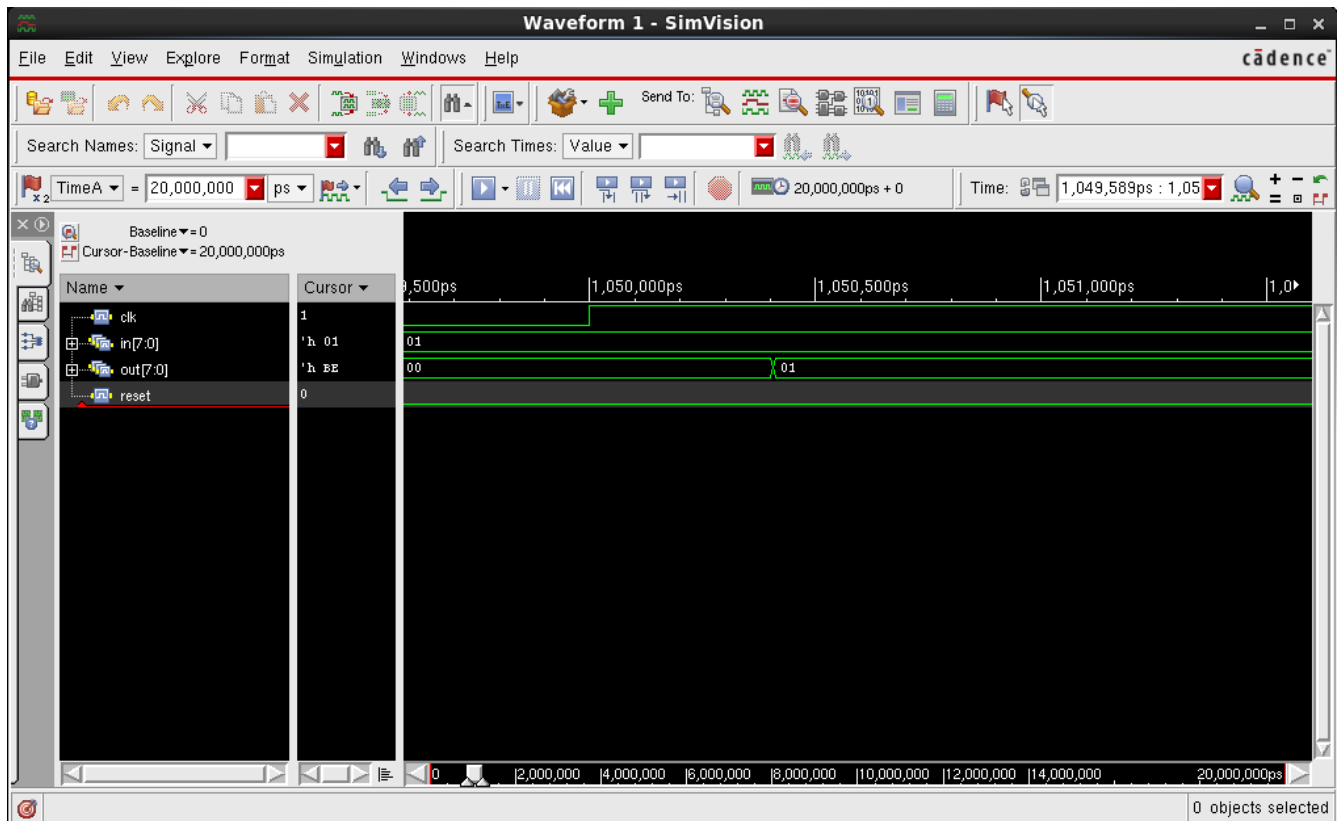
In the env.tcl file, change the simulation mode to “pnr” and then save and exit the editor.

Now type

- sim



If you zoom in, you should see the output signals transitioning after the clock transitions. While wire delay is represented, in this small design the wires are short and their delay is negligible so the output should be very similar to what we say when we simulated the synthesized netlist on the next page.



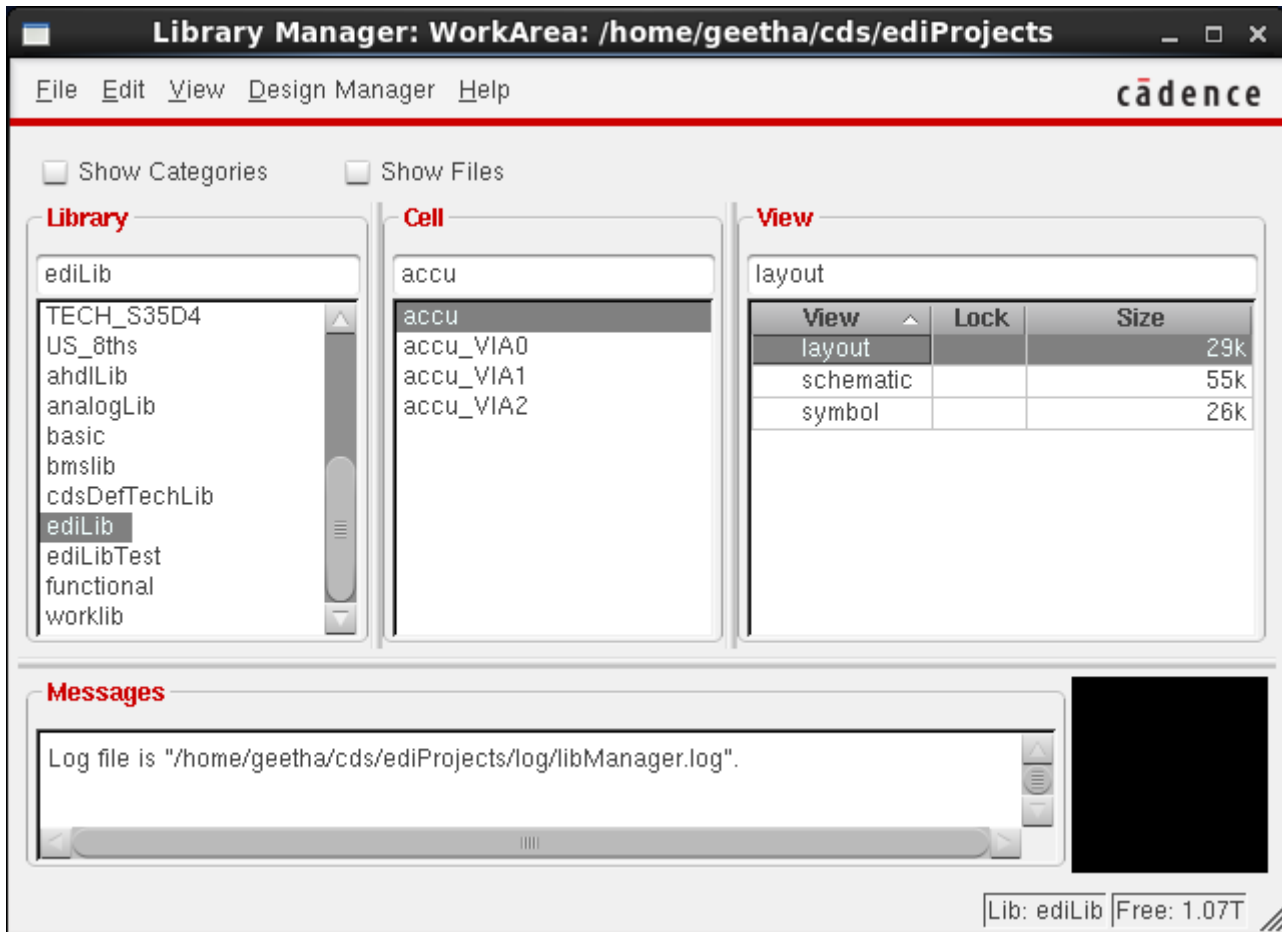
You are done with the EDI(Encounter Digital Implementation) tools now, next step would be to convert this accumulator design & layout from EDI to Cadence Virtuoso and also convert the netlist from edi to schematic in cadence . In order to do this you can execute below two commands

- edi2ic
- edi2sch

Once you do this, you will be able to see your accumulator design with schematic, symbol and layout in the Cadence Virtuoso. Use the below command to launch the Cadence tools

- icd_ams

The accumulator design(**accu**) will be saved in the **Library Manger** → **ediLib**, notice that the schematic generated is made by using the digital standard cells (say INV(inverter), NOR gates, ADD32(Adder) and DF3(D flipflop) etc..) from the CORELIB in cadence. Layout that is generated also uses the primitive standard cells layouts.



Now you can proceed further to check if the layout confirms to all the manufacturers rules using Calibre DRC, it should just show the benign DRC errors. And finally, you need to do Calibre LVS check in order to confirm that the layout yields a netlist that should match the netlist generated by the schematic. you will see a smiling face in the Calibre - RVE dialog box, showing that the netlists match.

Congratulations you have successfully completed your first standard cell design. **Exit** the cadence tool. Make sure all windows are closed.

12 Parasitic Extraction User Guide

Calibre Parasitic Extraction (PEX) User Guide SIUE IC Design Lab

Bryan J Orabutt

August 5, 2020

1 PEX Introduction

Parasitic extraction, or PEX, is a tool that allows a designer the ability to create a netlist for simulation that includes parasitic elements. The PEX tool adds additional capacitors, resistors, and diodes to the netlist based on the features that have been drawn in the device layout view. For example, the output wire polygon on an inverter cell will have some capacitance to the substrate (GND) that will not be present in the electrical schematics but will be present in the calibre view generated by PEX.

2 Running PEX

As mentioned above PEX creates parasitic components in the netlist based on features drawn in the device layout. So the first step that must be completed before PEX can be run for any design is to create a device layout and **ensure it generates a clean LVS**. If LVS has errors it is almost certain PEX will fail. Once a layout with a clean LVS is generated you can open the layout view from the library manager and select **Run PEX...** from the **Calibre** menu as shown in Figure 1.

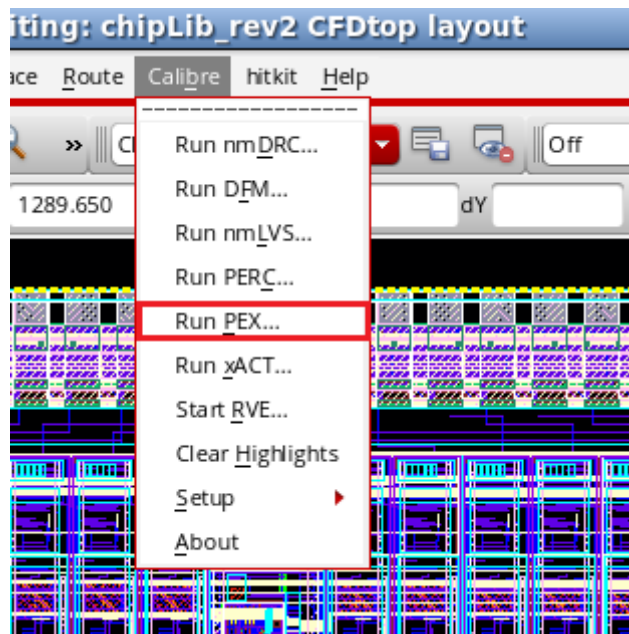


Figure 1: Running PEX from the Calibre menu

After clicking this button a new window will open up. Ensure that your rules file is properly selected as shown in Figure 2. The rules file should be the same for all PEX runs and the PEX Run Directory should be the the **Calibre/PEX/** folder inside the current project directory.

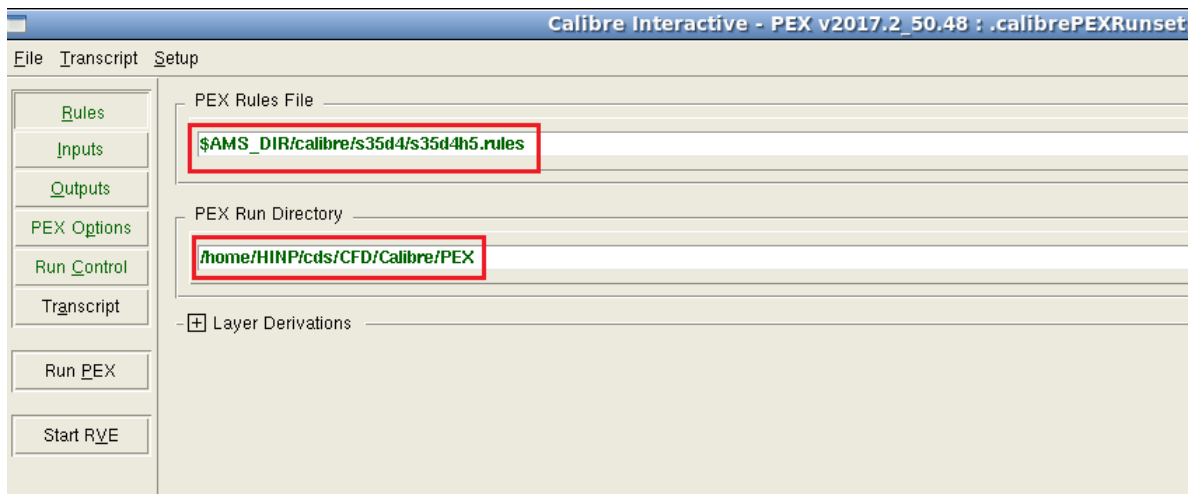


Figure 2: PEX rules setup

Next select the **Outputs** tab. Here you will want to make sure that your extraction type is correctly set for the type of extraction you wish to perform. There are five choices:

- R + C + CC: Extracts resistors, capacitors to substrate, and cross coupling capacitors between nodes.
- R + C: Extracts only resistors and capacitors to substrate.
- R: Extracts only resistors.
- C + CC: Extracts only capacitors to substrate and cross coupling capacitors between nodes.
- No R/C: Creates a normal netlist with no parasitic elements extracted.

Typically for a full chip extraction we will do **C + CC**. Ensure the extraction type is **Transistor Level**, format is **CALIBREVIEW**, and Use Names From is **LAYOUT** as shown in Figure 3.

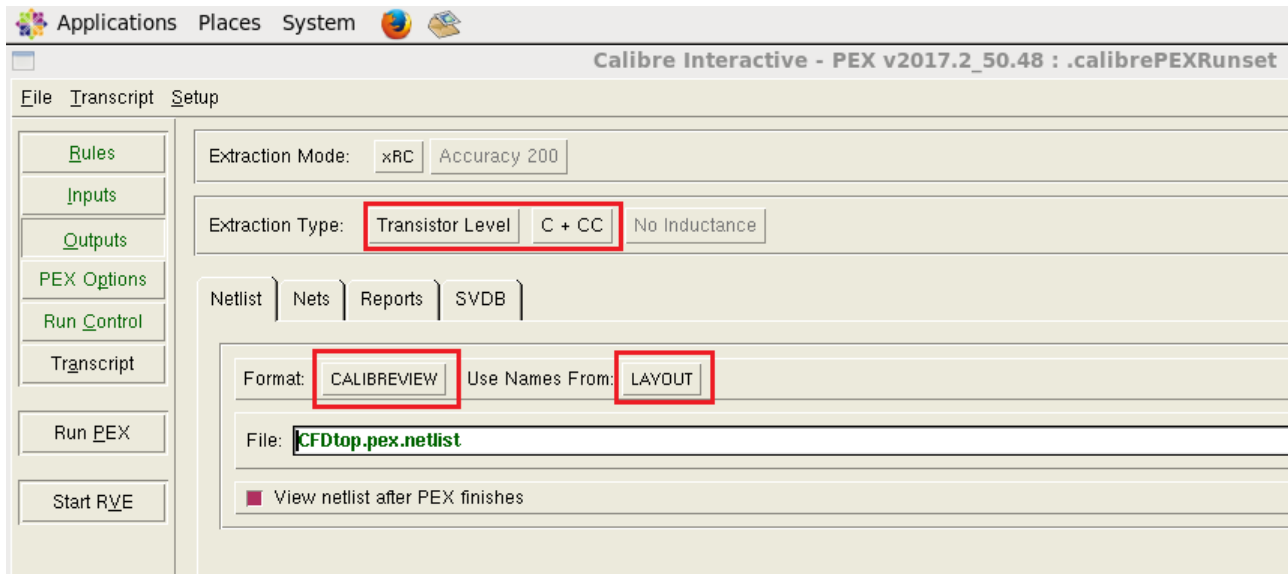


Figure 3: PEX output setup

Next move to the **PEX Options** tab. If this is not visible please enable it via the setup menu as shown in Figure 4.

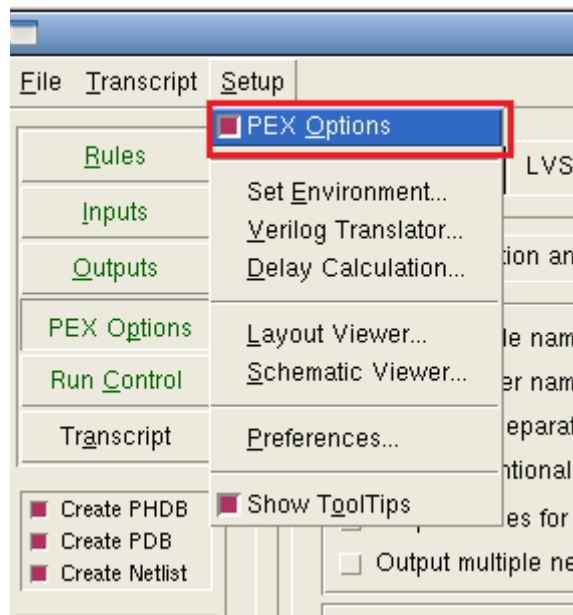


Figure 4: Enable PEX Options menu

Once you are in the PEX Options tab, you will first want to setup the **Netlist** options, so select the Netlist tab. Under the Netlist tab there are many sub tabs, start with the **Format** subtab. Here we will specify the ground node name. This is the net that all capacitors to the substrate will be tied to so this net name will need to be a **real ground net in your circuit**. Once you make you changes your menu should look similar to Figure 5

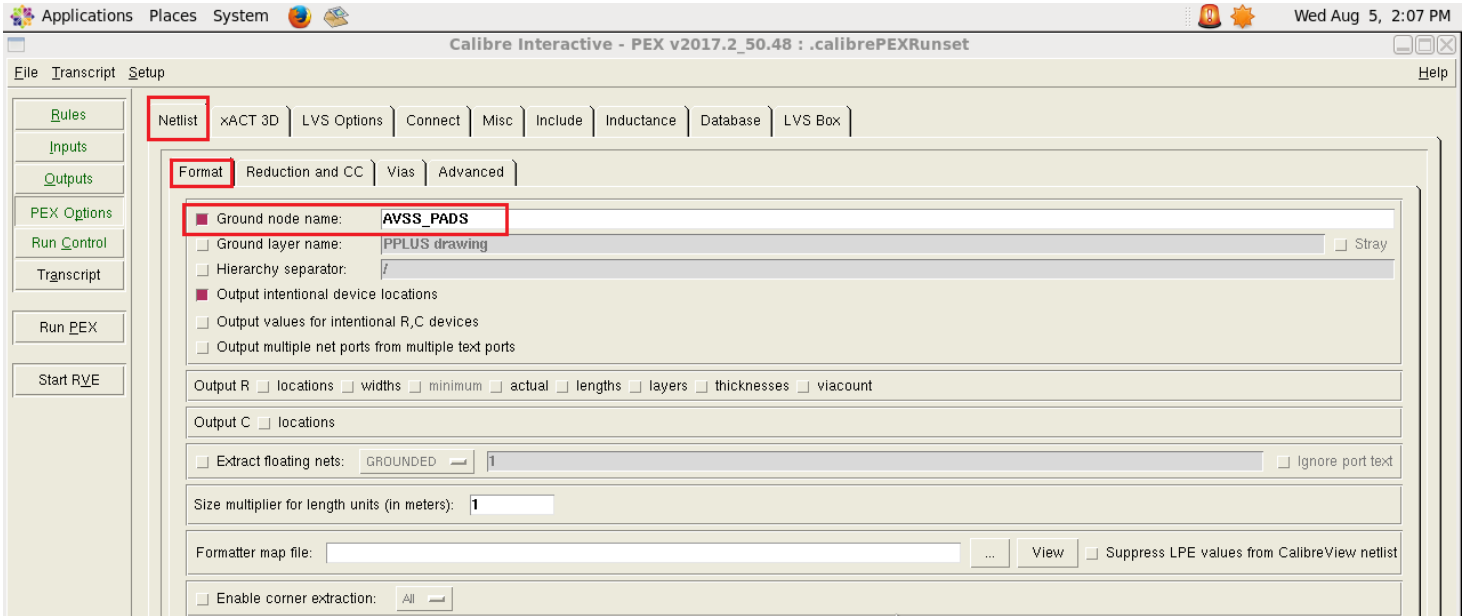


Figure 5: PEX netlist format setup

Next move to the **Reduction and CC** subtab (still within the Netlist main tab). Here we can specify thresholds for our parasitic components. The appropriate threshold to set will depend on the circuit you are trying to extract. For a full chip extraction a capacitance threshold of 25 fF is typical. However a smaller cell like an inverter might need to go as small as 0.1 fF to extract any parasitics. Edit only the fields shown in Figure 6 to change capacitance and resistance extraction thresholds.

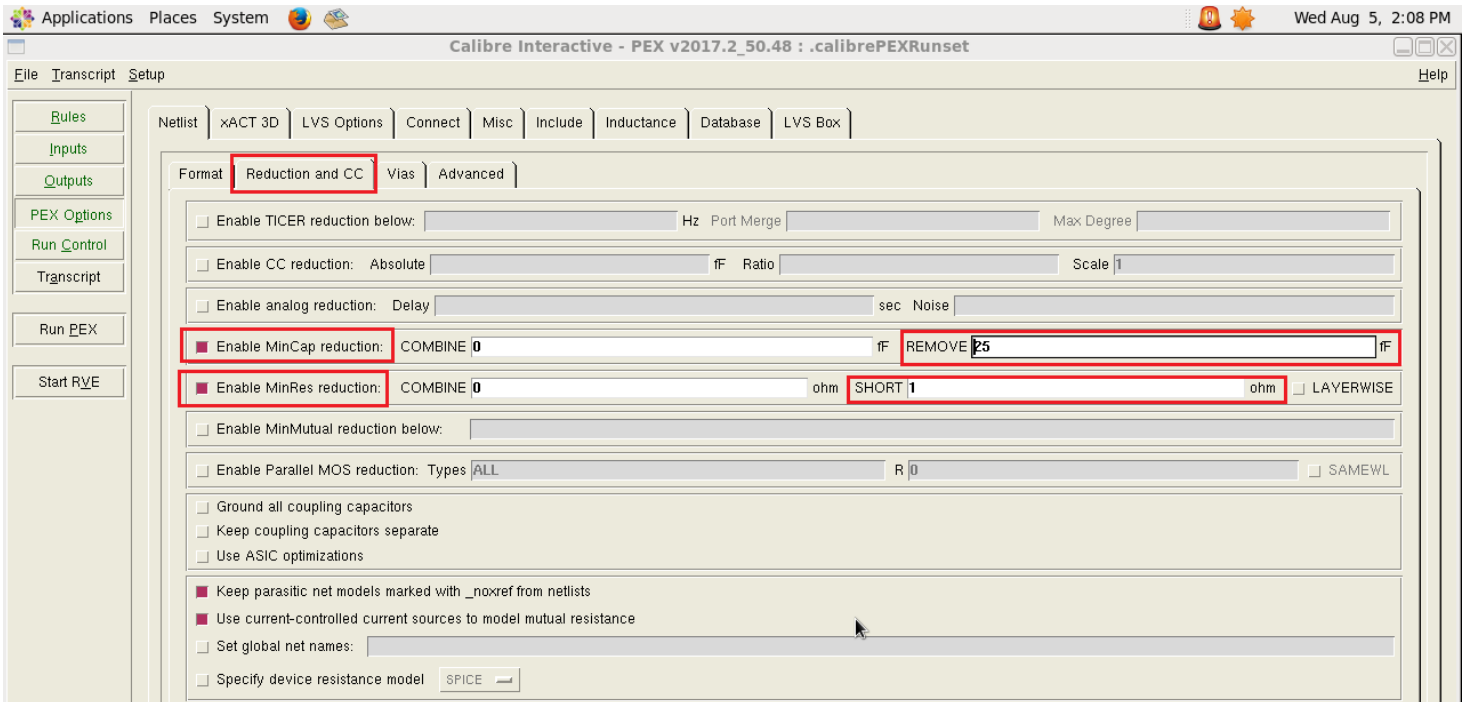


Figure 6: PEX parasitic reduction setup

Next switch the main tab from Netlist to **LVS Options**. You should not have to change much here. Ensure the **Power nets** and **Ground nets** fields are *not* populated. These fields are only relevant if special power net LVS rules are included and we do not use those rules. Also ensure that **Recognize all gates** is selected. This will allow the LVS tool to recognize logic gates from their transistor level components, making the LVS comparison more accurate. Your window should look like Figure 7.

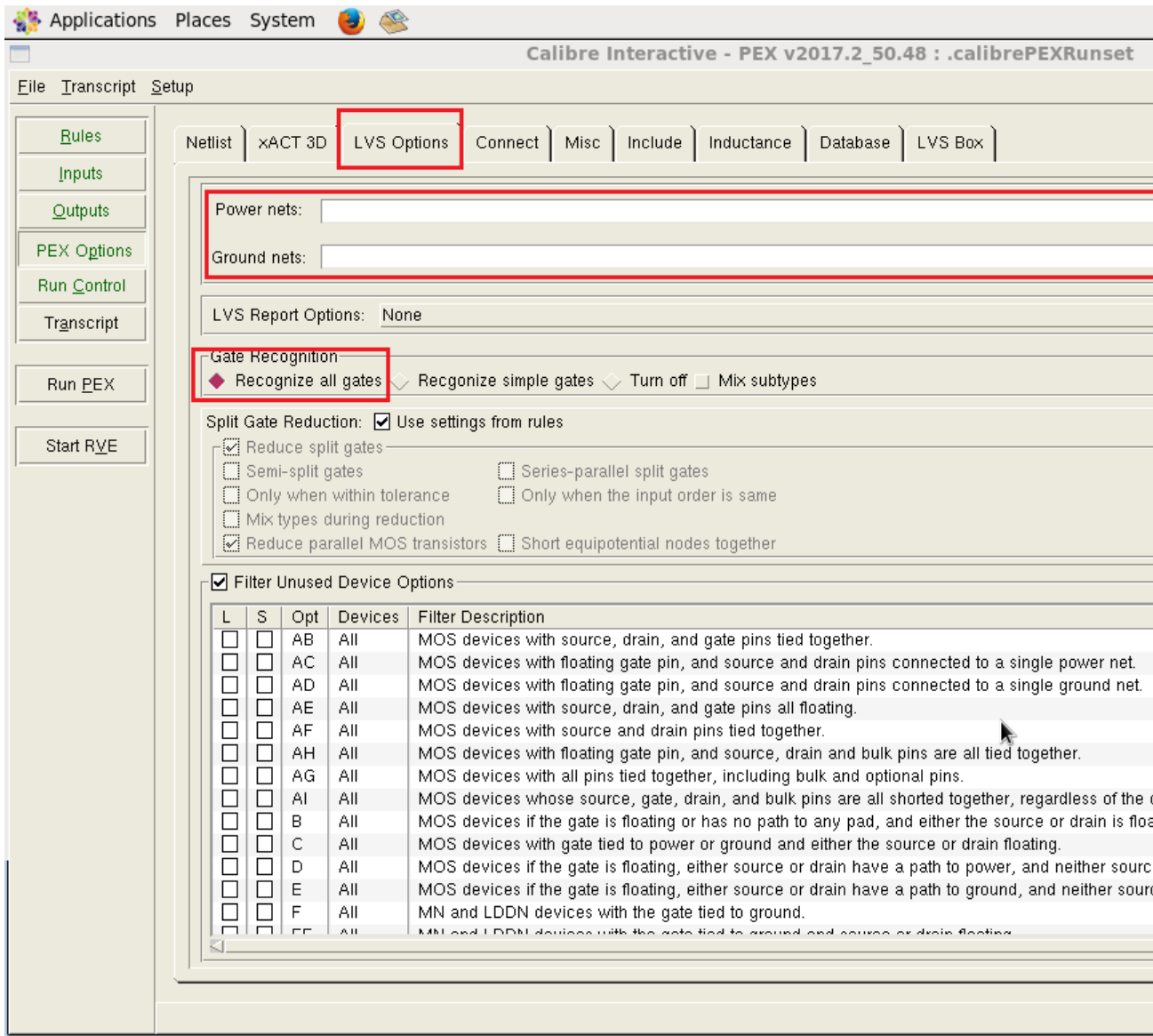


Figure 7: PEX LVS options setup

The last tab that will need checking is the **Include** tab. In this tab we need to specify some extra PEX rules that should be included in this run. The following rules will need to be included:

- LAYOUT CASE YES – This will ensure that PEX treats layout net names as case sensitive.
- SOURCE CASE YES – This will ensure that PEX treats schematic net names as case sensitive.
- LVS COMPARE CASE YES – This will ensure that when comparing layout nets to schematic nets, the PEX LVS tool will treat the comparison in a case sensitive way.

Your window will look like Figure 8 when all changes are made.

At this point you can click **Run PEX**. PEX uses the hierarchical LVS tool by default, which is generally faster. However this tool can give **false error**. Make sure to check out the errors (if any) that are reported and to run RVE and look at the LVS report. If it looks fine and you had a clean LVS before running PEX you can move on. Otherwise please identify the source of the problem or consult with someone more experienced.

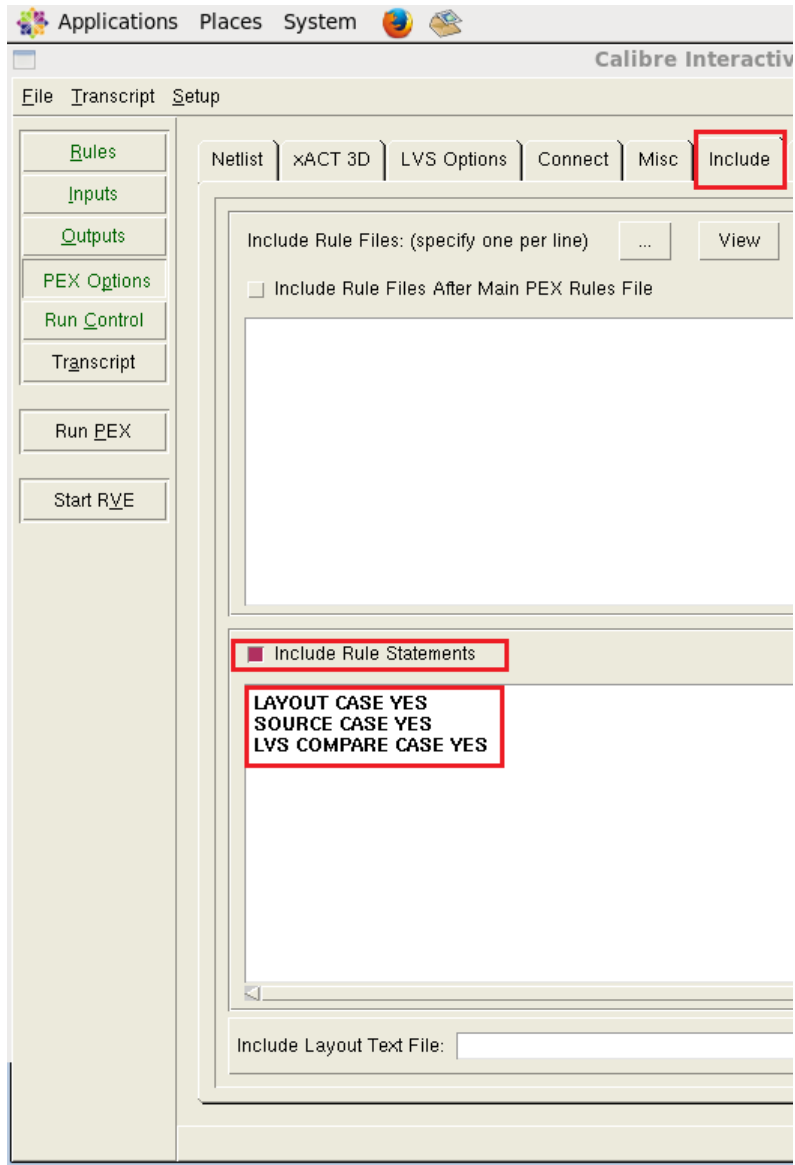


Figure 8: PEX Include options

Once PEX finishes running a new window will pop up to enter the calibre view setup information. This window should be pre-populated correctly but ensure it matches what is shown in Figure 9. Make doubly sure that the **Calibre View Type** is set to schematic or it **will not simulate**.

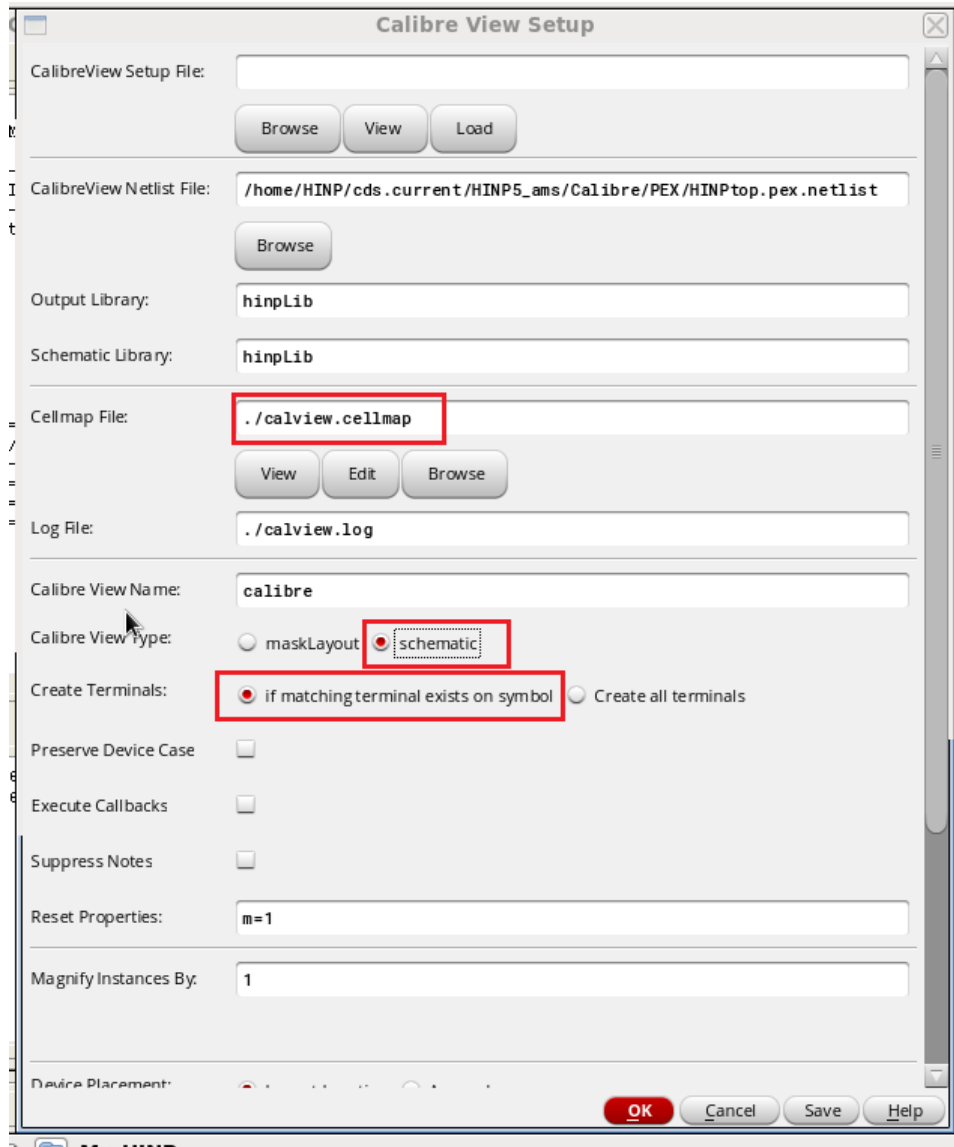


Figure 9: PEX calibre view setup

Depending on the size of the circuit being extracted, this could take some time to finish. Once done a window will pop up telling you if there were any warnings and errors. If there are no errors you can simulate. It is normal to have some warnings. Several thousand warnings is not atypical for a full chip extraction. You can verify extraction succeeded by viewing the netlist or opening the calibre view from the library manager and identifying the parasitic components.

3 Simulating a Calibre view

Calibre is a separate piece of software from Cadence Virtuoso, and as such the simulator tool does not use calibre view schematics for simulation. In order to simulate our extracted schematic we will need to create a configuration view to tell spectre to use our calibre view instead of the normal schematic. The first step for this is to create a test bench schematic. Once the test bench schematic is done, create a new cell view for that test bench from the library manager as shown in Figure 10. In the dialog box that pops up change the view type to **config** as shown in Figure 11.

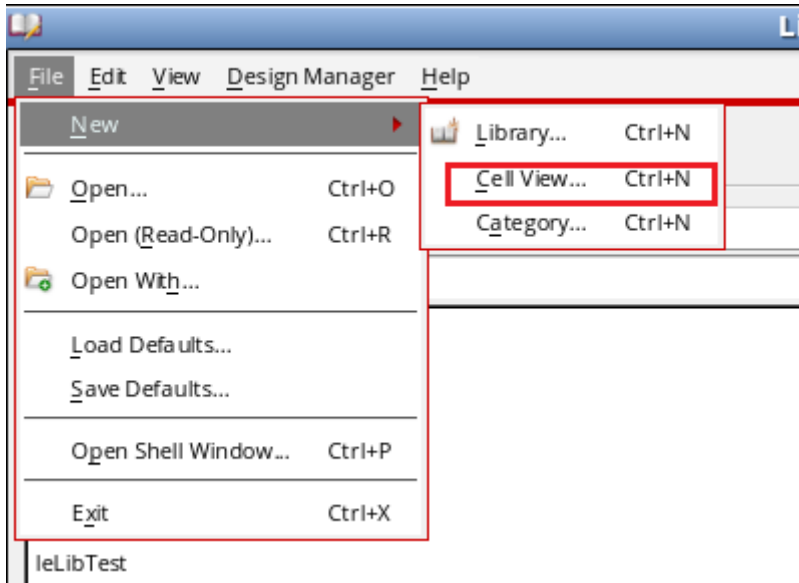


Figure 10: Creating a new cellview

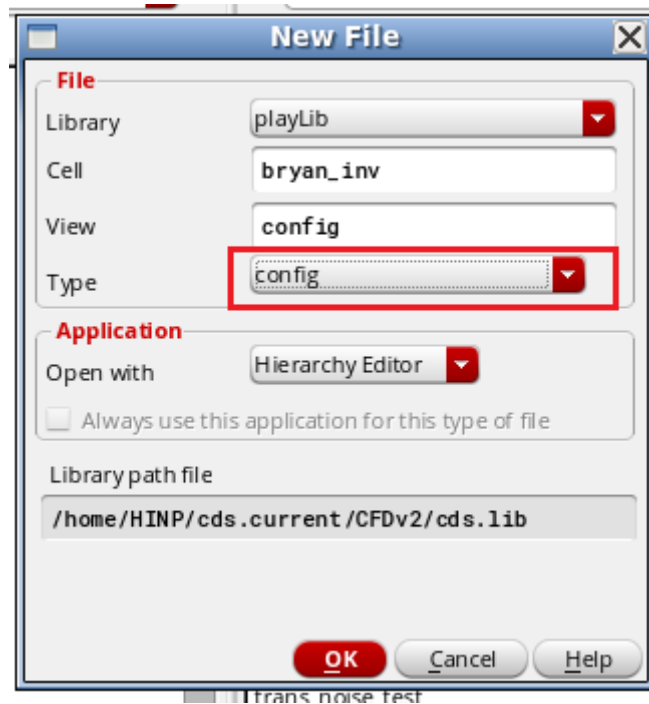


Figure 11: Creating a config view

You will be presented with a new dialog box to setup your config view. The easiest way to do this is to click the **Use Template** button and use the **spectre** template from the drop down as shown in Figure 12.

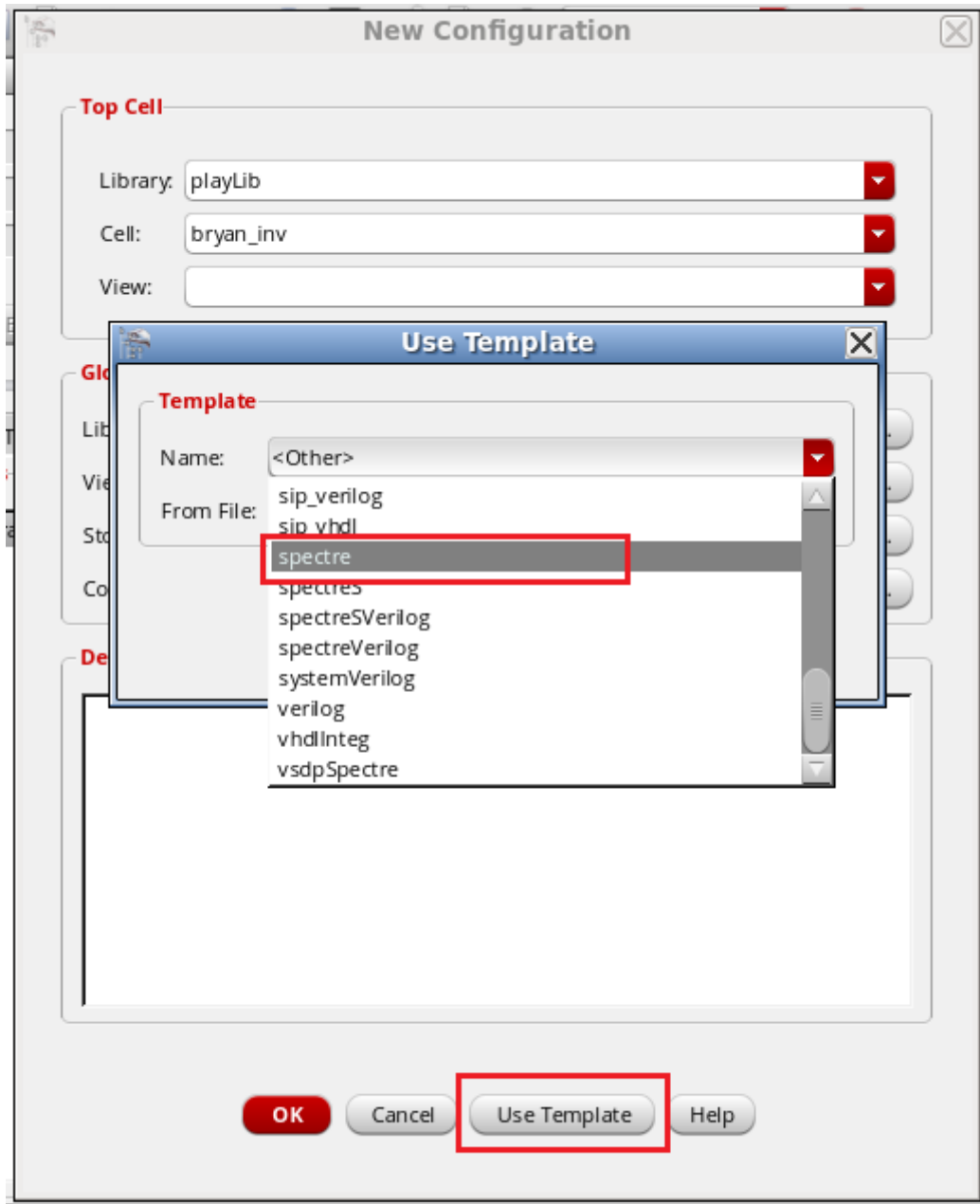


Figure 12: Spectre config view template

The config view setup window will now be populated with some default values for the spectre simulator. Change the libraries to include the libraries the testbench is in, as well as all of the libraries containing instances instantiated in the test bench. Finally, add **calibre** to the view list so that our calibre view will be visible to spectre. Once completed the window should look similar to Figure 13. If it is you can click **OK**

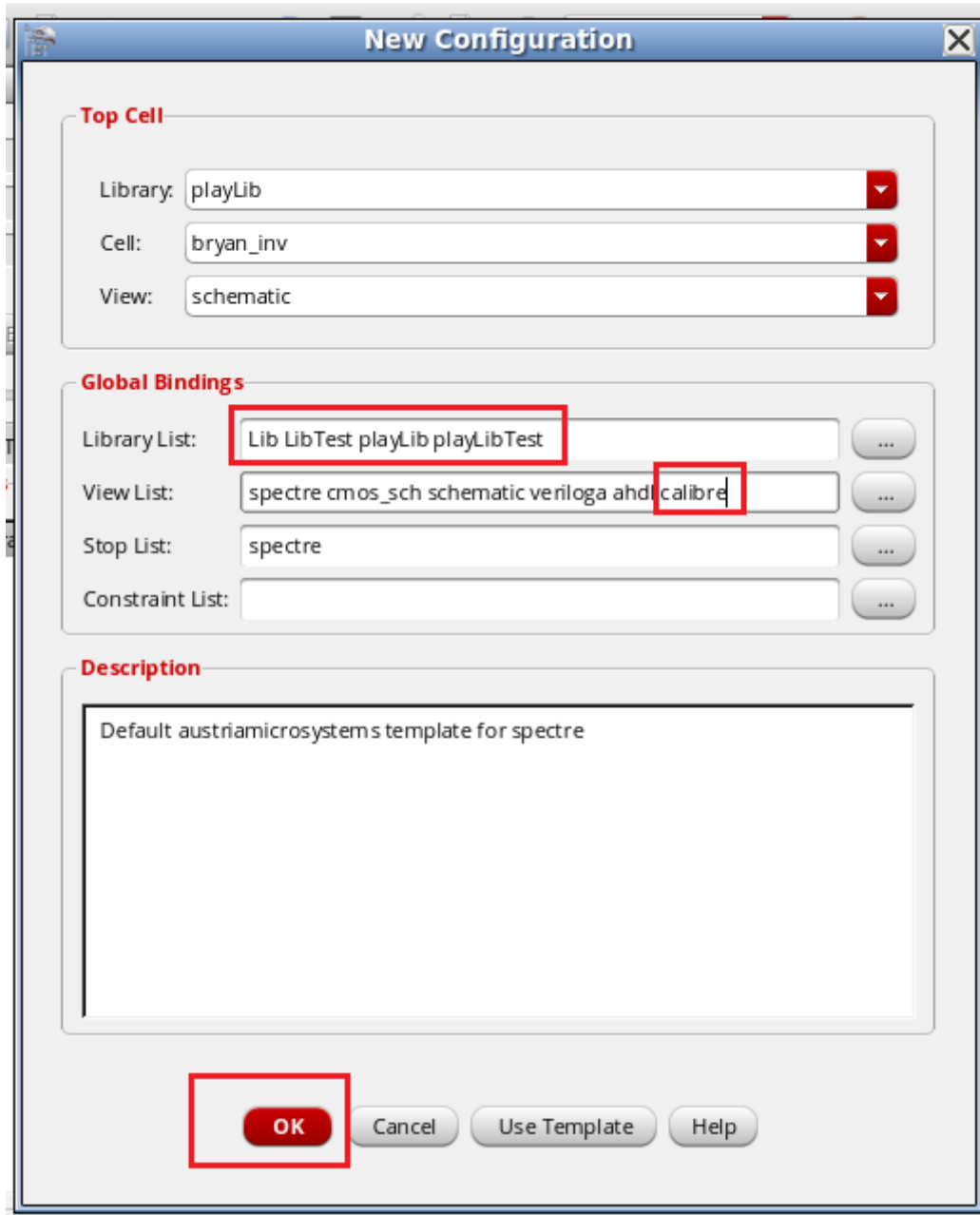


Figure 13: Complete config view setup window

The last step before simulation is to tell spectre which view it should simulate using the hierarchical view window. Find the cell you created the calibre view for in the list of instances, and change it's **view to use** field to **calibre** and click save. You should see **calibre** in blue lettering meaning the view was found (see Figure 14). You can now run a simulation as you normally would, just make sure the design being simulated is the config view rather than the schematic view and spectre will simulate your extracted schematic!

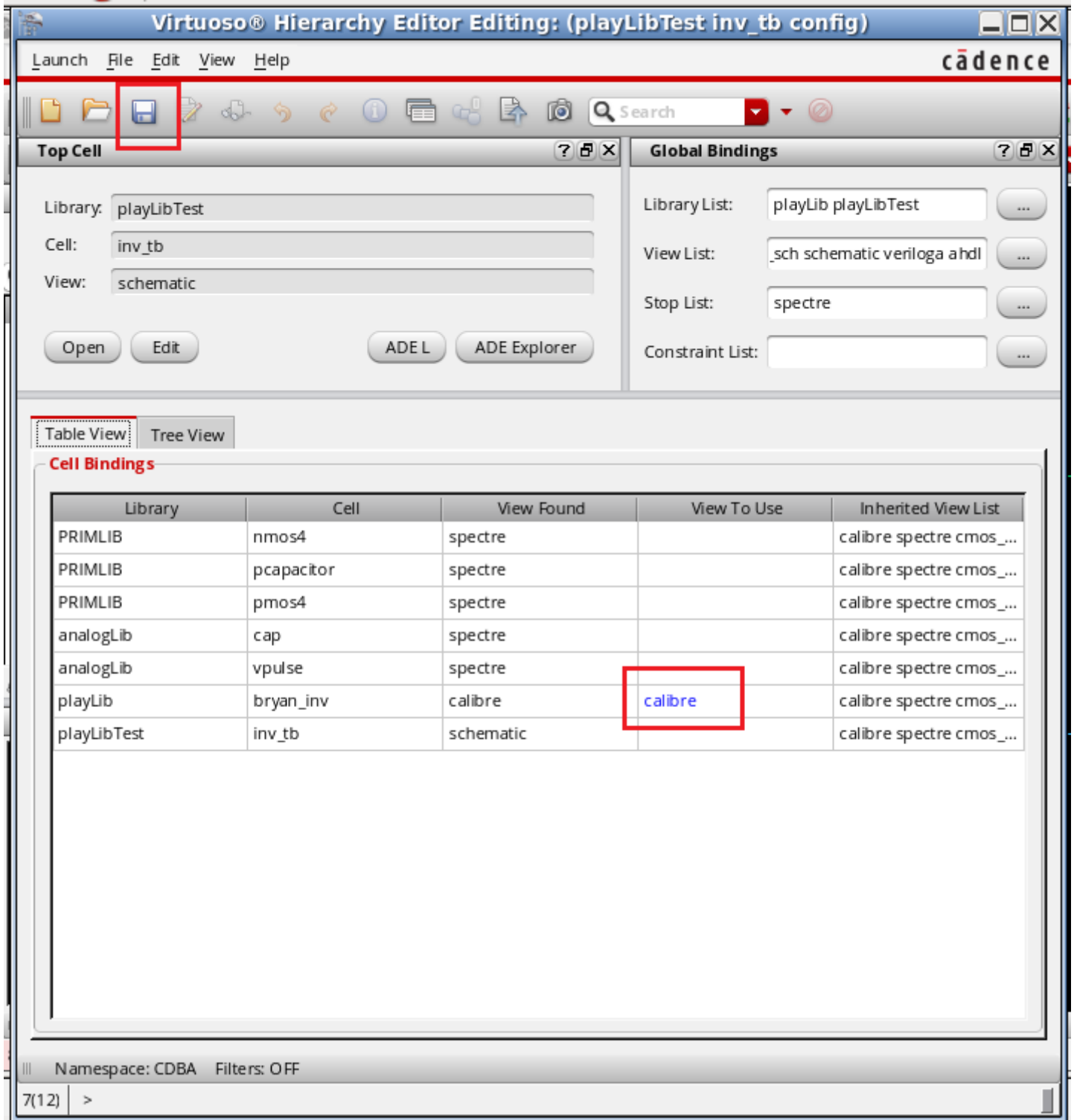


Figure 14: Hierarchical view window

13 Cadence Hot Keys

Cadence ICFB Hot Keys

Library Manager:

ctrl-r opens the selected view (the cell& view which is selected in library manager) for read

ctrl-o opens the selected view for editing

Schematic Diagram (frequently used):

w add a wire

i add an instance

p add a pin

l label to a wire

e display options like, grid size, snap size etc

q select an object and press **q** to open the property dialogue box

{ or **shift-z** zoom-out by 2×

} or **ctrl-z** zoom-in by 2×

c copy

m move: if you move an object, none of the wires connected to it move to maintain connectivity

s stretch: if you stretch an object all connections to it also extend to maintain connectivity

f3 (when in move or stretch mode): opens dialogue box to select move/stretch mode

a) snap mode: any angle. orthogonal, diagonal, etc. When we move an object, it can be restricted to move only orthogonally, or in addition to orthogonal diagonally too, or unrestricted (any angle) this changes those settings

b) turn/flip options: If before moving/stretching to final destination you want to rotate or flip the object, then this window (invoked by f3) lets you do that.

f2 save

f8 check and save

u undo

shift-U redo

<delete> delete an object

ctrl-d deselect all.

Schematic Diagram (not frequently used):

f3 save

f5 open

tab pan

f fit: fits the entire schematic in the window

shift-X descend to edit by one

b to go one level up and **shift-b** to return to top

x descend to read by one level

ctrl-r redraw the window

shift-v world view: see the whole schematic in a small window at bottom right showing which part of it you are at in the main window

ctrl-w close the window
shift-q properties of the whole cell view like name etc

Layout Tool (frequently used):

f fit
shift-f in hierarchical layout show all levels as if flat
ctrl-f hide all hierarchy and show only outline of instances
r rectangle
q property of an object
ctrl-z *zoom in*
shift-z *zoom out*
f2 save
t *tap*: if you select a layer, saw NW in layout and press tap, that layer gets selected in LSW (layer selection window). Then you can use **r** to draw rectangles of that layer: Normally we can select a layer in LSW and when we press “r” a rectangle of that layer gets drawn. But if we press “t” (called tap) and then select a shape/rectangle, the layer of that shape/rectangle gets selected in LSW and then pressing r creates rectangle of that layer
p *path*: makes a min width path of the layer selected in LSW : If some layer is selected (highlighted) in LSW, then “p” starts to create a path of that layer with width same as the min width for that layer defined in the drc (if it is loaded into icfb)
ctrl-a select all
ctrl-d deselect all
c *copy*
m *move*: move a whole rectangle
s *stretch*: can stretch just a side of a rectangle
f *fit*
k ruler
i add an instance
u *undo*
shift-U redo
shift-r *reshape*: use it to reshape a layer so as to make it bigger, e.g., a turn in a metal wire
shift-c *chop*: chop a rectangle, i.e., reduce its size as you want
shift-m merge all rectangles selected as per their layer-purpose name (lpp) (layer purpose pair) for example (“met1” “drawing”) is a 2 value array (pair) having value of layer (met1) and its purpose (drawing), i.e. all selected and touching rectangles or paths in same layer-purpose pair (say all met1 drawings) (my above comment will clarify this too) get merged into one.
e display options like grid size, snap size etc
f6 redraw