EEE 4134 VLSI I Laboratory
Lab 0 (Introductory Lab)
Logging into Cadence Server, Tool Setup, Cell Library Creation, Introduction to Custom IC Design flow

Objectives:

- To login, start a shell tool and start the Cadence Virtuoso software
- To learn about PDK and add the PDK library to the Library Manager
- To create a working library and to get familiar with technology
- To be familiar with Custom IC Design flow

Logging in, starting a shell tool, and starting the Cadence Tool Suite

1. Find Desktop shortcut icon for XLaunch. Double-click on it. Click Next, Next, Next, Finish (in that order) in the windows that pop-up one after another.

After it starts, you will see the Xming icon at the bottom right corner of your Desktop screen.

2. Find icon for Putty. Double click on it to open it. ‘Putty Configuration’ window will pop-up.

3. Select VLSI_LAB under ‘Saved Sessions’ category. Click Load. The window will look like the following one:
Click **Open**. A Security Alert window may pop-up. Click **Yes**.

4. Now you will see a Terminal window which prompts you for login.

```
172.16.16.150 - PuTTY

login as: [your student ID]
```

5. Log in to your workstation using user ID and password. Your user name and your password will be your student ID. When you are typing your password, the command window will not display the characters you type in, so make sure you are typing the right password. After logging in to your account, Terminal window should look like the following:

```
login as: 120105001
120105001@172.16.16.160's password:
Last login: Thu Mar 30 13:54:21 2017 from 172.16.16.173
```

6. Type **csh** and press ‘Enter’ key.

Then type **source cshrc_q** and press ‘Enter’ key.

The following message will be displayed in the Terminal window:

```
Welcome to Cadence tools Suite
```

That means you can use Cadence tools now.

7. Go to your working directory by typing: **cd cic**

8. Type **virtuoso&**

A sample command prompt screen is shown below:

```
120105001@aust~
120105001@172.16.16.160's password:
Last login: Thu Mar 30 16:17:23 2017 from 172.16.16.174
[120105001@aust ~]$ csh
[120105001@aust ~]$ source cshrc_q
Welcome to Cadence tools Suite
[120105001@aust ~]$ cd cic
[120105001@aust ~]/cic]$ virtuoso$
[1]$ 7618
```

9. **Virtuoso® Command Interpreter Window (CIW)** appears at the bottom of the screen. From the CIW menus, all Cadence main tools, online help and options can be accessed. In the window area, all kind of messages (info, errors, warnings, etc) generated by the different Cadence tools appear. You can also introduce commands.
Another window ‘What’s new in IC6.1.6 Overview’ appears too. Execute **File ➔ Close and Do Not Show Again** and this window will not appear the next time you open Virtuoso.

**NOTE:** You have to perform these steps above in every class where Cadence tools will be used.  
*The following figure summarizes the steps:*

```
Start XLaunch
Start Putty
Open VLSI_LAB session
Login to server
Type the commands one by one and press Enter:
csh
source cshrc_q
cd cic
virtuoso&
```
Custom IC Design Flow

The following figure shows the basic design flow of a custom IC design, together with the Cadence tools required in each step:

First, a schematic view of the circuit is created using Cadence Virtuoso Schematic Editor L. Then, the circuit is simulated using Cadence Analog Design Environment (ADE L). Different simulators can be employed; some sold with the Cadence software (e.g., Spectre) some from other vendors (e.g., HSPICE) if they are installed and licensed.

Once circuit specifications are fulfilled in simulation, the circuit layout is created using Virtuoso Layout Editor L. The resulting layout must verify some geometric rules dependent on the
technology (design rules). For enforcing it, a Design Rule Check (DRC) is performed. Optionally, some electrical errors (e.g. shorts) can also be detected using an Electrical Rule Check (ERC). Then, the layout should be compared to the circuit schematic to ensure that the intended functionality is implemented. This can be done with a Layout Versus Schematic (LVS) check. All these verification tools are included in the Assura software in Cadence.

Finally, a netlist including all layout parasitics should be extracted using Quantus QRC tool, and a final simulation of this netlist should be made. This is called a Post-Layout simulation, and is performed with the same Cadence simulation tools.

Once verified the layout functionality, the final layout is converted to a certain standard file format (GDSII, CIF, etc.) depending on the foundry using the Cadence conversion tools.

**Learning fundamentals of PDK and adding PDK library to the Library Manager**

Cadence is an Electronic Design Automation (EDA) environment that allows integrating in a single framework different applications and tools (both proprietary and from other vendors), allowing to support all the stages of IC design and verification from a single environment. These tools are completely general, supporting different fabrication technologies. When a particular technology is selected, a set of configuration and technology-related files are employed for customizing the Cadence environment. This set of files is commonly referred as a process design kit.

All VLSI designs start with a Process Design Kit known briefly as PDK. A PDK contains the process technology and needed information to do device-level design in the Cadence Design Framework II (DFII) environment.

Throughout the labs we will use a generic, foundry independent 90nm CMOS mixed-signal process kit developed by Cadence. We will call it generic PDK 90 nm briefly as gpdk090. A PDK contains all the necessary design and technology data to successfully design and simulate a VLSI chip on a particular foundry. The foundry provides the necessary technological data, design rules, and the device models. Also PDK contains schematic symbols with all necessary views, as well as device extraction rules for Layout versus Schematic (LVS) check. It also provides parasitic extraction rules.

**Creating a library and attaching technology to library**

All the entities in Cadence are managed using libraries, and each library contains cells. Each cell contains different design views (the structure is similar –and physically corresponds - to a directory (library) containing subdirectories (cells), each one containing files (views). Thus, for instance, a certain circuit (e.g. an inverter) can be stored in a library, and such library can contain the different logic blocks (basic gates, flip-flops, registers, etc) stored as cells. Each block (cell) contains different views (schematic, layout, symbol, etc.).
There are usually three types of libraries:

- A set of common Cadence libraries that come with the Cadence software (containing basic components, such as voltage and current sources, R, L, C, etc).
- Libraries that come with a certain design kit (e.g. gpdk090) and that are related to a certain technology (e.g. transistors with a certain model attached, etc).
- User libraries; where the user stores its designs. These designs employ components from the Cadence/design kit libraries.

It is recommended that you use a library to store related cell views; e.g., use a library to hold all the cell views for a single project (that can involve a complete chip design). In our example, we are going to create a new library for our design and attach it to desired technology library.

1. In the CIW, execute **File → New → Library**.
2. The ‘New Library’ form appears. In the name field of the New Library type mylib or any name of your choice.

   ![New Library Form](image)

3. Select **Attach to an existing technology library** and click **OK**. ‘Attach Library to Technology Library’ window will appear.

   ![Attach Library to Technology Library Form](image)

4. Select gpdk090 technology library and click **OK**. This will be the technology chosen for your design (that you will employ eventually for fabrication). Now all the designs made in this library
are technology-dependent (e.g., the schematic MOS symbols have by default the model for this
technology, the available layout layers correspond to this technology, etc.).

5. In the CIW, the following message will appear:

```
Loaded gpdk100/libInit.il successfully!
Created library "mylib" as "/home/Fall16/120105001/cic/mylib"
INFO (TECH-160011): Design library 'mylib' successfully attached to technology library 'gpdk100'.
```
EEE 4134 VLSI I Laboratory
Lab 1
Introduction to Virtuoso Schematic Editor, Creating Inverter schematic, Performing transient simulation of Inverter schematic, Power and delay measurement of designed inverter for different corners

Objectives:
- To learn how to draw schematic of basic logic gates in Cadence Virtuoso
- To learn how to perform transient simulation of logic gates
- To learn about process corners and their effects on delay and power dissipation
- To learn how to measure power dissipation and propagation delay of logic gates

Schematic Entry: Creating a Schematic cell view

In this exercise, you will learn how to enter simple schematic and run a simulation to perform timing simulation of an inverter designed using gpdk090 technology.

1. In the Command Interpreter Window (CIW), execute File ➔ New ➔ Cellview. Set up the ‘New File’ form as follows:
   - Library: mylib
   - Cell: inverter
   - View: schematic
   - Type: schematic
   - Application: Open with: Schematics L

2. Click OK when done. The following window may appear. Click Yes/Always.

A blank schematic window for the inverter design appears.
Schematic Entry: Adding an Instance to Schematic

Next, we will create simple schematic of an inverter consisting of an NMOS and a PMOS.

1. To create an instance, you can execute Create → Instance in Virtuoso schematic editor window or simply use shortcut key “i”. The following window will appear:

2. Click Browse to select a library component. Another window will show up. Choose Library: gpdk090, Cell: nmos1v, View: symbol. (Note that while you are doing this, the ‘Add Instance’ form is getting updated as well).
3. Make sure that the view name field in the form is set to symbol. After you complete the form, move your cursor to the schematic window and click left to place the component.

Similarly, add pmos1v cell.

4. Now we can adjust the sizes of the transistors by editing instance properties. Left click on the NMOS to select the component. Then, press “q” to modify its properties, or in schematic editor window, execute Edit → Properties → Object.

You will update the Library Name, Cell Name, and the property values given in the table below as you place each component. The inverter design contains the following cells from the following libraries.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Cell Name</th>
<th>Properties/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpdk090</td>
<td>nmos1v</td>
<td>For NM0, Width=240n (this is 2x the minimum channel width)</td>
</tr>
<tr>
<td>gpdk090</td>
<td>pmos1v</td>
<td>For PM0, Width=480n</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd</td>
<td></td>
</tr>
<tr>
<td>analogLib</td>
<td>gnd</td>
<td></td>
</tr>
</tbody>
</table>

For example, while modifying the transistor width for NMOS, set Total Width to 240n, and then press ‘Tab’ key and the Finger Width will be set to the same value. Click OK.

Repeat this for PMOS to set Total Width and Finger Width to 480n. To deselect any object, press keyboard command “Ctrl+d”.

If you place component in the wrong location, select the component and press ‘m’ on keyboard and move it to your desired location. After entering the components, click Cancel in the Create Instance form or press Esc keeping your cursor in the schematic window.

Next, instantiate power nets (cell vdd and gnd from analogLib library).
5. Execute **Create → Pin** or press ‘p’ on keyboard. ‘Add Pin’ form will appear. Enter the name of the pin (e.g. in). **Direction** of the pin (e.g. input/output etc.). Add all the pins (in and out) to the schematic. For an inverter, gate input pin (e.g. in), is the input pin. And output pin (e.g. out) at the common node between drains of NMOS and PMOS is output pin. So, select **Direction** property as **input** for **in**, and **output** for **out**.

6. Use **Add → Wire** menu or simply press ‘w’ key to enter wiring mode / **Esc** to exit. It is a good practice to periodically save your work by clicking on **Check and Save** button (the checkmark button just below the Tools menu). You can also save your work from the drop-down menu **File → Save**.

The final schematic looks like the following one:
7. Click Check and Save.

8. Check CIW for errors. If there is no error, you should see the following message:

```
INFO (SCH-1170): Extracting "inverter schematic"
INFO (SCH-1426): Schematic check completed with no errors.
INFO (SCH-1181): "mylib inverter schematic" saved.
```

### Netlist Creation and Simulation using Spectre

The following flowchart shows the steps to be executed to simulate a design using ADE L:

1. In the Schematic editor window, execute **Launch → ADE L**. The following window may appear. Click **Always**.
Analog Design Environment (ADE) L window will appear.

2. Set up the model libraries by executing **Setup → Model Libraries**. ‘Model Library Setup’ Window will appear:

3. Click twice on the file name given under **Global Model Files**. An ash coloured button will appear.

Click on the button. ‘Choose Model File’ window will appear.
4. Select `gpdk090_mos.scs` from the list. Click **Open**.
In this model file, there are models to simulate various corners like fast-fast (FF), fast-slow (FS),
typical-typical (TT) etc. These are called process corners, depending on the speed of MOS
transistors (NMOS and PMOS). Refer to the following figure for the definition of process corners:

![Process Corners Diagram]

We will choose the section typical from the **Section** scroll bar and select the section 'TT_s1v'.
These will enable us to use the TT models of the 1.2 V MOS transistors. Click **OK**.

<table>
<thead>
<tr>
<th>Model File</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/tech/process/gpdk090/models/spectre/gpdk090_mos.scs</code></td>
<td>TT_s1v</td>
</tr>
</tbody>
</table>

5. Now execute **Setup** ➔ **Stimuli** to assign signals to pins of the inverter.
6. In ‘Setup Analog Stimuli’ window, select Global Sources. Now you can see global power net vdd!. Click on Enabled, Select dc under Function and Voltage under Type. The filled up form for ‘vdd!’ will look like the one below. Click Apply (it is a must, clicking OK will close the window and the value will not be saved).

7. For input pin ‘in’, we have to set a pulse waveform. The following figure shows the definition of pulse parameters:

For setting signal to input pin ‘in’, select Inputs in Setup Analog Stimuli window. Click on ‘Enabled’, select Function: ‘pulse’, Type: ‘Voltage’. Parameters for pulse source will be as follows: Voltage1 = 0V, Voltage2 = 1.2V, Period = 40n, Delay time = 3n, Rise time = 3n, Fall time = 3n, Pulse width = 20n. Click Apply and then click OK.
8. Now choose the analysis to be done from **Analyses ➔ Choose**. Select transient (**tran**) analysis to be done. Provide 100n as ‘stop time’. *(Analysis: tran, Stop Time: 100n, Accuracy Defaults: moderate).* Click **OK**.
9. Select the output to be plotted by executing Outputs → To be plotted → Select on Design in the ADE window. Schematic editor window will pop up, select ‘out’ and ‘in’ to be plotted and saved. When you select them, you will see colors being assigned to these pins.

10. Your Analog Design Environment window should now look like the following:
11. Before closing the Virtuoso Analog Design Environment window, it is a good idea to save design settings in a state file, so we can load it up next time. To do this, execute **Session → Save State** and save state name in the ‘Save As’ field as state inverter. Next time you run Cadence, you can simply load the simulation settings from this file.
12. Now run the simulation by executing Simulation ➔ Netlist and Run in the ADE window. The simulation will run and the output will appear in Virtuoso Visualization & Analysis XL window as shown below.

13. Finally, we are going to separate the plots into two sub-graphs. Click on the following icon for splitting graphs.

The final plot should look like the one shown below:
Definition of rise time, fall time and propagation delay

Three main timing parameters are associated with CMOS devices – rise time, fall time, and propagation delay. Most often, in discussion with regard to these parameters, the system response of an inverter is used. The following figure defines rise time, fall time and propagation delay of a gate with the example of an inverter:

Voltage (in)

Voltage (out)

Time
Referring to the above figure, rise time is the time that it takes to charge the output capacitive load. Fall time is the time it takes for the output capacitive load to discharge. The rise and fall time are usually measured from 10% to 90% and from 90% to 10% of the steady state value of a waveform, respectively.

Propagation delay is the time difference between approximately 50% of the input transition and approximately 50% of the output transition.

**Measuring propagation delay using Waveform calculator**

Waveform calculator can be used to perform many different measurements and transformations on the waveforms displayed in the waveform window. This includes – computing the average of a waveform (e.g. power) over the entire length of the simulation or in a given period of time, finding the propagation delay of between input and output signals, or addition/subtraction/multiplication/division of waveforms, etc.

1. Execute **Tools → Calculator** in Virtuoso Visualization & Analysis XL window. ‘**Virtuoso Visualization & Analysis XL calculator**’ window will pop-up:
2. Select ‘vt’. Go to Schematic editor window and click on input node ‘in’. An expression (e.g. VT(“/in”)) will appear. Copy the expression.

3. In the Function Panel, select ‘Special functions’ and select ‘delay’.

4. The following window will appear. Put the expression previously obtained in the field ‘Signal1’. Do the same for output signal ‘out’ to fill in the field ‘Signal2’.

Fill up the rest of the form as follows:

5. Click OK. The following expression should appear:
6. Click on **Evaluate the buffer** icon.

The propagation delay (in seconds) will be displayed in the window.

---

### Measuring rise time and fall time using Waveform calculator

1. Open the `delay` function window under **Waveform calculator** in the same way that you followed for propagation delay measurement. This time both **Signal1** and **Signal2** will be `VT("/out")`.

2. **Threshold value 1 and 2** should be **0.12** (10% of 1.2 V supply) and **1.08** (90% of 1.2 V supply) respectively for 10% to 90% rise time calculation. **These values should be swapped for fall time calculation.**

3. **For rise time/fall time calculation, both the Edge numbers** must be the same.

4. **The Edge types** should be rising for rise time calculation and falling for fall time calculation.

*Example:* Rise time calculation of rising edge 2 for an inverter:
5. Click OK after filling in the form as shown above. The following expression should appear:

```
Process corner Simulation
```

Re-simulate the circuit using Fast Fast (FF) corner. For this purpose execute Setup → Model Libraries. Go to section and select FF_s1v. This will select the fast corner model. Re-simulate the circuit and find propagation delay.

```
Power Measurement using Waveform Calculator
```

In this tutorial, we will compute the average power consumed in a circuit for the duration of transient simulation window.

1. To do this, make sure that before running simulation you select the Outputs → Save All option in ADE L window. ‘Save Options’ window will appear. Under ‘Select power signals to output (pwr)’ option, put a tick mark in all option. Click OK.
2. Then simulate the circuit as usual, by executing **Simulation → Netlist and Run**. Execute **Tools → Result Browser** in ADE L window. ‘Result Browser’ window will appear to the left side in ‘Virtuoso Analysis and Visualization XL’ window.

3. Double-click on tran. From the signals list, double-click on :pwr
4. The waveform display window will show the “:pwr” (the instantaneous power consumed by the whole circuit) along with ‘in’ and ‘out’ signals.

5. Now, open Waveform calculator window. The calculator window appears. Make sure the “Wave” and “Clip” options are selected.
6. Now switch back to the waveform window and left click the mouse once on the power waveform. Then switch back to the calculator window. The buffer window should be filled in as follows:

7. Now select ‘average’ from ‘Special Functions’ Menu.

8. The buffer will now look like the following one:

9. Click on Evaluate the buffer icon and the average power dissipation in that time window will be displayed (about 1.838 μW in this example).

QUESTIONS:
Why does delay occur in logic gates? Why do logic gates consume power and when do they consume? What is the effect of delay on logic circuit design? What are the designer’s goals?
### Appendix: Cadence Virtuoso® Schematic Editor L Shortcuts

<table>
<thead>
<tr>
<th>Shortcut key</th>
<th>Tasks performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>Add a wire</td>
</tr>
<tr>
<td>i</td>
<td>Add an instance</td>
</tr>
<tr>
<td>p</td>
<td>Add a pin</td>
</tr>
<tr>
<td>l</td>
<td>Add label to a wire</td>
</tr>
<tr>
<td>e</td>
<td>Display options</td>
</tr>
<tr>
<td>q</td>
<td>Select an object and press q to open ‘Edit Object Property’ dialogue box</td>
</tr>
<tr>
<td>[</td>
<td>Zoom out</td>
</tr>
<tr>
<td>]</td>
<td>Zoom in</td>
</tr>
<tr>
<td>c</td>
<td>Copy</td>
</tr>
<tr>
<td>m</td>
<td>Move</td>
</tr>
<tr>
<td>u</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+u</td>
<td>Redo</td>
</tr>
<tr>
<td>f</td>
<td>Fit the entire schematic in the window</td>
</tr>
</tbody>
</table>
EEE 4134 VLSI I Laboratory
Lab 2
Parametric analysis of DC and transient simulation of an inverter and Symbol creation

Objectives:
- To learn how to perform parametric simulation in ADE L
- To learn how to create symbol view from schematic view

Parametric Analysis in ADE L: DC simulation

1. To open the schematic of inverter, execute File → Open in CIW. In the ‘Open File’ window, select the inverter schematic from the list. Click OK.

2. Schematic editor window will open. Execute File → Save a copy. In the following window, change the name of the cell to inverter2. Click OK. Close the schematic editor window and open the inverter2 cell from CIW.

3. This time we will perform both DC and parametric simulation at the same time on inverter schematic. We will obtain the transfer characteristics of inverter from DC simulation and by varying the width of the PMOS transistor; we will observe its effect on transfer characteristics.
Select the PMOS transistor in the schematic editor window, click ‘q’ and ‘Edit object properties’ window will open. Place w under ‘Total Width’ and press tab on keyboard. The ‘Finger Width’ field will be automatically changed as follows:

<table>
<thead>
<tr>
<th>CDF Parameter</th>
<th>Value</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Name</td>
<td>gpdk_090_pmos1v</td>
<td></td>
</tr>
<tr>
<td>Multiplier</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>100n M</td>
<td></td>
</tr>
<tr>
<td>Total Width</td>
<td>w M</td>
<td></td>
</tr>
<tr>
<td>Finger Width</td>
<td>iPar(&quot;w&quot;) / iPar(&quot;fingers&quot;) M</td>
<td></td>
</tr>
</tbody>
</table>

4. Place symbol of an instance vdc from analogLib to the schematic. In the ‘DC voltage’ field, type vin and press tab on keyboard.

5. Connect the voltage source between ‘in’ and ‘gnd’.

Now place another instance of vdc on the schematic, in the ‘DC voltage’ field, put 1.2 and connect it between vdd and gnd!

The final schematic should look like the following:
Click Check and Save.

6. Execute Launch → ADE L and setup Model Library to gpdk090_mos.scs and section to TT_s1v

7. Execute Variables → Edit and ‘Edit Design Variables’ window will open.

8. Select ‘Copy From’. You will see ‘w’ and ‘vin’ appear in the ‘Design Variables’ window. Click on ‘w’ and in ‘Value (Expr)’ field, put a default value of 480n. Click Apply. Similarly click on ‘vin’ and in ‘Value (Expr)’ field, put a default value of 0.6.
Then click OK.

9. Execute **Analyses ➔ Choose** and in the ‘Choosing Analyses’ form, select *dc*. Click on ‘save dc operating point’.

10. Under ‘Sweep Variable’, select ‘Design Variable’ and click on ‘Select design variable’.

Select ‘vin’ and click OK.

11. Under ‘sweep range’, select ‘start-stop’ and put a start value of 0 and a stop value of 1.2. Select ‘Sweep type’ to be ‘linear’ and ‘Step size’ to be 0.01. Click OK.
12. Execute Outputs ➔ To be plotted and select ‘out’ pin on the schematic. ADE L window will now look like the following:

14. Click on ‘Add Variable’. From the drop down menu, select ‘\(w\)’. Put From: 240n, To: 720n, **Step Mode**: linear steps, **Step Size**: 120n.

Click on ‘**Run selected sweeps**’ icon to start simulation.

The simulated waveforms will be displayed.
These transfer characteristics can be further explored to find Noise margin, and inversion voltage for different widths of PMOS.

15. This parametric analysis option can be saved for later use by clicking on the save icon.

In the following window, give a name to the configuration file and click **Save**.

![Save as Parametric File](image)

16. Also save the state of the ADE L window.

**Parametric Analysis in ADE L: Transient simulation**

1. For parametric analysis of transient simulation, open the schematic of the cell *inverter* from library *mylib* and save it as *inverter3*. Open the schematic of the cell named *inverter3*.

2. Change the ‘Total Width’ of PMOS to “w” as you have done before. Click **Check and save**.
3. Now, launch ADE L, and Setup Model Library, Analysis type, Stimuli and Outputs to be plotted in the same way as you have done in Lab 1.

4. Then execute Variables ➔ Edit. Select ‘Copy From’. “w” will appear under ‘Design Variables’, select it and enter a default value of 480n under ‘Value (Expr)’ and click OK.

5. Now, execute Tools ➔ Parametric Analysis. Select ‘w’ from the ‘Add variable’ list, and fill in the rest as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Sweep?</th>
<th>Range Type</th>
<th>From</th>
<th>To</th>
<th>Step Mode</th>
<th>Step Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>480n</td>
<td>✔</td>
<td>From/To</td>
<td>240n</td>
<td>720n</td>
<td>Linear Steps</td>
<td>120n</td>
</tr>
</tbody>
</table>

Click on ‘Run selected sweeps’.
6. From the waveform display window, change the time scale and observe a specific transition from high to low or low to high.

You can clearly see the effect of changing width on the inverter performance.

7. Let’s find the delays for this transition. Open Waveform calculator and fill in the form as follows for calculating the delay for this transition.

Click OK and click on ‘Evaluate the buffer and display the results in a table’ icon.

The results will be displayed in a table:
You can see that some of the propagation delays are negative. Can you explain why? Also notice that propagation delay increases with the width of PMOS transistor for a falling transition in inverter output. Can you explain why? What will happen during a rising transition?

Symbol Creation

In this section, you will create a symbol for your inverter design so that you can use this symbol view for the schematic in a hierarchical design. In addition, the symbol has attached properties (cdsParam) that facilitate the simulation and the design of the circuit.

1. Open the schematic of the cell `inverter` from library `mylib`. Save a copy named `INVXI`. Change the `width` of the PMOS transistor to 240n.

2. In the schematic editor window for `INVXI`, execute `Create ➔ Cellview ➔ From Cellview`. ‘Cellview from cellview’ window appears. Click OK.

In the ‘Symbol Generation Options’ window, you can choose the location of the pins.

Page 38
Click OK. Symbol Editor window will open.
3. Click **Delete** icon in the symbol window, delete the outer red rectangle, green rectangle and *instanceName* property.

4. Execute **Create → Shape → polygon**, and draw a shape similar to triangle. After creating the triangle, press **Esc** key.

5. Execute **Create → Shape → Circle** to make a circle at the end of the triangle. You can move the pin names according to the location.

6. Execute **Create → Selection Box**. In the ‘**Add Selection Box**’ form, click ‘**Automatic**’.

A new red selection box is automatically added.

7. After creating symbol, click on the save icon in the symbol editor window to save the symbol. In the symbol editor window, execute **File → Check and Save**. Then close the symbol editor window.

QUESTIONS:
What is the effect of load capacitance on delay and power consumption? What will happen to delay and power consumption if sizes of transistors are increased? What is the effect of transistor sizes on noise margin level? What is the significance of inversion voltage? What is the effect of process corners on delay and power consumption?
Objectives:

- To create a layout view of the basic inverter circuit from scratch in Virtuoso Layout Editor
- To design the layout keeping basic design rules in mind
- To design cell layout of a constant height for use in hierarchical design

Introduction to Layout, DRC and LVS

Layout is representation of a circuit in terms of planar geometric shapes (e.g. rectangles, polygons) showing the patterns of metal, polysilicon, oxide, or diffusion layers that make up the components (resistors, inductors, capacitors, transistors) of the integrated circuit.

When using a standard process (e.g. 45nm, 90nm or 180nm process available in our lab), the behaviour of the final integrated circuit depends significantly on the positions and interconnections of the geometric shapes due to parasitic resistances and capacitances contributed by them. While designing a layout, designer must keep in mind performance (e.g. power-delay product) and size (area occupied by the chip) criterion.

While designing digital circuits, one usually follows an ASIC design flow, where, the height of standard cells that are used is the same throughout the cell library, but their widths must vary according to their logical functions and drive strengths. The following figure shows a generalized standard cell height concept:

![Standard Cell Height Concept](image)

Although we will follow a full-custom IC design flow, we will maintain same cell height throughout our cell library.

The generated layout must pass a series of checks in a process known as physical verification. The most common checks in this verification process are:

- Design Rule Checking (DRC)
- Layout Versus Schematic (LVS) checking
- Parasitic extraction and post-layout simulation

Design Rule Check (DRC):

Design Rule Checking (DRC) is the process that determines whether the designed layout of a circuit satisfies a rules specified by the process being used.

Design Rules are a series of rules (e.g. area, width, overlap, enclosure, extension, spacing) provided by semiconductor manufacturers which are specific to a particular semiconductor manufacturing process. Design rules specify certain geometric and connectivity restrictions to ensure that the process can fabricate the device properly.
Layout versus Schematic (LVS) Check:
The Layout Versus Schematic (LVS) is the verification step to determine whether a particular integrated circuit layout corresponds to the original schematic or circuit diagram of the design. A successful Design rule check (DRC) ensures that the layout conforms to the rules designed/required for faultless fabrication. However, it does not guarantee if it really represents the circuit we desire to fabricate. This is why an LVS check is used.

**Layout design using Virtuoso Layout Suite L Editor**

1. Invoke Virtuoso Layout Suite L Editor from the CIW by executing **File ➔ New ➔ Cellview**. The ‘New File’ form appears. Fill it in as shown in the figure below:
   **Cell:** INVX1, **View:** layout. Click OK.

2. Click ‘Always’ if the following window appears before Layout window appears.

The following window of Virtuoso Layout Suite L Editor will appear.
On the left side of the window, you will find a panel called ‘Layers’. This panel is divided in three main categories which are: layer color, layer name and layer purpose. The details are described in the table below:

<table>
<thead>
<tr>
<th>Color</th>
<th>Matches the color in the Editing window. Each layer has its own color and pattern. Each layer has two colors associated with it: a fill color and an outline color. These colors can be changed to fit your taste by editing the technology file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The type of layer (Nwell, Oxide, Poly, Metal1, etc)</td>
</tr>
<tr>
<td>Purpose</td>
<td>In gpdk090 the only purpose classifications are: drw = drawing, slot = slot. Drawing is used in layout, slot is used to create a hole for metal stress relief</td>
</tr>
</tbody>
</table>

Verify that the layers display corresponds to the gpdk090 layers shown in the GPKD 90 nm Mixed Signal Process Specification manual (gpdk090_DRM.pdf).

3. Before starting to design layout, you need to set the layout display configuration. Execute the following in the Virtuoso Layout Editor: **Options → Display** or press ‘e’ on keyboard. Configure the form as shown in the figure below: You have to set the following parameters only:
4. Now we are going to build the layout of the inverter. An inverter has an NMOS and a PMOS transistor. First we will build an NMOS transistor.
Layout of NMOS inverter consists of oxide, Nimp, Cont and Poly layers. Study the rules of these layers and calculate the minimum size of the Poly, Cont, Oxide and Nimp layer to create an NMOS transistor.

The rules related to the NMOS transistor can be summarised as follows:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Minimum Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact size</td>
<td>(0.12 , \mu m \times 0.12 , \mu m) (Fixed)</td>
</tr>
<tr>
<td>Poly width (Minimum)</td>
<td>(0.1 , \mu m)</td>
</tr>
<tr>
<td>Contact to poly spacing (Minimum)</td>
<td>(0.1 , \mu m)</td>
</tr>
<tr>
<td>Contact to oxide enclosure (Minimum)</td>
<td>(0.06 , \mu m)</td>
</tr>
<tr>
<td>Poly/Nimp extending from oxide (Minimum)</td>
<td>(0.18 , \mu m) (gate side enclosure)</td>
</tr>
<tr>
<td>Nimpenclosing oxide (Minimum)</td>
<td>(0.14 , \mu m) (enclosure other than gate sides)</td>
</tr>
<tr>
<td>Minimum Metal 1 width</td>
<td>(0.12 , \mu m)</td>
</tr>
<tr>
<td>Maximum Metal 1 width</td>
<td>(12.0 , \mu m)</td>
</tr>
<tr>
<td>Minimum Metal 1 to Contact enclosure</td>
<td>(0.06 , \mu m) (on at least two opposite sides)</td>
</tr>
</tbody>
</table>

The following figure illustrates some of the design rules mentioned above:

Now study the PMOS transistor structure in the GPDK 90 nm Mixed Signal Process Spec. The PMOS transistor consists of Oxide, Poly, Pimp, Cont and Nwell layer. Study the rules of these layers and calculate the minimum size of Poly, Cont, Oxide, Pimp and Nwell layer to create a PMOS transistor. The rules related to PMOS are same as NMOS except the there is an additional layer, the Nwell, whose rules are as follows:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Minimum Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Nwell width</td>
<td>(0.6 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell spacing to Nwell (same potential)</td>
<td>(0.6 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell spacing to Nwell (different potential)</td>
<td>(1.2 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell spacing to N+ active area</td>
<td>(0.3 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell spacing to P+ active area</td>
<td>(0.3 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell enclosure to P+ active area</td>
<td>(0.12 , \mu m)</td>
</tr>
<tr>
<td>Minimum Nwell enclosure to N+ active area</td>
<td>(0.12 , \mu m)</td>
</tr>
<tr>
<td>Minimum N+ Active Area to P+ Active Area Spacing</td>
<td>(0.16 , \mu m)</td>
</tr>
</tbody>
</table>
Now we start building the NMOS and PMOS transistor layout. Look at the LSW and find the current drawing layer.

5. Click on the following icon in **Virtuoso Layout Suite L Editor** window so that it notifies you anytime you make a violation of any design rule. When clicked, it will show ‘**DRD Notify ON**’. DRD stands for Design Rule Driven.

6. Select ‘**Cont (drw)**’ (contact) layer from the ‘**Layers**’ panel and draw a rectangle of ‘**Cont (drw)**’ layer using **Create ➔ Shape ➔ Rectangle** or simple pressing ‘**r**’. Press ‘**Esc**’ to stop ‘**create rectangle**’ tool. In gpdk090 technology, **Cont** layers must be of dimension **0.12 μm x 0.12 μm**. So, if your rectangle is not of that dimension, click on the rectangle, press ‘**q**’. In the following window, check if the criterion has been met and change ‘**Width/Height**’ if required.

![Edit Rectangle Properties](image)

**Contact to poly spacing** must be **0.1 μm** in this technology and the **channel length** of NMOS/PMOS in our design is **0.1 μm**. So, we need a minimum space of **0.3 μm** between the contacts at source and drain.

7. Press ‘**k**’ to invoke the ‘**ruler**’ tool. Use it to measure lengths whenever needed. To copy, press ‘**c**’. After placing two contacts, the layout looks like this:

![Layout with contacts](image)

8. Now, **contact to oxide spacing** is minimum **0.06 μm**. So, draw a rectangle of ‘**Oxide (drw)**’ layer so that it covers both the contacts and extends from each side by **0.06 μm**.

While drawing this, you will see Design rule violations when they are committed.
9. After drawing oxide layer, the layout should look like this:

10. Now we will draw ‘Nimp (drw)’ layer, which must extend from the oxide layer by a minimum of 0.14 μm. First, draw a rectangle and then extend it to meet design rules. Use stretch tool by pressing ‘s’. Layout will look like the following:

11. Now, copy it and create another copy of all these layers by selecting all and pressing ‘e’.
12. Click on the ‘Nimp (drw)’ layer of the copy in the upper portion of the layout and press ‘q’ to edit properties. From ‘Edit Rectangle Properties’ window, select ‘Pimp (drw)’ layer under ‘Layer’ option. Click OK.

13. With more metal layers available in today’s silicon processes, using the routing approach, such as first metal traverse vertically and second metal traverse horizontally, would be advantageous in standard cell physical design. Using this method, the second layer (e.g. Metal2) can be used for power and ground routing over internal standard cell transistors. In standard cell layout, it is preferable to use first conducting layer, such as Metal1, as much as possible to make internal connections of NMOS and PMOS transistors within the cell. If there is a need to use other conducting layers, such as, Metal2, use of such layers must be kept to a minimum. It is desired to use first routing (e.g. Metal1) layer for standard cell ports.

Our cells will have a height of 5 μm. Place the two parts (NMOS and PMOS) 2 μm apart, and create a ruler so that the cell height can be checked whenever needed and the separation between the NMOS and PMOS can be maintained properly. Now, the layout will look like the following:
14. Next, draw a ‘Poly (drw)’ path by selecting ‘Poly’ layer from the ‘Layers’ panel and pressing ‘p’ to invoke ‘create path’ tool. This layer must be of 0.1 \( \mu m \) in width and in between the two contacts, extending from the oxide layer by 0.18 \( \mu m \) (at least, on both sides). After placing the ‘poly’ gate, the layout will look like the following one:
15. Now that you know most of the shortcuts and layers, draw contact for body terminals for NMOS and PMOS. These portions should consist of Cont, Oxide and Nimp (for body of PMOS) or Pimp (for body of NMOS). Check DRD notifications for design rule violations. The following figure shows a Psubstrate and an Nwell contact. The measurement dimensions are shown only on the left one, as they are same for both contacts.
16. Connect the Drain regions of the NMOS and PMOS. Also connect the source of both MOS’s to respective body terminals using ‘Metal1 (drw)’ layer. Connect the drains of the MOS’s using ‘Metal1 (drw)’ layer.

17. PMOS should be in ‘Nwell (drw)’. So draw an ‘Nwell (drw)’ rectangle surrounding both the PMOS and the body contact for PMOS. The layout will look like the following:
18. Now, we have to place pins. The gate is in ‘poly (drw)’ layer. Let’s bring it to ‘Metal1(drw)’ layer by extending the ‘Poly (drw)’ layer, creating a contact between ‘Poly’ and ‘Metal1’ layer by pressing ‘o’ to ‘create via’ and selecting ‘M1_POv’ under ‘via definition’ and placing it on layout.

19. Also draw a ‘Metal1 (drw)’ rectangle on the via, because the default Metal1 rectangle area is less than the required minimum.
20. Now, Execute *Create \(\rightarrow\) Pin* to create pins for vdd!, gnd!, in and out.

For in, vdd! and gnd! select ‘input’ as ‘I/O type’ and for out select ‘output’ as ‘I/O type’. Now, draw rectangles on the Poly-Metal1 via for ‘in’ pin, PMOS source-to-body ‘Metal1’ connection for ‘vdd!’ pin and NMOS source-to-body connection for ‘gnd!’ pin. For ‘out’ pin, draw the rectangle on the Metal1 layer connecting the two drains of MOS’s.
You may add label to pins.

21. Finally, add Metal2 paths of 0.5 $\mu$ width for power rails and connect them to power nets in Metal1 by using Metal1 to Metal2 via by invoking ‘create via’.

The final layout will look like the following:
QUESTIONS:
What is layout design? What rules a designer must keep in mind while designing layout? Why does DRC rules are provided by the foundry? What principles in layout designing are best practices for a designer?
## Appendix A (Shortcut keys for Cadence Virtuoso® Layout Editor L)

<table>
<thead>
<tr>
<th>Shortcut Key</th>
<th>Tasks performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Fit display to window</td>
</tr>
<tr>
<td>r</td>
<td>Draw rectangle</td>
</tr>
<tr>
<td>q</td>
<td>Edit property of an object</td>
</tr>
<tr>
<td>p</td>
<td>Makes a min width path of the layer selected in LSW</td>
</tr>
<tr>
<td>Ctrl+a</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+d</td>
<td>Deselect all</td>
</tr>
<tr>
<td>c</td>
<td>Copy</td>
</tr>
<tr>
<td>m</td>
<td>Move</td>
</tr>
<tr>
<td>s</td>
<td>Stretch side of a rectangle</td>
</tr>
<tr>
<td>k</td>
<td>Invoke ruler tool</td>
</tr>
<tr>
<td>Shift+k</td>
<td>Delete all rulers</td>
</tr>
<tr>
<td>i</td>
<td>Add an instance</td>
</tr>
<tr>
<td>u</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+u</td>
<td>Redo</td>
</tr>
<tr>
<td>e</td>
<td>Display options</td>
</tr>
<tr>
<td>o</td>
<td>Add via between layers</td>
</tr>
<tr>
<td>l</td>
<td>Create a label</td>
</tr>
</tbody>
</table>
Appendix B (gpdk090 Design Rules Guide (Abridged Version for VLSI-I Lab))

Terminology Definitions

**Spacing** - distance from the outside of the edge of a shape to the outside of the edge of another shape.

![Spacing Diagram]

**Enclosure** - distance from the inside of the edge of a shape to the outside of the edge of another shape.

![Enclosure Diagram]

**Overlap** - distance from the inside of the edge of a shape to the inside of the edge of another shape.

![Overlap Diagram]

**Butting** - outside of the edge of a shape touching the outside of the edge of another shape.

![Butting Diagram]
<table>
<thead>
<tr>
<th>Description</th>
<th>Value, (µm/µm²) wherever applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum oxide (active) area</td>
<td>0.06</td>
</tr>
<tr>
<td>Minimum 1.2V N/P Channel gate length</td>
<td>0.1</td>
</tr>
<tr>
<td>Minimum poly interconnect width</td>
<td>0.1</td>
</tr>
<tr>
<td>Minimum gate/poly interconnect space</td>
<td>0.12</td>
</tr>
<tr>
<td>Minimum N/P-channel gate extension beyond active area</td>
<td>0.18</td>
</tr>
<tr>
<td>Minimum poly interconnect to related/unrelated active area space</td>
<td>0.1</td>
</tr>
<tr>
<td>Minimum poly interconnect area</td>
<td>0.1</td>
</tr>
<tr>
<td>Bent gate is not allowed</td>
<td></td>
</tr>
<tr>
<td>Minimum N+/P+ implant width</td>
<td>0.24</td>
</tr>
<tr>
<td>Minimum N+/P+ implant space</td>
<td>0.24</td>
</tr>
<tr>
<td>Minimum N+/P+ implant to active area enclosure</td>
<td>0.14</td>
</tr>
<tr>
<td>Minimum N+/P+ implant to gate side enclosure</td>
<td>0.18</td>
</tr>
<tr>
<td>Minimum N+ to P+ active area (inside Nwell) spacing</td>
<td>0.16</td>
</tr>
<tr>
<td>Minimum N+/P+ implant area</td>
<td>0.15</td>
</tr>
<tr>
<td>N+ implant is not allowed over P+ implant</td>
<td></td>
</tr>
<tr>
<td>Minimum P+ to N+ active area (outside Nwell) spacing</td>
<td>0.16</td>
</tr>
<tr>
<td>Maximum and minimum Contact width/length</td>
<td>0.12</td>
</tr>
<tr>
<td>Minimum Contact to Contact spacing</td>
<td>0.14</td>
</tr>
<tr>
<td>Minimum Contact on Active Area to gate spacing</td>
<td>0.1</td>
</tr>
<tr>
<td>Minimum gate Contact to Active Area spacing</td>
<td>0.12</td>
</tr>
<tr>
<td>Minimum Active Area to Contact enclosure</td>
<td>0.06</td>
</tr>
<tr>
<td>Minimum Poly to Contact enclosure</td>
<td>0.04</td>
</tr>
<tr>
<td>Minimum Poly to Contact enclosure on at least two opposite sides (end of line)</td>
<td>0.06</td>
</tr>
<tr>
<td>Contact on gate is not allowed, Contact must be covered by Metal1 and active area/poly</td>
<td></td>
</tr>
<tr>
<td>Minimum Metal 1 width</td>
<td>0.12</td>
</tr>
<tr>
<td>Maximum Metal 1 width</td>
<td>12</td>
</tr>
<tr>
<td>Minimum Metal 1 to Metal 1 spacing</td>
<td>0.12</td>
</tr>
<tr>
<td>Minimum Metal 1 to Contact enclosure</td>
<td>0</td>
</tr>
<tr>
<td>Specification</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Minimum Metal 1 to Contact enclosure on two opposite sides of the Contact</td>
<td>0.06</td>
</tr>
<tr>
<td>Minimum Metal 1 area</td>
<td>0.07</td>
</tr>
<tr>
<td>Minimum Metal k width</td>
<td>0.14</td>
</tr>
<tr>
<td>Maximum Metal k width</td>
<td>12</td>
</tr>
<tr>
<td>Minimum Metal k enclosure of Via k-1</td>
<td>0.005</td>
</tr>
<tr>
<td>Minimum Metal k enclosure of Via k-1 on at least two opposite sides</td>
<td>0.06</td>
</tr>
<tr>
<td>Minimum Metal k area</td>
<td>0.08</td>
</tr>
<tr>
<td>Minimum and maximum Via k width</td>
<td>0.14</td>
</tr>
<tr>
<td>Minimum Via k to Via k spacing</td>
<td>0.15</td>
</tr>
<tr>
<td>Minimum Metal k to Via k enclosure</td>
<td>0.005</td>
</tr>
<tr>
<td>Minimum Metal k to Via k enclosure on at least two opposite sides of Via k</td>
<td>0.06</td>
</tr>
<tr>
<td>Minimum of four Via k with spacing &lt;= 0.30µm or nine Via k with spacing &lt;= 0.60µm are required when connecting Metal k and Metal k+1 when one of the Metals has a width &gt; 1.0µm at the connection point</td>
<td></td>
</tr>
<tr>
<td>Minimum Nwell width</td>
<td>0.6</td>
</tr>
<tr>
<td>Minimum Nwell spacing to Nwell (same potential)</td>
<td>0.6</td>
</tr>
<tr>
<td>Minimum Nwell spacing to Nwell (different potential)</td>
<td>1.2</td>
</tr>
<tr>
<td>Minimum Nwell spacing to N+/P+ Active Area</td>
<td>0.3</td>
</tr>
<tr>
<td>Minimum Nwell enclosure of N+/P+ Active Area</td>
<td>0.12</td>
</tr>
</tbody>
</table>
EEE 4134 VLSI I Laboratory
Lab 4
DRC, LVS, RCX and Post-layout simulation of an inverter

Objectives:
- To perform Design rules check (DRC), Layout vs. Schematic check (LVS) of inverter layout
- To extract parasitic resistance and capacitance from layout of designed inverter
- To perform transient simulation of extracted view
- To create layout views for body ties of NMOS and PMOS for further use

DRC Rules check by Cadence’s ASSURA

1. Now we would like to check the DRC rules by ASSURA. Execute Assura ➔ Technology. In the following window, type the path of ‘Assura Technology File’ as shown. Click OK.

   ![Assura Technology File Path](image)

2. Execute Assura ➔ Run DRC. A DRC window appears as shown below. Fill the form as indicated in the picture.

   Give a Run name. Select ‘gpdk090’ under ‘Technology’. Then click OK.
3. A DRC completed window appears as shown below, after completion of DRC run:

![DRC Run Result](image)

The DRC run "drc1" from 
"/home/fall16/120105001/cic/" has completed successfully.

Do you want to view the results of this run?

Click Yes.

4. ‘Error layer Window’ (ELW) appear as shown below with INVX1 layout window which shows the errors.
5. To correct the errors, find the error location by clicking on the error and then clicking on the right arrow key on ELW. To hide all the errors, click on ‘NV’ (no Layers visible) button. The particular error in the following figure is due to a lower than 0.18 \( \mu \text{m} \) enclosure of gate by Pimp (drw) layer. Stretch the Pimp layer to correct it. Similarly, do this for all errors in your layout. No one can list all the errors one may commit, try correcting one after another and be familiar with them by solving them through practice.

After correcting all the errors, run the DRC again.

6. If your design is error free, you should get the following message.
LVS check by Cadence’s ASSURA

1. Execute Assura → Run LVS. Select gpdk090 and give a Run name.

Click OK.

2. After completion of lvs run, you will get a result window. If you have done everything right, it will say that Schematic and Layout match.

Suppose that, you have done a mistake, your schematic says the PMOS width is 480n where layout says it is 240n. What will happen in LVS report? Let’s see.
Note that, LVS report says that 1 cell had parameter mismatch. That means one cell had two different dimensions in schematic and layout. Click OK. ‘LVS Debug’ window will appear:

3. Select INVX1 (mylib). Notice that summary window shows the list of errors.

4. Only parameter mismatch has occurred. Click on Parameters and Open Tool.
5. ‘Parameters mismatch tool’ will open.

6. Select PM0 pmos1v. Now in the **Message** box, you can see the mismatch error.

7. Change the width of PMOS to the value of width in layout and run LVS again. This time you will get an LVS match.
Parasitic Extraction by Quantus QRC

1. Execute Assura ➔ Open Run. Select the final error free lvs run name in the run name field (it automatically loads the last lvs run). Click OK.

2. Execute Assura ➔ Run Quantus QRC. Select Extracted View in the output field under Setup tab.
3. Go to Extraction tab. Select RC as Extraction type and put the name of your reference node (gnd! in the given case) and click OK.
4. After completion of the run, you will get the following message:

- Quentus QRC Run
- The Quentus QRC run "lvs1" completed successfully
- The output is in:
- Library: mylib
- Cell: INVX1
- View: av_extracted

5. Execute File ➔ Open and fill in the form as follows. Click OK.

- Open File
- Library: mylib
- Cell: INVX1
- View: av_extracted
- Type: layout
- Application
- Open with: Layout L
- Library path file: /home/fall16/120105001/cic/cds.lib

Extracted view will open.
6. Now Launch ADE L from the av_extracted view and setup everything other than outputs to be plotted in the same way as you have done in Lab 1. After setting everything else, execute Outputs → To be plotted → Select on design. Go to av_extracted view. Click on the in pin location on that view. The following window will appear. Select pin name in and click OK. Do the same for out pin.

Then run the simulation and observe the output waveforms. Then measure power and delay using waveform calculator.

QUESTIONS:
What happens to delay and power consumption after parasitic RC effects are included? And explain why it happens.
Creating Layout views of Body ties

Now you know what body tie is. We will now make two instances for body ties one for \texttt{Psub} and one for \texttt{Nwell}.

1. Execute \textit{File} \rightarrow \textit{New} \rightarrow \textit{Cellview} and fill in the New File form as follows. Click \textbf{OK}.

2. Draw psubstrate contact in the same way as you have made it in Lab3.
3. Save it and make nwell contact similarly (name it M1_NWELL), just change the Pimp layer to Nimp and everything else is the same.

Save these two for later use.
Objectives:
- To be familiar with schematic-driven layout with the example of a 2-input NAND gate.
- To perform Schematic Level Verification, Layout Design, DRC and LVS check and perform post-layout simulation from extracted view.

Creating Layout using Virtuoso Layout Editor XL

1. Virtuoso Layout Editor XL is a schematic-driven layout generation tool. To learn schematic driven layout we will create the schematic view of a 2-input NAND gate cell which we named NAND2X1.

2. Instantiate the following cells to your schematic:

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Cell Name</th>
<th>Properties/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpdk090</td>
<td>nmos1v</td>
<td>For NM0 and NM1, Width = 2400n</td>
</tr>
<tr>
<td>gpdk090</td>
<td>pmos1v</td>
<td>For PM0 and PM1, Width = 4800n</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd</td>
<td></td>
</tr>
<tr>
<td>analogLib</td>
<td>gnd</td>
<td></td>
</tr>
</tbody>
</table>

Use your experience from Lab1 to draw the schematic diagram of the nand gate.
3. Launch ADE L and simulate the design to verify its functionality. Setup Model library, Analysis type and Outputs to be plotted as you have done in Lab1. While setting inputs for signals A and B, you have to use different periods and delays for the two signals, so that you can observe all four cases (00, 01, 10, 11) of input signals. Also make sure 0→1 and/or 1→0 transitions for both input signals do not occur at the same time. The following figure shows a sample of two signals meeting these criteria:

A sample waveform window would look like the following after simulation:

Check the functionality of the schematic (whether it acts like a nand gate).
4. Then create a symbol in the same way you have made symbol of inverter in lab2. Make it look like a nand gate.

![NAND gate diagram]

5. In schematic editor window, execute: **Launch → Layout XL**. The following window will appear:

6. Click OK. ‘New File’ window for layout will appear. Click OK.

‘Virtuoso Layout Editor XL’ window will appear.
7. Execute **Connectivity → Generate All From Source**. The following pop-up window will appear:
8. Go to I/O pins tab. The dialog box shows that all I/O pins are in Metal1 layer (Metal1 drw). Also put a tick mark on Create label as Label. Click Options. Set Height to 0.1.
Click OK, and OK.

9. The initial pin and transistor placement in layout will look like the following:
10. Execute *Options → Display* or Press ‘e’ on keyboard to open ‘Display Options’. Fill it in as shown:
The transistors and pins are shown inside a bounding box, which is an estimate of the optimum size of the final layout. Automatic router will use the bounding box to constrain all routing to occur within the box. The bounding box may need to be re-sized to accommodate all components. An important concept to keep in mind during resizing is that standard cells typically have fixed height (so that power/ground rails line up correctly for routing purposes).

Delete the PR Boundary for now.

Virtuoso Layout Editor XL (VXL) and gpdk090 allow us to create stacked transistors with shared source/drain areas. Zoom in to two transistors at the bottom (to zoom in, type “z” and draw a box around the transistors). Click on the transistor on the right and type “m” to move the object. As you start dragging the object to the left, fly-lines indicating connectivity will appear as shown below:
12. When the source/drain areas are overlapped, left-click to fix the position. You should see a transistor stack with shared source/drain areas like this (depending on how far you move, you may need to move left/right a bit):

This is a nice NMOS stack for the NAND gate. As you can see, the source/drain contacts have disappeared. Back to the big picture, zoom to fit (press “f”).
Let’s do the same exercise for the PMOS transistors. The PMOS transistors in nand gate do have shared drain contacts because they work in parallel. Connectivity information is extracted from schematic by VXL. The pull-up network looks like the following:

13. Now, connect different layers using path tool (press ‘p’ on keyboard), and fill areas by drawing rectangles where necessary (press ‘r’ on keyboard). To connect one layer to another (e.g. Poly to Metal1 or Metal1 to Metal2), create via by pressing ‘o’ on keyboard and selecting proper ‘Via Definition’.

14. Instantiate M1_PSUB and M1_NWELL cells (that you have created earlier) by pressing ‘i’ on keyboard and selecting the layout view from library browser.
15. Wire up the layout. When you do so, you may encounter multiple options for certain pins. For example, when you select the PMOS to connect its source to VDD, there are multiple Metal1 wires in the PMOS. The desired path will be highlighted and you’ll see the fly-line. Continue until you finish routing all the signals. Move vdd! and gnd! pins to the power rails. As you are moving the pins around, notice the fly-lines that indicate the connections.

A practice that you can follow while wiring is to use Metal1 for all vertical wiring and Metal2 for all horizontal wiring inside the cell. Also make the cell height 5 \( \mu m \).

16. Your final layout will look something like the following:
17. Perform DRC, LVS and QRC for NAND2X1 as you have done in Lab 4. Generate `av_extracted` view and simulate the circuit from that view to verify the functionality.

**QUESTIONS:**
Why should you stack transistors in a layout which have shared drain/source? What are the differences between stacked transistors and unstacked ones in terms of performance of post-layout simulation?
Introduction to Hierarchical Design (2-input AND gate using 2-input NAND gate and an inverter)

Objectives:
- To be familiar with concept of hierarchical design
- To perform Schematic Level Verification, Layout Design, DRC and LVS check
- To perform post-layout simulation of top level design

Introduction to Hierarchical Design

By this time, you should have completed layout of \textit{INVX1} and \textit{NAND2X1}. Now, you will learn how to perform hierarchical design.

1. Create a new cellview of type schematic named \textit{AND2X1}.
2. Instantiate \textit{NAND2X1} and \textit{INVX1} symbol from your library \textit{mylib} in that schematic.
   Your final schematic should look something like the following:

![Schematic Diagram]

3. Now make a symbol of \textit{AND2X1}.

![Symbol Diagram]

4. Now create a new cellview named \textit{AND2X1\_test}, instantiate \textit{AND2X1} and \textit{vdd} in that cell and the final schematic should look like the following:

![Final Schematic]
5. Now, launch ADE L, setup Stimuli, Model library, Analysis type and Outputs to be plotted in the same way as you have done in Lab 5. Run the simulation.

6. If functional verification is okay, then execute Launch ➔ Layout XL from the schematic of AND2X1.

7. Follow procedure of Lab 5 to generate instances and set display options. You will get something like the following:
8. Connect them as required and the final layout should look like the following (probably better!):
9. Perform DRC, LVS and QRC as you have done in Lab 4 and Lab 5.

**QUESTIONS:**
What is hierarchical design? What design principles will make hierarchical layout design easier?
Introduction to Verilog HDL and Quartus II

Objectives:

- To get familiar with Quartus II
- To get introduced to Hardware Description Language (HDL)
- To understand behavioral and structural Verilog descriptions

Verilog HDL

A hardware description language (HDL) is similar to a typical computer programming language except that an HDL is used to describe hardware rather than a program to be executed on a computer. Two HDLs are IEEE standards: Verilog HDL and VHDL (Very High Speed Integrated Circuit Hardware Description Language).

Creating a New Project

1. Run Quartus II by clicking on the shortcut icon on desktop named *Quartus II 9.0sp1 Web Edition*. A getting started window may open up. You may put a tick on ‘Don’t show this screen again’, if you do not want this to appear again.

2. Each logic circuit or sub-circuit being designed with Quartus II software is called a *project*. The software works on one project at a time and keeps all the information for that project in a single directory (folder) in the file system. To begin a new logic circuit design, the first step is to create a directory to hold its files. Create a folder in *D:\*, *E:\* or *F:\* drive named after your *student ID*. Execute File ➔ *New Project Wizard*. Click *Next* on the window that opens up.

3. In the following window, change the working directory of the project to your directory (e.g. *D:\120105001*) and give a name to the project as shown. Note that the project must have a name, which is usually the same as the top-level design entity that will be included in the project. Click *Next*. 


4. We can specify any existing file to be included in the project, if we want, in the next window. But we will skip it for now. Click Next on the next three windows that open up. Then in the last window, click Finish.

Creating a new Verilog HDL file under a project (Design Entry), Compilation and Simulation

1. Execute File ➔ New. A window will open up. Select Verilog HDL File and click OK.

2. The following Text Editor Window will open up and you can write your Verilog code in here and save it using the same name as the project.

3. Write your Verilog code. This example shows Verilog code for a 2to1 MUX.
The syntax of Verilog code is sometimes difficult for a designer to remember. To help with the issue, the Text Editor provides a collection of Verilog templates. The templates provide examples of various types of Verilog statements, such as a module declaration, an always block, and assignment statements. It is worthwhile to browse through the templates by selecting Edit → Insert Template → Verilog HDL to become familiar with this resource.

4. Click on the purple play button to start compilation of your Verilog code.

5. After successful compilation, you will get the following message. Ignore the Warnings for now. Click OK.

If the compiler does not report zero errors, then there is at least one mistake in Verilog code. In this case, a message corresponding to each error found will be displayed in the Messages window. Double-clicking on an error message will highlight the statement which is affected by the error, in the Verilog code in the Text Editor window. The user can obtain more information about a specific error by selecting the error and pressing the F1 function key. Correct the error and recompile the design.

6. Now, execute File → New to create a vector waveform file which is required for simulating inputs and outputs. Select Vector Waveform File from the list and click OK.

7. The following window will open up.
8. Double-click on the white space under ‘Name|Value at 15.08 ns’. Or Right-click on that space and select **Insert ➔ Insert Node or Bus**.

9. In the ‘**Insert Node or Bus**’ window, click **Node Finder**.

10. In the ‘**Node Finder**’ window, Click **List**. Make sure ‘**Pins:all**’ is selected under ‘**Filter**’.

11. Now the window will look like the following one. Click on the ‘>>’ Button.
12. Now, it should appear like the following. Click OK.

13. In the following window, click OK.
14. The vector waveform file will now look like the following:

Now, you can clearly see the inputs and outputs.

15. Select an input and from the left palette, click on the ‘Overwrite clock’ icon.

16. In the ‘Clock’ window, set parameters of the clock.
17. Now, after setting all the input clocks, Vector Waveform File will look like the following. Note that, the output Y is displayed as having an unknown value at that time, which is indicated by a hashed pattern; its value will be determined during simulation.

18. Save the Vector Waveform File. It must have the same name as the Verilog file.

19. A designed circuit can be simulated in two ways. The simplest way is to assume that there is no delay in propagation of signals through the circuit. This is called functional simulation. A more complex way is to take all propagation delays into account, which leads to timing simulation. Typically, functional simulation is used to verify the functional correctness of a circuit as it is being designed. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit.

Click on the blue play button and you should observe the simulated waveforms.
Note there is a delay between input and outputs which simulates the real effect of gate delays. If you are interested in only functional analysis rather than timing analysis, go through some more steps.


21. In the following window, select Functional as Simulation Mode and click on Generate Functional Simulation Netlist.

22. After generating functional simulation netlist, the following window will appear. Click OK.

23. Click on the blue play button and you should observe the following message.
24. Now observe the simulation waveforms from *Processing ➔ Simulation Report*, and note that there is no time delay between inputs and output and the function of this code is indeed that of a 2to1 MUX.

**QUESTIONS:**

1. Which ones of the following identifier names are incorrect according to Verilog syntax?
   - 2to1MUX, MUX$2to1, mux_2to1, reg, reg4bit, 130105_counter

2. What are the differences between ‘structural’ and ‘behavioral’ Verilog codes? Show an example of each for a particular logic circuit.
Objectives:

- To design combinational circuits with Verilog codes, and verify the codes through simulation
- To learn about hierarchical design using Verilog HDL

Verilog codes for combinational logic circuits

**Half-Adder**

```verilog
define half_adder(A,B,Cout,S);
  input A,B;
  output S, Cout;
  assign S = A^B;
  assign Cout = A&B;
endmodule
```

**2-to-1 MUX**

```verilog
define mux21_fn(I0,I1,S,f);
  input I0,I1,S;
  output reg f;
  always @(I0,I1, S)
  begin
    if (S == 0)
      f = I0;
    else
      f = I1;
  end
endmodule
```

**Priority Encoder**

```verilog
define priority(W, Y, z);
  input [3:0]W;
  output reg [1:0]Y;
  output reg z;
  always @(W)
  begin
    z = 1;
    casex (W)
    4'b1xxx: Y = 3;
    4'b01xx: Y = 2;
    4'b001x: Y = 1;
    4'b0001: Y = 0;
    default: begin
      z = 0;
      Y = 2'bx;
    end
  endcase
endmodule
```
Full Adder using Half-Adders

```verilog
module FA_001(A, B, C, Cout, S);
input A, B, C;
output Cout, S;
wire C1, C2, S1;

HA_001 f1(A, B, C1, S1);
HA_001 f2(S1, C, C2, S);
assign C = C1 | C2;
endmodule

module HA_001(a, b, c, s);
input a, b;
output c, s;
assign c = a & b;
assign s = a ^ b;
endmodule

2-to-4 Decoder

```verilog
module dec2to4(W, En, Y);
input [1:0] W;
input En;
output reg [0:3] Y;
integer k;
always @(W, En)
for(k = 0; k <= 3; k = k+1)
if ((W == k) && (En == 1))
Y[k] = 1;
else
Y[k] = 0;
endmodule
QUESTIONS:
1. What are the differences between ‘concurrent’ and ‘procedural’ statements? Show an example of each.
2. Examine whether the following two code segments will yield different outputs. Explain your reasoning.

![Code segments]

3. Use ‘subcircuit’ concept of Verilog syntax to make a 4-to-1 MUX using 2-to-1 MUXes. Write the Verilog code.
4. Read the following Verilog code and find out what it does. State your reasoning.

4. Write Verilog codes for-
1 digit BCD Adder, 4-to-1 MUX, 4-bit comparator, 7-segment decoder, n-bit adder, 4-bit Full-Adder using 1-bit Full-Adder modules, 16-to-1 MUX using 4-to-1 MUXes
EE 4134 VLSI I Laboratory
Lab 9
RTL synthesis and Sequential Logic Circuit design in Verilog HDL using Quartus II

Objectives:
- To get familiar with RTL Synthesis in Quartus II
- To design sequential logic circuits in Verilog HDL, and verify the codes through simulation

RTL Synthesis in Quartus II

Logic synthesis is a process by which an abstract form of desired circuit behavior, is turned into a design implementation in terms of logic gates, by a synthesis tool. Common examples of this process include synthesis of HDLs, including VHDL and Verilog. Some synthesis tools generate bit-streams for programmable logic devices such as PALs or FPGAs, while others target the creation of ASICs. Logic synthesis is one aspect of electronic design automation.

In digital circuit design, register-transfer level (RTL) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. Register-transfer-level abstraction is used in hardware description languages (HDLs) to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

1. Write Verilog code for a logic circuit, compile it and verify its functionality through simulation.

Suppose, you have completed all these steps for Verilog code of a half-adder.

```
module halfadder (A, B, Cout, S);
input A, B;
output S, Cout;
assign S = A&B;
assign Cout = A&B;
endmodule
```

Then you can proceed to obtain RTL view for this logic circuit.

2. Execute Tools ➔Netlist Viewers ➔RTL Viewer. A window will appear and you can see the RTL view.

As you can see, this is the logic circuit for a half-adder.
Verilog codes for sequential logic circuits

Latch

```verilog
module latch(D, clk, Q);
    input D, clk;
    output reg Q;
    always@(D, clk)
    if (clk)
        Q = D;
endmodule
```

Flip-flop

```verilog
module flipflop(D, Clock, Q);
    input D, Clock;
    output reg Q;
    always@(posedge Clock)
        Q = D;
endmodule
```

Shift Register

```verilog
module shift3(w, Clock, Q);
    input w, Clock;
    output reg [1:3] Q;
    always@(posedge Clock)
        begin
            Q[3] = w;
            Q[2] = Q[3];
            Q[1] = Q[2];
        end
endmodule
```

Counter

```verilog
module count4(Clock, Resetn, E, Q);
    input Clock, Resetn, E;
    output reg [3:0] Q;
    always@(posedge Clock, negedge Resetn)
    if(Resetn == 0)
        Q = 0;
    elseif (E)
        Q = Q + 1;
endmodule
```
QUESTIONS:

1. Find out the function of following logic circuit and write a Verilog code to obtain the same logic circuit in ‘RTL Viewer’.

2. Write Verilog codes for an n-bit up/down counter, which counts up if select pin=0 and counts down if select pin=1.

3. Write Verilog code for a universal shift register which can shift left/right and can load data both in parallel mode and serial mode.

4. Write Verilog code for a J-K flip-flop using D flip-flop and a T flip-flop using J-K flip-flop

5. Synthesize already written Verilog codes to obtain RTL diagrams.

6. Write the Verilog code for a logic circuit which counts to 3 if selection input (composed of 2 bits) \( S_1S_0 = 01 \), counts to 7, if \( S_1S_0 = 10 \), counts to 15, if \( S_1S_0 = 11 \). If \( S_1S_0 = 00 \), then it halts counting. Use positive edge-triggered counter.

7. What are the differences between ‘synchronous’ and ‘asynchronous’ resettable flip-flops?

8. Find 10 mistakes in the following Verilog code which implements a 5-bit shift register:

   ```verilog
   module 8ques(w,clk,Q)
   Parameter n=5;
   input w,clk
   output [n-1:1]q;
   always@(posedge clk);
   begin
     q[5]=w;
     q[4]=q[5];
     q[3]=q[4];
     q[2]=q[3];
     q[1]=q[2];
   end
   end
   ```

9. What are the differences between ‘blocking’ and ‘non-blocking’ assignments? Show how only the use of them creates entirely different logic circuits.
References and Further Readings

   by Neil Weste, David Harris
   by Stephen Brown, Zvonko Vranesic
3. *Physical Design Essentials: An ASIC Design Implementation Perspective*
   By Khosrow Golshani
4. *Digital VLSI Chip Design with Cadence and Synopsys CAD tools*
   by Erik Brunvand
5. Fully custom design tutorials of EEE Department, Hong Kong University
6. VLSI I and II Lab Manuals of EEE Department, Bangladesh University of Engineering and Technology (BUET)
7. Custom IC Design Manual provided by University Support Team, Cadence® Design Systems, Bangalore, India

Acknowledgement

Lab Manual prepared and compiled by : Anupam Golder
Dept. of EEE
AUST