

Chapter 5

Virtuoso Layout Editor

VIRTUOSO is the layout editor from Cadence that you use to draw the composite layout for the fabrication of your circuit. It is part of the **dfl** tool suite. The process of drawing layout amounts to drawing lots of colored rectangles with a graphical editor where each color corresponds to a fabrication layer on the integrated circuit. There are a lot of design rules for how those layers interact that must be followed very carefully. The circuit described by these colored rectangles can be extracted from this graphic version and compared to the transistors in the schematic. The layout is eventually exported in standard format called **stream** or **gdsII** and sent to the foundry to be fabricated.

This chapter will show how to use the Virtuoso Layout Editor to draw the composite layout and how to check that the layout obeys design rules using DRC (Design Rule Checking). It's also critical that the layout correspond to the schematic that you wanted. A Layout Versus Schematic (LVS) process will check that the extracted circuit from your layout matches the transistor schematic that you wanted. What we're after is for all the different views of a cell to match in terms of the information about the cell that they share. The **schematic** or **cmos_sch** view should describe the same circuit that the **layout** view describes, the **symbol** view should have the same interface as both of those views, and the **behavioral** view should also share that interface. Eventually we'll add **extracted** and **abstract** views for other tools, which should also match the essential information. You may also use **config** views for analog simulation.

5.1 An Inverter Schematic

We'll demonstrate the layout process with a very simple inverter circuit. First you need a **schematic** view that describes the circuit you want. You

can then simulate this **schematic** view with a Verilog simulator like Verilog-XL or NC_Verilog to verify that it's doing the right thing. This is important because once you design the layout in the **layout** view, you need something to compare it to to make sure the layout is correct. In our case we always refer back to the **schematic** view as the “golden” specification that describes the circuit we want.

Starting Cadence icfb

Because Virtuoso is part of the Cadence dfl tool suite, the first step is starting Cadence icfb with the `cad-ncsu` script. The procedure for starting icfb, making a new library, and attaching the technology are described in Chapter 2. Make sure you have a library with the **UofU_TechLib_ami06** technology attached.

Making an Inverter Schematic

The procedure for designing a schematic using the Composer schematic capture tool is described in Chapter 3. The procedure for using transistors in a schematic is specifically described in Section 3.3. If you use **nmos** and **pmos** transistors from the **NCSU_Analog_Parts** library you will get transistors that simulate with zero delay in the Verilog simulators. If you use transistors from the **UofU_Analog_Parts** library you will get transistors that simulate with 0.1 time units of delay in those same simulators. This is described in Section 4.4.

For this inverter, use **Composer** from the **Library Manager** to create a new cell view with a cell name of `inverter`. Make the view type `cmos_sch` because this will be an individual standard cell consisting of nothing but transistors. Make an inverter using **nmos** and **pmos** transistors from either of the **Analog_Parts** libraries. Use the properties on the transistors to change the width of the transistors so that the **pmos** device is `6u M` wide and the **nmos** device is `3u M` wide. You can do this when you instantiate the devices, or later using the properties dialog box (use the `q` hotkey or the **properties** widget). Your **inverter** schematic should look like Figure 5.1 when you're done.

*It's easy to change all cells (like **nmos**) from one library to be from another library later. You can change the **reference library** from **NCSU_Analog_Parts** to **UofU_Analog_Parts** for example.*

Making an Inverter Symbol

Now is a good time to make a symbol view of the inverter too. Follow the procedure from Chapter 3 to make a symbol view. If you like gates to look like the standard symbols for gates (always a good idea) your symbol might end up looking something like that in Figure 5.2.

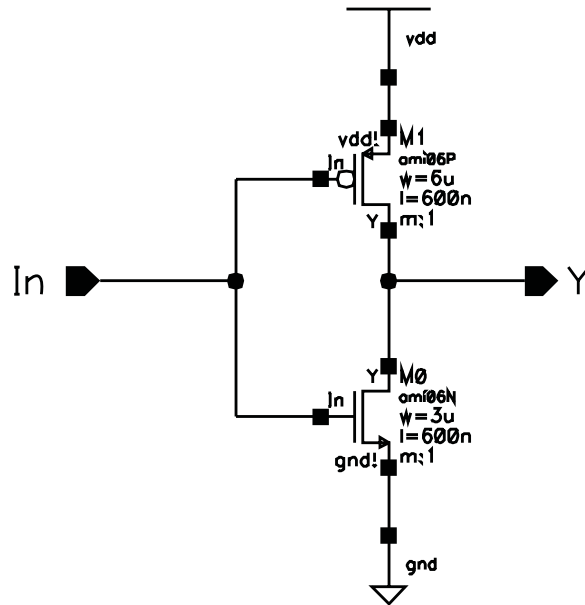


Figure 5.1: Inverter Schematic

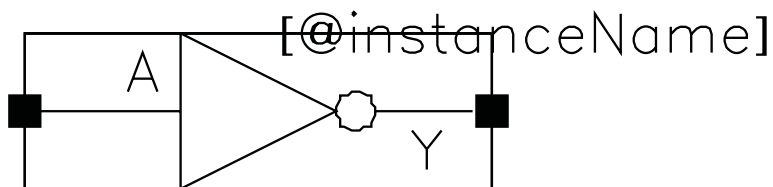


Figure 5.2: Inverter Symbol

Now **check and save** the **symbol** and **cmos_sch** views. They should both check and save without errors before you proceed to the layout!

5.2 Layout for an Inverter

Note that there is a tutorial on Virtuoso that comes with the tools. Although it will not use our same technology information it may show you even more tricks of the tools.

The process of drawing CMOS layout is to use the graphical editor to draw colored rectangles that represent the fabrication layers in the CMOS fabrication process.

Creating a new layout view

In the **Library Manger**, in the same library as you used for the **inverter cmos_sch** view, create a new cell view. This cell view should have the **Cell Name** of `inverter` so that it will be a different view of the same inverter cell. The view type should be `layout`, and the tool should be `Virtuoso` (see Figure 5.3).

This will open up two windows; The **Virtuoso Editing** window where the layout is drawn and the **LSW** (Layer Select Window), where you select the layers (diffusion, metal1, metal2, polysilicon, etc) to draw.

If the layout is going to be a Standard Cell, the height of the cell as well as the width of the VDD and VSS power supply wires must be defined so that cells can eventually abut each other and have the power supply connections made automatically as the cells abut each other. To avoid DRC errors when abutting the cells, it is also important to keep the left and right borders of the cell free of any drawing that might cause an error when the cell is abutted to another cell. The n-tub is usually aligned with the cell borders so that it connects into a seamless rectangle when abutted. Usually cell templates of standard heights containing VDD and VSS contacts (wires) as well as default nwell dimensions are used. For this tutorial, you can just pick some reasonable dimensions, but keep in mind that later you'll want to plan for a particular power pitch.

Drawing an nmos transistor

There are two types of active defined in our technology file: nactive and pactive. They're different colors so you can tell them apart.

Click on the nactive (light green) layer in the LSW window. In the Virtuoso Layout Editor window, press `r` to activate the Rectangle command. Now you can draw a rectangle by selecting the start and end points of the rectangle. This first **nactive** rectangle is shown in Figure 5.4.

Press `k` to activate the **Ruler** command. You can click on one of the corners of the green rectangle to place the ruler and measure the sides (typ-



Figure 5.3: Dialog for Creating a **Layout View** of the **inverter** cell

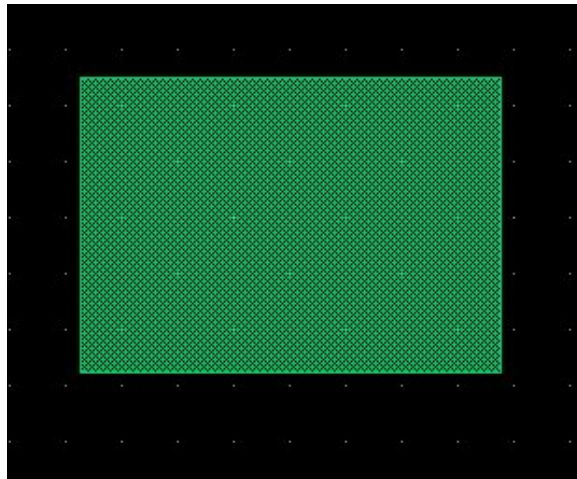
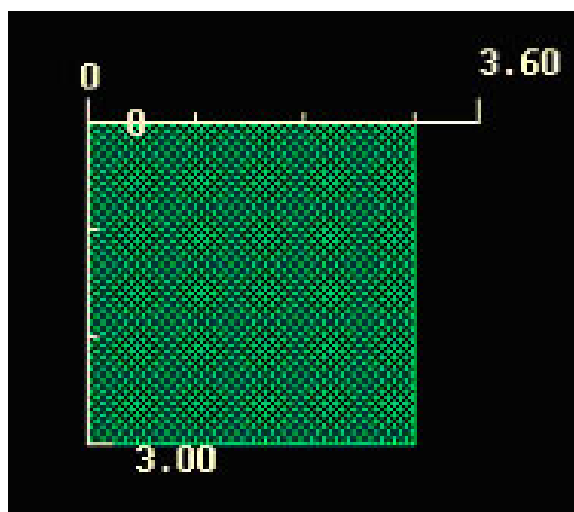


Figure 5.4: Initial **nactive** rectangle

Figure 5.5: **nactive** Rectangle with Measurement Rulers

The terms “active” and “diffusion” are both used to describe the same layers in the layout.

ing **[K]** will clear all rulers). Remember that our N-transistor in the inverter **cmos_sch** view had a W/L of 3um/0.6um. It’s important to keep the the width and length of a transistor in mind when you’re drawing active regions. The length is measured in the direction between the source and drain connections. The width is measured in the orthogonal direction. The width of the diffusion will be usually the same as the width of the transistors but the length of diffusion region has to account for the source/drain contacts and the actual gate length. In order to keep the parasitic capacitance low, the source and drain active regions should be kept as small as possible. This means don’t have extra unnecessary active area between the device and the contact. A general rule of thumb is to have the length of the diffusion region to be 3um + Gate length in the 0.5 micron technology that we’re using.

In Figure 5.5 you can see the rulers measuring a rectangle of **nactive**. A very useful command to know about is the **[S]** **stretch** command. You can use this to stretch the rectangle to be the right size once you have the rulers to guide you.

Now you need to add the source and drain contacts to the **nactive** regions. Contacts are “sandwiches” of active, contact (**cc**), and metal1 (**M1**) layers overlapped on top of each other. Contacts are special in the **MOSIS SCMOS** rules in that they have to be an exact size. Most other layers have a mx or min size, but contacts in the **UofU_Techlib_ami06** technology must be exactly 0.6um X 0.6um in size.

The easiest way to draw a contact is to use the pre-defined contacts in the technology. We have pre-defined in the technology file all the contacts

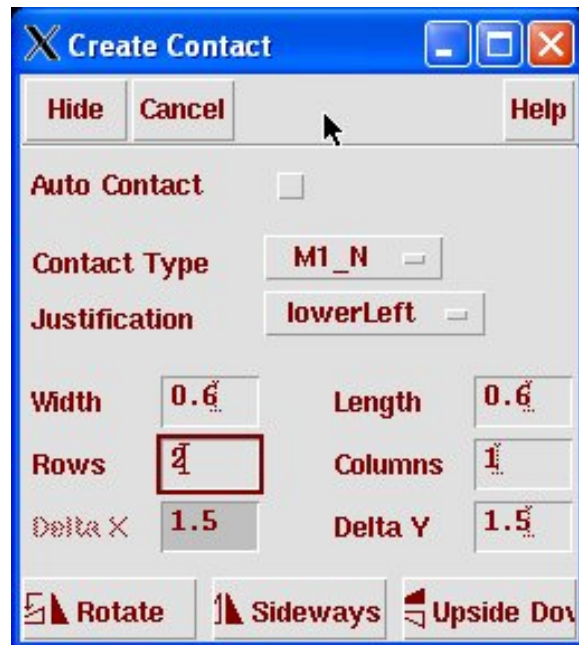




Figure 5.6: Create Contact Dialog Box

that you're likely to need. use the **Create** → **contact...** menu choice or press . This starts up a **create contact** window. In this window you can select any from the list of predefined contacts. You can also make an array of contacts if you have an area to contact that is larger than the area covered by just a single contact. In general you should use an array of as many contacts as will fit in the area that you're trying to contact. Contacts have non-negligible resistance compared to the pure metal connections so adding more contacts reduces the resistance of the connection by offering parallel resistors for the entire connection. The **create contact** dialog box looks like that in Figure 5.6. Note that we're making arrays of contacts with one column and two rows. You can change this later if desired by selecting the contact array and using the  properties.

Pre-defined contacts are defined in the technology file. The pre-defined contacts that are available in this technology file include:

M1.P: Connection between Metal 1 and Pactive (source and drain of **pmos** devices) that includes the pselect layer

M1.N: Connection between Metal 1 and Nactive (source and drain of **nmos** devices) that includes the nselect layer

NTAP: Nwell connection used for tying the Nwell to Vdd

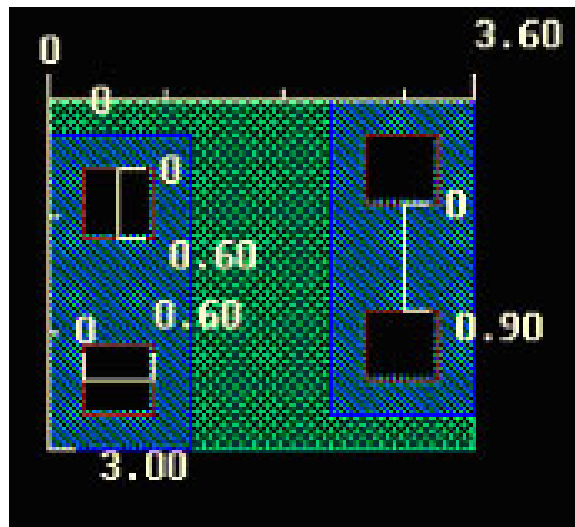


Figure 5.7: **nactive** showing source and drain connections

SUBTAP: Substrate connection used for tying the substrate to GND

P_CC: Connection between Metal 1 and Pactive without select layers (can be used if it's already inside a pselect layer)

N_CC: Connection between Metal 1 and Nactive without select layers (can be used if it's already inside an nselect layer)

M1_POLY: Connection between Metal 1 and Polysilicon

M1_ELEC: Connection between Metal 1 and the Electrode layer (Poly2).

M2_M1: Connection between Metal 1 and Metal 2

M3_M2: Connection between Metal 2 and Metal 3

Our **nmos** transistor layout with arrays of two **M1_CC** contacts on source and drain and rulers showing all the dimensions is shown in Figure 5.7. Of course, you can draw the contacts using rectangles of **nactive**, **cc**, and **M1** layers directly, but it's much easier to use the pre-defined contacts.

Now you can draw the gate of the transistor by putting a rectangle of polysilicon (**POLY**) over the active. It's important to note that the poly must overlap the active in order to make a transistor. When the transistor is fabricated there will be no diffusion implant in the channel region underneath the poly, but for the layout you must overlap active and poly to make transistors. The process that extracts the transistor information from the layout uses this overlap to identify the transistors.

It turns out that it's not enough to draw **nactive** to let the foundry know that you want n-type silicon for this device. You also need to put an **nselect** "selection" layer around the **nactive** region to signal that this active region should be made n-type. This is because of how the layers are represented in the fabrication data format. In that format there's only one layer for all active regions, and the N-type and P-type active is differentiated with the select layers. We use different layers for **nactive** and **pactive** and make them different colors so it's easy to tell them apart, but they both become generic "active" when they are output in **stream** format so the select layers are required.

You could have used the **M1_N** contacts but because we drew a separate **nselect** rectangle around the whole transistor, it's not necessary to include the redundant **nselect** around the contacts. It wouldn't hurt anything, though. As long as the overlapped **nselect** rectangles cover the area that you want, it's just a matter of aesthetics whether you have one big rectangle or lots of overlapping smaller rectangles. Personally, I like the cleaner look of one large rectangle. It helps me see what's going on without the visual clutter of lots of overlapping edges.

The final **nmos** transistor layout with the (red) polysilicon gate, **nselect** region (green outline) is shown in Figure 5.8. Note that the *width* of the red **POLY** layer defines the transistor's **length** (0.6u M). The *width* of the transistor is the height of the **nactive** layer. The transistor gate must overlap the width of the transistor by 0.6 microns. Note also that the **nselect** layer must overlap the **nactive** by 0.6 microns. The source and drain regions of this transistor have connections to the M1 (first level of metal interconnect) through the **M1_CC** contacts. You can connect to these regions using a rectangle of M1 later.

Drawing a pmos transistor

Now complete the same set of steps to make a **pmos** transistor somewhere in your layout window. This is a very similar process to making the **nmos** device. The differences are:

- Use **pactive** instead of **nactive** for the active (diffusion) layer.
- Use **M1_CC** (or **M1_P**) contacts for the source and drain regions, or use a "sandwich" of **pactive**, **cc**, and **M1** layers with the correct dimensions. Remember that the **cc** contact must be exactly 0.6u M by 0.6 u M in this technology. The **pactive** and **M1** layers must overlap the **cc** layers by 0.3u M. As before, it's easier to use the pre-defined contacts using the **Create** → **contact...** menu choice or press .

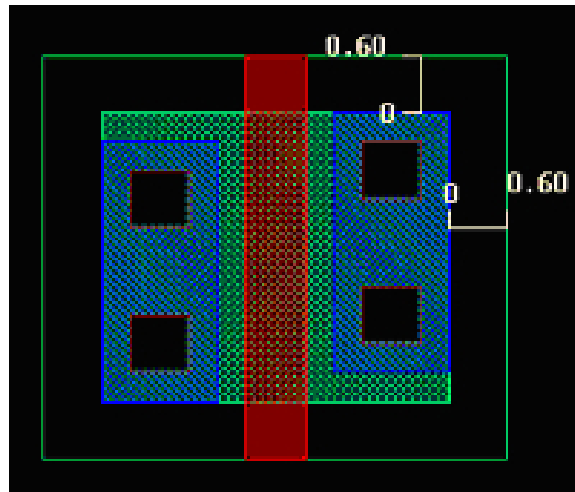


Figure 5.8: NMOS transistor layout

- Because of the mobility difference between **nactive** and **pactive**, make the **pmos** device twice as wide as the **nmos** device to roughly equalize drive capabilities. In other words, make the **pmos** transistor 6u M wide, and keep the length the default 0.6u M long.
- Surround the **pactive** rectangle of the **pmos** device with **pselect** so that the foundry knows to make this active region P-type.

The result of making this **pmos** transistor is shown in Figure 5.9. Note that the transistor is twice as wide as the **nmos** device, and has arrays of **M1P** contacts that are four rows high.

However, because of the processing technology used for this set of technology files, we're not done yet. The **pmos** device must live inside of an **NWELL**. This allows the **pactive** of the **pmos** device to be surrounded by N-type silicon, while the **nmos** device we drew earlier lives in the P-type silicon substrate. Draw a box of **NWELL** around the entire **pmos** device with overlaps around the **pactive** of at least 2.1u M. The completed **pmos** device with its surrounding **NWELL** is shown in Figure 5.10.

Assembling the Inverter from the Transistor Layouts

Now that you have layout pieces of **nmos** and **pmos** devices, you can assemble them into an inverter by connecting the source and drain connections of the transistors to the appropriate places. Connections are made using rectangles of any conducting layers, but metal layers are the usual choice for making connections if that's possible.

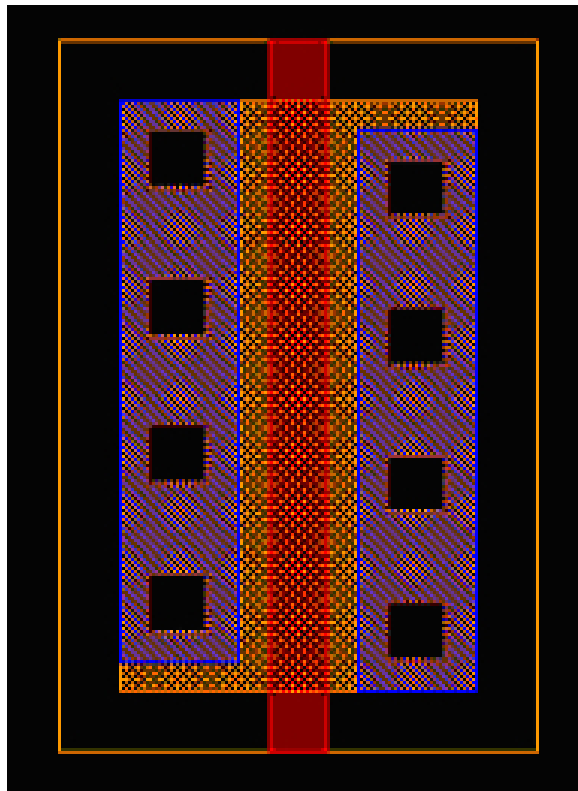


Figure 5.9: A **pmos** transistor 6 μ M wide and 0.6 μ M long

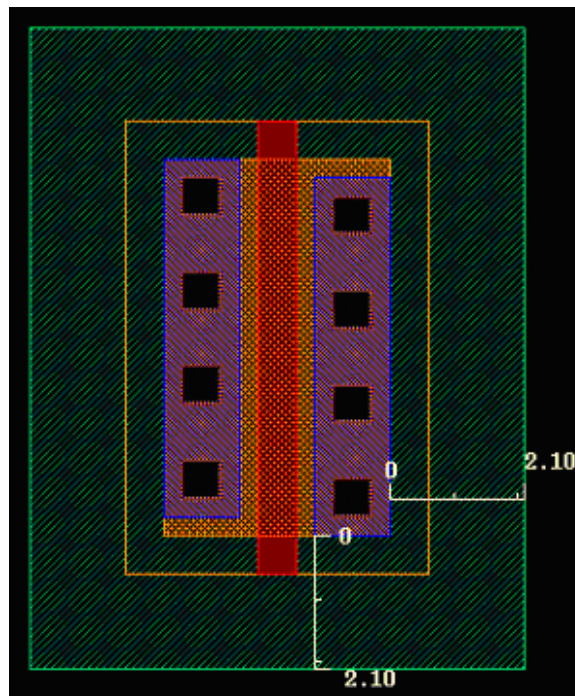
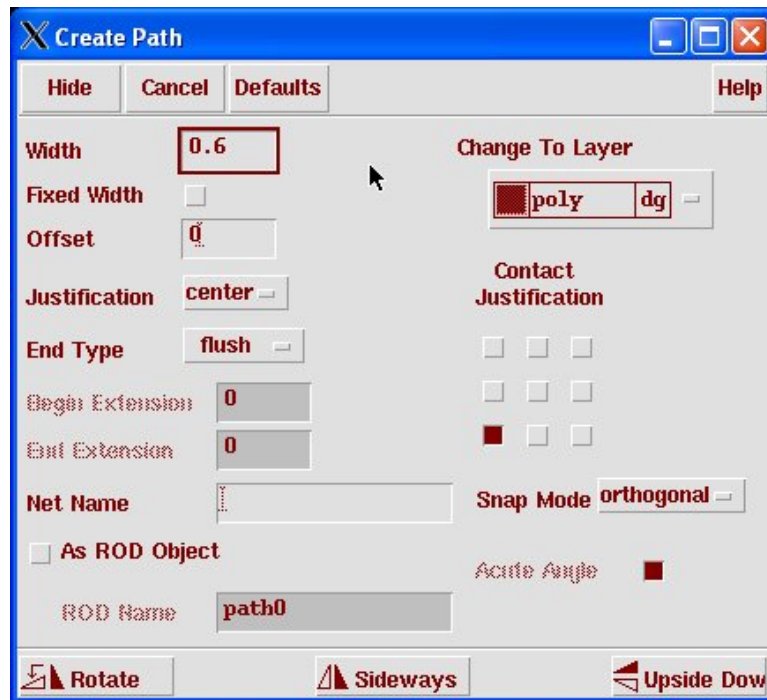


Figure 5.10: A **pmos** transistor inside of an **NWELL** region

Select the entire layout piece of one of your transistors using the left mouse button. Starting with the cursor in the arrow shape click and drag with the left button pressed. The selected rectangles are highlighted in white. Now hover your mouse over the selected pieces until the cursor changes to the “move” character (arrows pointing up, down, right, and left), and move the whole piece of layout using the left button again. This time everything that has been selected will move as you drag. Drag the layout so that the transistors are arranged vertically with a gap of at least 1.8u M between the **nwell** of the **pmos** and the **nselect** of the **nmos** transistors.

Once the transistors are placed, you can connect the drains of the two transistors to make the output node of the inverter using a rectangle of **metall1**. You can also connect the gates of the transistors together with a rectangle of **poly**, and make the input connection from the **poly** to **metall1** using a **M1_POLY** contact.

You can also use “paths” for making these connections. A “path” is like a “wire” in Virtuoso. To make a path, select the layer you want, and then select **Create** → **Path** or use the **[p]** hotkey. This enters “path creating mode” in the editor. You start paths with a left click, make bends in the path with another left click, and finish a path with a double left click. You can

Figure 5.11: Dialog box for the **path** command

also get a dialog box that controls how the path is created using the F3 function key. This dialog is shown in Figure 5.11 for a **poly** wire.

A few things to note about this **path** dialog are:

- You want to set the snap mode to **orthogonal** to keep your wires on vertical and horizontal axes
- You can change the **justification** of the wire from center to left or right depending on whether you want your mouse to define the center, or one of the edges of the path
- The width of the path defaults to the minimum allowable width for that layer in the technology. You can change this if you want a fatter wire, but don't make it narrower than the default.
- You can use the dialog box to change layers by selecting a different layer. The **path** mode will have you place an appropriate contact, and then switch to a path of that new layer.

The layout for the inverter with the input and output connections made between the two transistors is shown in Figure 5.12.

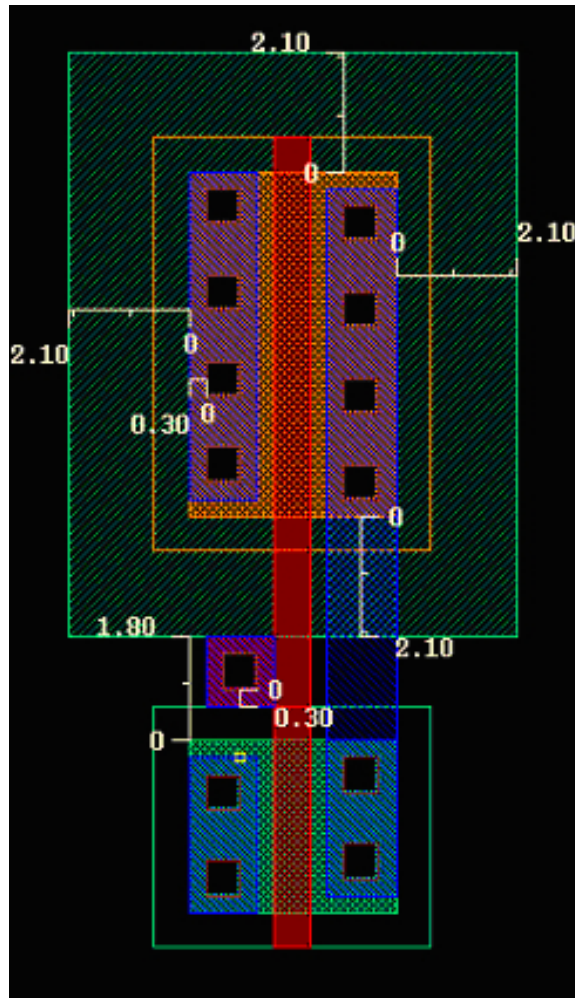


Figure 5.12: Inverter Layout with Input and Output Connections Made

Now make the source connections to the **Vdd** and **GND** power supplies at the top and bottom of the inverter. These are simply wide metal connections in **metal1**. Don't forget to make the connections from the wide wires to the source nodes of the transistors. Make the power supply wires at least 2.4u M wide to avoid electromigration problems in the future. The inverter with the power supply wires is shown in Figure 5.13.

Although this is a “functional” inverter layout at this point it's not complete. In CMOS you must make sure that the substrate is tied low with a connection to **GND** and that the **nwell** is tied high with a connection to **Vdd** to avoid latchup effects due to the parasitic devices formed by the wells. These contacts are most easily made using the **SUBTAP** and **NTAP** pre-defined contacts in the technology. These taps are contacts from the **nactive** and **pactive** active regions to **metal1**, but they are meant specifically for connections to the well and substrate. Any connection to **pactive** which is outside of an **nwell** is a substrate connection, and any connection to **nactive** inside of an **nwell** is a well connection. These contacts are a standard width so that they can be tiled in a consistent way. When you start fitting your cells to the library template you will see that cells in the **UofU_Digital** library are sized in units of 2.4u M chunks that match with the well and substrate tap widths for convenience.

The inverter with the well and substrate connections is shown in Figure 5.14. The **nwell** layer in this example has been extended by 0.3u M beyond the **Vdd** metal layer to make it easier to abut these cells in a larger layout later on.

Finally, we need to add connection name information to our layout. Remember that all the different views of a cell must be consistent in their I/O interface (at least). The inverter **cmos_sch** view and **symbol** view each have an input connection named **In** and an output connection named **Out**. In addition, our layout needs to identify the power supply connections explicitly. We'll specify the power supply connections with the names **vdd!** and **gnd!**. The “!” character is a flag to make these names global. If you don't use this flag you will have to export your power supply connections as pins to your cell. That's not incorrect, it's just a hassle.

Remember that Verilog names are case sensitive

In order to make connections in the layout view we need to add **pins** to the design. **Pins** are similar to the red-arrow pins in the **cmos_sch** view, but in the layout we identify pieces of the layout with the pins. These are called **shape pins** because they identify a shape in the layout with the pin. There are other types of pins you can make in **Virtuoso** but please use only **shape pins** in this tool flow because other steps in the process depend on this.

A **shape pin** is a piece of layout (a rectangle) in a particular layer (i.e. **metal1**) that also has a pin name attached to it. Add a **shape pin** using the **Create** → **Pin** menu choice. The dialog box is shown in Figure 5.15. Note

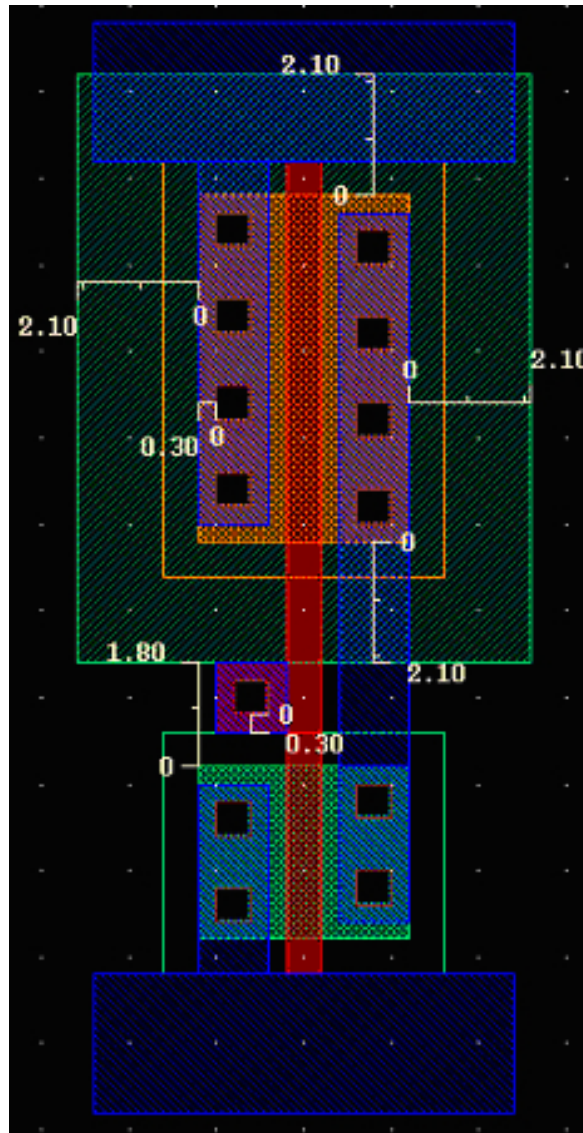


Figure 5.13: Inverter Layout with Power Supply Connections

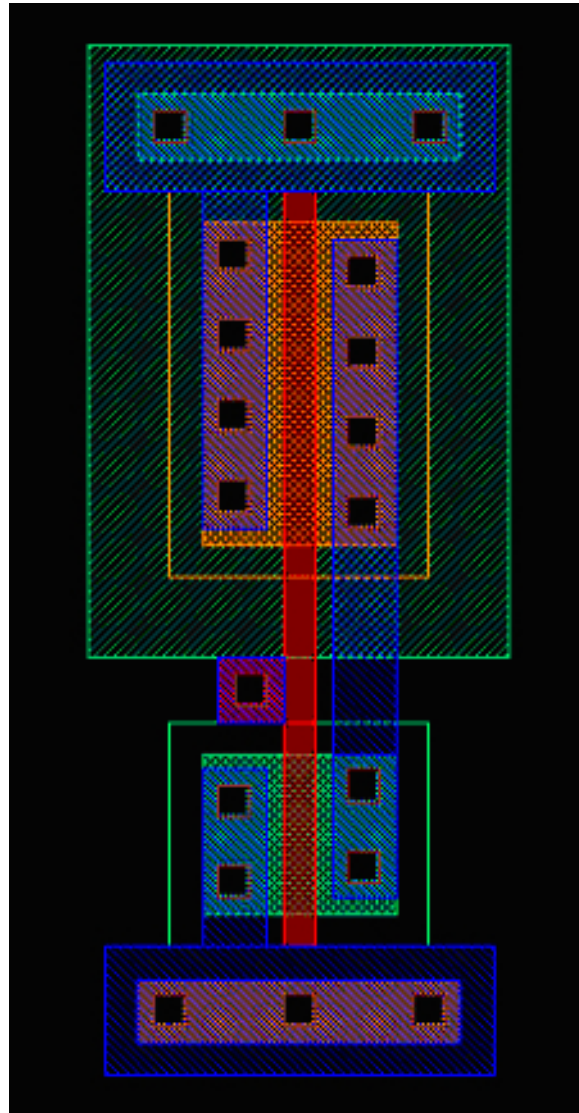


Figure 5.14: Inverter Layout with Well and Substrate Connections

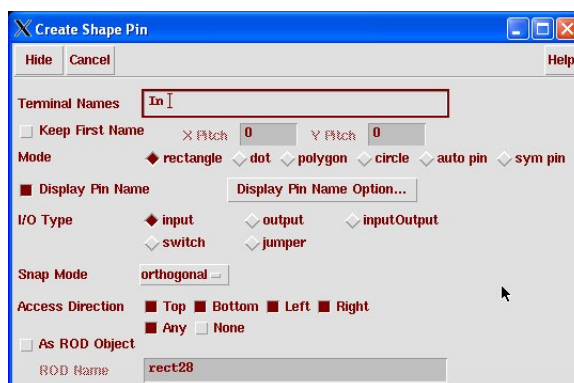


Figure 5.15: Shape Pin Dialog Box

that you won't see this exact dialog box until you select **shape pin** as the pin type. Once you are in the **shape pin** dialog box, make sure that you have selected the **Display Pin Name** option, that you're placing a rectangle, that you have selected the correct direction for the pin (i.e. **input**), and that you have entered the name (or names) you want to add. You select the layer that you want the pin to be made in (i.e. **metal1**) in the **LSW** (Layer Selection Window). When all this is correct, draw a rectangle of the layer in the spot that you want to have identified with the connection pin in the layout.

In our inverter place a **metal1 input shape pin** named **In** directly over the **M1.POLY** contact for the inverter's input, and a **metal1 output shape pin** named **Out** somewhere overlapping the output node connection. For the power supply connections, make the shape pins as large as the power supply wire, and in **metal1**. This will define that entire wire as a possible connection point for the power supply. Power supply connections are made with the names **vdd!** and **gnd!**, and should be **input/output** type. The final layout for the inverter showing the pins with the pin names visible is shown in Figure 5.16. Note that it's important for a later stage in this process that the little "+" near the pin name that defines where the label is to be inside the shape of the **shape pin**.

Before proceeding to the next step, make sure to save your layout. In fact, you should probably save the layout often while you're working. There's nothing quite as frustrating as doing hours of layout only to have the machine or the network crash and you lose your work! Save often!

Using Hierarchy in Layout

Once you have a piece of layout, you can use that layout as an instance in other layout. In fact, that's what you've already done with the pre-defined

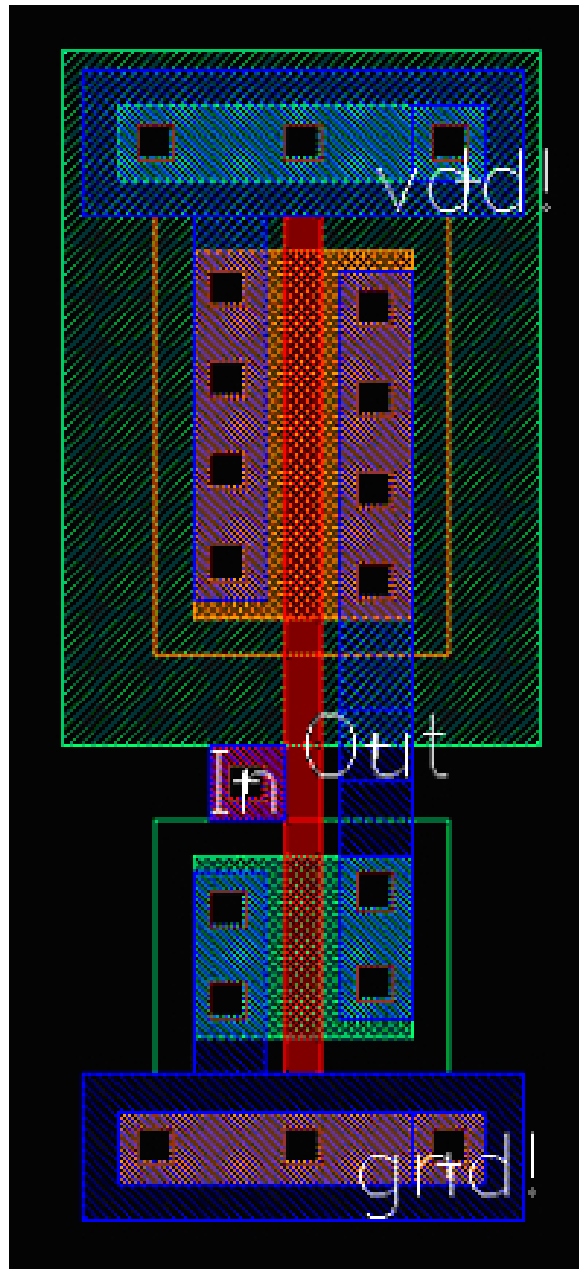


Figure 5.16: Final Inverter Layout

contacts in the **Add** → **Contact** step. The pre-defined contacts are actually pieces of layout in another layout view inside the **UofU_TechLib_ami06** technology library. When you put one in your layout you were really placing an instance of that cell in your layout.

You can see this by changing the scope of what's visible in the layout editor. That is, you can change things so that the only shapes you see are the shapes that are directly placed in your current cell view, and the hierarchical instances will be hidden with a red box around them. Try toggling the hierarchical display using `shift-f` and `ctrl-f` to see how this works.

As a very simple example of including other hierarchical cells in a layout, consider a simple circuit with four inverters in a row. Figure 5.17 shows a layout with four instances of an inverter where the output of each inverter is connected to the input of the inverter on its right. Figure 5.18 shows the same cell with the hierarchy expanded to look inside each of the inverter instances.

5.2.1 Virtuoso Command Overview

Virtuoso, like any full-featured layout editor, has hundreds of commands. This tutorial has only scratched the surface, but these commands should be all you need for most basic layout tasks. These tasks and commands are:

Drawing Rectangles: The basic components of a **layout** view are colored rectangles that represent fabrication layers. Layers are selected in the **LSW** Layer Selection Window. Rectangles are drawn with the **Create** → **Rectangle** menu choice, the rectangle widget, or the `[r]` hotkey.

Connecting Rectangles: Rectangles of the same material are electrically connected by abutment or by overlapping. Rectangles of different materials are electrically connected using contacts or vias. These can be drawn with a “sandwich” of layers with an appropriate contact shape between them (**cc**) for connecting **metal1** “down” to active or poly, and **via<n>** for connecting **metal<n>** to **metal<n+1>**. So, for connecting **metal2** to **metal3** a **via2** is used. Vias and contacts must be exactly 0.6u M by 0.6u M in this **UofU_Techlib_ami06** technology.

Using Paths: A convenient way to draw “wires” out of rectangles is to use the **path** feature through the **Create** → **Path...** menu choice or the `[p]` hotkey. Choose the layer from the **LSW**. Single-click the left button to change directions, double click the left button to end a wire.

Moving and Stretching: Select layout using the left button, or click and drag to select groups of layout objects. If the cursor changes to the

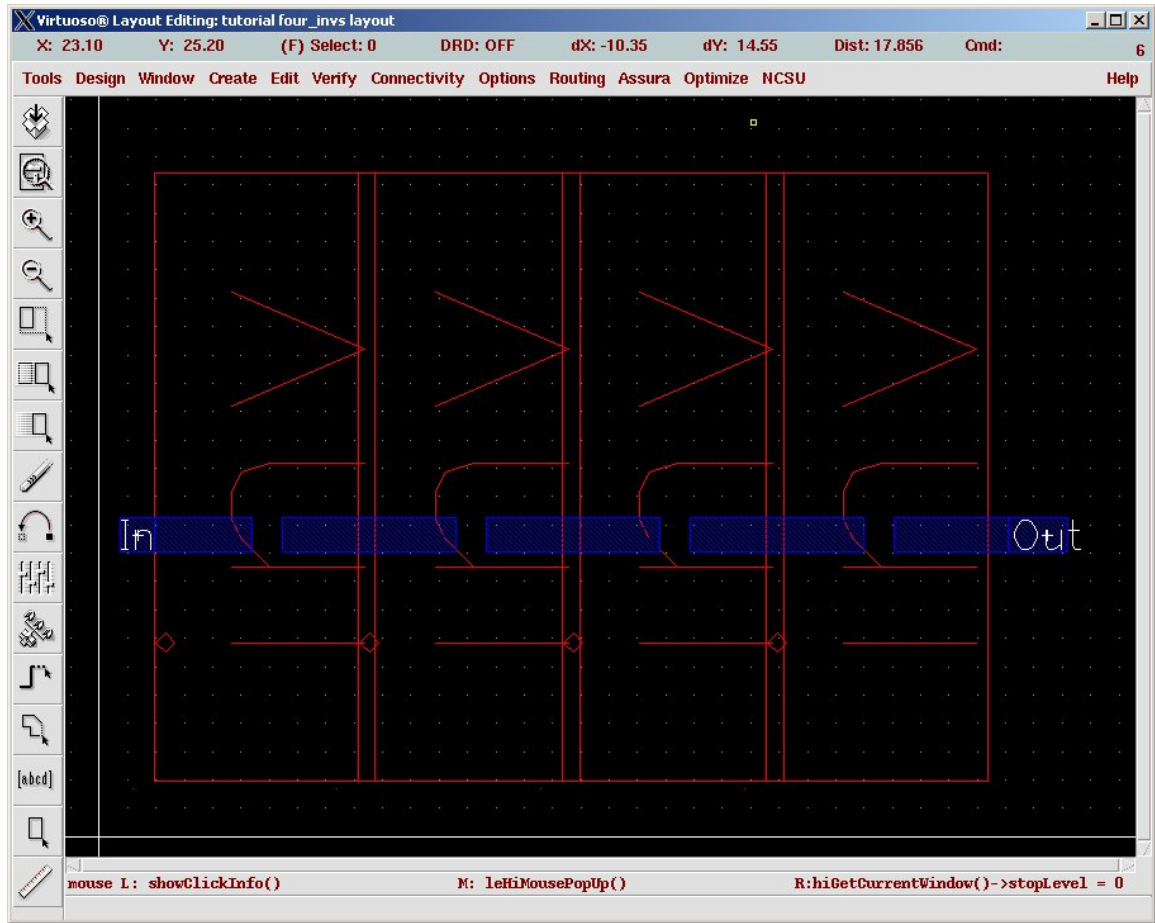


Figure 5.17: Layout with four inverter instances

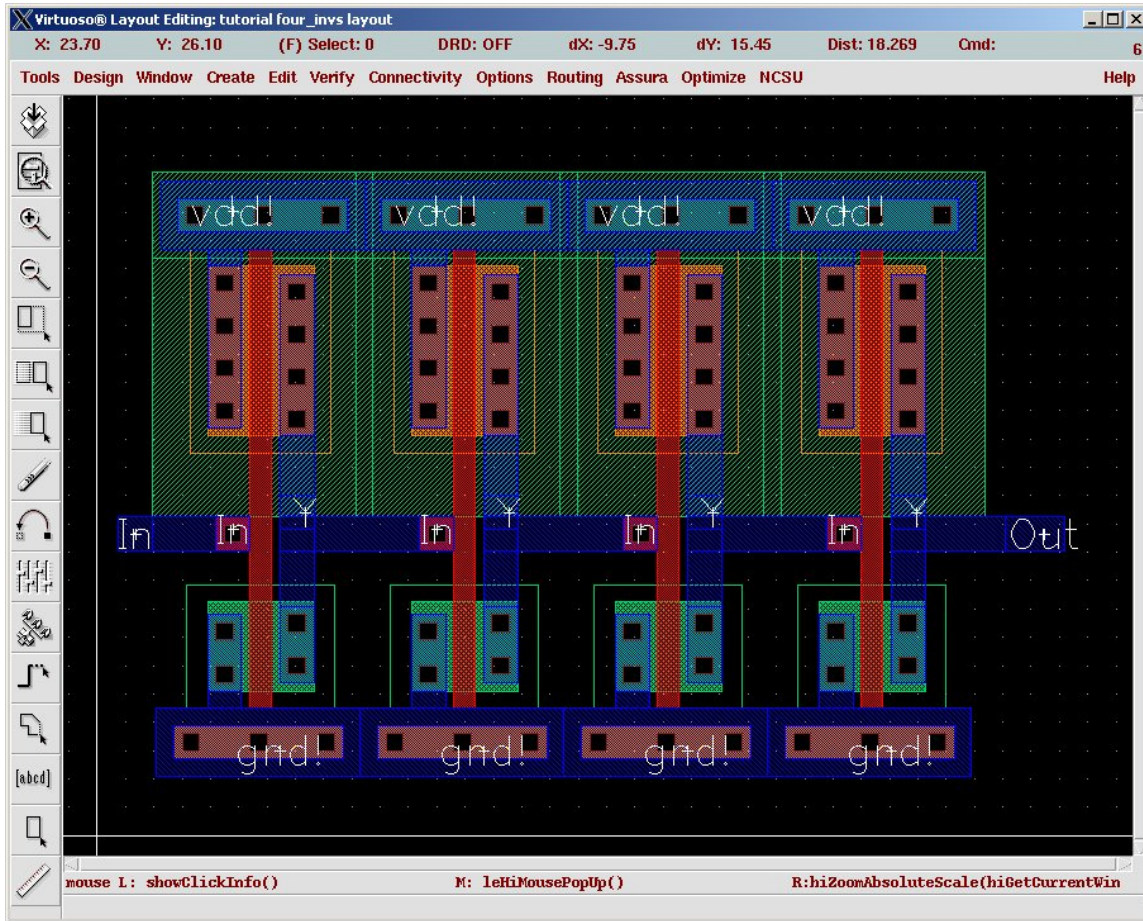


Figure 5.18: Layout with four inverter instances expanded to see all levels of layout

“movement arrows” you can move the selected shapes using the left button with click and drag. Reshaping can be done with the **Stretch** (hotkey) command.

Changing the View Hierarchy: You can choose how many levels of hierarchy to view using the **Options** → **Display** meny choice and change the **Display Levels** numbers. You can also use the hotkey to hide all but the top level, and (shift-f) hotkey to expand the display to all levels of the hierarchy. Note that the hotkey fills and centers on the screen with the current cell.

Viewing Connectivity: You can use the **Connectivity** → **Mark Net** meny to mark an entire connected net. This will highlight the entire connected net through all conducting layers. Note that it does not *select* that net, it simply marks it so you can see what’s connected to what.

Hierarchical Instantiation: You can include instances of other layout cells in your current layout with the **Create** → **Instance** menu, the hotkey, or the **instance** widget. These instances will show up as blank boxes with a red outline if you are not viewing deep enough in the hierarchy, and as filled in layouts if you have expanded the viewing hierarchy.

5.3 Printing Layouts

In order to print your layout view you can use the **Design** → **Plot** → **Submit...** menu. This will submit your design to be plotted (printed) either on a printer or to a file. The **Submit Plot** dialog box is shown in Figure 5.19. This Figure shows the dialog box after the plotter has been configured to send the output to an EPS (Encapsulated PostScript) file on A-sized paper to a file called **foo.ps**. These choices are set up in the **Plot Options...** dialog box as seen in Figure 5.20. In this dialog you can select the plotter (printer) that you would like to send the graphics to. You can either select a printer name (like the CADE printer), or save the file as EPS. You can choose to center the plot, fit it to the page, or scale the plot to whatever size you like. You can also choose to queue the plot for a later time, change the name of the file (if you’re plotting to a file), and choose to have email sent when the plotting completes (a somewhat silly feature, unless you’re really queuing the plot for sometime later). The EPS option will print to a color postscript file so that if you have access to a color printer you can print in color. I like to un-select the **Plot With Header** option so that I don’t get an extra header page included with the plot.

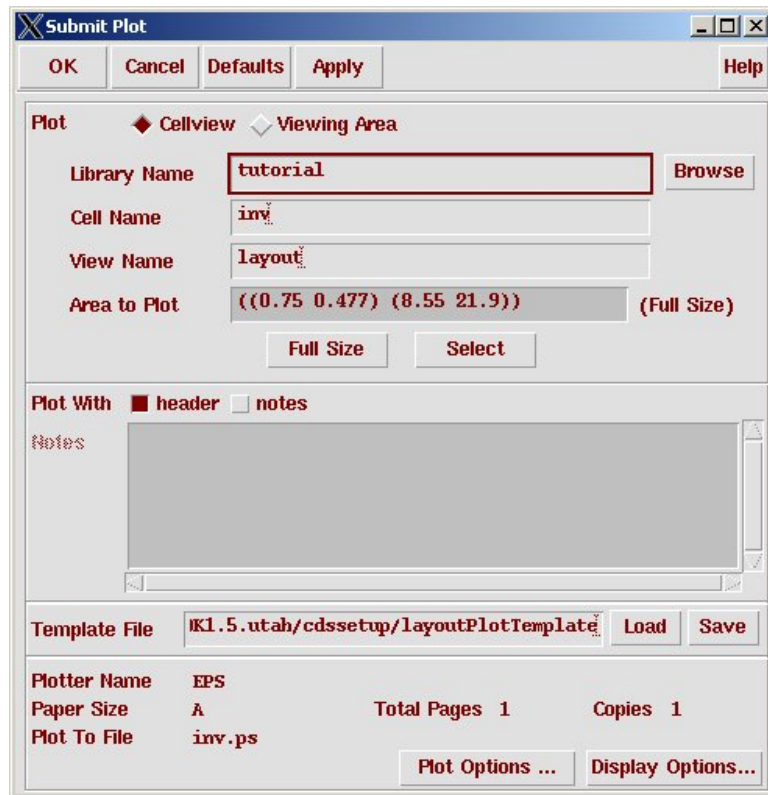


Figure 5.19: **Submit Plot** dialog box

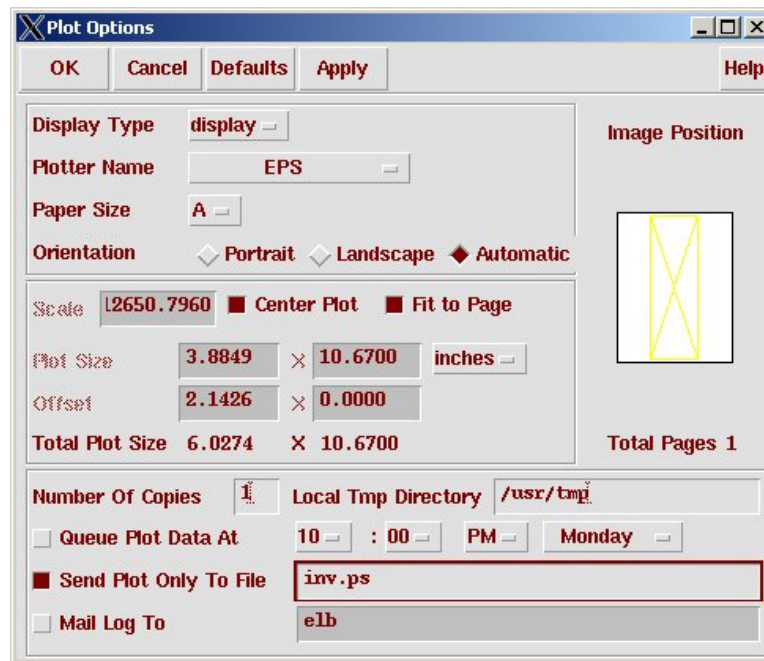


Figure 5.20: Plot Options dialog box

5.4 Design Rule Checking

Because there are so many rules about how you can draw shapes in the different mask layers, it's very hard to tell just by looking if you have obeyed all the design rules (The MOSIS SCMOS rev8 design rules are shown in Appendix D). Letting the CAD tool check the design rules for you is known as *Design Rule Checking* or DRC. When you're drawing new layout using Virtuoso it's helpful to run the DRC check frequently as you're doing your layout. There are three DRC tools that we have access to in this CAD flow, but only DIVA is currently fully supported for our class technology. The tools are:

DIVA: This is the DRC engine that is integrated with Virtuoso and supported by our current class technology files. It will be described in detail in the next section.

Assura: This is a faster and more capable DRC engine that is also integrated with Virtuoso, but not yet fully supported by our technology information. It uses a different DRC rules file that is currently under development.

Calibre: This is the "industry standard" DRC engine from Mentor. It re-

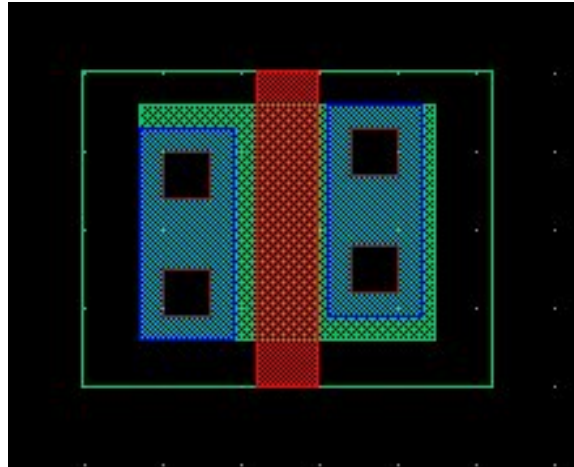


Figure 5.21: NMOS transistor layout (with DRC errors)

quires yet a different DRC rules file so it's not supported by our class flow at the time being.

5.4.1 DIVA Design Rule Checking

If you have a VIRTUOSO layout window open you can check that layout for design rule violations using DIVA from that layout window. Consider a piece of layout as shown in Figure 5.21. This is a single nmos transistor similar, but not exactly the same as, Figure 5.8.

To run DIVA DRC choose the **Verify** → **DRC...** menu choice. A new window pops up as shown in Figure 5.22. Notice that the **Rules file** field is already filled in with **divaDRC.rul**. This is the rules file in the **UofU.TechLib.ami06** library that checks for MOSIS SCMOS Rev8 design rules. Click **OK** to start the design rule check. The results of the DRC will be in the main **CIW** window, and if there are errors those errors will be highlighted in the layout with flashing white shapes.

In this case there are 4 total errors flagged by the DRC process. These rules are described in the **CIW** where the **SCMOS Rule** number tells you which of the rules from Appendix D has been violated. The violation geometry is also highlighted in the layout as shown in Figure 5.24. From these two sources we can see that the top of the **poly** gate needs to extend further over the **nactive**, the **nselect** also needs to be extended at the top of the transistor, and the **cc** contact layer is too close to the transistor gate.

In a larger layout it may be hard to spot all the error highlights. In this case you can use tools in VIRTUOSO to find the errors for you and change

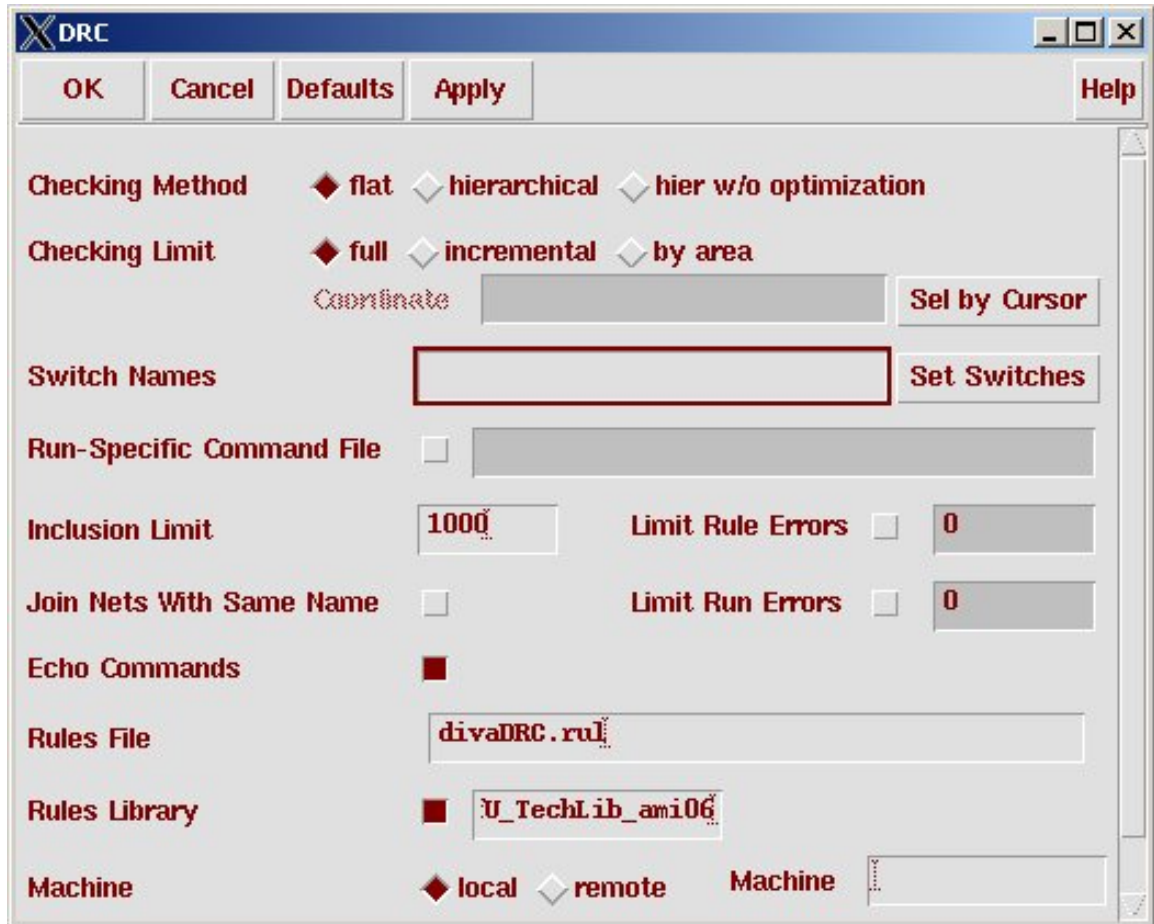


Figure 5.22: DIVA DRC control window

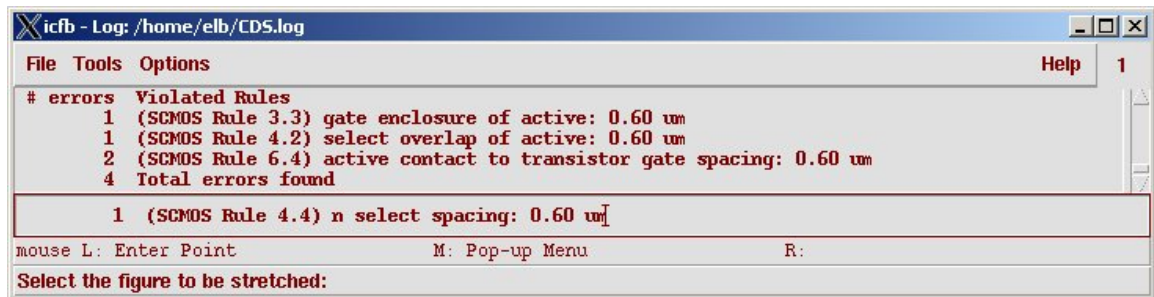


Figure 5.23: Results from the DRC in the CIW window

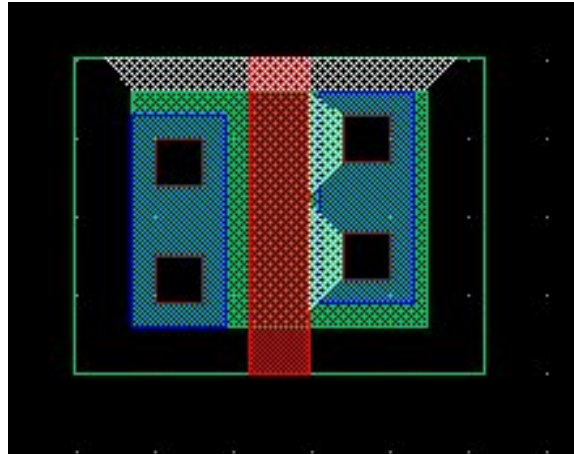


Figure 5.24: NMOS transistor layout (with DRC errors flagged)

the layout view so that you can see those errors. If you select **Verify** → **Markers** → **Explain** you can click on an individual error marker and have it explained. If you do this to the white marker at the top of the example in Figure 5.24 you get the explanation as shown in Figure 5.25. If you would like to step through all the errors in the current layout view, you can select **Verify** → **Markers** → **Find...** which pops up the window seen in Figure 5.26. Selecting the **Zoom To Markers** option and then clicking the **Next** button will zoom to each DRC violation in turn and put the explanation in a window like that in Figure 5.25. It's a great way to walk through and fix DRC violations one by one in a larger layout. When you are finished and would like to get rid of all the white DRC markers you can use **Verify** → **Markers** → **Delete** or **Verify** → **Markers** → **Delete All...** to erase them from the screen.

When you have fixed all the violations and re-run the DIVA DRC process you should see **Total errors found: 0** in the **CIW**. Although the previous example of the inverter was completed using the rulers, in practice I would have used the DRC checker after each of the major steps to make sure that things were legal as I was drawing the layout.

5.4.2 Assura Design Rule Checking

This is to be added. The Assura tool can read a DIVA DRC file, but it needs some modification. This has been done, but not documented yet (10/2006).

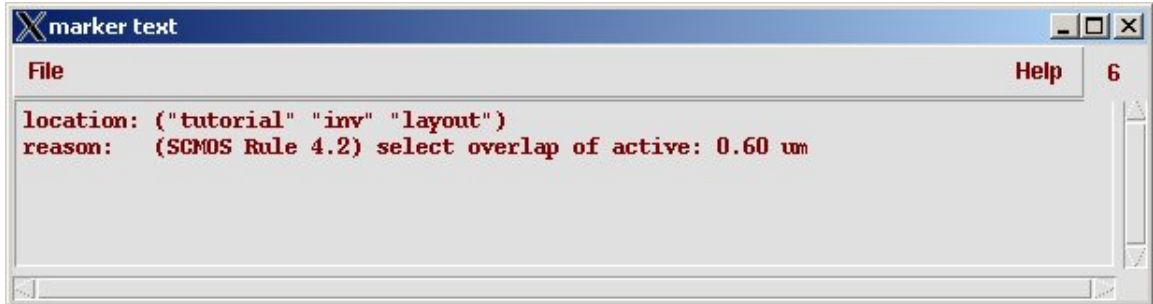


Figure 5.25: Explanation of DRC violation

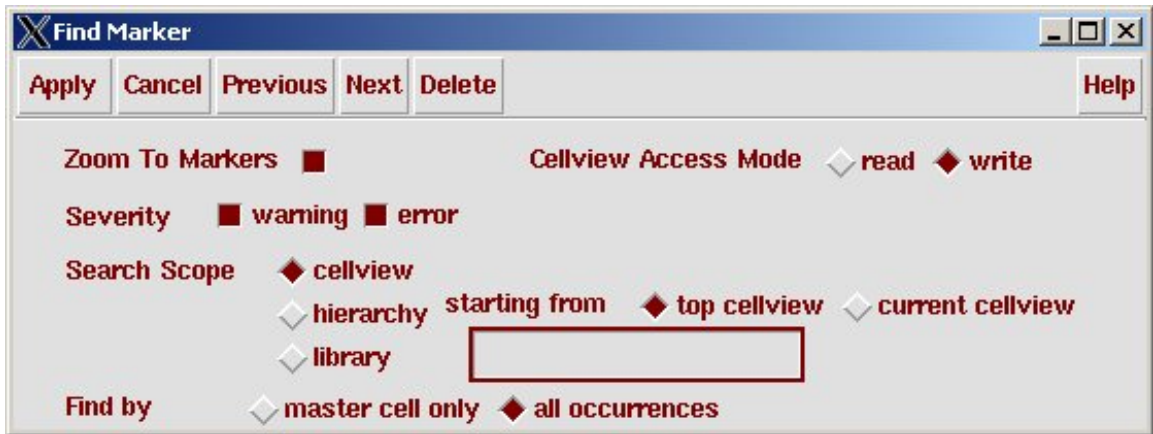


Figure 5.26: Finding all DRC violations

5.5 Generating an Extracted View

Another view that is essential to the design process is an *extracted* view of a cell. An extracted view is one that takes the layout view of the cell and extracts the transistor schematic from that layout. The extraction tool does this by knowing the electrical properties of the layers and the overlap of the layers. For example, one of the extraction rules is that when a **polysilicon** rectangle overlaps a rectangle of **nactive**, the area of the overlap is an nmos transistor. The connection between the transistors is extracted from the conducting layers and the connection between those layers.

To run the extraction process using DIVA start by opening the **layout** view of a cell (for example, the inverter from Figure 5.16). From the layout window choose the **Verify** → **Extract...** menu choice. A new window pops up as shown in Figure 5.27. Notice that the **Rules file** field is already filled in with **divaEXT.rul**. This is the rules file in the **UofU_TechLib_ami06** library that extracts the transistors using the technology defined according to the MOSIS SCMOS Rev8 design rules. Another thing to consider about extraction is whether you want to extract with or without parasitic capacitances. Usually you want to include parasitics because it will give a more accurate (although slightly slower) analog simulation of the cell later on. To include the parasitic capacitors use the **Set Switches** button in the extraction dialog box. This will bring up another box (Figure 5.28) that has a number of switches. The most important is **Extract Parasitic Caps**. If you select this and click OK you will see this switch show up in the main extraction dialog box.

Once things are set the way you want them, click **OK** to start the extraction. The results will show up in the icfb Command Interpreter Window (CIW). Note that you should run the extractor only after passing the DRC phase. A correct extraction should show 0 errors as shown in Figure 5.29.

*All layers in the virtuoso layout editor are actually “layer/purpose pairs.” The “purpose” of the layers is **drawing**. The layers in the extracted view are “net” purpose and have outline views to indicate this different purpose.*

After extraction you will see that for your cell (the **inv** in this example) will now have a new view named **extracted**. You can open this view in **icfb** and look at the result. For the inverter you should see outline views of the mask layers and transistor symbols for each of the extracted transistors. The extracted transistors will be annotated with their width and length as defined by the rectangles that are drawn. See Figure 5.30 for an example. The extracted view is used to compare the layout with a schematic to make sure that the layout accurately captures the desired schematic. This is a *critical* step in the design process!

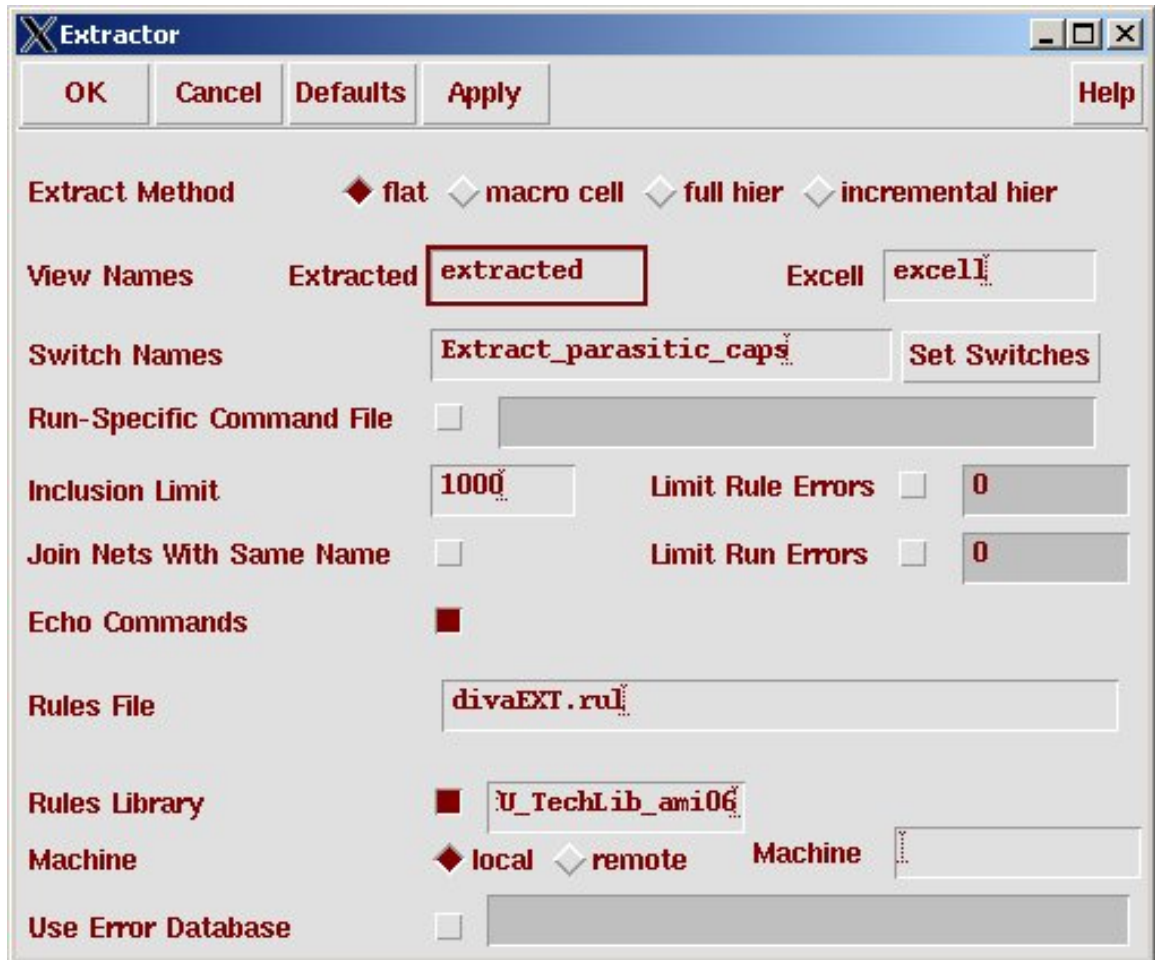


Figure 5.27: DIVA extraction control window

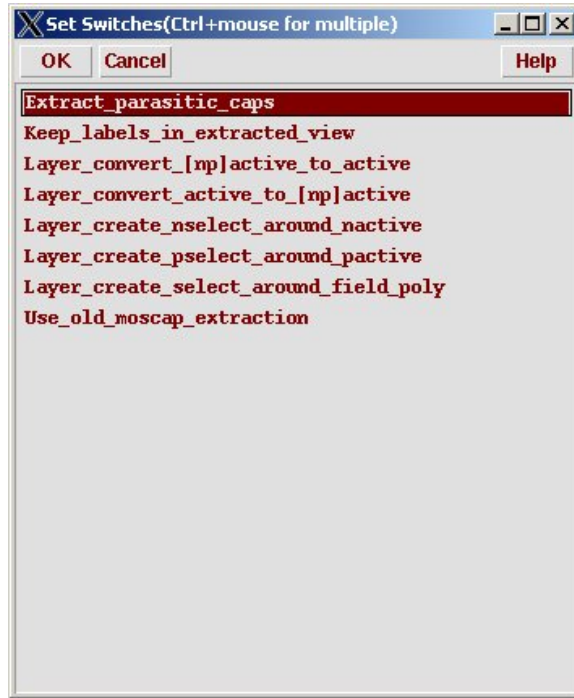


Figure 5.28: DIVA extraction special switches

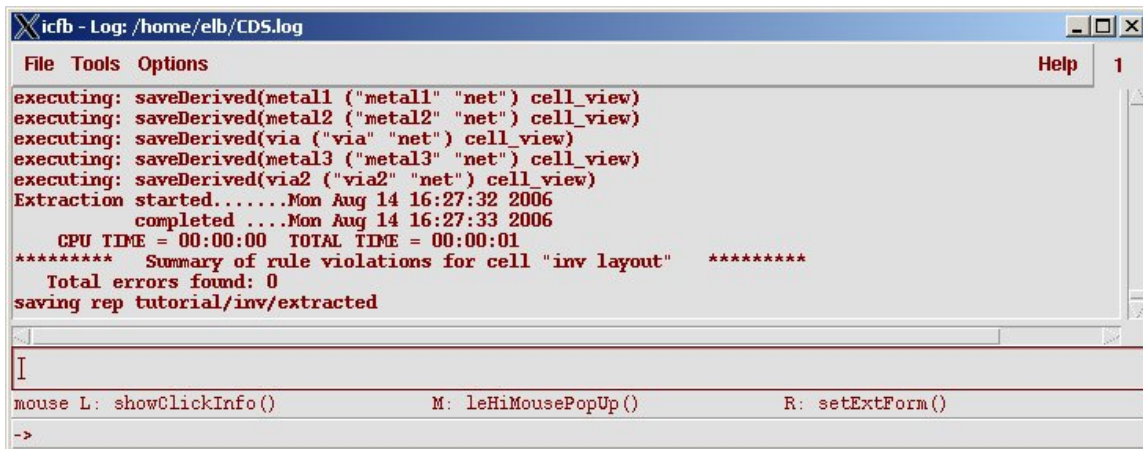


Figure 5.29: DIVA extraction result in the CIW

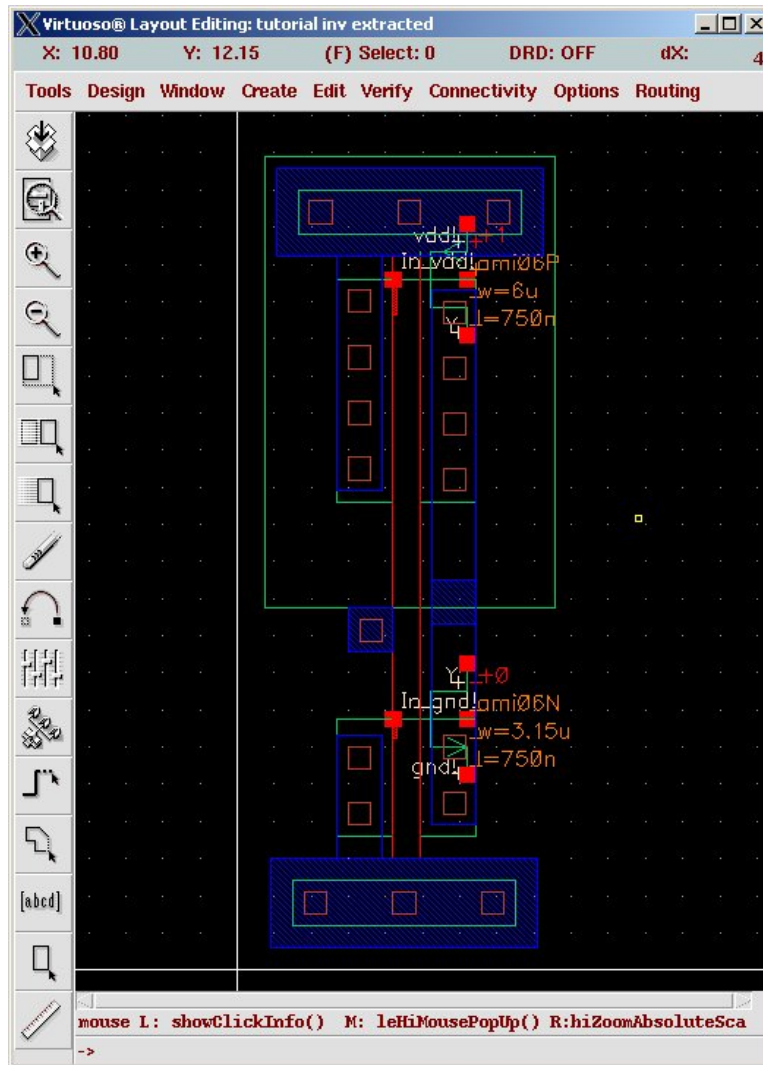


Figure 5.30: Extracted view of the inverter

5.6 Layout Versus Schematic Checking (LVS)

The process of verifying that an extracted layout captures the same transistor netlist as a schematic is known as *Layout versus Schematic* or (LVS) checking. In order to do this check you need both a **schematic** view and an **extracted** view of the cell. Note that this LVS process will not tell you if your schematic is correct. It will only tell you if the schematic and layout views describe the same circuit!

To run the LVS process using DIVA start by opening the **layout** view of a cell (for example, the inverter from Figure 5.16). From the layout window choose the **Verify** → **LVS...** menu choice. A new window pops up as shown in Figure 5.31. Notice that the **Rules file** field is already filled in with **divaLVS.rul**. This is the rules file in the **UofU_TechLib_ami06** library that compares the transistor netlist defined by the schematic (extracted from the schematic using the **netlister**) with the netlist defined by the **extracted** view. When this window pops up you may see a window as shown in Figure 5.32. This window is asking if you want to use old information for the LVS, or re-do the LVS using the information in the LVS form. You almost always want to select the **use form contents** option here before moving to the LVS window.

Once you get to the LVS window, you need to fill in the **Library**, **Cell**, and **View** for each of the **schematic** and **extracted** sections in the window. You can either type these in by hand, or use the **Browse** button to pop up a little version of the **Library Manager** where you can select the cells and views by clicking. Note that the LVS window in Figure 5.31 has the **Rewiring** and **Terminals** buttons selected. **Rewiring** means that the netlists can be changed so that the process will continue after the first error. **Terminals** means that the terminal labels in both the schematic and extracted views will be used as starting points for the matching. To have this be useful the terminals must be named identically in the layout and schematic.

You can also change some of the LVS options in the **NCSU** → **Modify LVS Rules ...** menu choice in the layout editor. This window, shown in Figure 5.33, sets various switches that modify how the LVS rules file interprets the two netlists. The default switches are shown in the figure and the names should be reasonably self-explanatory. Mostly they involve issues of whether structures in the netlist should be combined such that their effect is the same regardless of their physical connection. For example, the **Allow FET series Permutation** switch controls whether two netlists should be considered the same if series FETs are in a different order in the schematic and the layout. Often you don't care about the order in a series connection. If you do, unselect this switch before running LVS. **Combine Parallel FETs** controls whether two parallel FETs should be considered the same as

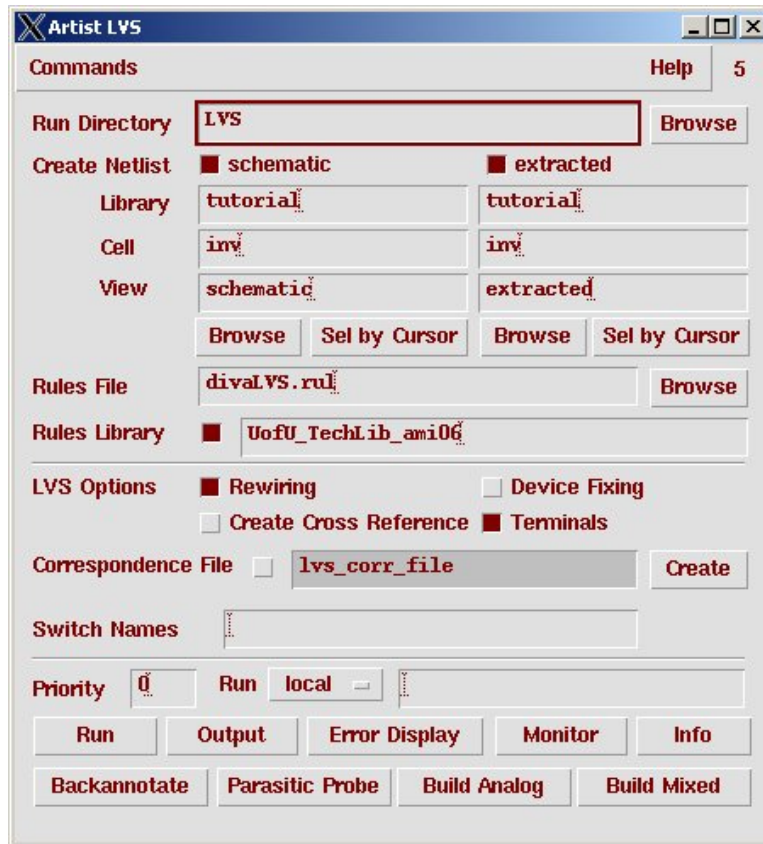


Figure 5.31: DIVA LVS control window

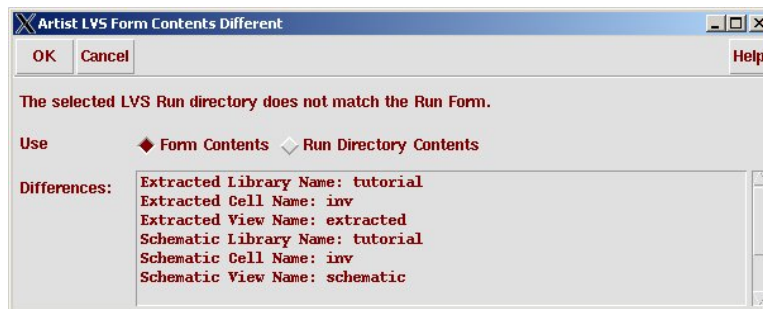


Figure 5.32: DIVA LVS Control Form

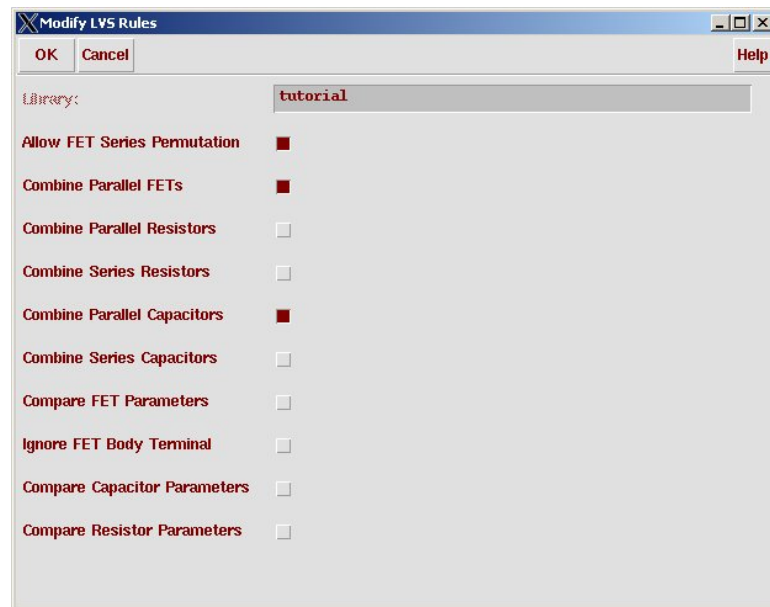


Figure 5.33: NCSU form to modify LVS rules

one FET. The logical effect is the same, so usually this is allowed. However, by default the LVS process does not check for matching FET parameters between the schematic and layout. If you would like this extra level of checking, you can select the **Compare FET Parameters** switch and the result is that two FETs in the same position in the netlist will not match if their width and length parameters do not match.

Once the window is filled in as in Figure 5.31 you can start the LVS process using the **Run** button. This will sometimes ask if it can save the schematic or extracted view before continuing. You almost always want to agree since you want to LVS against the most current views. Clicking on the **Run** button will have little visible effect. Look in the CIW and you'll see **LVS job is now started...** to indicate that things are running. If you think about LVS in general algorithmic terms, it's really performing exact graph matching on two circuit netlist graphs to see if the graphs are the same. This is a very hard problem in general (many graph matching problems are NP-complete, although it's not known if exact graph matching falls into this category). Hints such as using named terminals can make the problem much easier, but for large circuit this can still take quite a bit of time.

When the process finishes you will see a window like that in Figure 5.34. In this case the report is that the process **has succeeded**. If you get this result that's a great start, but all it means is that the LVS algorithm has finished. It does *not* tell you whether the graphs have matched (i.e. if the schematic



Figure 5.34: DIVA LVS completion indication

and extracted view describe the same circuit). For that you need to click the **Output** button in the LVS window. In this case, the Output window (Figure 5.35 tells us that although the numbers of **nets**, **terminals**, **pmos** and **nmos** transistors are the same in the two views, that the netlists failed to match. Scrolling down in this window (Figure 5.36) you can see that the problem is with the naming of the output terminal. The output terminal in this example was named **Y** in the layout (extracted) view and **Out** in the schematic view. This can be corrected by changing the name of one of the terminals and re-running LVS. After doing this the output of LVS shows that the netlists do indeed match.

*Remember that if you change anything in the **layout** view you need to re-run the extraction process to generate a new **extracted** view before re-running LVS.*

If instead of the **has succeeded** message after running LVS you get a **has failed** message you need to figure out what caused the LVS process to not run to completion. To see the process error messages to help figure this out select the **Info** button in the LVS control window (Figure 5.31), and in the **Display Run Information** window that pops up (Figure 5.37) select the **Log File** button to see why the LVS process failed. Usually this is because one of the two netlists could not be generated. The most common problems are either you didn't have the correct permissions for the files, or you may have forgotten to generate the extracted view altogether. You can also use the **Display Run Information** window to help track down issues where LVS reports that the netlists fail to match.

Tracking down LVS issues can be quite frustrating, but it must be done. If LVS says that the schematic and layout don't match, you must figure out what's going on. Experience shows that even though you can see no issues with your layout, that the LVS process is almost never wrong. Here are some thoughts about debugging LVS issues:

- - If LVS fails to run, then you have a problem right up front. Usually it's because you have specified the files incorrectly in the LVS dialog box. You can see what caused the problem by clicking on the Info button in the LVS dialog box, and then looking at the RunInfo LogFile.

```

/home/elb/IC_CAD/cadencetest/LVS/si.out
File Help 6
@(#)SADS: LVS.exe version 5.1.0 07/23/2006 23:38 (cicln01) $

Command line: /uusoc/facility/cad_tools/Cadence/IC5141ISR200607280110/tools/dfII/bin/
Like matching is enabled.
Net swapping is enabled.
Using terminal names as correspondence points.
Compiling Diva LVS rules...

Net-list summary for /home/elb/IC_CAD/cadencetest/LVS/layout/netlist
count
  4          nets
  4          terminals
  1          pmos
  1          nmos

Net-list summary for /home/elb/IC_CAD/cadencetest/LVS/schematic/netlist
count
  4          nets
  4          terminals
  1          pmos
  1          nmos

Terminal correspondence points
N2      N2      In
N3      N1      gnd!
N1      N0      vdd!

Devices in the rules but not in the netlist:
  cap nfet pfet nmos4 pmos4

The net-lists failed to match.

                layout schematic
                instances
un-matched      0      0
rewired         0      0
size errors     0      0
pruned         0      0
active          2      2
total           2      2

                nets
un-matched      0      0
merged         0      0
pruned         0      0
active          4      4
total           4      4

```

Figure 5.35: DIVA LVS output


```
File Help 6
pruned          0      0
active          4      4
total           4      4

                terminals
un-matched      1      1
matched but
different type  0      0
total           4      4

Probe files from /home/elb/IC_CAD/cadencetest/LVS/schematic
devbad.out:
netbad.out:
mergenet.out:
termbad.out:
T -1 Out /Out
? Terminal Out in the schematic is not present in the layout.
prunenet.out:
prunedev.out:
audit.out:

Probe files from /home/elb/IC_CAD/cadencetest/LVS/layout
devbad.out:
netbad.out:
mergenet.out:
termbad.out:
T -1 Y /Y
? Terminal Y in the layout is not present in the schematic.
prunenet.out:
prunedev.out:
audit.out:
```

Figure 5.36: DIVA LVS output (scrolled)

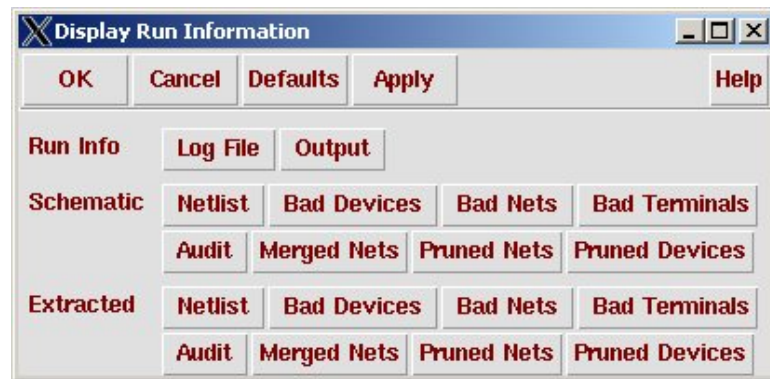


Figure 5.37: DIVA LVS Run Information Window

- If the LVS is successful, but the netlists fail to match, now you have a detective problem of tracking down the cause of the mismatch. Now you need to look at the Output Report to see what's going on.
- The first thing to look at is the output report and check the netlist summary for both **schematic** and **layout (extracted)** views. If the number of nets is different, that can give you a place to start looking. Let's assume that you've already simulated the schematic using a **Verilog** simulator so you believe that's working. (This should definitely be true!).
 - If you have fewer nets in the layout than you do in the schematic, suspect that there are things connected together in the layout that weren't supposed to be. This is easy to do by overlapping some metal that you didn't intend to overlap (for example).
 - If there are more nets in the layout than in the schematic, then there are nodes that were supposed to be connected, but weren't. This too is easy to do, by leaving out a contact or via, for example.
 - Now you go into detective mode to figure out what's different. For small cells you just poke around in the layout with **Connectivity** → **Mark Net** to see the connected layout and see if there's anything connected that shouldn't be.
 - You can also open the extracted layout and look at things there. The advantage of this is that the extracted layout has the netlist already constructed so when you select something you're selecting the whole connected layout net by default. You can also use **Q** with the selected net to see thing like net number of the selected net.

- You can also use the error reporting mechanism in LVS to see which nets are different. After running LVS, open a schematic window and an extracted view window for the cell in question. Now click on the **Error Display** button in the LVS dialog box. This will let you walk through the errors in the LVS output file one at a time and will highlight in the schematic or the extracted view which node that error is talking about. This can be very helpful if you have only a few errors in the output file, but can also be more confusing than helpful if you have a lot of errors. This is because once you have one error, it can cause lots of things to look like errors. This can, however, be a good way to see that the output file is talking about, and you might walk through the errors and see a big connected net in the layout that you thought should be two separate nodes (for example).
- If you'd like to look at the extracted view and search for a particular net, you can do it through a slightly non-obvious procedure. Of course, you can use the **Error Display** to step through the errors until you get to the one with the net you're interested in. Or you can open the extracted view, select **Verify** → **Shorts**. Dismiss the informational dialog box to get the **Shorts Locator** box. Type the net name or number in the **Net Name** box and click on **Run**. This will highlight that node in orange. Click **Finish** to start over with another net. Clearly this dialog box does something more interesting than just highlight a net. It has something to do with tracking down shorts between nets, but I have to admit I don't know exactly what it does. I do know that it highlights nets by name in extracted views!
- Remember that a single LVS error can cause lots of things to look like errors. Once you find one error, rerun LVS and see what's changed. A single fix can cause lots of reported violations to go away.
- If all else fails and you're convinced that things are correct, try removing the `~/IC_CAD/cadence/LVS` directory and re-running LVS. It's possible (but rare) that you mess LVS up so badly that you need to delete all its generated files and start over.

If your cell passes LVS then it's a pretty good indication that your layout is a correctly functioning circuit that matches the behavior of the transistor schematic.

5.6.1 Generating an analog-extracted view

The LVS process compares the **schematic** view with the **extracted** view to see if the netlists are the same. Once LVS says that the netlists are the same,

there is one further step that you can take. Click on the **Build Analog** button in the **LVS** control window (Figure 5.31). Make sure to **Include All** the **extracted paracitics** so that if you extracted paracitic capacitors they will be included, and click **OK**. This will generate yet another **Cell View** called **analog-extracted**. This view is very similar to the **extracted** view, but has additional power supply information so that the view can be simulated with the analog simulator Spectre. This will be seen in Chapter 6.

5.7 Overall Cell Design Flow (so far...)

The complete cell design flow involves each of the steps defined previously. They are:

1. Start with a schematic of your desired circuit. The schematic can be made using components from the **UofU_Digital_v1.1** library and the transistors from the **NCSU_Analog_Parts** and **UofU_Analog_Parts** libraries. Use **Composer** as described in Chapter 3.
2. Simulate that schematic using one of the Verilog simulators defined in Chapter 4 to make sure the functionality is what you want. Make sure to use a self-checking testbench.
3. Now draw a layout of the cell using the **Virtuoso** as described in this Chapter.
4. Make sure that the **layout** view passes DRC.
5. Generate an **extracted** view.
6. Make sure that the **schematic** and **extracted** view match by using the LVS checker.
7. Generate the **analog-extracted** cell view for later analog simulation.

5.8 Standard Cell Template

Designing a standard cell library involves creating a set of cells (gates, flip flops, etc.) that work together throughout the CAD tool flow. In general this means that the cells should have all the views necessary for designing with the cells, and should be compatible in terms of their attributes so that they can all work together. At this point in our understanding of the CAD flow this means that each cell in the cell library needs the following views:

schematic: This view defines the gate or transistor level definition of the cell. It can be simulated using any of the Verilog simulators in Chapter 4 as a switch-level or a behavioral-level simulation.

behavioral: This view is the Verilog description of the cell. It should include both behavior and **specify** blocks for timing so that the timing can be back-annotated with better estimates as the design progresses through the flow.

layout: This view describes the mask layout of the cell. It should pass all DRC checks. It should also follow strict physical and geometrical standards so that the cells will fit with each other and work together (the subject of this Section).

extracted: This view extracts the circuit netlist from the layout and is generated by the extraction process. It should be used with the LVS checker to verify that the **layout** and **schematic** views represent the same circuit.

analog-extracted: This view is an augmented version of the **extracted** view that includes information so it can be used by the analog simulator **Spectre**. It is generated from the extracted view through the LVS process.

In addition to these views, subsequent Chapters in this text will introduce a number of additional views that are required for the final, complete cell library. They include:

abstract: This is a view that is derived from the layout. It tells the place and route tool where the ports of the cell are, and where the “keep-outs” or obstructions of the cell are that it should not try to route over. Generating this view from the layout is described in Chapter 9.

LEF: This is a Library Exchange Format file that is derived from the abstract view of the cell and is read by the place and route tool (**SOC Encounter**) so that it can get information about the technology that it is routing in, and about the abstract views of the cells in the library. This view is also generated by the **Abstract** program.

Verilog Interface: The system you design will eventually be input to the place and route tool as a structural verilog file that describes the standard cell gates used in your design and the connections between them. In addition to this file you need a simple I/O interface of each cell (separate from the LEF file) so that the place and route tool can parse the structural verilog file. This will also be described in Chapter 9.

Lib: This is a standard format, called **Liberty** format (usually with a **.lib** filename extension), for describing cells so that the synthesis tools can understand the cells. The file describes the I/O interface, the logic function, the capacitance on the cell pins, and extracted timing for the cell. This file can be generated by hand, using analog simulation using Spectre or Spice, or using the SignalStorm tool from Cadence and will be described in Chapter 7.

5.8.1 Standard Cell Geometry Specification

In standard cell based design flow, the automatic place and route tool (SOC Encounter in this flow) places the standard cells selected by the synthesis engine (Synopsys design compiler or Cadence BuildGates) in a set of rows. To make this packing as efficient as possible we can plan our cells so that they can be placed adjacent to each other. With some simple rules about how the internal layout of the cells is designed, we can guarantee that any two cells can be placed next to each other without causing any DRC violations. Some overall design considerations are:

- Since the cells are going to be placed in rows adjacent to each other we will make all our cells the same height so that they fit nicely into rows.
- All our CMOS based standard cells have a set of **pmos** transistors in an **nwell** for the pull-up network and **nmos** transistors for the pull-down network. We will always place the **nwell** with **pmos** transistors on top of the layout and **nmos** transistors at the bottom. By doing this and having the same height in the **nwell** layer for all cells, we can have a continuous well throughout the row of cells.
- All standard cells have a power and ground bus. To avoid shorting of power and ground while placing cells adjacent to each other, we place the **vdd!** bus on top and **gnd!** bus at the bottom of the cell (which makes sense because we have the pull up network on top and pull down at the bottom). Having equal heights for these buses for all cells and running these buses throughout the width of the cells allow for automatic connection of adjacent cell power and ground buses by abutment.
- All the standard cell layouts will have a fixed origin in the lower left corner of the layout, but pay attention, it is not the absolute lower left of the layout!

Hence we define the following standard cell dimensions for class cells that will be implemented in the AMI C5N 0.5 μ CMOS process.

Cell height = 27 microns. This height is measured from the center of the **gnd!** bus to the center of the **vdd!** bus.

Cell width = multiple of 2.4 microns. This matches the vertical **metal2** routing pitch and also the pitch of the **NTAP** and **SUBTAP** well and substrate contacts.

nwell height from top of cell = 15.9 microns. This is 2/3 of the cell height because **pmos** transistors are usually made wider than **nmos** transistors.

vdd! and gnd! bus height = 2.4 microns. This height is centered on the supply lines so that 1.2 microns is above the cell boundary and 1.2 microns is below the cell boundary. This way if the cells are abutted vertically with the supply lines overlapping those supply lines remain 2.4 microns wide.

vdd! and gnd! bus width = 1.2 microns overhang beyond the cell boundary. This is for convenience so that the **metal1** layer of the well and substrate contacts can overlap the cells safely.

Based on the **nwell** size and placement, and the MOSIS SCMOS rules, we can now standardize the **nselect** and **pselect** sizes as well. The closest that select edges can be to the well edges is 1.2 microns (SCMOS rules 2.3 and 4.2). So, the select restrictions are:

pselect height from inner edge of vdd! bus = 13.5 microns

nselect height from inner edge of gnd! bus = 8.7 microns

nselect and pselect width = cell width

nwell overhang outside cell width = 1.5 microns (to top, left and right of the cell)

To avoid *latch-up* in the circuit we place substrate and well contacts to make sure that the **nwell** is tied to vdd and the substrate is tied to gnd. As seen in Section 5.2, we use predefined contacts for these connections in the form of **NTAP** and **SUBTAP** contacts. Each of these contacts is 2.4 microns wide so you can think of the overall cell width as a multiple of 2.4 microns, or as an integer number of **NTAP** and **SUBTAP** cells. These contacts are placed centered on the **vdd!** and **gnd!** lines. Note that placing this many well and substrate contacts is overkill, but does not have negative effects.

The usual rule of thumb for the minimum number of well contacts is to place one well contact for every square of well material.

It's a good idea to build a standard cell layout *template* incorporating these basic design specifications. This can serve as a starting point for the

design of new cells. It's also a good idea to keep your library cells in a separate Cadence library from your designs. This keeps the distinction clear between the library cells and the designs that use those cells. You could, for example, make a template with the **gnd!** bus on the bottom, including the **SUBTAP** contacts, the **vdd!** bus on the top, including the **NTAP** contacts, and the **nwell**, **nselect**, and **pselect** layers as defined earlier. An example of such a template with rulers showing the dimensions, and example **pmos** and **nmos** transistors is shown in Figure 5.38. This cell, even without actual connections between the transistors, should pass DRC with no violations.

*The cell boundary will be generated as a rectangle of type **prBound** by the **Abstract** program. This layer is not editable by default, but can be enabled if needed through the **LSW** window.*

The notion of a *cell boundary* was used without specific definition in the preceding text. The cell boundary is the rectangle that defines the outline of the cell from the place and route tool's point of view. We have defined our library so that the cell boundary is actually inside of the cell layout geometry so that when the cells are abutted with respect to the cell boundary, they actually overlap by a small amount. The cell boundary is shown in Figure 5.39, but you don't have to draw it by hand. It will be derived during the process of extracting an **abstract** view as described in Chapter 9.

*In order to make sure that there will not be any DRC violations when cells are abutted, it is very important that no geometry on any layers (other than the layers already in the template) falls within 0.6 microns from the left and right of the cell boundary or within the **vdd!** or **gnd!** buses.* This rule is in addition to other SCMOS Design Rules within the cell and applies even if the DRC of the individual standard cell does not complain. For example, in the layout shown in Figure 5.39, the **nactive** and **pactive** layers are placed 0.6μ away from the cell boundaries on left and right side. They cannot come any closer to the boundary. Also, in the **layout** view shown, the **pmos** and **nmos** transistor gates (**poly** layer) are placed just touching the **vdd!** and **gnd!** bus inner edges. The gates could go farther away from the bus inner edges but cannot extend into the bus structure. If you need more space than these rules allow, you will have to make your whole cell wider. Each time you make a cell wider, it has to be made wider in increments of 2.4μ .

5.8.2 Standard Cell I/O Pin Placement

The design rules for a specific VLSI process define the minimum dimensions and spacing for all layers, including the metal layers that are used for routing signals. The minimum spacing between two metal routing layers is called the *pitch* of that layer. It's also more efficient for overall wiring density if the metal wiring on each layer of metal is restricted to a single direction. This is well known by people who make multi-layer printed circuit

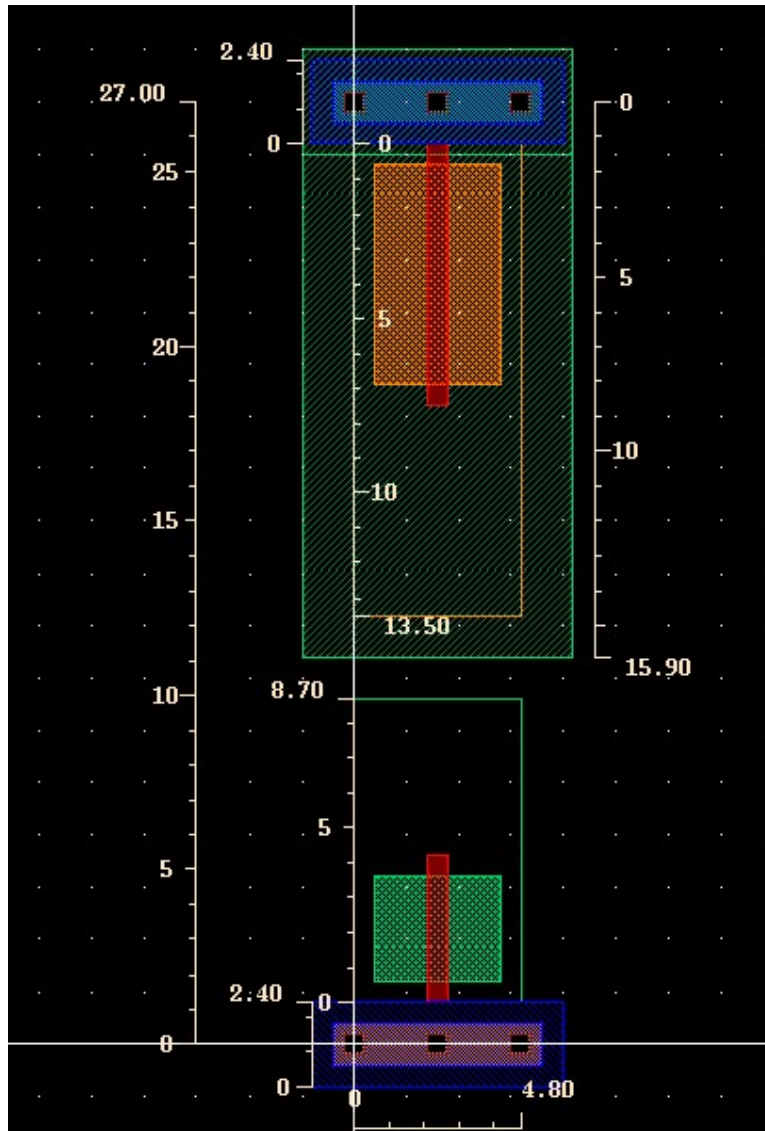


Figure 5.38: A **layout** template showing standard cell dimensions

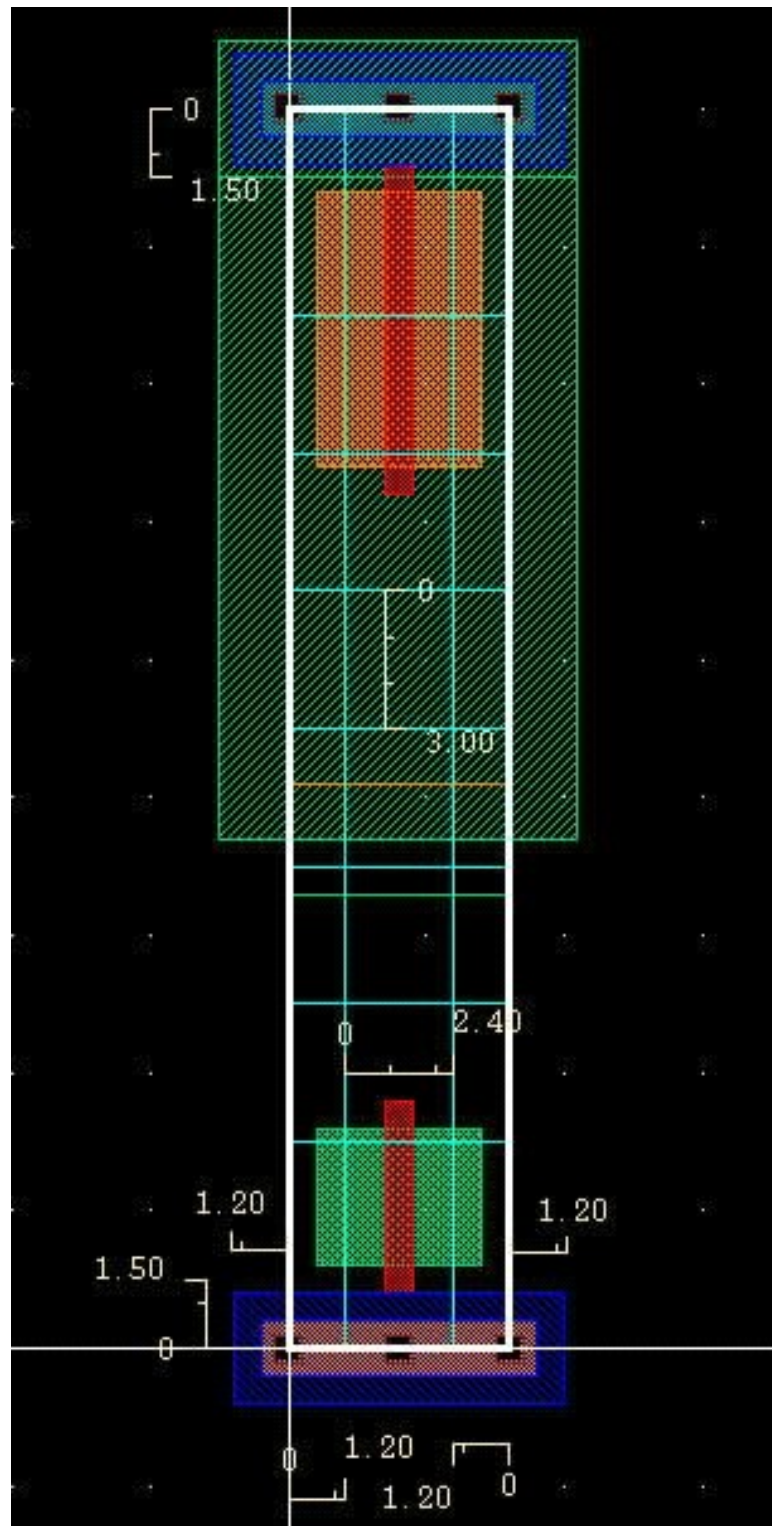


Figure 5.39: A **layout** template showing cell boundary (**prBound** layer)

boards and the principle is the same for VLSI chips. In our case we will eventually tell the place and router to prefer horizontal direction routing for layers **metal1** and **metal3**, and vertical routing for layer **metal2**. The pitch for **metal2** vertical routing according to the Mosis SCMOS design rules is 2.4μ which, conveniently (but not coincidentally), matches our standard cell dimension increment and well and substrate contact width. The **metal3** routing pitch is 3.0μ and so sets the routing pitch for all horizontal routing by the place and route tool even though **metal1** can actually be routed on a finer pitch when routed by hand.

It turns out to make the job of the place and route tool much easier if the connection points to the cell are centered on horizontal and vertical routing pitch dimensions. The grid lines in Figure 5.39 (actually 0-width rectangles of type **wire** in the layout) show the horizontal and vertical routing channels in the cell. We follow offset grid line placement rules where the grid lines are offset by half of the vertical and horizontal grid line pitch from the origin. Thus the vertical grid lines start at 1.2μ from the y-axis and are spaced 2.4μ thereafter. Similarly, the horizontal grid lines start at 1.5μ from the x-axis and are spaced 3.0μ thereafter. However it should be noted that the first and last horizontal grid lines cannot be used, as any geometry placed on them extends into the **vdd!** or **gnd!** bus violating the rules defined above.

*Remember to use **shape pins** for all connection pins in a cell.*

Input and output pins for a cell should be placed at the intersection of the horizontal and vertical routing channels if at all possible. Also, if possible, do not place two pins on the same vertical or horizontal routing channel. For our cells it is most efficient if all cell I/O pins are made in **metal2**. This makes it easy for the router to jump over the **vdd!** and **gnd!** buses to make contact with the pin using a **metal2** vertical wire.

As far as possible use only **metal1** for signal routing within the standard cell layout. Routing on **metal2** layer can be used if necessary, but if it is used it should always run centered on a vertical routing channel line and never run horizontally except for very short distances (this is to reduce metal2 blockage layers enabling easier place and route). In any case, make sure that there are no horizontal metal2 runs around I/O pins. You should try to avoid any use of **metal3** in your library cells. That routing layer should be reserved for use by the place and route tool. I know of at least one commercial standard cell library for a three-metal process like ours that contains over 400 cells and uses only **metal1** in the cells (other than the 1.2μ by 1.2μ **metal2** connection pins) so it is possible to define lots of cells without using more than one metal layer.

5.8.3 Standard Cell Transistor Sizing

For critical designs MOSIS recommends the minimum transistor width to be 1.5μ for the AMI C5N process. The minimum transistor length for AMI C5N process is 0.6μ . Hence the minimum dimension for an **nmos** transistor pulling the output directly to **gnd!** is $W/L = 1.5\mu/0.6\mu$. To roughly equalize the strength of the pullup and pulldowns the minimum dimension of a **pmos** transistor pulling the output directly to **vdd!** should be $W/L = 3.0\mu/0.6\mu$. For series stack of transistors between output and **vdd!** or **gnd!**, the transistor width should be the number of transistors in the stack multiplied by the minimum width for a single transistor stack. If you can't make the series transistors quite wide enough, at least do your best. Also, a stack greater than four should not be used. For example, if there are three **pmos** transistors in a series stack between output node and **vdd!**, then these transistors will be sized $W/L = (3 \times 3.0)\mu/0.6\mu = 9.0\mu/0.6\mu$ if possible.

*The 2:1 width ratio roughly compensates for the mobility of majority carriers in **pmos** transistors (holes) which is roughly half of that of **nmos** transistor carriers (electrons).*

You can also create cells with different drive strengths by adjusting the width of the output transistors. For example, an inverter with a 1.5μ pull-down and 3.0μ pullup could be considered a 1x inverter, and one with a 3.0μ pull-down and 6.0μ pullup would then be a 2x inverter denoting that its output drive is two times as strong as the 1x inverter.

A 2x inverter has extra output drive, but also extra input capacitance, so the resulting logical effort is the same, but the electrical effort is different.

A standard cell template for a cell that is four contacts wide is shown in Figure 5.40. The standard 1x inverter from the **UofU_Digital_v1_1** cell library is shown in Figurefig:inv1x, and the 1x two-input nand gate from that library is shown in Figure 5.42.

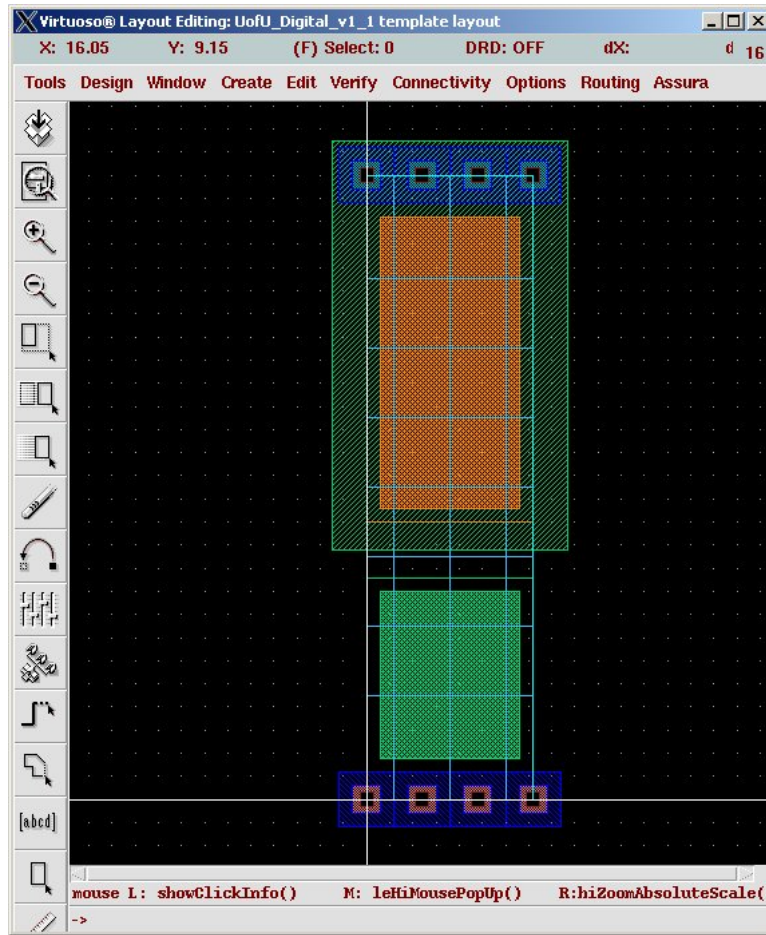


Figure 5.40: A **layout** template for a four-wide cell

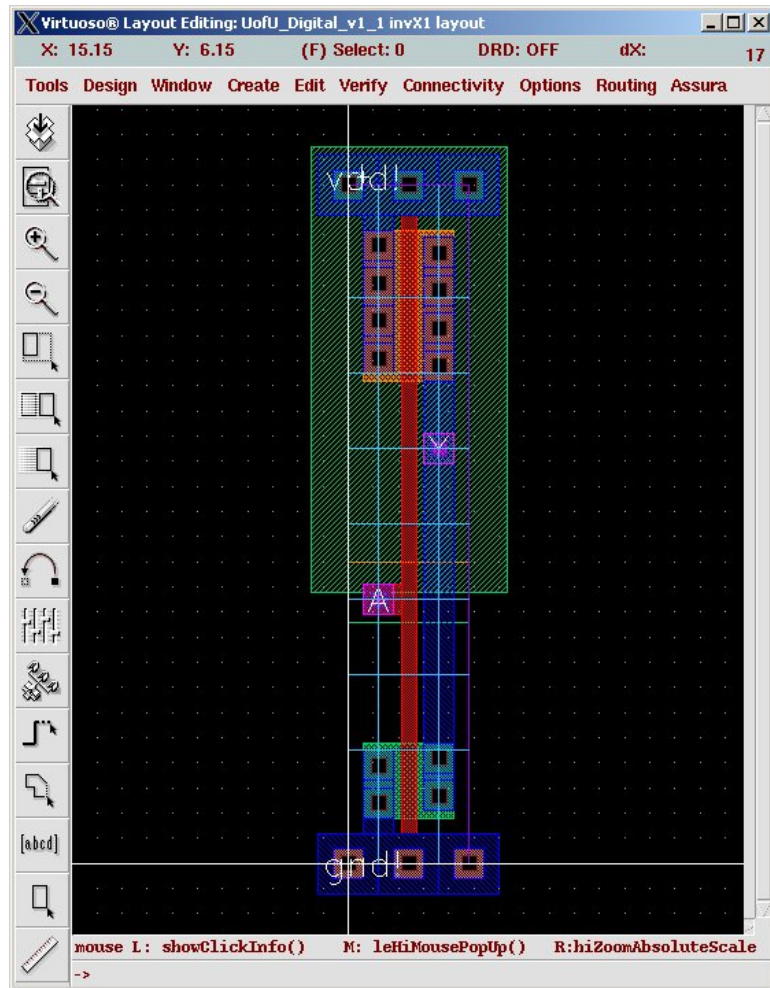


Figure 5.41: Standard cell layout for a 1x inverter

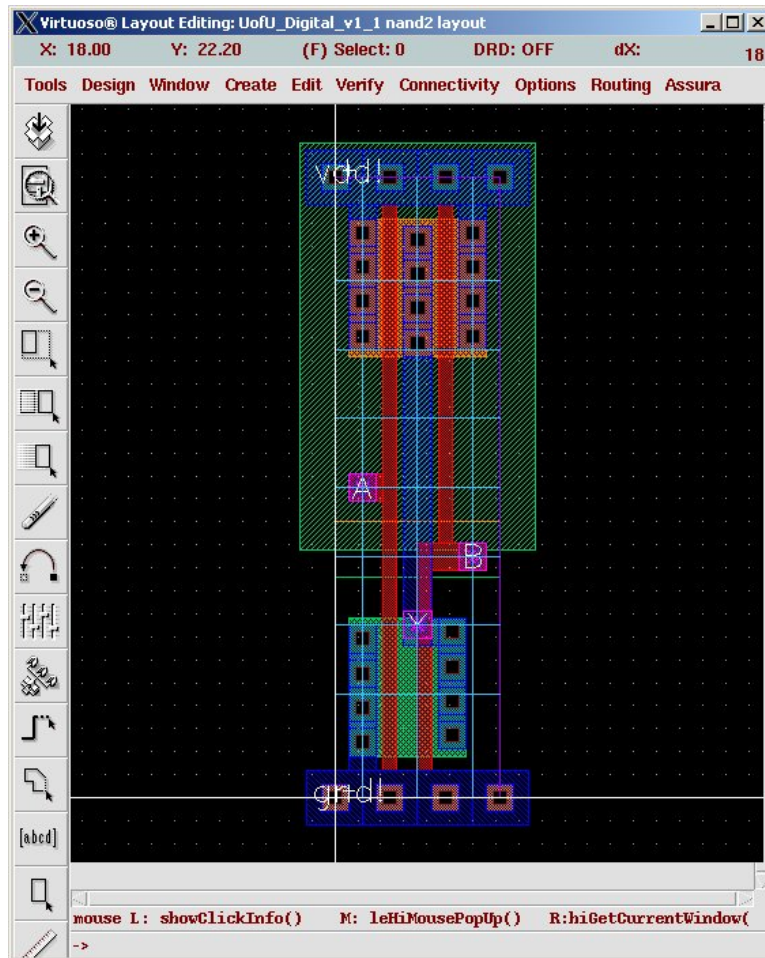


Figure 5.42: Standard cell layout for a 1x NAND gate

