CMOS VLSI Design
Lab 1: Cell Design and Verification

This is the first of four chip design labs developed at Harvey Mudd College. These labs are intended to be used in conjunction with *CMOS VLSI Design*, 4th Ed. They teach the practicalities of chip design using industry-standard CAD tools from Cadence and Synopsys.

This lab teaches you the basics of how to use the computer-aided design (CAD) tool to design, simulate, and verify schematics and layout of logic gates. It also serves as a stand-along tutorial to quickly get up to speed with the Cadence tools. The first step is to draw a schematic indicating the connection of transistors to build cells such as NAND gates, NOR gates, and NOT gates. These cells are simulated by applying digital inputs and checking that the outputs match expectation. A symbol for the cell is also created. The next step is to draw a layout indicating how the transistors and wires are physically arranged on the chip. The layout is checked to ensure it satisfies the design rules and that the transistors match the schematic.

The next three labs extend this one to build the 8-bit MIPS microprocessor described in Chapter 1. Most of the microprocessor is provided, but one of each interesting piece has been removed. In Lab 2, you learn to put gates together to form a datapath. In Lab 3, you synthesize a controller. In Lab 4, you place the controller and datapath together and route the whole thing to input/output pads to form a complete chip.

I. An Overview of VLSI CAD Tools

VLSI designers have a wide variety of CAD tools to choose from, each with their own strengths and weaknesses. The leading Electronic Design Automation (EDA) companies include Cadence, Synopsys, Magma, and Mentor Graphics. Tanner also offers commercial VLSI design tools. The leading free tools include Electric, Magic, and LASI.

This set of laboratories uses the Cadence and Synopsys tools because they have the largest market share in industry and are capable of handling everything from simple class projects to state-of-the-art billion-transistor integrated circuits. The full set of tools is extremely expensive (on the order of $1M per user), but the companies offer academic programs to make the tools available to universities at a much lower cost. The tools run on Linux and other flavors of Unix. Setting up and maintaining the tools involves a substantial effort. However, Erik Brunvand's book, *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*, greatly reduces the difficulty of learning to install and use the tools. Once they are setup correctly, the basic tools are easy to use, as this tutorial demonstrates.

Some companies use the Tanner tools because their list price is much lower and they are easy to use. However, their academic pricing is comparable with Cadence and Synopsys, giving little incentive for universities to adopt Tanner.

The Electric VLSI Design System is an open-source chip design program developed by Steve Rubin with support from Sun Microsystems. It is written in Java and hence runs on virtually any operating system, including Windows, Linux, and Mac. It is easy to use with older fabrication processes and was used in previous incarnations of this lab. Electric presently does not read the design rules for state-of-the-art nanometer processes and poorly integrates with synthesis and place & route.

Magic is a free Linux-based layout editor with a powerful but awkward interface that was once widely used in universities. The Layout System for Individuals, LASI, developed by David Boyce, is freely available and runs on Windows. It was last updated in 1999.

There are two general strategies for chip design. *Custom design* involves specifying how every transistor is connected and physically arranged on the chip. *Synthesized design* involves describing the function of a digital chip in a hardware description language such as Verilog or VHDL, then using a computer-aided design tool to automatically generate a set of gates that perform this function, place the gates on the chip, and route the wires to connect the connect the gates. The majority of commercial designs are synthesized today because synthesis takes less engineering time. However, custom design gives more insight into how chips are built and into what to do when things go wrong. Custom design also offers higher performance, lower power, and smaller chip size. The first two labs emphasize the fundamentals of custom design, while the next two use logic synthesis and automatic placement to save time.

II. Tool Setup

These labs assume that you have the Cadence and Synopsys tools installed.

The tools generate a bunch of random files. It's best to keep them in one place. In your home directory, create some directories by typing:

```
mkdir IC_CAD
mkdir IC_CAD/cadence
```

III. Getting Started

Before you start the Cadence tools, change into the cadence directory:

```
cd ~/IC_CAD/cadence
```

Each of our tools has a startup script that sets the appropriate paths to the tools and invokes them. Start Cadence with the NCSU extensions by running

```
cad-ncsu &
```

A window labeled Virtuoso will open up. If it does not, please refer back to the Hitchhikers Guide to help you debug any issues. A "What's New" and a Library Manager window may open up too. You can turn off the "What's New" window in the future by choosing Edit • Off at Startup.

You may see some warnings about "no route to host" and missing fonts in the terminal; you can safely ignore these. Scroll through the Virtuoso window and look at the messages displayed as the tool loads up. Get in the habit of watching for the messages and recognizing any that are out of the ordinary. This is very helpful when you encounter problems. At present, you shouldn't see any warnings in Virtuoso.

All of your designs are stored in a *library*. If the Library Browser doesn't open, choose Tools • Library Manager. You'll use the Library Manager to manipulate your libraries. Don't try to move libraries around or rename them directly in Linux; there is some funny behavior and you are likely to break them.[1]

When you are all done, be sure to quit Cadence by choosing File • Exit… in the Virtuoso window. If you don't, you may lose the work you've done and/or leave your library in a corrupted and unstable state. An easy and unpleasant mistake is to close the lid of your laptop and cut off your network connection before quitting Cadence, thereby terminating your X windows session and all of your programs including Cadence. If this occurs, you may find a `panic.log` file in your home directory. The file will include directions to try to recover your work. Restart `cad-ncsu`. Before doing anything else, enter the instructions from `panic.log` into the Virtuoso window command line. For example:

```
dbOpenPanicCellView("lab3_xx" "test" "schematic")
```

Familiarize yourself with the Library Manager. Your cds.lib file includes many libraries from the North Carolina State University Cadence Design Kit supporting the different MOSIS processes. It also includes libraries from the University of Utah. The File menu allows you to create new libraries and cells within a library, while the Edit menu allows you to copy, rename, delete, and change the access permissions.

Create a library by invoking File • New • Library… in the Library Manager. Name the library lab1_xx, where xx are your initials. Leave the path blank and it will be put in your current working directory (`~/IC_CAD/cadence`). Choose the "Attach to existing tech library" choose UofU_TechLib_ami06 (also known as UofU AMI 0.60u C5N (3M, 2P, high-res)). This is a technology file for the American Microsystems (now Orbit Semiconductor) 0.6 μm process, containing design rules for layout.

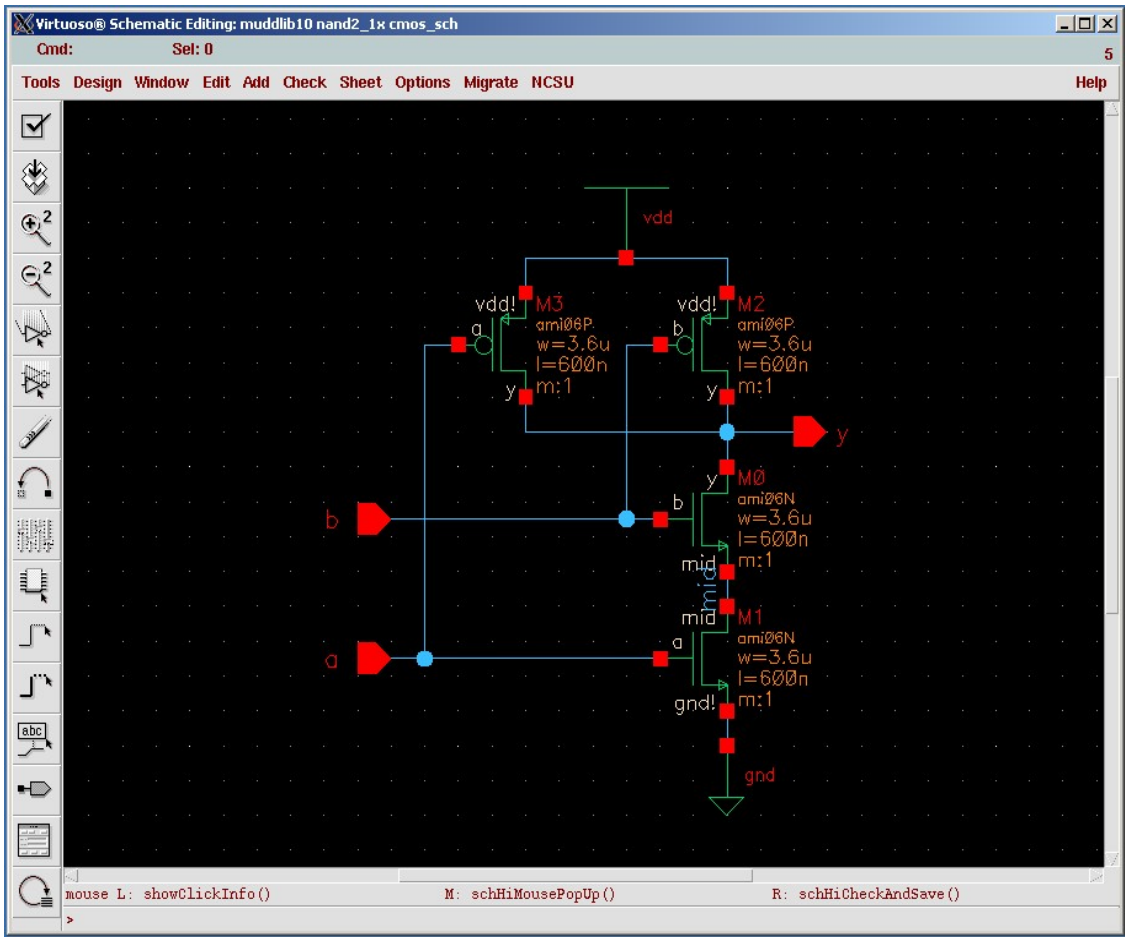
[1] Copying a library does work correctly, and you will find this helpful when working with a partner on the final project.

IV. Schematic Entry

Our first step is to create a schematic for a 2-input NAND gate. Each gate or larger component is called a *cell*. Cells have multiple views. The schematic view for a cell built with CMOS transistors will be called cmos_sch. Later, you will build a view called layout specifying how the cell is physically manufactured.

In the Library Manager, be sure your lab1_xx is highlighted. Choose File • New • Cell View… In your lab1_xx library, enter a cell name of nand2 and a view name of cmos_sch. The type should be schematic and always open with Schematics XL. You may get a window asking you to confirm that cmos_sch should be associated with this tool. The schematic editor window will open.

Your goal is to draw a gate like the one shown in Figure 1. We are working in a 0.6 μm process with λ = 0.3 μm. Unfortunately, the University of Utah technology file is configured on a half-lambda grid, so grid units are 0.15 μm. Take care that everything you do is an integer multiple of λ so you don't come to grief later on. Our NAND gate will use 12 λ (3.6 μm) nMOS and pMOS transistors.


Virtuoso® Schematic Editing: muddlib10 nand2_1x cmos_sch
Cmd:
Sel: 0
5
Tools Design Window Edit Add Check Sheet Options Migrate NCSU
Help
vdd
vdd! M3 ami06P w=3.6u l=600n m:1
a
y
vdd! M2 ami06P w=3.6u l=600n m:1
b
y
y
y M0 ami06N w=3.6u l=600n m:1
b
mid
mid
mid M1 ami06N w=3.6u l=600n m:1
a
gnd!
gnd
b
a

```
mouse L: showClickInfo()          M: schHiMousePopUp()          R: schHiCheckAndSave()
>
```

Figure 1: nand2 cmos_sch

Choose Create • Instance to open a Component Browser window. (The menu lists the keyboard shortcut for each command, such as *i* for create instance. You'll want to learn the shortcuts you use most often.) Choose UofU_Analog_Parts for the library, and then select nmos. The Create Instance dialog will open. Set the Width to 3.6u (u indicates microns). Click in the schematic editor window to drop the transistor. You can click a second time to place another transistor. Return to the Component Browser window and choose pmos. Drop two pMOS transistors, being sure their widths are also 3.6u. Then return to the browser and get a gnd and a vdd symbol. (Note that the Component Browser has a nasty habit of getting minimized. Sometimes you have to find and unminimize it.)

When you are in a mode in the editor, you can press ctrl-c or Esc to get out of it. Other extremely useful commands include Edit • Move, Edit • Copy, Edit • Undo, and Edit • Delete. Edit • Properties • Object… is also useful to change things like transistor sizes or wire names. Move the elements around until they are in attractive locations. I like to keep series transistors one grid unit apart and place pMOS transistors two grid units above the nMOS. Look at the bottom of the schematic editor window to see what mode you are in.

Next, use Create • Pin… (Keyboard shortcut *p*) to create some pins. In the Create Pin dialog, enter "a b" (no quotation marks, and a space between the two pin names). Make sure the direction is "input." The tools are case-sensitive, so use lower case everywhere. Place the pins, being sure that a is the bottom one. Although pin order doesn't matter logically, it does matter physically and electrically, so you will get errors if you reverse the order. Then place an output pin y.

Now, wire the elements together. Choose Create • Wire (narrow). Click on each component and draw a wire to where it should connect.

It is a good idea to make sure every net (wire) in a design has a name. Otherwise, you'll have a tough time tracking down a problem later on one of the unnamed nets. Every net in your schematic is connected to a named pin or to power or ground except the net between the two series nMOS transistors. Choose Create • Wire name… Enter mid or something like that as the name, and click on the wire to name it.

Choose File • Check and Save to save your schematic. You'll probably get one warning about a "solder dot on crossover" at the 4-way junction on the output node. This is annoying because such 4-way junctions are normal and common. Choose Check • Rules Setup… and click on the Physical tab in the dialog. Change Solder On CrossOver from "warning" to "ignored" and close the dialog. Then Check and Save again and the warning should be gone. If you have any other warnings, fix them. A common mistake is wires that look like they might touch but don't actually connect. Delete the wire and redraw it.

Poke around the menus and familiarize yourself with the other capabilities of the schematic editor.

V. Logic Verification

Cells are commonly described at three levels of abstraction. The register-transfer level (RTL) description is a Verilog or VHDL file specifying the behavior of the cell in terms of registers and combinational logic. It often serves as the specification of what the chip should do. The schematic illustrates how the cell is composed from transistors or other cells. The layout shows how the transistors or cells are physically arranged.

Logic verification involves proving that the cells perform the correct function. One way to do this is to simulate the cell and apply a set of 1's and 0's called test vectors to the inputs, then check that the outputs match expectation. Typically, logic verification is done first on the RTL to check that the specification is correct. A testbench written in Verilog or VHDL automates the process of applying and checking all of the vectors. The same test vectors are then applied to the schematic to check that the schematic matches the RTL. Later, we will use a layout-versus schematic (LVS) tool to check that the layout matches the schematic (and, by inference, the RTL).

You will begin by simulating an RTL description of the NAND gate to become familiar with reading RTL and understanding a testbench. In this tutorial, the RTL and testbench are written in System Verilog, which is a 2005 update to the popular Verilog hardware description language.

There are many Verilog simulators on the market, including NC-Verilog from Cadence, VCS from Synopsys, and ModelSim from Mentor Graphics. This tutorial describes how to use NC-Verilog because it integrates gracefully with the other Cadence tools. NC-Verilog compiles your Verilog into an executable program and runs it directly, making it faster than the older interpreted simulators.

Make a new directory for simulation (e.g. `nand2sim`). Copy `nand2.sv`, `nand2.tv`, and `testfixture.verilog` from the course directory into your new directory.

```
mkdir nand2sim
cd nand2sim
cp /courses/e158/15/lab1/nand2.sv .
cp /courses/e158/15/lab1/nand2.tv .
cp /courses/e158/15/lab1/nand2.testfixture testfixture.verilog
```

`nand2.sv` is the SystemVerilog RTL file, which includes a behavioral description of a nand2 module and a simple self-checking testbench that includes `testfixture.verilog`. `testfixture.verilog` reads in testvectors from `nand2.tv` and applies them to pins of the nand2 module. After each cycle it compares the output of the nand2 module to the expected output, and prints an error if they do not match. Look over each of these files and understand how they work.

First, you will simulate the nand2 RTL to practice the process and ensure that the testbench works. Later, you will replace the behavioral nand2 module with one generated from you Electric schematic and will resimulate to check that your schematic performs the correct function.

At the command line, type `sim-nc nand2.sv` to invoke the simulator. You should see some messages ending with

```
ncsim> run
Completed  4 tests with  0 errors.
Simulation stopped via $stop(1) at time 81 NS + 0
```

You'll be left at the ncsim command prompt. Type `quit` to finish the simulation.

If the simulation hadn't run correctly, it would be helpful to be able to view the results. NC-Verilog has a graphical user interface called SimVision. The GUI takes a few seconds to load, so you may prefer to run it only when you need to debug. To rerun the simulation with the GUI, type `sim-ncg nand2.sv`

A Console and Design Browser window will pop up. In the browser, click on the + symbol beside the testbench to expand, then click on dut. The three signals, a, b, and y, will appear in the pane to the right. Select all three, then right-click and choose Send to Waveform Window. In the Waveform Window, choose Simulation • Run. You'll see the waveforms of your simulation; inspect them to ensure they are correct. The 0 errors message should also appear in the console. If you needed to change something in your code or testbench or test vectors, or wanted to add other signals, do so and then Simulation • Reinvoke Simulator to recompile everything and bring you back to the start. Then choose Run again.

Make a habit of looking at the messages in the console window and learning what is normal. Warnings and errors should be taken seriously; they usually indicate real problems that will catch you later if you don't fix them.

VI. Schematic Simulation

Next, you will verify your schematic by generating a Verilog deck and pasting it into the RTL Verilog file. While viewing your schematic, click on Launch • Simulation • NC-Verilog to open a window for the Verilog environment. Note the run directory (e.g. nand2_run1), and press the button in the upper left to initialize the design. (It is important to let Virtuoso create and initialize the directory for you; if you manually create a run directory in Linux, you'll encounter errors.) Then press the next button to generate a netlist. Look in the Virtuoso window for errors and correct them if necessary. You should see that the pmos, nmos, and nand2 cells were all netlisted. Beware that the netlister indicates "successful" even if it produces garbage! In the case that the netlister fails, try removing the directory created and redoing the process.

In your Linux terminal window, `cd` into the directory that was created. You'll find quite a few files. The most important are `verilog.inpfiles`, `testfixture.template`, and `testfixture.verilog`. Each cell is netlisted into a different directory under `ihnl`. `verilog.inpfiles` states where they are. Take a look at the netlist and other files. `testfixture.template` is the top level module that instantiates the device under test and invokes the testfixture.verilog. Copy your testfixture.verilog and test vectors from your nand2sim directory to your nand2_run1 directory using a command such as

```
cp ../nand2sim/testfixture.verilog .
cp ../nand2sim/nand2.tv .
```

Back in the Virtuoso Verilog Environment window, you may wish to choose Setup • Record Signals. Click on the "All" button to record signals at all levels of the hierarchy. (This isn't important for the nand with only one level of hierarchy, but will be helpful later.)

Then choose Setup • Simulation. Change the Simulation Log File to indicate simout.tmp –sv. This will print the results in simout.tmp. The –sv flag indicates that the simulator should accept SystemVerilog syntax used in the testfixture.verilog.

Set the Simulator mode to "Batch" and click on the Simulate button. You should get a message that the batch simulation succeeded. This doesn't mean that it is correct, merely that it run. In the terminal window, view the simout.tmp file. It will give some statistics about the compilation, and then should indicate that the 4 tests were completed with 0 errors.

If the simulation fails, the simout.tmp file will have clues about the problems. One common problem is to renetlist after you copied over testfixture.verilog, rewriting your testfixture with a default one that does nothing.

Change the simulator mode to Interactive to rerun with the GUI. Be patient; the GUI takes several seconds to start and gives no sign of life until then. Add the waveforms again and run the simulation. You may need to zoom to fit all the waves. For some reason, SimVision doesn't print the $display message about the simulation succeeding with no errors. You will have to read the simout.tmp file at the command line to verify that the test vectors passed.

If you find any logic errors, correct the schematic and resimulate.

VII. Symbol

Each schematic can have a corresponding symbol to represent the cell in a higher-level schematic. You will need to create a symbol for your 2-input NAND gate. When creating your symbol, it is a good idea to keep everything aligned to the grid, this will make connecting symbols simpler and cleaner when you need it for another cell.

While looking at your nand2 cmos_sch, choose Create • Cellview • From Cellview… Choose from cmos_sch to symbol, and click OK. Cadence will create a generic symbol based on the exports looking something like Figure 2.

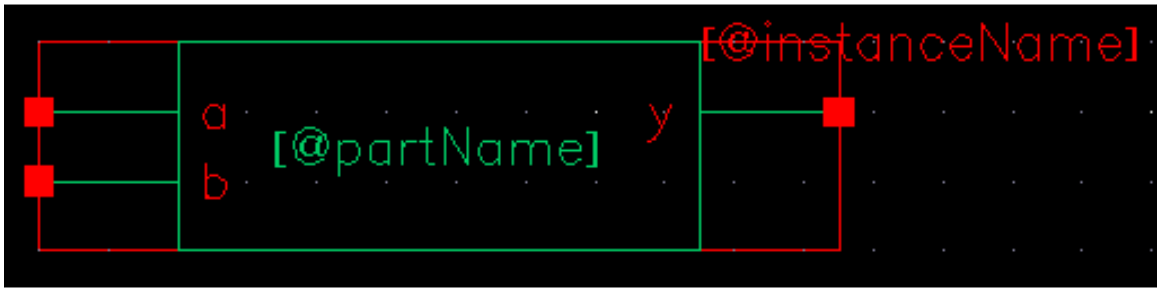


Figure 2: nand2 symbol

A schematic is easier to read when familiar symbols are used instead of generic boxes. Modify the symbol to look like Figure 3. Pay attention to the dimensions of the symbol; the overall design will look more readable if symbols are of consistent sizes.

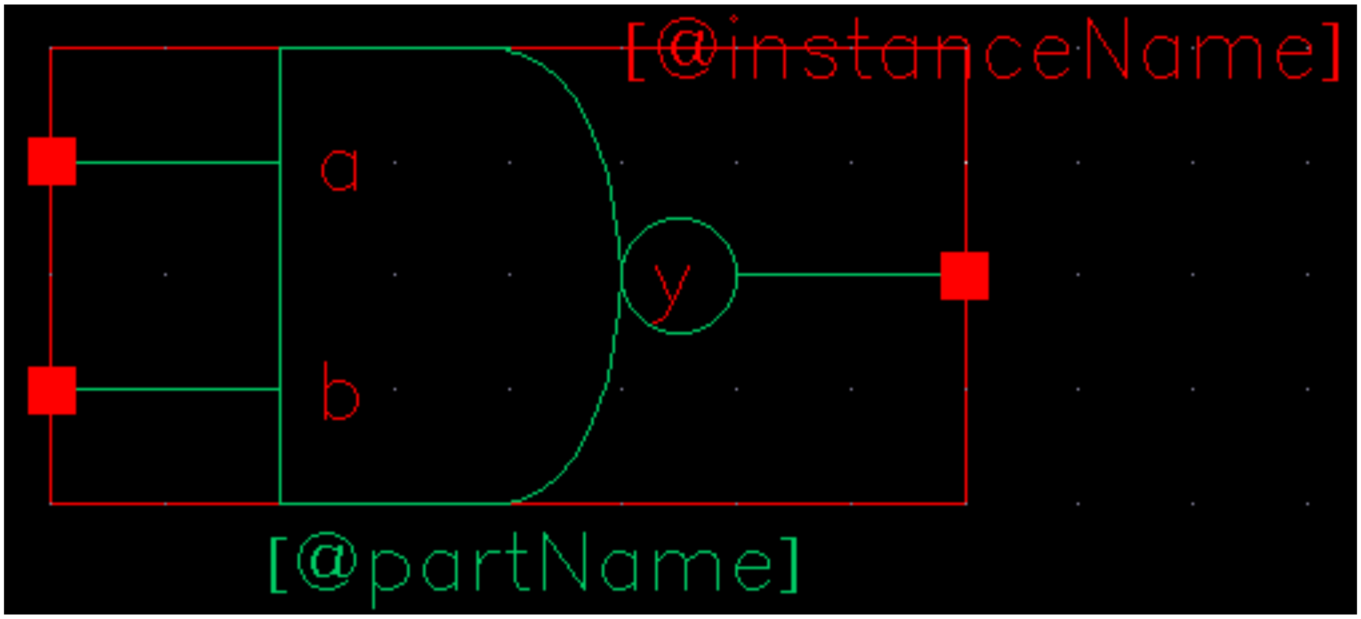
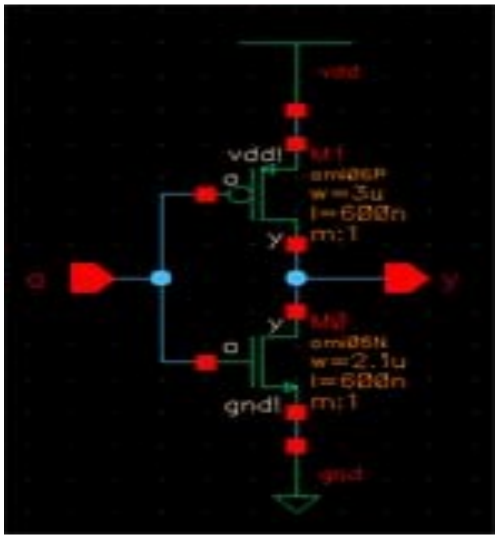


Figure 3: nand2 symbol final version

The green body of the NAND is formed from an open C-shaped polygon, a semicircle, and a small circle. To form the semicircle, choose Create • Shape • Arc. Experiment with the arc drawing tool. Similarly, Create • Shape • Line to make the polygon and Create • Shape • Circle to make the output bubble. Move the lines and terminals around to make it pretty. The Edit • Stretch command may be helpful. Finally, choose Create • Selection Box… and choose Automatic. This creates a red box around the symbol that will define where to click to select the symbol when it appears in another schematic. Choose File • Check and Save when done.

VIII. NOT Gate

Next, design a NOT gate. Name it *inv*. Draw the cmos_sch and the symbol, as shown in Figure 4. Make the pMOS width 10 λ and the nMOS width 7 λ. Be sure to check and save when done.



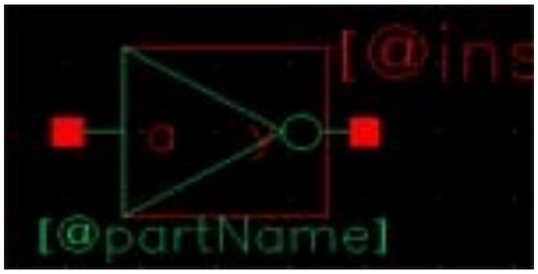


Figure 4: inv cmos_sch and symbol

IX. Hierarchical Design

A CMOS AND gate consists of a NAND gate followed by a NOT gate. The schematic is constructed by connecting the symbols for the two gates that you have already drawn. This is an example of *hierarchical design*, reusing preexisting components to save work.

Create a new schematic called and2. (Note, this time it is schematic rather than cmos_sch because the cell will instantiate other cells rather than transistors.) Add instances from your lab1_xx library of nand2 and inv. Wire the two together and create ports on inputs *a* and *b* and output *y*. Name the wire between the two gates something yb. When you are done, your and2 schematic should look like Figure 5.

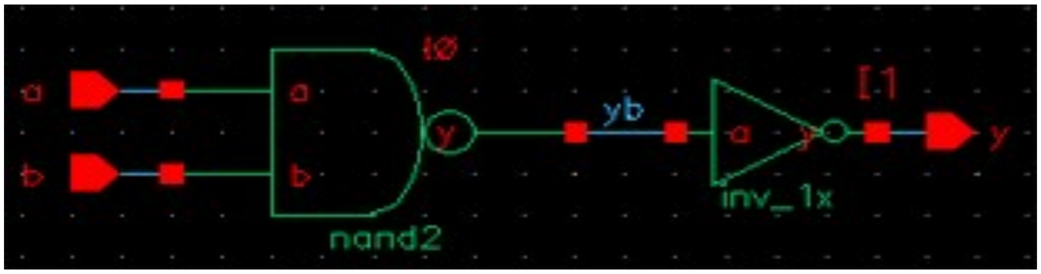


Figure 5: and2 schematic

Simulate your and2 gate to ensure it works. Copy the nand2.tv and testfixture.verilog files to the run directory for your and2 gate and modify them to contain the proper vectors for the and2 function. (If you encounter netlister errors about connectivityLastUpdated, be sure you have checked and saved the schematics of all of the components.)

Make a symbol for the and2. It should be similar to the nand2, but should leave off the output bubble. You may save yourself some time with judicious use of copy and paste or by using the copy command in the Library Manager.

XII. Layout

To start a layout of a 2-input NAND gate, create a new cellview for nand2 called layout. It should be of type layout and always open with Layout GXL.

You may be prompted about the Source View Definiation; click OK to accept the nand2 cmos_sch. You may also get some warnings about layers and contet files.

Your goal is to draw a layout exactly like the one shown in Figure 6. It is important to choose a consistent layout style so that various cells can “snap together” like LEGOs. Neatness and precision are imperative to get a good layout. The heavy dots are on a 10-λ grid and the smaller dots are on a 2-λ grid. In this project’s style, 8 λ-wide power and ground wires run horizontally in metal1 at the top and bottom of the cell, respectively. The spacing between power and ground is 90 λ center to center. nMOS transistors occupy the bottom part of the cell and pMOS transistors occupy the top part. Each cell has well and substrate contacts spaced 8 λ apart under the power and ground wires. Inputs and outputs are placed on metal2 contacts. No other metal2 or metal3 should be used in the cells.

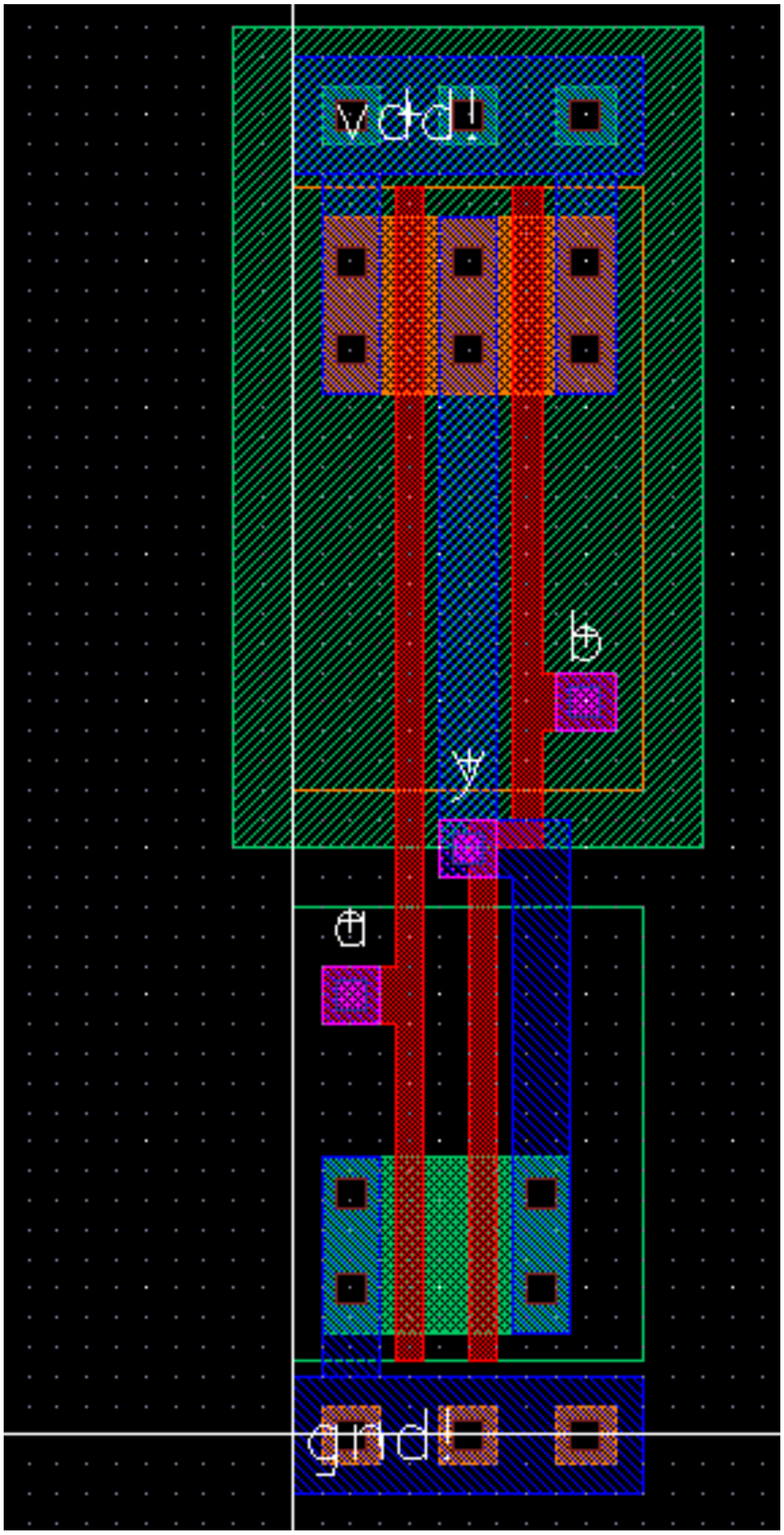


Figure 6: nand2{lay}

The layers pane on the left contains all of the layers you will use and many that you will not. In these labs, you will need nwell, nactive, pactive, nselect, pselect, poly, metal1, metal2, metal3, cc (contact cut), via, and via2. You will not use active, elec, or any of the other layers. In this design process, nactive and pactive stand for n+ diffusion and p+ diffusion, respectively. They must be surrounded by a rectangle of nselect or pselect, respectively; in a cleaner flow, the select layers might be automatically generated.

By default, the layout editor snaps to a 0.5 λ (0.15 μm) grid. Getting off the lambda grid will cause you grief, so start by changing this. Chose Options • Display… and set the X and Y snap spacing to 0.3 (the units are microns). You may wish to set the major grid spacing to 3 microns (10 λ) and the minor grid spacing to 0.3 micron (1 λ).

Start by placing your power and ground busses in metal1. Click on metal1 in the layers pane. Choose Create • Shape • Rectangle. Draw a rectangle from (0, -1.2) to (7.2, 1.2). This is a rectangle 8 λ wide and 24 λ long, centered on the y-axis with the left edge on the x-axis. Draw a second rectangle 90 λ (27 microns) above the first.

Next, draw the n-active and p-active. According to the schematic, the transistors are 12 λ wide, so the active rectangles should be 12 λ tall. Each should start 3 λ away from the ground or power bus. As you are doing this, you may find it helpful to zoom in and out. There are several zoom commands under the Window menu. Learn the keyboard shortcuts for the layout editor to make your life easier. Pan with the arrow keys. Use Tools • Create Ruler to measure distances (you'll get pretty good at doing this by eye after a bit of practice), and Tools • Clear All Rulers to eliminate them when you are done.

Virtuoso has all the usual editing commands such as moving, copying, and deleting. Edit • Stretch is extremely handy for simultaneously moving some objects and stretching others. Edit • Properties is helpful to change the size or layer of a shape. Edit • Merge merges multiple selected rectangles on a given layer into a single convenient polygon.

Next, draw two poly gates. They should be 2 λ wide and extend 2 λ beyond the transistors in each direction. The gate on the right must bend because the spacing between the series nMOS transistors is only 3 λ, while the spacing between the parallel pMOS transistors is 6 λ. You can make the bend by drawing the poly wire as three separate rectangles. Add metal wires to connect the transistors to power, ground, and the output. Then add 2 x 2 λ contacts (cc) between metal and the n-active. In cells like this where the diffusion is wide enough, placing multiple contacts reduces the series resistance and increases reliability in case one contact is malformed in manufacturing.

Recall that substrate and well taps are required to keep the diodes reverse biased. We will place taps under the power and ground wires on 8 λ centers. Draw three 4 x 4 λ squares of p-active under ground, starting 2 λ from the edge. Place 2 x 2 λ contacts on the center of the taps to connect them to metal. Do the same with n-active under the power wire.

Now is a good time to draw the n-well and the select layers. The n-well should extend from 40 λ to 96 λ vertically, and should extend 4 λ beyond the edge of the cell in both directions (-4 to 28 λ). The p-select should extend from 44 to 85 λ vertically and should be as wide as the power line (0 to 24 λ). The n-select should extend from 5 to 36λ vertically and should also be as wide as the power line. Another rectangle of n-select must be drawn exactly covering the power line to surround the n-well taps, and a rectangle of p-select should be drawn covering the ground line to surround the substrate taps.

Later, we will connect cells using vertical metal2 wires and horizontal metal3 wires. Metal2 is drawn on an 8 λ grid (width = 4 λ, spacing = 4 λ). Metal3 is drawn on a 10 λ grid (width = 6 λ, spacing = 4 λ). Figure 7 shows the routing grid superimposed on the cell. The metal2 wires will run in the same columns as the well and substrate contacts, centered 4, 12, 20 λ, etc. right of the cell origin. The metal3 wires will run 0, 10, 20, 30, 50, 60, 70, 80, and 90 λ up from the cell origin. Inputs and outputs should be placed on 4 x 4 l squares of metal2 in the appropriate columns. For example, *a*, *b*, and *y* are all on this grid.

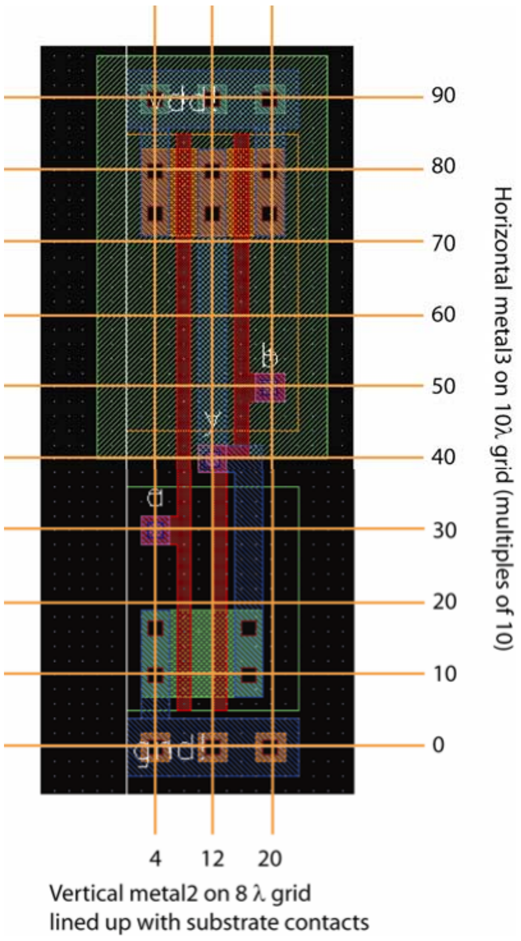

vdd!
b
y
a
gnd!
90
80
70
60
50
40
30
20
10
0
Horizontal metal3 on 10λ grid (multiples of 10)
4
12
20
Vertical metal2 on 8 λ grid
lined up with substrate contacts

Figure 7: Wiring grids and export locations

To create the inputs, you will need a stack going all the way from metal2 down to polysilicon: metal2, via, metal1, cc, and poly. The metal can be 4 x 4 λ, and the contacts/vias 2 x 2 λ, but the poly may need to extend a bit further to reach the main polysilicon lines. The output is already on metal1, so you only must add metal2 and a via.

Finally, place pins to define where the cell connects to other cells at the next level of the hierarchy. Select metal2 in the layer window. Choose Create • Pin… Set the terminal name to a. Choose rectangle mode, weak connectivity, and input I/O type. Turn on Display Pin Name. Then draw a 4 x 4 square of metal2 representing the pin on top of the metal2 already present for input a. Do the same for b. y is also the same, but is an output. Also create large metal1 inputOutput pins called vdd! and gnd! Covering the entire power and ground busses. The! is pronounced "bang" and indicates a global net.

Sanity check your design. Double-check that everything is aligned to a 1 λ grid and that the inputs and outputs are on the wiring grid. Ports should use all lower case letters. Make sure you don't have any 3 λ wide wires or any stray pins or other junk in the layout.

Run the Design Rule Checker by selecting Verify • DRC… Click OK to run with the default settings (flat, don't join nets with same name, divaDRC.rul, UofU_TechLib_ami06 rules library). Look at the Virtuoso window for errors. It is normal to have quite a few the first time you do layout, and all of them must be corrected. Use Verify • Markers • Explain to zoom to the errors and get more information about them. Verify • Markers • Delete All removes all the markers that may be cluttering the screen.

Your design should now resemble Figure 6 and should pass DRC. Save the layout.

XIII. LVS

The next step is to prove that the layout matches the schematic. This is done by extracting the transistors and their connections from the layout, then running a layout-vs.-schematic tool.

In the layout editor, choose Verify • Extract… and run it with the default settings to create an extracted cellview. Look in the Virtuoso window for errors. You may wish to open the new view from the library manager to see what was produced.

In the layout editor, choose Verify • LVS… For the schematic input, select the cmos_sch view of nand2 in your lab1_xx library. For the extracted input, choose the extracted view of the same cell and library. Then click Run to launch the LVS job.

If you are lucky, you will get a window popping up saying that the job succeeded. This doesn't mean that LVS passed, but merely that it ran. Click the Output button to look at the results. Familiarize yourself with the results. You want to see a statement that "The net-lists match." Tracking mismatches is an acquired skill and can be tricky. Start with a careful visual examination of both the layout and schematic to be sure you've drawn what you intended. Often, looking in the LVS directory mentioned in the report will give more information about problems. This may involve a tedious effort of reading the netlist file and sketching a diagram of the mismatched components.

XIV. Cell Library Guidelines

Recall that we want our cells to snap together easily. Therefore, all gates should be 90 λ tall from the center of gnd to the center of VDD. The width of the gate depends on the number of inputs. To match our routing grid, a logic gate is typically 8 λ wide for each input or output.

When you draw layouts, follow these guidelines:

- Place the nMOS transistors close to gnd and the pMOS transistors close to power (with poly ending 1 λ away from the wide wire). The largest transistors that can be drawn are 27 λ for nMOS and 37 λ for pMOS.
- Neatness counts. Pack transistors as close as possible to minimize stray capacitance and resistance. Keep wires as short as possible.
- Place inputs and output pins on metal2. The pins should fall on the routing grid. There should never be more than one pin in a column.

XV. Hierarchical Design

To illustrate hierarchical layout, we will build an AND gate using a NAND and an inverter. Figure 8 shows what the completed AND gate should look like.

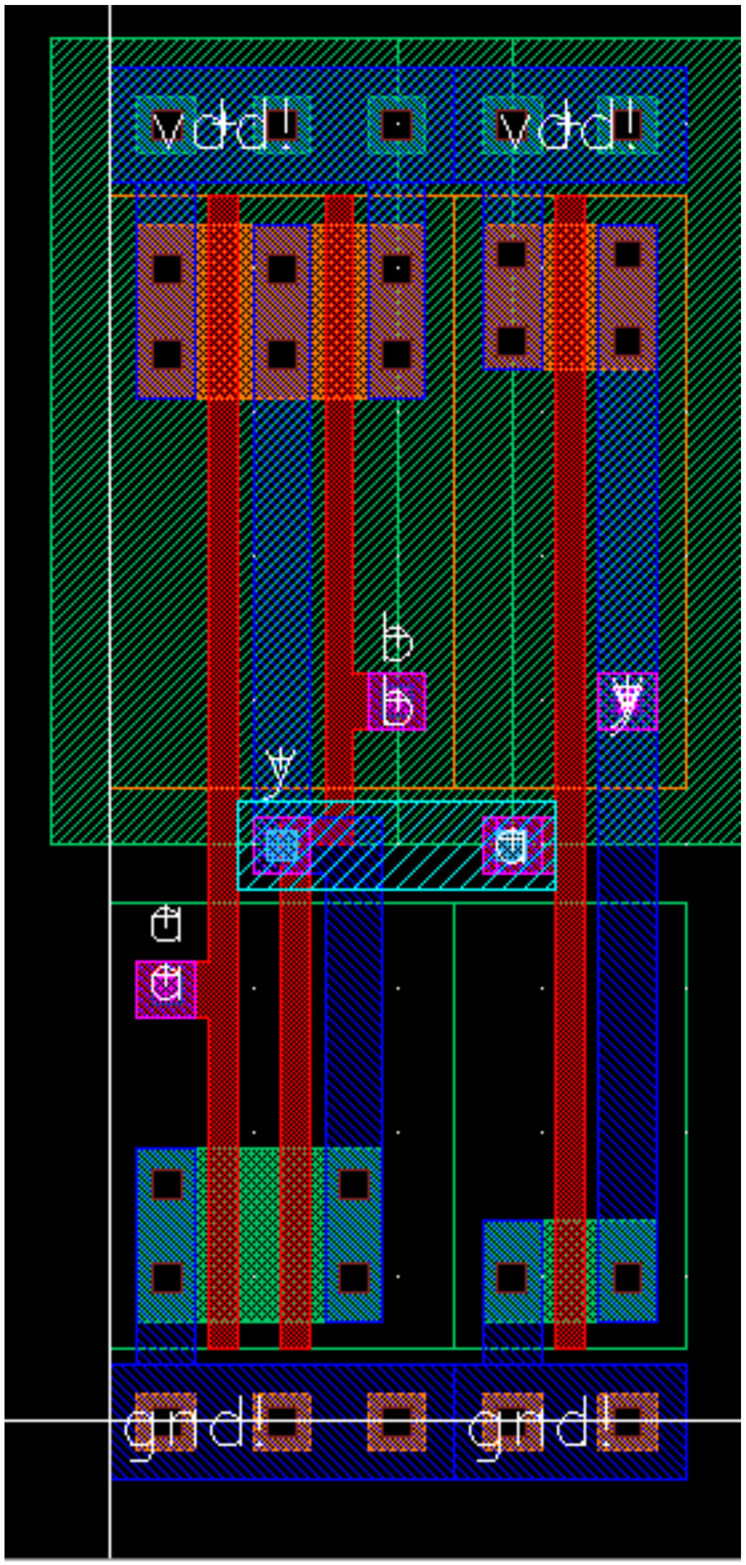


Figure 8: and2 layout

First, draw the inverter layouts. Be sure that the nMOS and pMOS widths match the schematic. Check that it passes DRC and LVS.

Next, create a new cell layout for the and2. Choose Create • Instance and place the nand2 and inv layouts. They show up as red bounding boxes. Use Options • Display to look inside the box. One option is to set the Stop Display Level to a big number (such as 10) to look inside cells. But a more convenient option for us is to click on the Instance Pins box to show just the pins.

Move the cells so that their power and ground busses abut. Place a 2 x 2 λ via2 on the y pin of the nand and the a pin of the inv. Then draw a metal3 wire connecting the two gates. Metal3 is thicker and has sloppier tolerances, so it must be 6 λ wide and extend 2 λ beyond each via2. At this point, your design should resemble Figure 9.

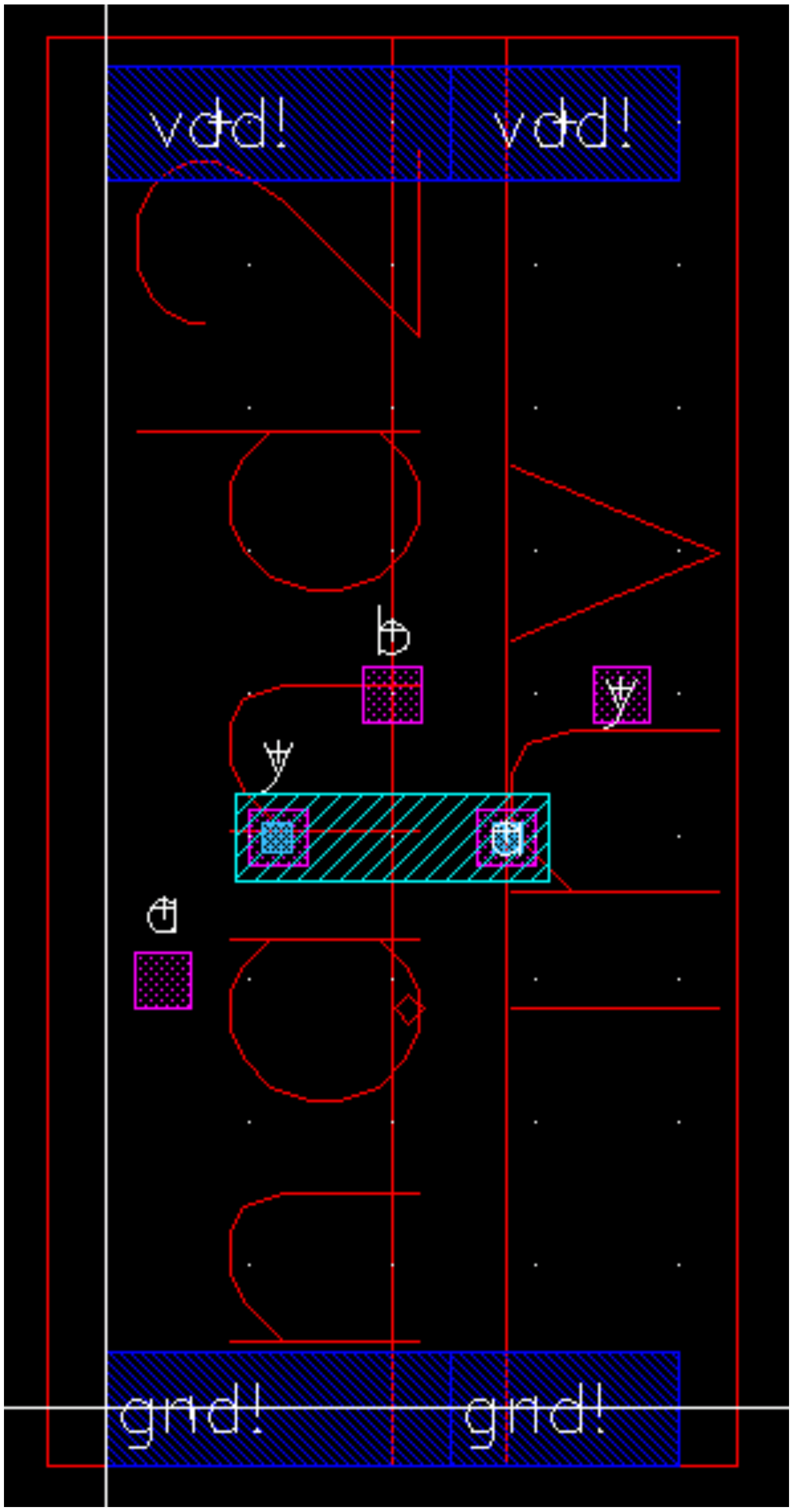


Figure 9: and2 hierarchical view

Now you will need new pins at this level of hierarchy for a, b, y (from the inverter output), vdd!, and gnd!. The inputs and outputs should be 4 x 4 metal2 squares on top of the pins from the lower level of hierarchy, while the power and ground should be metal pins covering the entire power/ground busses

Check your design with DRC and LVS and fix any errors.

XVI. For More Practice

Design schematics, symbols, and layouts for the following two gates. Simulate the schematics to prove that they work correctly. You will have to create your own System Verilog testbenches and test vector files. Check DRC and LVS.

- 2-input NOR gate: nor2 (use transistor widths of 8 λ for the nMOS and 16 λ for the pMOS)
- 2-input OR gate: or2 (using a nor2 and an inv)

XVII. What To Turn In

Please provide a hard copy of each of the following items. You may wish to use the Windows Snipping Tool or the Mac Grab program to take screen shots of your designs.

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for the future.
2. A printout of your nor2 schematic.
3. A printout of your nor2 layout.
4. A printout of your or2 schematic.
5. A printout of your or2 layout.
6. A printout of your or2 test vectors. Does the schematic pass simulation?
7. Does the or2 layout pass DRC? LVS?

Credits

This series of labs was originally developed by David Diaz and David Money Harris in 2001 using Electric. It was revised in 2007 by Nathaniel Pinckney. In 2010, the labs were ported to Cadence and Synopsys by William Koven and Ivan Sergeev. In 2015, the labs were updated by Avi Thaker, Austin Fikes and Apoorva Sharma to support Cadence IC615 and the NCSU CDK 1.6.0 beta.