55:131 Introduction to VLSI Design
Project 1: Cell Design and Verification
Due Date: Friday, September 27, 2013

This is the first of three projects developed for our class. This project teaches you the basics of how to use the Electric computer-aided design (CAD) tool to design, simulate, and verify schematics and layout of logic gates. It also serves as a tutorial to quickly get up to speed with Electric.

I. What is Electric?

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.

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 gate-level netlist that perform this function, place pre-designed cells on the chip, and route the wires to connect the connect the cells. The vast majority of commercial designs are synthesized today because synthesis takes less engineering time.

Custom design offers higher performance, lower power, and smaller chip size than standard cell design. Many aspects of custom design are used when a new cell is needed in a standard cell library. Custom design is useful in this project because it provides more insight into how chips are built.

The first step in custom design 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.

II. Installing Electric

Electric is a Java program that can be freely downloaded from staticfreesoft.com. You can copy it to your own computer and run it locally if you have Java installed.

III. Getting Started

To start Electric, simply double click on electricBinary-9.03.jar. All of your designs are stored in a library. Create a library by selecting File • New Library. Name the library gates_xx, where xx are your initials. Save the library using File • Save. Put it in your personal directory. The library will be saved in jelib (Java Electric Library) format. Electric sometimes has trouble with file or path names that have spaces. On Windows, it is also more stable to save your library on a disk with a letter name (e.g. H:\) rather than on a network drive (e.g. \\dmatthew\home).

Like all complex CAD tools, Electric occasionally crashes. Be sure to save often so that you do not lose too much work. While Electric rarely corrupts libraries, it is wise to periodically create backups of your libraries just in case.

For large projects Electric can consume lots of memory, so it is a good idea to increase Electric's memory size. Click File • Preferences • General • increase "Maximum memory" to 128 or higher. Click OK to save.

IV. Schematic Entry

Our first step is to create a schematic for an inverting multiplexer (mux) gate. Each gate or larger component is called a cell. Cells have multiple views. For example, your mux schematic will be in the mux{sch} view, while your layout will eventually go in the mux{lay} view.

Choose Cell • New Cell. Name the cell mux and select the schematic view. The editor window should have a heading gates_xx:mux{sch} and the component panel on the left will switch to schematic components.

Your goal is to draw a gate like the one shown in Figure 1. Choose Windows • Toggle Grid to turn on a grid to help you align objects. Left-click on an nMOS transistor symbol in the components menu on the left side of the screen. Left-click on your schematic window to drop the transistor into your layout. Repeat until you have five nMOS transistors, five pMOS transistors, the circular power symbol, and the triangular ground symbol arranged on the page. You may move the objects around by left-clicking and dragging. The transistors default to a width/length value of 2/2. Double-click on all of the nMOS transistors and change their widths to 4. Double-click on all of the pMOS transistors and change their widths to 8.

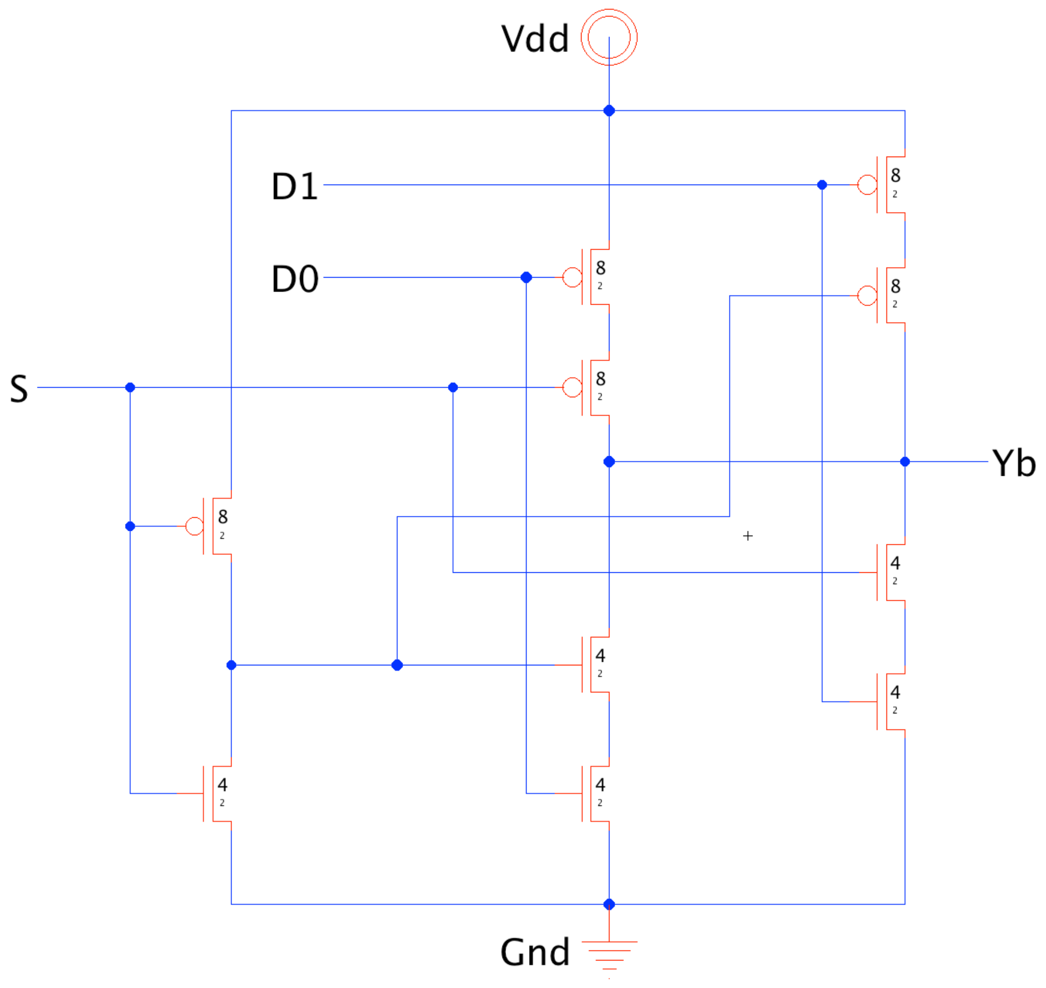

Figure 1: mux{sch}

Now, make the connections. Left-click on a port such as the gate, source, or drain of a transistor. Right-click on another port to create a wire connecting the ports. Continue until all the blue wiring is completed.

Finally, you will need to provide exports defining inputs and outputs of the cell. Left click on the end of a wire where you need to create the export for input S. You should see a small square box highlighted at the end of the wire. If the entire wire is highlighted, you clicked on the middle of the wire instead of the end, so try again. You may need to click "see" in the arc properties to see the box. Once you have selected the end of the wire, choose Export • Create Export. Give your export the name S. Give it the characteristic Input. Repeat with the other inputs. Export Yb as an Output.

Use File • Save (Ctrl-s) to save your library. Get into the habit of saving often because Electric has been known to crash. Also, learn the keyboard shortcuts for the commands you use frequently.

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 design performs the correct function. Typically, logic verification is done first on the RTL to check that the specification is correct. A testbench written in Verilog, SystemVerilog or VHDL automates the process of applying and checking all of the test cases.

In a similar manner, test vectors are 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 mux gate to become familiar with the gate's function and understanding a testbench. In this tutorial, the RTL and testbench are written in Verilog, which is a hardware description language.

This project uses the Questasim simulator, a commercial digital simulator from Mentor Graphics. It is available at a modest fee to universities, and a free starter version limited to 10,000 lines can be downloaded with the Xilinx WebPACK.

Questasim is installed on the Linux-based computers in the Elder and Herring labs. After logging in, you'll need to source the Mentor setup file by typing the line below at the Linux prompt (assuming you are using the *bash* shell):

```
$source /usr/css/etc/mentor_setup.sh
```

To avoid having to source this file every time you start a session, you can add the source command line in your .bashrc file (which is in your home directory):

- open your .bashrc file with a text editor
- add the following two lines at the end of the file (the first line is only comment):

```
#source of setup file for Mentor Tools
source /usr/css/etc/mentor_setup.sh
```

Make sure you type the file's path correctly! This way, in the future, when you start a session you will not need to type the "source" command.

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

Look in the project directory for mux.v and mux_tb.v, and open them in a text editor. mux.v is the Verilog RTL file and mux_tb.v is the testbench. The testbench applies stimulus to pins of the mux module. After each application it compares the output of the mux module to the expected output, and prints an error if they do not match. Note that testbench only tests one test case (application of a stimulus and check of the output). You need to add test cases to exhaustively test the gate.

Invoke Questasim on one of the Linux machines by typing vsim at the Linux prompt. The first time you start a new project, you should create a work library by typing "vlib work" at the Questasim prompt. Compile the RTL by typing "vlog mux.v". Compile the testbench by typing "vlog mux_tb.v". Load the design by typing "vsim –novopt mux_tb". Note how the testbench instantiates the mux gate. The testbench is the "top-level" to be simulated, so it is the module which is loaded into Questasim. Also note that the simulation optimizer in newer versions of Questasim can be problematic, so the "-novopt" switch is used here. Note that these steps (library creation, compilation, and loading the simulation), like most operations, can also be done through the Questasim menus or with scripts.

Once the design is loaded, the simulator will start and some windows will open so that you can explore your design. Select View • Wave to open a waveform viewer; some of the other debug windows may be helpful too. On the left of the screen is the Workspace pane, which normally includes a list of files in the project and compiled modules in the libraries. During simulation it also includes a "sim" tab containing a hierarchy of simulated modules. Immediately to the right of the Workspace pane is the Objects pane, which displays a list of signals for the module selected in the Workspace pane. Select all of the signals in the top-level module and drag them into the waveform viewer so that you can watch them during simulation. Alternatively, the wave window and signals can be added by typing "add wave –r *". The "-r" switch means recursively (so it will include signals in the mux gate in this case, as well as the testbench) and "*" means "all".

Type run -all to run through all of the tests. You should see that there were no errors in the transcript pane. Learn to scroll in and out in the wave window. Though it doesn't matter here, figure out how to display signals in binary, decimal, and hexadecimal form.

It is best to use a self-checking testbench that automatically applies the stimulus and checks the results. Look through the RTL and testbench files and understand how they work. Though very primitive in this case, they contain a few basic features which you'll use in more sophisticated testbenches. If you are unfamiliar with Verilog or want a review, please refer to the Appendix of the CMOS VLSI Design textbook, checkout the engineering library, or stop by the instructor's office during office hours.

Make a habit of looking at the messages in the transcript window and learning what is normal. Warnings and errors should be taken seriously; they usually indicate real problems that need to be addressed.

VI. Schematic Simulation

Next, you will verify your schematic from Electric by generating a Verilog deck, compiling it, and testing it with the test bench. While viewing your schematic in Electric, click on Tool • Simulation (Verilog) • Write Verilog Deck… and save the resulting deck as mux_schematic. Open up the deck and copy/paste the text of the mux module into the mux.v Verilog file. You can comment out the RTL version of the mux by enclosing it within /* and */. You may need to reorder the pins or rename the instantiated module in testbench to match the generated module. Save your new module.

Repeat the simulation steps above to test the Verilog deck from the schematic. Rerun the simulation and check that it works. If you find any errors, correct the schematic and resimulate.

VII. Icon

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

While looking at your mux{sch}, choose View • Make Icon View. Electric will create a generic icon based on the exports looking something like Figure 2. It will drop the icon in the schematic for handy reference; drag the icon away from the transistors so it leaves the schematic readable.

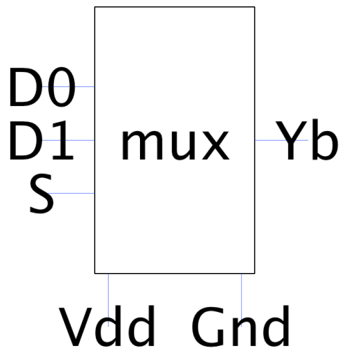

Figure 2: mux{ic} from Make Icon

The Make Icon command is handy because it creates ports and also places a cell center (a + symbol in the middle of the cell). The cell center is important for Electric to keep icons aligned to a grid and prevent nasty messes later on. A schematic is easier to read when familiar icons are used instead of generic boxes. Modify the icon to look something like Figure 3.

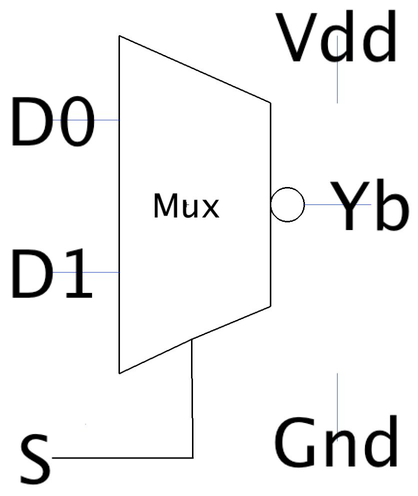

Figure 3: mux{ic} final version

Click on the icon and choose Cell • Down Hierarchy • Down Hierarchy to drop in to the icon. The technology will automatically change to artwork so a palette will appear on the left with various shapes. Click on the generic box and delete it but leave the input and output lines. Turn on the grid. The body of the mux in figure 3 is formed from solid thick arcs and a small circle. Place a circle and adjust its size to 1x1 and position (you may need to use ½ grid positioning).

Solid arcs can be used to form the mux gate body shown in Figure 3. Choose Edit • Modes • Edit • Click/Zoom/Wire when you are done to return to the normal cursor. If your shape is incorrect, delete it, drop another solid arc, and try again.

Electric is finicky about moving the lines with inputs or outputs. If you left-click and drag to select the line along with the input, everything moves as expected. If you try to move only the export name, it won't move as you might expect. Therefore, make a habit of moving both the line and export simultaneously when editing icons. Note that the line is just an open-polygon and can be shortened if desired by entering Outline Edit mode.

Use the Text item in the artwork palette to place a label in the center of the icon. Double-click the label to change text to "mux."

VIII. Electric Manual

If you need more information at any point, bring up the user manual by selecting Help • User's Manual… Now that you know the basics, you will benefit from skimming through the manual to learn the subtleties and find out the best way to do things. In particular, **look at Sections 1.6, 1.8, and 1.9 and Section 2**. Make sure that you **understand nodes, arcs, and ports**; that will help you understand why Electric does some of the seemingly odd things that it does.

IX. Manufacturing Processes

Before starting layout, you must choose the manufacturing technology in which your chip will be built. In particular, Electric needs to know the feature size and design rules for your technology. This tutorial will target the AMI 0.5 µm process using the MOSIS scalable CMOS submicron design rules with $\lambda = 0.3$ µm. We choose this process because MOSIS subsidizes fabrication costs for educational chips on this process. The submicron scalable design rules allow the design to be ported to many other processes without layout changes, and the rules have fewer idiosyncrasies than the deep submicron rules. Note that the minimum drawn transistor length is $2\ \lambda = 0.6$ µm. MOSIS automatically scales polysilicon gates down by 0.1 µm before the chip is fabricated, so the actual transistor length is 0.5 µm. The scalable SCMOS submicron rules are illustrated on the inside back cover of the textbook. All dimensions are given in λ.

To set the technology for a library, select File • Preferences • Technology. Click on the technology item on the left. Change the Startup technology and Layout technology to use for Schematics to mocmos. Set the number of metal layers to 3 and the rules to Submicron. Check Alternate Active and Poly contact rules, but not Second Polysilicon Layer, Disallow Stacked Vias, or Analog. Then click the Scale menu. Choose the mocmos technology and set the Technology scale to 300 nm (0.3 microns).

By default, Electric creates transistors with horizontal polysilicon. We will use transistors with vertical polysilicon. To avoid having to rotate all our transistors, choose File • Preferences. In the Technology item of the Technology folder, check the “Rotate layout transistors in menu” box. Electric also creates 3 λ wide metal wires by default, while we prefer 4 λ wires. In the Arcs item of the General folder, select Technology of mocmos and Arc Type of Metal-1. Set the default width to 4. Do the same for Metal-2. Set the default width of Metal-3 to 6 λ.

X. Layout

To create a layout of the inverting mux gate, choose Cell • New Cell to bring up the New Cell dialog. Enter mux as the name and layout as the view. You may also wish to choose Cell • Edit Cell to open the mux{sch}, then position the windows side-by-side for easy reference.

Your goal is to draw a layout exactly like the one shown in Figure 4. 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. 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 100 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 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.

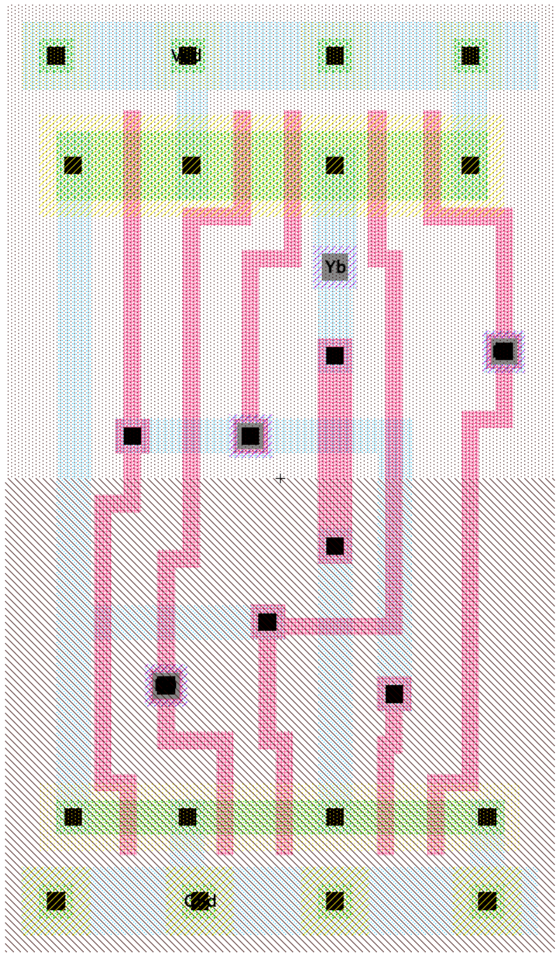

Figure 4: mux{lay}

Start by drawing your nMOS transistors. Recall that an nMOS transistor is formed when polysilicon crosses N-diffusion. N-diffusion is represented in Electric as green diffusion surrounded by a dotted yellow N-select layer all within a hashed black P-well background. This set of layers is conveniently provided as a 3-terminal transistor node in Electric. Move the mouse to the components menu on the left side of the screen. As you move the mouse over various objects, the node name will appear on the status line next to the word NODE near the bottom left corner of the screen. Left click on the nMOS transistor, and click again in the layout window to drop the transistor in place. There are four nMOS transistors in the mux gate (one of which is part of an inverter for the "S" input. Change their width to 4.

We need five transistors, so copy and paste the transistor you have drawn or use the Edit • Duplicate command. Don't put the transistors too close to each other or anything else, until they are connected appropriately. At that point they can be moved alongside each other as necessary but keep them far enough apart to leave room for the contacts. You will probably find it helpful to turn on the grid using the Window • Toggle Grid command. The grid defaults to small dots every λ and large dots every 10 λ. By default, objects snap to a 1- λ grid. This can be changed by using Edit • Modes • Movement menu. You can also move objects around with the arrow keys on the keyboard. Pressing the h or f keys cause the arrows to move objects by a half or full λ, respectively. You will avoid messy problems by keeping your layout on a λ grid.

Next we will create the contacts from the N-diffusion to metal1. Diffusion is also referred to as active area or just active. Drop a square of Metal-1-N-Active-Contact in the layout window and double-click to change the properties to an X size of 4 and Y size of 4. You will need contacts for the connections to Yb and the other ends of the nMOS transistors, so duplicate the contact you have drawn. Move the contacts near the nMOS transistors and draw diffusion lines to connect them to the transistors. Then move the contacts even closer; you only need a gap of 1 λ between the metal and polysilicon. Using similar steps, draw four pMOS transistors (L=2, W=8) and create contacts from the P-diffusion to metal1. Use a P-diffusion to metal1 contact size of X = 4 λ and Y = 8 λ. At this point, your layout should look something like Figure 5:

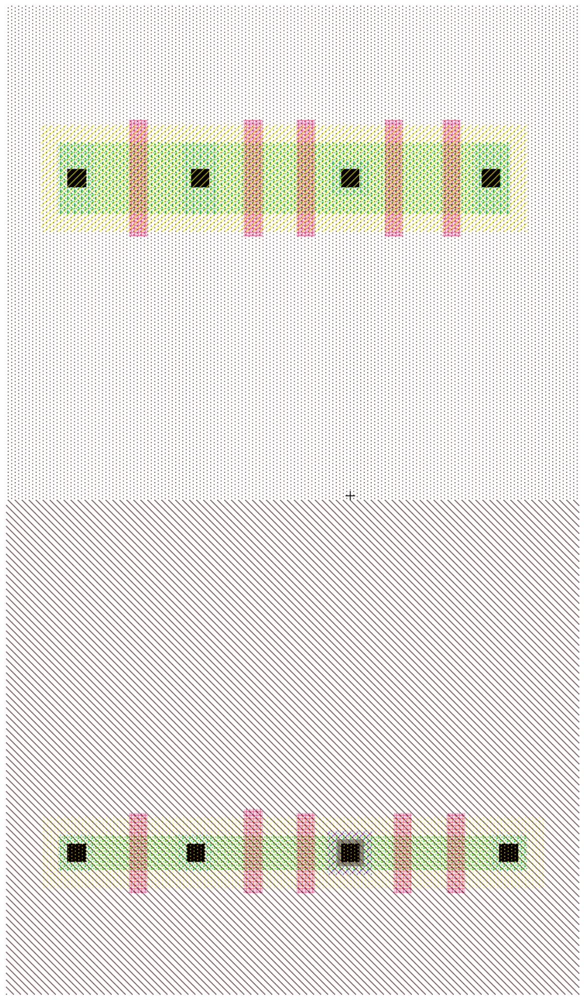

Figure 5: Contacted transistors for mux layout

Run the design rule checker to see if your layout has any errors. Invoke it using Tools • DRC • Check Hierarchically. The results are reported in the Electric Messages window. If there are errors, use the > key to step through each one and highlight it on the layout. Fix the errors and repeat until all errors are corrected. It is recommended that you run DRC often.

Now add metal1 power and ground lines. The lines should be 4 λ wide. You can add a line by selecting a pin such as metal-1-pin from the palette then right-clicking nearby to draw the line. Double-click on the line to bring up the arc properties and change the width to 4. Click on the ground wire and export it as gnd with characteristic ground. Similarly, export power as vdd with characteristic power.

To move the transistors into position near the ground wire, left click and drag the mouse to select all the nMOS transistors and contacts. Release the mouse, then hold the left mouse button down again over the selected stuff to drag it. Move the transistors so the polysilicon ends 1 λ above the ground wire.

Left-click on the 2nd from left most active contact in the n-diffusion. Right-click on the ground line. Electric will create a wire between the contact and ground. If the width comes up at 8 λ, double-click on the wire and change its width to 4 λ. Complete the other connections to power and ground. When you are done, power and ground should extend 2 λ beyond the contents of the cell (excluding wells) on either side so that cells may "snap together" with their contents separated by 4 λ so design rules are satisfied.

Draw wires to connect the polysilicon gates, forming inputs D0, D1, and S, and the metal1 output wire Yb.

You may find it helpful to zoom in and out to view the layout. There are several Zoom commands under the Windows menu. Some have useful keyboard shortcuts.

If several pieces of layout are on top of each other, hold the control key and click several times until the desired one is selected. When you are drawing a wire, if Electric generates the wrong kind of layer (for example, diffusion instead of metal1), click on the palette under the Pure column to choose a layer that Electric should use.

Recall that well contacts are required to keep the diodes between the wells and source/drain diffusion reverse biased. We will place N-well and P-well contacts in each cell under the power and ground wires. Drop 3 Metal1-N-Well-Con contacts on the center of the power wire. Click on each contact, then click nearby on the power wire to form a connection. Repeat with P-well contacts on the ground wire.

We will connect cells using metal2 wires and metal3 wires. Depending on which chip uses the cell, these metal layers could be vertical or horizontal. For this process, metal2 is drawn on an 8λ grid (width = 4 λ, spacing = 4 λ) and metal3 is drawn on a 10λ grid (width = 6 λ, spacing = 4 λ). To allow flexibility, keep vias in your design spaced at least10 λ apart horizontally, and at 15, 25, 35, 45, 55, 65, 75 or 85λ from the cell bottom. Inputs and outputs should be placed on metal1-metal2 contacts called vias, centered on the metal2 columns (in the same column as the substrate contacts). For example, D0, D1, S, and Yb are all on this grid in figure 4.

For the output, create a metal1-metal2-con called a via. Connect it to the metal1 output wire. Then select the via and create an export named Yb of type output. Create vias for the other inputs. Place metal1-polysilicon1-con poly contacts near the vias. Connect the via to the contact and the contact to the polysilicon wires. **It may be easiest to place components some distance apart, wire them together, and move them to overlap as shown in the figure.** Export the vias as D0, D1, and S. Electric is agnostic about the choice of well and substrate; it generates both n- and p-well layers. In our process that has a p-substrate already, the p-well, indicated by black slanting lines, will be ignored. The n-well, indicated by small black dots, will define the well on the chip. Electric only generates enough well to surround the n and p diffusion regions of the chip. It is a good idea to create rectangles of well to entirely cover each cell so that when you abut multiple cells you don't end up with awkward gaps between wells that cause design rule errors. Electric uses pure layer nodes to create rectangles of some layer. Choose from the Components palette Pure • N-Well-Node. Drop the small rectangle of N-well on your layout. Pure layer nodes are hard to select in Electric. This makes them difficult to accidentally select when you are editing something else. If you need to select a hard-to-select object that is not already selected, click on the toolbar and choose the arrow surrounded by blue, named "Toggle Special Select." When it is clicked, you can now select hard-to-select objects, but not normal objects. Click the arrow again to return to normal selection mode.

Double-click on the N-Well pure layer node and change the width to 65 and the height to 56. Drag the layer over the pMOS transistors in your layout. It should extend 2 λ above the power wire and fully cover all the other bits of N-well. Repeat the process with a P-Well of width 65 and height 56 to cover the nMOS transistors, starting 2 λ below the ground wire. Sanity-check your design. Double-check that everything is aligned to a 1 λ grid and exports are correct. Make sure you don't have any 3 λ wide wires or any stray pins or other junk in the layout.

Your design should now resemble Figure 4 and should pass DRC. Save the library.

XI. NCC and ERC

Electric has a powerful layout-versus schematic (LVS) feature called Network Consistency Check (NCC) that compares a layout and schematic to ensure they are topologically identical. While looking at either the layout or the schematic, choose Tools • NCC • Schematic and Layout Views of Cell in Current Window.

If the layout and schematic match export names, transistor topology, and transistor sizes (enabled by File • Preferences • Tools • NCC • Check Transistor Sizes), NCC will report success. If not, NCC will display information about the mismatches. Tracking down the problem is an acquired but essential skill.

Electric also has an electrical rule checker that ensures every well and substrate area has at least one well or substrate contact. Choose Tools • ERC • Check Wells. The message window should report that no errors were found.

XII. Simulation
Verilog decks can be generated from layouts. Generate a deck for your mux{lay} and simulate it in Questasim using the steps similar to those listed earlier. If NCC passed, the simulation should work too. Usually there is no need to simulate the layout, but it is a good idea to simulate a chip's final layout once before manufacturing, if possible, to increase your confidence in the design.

SPICE decks can also be generated from layouts. Generate a deck for your mux{lay} and save it.

XIII. Layout Templates
Recall that we want our cells to snap together easily. Therefore, all gates should be 100 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.

Drawing the power and ground busses, pure layer nodes, and well and substrate contacts are repetitive. Therefore, it is useful to define layout templates that already have these structures, along with some transistors and contacts.

XIV. What To Turn In
Please submit an electronic copy of each of the following items into ICON. Make sure the images are easy to read to avoid grading difficulties. Choose File • Export • PNG (Portable Network Graphics) to save screen shots in the Word-friendly format.

1. Please indicate how many hours you spent on this project. This will not affect your grade, but will be helpful for calibrating the workload for the future.
2. A copy of your mux schematic.
3. A copy of your mux layout.
4. A copy of your mux Verilog deck, including the testbench.
5. A copy of your mux SPICE deck.
6. How many test cases did you add to the testbench? Explain why it exhaustively tests the mux.
7. What is purpose of running DRC? Would it be possible to create and build a design which passes DRC but doesn't work correctly? Explain why or why not.
8. What is the purpose of running logic simulation? Would it be possible to create and build a design which passes logic simulation but doesn't work correctly? Explain why or why not.