NYDesign Workshop on Analog Circuit Design with Open Source Tools

This tutorial is meant to be used with the Google Cloud servers made available for the NYDesign workshop with Efabless. It assumes that you have logged into the virtual machine from a browser window and have the desktop in front of you.

Getting Started

Bring up a terminal window on the desktop.

In your directory "`designs`" is an example of an analog circuit design, "`sample_and_hold`". This circuit is, as the name implies, a sample-and-hold circuit, designed in the SkyWater Open PDK process, for 3.3V (or 5.0V) rail-to-rail operation. It was designed as part of an 8-bit SAR (Successive Approximation Register) ADC (Analog to Digital Converter).

The problem with designing analog circuits is that it is labor-intensive at all levels, and we do not have the time available in the workshop to design any meaningful analog circuit from scratch. So we will learn the basics of each of the tools by doing a simple exercise, and occasionally look at the completed sample & hold as a more realistic analog design.

As discussed in the slide presentation, an analog circuit that has potential for re-use should be put in its own subdirectory, which can then be moved, copied, or linked as needed into a larger project, such as the "`caravel_user_project_analog`" (which itself becomes a component of the "caravan" chip. The "caravel" and "caravan" designs, having full padframes, are necessarily top-level, stand-alone projects).

First, take a look at the structure of "`sample_and_hold`":

```
ls sample_and_hold
```

Note that it has subdirectories for each tool involved in the design process.

- "`xschem`" is for schematic capture
- "`spice`" is for simulations and LVS
- "`mag`" is for layout
- "`gds`", "`lef`", and "`verilog`" are for the final products of the design.

Check out the contents of each directory.

- "`xschem`" has schematics (`.sch`), symbols (`.sym`), and a startup script (`xschemrc`)

- "`spice`" has netlists (`.spice`), a configuration file (`.spiceinit`), among a few other files
- "`mag`" has layout files (`.mag`), a netlist (`.spice`), and a startup script (`.magicrc`).
- "gds" has a single file which is the final layout in GDS format (`.gds`)
- "lef" has a single file which is an abstract view of the final layout in LEF format (`.lef`)
- "verilog" has a simple behavioral model of the circuit which can be used in simulation (`.v`)

Starting from the "designs" directory on the cloud platform, create a new project directory for yourself. Call it "example":

```
mkdir example
cd example
```

Then make directories for the various tools:

```
mkdir xschem
mkdir spice
mkdir mag
```

This is all you need for now. Each of these directories corresponds to a tool, and each tool needs a startup file, which is ideally located in the directory where the tool is run, so that it is read by the application automatically. We'll do this for all three directories, copying the relevant files from the sample_and_hold directory:

```
cd xschem
cp ../../sample_and_hold/xschem/xschemrc .
cd ../spice
cp ../../sample_and_hold/spice/.spiceinit .
cd ../mag
cp ../../sample_and_hold/mag/.magicrc .
```

Now go back to the xschem directory:

```
cd xschem
```

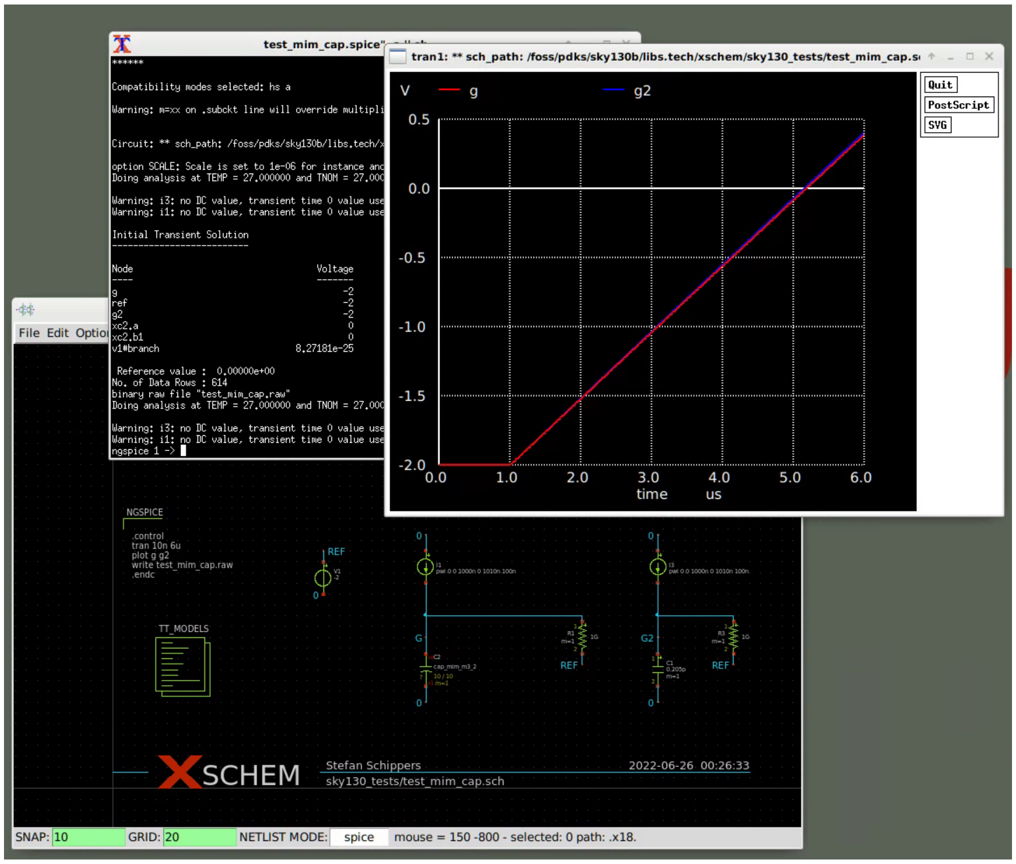
Then run "xschem" from the command line, without any arguments.

```
xschem
```

This will bring up the xschem window with a screen full of example files and testbenches for the SkyWater process. This is a good place to learn how to create netlists and run simulations. Click on one of the testbenches in the list on the left side of the window, to highlight the entry. Then type "`e`" to edit the testbench.

Whichever testbench you chose, the testbench is already set up for simulation and you can simulate in ngspice simply by clicking the "Netlist" button at the top, followed by "Simulate". The xschem startup script tells xschem to put simulaton files in "`../spice`", so that's where you will find the netlist and the test data (`.raw` file).

Type "`exit`" in the ngspice window to exit ngspice and close any plot windows.



```
******

Compatibility modes selected: hs a

Warning: m=xx on .subckt line will override multipli


Circuit: ** sch_path: /foss/pdks/sky130b/libs.tech/x

option SCALE: Scale is set to 1e-06 for instance anc
Doing analysis at TEMP = 27.000000 and TNOM = 27.000

Warning: i3: no DC value, transient time 0 value use
Warning: i1: no DC value, transient time 0 value use

Initial Transient Solution
--------------------------

Node                                   Voltage
----                                   -------
g                                           -2
ref                                         -2
g2                                          -2
xc2.a                                        0
xc2.b1                                       0
v1#branch                          8.27181e-25

 Reference value :  0.00000e+00
No. of Data Rows : 614
binary raw file "test_mim_cap.raw"
Doing analysis at TEMP = 27.000000 and TNOM = 27.000

Warning: i3: no DC value, transient time 0 value use
Warning: i1: no DC value, transient time 0 value use
ngspice 1 ->
```

```
.control
tran 10n 6u
plot g g2
write test_mim_cap.raw
.endc
```

Schematic Entry

Now we'll learn the basics of drawing a schematic with a trivially simple example. In xschem, select

File->New Schematic

It's very easy to forget that you're working in an unnamed schematic ("`untitled.sch`"), so just go ahead and choose

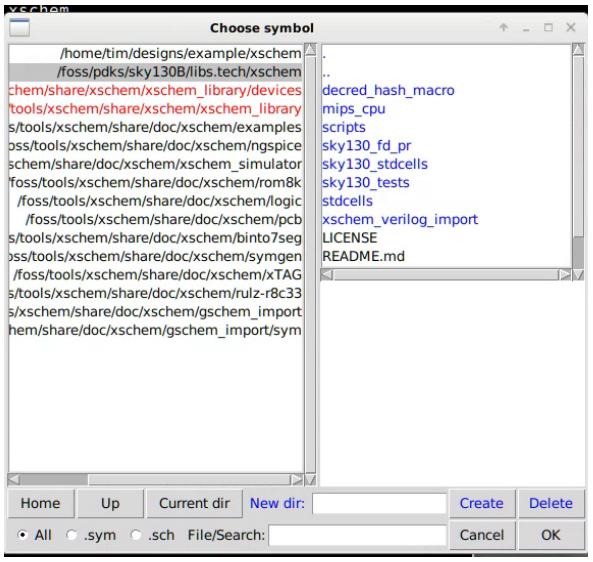
File->Save As

and in the pop-up window, replace "`untitled.sch`" with "`example.sch`", and click "OK".

I'm going to make a simple inverter, to exercise all the basic xschem methods.

To place components, use the "Insert" key on your keyboard. In the pop-up window, the left side contains libraries that xschem knows about, and the right side contains symbols that you can use from a selected library. There are mainly three libraries that you'll need to access: The local directory where you're working, the PDK device library, and the built-in xschem library.

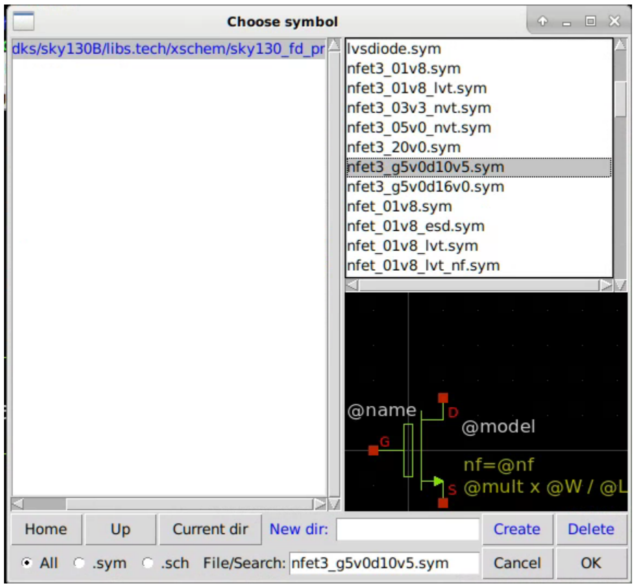
Select the PDK device library. It's hierarchical, so on the right side you need to select a sub-library. For low-level SPICE components like transistors, resistors, and capacitors, select "`sky130_fd_pr`".



```
/home/tim/designs/example/xschem
/foss/pdks/sky130B/libs.tech/xschem
chem/share/xschem/xschem_library/devices
tools/xschem/share/xschem/xschem_library
s/tools/xschem/share/doc/xschem/examples
oss/tools/xschem/share/doc/xschem/ngspice
schem/share/doc/xschem/xschem_simulator
foss/tools/xschem/share/doc/xschem/rom8k
/foss/tools/xschem/share/doc/xschem/logic
/foss/tools/xschem/share/doc/xschem/pcb
s/tools/xschem/share/doc/xschem/binto7seg
oss/tools/xschem/share/doc/xschem/symgen
/foss/tools/xschem/share/doc/xschem/xTAG
s/tools/xschem/share/doc/xschem/rulz-r8c33
s/xschem/share/doc/xschem/gschem_import
hem/share/doc/xschem/gschem_import/sym
```

```
.
..
decred_hash_macro
mips_cpu
scripts
sky130_fd_pr
sky130_stdcells
sky130_tests
stdcells
xschem_verilog_import
LICENSE
README.md
```

Now select a thick-oxide N-type MOSFET, which is "`nfet_g5v0d10v5.sym`". A picture of the device will appear in the pop-up window. Click "OK" to accept.



This takes you back to the schematic window, where an image of the device follows the pointer around until you click the left mouse button to place it. Do that now. Click anywhere in the window to un-select the device.

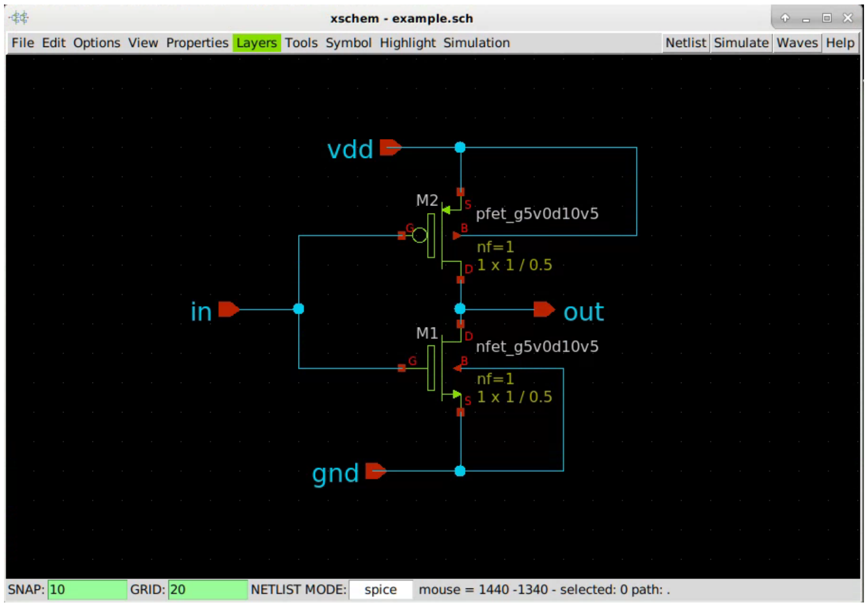
Use the mouse middle button and scroll wheel to pan around the screen and zoom, respectively.

Do the same procedure to grab and place an instance of the corresponding pFET device "`pfet_g5v0d10v5.sym`". Place it above the nFET.

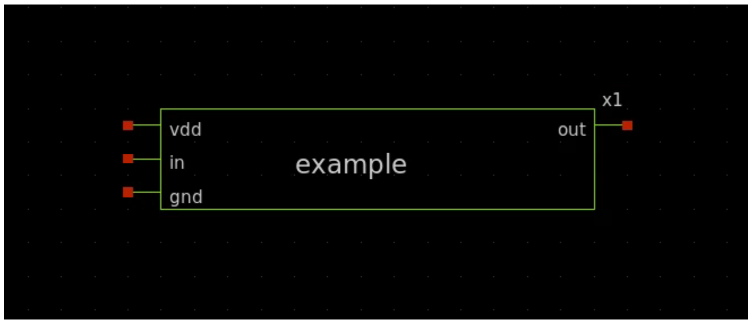
The next important method in xschem is wiring, which you can do with the "w" key. If the virtual machine has problems with finding the pointer position corresponding to a keystroke, then select
"View->Show Toolbar" and click on the toolbar wire icon followed by the left mouse button to start a wire. Create wires connecting the gates and drains of the FETs together.

Now we need pins, which come from the built-in library "`xschem_library`". This has only one subdirectory "`devices`", so select that, then select and place, in turn, the "`ipin.sym`" and "`opin.sym`" symbols. Place these at input and output, respectively, and place an "`ipin`" for power and ground. Never use global signals for power and ground on a schematic for a chip design.

Edit labels by clicking on the pins' "`xxx`" label, then typing "`q`" to get the pop-up window of properties.


xschem - example.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
vdd
M2
pfet_g5v0d10v5
nf=1
1 x 1 / 0.5
in
out
M1
nfet_g5v0d10v5
nf=1
1 x 1 / 0.5
gnd
SNAP: 10 GRID: 20 NETLIST MODE: spice mouse = 1440 -1340 - selected: 0 path: .

When done, you'll need to make a symbol from the schematic, which can be done automatically using the menu item "Symbol->Make Symbol From Schematic".



Do not ever use the design schematic for simulation. For simulation, create a testbench schematic and instantiate the symbol for the design schematic inside it. With what you know already, you can create a new schematic, call it "`example_tb.sch`", place the symbol for "`example.sch`" inside it, and wire it up with pins. I will leave it as an exercise to finish the wiring and run a transient simulation and plot; all that is similar to the process examples in xschem.

The components that you will need for a testbench all come from the xschem library of basic components: SPICE voltage sources ("`vsource`"), ideal resistors and capacitors for loading an output, and (optionally) pins. For simulation you will need additional lines in the netlist, which can come from "`code`" or "`code-shown`" symbols. Both do the same thing; the "`code-shown`" just displays the text contents. By convention of the xschem examples, the "`code`" block has the "`.lib`" statement which tells ngspice to load all of the process device models, and the "`code-shown`" block has the simulation analysis, plot, and output commands.

While the design schematic (as opposed to the testbench schematic) should not be exported to a netlist for simulation, it does need to be exported to a netlist for generating layout. For that, you want to make sure that the layout captures the pins of the schematic, so select the menu item "Simulation->LVS netlist: Top level is a .subckt" to make sure that the whole netlist has a subcircuit definition for the design schematic. Then click on the "Netlist" button. If everything is wired correctly, then the netlist will be generated in the "`example/spice`" directory as "`example.spice`".

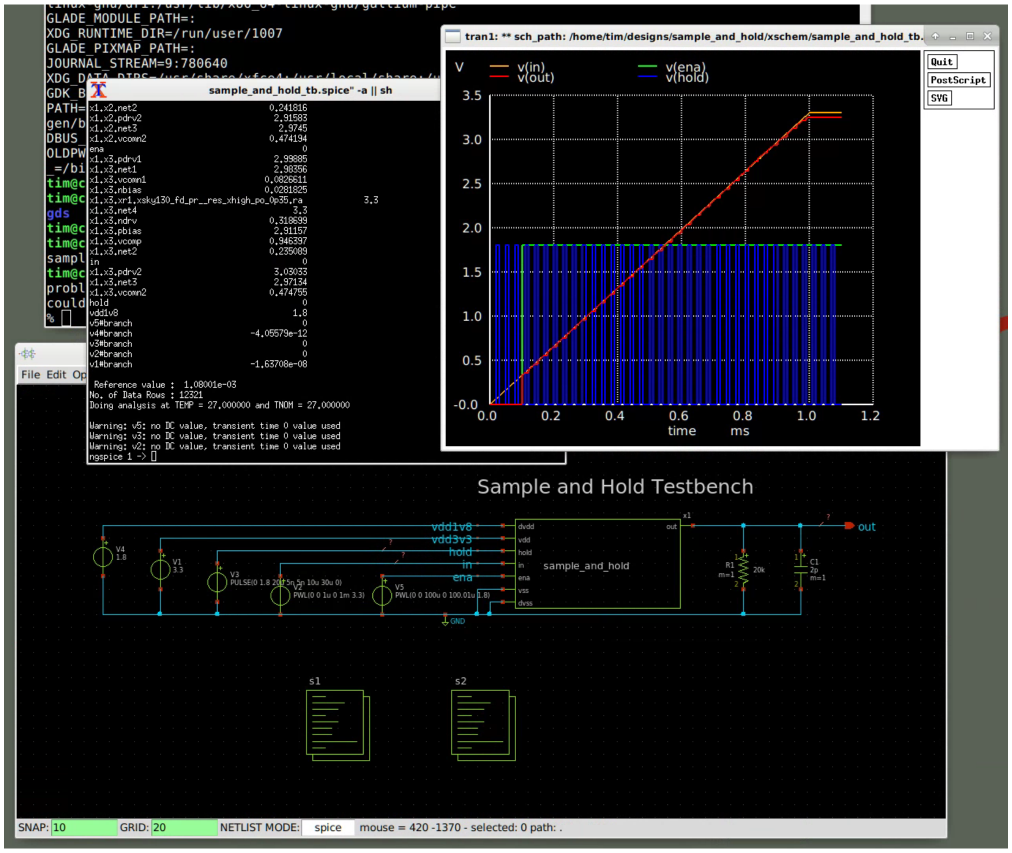
Simulation

Before moving on to layout, take a look at the sample & hold circuit and simulation.

```
cd ../../sample_and_hold/xschem
xschem sample_and_hold_tb.sch
```

This circuit has hierarchy to it; select a symbol and use the "`e`" key to descend into its schematic; use "`Ctrl-e`" to pop back up to the parent cell. The sample_and_hold schematic itself is made of a rail-to-rail source follower amplifier to buffer an analog input, followed by a switch and capacitor to hold a voltage value, and a second rail-to-rail source follower amplifier to keep the holding capacitor isolated. The amplifier uses multiple input stages with different transistors to cover the whole voltage range, including a zero-voltage threshold nFET device available in this process. The switch is a "balanced" design that compensates for the charge sharing between input and output as the switch closes. Note that the switch uses high voltage transistors, but the digital input "hold" is assumed to be in the digital 1.8V domain, so the design uses a level-shifter cell from the "`hvl`" standard cell library to boost the 1.8V signal to 3.3V.

Return to the testbench top level schematic and run the testbench simulation by clicking on "Netlist" and "Simulate". The testbench applies a voltage ramp to the input while periodically applying "hold". The "enable" signal is held low at the beginning. Note in the output plot how the output voltage remains fixed when the "hold" signal is high, and tracks the input when the "hold" signal is low.



sample_and_hold_tb.spice" -a || sh
tran1: ** sch_path: /home/tim/designs/sample_and_hold/xschem/sample_and_hold_tb.
v(in)
v(out)
v(ena)
v(hold)
time
ms
Quit
PostScript
SVG
Sample and Hold Testbench
sample_and_hold
SNAP: 10
GRID: 20
NETLIST MODE: spice
mouse = 420 -1370 - selected: 0 path: .

```
GLADE_MODULE_PATH=:
XDG_RUNTIME_DIR=/run/user/1007
GLADE_PIXMAP_PATH=:
JOURNAL_STREAM=9:780640
```

```
x1.x2.net2                                  0.241816
x1.x2.pdrv2                                 2.91583
x1.x2.net3                                  2.9745
x1.x2.vcomn2                                0.474194
ena                                         0
x1.x3.pdrv1                                 2.99885
x1.x3.net1                                  2.98356
x1.x3.vcomn1                                0.0826611
x1.x3.nbias                                 0.0281825
x1.x3.xr1.xsky130_fd_pr__res_xhigh_po_0p35.ra                 3.3
x1.x3.net4                                  3.3
x1.x3.ndrv                                  0.318699
x1.x3.pbias                                 2.91157
x1.x3.vcomp                                 0.946397
x1.x3.net2                                  0.235089
in                                          0
x1.x3.pdrv2                                 3.03033
x1.x3.net3                                  2.97134
x1.x3.vcomn2                                0.474755
hold                                        0
vdd1v8                                      1.8
v5#branch                                   0
v4#branch                                   -4.05579e-12
v3#branch                                   0
v2#branch                                   0
v1#branch                                   -1.63708e-08

 Reference value :  1.08001e-03
No. of Data Rows : 12321
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000

Warning: v5: no DC value, transient time 0 value used
Warning: v3: no DC value, transient time 0 value used
Warning: v2: no DC value, transient time 0 value used
ngspice 1 ->
```

Layout

If you copied the `.magicrc` file into the `mag/` directory, then you can just start magic with

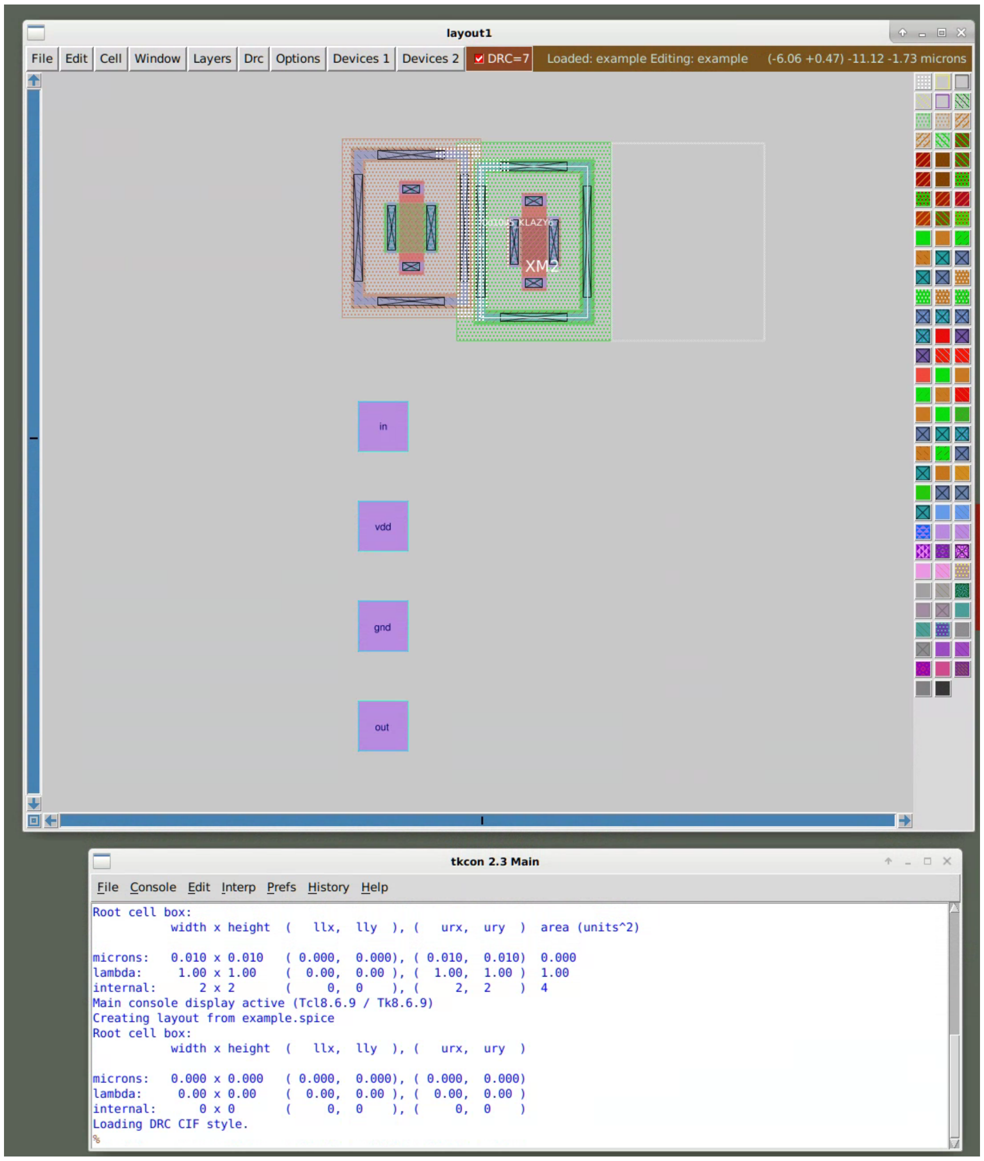
```
cd ../../example/magic
magic -d XR
```

The "`-d XR`" option uses a 2D rendering engine for the graphics, which look much better than the default.

To create an initial layout from a schematic, use the menu option

File->Import SPICE

In the pop-up window, select the example/spice/ directory, select the "`example.spice`" file, and click "Open". Use the "`v`" key in the layout window to view the whole layout after import.



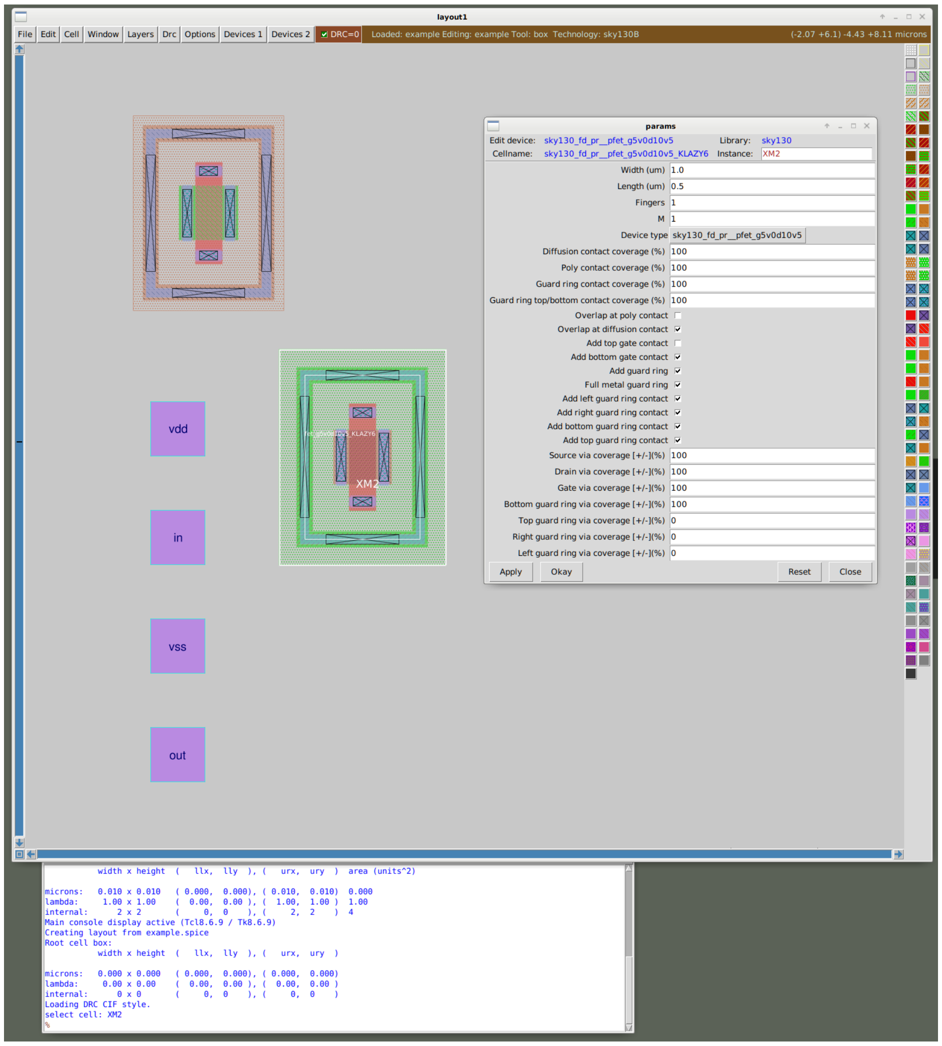
```
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   0.010 x 0.010   ( 0.000,  0.000), ( 0.010,  0.010)  0.000
lambda:     1.00 x 1.00    (  0.00,  0.00 ), (  1.00,  1.00 )  1.00
internal:      2 x 2       (     0,  0    ), (     2,  2    )  4
Main console display active (Tcl8.6.9 / Tk8.6.9)
Creating layout from example.spice
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )

microns:   0.000 x 0.000   ( 0.000,  0.000), ( 0.000,  0.000)
lambda:     0.00 x 0.00    (  0.00,  0.00 ), (  0.00,  0.00 )
internal:      0 x 0       (     0,  0    ), (     0,  0    )
Loading DRC CIF style.
%
```

Magic is not a "silicon compiler", so the resulting layout is not a complete layout---it's not wired up, and the devices are not even placed in any meaningful way. Devices probably have the wrong properties, as there are layout properties that don't exist in schematic form, such as whether to add a guard ring, or where to place contacts. But with the "Import SPICE" method, you do get all of the devices created with the essential parameters like transistor width and length.

If you want to change parameters, select an instance with the "`i`" key while the mouse pointer is over the instance, and if that instance is a parameterized device, then you can type "`Ctrl-p`" to bring up the list of parameters, like using the "`q`" key in xschem. You can do that now to the existing layouts of the transistors and, say, modify the layouts to remove one of the gate contacts so that the gate doesn't have to be contacted on both sides, and add contacts at top and bottom for contacting the device guard rings to the power supplies. Click on "Okay" to confirm the changes and redraw the instance with the new parameters.



Loaded: example Editing: example Tool: box Technology: sky130B   (-2.07 +6.1) -4.43 +8.11 microns

Edit device:	sky130_fd_pr__pfet_g5v0d10v5
Library:	sky130
Cellname:	sky130_fd_pr__pfet_g5v0d10v5_KLAZY6
Instance:	XM2
Width (um)	1.0
Length (um)	0.5
Fingers	1
M	1
Device type	sky130_fd_pr__pfet_g5v0d10v5
Diffusion contact coverage (%)	100
Poly contact coverage (%)	100
Guard ring contact coverage (%)	100
Guard ring top/bottom contact coverage (%)	100
Overlap at poly contact	☐
Overlap at diffusion contact	✓
Add top gate contact	☐
Add bottom gate contact	✓
Add guard ring	✓
Full metal guard ring	✓
Add left guard ring contact	✓
Add right guard ring contact	✓
Add bottom guard ring contact	✓
Add top guard ring contact	✓
Source via coverage [+/-](%)	100
Drain via coverage [+/-](%)	100
Gate via coverage [+/-](%)	100
Bottom guard ring via coverage [+/-](%)	100
Top guard ring via coverage [+/-](%)	0
Right guard ring via coverage [+/-](%)	0
Left guard ring via coverage [+/-](%)	0

```
           width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   0.010 x 0.010   ( 0.000,  0.000), ( 0.010,  0.010)  0.000
lambda:     1.00 x 1.00    (  0.00,  0.00 ), (  1.00,  1.00 )  1.00
internal:      2 x 2       (     0,  0    ), (     2,  2    )  4
Main console display active (Tcl8.6.9 / Tk8.6.9)
Creating layout from example.spice
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )

microns:   0.000 x 0.000   ( 0.000,  0.000), ( 0.000,  0.000)
lambda:     0.00 x 0.00    (  0.00,  0.00 ), (  0.00,  0.00 )
internal:      0 x 0       (     0,  0    ), (     0,  0    )
Loading DRC CIF style.
select cell: XM2
%
```

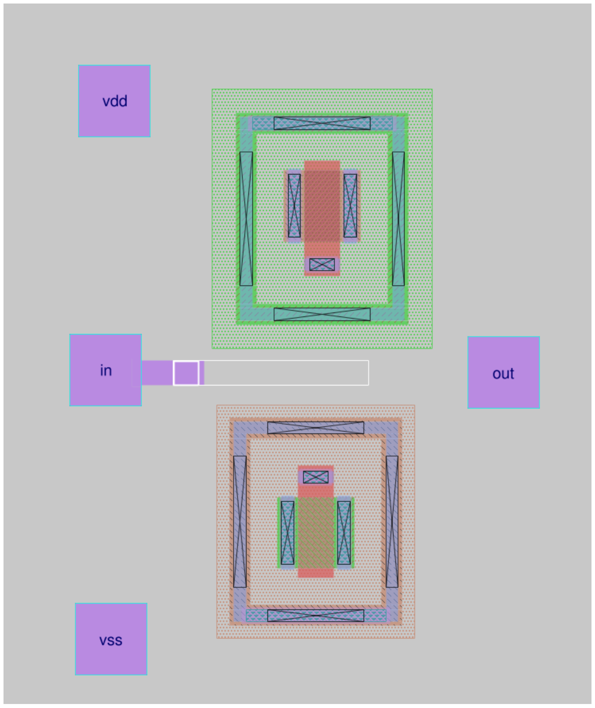
To move the device instances where you want, select the instance with the "`i`" key, then put the mouse pointer where you want the instance to go, and press the "`m`" key ("m" for "move"). Put the transistors in a useful arrangement. Note the "DRC" in the title bar of the window. If there is a white dot stipple pattern anywhere in the window, then Magic has detected one or more DRC errors, and the DRC in the title bar will have a non-zero value. Usually after initial automatic placement when importing from SPICE, there will be many such errors until you move the instances away from each other. Make sure your final placement has no DRC errors.

To wire things up, you need to know a bit about how the tool Magic does things, and it does things differently than most layout tools. Most layout tools have an object-based database, and all layout consists of many objects like instances and wires. Magic treats instances as objects, but all drawn layout is treated as "paint" which covers the area of the layout, and you can do paint-like methods such as "paint", "erase", "fill", and "stretch".

Magic has a "cursor box" which is the white box in the layout window, and the mouse left and right buttons can be used to place the bottom left and top right corners, respectively. Some commands like the ones we used so far operate relative to the mouse pointer position. Other commands operate relative to this box. The "`paint`" command is one such operator.

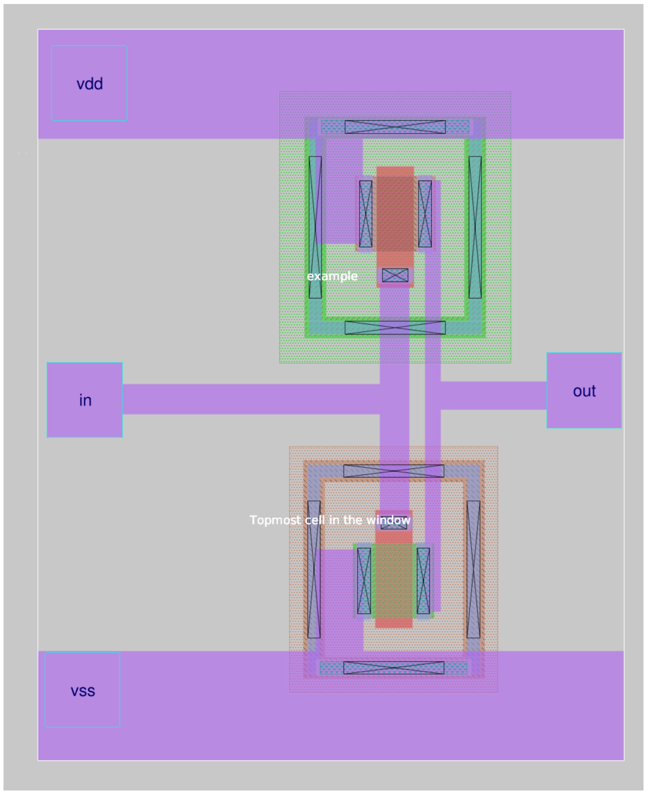
Magic is at heart a command-line driven application. There are key bindings and mouse bindings to most of the major commands, but you can always fall back on typing commands in the terminal. To start a wire, for instance, you can place and size the box where you want the wire to go. Then you can use the semicolon ("`;`") key to redirect input to the console, and type the command "`paint m1`". Alternatively, you can place the box where you want, find the icon for metal1 on the right, and paint into the box with a middle-mouse click. Then a third quick method is to put the box where you want, and click the middle-mouse button with the mouse pointer over any example of the layer type that you want to paint. This method works for erasing, too; place the box over the spot where you want to erase something, then click the middle mouse button while the pointer is over the empty background of the layout window.

If you want to quickly draw complex wiring, you can use the "wire tool". If you press the space bar in Magic, the application switches between "tools", where each "tool" redefines the mouse button bindings. The cursor changes with each tool so it's easier to know which tool you're working with. The default tool is the "box" tool, where the mouse buttons are bound to functions that position the box corners. If you press the space bar once, you're in the "wire" tool. With the "wire" tool, the mouse buttons let you create wires interactively, and easily move between metal layers. Hitting the space bar several more times will cycle back to the "box" tool.



In the "wire" tool, the left mouse button "picks" a layer type and size for a wire. With the "wire" tool in effect, put the mouse pointer over one of the wires that you previously painted, and click the left mouse button. This will select the type and size for the wire. Then you can extend the wire interactively, and each time you click the left mouse button, a wire segment will be painted. The right mouse button ends the wire. Holding the "shift" key down while pressing the left or right mouse buttons will create a contact and move up or down a metal layer, respectively.

Use the "`move`" command to place the instances and pins, use the box tool to paint the power and ground wires, and use the box and wire tools to wire the sources to power and ground, the gates to the input, and the drains to each other and the output. Use the "`select`" command ("`s`" key) by typing it multiple times with the pointer over any material on the layout——that will highlight a connected network, so you can check that the proper connections have been made.



You can use "Save and quit" from the "File" menu, or from the command line, type "`writeall`" and "`quit`". Then start magic again like this:

```
    magic -d XR example
```

When magic starts up, it loads the layout. But since it's a hierarchical layout with parameterized cells, they show up initially as empty boxes. To show the contents, select the whole cell by putting the pointer over an empty area of the layout and type "`s`". Then type "`x`" to expand the view.

LVS

Now that you have a layout, it's time to verify it. You'll need an extracted netlist. This can be done from the command line. Type:

```
extract do local
extract all
ext2spice lvs
ext2spice
```

This is a 2-step process that creates an intermediate format "`.ext`" file, then uses that to generate the "`.spice`" netlist. Now that the netlist has been made, you can quit magic.

For layout-vs.-schematic validation, go to the `spice/` directory:

```
cd ../spice
```

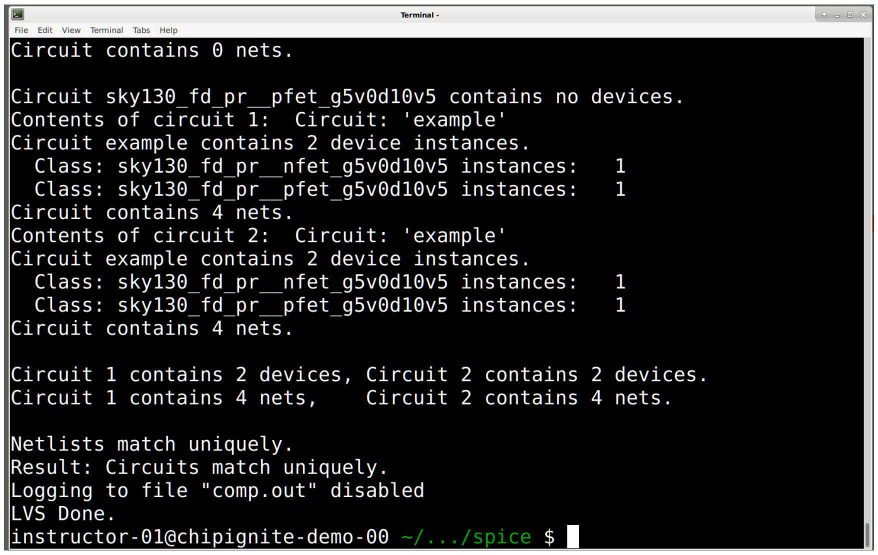
The sample_and_hold circuit has a helpful script for running LVS, so copy that over

```
cp ../../sample_and_hold/spice/run_lvs.sh .
```

Then change "`sample_and_hold`" to "`example`" everywhere. The line that reads the extra component netlist isn't needed, and use the setup file from the PDK. Then you can run LVS with

```
./run_lvs.sh
```

If all goes well, then the output says that the layout and schematic netlists are equivalent. Details of the output are in the file "`comp.out`".



```
Circuit contains 0 nets.

Circuit sky130_fd_pr__pfet_g5v0d10v5 contains no devices.
Contents of circuit 1:  Circuit: 'example'
Circuit example contains 2 device instances.
  Class: sky130_fd_pr__nfet_g5v0d10v5 instances:   1
  Class: sky130_fd_pr__pfet_g5v0d10v5 instances:   1
Circuit contains 4 nets.
Contents of circuit 2:  Circuit: 'example'
Circuit example contains 2 device instances.
  Class: sky130_fd_pr__nfet_g5v0d10v5 instances:   1
  Class: sky130_fd_pr__pfet_g5v0d10v5 instances:   1
Circuit contains 4 nets.

Circuit 1 contains 2 devices, Circuit 2 contains 2 devices.
Circuit 1 contains 4 nets,    Circuit 2 contains 4 nets.

Netlists match uniquely.
Result: Circuits match uniquely.
Logging to file "comp.out" disabled
LVS Done.
instructor-01@chipignite-demo-00 ~/.../spice $
```

GDS

Once validated, you can create GDS, and optionally, LEF views of the layout. Go back to the layout directory and start magic:

```
cd ../mag
magic -d XR example
```

Then, in the command window, type:

```
gds write example.gds
```

to get the final GDS for the design. If you are doing a mixed-signal design and want to have a tool like openlane treat the analog circuit as a macro component to place and route, then you may also want to create a LEF abstract view of the cell, which you can do with:

```
lef write -hide
```

You can look at the abstract view now by quitting and restarting magic, then typing

```
lef read example
load example
```

which will show you a simplified view of the cell that can be used by place & route tools.

Analog Wrapper

Finally, if you have created an analog design for tapeout on the Open MPW, you want that design to be embedded in the Caravan chip using the analog user project wrapper.

```
cd ../../caravel_user_project_analog/mag/
magic -d XR user_analog_project_wrapper
```

Unlike digital synthesis for the Caravel chip, the analog project wrapper is a large blank slate which is yours to route as you please. The one that comes with the repository has an example circuit in it; you will want to copy the "empty" version of the wrapper and use that for your project.

To place your project into the wrapper, just put the cursor box where you want the circuit to go and type

```
addpath ../../example/mag
getcell example
```

or perhaps

```
addpath ../../sample_and_hold/mag
getcell sample_and_hold
```

Then you'll be responsible for wiring up to the wrapper pins, verifying the top level circuit through simulation and LVS, and generating GDS. But that's all there is to it!