Lab 01: Open a cell you did not draw

You'll load this course into the workbench, open the foundry's own inverter, `sky130_fd_sc_hd__inv_1`, in Magic, and measure its two transistors from the layout file. Then you'll run four checks on the same cell (DRC under two rule sets, extraction, and LVS) and get them to agree about it. If they do, you know your toolchain works, so any problem you run into after this lab is a problem in your drawing.

Package: `labs/lab-01-open-the-cell/`

Before you start

- Read Getting started, What a layout is, SiliWiz and Anatomy of a standard cell.
- Finish IC101: you need the `workspace` clone and the noVNC desktop.
- Keep the Magic cheat sheet open in another tab.
- Have paper or a text file ready for the numbers you work out in Part 3.

Time: about 1 hour 15 minutes, plus 20 minutes for the optional SiliWiz part. `make` itself takes about a second and a half.

Part 1: Load the course

If you haven't set up the workbench yet, do IC101 Lab 03 first.

1. Start the noVNC desktop the way IC101 showed you, and open a terminal on it. Type every command in this lab in that terminal, not on your laptop.
2. Load the course environment and check the PDK. The setting lasts only as long as the terminal you load it in, so you'll need to load it again in every new terminal you open.

   ```
   . /foss/designs/common/.designinit
   echo $PDK
   ```

   `echo $PDK` must print `sky130A`. If it prints `ihp-sg13g2`, which is the image's default, the first line didn't run in this terminal.
3. Get the course files with `mod`, a command that `.designinit` defines.

   ```
   mod add ad104     # first time only
   mod ad104         # puts you in the course folder
   ```
4. Check the versions of the three tools whose output this course prints:

   ```
   magic --version
   netgen -batch quit | grep '^Netgen'
   ngspice --version | grep ngspice-
   ```

   ```
   8.3.681
   Netgen 1.5.323 compiled on Tue Aug 25 11:33:45 AM CEST 2026
   ** ngspice-47 : Circuit level simulation program
   ```

   If you have a different version, your outputs may not match the ones printed in these labs.

What you see	What it means
`.designinit: No such file or directory`	The workspace isn't set up, so do IC101 Lab 03. Until then you can still run `make` in any lab, because every lab package pins its own PDK.
`couldn't read Username for 'https://github.com'` on `mod add ad104`	The repo didn't answer an anonymous request. That's a problem on our side rather than anything you did, so please tell us in Discord.

For anything else, see IC101's Troubleshooting page.

You should see `sky130A` from step 2 and the three versions from step 4.

Part 2: Look at the cell in Magic

What is in the foundry's inverter, and what is missing from it?

1. From the course folder, go into the lab package and open the cell:

   ```
   cd labs/lab-01-open-the-cell
   make gui
   ```

   (`make edit` does the same thing, and it's the name every other lab uses.) Magic opens two windows with the cell loaded: `layout1`, which shows the drawing, and `tkcon`, a Tcl console where you type Magic commands. You get the terminal back while Magic stays open, so you can keep using both.

2. Using Anatomy of a standard cell as a map, find the items below. The figure in What a layout is shows which colour is which layer. To check a position or a size, put the cursor box on it (left-click its lower-left corner, right-click its upper-right) and type `box` in the console, which prints the box's width, height and corners in microns.

   - the n-well across the top, and the p-diffusion inside it
   - the n-diffusion in the bottom half
   - the one poly stripe that crosses both diffusions, which is why the two gates are connected without a wire
   - the two metal1 rails, straddling y = 0 and y = 2.72
   - the contacts, each of which is 0.17 µm square

3. Now look for something the cell doesn't have, a well tap. A tap is a contacted strip of `ptap` or `ntap`, which is silicon doped the *same* type as the well under it and tied to a rail. Source and drain contacts don't count as taps, because their diffusion is the opposite type to the well. (What a layout is explains what a tap is for.)

You should see every item in step 2, and no tap anywhere in the cell. Leave Magic open, because you'll use it again in Part 6.

Part 3: Measure the transistors

How wide and how long are the two transistors? You can work this out yourself from the layout file, before any tool extracts it for you.

1. In the terminal, print the file's header and its two transistor layers:

   ```
   F=$PDK_ROOT/sky130A/libs.ref/sky130_fd_sc_hd/mag/sky130_fd_sc_hd__inv_1.mag
   head -4 $F
   grep -A1 -e '<< scnmos >>' -e '<< scpmoshvt >>' $F
   ```

```
magic
tech sky130A
magscale 1 2
timestamp 1785844050
<< scnmos >>
rect 120 47 150 177
<< scpmoshvt >>
rect 120 297 150 497
```

`scnmos` is the NMOS and `scpmoshvt` the high-threshold PMOS. Each `rect` line is one rectangle, given as `x1 y1 x2 y2`, and it covers the patch of diffusion under the poly stripe, which is the transistor's channel (What a layout is).

2. The header says `magscale 1 2`, so one unit in the file is 0.005 µm (Drawing in Magic). For each rectangle, work out its height and its width in µm, and write them down. The height is the transistor's W and the width is its L.

You should see two `rect` lines, and four numbers in your notes: W and L for each transistor. Part 4 checks them against what the tools measure.

Part 4: Run the check

Do the tools measure the cell the way you did?

1. Run the lab:

```
make
```

The `Makefile` sets `PDK=sky130A` itself, so this works even if `.designinit` failed for you. It takes about a second and a half (1.35, 1.46 and 1.60 s on three timed clean runs) and prints:

```
== magic: load, DRC, extract   (sky130_fd_sc_hd__inv_1)
== netgen: layout vs the foundry's own netlist

  THE FOUNDRY'S INVERTER, MEASURED BY YOUR TOOLS

  ok  cell width                              1.38 um   (reference 1.38)
  ok  cell height                             2.72 um   (reference 2.72)
  ok  DRC violations, default style                 0   (reference 0)
  ok  DRC violations, full style                    3   (reference 3)
  ok  devices extracted                             2   (reference 2)

      cell area  1.38 x 2.72 = 3.7536 um^2

  THE TWO TRANSISTORS IT FOUND
      X0  sky130_fd_pr__nfet_01v8          W=0.65  L=0.15   drain area 0.169 um^2
      X1  sky130_fd_pr__pfet_01v8_hvt      W=1  L=0.15   drain area 0.26 um^2

  LVS: Circuits match uniquely

  THE THREE THINGS THE FULL RULE SET COMPLAINS ABOUT
      All nwells must contain metal-connected N+ taps (nwell.4)
```

```
      P-diff distance to N-tap must be < 15.0um (LU.3)
      N-diff distance to P-tap must be < 15.0um (LU.2)
      ...

PASS  Magic read the foundry's own layout, found zero violations under the
      default rule set and the three expected context rules under the full
      one, pulled two transistors out of the geometry, and netgen agreed
      they are the circuit the foundry says the cell is.
```

If it stops or a number differs:

What you see	What it means
`DRC violations, default style` is not 0	The technology did not load. Check that `$PDK_ROOT` points at the PDKs (`/foss/pdks` in the workbench image) and that `.magicrc` is in the folder you ran `make` from.
`results/report.txt is missing`	Magic didn't finish, so read `results/magic.log`.
`LVS: netgen wrote no report`	Read `results/netgen.log`. The cause is usually `$PDK_ROOT` again.
`magic: command not found`	You're not in the workbench container.

2. Compare the two transistor lines with the W and L you wrote down in Part 3.
3. Look at the count under the full rule set. The foundry's own cell fails the full rule set, and that's the answer we expect. (Magic calls a rule set a DRC *style*, which is why the output says `default style` and `full style`.) Why does the full rule set find three violations in a cell that has been taped out hundreds of times?

   Answer: A1, in Answers at the end.

You should see `PASS`, `0` under the default rule set and `3` under the full one, and `W=0.65 L=0.15` and `W=1 L=0.15` matching the numbers you wrote down.

Part 5: Read the extracted netlist

What circuit did Magic pull out of the geometry?

1. Print the netlist Magic extracted:

```
make netlist
```

```
* NGSPICE file created from sky130_fd_sc_hd__inv_1.ext - technology: sky130A

.subckt sky130_fd_sc_hd__inv_1 A VGND VNB VPB VPWR Y
X0 Y A VGND VNB sky130_fd_pr__nfet_01v8 ad=0.169 pd=1.82 as=0.169 ps=1.82 w=0.65 l=0.15
X1 Y A VPWR VPB sky130_fd_pr__pfet_01v8_hvt ad=0.26 pd=2.52 as=0.26 ps=2.52 w=1 l=0.15
.ends
```

2. Count the ports; there are six. `A` and `Y` are the input and output, and `VPWR` and `VGND` are the supplies. What are `VNB` and `VPB`, and why are they pins?

   Answer: A2, in Answers at the end.

3. Find `w=0.65` and `w=1`. Nobody typed those numbers in; Magic got them by measuring the same two rectangles you measured in Part 3.
4. Find `pfet_01v8_hvt`. It tells you the PMOS is the high-threshold flavour, which you'll need to remember in Lab 03.

You should see two devices, six ports, and `w=0.65 l=0.15` on the NMOS.

Part 6: Run the flow by hand

Which commands does Magic run for `make`, and which of them are easy to get wrong?

Everything Magic did for `make` in Part 4 comes down to a short list of commands that you can type yourself. Every remaining lab in the course uses this same flow, so it's worth running it once with your own fingers.

1. In the `tkcon` console (the Tcl console window that opened with Magic in Part 2), type these one at a time:

```
select top cell
drc euclidean on
drc check
drc catchup
drc count total
drc style drc(full)
drc check
drc catchup
drc count total
drc why
extract do local
extract all
ext2spice lvs
ext2spice -o /tmp/inv.spice
```

2. Watch the count change when you switch rule sets. The first `drc count total` prints `Total DRC errors found: 0`, the second prints `Total DRC errors found: 3`, and `drc why` then names the three rules.
3. The last line wrote the extracted netlist to `/tmp/inv.spice`. In the terminal, compare it with the one `make` wrote:

```
diff /tmp/inv.spice results/sky130_fd_sc_hd__inv_1.lvs.spice
```

   `diff` prints nothing when the files are the same.

Three of those lines are easy to leave out or get wrong:

- **`select top cell`**: `drc why` reports only on what's inside the cursor box, and a freshly opened cell has no box worth speaking of. If you skip this line, `drc why` tells you `No errors found.` on a cell that has three violations.
- **`drc catchup`**: `drc check` only queues the re-check, and `drc catchup` waits for it to finish. If you count before it finishes, the check hasn't found anything yet, so there's nothing to count.
- **`drc count total`**, not `drc count`. Plain `drc count` prints two lines, `Cell sky130_fd_sc_hd__inv_1 has 3 error tiles.` and then `Total DRC errors found: 0`. That second line has the same label as the output of `drc count total`, but it counts something else: the *hierarchical* total across subcells, which is always 0 on a flat cell. So if you read the last line of `drc count`, the number never seems to change.

You should see `0`, then `3`, then the same three rule names as in Part 4, and no output from `diff`, which means the flow you typed by hand gave the same netlist as `make`.

Part 7: Browse the library

How does the library offer more drive strength for the same logic?

1. Count the cells in the library and list the inverters:

   ```
   ls $PDK_ROOT/sky130A/libs.ref/sky130_fd_sc_hd/mag/ | wc -l
   ls $PDK_ROOT/sky130A/libs.ref/sky130_fd_sc_hd/mag/ | grep __inv_
   ```

   The first prints `446`. The second lists seven inverters: `inv_1`, `inv_2`, `inv_4`, `inv_6`, `inv_8`, `inv_12` and `inv_16`. They all have the same logic, and the drive strength goes up from each one to the next.

2. Open the whole library GDS in KLayout, which is a much better viewer than Magic. `make klayout` starts it in viewer mode, where it can't edit anything, so you can browse safely:

   ```
   make klayout
   ```

3. Before you look, write down a prediction: is `inv_8` taller than `inv_1`, wider, or both? Then open `inv_8` next to `inv_1` and see where the factor of eight went.

   Answer: A3, in Answers at the end.

You should see 446 cells, seven inverters, and an `inv_8` that is visibly wider than `inv_1`.

Part 8: Explore SiliWiz (optional)

This part runs in your browser and needs no setup. It takes about twenty minutes.

1. Open **https://app.siliwiz.com/**. SiliWiz describes the interface. Load the inverter from the **Preset** dropdown above the drawing area to see a finished example.
2. Open **LESSONS** (top right) and do the *nmos* lesson. Drag the cross-section cut line with the blue slider so that it passes through the middle of the poly stripe, and look at the cross-section: substrate, a thin oxide, poly on top, and doped regions on either side.
3. Now drag the cut so that it misses the poly. The cross-section becomes two doped regions with nothing joining them, because the transistor exists only where the poly crosses.
4. Move a contact until it hangs off the edge of the diffusion it sits on, and open the **CROSS SECTION & DRC** tab. SiliWiz's rules aren't SKY130's, but its complaint, "this thing does not surround that thing by enough", has the same shape as half the errors you'll fix in Lab 02.
5. Do the *parasitics* lesson. Stretch a metal wire longer and watch what happens to the simulated waveform. That's the argument of Lab 05, made in two minutes.

You should see the transistor appear and disappear in the cross-section as the cut moves on and off the poly, and a DRC complaint as soon as the contact leaves its diffusion.

Finishing

You're done when `make` ends in `PASS` and the W and L you worked out from the `.mag` file in Part 3 match the `W=` and `L=` it prints for both transistors.

Questions:

- Why does the full rule set find three violations in a cell that has been taped out hundreds of times? Part 4, step 3 answers it.
- Why does the last line of `drc count` never seem to change when you switch rule sets? Part 6 answers it.

Submit: nothing for this lab.

Next on the Course path: read Drawing in Magic, Reading DRC errors and The full DRC rule set, then do Lab 02: Break it on purpose.

Answers

A1. Part 4: Run the check

All three come from the same omission. The cell has no well tap, because the library puts its taps in a separate cell, and Anatomy of a standard cell explains why it's done that way. `make` prints the same explanation under the three rule names, where the output above shows `...`.

A2. Part 5: Read the extracted netlist

They're the bulk terminals, the fourth terminal on each device line. `VNB` is the NMOS body (the p-substrate) and `VPB` the PMOS body (the n-well). The cell exposes them as pins because it doesn't tie them to anything itself, since it has no well tap, as you found in Part 2.

A3. Part 7: Browse the library

It's only wider. Every cell in the library is 2.72 µm tall, and each step up in drive strength is the same logic with wider transistors (Anatomy of a standard cell), so the extra area has to go sideways.