Instances and hierarchy

By the end of Lab 03 you have drawn an inverter by hand, and you aren't going to draw a million more of them that way.

From here on the work is **instancing**: you place copies of finished cells next to each other and draw the wires between them. The rest of a chip is built the same way, and that's possible because the hard part, the transistors, is drawn only once. You'll compose your first two blocks this way in Lab 04.

Placing an instance

In Magic, you place an instance by typing these commands in the console:

```
addpath ../mag                  ;# where to look for cells
box 3um 0um 3um 0um             ;# a zero-size box at the spot you want
getcell my_inv                  ;# drop a copy with its origin there
identify inv2                   ;# give this copy a name
```

`getcell` places an **instance**, which refers to `my_inv` rather than copying its geometry. If you run those four commands in a new cell, from a folder next to `mag/`, and save it, the `.mag` file records the whole instance in four lines:

```
use my_inv  inv2 ../mag
timestamp 1787733743
transform 1 0 300 0 1 0
box 0 0 300 600
```

The `use` line names the cell, the instance and the folder to look for the cell in. The `timestamp` is a copy of the one at the top of `my_inv.mag`, `transform` puts the cell's origin at x = 300, which is 3 µm in this file's 0.01 µm units, and `box` is the outline of `my_inv` in its own coordinates. Magic writes the folder only on the first `use` of each cell in a file, so in Lab 04's `solutions/buf_ref.mag` the line for `inv1` ends in `../mag` and the line for `inv2` doesn't. When the cell is in the same folder as the file that uses it, as in Lab 04's `mag/my_buf.mag`, none of the `use` lines has a path.

Because every instance refers back to `my_inv.mag`, editing that one file changes all of them. That's the reason to use instances, and it's also why Lab 04's blocks are only a few lines long.

Abutment

Both of Lab 04's cells, `my_inv` and `my_nand2`, are **6.00 µm tall**, with their VGND rail at y = 0.00–0.40 and their VPWR rail at y = 5.60–6.00. When you put the second cell at an x equal to the width of the first, the two cells touch edge to edge, and:

- the two VGND rails become one continuous strip,
- the two VPWR rails become one continuous strip,
- the wells touch and merge,
- and the second cell has power without you drawing anything.

That is what makes a cell a **standard cell**: it keeps to a fixed height and fixed rail positions, so that placing cells is just arithmetic.

The foundry's library keeps to a height of 2.72 µm and a site width of 0.46 µm, so every cell in it is a multiple of 0.46 µm wide. In the capstone you place four of its cells at x = 0, 7.36, 9.66 and 11.96, each one the running total of the widths of the cells before it, and the rails line up without any further work from you.

Hierarchy

In outline, Lab 04's buffer is:

```
load my_buf
place my_inv inv1 0.0
place my_inv inv2 3.0
paint li from inv1/Y across to inv2/A
label A, label X, label VGND, label VPWR
```

When you extract it, you get this netlist:

```
.subckt my_buf A X VGND VPWR
Xinv1 A inv2/A VPWR VGND my_inv
Xinv2 inv2/A X VPWR VGND my_inv
.ends
```

At this level there are two instances and the net between them, and no devices at all. The devices are inside `my_inv`, which is written out separately in the same file. netgen compares hierarchically, so in one run it checks that `my_buf` is two `my_inv` in series *and* that `my_inv` is the inverter it's supposed to be.

The AND gate uses the same idea with a different pair of cells, `my_nand2` feeding `my_inv`. An AND gate has six transistors, and you don't draw any of them.

Three common mistakes

1. A label over a subcell's geometry names nothing.

Say you place two cells and abut them, and the rails connect. Then you put the box over a rail and type `label VGND`. A label always goes into the cell you're editing, so it lands in your cell, but the rail under it is geometry inside `inv1` and your cell has nothing of its own there. So Magic puts the label on empty space (if you type `label VGND c m1`, it says so: `Moving label "VGND" from metal1 to space`), and a label on space isn't attached to any net. *Your* cell then comes out of extraction with no supply ports at all, and netgen reports a pin-matching failure on a block that is otherwise perfect.

The fix is to give the top cell some geometry of its own to put the name on. Paint a metal1 rectangle over the row (it's the same net, so it changes nothing electrically) and put the label on that. Every composed block in this course does this, and the solutions READMEs for Lab 04 and the capstone explain it as a decision you can break on purpose to see what happens.

2. The layers above a cell are not always empty.

`and2_0` and `buf_1` contain metal1 only in their rails, but the flip-flop `dfxtp_1` also has internal metal1 straps at y = 1.415–1.645 and 1.755–1.985, running most of its width. If you route metal1 across the flip-flop without knowing that, you get `met1.2` spacing violations from geometry you didn't draw.

That's why real routers carry a blockage map for every cell, and it's also why the capstone's `D` pin takes its via at an unexpected place. Before you route over a cell, read the `<< metal1 >>` section of its `.mag` file; the capstone has you do this.

3. The path recorded in the `.mag` file is relative to where you saved it.

`use my_inv  inv2 ../mag` means "look in `../mag`, starting from the folder this `.mag` file is in", rather than from the folder you started Magic in. That's why you can load Lab 04's `solutions/buf_ref` from the lab folder and Magic still finds `my_inv` in the lab's `mag/`. If you move the file to a folder with no `../mag` next to it, though, Magic

can't find the cell: it prints a `couldn't be read` error for `my_inv.mag`, and the instance shows up as an empty box with a name in it. The labs avoid this by always changing to a known directory first and setting `addpath` after that, and the order is worth copying in your own work: **`cd` first, `addpath` second.**

What a place-and-route tool does

Placement and routing were the first parts of layout to be automated, and they are the most thoroughly automated, because the job is enormous and it's arithmetic. In a real flow,

1. synthesis picks a cell for each gate in the netlist, choosing drive strengths,
2. the place-and-route tool finds legal positions in rows for every one of them,
3. inserts tap cells and filler,
4. routes hundreds of thousands of nets on five metal layers,
5. and then checks its own work with the same DRC and LVS you are running.

In DD103 that whole flow runs in **about a minute** on a small design, and DD103′s own pages time it at 57 s for the vending machine and 59 s for the elevator. Placing four cells and routing three wires by hand in the capstone is no substitute for that tool, but it does show you what the tool's output is a picture of.