AD102 — Linear Circuits & Fabrication

Find out how resistors and capacitors are built on a SKY130 chip, and size your own to hit target values inside an area budget.

UofT ASIC Team

Contents

1 AD102 — Linear Circuits & Fabrication 5

I 1–2. Resistors

2 Getting started 11

3 Resistors on a chip 15

4 Ohms per square 19

5 What a resistor costs in area 23

6 Lab 01: A resistor you designed 27

II 3–4. Capacitors and inductors

7 Capacitors on a chip 39

8 What a picofarad costs 43

9 Parasitic capacitance 45

10 Inductors on a chip 47

11 Lab 02: How big is a picofarad? 53

III 5–6. RC on a chip

12 What to build instead of an inductor 65

13 Lab 03: A time constant in silicon 67

IV 7–8. Variation and matching

14 Process corners 77

15 Matching and Pelgrom's law 81

16 Lab 04: Two resistors that disagree 87

V 9–10. Capstone

17 From schematic to cross-section 95

18 Capstone: Defend a floor plan 99

Appendices 103

A The SKY130 passive catalogue 105

B ngspice deck recipes 111

1. AD102 — Linear Circuits & Fabrication

You already know what a resistor does. This course is about what a resistor, a capacitor and an inductor are when they are built on silicon, and what each one costs in chip area.

The course is part of the **UofT ASIC Team** education materials, and it's published at **https://uoftasic.com/ad102/**. The labs run inside the **workbench**, which is the IIC-OSIC-TOOLS container you start from the `workspace` repo you set up in IC101.

Who this course is for

AD102 is the second course in the analog track, for 2nd–3rd year ECE students who have finished IC101 and AD101.

ECE110 teaches linear circuit analysis: node voltages, mesh currents, $V = IR$, RC time constants, impedance. AD102 doesn't teach any of that again, and it isn't a substitute for it. If you have taken ECE110 you are ahead, and if you haven't, AD101 already covered everything this course uses.

What ECE110 leaves out is where the components come from. In a lab kit, a resistor comes out of a drawer with its value printed on the side. On a chip there's no drawer to take it from: a resistor is a strip of doped polysilicon that you draw, and its value follows from the drawing, from how long and how wide the strip is and which layer it's on. Changing the shape is the only way you have to change the value. The idea that this whole course builds on is that **a component value is a geometry, and geometry costs area.**

On a breadboard, component values are free and independent of each other: a 10 kΩ costs the same as a 100 Ω, and you can have as many as you like. On silicon neither of those is true. A 1 pF capacitor is roughly a 22 µm × 22 µm plate: 484 µm² of silicon, which simulates at 982.5022 fF, a little short of 1 pF (Lab 02 gives you the formula behind that number, and sizes the plate exactly). The 4-bit ripple-carry adder that the digital track builds in DD103, twelve SKY130 standard cells, fits in 110.1056 µm² of the same process, so that one capacitor takes the area of 4.4 adders. The analog habits you'll pick up here all follow from that comparison: ratios instead of values, unit devices, and keeping inductors off the chip below a gigahertz.

At a glance

Track	Analog
Prerequisites	IC101 then AD101, in that order
Tools	ngspice and XSchem, both inside the workbench, plus SiliWiz (a browser layout tool that shows cross-sections) in a browser tab
PDK	SKY130 (`sky130A`), included in the image
Time	About 15.5 hours, self-paced
Math	Algebra, $V = IR$, and the R, C and frequency material from AD101. No calculus.

Course path

Work down this table in order. A Read row is one or more guide pages: concepts, figures and worked examples, with nothing to run. A Do row is one lab, where you run every command yourself. Each lab starts by listing the pages to read first.

#		What	Time
1	Read	Getting started · Resistors on a chip · Ohms per square · What a resistor costs in area	1.5 h
2	Do	Lab 01: A resistor you designed. Load the course, measure the sheet and end resistance of SKY130 poly, and size two resistors to within 1 % of their targets.	2.5 h
3	Read	Capacitors on a chip · What a picofarad costs · Parasitic capacitance · Inductors on a chip	1.5 h
4	Do	Lab 02: How big is a picofarad?. Fit a MIM capacitor's area and edge terms, size two capacitors, and measure the three inductors SKY130 ships.	2.5 h
5	Read	What to build instead of an inductor	30 min
6	Do	Lab 03: A time constant in silicon. Measure the time constant of your own resistor and capacitor in one circuit, then build a 20 ns RC in under 400 µm².	2 h
7	Read	Process corners · Matching and Pelgrom's law	1 h
8	Do	Lab 04: Two resistors that disagree. Run three process corners and a seeded Monte Carlo, then design a 0.45 V divider that is accurate, precise and under 150 µm².	2 h
9	Read	From schematic to cross-section	30 min
10	Do	Capstone: Defend a floor plan. Read the layer stack, look at cross-sections in SiliWiz, and size a 1 MHz	1.5 h

#	What	Time
	low-pass filter whose area you can defend.	

The times are estimates, and they add up to the 15.5 hours above. There's no separate setup row, because setting up is the first part of Lab 01.

Keep these open while you work; they are for looking things up rather than reading through: The SKY130 passive catalogue · ngspice deck recipes

Stuck? Ask in the team Discord. Nobody expects you to work this out alone at 2 a.m. Getting help says what to put in your question.

What you'll do

1. Measure the sheet resistance of SKY130 high-sheet poly yourself. The model card (the PDK's SPICE description of the device) says 317.3885 Ω per square, but one drawn square simulates at 695.4646 Ω, more than twice that. You'll find where the extra ohms come from, then size a 10 kΩ resistor correctly on the first try (30.3315 µm long, landing at 9999.997 Ω), and two more of your own.
2. Work out what a picofarad costs in square micrometres, and why nobody puts a 1 µF capacitor on a chip.
3. Build an RC filter from the resistor and capacitor you designed and measure its time constant τ. Then measure its corner frequency, check it against $1/(2\pi\tau)$ to five figures, and compare its Bode plot with that of the same filter built from an ideal 10 kΩ and an ideal 1 pF: 15.8580 MHz for the filter you built against 15.9155 MHz for the ideal one, 0.36 % apart. Optionally, open the same circuit in XSchem and see a schematic turn into the device lines you have been editing by hand.
4. See the same resistor come out different at every process corner, and learn why analog designers build ratios rather than values.
5. Find out how you fabricate an inductor on a chip. Mostly you don't, so you'll also see what gets built instead.
6. Hit a filter spec using only devices the PDK actually ships, inside an area budget you have to defend.

The workbench

This course is pinned to the workbench image **`hpretl/iic-osic-tools:2026.08`**, which ships **ngspice-47** and **XSchem V3.4.8RC**. Tutorials online describe other versions, which have different flags, so when a command here disagrees with one you find online, use the one here: it's the one that runs on your machine. You'll check your versions in Lab 01, Part 1.

Each of Labs 01–04 has a **package**: a folder under `labs/` with its SPICE decks, a `Makefile` and a checker script. The packages run with `make` alone, with no environment setup, because every deck names its SKY130 model file by absolute path. You still do Lab 01, Part 1 once, to get the course files and check your tools. Apart from the SiliWiz parts, which run in a browser tab, every lab step runs in a terminal, and only the optional XSchem part of Lab 03 opens a window on the desktop. A fifth package, `passives-decks`, holds the decks whose output the guide pages quote, and some lab parts run it too. The capstone has no package, because all it needs is one command, a browser exercise and a page of arithmetic.

The first time you run each lab, its `make` ends in `FAIL`, and that's on purpose. Each package ships the design a first-year student would draw, usually sized straight off the model card, so that you read your own wrong number before you fix it. The message tells you which file to edit.

What you'll have at the end

You'll have a filter you designed from devices a real 130 nm process can make, simulated at every corner it will be manufactured at, with an area number you can defend. You'll also be in the habit of asking, for each part of an analog schematic, what shape it is and what it costs.

Next courses

- **AD103** — Nonlinear Circuits: the diode and the MOSFET, and the regions they operate in
- **AD104** — Layout: draw these devices yourself in Magic, DRC-clean and LVS-clean

I

1–2. Resistors

2 **Getting started** 11
2 What this course assumes 11
2 The PDK setting 11
2 Where your files live 12
2 The u suffix trap 12
2 Expected warnings 12
2 How the labs work 13
2 SiliWiz 14
2 Launch GUI tools from a terminal 14
2 Getting help 14

3 **Resistors on a chip** 15
3 Animation: resistor geometry 15
3 What the resistor is made of 15
3 The SKY130 resistor menu 16
3 The same strip in four materials 16
3 Writing W and L 17

4 **Ohms per square** 19
4 Why it is called “per square” 19
4 Sheet resistance of the SKY130 layers 19
4 Contact resistance 20
4 Head and body of one resistor 21
4 The design equation 21
4 Using sheet resistance 22

5 **What a resistor costs in area** 23
5 Three values, sized and simulated 23
5 Width and area 23
5 Changing the material 24
5 Folding a long resistor 25
5 How this shapes analog circuits 25

6 **Lab 01: A resistor you designed** 27
6 Before you start 27
6 Part 1: Load the course 27
6 Part 2: Measure the material 28
6 Part 3: Size a 10 kΩ resistor 29
6 Part 4: Change the width 30
6 Part 5: Change the recipe 30
6 Part 6: Split a resistor into head and body 31
6 Part 7: Size your own resistors 32
6 Finishing 33
6 Going further (optional) 33
6 Answers 35

2. Getting started

AD102 uses two tools from the workbench you set up in IC101. **ngspice**, the open-source descendant of Berkeley SPICE, simulates a circuit and tells you what it would do, and **XSchem** is the schematic editor you use to draw circuits for it. Both are already in the image, so you won't install anything on this page. Instead it explains how the pieces fit together, goes through the warnings you'll see every day, and describes the one mistake that costs students the most time. You'll do the setup itself in Lab 01, Part 1.

What this course assumes

Requirement	Notes
IC101 completed	The workbench comes up in your browser and the smoke test passes
AD101 completed	You can read a waveform, a spectrum, and a Bode plot
ECE110-level circuit analysis	Helpful, not required: $V = IR$, series and parallel, and what an RC corner is. AD101 covers the last one.

AD102 doesn't re-teach linear circuit analysis; node equations, Thévenin equivalents and phasors belong to ECE110. Instead this course asks two questions that ECE110 doesn't: where did the component come from, and what did it cost?

The PDK setting

A **Process Design Kit (PDK)** is the foundry's description of what a process can build: the device models, the schematic symbols, and the drawing rules. Think of it as the parts catalogue for a chip.

The workbench image can simulate more than one process, and the environment variable `PDK` chooses which one. The image's default is `ihp-sg13g2`, a real 130 nm process from IHP in Germany. It's a perfectly good process, but we measure SKY130 (`sky130A`) in AD102, so that's the one you need.

The workspace repo from IC101 has a setup script, `/foss/designs/common/.designinit`, that switches you to SKY130. You load it with a leading `.`:

```
. /foss/designs/common/.designinit
```

The `.` is the shell's `source` command. It runs the file in your current shell rather than in a child process, and that's the only way a script can change your shell's environment. If you type the path without the dot and the space, the file doesn't run in your current shell, so the command does nothing useful. The catch with sourcing is that the change lasts only as long as the shell it was made in, so every new terminal needs it again. On the IC101 workbench the image does this for you whenever a terminal opens (the `workspace` repo's own `.designinit` chains to this file), so you'll usually find `PDK` already set to `sky130A`. Loading it a second time does no harm.

When XSchem starts, it reads `$PDK` to decide which symbol library to load. Before you load `.designinit`, XSchem takes its settings from `~/.xschem/xschemrc`, which does this:

```
# Source the PDK xschemrc file
source $env(PDK_ROOT)/$env(PDK)/libs.tech/xschem/xschemrc
```

After `.designinit`, XSchem uses the workspace's own `common/xschemrc` instead, and that file reads `$PDK` too. Either way, it's the `PDK` variable that picks the parts catalogue. On the default you get a library called `sg13g2_pr`, and after `.designinit` you get `sky130_fd_pr`. (`_pr` is the foundry's shorthand for **primitive**

devices: the transistors, resistors and capacitors themselves, as opposed to logic gates built from them.) If you launch XSchem on the default, you get a schematic editor full of `sg13g2_*` devices with no SKY130 resistor anywhere, and no error message either, because nothing is broken; you've simply opened a different foundry's catalogue.

So make a habit of running `echo $PDK` before you launch XSchem. It should print `sky130A`.

The lab decks don't depend on the PDK setting, though XSchem does (see How the labs work below). You still need `.designinit` to get the course files, because it also defines `mod`.

Where your files live

`mod`, the shell function that `.designinit` defines, has two forms: `mod add ad102` clones this course into `/foss/designs/modules/ad102`, and `mod ad102` takes you there afterwards. The folder `/foss/designs` inside the container is really your `workspace` clone on your own computer, shared into the container (a bind mount). That means the course files are in `workspace/modules/ad102` on your disk, and deleting the container doesn't delete your work.

The `u` suffix trap

This mistake produces no error message, so we describe it here, before you meet it in Lab 01.

In an ngspice deck, most numbers take SI suffixes: `1k` is a thousand ohms, `0.7p` is 0.7 picofarads and `3n` is three nanoseconds. It therefore seems reasonable to write a device one micron wide as `W=1u`.

SKY130 device widths and lengths don't work that way, because in this PDK `W` and `L` are already plain numbers in microns. `W=1` means one micron, and `W=1u` means one micron × 10^{-6}, which is smaller than an atom.

Here are the two ways of writing a SKY130 high-sheet poly resistor that is one micron wide and ten long. This is an excerpt, not a file to save; the complete deck ships as `labs/lab-01-a-resistor-you-designed/spice/u_trap.spice`.

```
Xa a 0 0 sky130_fd_pr__res_high_po W=1  L=10
Xb b 0 0 sky130_fd_pr__res_high_po W=1u L=10u
```

The deck pushes exactly 1 µA through each one and reads the resistance as V/I. The right one gives 3550.443 Ω and the wrong one gives 3193.812 Ω. The wrong one doesn't blow up, and it doesn't return zero, infinity or a negative number that you'd notice. Instead it returns an ordinary resistance 10 % away from the right answer, which is the size of error you'd blame on a model, a corner, or yourself.

The only sign of the mistake is one line in the log, buried in the middle of the run:

```
Warning: r.xb.rbody: resistance too low or not given, set to 1 mOhm
```

Even so, ngspice exits with status 0. You'll run this deck yourself in Lab 01, Part 8.

Transistors fail more loudly. The same mistake on a MOSFET, which you'll meet again in AD103, stops the simulator:

```
could not find a valid modelname
    Simulation interrupted due to error!

Error: incomplete or empty netlist
       or no ".plot", ".print", or ".fourier" lines in batch mode;
no simulations run!
```

A foundry's model table is split into **bins**: each row is valid over a stated range of widths and lengths. A device whose geometry nobody characterised matches no bin, so ngspice rejects it. That error is the better of the two outcomes, because you can't miss it, whereas the resistor's 3193.812 Ω arrives quietly.

In any SKY130 deck, `W`, `L`, `w` and `l` carry no unit suffix. If a resistance is close but not right, look for a stray `u` before you look anywhere else.

Expected warnings

Whenever you run a deck that contains a SKY130 poly resistor, ngspice prints this before it has done a single calculation:

```
Warning: Model issue on line 4842 :
  .model xa:rhead_model r sw_et=0 isnoisy=0 rsh=    3.458312000000000e+02  ...
unrecognized parameter (sw_et) - ignored
unrecognized parameter (isnoisy) - ignored
unrecognized parameter (p2) - ignored
unrecognized parameter (q2) - ignored

Warning: Model issue on line 4843 :
  .model xa:rbody_model r sw_et=0 isnoisy=0 rsh=    3.173885000000000e+02  ...
unrecognized parameter (sw_et) - ignored
unrecognized parameter (isnoisy) - ignored
unrecognized parameter (p2) - ignored
unrecognized parameter (q2) - ignored
unrecognized parameter (p3) - ignored
unrecognized parameter (q3) - ignored
```

That's ten `unrecognized parameter` lines and two `Warning:` headers for one resistor. A deck with many resistors repeats the block (Lab 01's 22-strip deck prints 170 such lines), and none of it means anything is wrong. SkyWater's resistor model cards carry parameters for noise and self-heating that this build of ngspice doesn't implement, so ngspice names each one, skips it and carries on, and your resistance is unaffected.

The block does tell you something useful, though: each resistor has two models. There's a `rhead_model` at `rsh = 345.8312` Ω per square and a `rbody_model` at `rsh = 317.3885` Ω per square. ("Ohms per square" is the unit chip designers use for a material's resistance; Ohms per square explains it.) The ends of a drawn resistor aren't made of the same material as the middle, which is why one drawn square of this device measures 695.4646 Ω rather than the 317.3885 Ω on the card. Explaining that gap is the first job in Lab 01.

A genuine ngspice failure starts a line with `Error:`, and the run ends with `no simulations run!`. That's how you tell a real failure from `Warning:` lines like the ones above, which you can ignore.

How the labs work

Every AD102 lab is a package with a `Makefile`. You go to the lab folder and type `make`, and it ends by printing a verdict: a line that says whether what you built matches what the lab expected, so you don't have to pick through a wall of numbers yourself. `make` runs everything, and each lab also has smaller targets, such as `make mine` (simulate only the file you edit) and `make check` (print the verdict again from the logs already written). Each package's README lists them.

All four lab packages run in a bare container with no environment setup at all: no `.designinit`, no `mod`, and `$PDK` still set to `ihp-sg13g2`. That's because every deck names its SKY130 model file by absolute path, so the PDK setting described above can't affect them. We built them that way on purpose, so if your setup went wrong, run `make` anyway and it will still work.

The one exception is an optional GUI step. `make edit` in Lab 03 opens the RC circuit in XSchem, and XSchem does read `$PDK` from your environment, so that step needs `.designinit` to have worked. The `make` that produces Lab 03's verdict doesn't need it.

Why the first `make` fails

Every AD102 lab package ships with the design a first-year student would draw, usually the naive one sized straight off the model card. The checker rejects it on purpose, and says so. This is Lab 01's message:

```
^ On a fresh clone this FAIL is the lab, not a broken
  package. spice/my_resistor.spice ships with the naive
  sizing on purpose. Edit the two L= numbers and run
  `make mine check` again.
```

Your job is to read your own wrong number, work out where the missing ohms went, and fix the two lengths that were wrong. `make` exits non-zero on a `FAIL`, so at this point a red exit code is what you should expect, and it doesn't mean your clone is broken.

SiliWiz

Part of this course is looking at a **cross-section**, which shows what a resistor or capacitor looks like sliced through, layer by layer. SiliWiz draws cross-sections in a browser, with nothing to install. You draw shapes on a simplified SKY130 layer stack, and it shows you the silicon underneath and simulates it. You'll use it in optional parts of Labs 01 and 02 and in the capstone.

Launch GUI tools from a terminal

These tools print their error messages to the terminal you launched them from, rather than in their own window, so launch GUI tools by typing their name in a terminal, never from a desktop icon. If you start XSchem from the applications menu there is no such terminal, so when something goes wrong the explanation is lost, and the program appears to do nothing at all.

Getting help

The course is self-paced, but that doesn't mean you have to work through it alone.

- Ask in the team Discord. No question is too basic, and most of what looks like your own mistake here is really a tool being unhelpful.
- Quote the exact error and the exit code. ngspice's first `Error:` line names the real problem, and everything after it follows from it. Paste that line, along with the command you ran and the output of `ngspice -v | head -2`.
- Say what the number was. Analog debugging is mostly comparing a number you got against a number you expected. "The filter is wrong" is hard to help with, while "I expected 3550.443 Ω and got 3193.812 Ω" can be answered in one reply.
- If a command on these pages doesn't do what the page says, that's a bug in the course, and we'd like you to report it. We ran every command here before writing it down, so a mismatch means something has changed, and we want to know.

Next: Resistors on a chip.

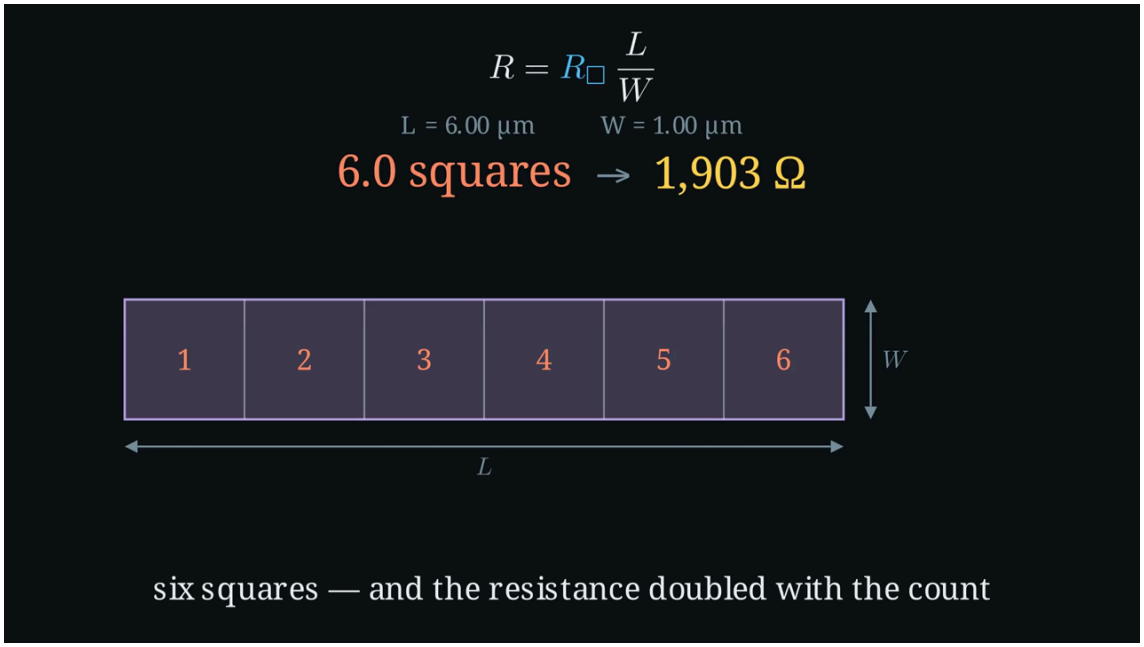
3. Resistors on a chip

In a first circuits course such as ECE110, someone handed you a resistor. It came out of a drawer with four coloured bands on it, and someone in a factory had already decided it was 10 kΩ, so your job was to put it in a mesh and write KCL. Every equation you learned there still holds on a chip.

What changes is where the 10 kΩ comes from. There is no drawer of parts on a chip. A chip is one crystal of silicon with patterns printed on it, and the only thing you can decide is what shape to print. So "how do I get 10 kΩ?" becomes a question about geometry, and because geometry costs area and area costs money, it's a question about cost as well. That question is the subject of AD102, and once you can answer it, the rest is arithmetic.

Animation: resistor geometry

This clip runs for 192 seconds and has no sound; the captions carry the narration. It covers this page and the next two (the symbol, the strip, ohms per square, and what 100 kΩ costs in silicon), so come back to it after you've read those as well.



Your browser will not play embedded video — download the clip.

Source: `tooling/manim/scenes/ad102_resistor_geometry.py`, the `ResistorGeometry` scene. Every number shown in it is one this course measured itself, and the two footprints in the closing comparison are drawn to the same scale.

Before you press play, make a prediction. A strip of poly 40 µm long and 2 µm wide has some resistance. If you redraw it 40 µm long and 4 µm wide, does the resistance go up, go down or stay the same, and by how much? Write your number down, because the animation counts it out on screen and you can check it there.

What the resistor is made of

Without its packaging, a resistor is a piece of material that current has to squeeze through. A long, thin piece has a high resistance, and a short, wide piece has a low one. The through-hole resistor you soldered works the same way: inside it is a spiral of resistive carbon film cut into a ceramic rod, and the spiral is cut longer to give more ohms.

On a chip, the resistive material is a **strip of doped silicon or polysilicon**, which you draw as a rectangle on one mask layer. Current goes in at one end and out at the other, and three things set the value:

1. What the strip is made of, which comes down to how heavily it was doped. You don't choose this; the foundry's recipe fixes it.
2. How long you draw it.
3. How wide you draw it.

You control the length and the width directly, but for the material you can only choose from a menu that the foundry provides.

The SKY130 resistor menu

Here is that menu, taken from the header comment of the Magic technology file, `/foss/pdks/sky130A/libs.tech/magic/sky130A.tech`:

```
# sky130_fd_pr__res_generic_nd   rdn      n+ diff resistor
# sky130_fd_pr__res_generic_pd   rdp      p+ diff resistor
# sky130_fd_pr__res_generic_l1   rli      local interconnect resistor
# sky130_fd_pr__res_generic_po   npres    n+ poly resistor
# sky130_fd_pr__res_high_po_*    ppres    p+ poly resistor (300 Ohms/sq)
# sky130_fd_pr__res_xhigh_po_*   xres     p+ poly resistor (2k Ohms/sq)
# sky130_fd_pr__res_iso_pw       rpw      pwell resistor (in deep nwell)
```

Those seven entries are seven materials rather than seven values. The two `p+ poly` entries, `res_high_po` and `res_xhigh_po`, are on the same layer with the same drawing rules, and they differ by a factor of about 6.7 only because they get a different implant dose. That factor is the only choice of value the foundry gives you; everything else you get by drawing.

`res_high_po` and `res_xhigh_po` are also the **precision** resistors. SkyWater characterised them carefully because analog designers build references out of them, and they're the two this course uses.

The same strip in four materials

Suppose you draw one strip, 314.046460 µm long and 1 µm wide, and change nothing but the resistor model it uses. Section C of `spice/r_head_and_body.spice` does exactly that:

```
--- C. one 314 um x 1 um strip, four implants (ohms) ---
r_high = 1.000000e+05
r_xhigh = 6.653872e+05
r_pdiff = 6.138915e+04
r_ndiff = 3.689775e+04
```

model	value of that one strip
`sky130_fd_pr__res_high_po`	**100.0000 kΩ**
`sky130_fd_pr__res_xhigh_po`	665.3872 kΩ
`sky130_fd_pr__res_generic_pd` (p+ diffusion)	61.38915 kΩ
`sky130_fd_pr__res_generic_nd` (n+ diffusion)	36.89775 kΩ

Every row has the same drawing and the same silicon area, and yet the values spread over a ratio of 18:1. Choosing the material is the first design decision you make, and it's the only one that doesn't cost area. You'll run this deck yourself in Lab 01, Part 6.

When you do, the run first prints a block of model warnings (see Getting started). One of them, the `rbody_model` warning, shows `rsh` as `3.173885e+02`. That number is the sheet resistance of `res_high_po` in ohms per square, which is the most important parameter of the device, and it sits in a warning you would normally scroll past.

Writing W and L

As Getting started explained, `W` and `L` on a SKY130 device are plain numbers in microns, with no unit suffix:

```
Xa a 0 0 sky130_fd_pr__res_high_po W=1 L=10       <- one micron by ten microns
Xa a 0 0 sky130_fd_pr__res_high_po W=1u L=10u     <- a millionth of that, in both
```

Every deck in this course uses the first form. The second multiplies both numbers by 10^{-6}, which puts the device outside every bin its model was fitted over. On a MOSFET that stops the run with `could not find a valid modelname`, but on a resistor the run carries on, which is what makes the mistake so hard to catch: the strip that reads 3550.443 Ω when it's written `W=1 L=10` reads 3193.812 Ω when it's written `W=1u L=10u`, an ordinary-looking resistance 10 % off, and ngspice still exits with status 0. Apart from Lab 01's `u_trap.spice`, which exists to show you the mistake, no deck in this course has a `u` in it, and none of yours should either.

Next: Ohms per square, the unit that turns a rectangle into ohms.

4. Ohms per square

This page explains the unit "Ω/□", and why chip designers work with it instead of with resistivity.

Start from the physics you already know. A block of material with resistivity ρ, length L and cross-section A has a resistance of

$$R = \rho \frac{L}{A} \tag{4.1}$$

On a chip the block is a thin film, and its thickness t was fixed by the foundry. You can't change it, and it's the same for every device on the wafer, so it makes sense to write the cross-section as width × thickness, $A = Wt$:

$$R = \rho \frac{L}{Wt} = \left(\frac{\rho}{t}\right) \frac{L}{W} = R_{\square} \frac{L}{W} \tag{4.2}$$

That gathers the two things you can't control, ρ and t, into one number:

$$R_{\square} = \frac{\rho}{t} \qquad \text{units:} \quad \frac{\Omega\,\mathrm{m}}{\mathrm{m}} = \Omega \tag{4.3}$$

$R_{\square}$ is called **sheet resistance** and is quoted in "ohms per square", written **Ω/□**. Its dimensions are plain ohms. The "per square" doesn't mean you divide by anything; it's a reminder of what the number describes, which the next section explains.

Why it is called "per square"

If you set $L = W$ in $R = R_{\square} L/W$, the width cancels:

$$R = R_{\square} \tag{4.4}$$

A square of the material has the same resistance no matter how big the square is. A 1 µm × 1 µm square of poly and a 500 µm × 500 µm square of the same poly have identical resistance. Making the square wider adds parallel paths, making it longer in proportion adds series length, and the two effects cancel exactly.

So L/W is the **number of squares you drew end to end**. A strip 40 µm long and 2 µm wide is twenty squares in series, and that stays true whether you fold it, bend it, or run it across the whole die.

The unit is useful because it turns an electrical question into a counting question, and counting squares is something you can do by looking at a drawing.

The animation on Resistors on a chip shows this happening. It draws the squares onto a strip and numbers them, then stretches the strip and widens it, and you can watch the count and the resistance move together. Then it grows the length and width at the same time, so the count stays at one square and the resistance stays at 317 Ω while the shape grows ten-fold. (That 317 Ω is the sheet resistance alone; the contacts add to it, as the head-and-body section below shows.) If counting squares still seems odd, watch that part of the animation.

Sheet resistance of the SKY130 layers

The sheet resistance of every layer ships with the PDK. These numbers come from `/foss/pdks/sky130A/libs.tech/magic/sky130A.tech`, the technology file for Magic (the layout tool you'll use in AD104). The block that lists them starts with the comment *"# Resistances are in milliohms per square"*, and here it is, abridged:

```
resist (pwell,isosub)/well     4400000
resist (nwell)/well             950000
resist (*ndiff,nsd)/active      120000
```

```
resist (*pdiff,*psd)/active     197000
resist mrp1/active               48200
resist xhrpoly/active           319800
resist uhrpoly/active          2000000
resist (allli)/locali            12800
resist (allm1)/metal1              125
resist (allm3)/metal3               47
resist (allm5)/metal5               29
resist mrdl/metali                   5
```

Divided by 1000 to give ohms per square, and sorted from highest to lowest:

layer	$R_\circ$ (Ω/□)	ratio to met1
p-well	4400	35 200
`uhrpoly`, the `res_xhigh_po` implant	2000	16 000
n-well	950	7 600
`xhrpoly`, the `res_high_po` implant	319.8	2 558
p+ diffusion	197	1 576
n+ diffusion	120	960
poly (ordinary, silicided)	48.2	386
local interconnect (`li`)	12.8	102
met1, met2	0.125	1
met3, met4	0.047	0.38
met5	0.029	0.23
redistribution layer	0.005	0.04

That's a range of nearly six orders of magnitude on one chip, from 4400 Ω/□ down to 0.005 Ω/□. Met5 at 0.029 Ω/□ and n-well at 950 Ω/□ differ by a factor of 32 759.

> **The ngspice models have their own numbers.** The model files at `/foss/pdks/sky130A/libs.tech/ngspice/r+c/res_typical__cap_typical.spice` carry their own copies of these values. `rdn=120`, `rdp=197`, `rp1=48.2` and `rm1=0.125` agree exactly with Magic's table, but `rl1=12.2` (not 12.8), `rm5=0.0285` (not 0.029) and `rnw=1700` (not 950) don't. Magic's table is what the *extractor* uses when it reads your layout, and the ngspice file is what the *simulator* uses, so when the two disagree, your simulation results came from the ngspice number. Disagreements like this are a normal part of reading a PDK, and they don't mean that something is broken.

Contact resistance

Current has to get into the strip, and it does that through a **contact cut**: a small hole in the insulator, filled with metal. The same Magic file lists the resistance of each cut:

contact	Ω each
`ndc`, metal to n+ diffusion	185
`pdc`, metal to p+ diffusion	585
`pc`, metal to poly	152
`mcon`, li to met1	9.3
`via`, met1 to met2	4.5

contact	Ω each
via2, via3	3.41
via4, met4 to met5	0.38

A single contact to p+ diffusion costs 585 Ω, which is nearly two squares of res_high_po material. That's why a real resistor device has a wide **head** at each end, with room for a row of contacts in parallel.

Head and body of one resistor

sky130_fd_pr__res_high_po is a .subckt, and inside it a node called rb sits between the contact head and the resistive body. If you probe that node, you can split one measured resistance into its two parts.

Start with a first guess. The sheet resistance is 319.8 Ω/□ (from Magic) or 317.3885 Ω/□ (the rsheet parameter on the ngspice model card). A strip drawn W = 1 µm, L = 1 µm is one square, so it should measure about 318 Ω, and the L = 10 µm strip should be ten times that, about 3180 Ω.

Here is what labs/passives-decks/spice/r_head_and_body.spice prints:

```
--- A. one strip, split into head and body (ohms) ---
r_l1 = 6.954646e+02
head_l1 = 2.998915e+02
body_l1 = 3.955731e+02
r_l10 = 3.550443e+03
head_l10 = 2.998915e+02
body_l10 = 3.250551e+03
```

Both guesses are wrong, and the six lines show why. One square of res_high_po measures 695.4646 Ω, more than twice the sheet resistance, because 299.8915 Ω of that is in the head rather than the body. (The one-square body itself is 395.5731 Ω rather than 317 Ω, because the model makes the body slightly longer than you drew it; the next section has the numbers.)

The head at L = 10 µm is also 2.998915e+02, the same to seven digits, because making the strip longer didn't change the contacts. The head is a fixed cost that you pay once per resistor.

The body is the part that scales with length, so the difference between the two bodies, divided by the difference in length, gives the resistance of each extra square:

$$\frac{3250.551 - 395.5731}{10 - 1} = \frac{2854.978}{9} = \boxed{317.2198\ \Omega/\square} \tag{4.5}$$ (4.6)

That's the sheet resistance as the simulator measures it, and it's within 0.06 % of the rsheet = 317.3885 on the model card. You'll run this deck yourself in Lab 01, Part 6.

The design equation

Rearranged into the form you'll design with, the resistance of a strip with W = 1 µm is

$$R(L) = 378.2448 + 317.2198\,L \qquad (L \ \text{in µm}) \tag{4.7}$$

The 378.2448 Ω intercept is the head plus a small, fixed extension of the body that the model adds. The card says leff = l + 0.247, which means the strip is electrically 0.247 µm longer than you drew it. Lab 01 extracts both constants from a different deck, one with 22 strips of different lengths and widths, and gets 317.2198 ohm/square 378.2448 ohm of ends, the same two numbers. Lab 01, Part 4 also does the fit at other widths, where the per-micron slope and the end resistance both change.

Here is the equation checked against the simulator:

L (µm)	$378.2448 + 317.2198L$	ngspice
1	695.4646	695.4646
10	3550.443	3550.443
50	16 239.23	16 239.23

Before you trust any on-chip resistor value, ask what the ends cost, and make a habit of it. At L = 1 µm the ends are 54 % of the device, and at L = 100 µm they are 1.2 %, so if your resistor is short, you're mostly paying for contacts.

Using sheet resistance

Sheet resistance is what lets a layout engineer look at a drawing and say "that's about 3 kΩ" without a simulator: they count the squares and multiply by the number from the table. It's also the reason a wire has a resistance you have to take into account. A 1000 µm run of minimum-width met1 is 1000 / 0.14 = 7143 squares, and 7143 squares × 0.125 Ω/□ = **893 Ω**, which is a real resistor in your signal path whether you drew one there or not.

Next: What a resistor costs in area, where you find out how much silicon a 100 kΩ resistor occupies.

5. What a resistor costs in area

If a resistance on a chip is just a length of material, it's fair to ask how long a real one has to be, and whether that length is a problem. This page works out the numbers.

Ohms per square gives the resistance of `sky130_fd_pr__res_high_po`, drawn 1 µm wide, as a function of its length:

$$R(L) = 378.2448 + 317.2198\,L \qquad (L \ \text{ in µm}) \tag{5.1}$$

If you solve that for L, it becomes a design procedure: to get a resistance R, you draw a length

$$L = \frac{R - 378.2448}{317.2198} \tag{5.2}$$

Three values, sized and simulated

Section B of `labs/passives-decks/spice/r_head_and_body.spice`, the same deck Ohms per square uses, sizes three resistors this way and simulates them:

you want	L from the formula	ngspice says
1 kΩ	1.960014 µm	1000.000 Ω
10 kΩ	30.331509 µm	10 000.00 Ω
100 kΩ	314.046460 µm	100 000.0 Ω

```
--- B. sized to 1 k, 10 k, 100 k (ohms) ---
r_1k = 1.000000e+03
r_10k = 1.000000e+04
r_100k = 1.000000e+05
```

The formula and the simulator agree, and the last row says that a 100 kΩ resistor is a strip of polysilicon 314 micrometres long.

To get a sense of scale, compare it with the smallest inverter in the SKY130 standard-cell library, `sky130_fd_sc_hd__inv_1`, which has `area : 3.7536` µm² in `libs.ref/sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib`. Your 100 kΩ resistor is 314.046460 × 1 µm = 314.05 µm² of drawn poly, which is the silicon of **84 inverters**. So a passive component you wouldn't think twice about on a breadboard costs as much floor space as a small logic block.

Larger values get expensive quickly, because the length grows in proportion to the resistance. A 1 MΩ resistor in the same material at the same width needs

$$L = \frac{10^6 - 378.2448}{317.2198} = 3151.2\ \text{µm} = 3.15\ \text{mm} \tag{5.3}$$

which is longer than many whole dies. Folding (later on this page) won't rescue a resistor that size, so instead you redesign the circuit so that it doesn't need one.

Width and area

How every on-chip resistor gets drawn follows from rearranging the resistance formula. For a fixed resistance, the length you need is proportional to the width, so

$$L \propto R\,W \quad \Rightarrow \quad \text{area} = W\,L \propto R\,W^2 \tag{5.4}$$

and the area goes as the square of the width. If you draw a resistor twice as wide, it has to be twice as long to keep the same resistance, so you pay for four times the silicon. Section D of the deck builds the same 100 kΩ at four more widths to check this:

```
--- D. 100 k at four widths (ohms), and the area each one costs ---
r_w0p35 = 9.999975e+04
r_w0p69 = 1.000002e+05
r_w2 = 9.999982e+04
r_w5 = 1.000004e+05
```

W (µm)	L (µm)	ngspice (Ω)	area WL (µm²)	in `inv_1s`
0.35	101.9319	99 999.75	**35.68**	9.5
0.69	216.2792	100 000.2	149.23	39.8
1	314.046460	100 000.0	314.05	83.7
2	629.5313	99 999.82	1 259.06	335.4
5	1576.1560	100 000.4	7 880.78	2 099.5

Every row gives the same value to five digits, yet the bottom row takes 221 times the area of the top row. The square law on its own predicts $(5/0.35)^2 = 204$. Most of the rest comes from the 0.35 µm strip, which the model treats as slightly narrower than drawn, so it has about 7 % more ohms per square than the wider ones and needs less length (Lab 01, Part 4 measures this). Its larger end resistance shortens it a little more.

So draw a resistor at the narrowest width that the design rules and your accuracy budget allow. For `res_high_po`, the narrowest width SKY130 permits is 0.35 µm. Make it wider only when you have a reason to, such as better matching (Matching and Pelgrom's law) or carrying more current.

Changing the material

You choose the width and length freely, but the material has to come from the short list the process offers. Section C of the deck draws the same 314.046460 × 1 µm strip with four different implants:

model	value	Ω/□ (measured)
`res_generic_nd` (n+ diffusion)	36.89775 kΩ	117.5
`res_generic_pd` (p+ diffusion)	61.38915 kΩ	195.5
`res_high_po`	100.0000 kΩ	317.2198
`res_xhigh_po`	665.3872 kΩ	2118.7619

`res_xhigh_po` gives 6.7 times as many ohms per micron of length, so the 1 MΩ resistor shrinks from 3.15 mm to

$$L = \frac{10^6 - 34.2111}{2118.7619} = 472.0\ \mu\text{m} \tag{5.5}$$

That's still big (472 µm² of poly at 1 µm wide, or 126 inverters), but it fits on a die.

> **Why 2118.7619 and not 2000.** The `res_xhigh_po` model card says `rsheet = 2000.0`, but it also says `weff = {w-0.056}`, which means the strip comes out of the fab 56 nm narrower than you drew it, because the etch undercuts the edge. At a drawn W = 1 µm the material is really 0.944 µm wide, so the resistance per micron of length is $2000/0.944 = 2118.6441$ Ω. Lab 01 measures 2118.7619, which agrees to four digits; the small remainder is the model's nonlinear body term. After `leff` on Ohms per square, this is the second place in the course where what you drew and what you got differ by an amount you can calculate, and Process corners is about that gap.

Folding a long resistor

A strip 314 µm long and 1 µm wide doesn't fit anywhere sensible on a die, so real layouts **fold** it. You might draw sixteen strips of 20 µm side by side, for example, and join them end to end with short turns of the same poly, so the current runs back and forth like a plough across a field. Folding shifts the value, though. The straight runs still count square for square, but the current cuts across the inside of each corner, so a corner square counts as only roughly half a square, and you have to re-tune the length to bring the folded strip back to the value you sized it for. Folding also makes you pay for:

- the required spacing between neighbouring poly strips;
- a guard ring, if the resistor needs isolating from its neighbours;
- **dummy strips** at each end, which are extra resistors connected to nothing. Etching behaves differently at the edge of a pattern than in the middle, so a dummy gives the outermost real strip the same surroundings as the inner ones, and the edge effect falls on a device nobody uses.

Drawn straight, a resistor is the rectangle the formula describes: a bar of poly with one contact at each end.



Figure 5.1: A resistor drawn as a single straight bar of poly, with a metal contact at each end and a boundary drawn around it

Folded, it's the same bar laid back and forth. The poly itself turns each corner, and there are still only the two contacts, one at each end:

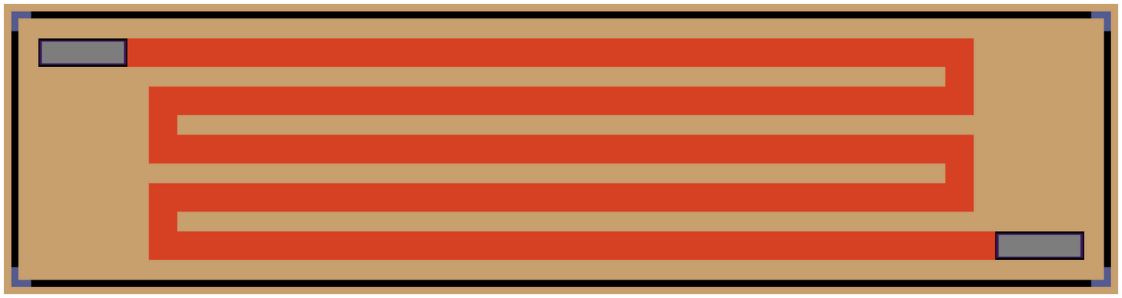


Figure 5.2: The same resistor folded into five parallel passes, joined at alternating ends, with the two contacts now at opposite corners

This one has five passes rather than the sixteen the 100 kΩ above would need, since the figure is there to show the shape, not the count. The red is the resistor body, and the grey blocks at the ends are its contacts. Each turn you add buys you height and costs you two corners.

A folded 100 kΩ resistor with its guard ring and dummies typically occupies two to three times its 314 µm² of drawn poly. Nobody publishes an exact multiplier, because it depends on the number of folds and on the spacing rules. The spacing alone goes a long way: the SKY130 device generator that Magic uses (`/foss/pdks/sky130A/libs.tech/magic/sky130A.tcl`) keeps 0.48 µm between neighbouring `res_high_po` strips and 0.48 µm between a strip and its guard ring, so in a fold of 1 µm strips every strip brings a gap half its own width.

If you'd like to see the layers in "a strip of poly with two contacts", you can draw one in SiliWiz, a layout tool that runs in the browser and shows the cross-section as you draw. That's the optional Lab 01, Part 9.

How this shapes analog circuits

These numbers shape every analog block you'll read about. Bias networks use ratios of small resistors instead of one big one, and bandgap references use a handful of matched units rather than one exact value.

Filters that would need a megohm on a breadboard are built from switched capacitors instead (What to build instead of an inductor). Designers work this way because a 100 kΩ resistor costs as much silicon as 84 inverters.

Next on the Course path: Lab 01: A resistor you designed. After that, Capacitors on a chip asks the same question about capacitors, and the area bill is much bigger.

6. Lab 01: A resistor you designed

You'll measure the sheet resistance and end resistance of a SKY130 poly resistor from your own simulation, then use those two numbers to size two resistors that land within 1 % of their targets. Along the way you'll work out what a resistance costs in silicon compared with a piece of digital logic.

Package: `labs/lab-01-a-resistor-you-designed/`

Before you start

- Read Getting started, Resistors on a chip, Ohms per square and What a resistor costs in area.
- You need the workbench from IC101: the noVNC desktop comes up in your browser and the IC101 smoke test passes. Part 1 loads this course into it.
- The only circuit theory you need is Ohm's law and a voltage divider.

Parts 1 to 7 are required. Everything under Going further, at the end, is optional.

Time: about 2.5 hours, not counting the optional parts. Each ngspice run is silent for anywhere from about 10 seconds to just over a minute before it prints anything, depending on your machine (Warnings and delays).

Part 1: Load the course

Can you reach the AD102 files, on the tool versions this course's printed outputs come from? If you haven't set up the workbench yet, do IC101 Lab 03 first.

1. On your laptop, from your `workspace` clone, start the desktop (`./scripts/start_vnc.sh`, or `scripts\start_vnc.bat` on Windows), open http://localhost/ and open a terminal on the desktop. Type every other command in this course in that terminal, not on your laptop.
2. Load the course environment and get the course files. The first time, type all four lines; after that, `mod ad102` on its own takes you back to the course.

```
. /foss/designs/common/.designinit    # a new terminal usually does this for you; again is harmless
echo "PDK is: $PDK"
mod add ad102                    # first time only: clones github.com/uoftasic/ad102
mod ad102                            # every time after that
```

You should see the `ASIC-EDU: PDK=sky130A ...` banner, `PDK is: sky130A`, then `OK  module ready: /foss/designs/modules/ad102` from `mod  add`, then `cwd:  /foss/designs/modules/ad102` from `mod ad102`. `mod` and the course files have to work; if they don't, ask for help before going on. If `$PDK` says something other than `sky130A`, ask too, but carry on: every lab deck names its model files by absolute path, so `make` works whatever `$PDK` says. Only XSchem, in the optional last part of Lab 03, reads it (Getting started explains).

If you see	It means
`bash: mod: command not found`	This terminal has not loaded the course environment. Run the `.designinit` line again in it.
`mod: no such module: ad102`	You ran `mod ad102` before `mod add ad102`.
`fatal: could not read Username for 'https://github.com'`	git couldn't read the repo anonymously and is asking you to log in. Check that you typed `ad102`. If you did, ask in Discord. Do not type your password.

For anything else, such as a blank desktop page, see IC101's Troubleshooting page.

3. Check the two tool versions this course's numbers were recorded with:

```
ngspice -v | head -3
xschem -v | head -1
```

You should see `** ngspice-47 : Circuit level simulation program` on the second line and `XSCHEM V3.4.8RC`. If you see different versions, you are on a different image, and commands on these pages may fail in ways this course does not describe; ask in Discord before you go on.

4. Go to this lab's folder and list the packages:

```
cd /foss/designs/modules/ad102/labs/lab-01-a-resistor-you-designed
ls ..
```

You should see: `ls ..` lists the four lab packages plus `passives-decks`:

```
lab-01-a-resistor-you-designed     lab-03-a-time-constant-in-silicon  passives-decks
lab-02-how-big-is-a-picofarad      lab-04-two-resistors-that-disagree
```

Part 2: Measure the material

Does the one number on the model card predict what a strip of `res_high_po` measures?

`spice/sheet.spice` builds twenty-two strips of poly, forces exactly 1 µA through each, and prints the voltage across it. Dividing that voltage by 1 µA gives the resistance, so each number is one application of Ohm's law to one device, with no curve fitting. The first block is a **length ladder** of `sky130_fd_pr__res_high_po`, with W fixed at 1 µm and L going from 1 µm to 100 µm.

1. Read the one number the foundry advertises on the model card:

```
grep                rsheet                /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__res_high_po.model.spice
```

```
+ rsheet = 317.3885
+ rsh = {rsheet*(1+sky130_fd_pr__res_high_po__var)*(1+res_match)}
```

The first line is the number; the second shows how the model uses it.

2. On paper, predict what the seven strips in the length ladder will measure. At $W = 1$ µm, L/W is just L, so you are predicting 317.3885 × 1, × 2, × 5, × 10, × 20, × 50 and × 100.

3. Run the lab:

```
make
```

This runs two ngspice decks, `spice/sheet.spice` and `spice/my_resistor.spice`, and then the checker script, `src/check_res.py`. It ends in `FAIL`, which you fix in Part 7. Leave the output in your terminal, because Parts 2 to 5 read from it. If it scrolls away, `grep -E '^---|^r_' results/sheet.log` prints the deck's blocks again, and `make check` prints the checker's part.

4. Find block A in the output and compare it with your predictions:

```
--- A. length ladder, W = 1 um (ohms) ---
r_l1 = 6.954646e+02
r_l2 = 1.012684e+03
r_l5 = 1.964344e+03
r_l10 = 3.550443e+03
r_l20 = 6.722641e+03
```

```
r_l50 = 1.623923e+04
r_l100 = 3.210022e+04
```

How far off are your predictions for the longest and the shortest strip, and what does the difference tell you?

Answer: A1, in Answers at the end.

5. Each extra micrometre of length adds the same resistance (for example, $(1012.684 - 695.4646)/1$ and $(1964.344 - 1012.684)/3$ both come to 317.22 Ω), so R is a straight line in L. The line doesn't pass through the origin, though, so you need to write it with an intercept:

$$R = R_{\text{end}} + R_{\square}\frac{L}{W} \tag{6.1}$$

Use the two extremes, $L = 1$ and $L = 100$, to solve for both unknowns. Write them down.

6. Check your line against two measurements you did not use, at $L = 10$ and $L = 20$.

You should see:

$$R_{\square} = \frac{32100.22 - 695.4646}{100 - 1} = 317.2198\Omega/\square \qquad R_{\text{end}} = 695.4646 - 317.2198 = 378.2448\Omega \tag{6.2}$$

At $L = 10$ the line gives $378.2448 + 3172.198 = 3550.4428$, against a measured 3550.443; at $L = 20$ it gives 6722.6408, against 6722.641. It matches every point to the last printed digit, across a 100-to-1 range of lengths. The checker prints the same extraction, with the course's own reference run in the last column (the `xhigh_po` row is the other material, which you extract yourself in Part 5):

```
  the two numbers that define the device
    device          ohm/square    end ohms                reference
    high_po           317.2198    378.2448      317.2198 / 378.2448
    xhigh_po         2118.7619     34.2111      2118.7619 / 34.2111
```

The 378 Ω is a fixed overhead on every resistor, the same whatever length you draw; Part 6 finds out where it comes from. What fraction of a 100 kΩ resistor is it, and of a 500 Ω one?

Answer: A2, in Answers at the end.

Part 3: Size a 10 kΩ resistor

How much does leaving out the end resistance cost on a real design?

`spice/sheet.spice` already contains two 10 kΩ designs at $W = 1$ µm (block B).

1. Work out the naive length, using only the sheet resistance on the card: $L = 10000/317.3885 = 31.5074$ µm. This is `r_naive` in the deck.
2. Work out the corrected length, $L = (10000 - R_{\text{end}})/R_{\square}$, from the two numbers you extracted, and write it down. The deck's version of this design is `r_fixed`.
3. Put the naive length into your line from Part 2 and predict how far from 10 kΩ it lands. Then find block B in the output.

You should see:

```
--- B. the 10 kohm design, naive vs fixed (ohms) ---
r_naive = 1.037302e+04
r_fixed = 9.999997e+03
```

The naive design is 10,373 Ω, 3.7 % too big. The deck draws `r_fixed` at 30.3315 µm (check your length against it), and it lands at 9999.997 Ω, within three parts in ten million of the target.

An error of 3.7 % sounds small, so why is it worth fixing?

Answer: A3, in Answers at the end.

Part 4: Change the width

Is ohms per square a property of the material alone, and what does a narrow strip buy you?

The length ladder was all at $W = 1$ µm. Blocks C and C2 of the `make` output from Part 2 sweep four other widths at two lengths, $L = 10$ µm and $L = 5$ µm:

```
--- C. width ladder, L = 10 um (ohms) ---
r_w0p35 = 1.067497e+04
r_w0p69 = 5.123000e+03
r_w2 = 1.785180e+03
r_w5 = 7.167730e+02
--- C2. the same four widths at L = 5 um (ohms) ---
r_v0p35 = 5.816766e+03
r_v0p69 = 2.823272e+03
r_v2 = 9.925276e+02
r_v5 = 3.998070e+02
```

1. For each width, fit a slope (Ω per µm of length) and an intercept (end Ω) from the two lengths, the same way you did in Part 2. That is two subtractions per width.
2. Multiply each slope by its width to get Ω per square.
3. For each width, work out the body area that 1 kΩ needs: $W \times 1000/(\Omega \text{ per µm})$. That is the last column of the table below.

You should see: (the $W = 1$ µm row comes from the length ladder in Part 2)

W (µm)	Ω per µm of length	end Ω	Ω per square	µm² of body per kΩ
0.35	971.641	958.562	340.074	0.3602
0.69	459.946	523.544	317.362	1.5002
1	317.220	378.245	317.220	3.1524
2	158.530	199.875	317.061	12.6159
5	63.393	82.841	316.966	78.8728

Read the Ω-per-square column from the bottom up. Where does it stop being constant, and why?

Answer: A4, in Answers at the end.

Now read the last column: 0.36 µm² per kΩ at the narrowest width against 78.9 at the widest, a factor of 219 for the same material. Why is that so much more than the ratio of the widths?

Answer: A5, in Answers at the end.

Part 5: Change the recipe

What does the other poly implant change, and what does a resistor cost next to a piece of digital logic?

Block D repeats the length ladder on `sky130_fd_pr__res_xhigh_po`, at five of the lengths (1, 2, 10, 50 and 100 µm): the same layer and drawing rules, with a different implant dose.

```
--- D. res_xhigh_po, W = 1 um (ohms) ---
r_x1 = 2.152973e+03
r_x2 = 4.271620e+03
r_x10 = 2.122088e+04
r_x50 = 1.059694e+05
r_x100 = 2.119104e+05
```

1. Extract $R_\square$ and R_{end} from block D, using the two extremes as in Part 2.
2. Compare with what its model card advertises:

```
grep               rsheet               /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__res_xhigh_po__base.model.spice
```

```
.param  rsheet = 2000.0
+ rbody = {(l*rsheet+rbody_var)*(1+res_match)/w}
```

As in Part 2, the first line is the number and the second shows how the model uses it.

3. Price your resistors in digital logic. In DD103, a 4-bit ripple-carry adder (twelve SKY130 standard cells that add two nibbles) synthesises to 110.11 µm². At $W = 1$ µm a strip's area is its length in µm². With the formula from Part 3, step 2, $L = (R - R_{\text{end}})/R_{\square}$, work out the length of a 1 kΩ, 10 kΩ and 100 kΩ `res_high_po` and a 100 kΩ `res_xhigh_po`, all at $W = 1$ µm, then divide each area by 110.11.

You should see: the card says `rsheet = 2000.0`. Your extraction says 2118.7619 Ω/□, 5.9 % higher, with only 34.21 Ω of ends. Compared with `res_high_po`, that is 6.7 times the ohms per square and a tenth of the fixed overhead. Your price list should match this one:

you drew	area	in adders
1 kΩ, `res_high_po`, W = 1 µm	1.96 µm²	0.018
10 kΩ, `res_high_po`, W = 1 µm	30.33 µm²	0.28
100 kΩ, `res_high_po`, W = 1 µm	314.05 µm²	2.85
100 kΩ, `res_xhigh_po`, W = 1 µm	47.18 µm²	0.43

The checker prints three of these lengths too, to four decimals, in the block headed `and the design that follows from them`: 30.3315 µm for 10 kΩ, and 314.0465 µm and 47.1812 µm for the two 100 kΩ rows. The same block has rows you don't need here: 4.7036 µm for 10 kΩ of `res_xhigh_po`, and a 2200 Ω row for each material. You size a 2200 Ω resistor yourself in Part 7; work it out from your own two numbers rather than copying that row.

The card says 2000.0 and your simulation says 2118.7619. Which one do you design with, and why?

Answer: A6, in Answers at the end.

The areas above are the strip only. A drawn resistor also needs its two contact heads, spacing to whatever is beside it, and usually a guard ring, so treat them as a lower bound rather than a quote: What a resistor costs in area puts a folded 100 kΩ, with its guard ring and dummies, at two to three times its drawn poly. Even so, at 100 kΩ one of the cheapest things in the analog toolbox outweighs a working arithmetic unit. That is why analog blocks look enormous next to digital ones in a die photograph, and why a chip designer never answers "just add a resistor" straight away.

Part 6: Split a resistor into head and body

Which part of the resistor is the 378 Ω you extracted: the contact head, or the body?

Resistors on a chip and Ohms per square quote a different deck from the one you ran in Part 2: `labs/passives-decks/spice/r_head_and_body.spice`. `sky130_fd_pr__res_high_po` is a subcircuit (`.subckt`), and inside it a node called `rb` sits between the contact head and the resistive body. This deck measures the voltage at that node too, so one measured resistance splits into its two parts.

1. Go to the `passives-decks` package and run the resistor deck:

```
cd ../passives-decks
make resistor
```

It runs one deck and prints its blocks, and there's no `PASS` or `FAIL` at the end.

2. Before you look at block A, predict: when the strip goes from $L = 1$ µm to $L = 10$ µm, which part changes, the head or the body?

3. Read block A:

```
--- A. one strip, split into head and body (ohms) ---
r_l1 = 6.954646e+02
head_l1 = 2.998915e+02
body_l1 = 3.955731e+02
r_l10 = 3.550443e+03
```

```
head_l10 = 2.998915e+02
body_l10 = 3.250551e+03
```

4. Work out the slope of the body from `body_l1` and `body_l10`.
5. Go back to the lab folder:

```
cd ../lab-01-a-resistor-you-designed
```

You should see: the head is 299.8915 Ω at both lengths, the same to seven digits, because you did not change the contacts. The bodies scale:

$$\frac{3250.551 - 395.5731}{10 - 1} = \frac{2854.978}{9} = 317.2198\ \Omega/\square \tag{6.3}$$

That's the same sheet resistance you extracted in Part 2, this time from a different deck.

The head is 299.89 Ω, but the end resistance you extracted in Part 2 is 378.2448 Ω. Where are the other 78 Ω?

Answer: A7, in Answers at the end.

Part 7: Size your own resistors

Can you size two resistors from your own two numbers so that each lands within 1 % of its target?

`spice/my_resistor.spice` contains two devices and two `L=` numbers. You edit only those two numbers.

	device	W	target
R1	`sky130_fd_pr__res_high_po`	1 µm	2.2 kΩ
R2	`sky130_fd_pr__res_xhigh_po`	1 µm	50 kΩ

The file ships with the naive lengths, so the check at the end of Part 2's `make` failed:

```
  R1  sky130_fd_pr__res_high_po  W=1
    target     2200.0 ohm
    got        2577.1 ohm   (+17.14 %)   FAIL
  R2  sky130_fd_pr__res_xhigh_po W=1
    target    50000.0 ohm
    got       53001.1 ohm   (+6.00 %)   FAIL
```

1. R1 came out 17.14 % high. Compare that with the 378 Ω of ends as a fraction of 2.2 kΩ. What does it tell you?

   Answer: A8, in Answers at the end.

2. Work out both lengths from the $R_\square$ and R_{end} you extracted for each device in Parts 2 and 5, and write them down.
3. Compare yours with the checker's. Print its report again (the Part 2 output has probably scrolled away):

   ```
   make check
   ```

   Below the two `FAIL` lines, the checker explains each failure and ends the explanation with the length it would use. Its last lines note that on a fresh clone this `FAIL` is expected and doesn't mean the package is broken, because fixing it is the lab.
4. Edit the two `L=` numbers in `spice/my_resistor.spice` with any of the desktop's editors (`gedit`, `nano` and `vim` are installed; for example `gedit spice/my_resistor.spice &`). Write them as plain microns, with no `u` suffix, to four decimals like the numbers already there. The file marks the two lines to change:

   ```
   X1 r1 0 0 sky130_fd_pr__res_high_po  W=1 L=6.9315
   *                                       ^^^^^^ R1: 2.2 kohm goes here
   ```

```
X2 r2 0 0 sky130_fd_pr__res_xhigh_po W=1 L=25.0000
*                                        ^^^^^^^ R2: 50 kohm goes here
```

5. Re-run your file and the checker. `make mine` simulates only `spice/my_resistor.spice` and `make check` prints the checker's report again, so you can do both in one command:

```
make mine check
```

You should see: both resistors marked `PASS` (each must land within 1 % of its target), then

```
PASS -- every number matches the reference run.
```

Finishing

You're done when `make mine check`, run on the two lengths you worked out in Part 7, prints `PASS -- every number matches the reference run.`

That line needs each of your resistors within 1 % of its target. The checker also compares the resistances from `sheet.spice` with the reference run (within 0.5 %), and the two constants extracted from them (within 1 %). A DC operating point has no randomness in it, so on the pinned image (ngspice-47, `sky130A`, `tt` corner, 27 °C) your numbers match the ones on this page to every digit shown.

Question: on area alone, which material would you use for a 100 kΩ resistor, and what does it cost in adders?

Answer: A9, in Answers at the end.

After your own numbers pass, read `solutions/README.md` for the reference sizing and some open questions.

Submit: nothing for this lab.

Going further (optional)

None of these is in the time estimate, and no later part needs them. What to do next on the Course path is the last line of this page.

End resistance from the model card

Open the model card from Part 2, step 1, `sky130_fd_pr__res_high_po.model.spice`. The body is declared `l = {leff}` with `leff = {l + 0.247}`, so the strip is electrically 0.247 µm longer than you drew it, worth $317.2198 \times 0.247 = 78.35\ \Omega$. The head is a second resistor, `rsh = 345.8312` over a width of `weff + 0.1558 = 1.1548` (`weff` is the effective width, 0.999 µm for a strip drawn 1 µm wide; the first answer in Part 4 has more on it), giving $345.8312/1.1548 = 299.47\ \Omega$. Its temperature coefficients (`tc1 = -4.3e-4`, `tc2 = 12e-6`, `tnom = 30`) scale that by $1 + (-4.3 \times 10^{-4})(-3) + (12 \times 10^{-6})(9) = 1.001398$ at the 27 °C ngspice simulated at $(27 - 30 = -3$, and $(-3)^2 = 9)$, giving 299.89 Ω. And $299.89 + 78.35 = 378.24$, against the 378.2448 you measured. The arithmetic closes to five figures using only constants in a text file you can `cat`.

More of the `make resistor` output

Part 6's `make resistor` printed three more blocks.

- Block C is one 314.046460 µm × 1 µm strip (the length that gives 100 kΩ of `res_high_po`) simulated as four different resistor models: the two poly implants, and the p+ and n+ diffusion resistors. The drawing and the silicon area are the same, and the values spread over 18:1:

```
--- C. one 314 um x 1 um strip, four implants (ohms) ---
r_high = 1.000000e+05
r_xhigh = 6.653872e+05
r_pdiff = 6.138915e+04
r_ndiff = 3.689775e+04
```

- Blocks B and D are the worked examples on What a resistor costs in area: 1 k, 10 k and 100 k sized with the design equation, and the same 100 kΩ at four widths. Check that each lands on its target.

Part 8 (optional): The `u` suffix trap

`spice/u_trap.spice` holds the same resistor written twice, once as `W=1 L=10` and once as `W=1u L=10u`. Getting started explains what goes wrong.

1. Run it:

```
make utrap
```

You should see:

```
--- the same device, written two ways (ohms) ---
good = 3.550443e+03
bad = 3.193812e+03

   and the one line that told you, buried in the log:
Warning: r.xb.rbody: resistance too low or not given, set to 1 mOhm
   ngspice exit status: 0
```

The wrong spelling gives 3193.812 Ω instead of 3550.443 Ω, 10 % off, with no error and exit status 0. Apart from `u_trap.spice`, which exists to show the mistake, there is no `u` suffix anywhere in this package, and there should be none in yours.

Part 9 (optional): Draw a resistor in SiliWiz

SiliWiz is a browser layout tool from the Tiny Tapeout project. It shows a cross-section of your drawing while you draw it, which makes "a strip of poly with two contacts" concrete.

1. Open **https://app.siliwiz.com/?preset=blank**.
2. Pick the **polyres** layer from the palette on the left and draw a long thin rectangle.
3. Pick **metal1 via** and drop one square at each end of it.
4. Pick **metal1** and put a contact pad over each via.
5. Click each metal1 pad, choose **Set Label** (or press `S`), and name them `in` and `out`.
6. Tick **Show SPICE** at the bottom. Find the line starting `R0 in out`; that is your resistance.
7. Watch the **cross-section** view as you drag the rectangle longer and thinner.

You should see: a line starting `R0 in out` in the SPICE output, and its value changing as you reshape the strip.

There is a guided version of this exercise at https://tinytapeout.com/siliwiz/resistors/.

> **Warning:** SiliWiz is not SKY130, and its numbers are not the ones in this lab. Its `polyres` layer is documented as *"400 ohms per square"*. SKY130's `res_high_po` is 317.2198 Ω/□ measured, and 319.8 Ω/□ in the Magic layout tool's tables (see Ohms per square). SiliWiz uses round numbers chosen for teaching. Use it for the cross-section and ngspice for the value.

Check the rest of the image

AD102 doesn't use these tools. They are the AD104 layout tools, listed so you can confirm the image is complete:

Tool	Command	Expect
Magic	`magic -dnull -noconsole --version`	`8.3.681`
KLayout	`klayout -v`	`KLayout 0.30.11`
Netgen	`netgen -batch quit | head -1`	`Netgen 1.5.323 compiled on …`

`netgen -batch quit` also prints `Warning: netgen command 'format' use fully-qualified name '::netgen::format'` and a second line like it. Those are Tcl namespace notices from the tool itself. They have nothing to do with your design, so there's nothing to fix.

Next on the Course path: read the four pages in row 3, starting with Capacitors on a chip, then do Lab 02: How big is a picofarad?.

Answers

A1. Part 2: Measure the material

You predicted 31,739 Ω for the longest strip, and it measured 32,100 Ω, which is 1 % high. You predicted 317 Ω for the shortest, and it measured 695 Ω, a factor of 2.2 higher. The same formula is that close at one end and that far off at the other, so something is being added that doesn't depend on how long you drew the strip.

A model that is right at one end of your design range and wrong at the other is worse than one that is wrong everywhere, because it passes every check you run on the big case.

A2. Part 2: Measure the material

It's 0.4 % of a 100 kΩ resistor, which you can ignore, but it's most of a 500 Ω one.

A3. Part 3: Size a 10 kΩ resistor

Think about what resistors are for. A resistor that sets a bias current sets it 3.7 % wrong. A pair of them setting a gain of 10 gets the gain wrong too, because the fixed 378 Ω is a different fraction of a long resistor and a short one. And the error is systematic: every chip on every wafer gets the same 3.7 %, so no amount of averaging finds it. A random spread is a tolerance, which you budget for; an error that is the same on every chip is a mistake, which you fix.

A4. Part 4: Change the width

From the bottom up the column reads 316.966, 317.061, 317.220 and 317.362, constant to a tenth of a percent, so sheet resistance behaves as a property of the material. That holds until the top row, 0.35 µm, where it jumps 7 % to 340.074. Below 0.69 µm the model applies `weff = w - 0.001 - 0.0672 * (0.69 - w)`, so a strip drawn 0.35 µm wide conducts as though it were 0.326 µm wide, because lithography and etch don't give you exactly the width you drew.

The end resistance changes with width as well, from 82.841 Ω at 5 µm to 958.562 Ω at 0.35 µm, because the contact head is as wide as the strip. Its resistance is `rsh` divided by a width; see End resistance from the model card under Going further.

A5. Part 4: Change the width

You win twice by going narrow, once on width and again on the length you no longer need, so area per ohm goes as W^2. That makes "just draw it narrow" look free, but a narrow strip costs you something in matching, and Lab 04 measures how much.

A6. Part 5: Change the recipe

Design with the simulation. The headline number on a model card is a nominal value, whereas the device that gets fabricated behaves the way the model computes.

The implant is also the only knob in this lab that is not geometry. The foundry gives you a short menu of implant doses, and choosing from it is the one way to change ohms per square instead of buying more area.

A7. Part 6: Split a resistor into head and body

They're in the body. The model's `leff` makes the strip 0.247 µm longer than you drew it, and that extra length is worth 78.35 Ω. The 378.2448 Ω is the 299.89 Ω head plus that 78.35 Ω body extension. End resistance from the model card, under Going further, has the arithmetic.

A8. Part 7: Size your own resistors

378.2448/2200 is 17.2 %, about a sixth, so almost all of R1′s error is the ends that the naive length left out. On a 2.2 kΩ resistor the ends are no longer a rounding error.

A9. Finishing

`res_xhigh_po`: 47.18 µm², 0.43 adders, against 314.05 µm² and 2.85 adders for `res_high_po` (Part 5). Area doesn't settle it, though. `res_xhigh_po`'s model card has a voltage coefficient, `vc1_body = -1.00e-3`: its resistance moves by about 0.1 % per volt across it, which doesn't matter in a bias string but shows up as distortion in a signal path. The lab's `solutions/README.md` has the longer version.

II

3–4. Capacitors and inductors

7 Capacitors on a chip 39
7 The three kinds of capacitor in SKY130 39
7 Measured capacitance of each type 39
7 The MIM arithmetic 40
7 MOS capacitance and gate voltage 40
7 Choosing between them 41

8 What a picofarad costs 43
8 The exchange rate 43
8 What sizes are practical 43
8 Density and plate size 44
8 Worked example: one plate or a hundred 44
8 Costs besides area 44
8 Reading capacitor values 44

9 Parasitic capacitance 45
9 Parasitic capacitance in the PDK 45
9 Fringe capacitance of a thin wire 45
9 What a long wire costs 46
9 VPP capacitors and parasitic extraction 46

10 Inductors on a chip 47
10 The three SKY130 inductor models 47
10 Q 48
10 Self-resonance 49
10 Area 49
10 Why spirals are large and lossy 51
10 Inductors below a gigahertz 52

11 Lab 02: How big is a picofarad? 53
11 Before you start 53
11 Part 1: The area ladder 53
11 Part 2: Fit the area and edge terms 54
11 Part 3: Size one picofarad 55
11 Part 4: Three ways to build a capacitor 55
11 Part 5: The MOS capacitor's C–V curve 56
11 Part 6: Size your own capacitors 56
11 Part 7: The three inductors 57
11 Part 8: A 2.4 GHz resonator 58
11 Finishing 59
11 Going further (optional) 59
11 Answers 60

7. Capacitors on a chip

There is no ceramic disc on a chip, so what is a capacitor made of?

At bottom, a capacitor is **two conductors that are near each other but not touching**. The dielectric, the plates and the packaging are engineering built around that.

Put two flat plates of area A a distance d apart, with a material of relative permittivity ε_r between them:

$$C = \varepsilon_0 \varepsilon_r \frac{A}{d} \tag{7.1}$$

You know this formula already. What changes on a chip is that you can't choose d, because the foundry has already fixed the thickness of every layer. That leaves A, the area you're willing to spend, as the only thing you can vary. So on a chip, capacitance is quoted in **farads per square micrometre**, and the design question is always how much floor space you're buying.

The three kinds of capacitor in SKY130

MIM (metal–insulator–metal). A dedicated thin plate (`capm`) sits just above met3, with a thin dielectric made for the purpose between the two. That plate has no other job than making capacitors. The models are `sky130_fd_pr__cap_mim_m3_1` (between met3 and met4) and `..._m3_2` (met4 to met5).

MOS (a transistor used as a capacitor). Take an nFET, tie source, drain and body to ground, and use the gate as one plate against the channel underneath. The dielectric is the gate oxide, the thinnest insulator on the chip, so this is by far the densest capacitor available.

VPP (vertical parallel plate). A VPP needs no special layer at all. Ordinary metal wires are interleaved like the teeth of two combs, and the capacitance between their sidewalls does the work. SKY130 ships about eighty pre-drawn VPP structures, named by their footprint: `sky130_fd_pr__cap_vpp_11p5x11p7_l1m1m2m3m4_shieldm5` is 11.5 µm × 11.7 µm and uses li plus met1 through met4, shielded by met5.

Measured capacitance of each type

`labs/passives-decks/spice/c_area.spice` puts a 1 V AC source at 1 kHz across each device and reads the current it draws, so $C = \mathrm{Im}(I)/2\pi f$. Nothing is fitted to the result, so each number is exactly what the model produces. The deck prints:

```
--- capacitance in farads ---
c_mim_1x1 = 2.642250e-15
c_mim_10x10 = 2.065822e-13
c_mim_30x30 = 1.819782e-12
c_vpp = 1.473400e-13
c_mos_1v8 = 7.876883e-13
c_mos_0v0 = 2.207182e-13
```

construction	drawn size	capacitance	density
MIM	1 × 1 µm	2.64225 fF	2.64 fF/µm²
MIM	10 × 10 µm	206.5822 fF	2.07 fF/µm²
MIM	30 × 30 µm	1819.782 fF	2.02 fF/µm²
VPP fringe	11.5 × 11.7 µm	147.3400 fF	1.10 fF/µm²
MOS at 1.8 V	10 × 10 µm gate	787.6883 fF	7.88 fF/µm²
MOS at 0 V	the same device	220.7182 fF	2.21 fF/µm²

You'll run this deck in Lab 02, Part 5. Two things in that table need explaining: the MIM density, which isn't the same at every size, and the MOS capacitor, whose value changes with voltage.

The MIM arithmetic

You can reproduce the MIM model's numbers by hand, to the last digit. Its parameter file (`libs.tech/ngspice/r+c/res_typical__cap_typical__lin.spice`) says

```
+ camimc=  2.00e-15  ; Units: farad/micrometer^2
+ cpmimc = 0.19e-15 ; Units: farad/micrometer
```

That's 2.00 fF for every square micron of plate, plus 0.19 fF for every micron of plate *perimeter*, which comes from the field that bulges out at the edges. The model card also corrects the drawn size:

```
.param wc = 'w+m3_dw*1e6+tol_m3*1e6'
```

with `m3_dw = -0.025u`, so a plate you drew 30 µm across is manufactured 29.975 µm across. For the 30 × 30 device:

$$0.19 \times 2 \times (29.975 + 29.975) = 1797.00125 + 22.781 = \boxed{1819.782\ \text{fF}} \qquad (7.2)$$

ngspice printed `1.819782e-12`, which matches every digit.

The perimeter term also explains the density column: 2.64 fF/µm² for the 1 × 1 device, falling to 2.02 fF/µm² at 30 × 30. **Small capacitors are denser than big ones**, because the edge makes up a larger fraction of a small plate. It's the same kind of effect as the contact heads on a short resistor, and you'll see it again.

MOS capacitance and gate voltage

Look again at the last two rows of the table. They're the same transistor with the same 100 µm² of gate, yet it measures 787.6883 fF with 1.8 V on the gate and 220.7182 fF with 0 V. That's a factor of 3.57, which is a lot for a component whose whole job is to have a value.

`spice/c_moscap_cv.spice` sweeps the gate from 0 V to 1.8 V in 100 mV steps:

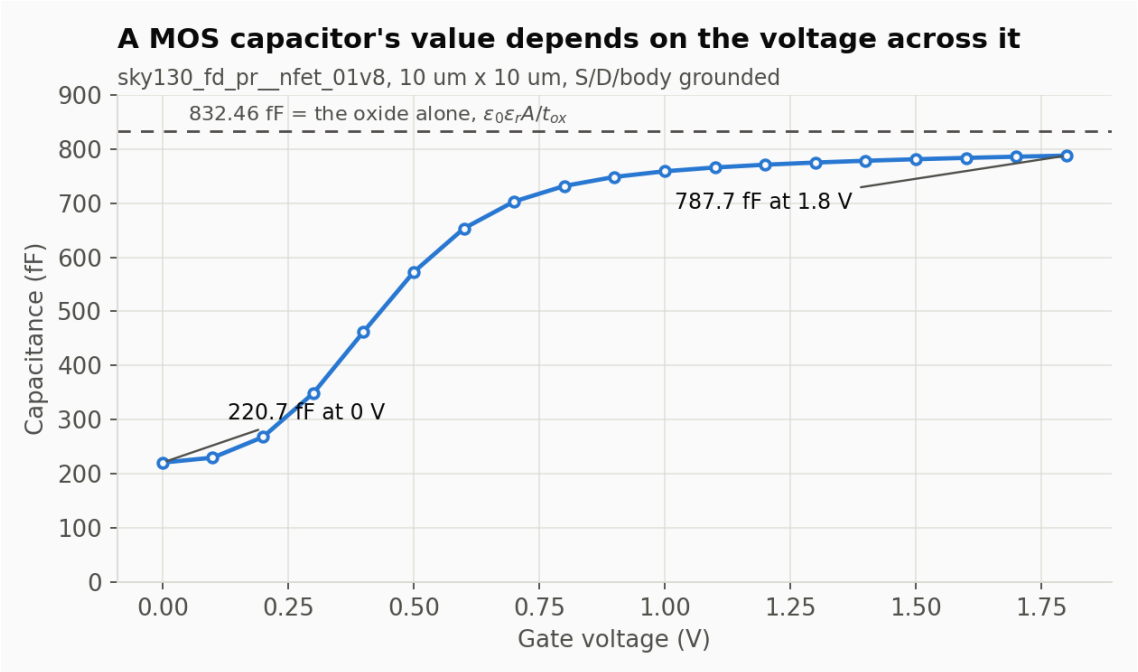


Figure 7.1: MOS capacitor C-V

The curve starts at 220.7182 fF at 0 V, climbs steeply between 0.3 V and 0.7 V, and flattens toward 787.6883 fF without quite reaching the dashed line.

The steep part is the threshold. Below it, the gate faces a depleted, nearly empty channel, so it behaves like two plates far apart. Above it, an inversion layer forms right under the oxide, so the second "plate" is now only a few nanometres away. What you're looking at is a MOSFET turning on, seen from the capacitance side, and you'll meet this capacitance again in AD103 as what sets a transistor's speed.

The dashed line is worked out from the oxide thickness rather than fitted to the curve, and the nFET model card (`sky130_fd_pr__nfet_01v8__tt.pm3.spice`) gives that thickness:

```
+ toxe = {4.148e-09 + ...}
```

That is 4.148 nm, so the oxide capacitance per unit area is

$$C_{ox} = \frac{\varepsilon_0 \varepsilon_r}{t_{ox}} = \frac{8.854 \times 10^{-12} \times 3.9}{4.148 \times 10^{-9}} = 8.3247\ \text{fF}/\mu\text{m}^2 \tag{7.4}$$

and over 100 µm² of gate that comes to 832.46 fF. The simulation tops out at 787.6883 fF, which is 94.6 % of it. The missing 5 % is the inversion layer's own finite capacitance, which sits in series with the oxide. That's close enough that you can use $C_{ox}A$ as a back-of-envelope estimate for a MOS capacitor and be within 6 %.

Choosing between them

	MIM	MOS	VPP
density	2.0 fF/µm²	~8 fF/µm²	~1.1 fF/µm²
linear?	**yes**	no: 3.57× swing	yes
leaks?	no	yes, through 4 nm of oxide	no
needs extra masks?	yes	no	**no**
sits above the transistors?	yes	no	no: it uses li and all five metals

Use **MIM** when the capacitance has to stay at the value you designed: in filters, references, and anything else where "3.57× depending on the signal" would break the circuit. Use **MOS** when you need a lot of farads and don't care about linearity. Supply decoupling is the classic case, because there the capacitor's whole job is to be a charge reservoir at a fixed voltage. Use **VPP** when you want a capacitor without paying for the MIM masks, or when you want a ratio rather than a value, since fringe structures match well.

The word "capacitor" covers three different devices here, each with its own way of failing. If you pick the wrong one, you can end up with a filter whose corner frequency depends on the input signal, and that kind of bug doesn't show up until the chip is built.

Next: What a picofarad costs, where the density column turns into floor plan.

8. What a picofarad costs

The capacitors in a chip schematic are given in femtofarads, while the ones in your parts drawer are in microfarads. The reason is that on a chip you pay for capacitance in square micrometres of area, and the exchange rate is poor.

In Capacitors on a chip we measured a MIM capacitor rather than quoting a number for it, and it gives about 2 fF per µm². This page works out what that density means for a floor plan.

The exchange rate

you want	MIM plate area	plate is about	in `sky130_fd_sc_hd__inv_1` (3.7536 µm²)
100 fF	49 µm²	7 × 7 µm	13
1 pF	494 µm²	22 × 22 µm	132
10 pF	4 940 µm²	70 × 70 µm	1 316
100 pF	49 400 µm²	222 × 222 µm	13 160
1 nF	494 000 µm²	703 × 703 µm	131 600
1 µF	494.6 mm²	*a plate 2.2 cm on a side*	132 million

These numbers come from the measured 30 × 30 µm device: 1819.782 fF over 900 µm² is 2.0220 fF/µm², so 1 pF needs $1000/2.0220 = 494.6$ µm². The last column gives the same area as a count of `sky130_fd_sc_hd__inv_1` inverter cells, each 3.7536 µm².

Compare the last row with the 100 nF ceramic capacitor you would solder next to a microcontroller without thinking about it. A typical small SKY130 die is about 3 mm × 3 mm, or 9 mm², and the 1 µF capacitor needs 494.6 mm², which is the area of fifty-five entire chips for one component. On-chip capacitors are small because a 1 µF capacitor physically can't fit on a die, and no amount of cleverness closes a gap that size.

Switching to the densest option doesn't get you there either. A MOS capacitor at 7.88 fF/µm² brings 1 µF down to 126.9 mm², which is still fourteen whole dies, and its value changes by 3.57× with the voltage across it.

What sizes are practical

capacitance	typical use
1–50 fF	sampling capacitors in an ADC, amplifier compensation, charge redistribution
50–500 fF	filter capacitors, matched arrays, bandgap compensation
1–20 pF	serious on-die decoupling, a big integrator, an off-chip driver's load
> 50 pF	you are budgeting a large fraction of the die for one component

These limits explain a lot about how analog IC design looks. A discrete designer builds a 1 kHz low-pass filter from 16 kΩ and 10 nF, but that capacitor would take 4.94 mm² on a die, which is more than half of a typical small die. So the on-chip designer doesn't build that filter. Instead they build a switched-capacitor

filter, or use a very large resistance made from a starved transistor, or push the corner frequency up by three decades and filter the rest off-chip. For more on this, see What to build instead of an inductor.

Density and plate size

Divide each measured MIM capacitor by its drawn area (1, 100 and 900 µm²):

```
c_mim_1x1 = 2.642250e-15
c_mim_10x10 = 2.065822e-13
c_mim_30x30 = 1.819782e-12
```

You get 2.64, 2.07 and 2.02 fF/µm², so the density falls as the capacitor gets bigger.

The extra capacitance in the small one comes from the field at the plate's edge, which the model adds as `cpmimc = 0.19e-15` farads per micron of perimeter. Perimeter grows with the side length while area grows with its square, so the edge adds 24 % at 1 µm but only 1 % at 30 µm. **Never estimate a small capacitor from area alone.** In Lab 02, Parts 1 and 2, you measure a ladder of plate sizes and separate the area term from the edge term yourself.

Worked example: one plate or a hundred

You need 1 pF, and you have room for a single 23 × 23 µm MIM. Alternatively, you could place one hundred 2.3 × 2.3 µm MIMs in parallel, which uses the same 529 µm² of plate. Which gives more capacitance? Work it out before reading on.

Most people say "the same", but the perimeter term makes them differ. One big plate gives $2.00 \times 22.975^2 + 0.19 \times 2 \times (22.975 + 22.975) = 1055.70 + 17.46 = 1073.2$ fF. Each of the hundred small plates gives $2.00 \times 2.275^2 + 0.19 \times 2 \times (2.275 + 2.275) = 10.35 + 1.729 = 12.08$ fF, and a hundred of them give 1208 fF. Cutting the plate into pieces gives you 12.6 % more capacitance for the same plate area, because together the pieces have a lot more edge than the one big plate.

Designers really do use this, and it's the reason VPP capacitors exist and the `sky130_fd_pr__cap_vpp_*` structures are drawn as interleaved fingers rather than slabs. In practice, a hundred small plates also need a hundred sets of connections and a lot of spacing between them, so nobody takes it to that extreme; the foundry took the idea as far as it goes when it drew the VPP cells.

Costs besides area

A capacitor also has three costs that don't show up in its value in farads.

Bottom-plate parasitic. A MIM plate is a large sheet of metal above the substrate, so it has its own capacitance to ground whether you asked for it or not, and the `.subckt` has a third terminal for exactly this. In a switched-capacitor circuit this parasitic is the main source of error, and much of the design work goes into arranging for it to be driven by something that doesn't care about it.

Series resistance. The MIM `.subckt` has two resistors in series with the capacitance:

```
rs1 a  b1 'r1' tc1 = {tc1rm3} tc2 = {tc2rm3}
rs2 b1 c1 'r2' tc1 = {tc1rvia3} tc2 = {tc2rvia3}
```

`r1` is the sheet resistance of the met3 plate itself, and `r2` is the resistance of the via array. So what you've drawn is really a capacitor in series with a small resistor, and at high frequency the resistor dominates.

Corner spread. The MIM's area capacitance is `2.00e-15` F/µm² at the typical corner, but `1.778e-15` at the low corner and `2.231e-15` at the high one, a spread of about ±11 %. Process corners covers what to do about that.

Reading capacitor values

When an analog paper says "C_1 = 240 fF", that capacitor is 119 µm² of somebody's floor plan, and the area was argued over in a design review. Area is the currency of this whole field, and apart from inductors, capacitors are the most expensive thing you can buy with it.

Next: Parasitic capacitance, the capacitors nobody drew in the schematic.

9. Parasitic capacitance

If a capacitor is just two conductors near each other, then every wire on a chip is a capacitor, with capacitance to the substrate, to the wires above and below it, and to the wires running alongside it. You didn't draw any of these capacitors, and you can't delete them. They're called **parasitics**, and on a modern chip they are usually the largest capacitance in the design.

Parasitics are why digital chips have a maximum clock rate and why a layout that passes DRC can still be slow, and they're the reason AD104 exists.

Parasitic capacitance in the PDK

The PDK tabulates these capacitances in `/foss/pdks/sky130A/libs.tech/ngspice/parameters/typical.spice`, on lines 2876–2902. The area terms are in F/m² and the sidewall terms are in F/m:

```
+ cp1f = 1.06e-04  cp1fsw = 8.64e-11
+ cl1f = 3.69e-05  cl1fsw = 8.30e-11
+ cm1f = 2.58e-05  cm1fsw = 1.07e-10
+ cm2f = 1.75e-05  cm2fsw = 1.08e-10
+ cm5f = 6.48e-06  cm5fsw = 7.85e-11
+ cm2m1 = 1.28e-04 cm2m1sw = 1.03e-10
```

Here are the same numbers converted to the units a layout engineer works in (1 F/m² = 1e-12 F/µm², 1 F/m = 1e-6 F/µm):

coupling	area	fringe, per edge
poly ⟶ substrate (`cp1f`)	0.106 fF/µm²	0.0864 fF/µm
li ⟶ substrate (`cl1f`)	0.0369 fF/µm²	0.0830 fF/µm
met1 ⟶ substrate (`cm1f`)	0.0258 fF/µm²	0.107 fF/µm
met2 ⟶ substrate (`cm2f`)	0.0175 fF/µm²	0.108 fF/µm
met5 ⟶ substrate (`cm5f`)	0.00648 fF/µm²	0.0785 fF/µm
met2 ⟶ met1 (`cm2m1`)	0.128 fF/µm²	0.103 fF/µm

For comparison, a MIM capacitor is 2.00 fF/µm², and met1 to substrate is 0.0258 fF/µm², which is 78 times less dense. That's the way round you want it, since the capacitor you built on purpose ought to beat the accidental one. Still, 78 times less dense is a long way from zero, and a chip has an enormous amount of wire.

Fringe capacitance of a thin wire

Take a met1 wire of minimum width, which the PDK gives one directory up, in `libs.tech/ngspice/sky130_fd_pr__model__r+c.model.spice`, as `wminm1= 0.14u`, and run it 1000 µm across the chip. Its capacitance to the substrate has two parts:

- **Plate area:** $1000 \times 0.14 = 140$ µm², times 0.0258 fF/µm² = 3.61 fF
- **Fringe:** two edges, 1000 µm each, times 0.107 fF/µm = 214.0 fF

The total is about 217.6 fF, and the parallel-plate part, the one you would have calculated from $\varepsilon A/d$, is only 1.7 % of it.

Almost everyone's first estimate of a wire's capacitance comes from its area, because that's the formula from the previous page, but for a minimum-width wire that estimate is off by a factor of sixty. The reason is geometric: the wire is 0.14 µm wide and sits on the order of a micron above the substrate, so it's more like a string than a plate, and most of its field leaves through the sides rather than the bottom. `cm1fsw` is the

capacitance of that sideways field, as SkyWater measured and tabulated it, and it's the reason extraction tools exist.

As a rule, for anything narrower than it is tall, the sidewall capacitance dominates, so use the `*sw` term rather than the area term.

What a long wire costs

Now put that wire on the output of the smallest inverter, `sky130_fd_sc_hd__inv_1`, which comfortably drives a few femtofarads of gate load. Hang 217.6 fF on it instead, roughly fifty times its normal load, and the rise time goes up by roughly the same factor. That's the mechanism behind the saying that long wires are slow, and it's why place-and-route tools spend most of their effort on wire length rather than gate count.

The same wire also has resistance. 1000 µm of minimum-width met1 is $1000/0.14 = 7143$ squares, and at 0.125 Ω/□ that comes to 893 Ω (see Ohms per square). The resistance and the capacitance are both spread along the wire's whole length, so the wire behaves as a distributed RC line rather than a single node, and the delay through it grows as the square of its length.

In Lab 02, Part 9 (optional) you can draw two wires in SiliWiz and watch their coupling capacitance change as you move them.

VPP capacitors and parasitic extraction

Parasitic capacitance has two more consequences. The first is that it's what makes a VPP capacitor work. Ordinary metal wires couple to their neighbours at about 0.1 fF per micron of edge whether you like it or not, so you can use that coupling on purpose: interleave enough fingers and the accident becomes the component. On Capacitors on a chip, `sky130_fd_pr__cap_vpp_11p5x11p7_l1m1m2m3m4_shieldm5` measured 147.3400 fF, and it's built entirely out of wire.

The second is that every number on this page came from a pre-characterised table, because the real value depends on whatever happens to be near the wire, and a table can't know that. A **parasitic extraction** (PEX) run reads your finished layout, works out these couplings for the shapes you actually drew, and hands back a netlist with hundreds of capacitors your schematic never had. In AD104 you run one and compare the circuit before and after extraction.

At some point every chip designer sees a circuit that simulated perfectly come back from extraction 40 % slower, with nothing in the schematic changed. We wrote this page so that when it happens to you, you'll know why.

Next: Inductors on a chip.

10. Inductors on a chip

Resistors and capacitors on a chip both turned out to be shapes you draw, so you might expect an inductor to be a spiral you draw too. It is, but that spiral is both a poor inductor and an enormous one, and both problems have the same underlying cause, which the end of this page explains.

So far every passive on a chip has been "a shape, and the shape costs area". Inductors don't fit that pattern, because although you can draw one and the foundry will make it, at most frequencies what you get back behaves like a resistor.

The three SKY130 inductor models

SKY130 models exactly three spiral inductors. They live in `libs.ref/sky130_fd_pr/spice/`, and unlike everything else in this course they aren't loaded by `sky130.lib.spice`, so you have to `.include` them yourself:

```
sky130_fd_pr__ind_03_90.model.spice
sky130_fd_pr__ind_05_125.model.spice
sky130_fd_pr__ind_05_220.model.spice
```

Warning: if you forget the `.include`, ngspice says

```
Error: unknown subckt: x1 a 0 ct 0 sky130_fd_pr__ind_05_220
    Simulation interrupted due to error!
Error: incomplete or empty netlist
```

That error means the netlist is missing the inductor's model file, and the PDK itself is fine.

The subcircuit in `sky130_fd_pr__ind_05_220.model.spice` is fourteen lines long:

```
.subckt  sky130_fd_pr__ind_05_220 a b ct sub
R31 net27 b r='1e-2'
R26 a net23 r='1e-2'
C2 net27 net31 c='223.7e-15'
C24 net35 net27 c='205e-15'
C25 net23 net37 c='223.7e-15'
R5 sub net31 r='5.358e3'
R4 net41 net27 r='2.059'
R13 net23 net35 r='9.064'
R10 sub net37 r='5.358e3'
R9 net23 net39 r='2.059'
    L1 net39 ct l=4.96e-9
    L3 ct net41 l=4.96e-9
.ends
```

There are two inductors of 4.96 nH, one on either side of the centre tap `ct`, so the total is **9.92 nH**. Around them are 4.138 Ω of series resistance (2.059 + 2.059 + 0.01 + 0.01); 447.4 fF of coupling into the substrate, as two 223.7 fF capacitors each draining through 5.358 kΩ into the `sub` terminal; and 205 fF directly across the winding, in series with a 9.064 Ω resistor (`R13`).

Only two of its twelve elements are inductors. The other ten describe the losses and parasitics, and the rest of this page is about what they do to the device.

Q

The figure of merit for an inductor is its **quality factor**:

$$Q = \frac{\mathrm{Im}(Z)}{\mathrm{Re}(Z)} \approx \frac{\omega L}{R_s} \tag{10.1}$$

An ideal inductor has $Q = \infty$ and a resistor has $Q = 0$, so Q tells you how much of the component is actually behaving as an inductor.

`labs/passives-decks/spice/l_spiral.spice` measures `ind_05_220` at four frequencies:

```
--- ind_05_220: L (H), Q, series R (ohm) at four frequencies ---
l_1khz              =  9.91271e-09
q_1khz              =  1.50516e-05
r_1khz              =  4.13800e+00
l_1mhz              =  9.91271e-09
q_1mhz              =  1.50516e-02
l_100mhz            =  9.92203e-09
q_100mhz            =  1.50196e+00
l_1ghz              =  1.07762e-08
q_1ghz              =  1.16534e+01
```

At 1 kHz, Q is 0.0000150516, which means that at audio frequencies the largest inductor SKY130 offers is 99.9985 % resistor: a 4.138 Ω resistor with a trace of inductance attached. Nothing in the circuit can change that, because $Q = \omega L / R$ and you don't get to choose ω.

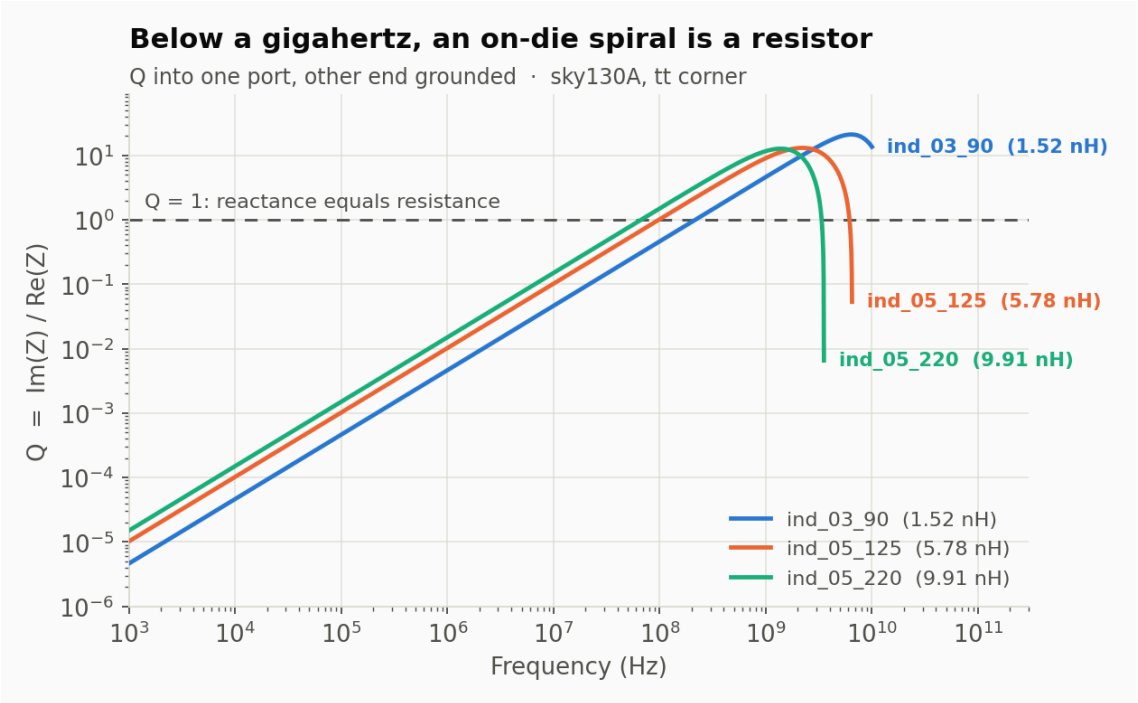


Figure 10.1: Q of the three SKY130 spirals versus frequency

Each line is straight on these log–log axes because Q is proportional to frequency, so every decade up multiplies Q by ten. That rise is the reason on-die inductors exist at all: at 2.4 GHz they are useful, and every Wi-Fi and Bluetooth radio ever made has spirals in its LC oscillator. Below about 100 MHz they take up space and do nothing useful.

```
--- best Q each spiral ever reaches, and where ---
qpeak_03_90         =  2.11268e+01 at=  6.38263e+09
```

```
qpeak_05_125        =  1.30871e+01 at=  2.21309e+09
qpeak_05_220        =  1.27198e+01 at=  1.38835e+09
```

The best any of them manages is Q = 21.1, from `ind_03_90`, and it gets there only at 6.38 GHz. Compare that with the resistor you have been simulating all course, whose value you can hit to five digits.

You'll run this deck, and the Lab 02 inductor sweep, in Lab 02, Part 7.

Self-resonance

The 223.7 fF capacitors in the model resonate with the inductance, and above the resonant frequency the whole structure behaves like a capacitor. Here is that frequency for two of the spirals:

```
--- self-resonance: the frequency where it stops being an inductor ---
srf_05_125          =  6.49382e+09 with=  1.99663e+03
srf_05_220          =  3.52777e+09 with=  2.18171e+03
```

(`meas ac` also prints the value it was searching on; the frequency is the first number.)

Above 3.52777 GHz the reactance of `ind_05_220` changes sign, and on the Q plot all three curves drop off sharply at the high-frequency end. That leaves `ind_05_220` barely a decade and a half of usable band, from 100 MHz, where Q is still only 1.50, up to 3.53 GHz, where it stops being an inductor at all. Its peak Q in that band is 12.7.

Area

The `ind_*` files are electrical models with no layout attached, but SKY130 does ship drawn spirals: three RF test coils inside `libs.ref/sky130_fd_pr/gds/sky130_fd_pr.gds`. They are separate PDK objects from the three models, with different names and characterisation, but they are the same kind of structure, and they are the only on-die inductors this process has drawn. These are their bounding boxes, measured with KLayout:

cell	footprint	area
`sky130_fd_pr__rf_test_coil1`	150.020 × 145.040 µm	21 758.9 µm²
`sky130_fd_pr__rf_test_coil2`	270.000 × 265.000 µm	71 550.0 µm²
`sky130_fd_pr__rf_test_coil3`	370.080 × 365.080 µm	135 108.8 µm²

Here is the middle one, rendered from that GDS with the SKY130 KLayout layer properties:

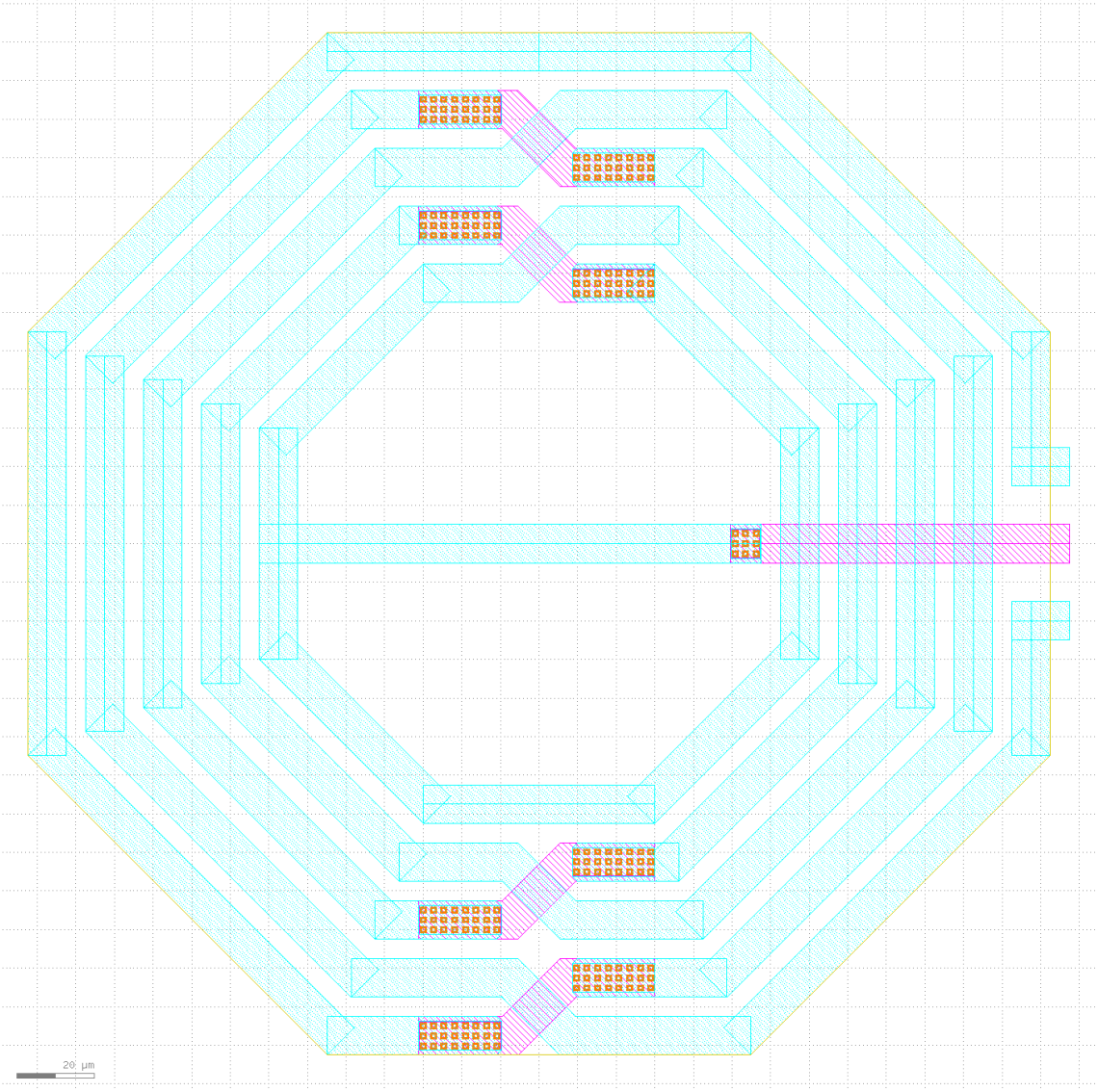


Figure 10.2: A SKY130 spiral inductor layout

It is made of concentric octagonal turns of met3 (cyan, 32 806.81 µm² of it), with met2 crossunders (magenta) where the winding has to cross itself, and 201 arrays of via2 joining the two layers so both metals carry the current in parallel. The bar running out through the middle is the third terminal, the same centre tap the netlist calls `ct`. The whole thing sits inside a marker on a layer named `inductor.drawing` (GDS 82/24), which tells the foundry's tools not to put anything underneath it.

Here is the same picture with forty-nine `sky130_fd_sc_hd__inv_1` standard cells placed in the middle, at the same scale:

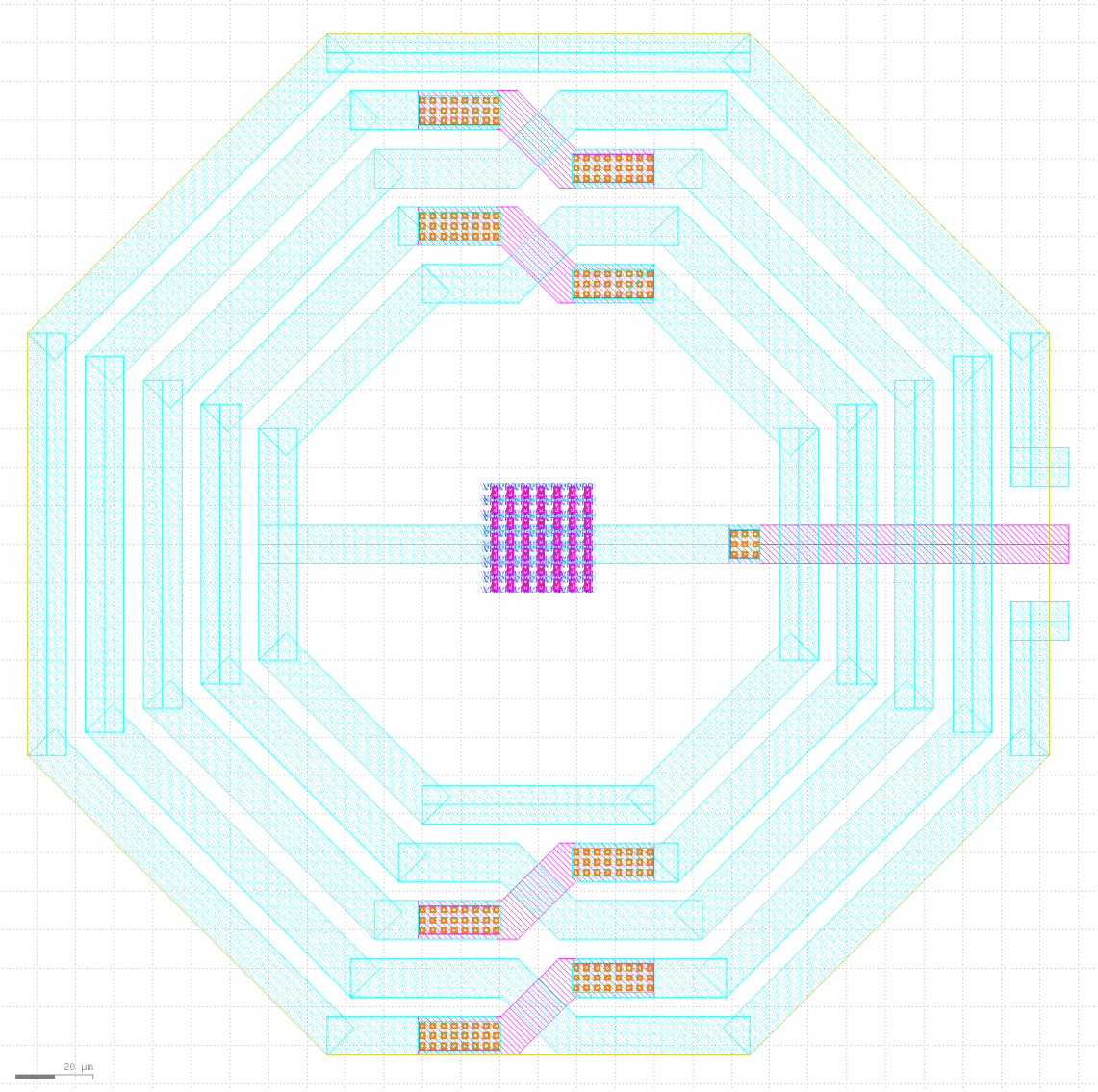


Figure 10.3: The same inductor with 49 inverters for scale

You can barely see the inverters, because `inv_1` is 3.7536 µm² and the coil is 71 550 µm². One inductor takes the area of 19 062 inverters, a small CPU's worth of silicon, for single-digit nanohenries and a Q in the low teens, in a frequency band you probably weren't using.

Why spirals are large and lossy

An inductor stores energy in a magnetic field in the space around the wire. A capacitor stores energy in an electric field across a gap you can make 4 nanometres thick, and a resistor is a strip whose length you control. A magnetic field has no thin-film equivalent: to get inductance you need area enclosed by current, and the only way to get that is to spend square micrometres. The field also goes straight down into the silicon substrate, which conducts, so it induces eddy currents. In the model the substrate is the `sub` terminal, and the only elements connected to it are the two 5.358 kΩ resistors, each behind a 223.7 fF capacitor. Both problems come from the same field, so you can neither make an inductor small nor stop it coupling to the wafer it is printed on.

That's why bond wires and package traces are sometimes used as the inductor instead. A 1 mm bond wire is roughly a nanohenry, and because it sits in the package rather than on the die, it costs no silicon.

Inductors below a gigahertz

If your circuit needs an inductor below a gigahertz, the circuit is wrong for a chip. The fix has to go in the circuit rather than the layout.

No layout trick, fold or metal stack fixes $Q = 1.5 \times 10^{-5}$ at 1 kHz, so you change the topology instead. Analog designers have been working around this since the 1960s, and there is a mature set of replacements.

Next on the Course path: Lab 02: How big is a picofarad?. After that, What to build instead of an inductor covers the replacements.

11. Lab 02: How big is a picofarad?

You'll measure capacitance from an AC current, separate a plate capacitor's area term from its edge term, and price one picofarad in each of the capacitor types SKY130 offers. Then you'll size two capacitors of your own, measure the inductance, Q and self-resonance of the PDK's on-chip spirals, and build a 2.4 GHz resonator from one.

Package: `labs/lab-02-how-big-is-a-picofarad/`

Before you start

- Finish Lab 01: `make mine check` in `labs/lab-01-a-resistor-you-designed` prints `PASS`.
- Read Capacitors on a chip, What a picofarad costs, Parasitic capacitance and Inductors on a chip.
- Open a terminal in the workbench and go to the lab folder:

```
cd /foss/designs/modules/ad102/labs/lab-02-how-big-is-a-picofarad
```

  You don't need to set up any environment for this lab, because every deck names its model files by absolute path.

Parts 1 to 8 are required. Everything under Going further, at the end, is optional.

Time: about 2.5 hours, not counting the optional parts. A full `make` runs four ngspice decks (`pf`, `mine`, `henry` and `tank`) and then a checker script, `src/check_cap.py`, which fits and prints the results. Three of the decks print nothing for anywhere from about 10 seconds to a minute each while the SKY130 model cards load, so don't take the silence for a hang; the Makefile budgets about three and a half minutes for the whole `make` on the pinned image (Warnings and delays).

The theory this lab uses is all on Capacitors on a chip, and here it is in the form the lab needs. Two plates of area A separated by a dielectric of thickness t give $C = \varepsilon A/t$. On a chip, ε and t are both fixed by the process, so capacitor design comes down to one constant, farads per square micrometre, and one decision, how much area to spend. You can't make the dielectric thinner, so the only way you can get more capacitance is to make the plate bigger. The process engineers could raise the density only by making t smaller, which means a lower breakdown voltage and more leakage, so density isn't free for them either. Real plates also have edges, where the field bulges out and adds capacitance that scales with the perimeter. For a square plate of side s:

$$C = \alpha s^2 + \beta s \tag{11.1}$$

The bigger the plate, the less the edge term matters, which is the same pattern you saw with Lab 01's end resistance.

Every capacitance in this lab's package is measured the same way: the deck hangs the device on a 1 V AC source at 1 MHz and reads the current the source supplies. For a capacitor $I = 2\pi f C V$, so $C = |I|/(2\pi f)$, which is a single division with no fitting.

Part 1: The area ladder

Does the density on the model card predict what a square MIM plate measures?

`spice/picofarad.spice` builds five square MIM capacitors with sides from 1 µm to 20 µm.

1. Read the density off the MIM model's parameter file:

```
grep -hE "^\+ c[ap]mimc" /foss/pdks/sky130A/libs.tech/ngspice/r+c/
res_typical__cap_typical__lin.spice
```

```
+ camimc=  2.00e-15  ; Units: farad/micrometer^2
+ cpmimc = 0.19e-15 ; Units: farad/micrometer
```

2. On paper, predict all five capacitances using only the first number, `camimc`.
3. Run the lab:

   ```
   make
   ```

   It ends in `FAIL`, which you'll fix in Part 6. Leave the output in your terminal, because the other parts read from it.
4. Find block A and compare it with your predictions.

You should see:

```
--- A. MIM area ladder, square plates (farads) ---
c_s1 = 2.642250e-15
c_s2 = 9.302250e-15
c_s5 = 5.328225e-14
c_s10 = 2.065822e-13
c_s20 = 8.131822e-13
```

For the 20 µm plate you predicted 800.0 fF, and it measures 813.18 fF, 1.6 % high. For the 1 µm plate you predicted 2.00 fF, and it measures 2.64 fF, 32 % high. Both come out high, but the relative error on the small plate is twenty times larger.

The two capacitors an analog designer most wants to match are usually a big one and a small one, such as a sampling capacitor and a reference, or the two arms of a divider. What happens to their ratio if you size both from the area term alone?

Answer: A1, in Answers at the end.

Part 2: Fit the area and edge terms

How many terms does it take to describe a MIM plate exactly?

$C = \alpha s^2 + \beta s$ has two unknowns, and you have five measurements.

1. Take the two extremes, $s = 1$ and $s = 20$, in femtofarads, and solve for α and β.
2. Compare with what the checker printed:

   ```
   two plates, two terms:  C[fF] = area*s^2 + edge*s
     area term   2.000887 fF per um^2       reference 2.000887
     edge term   0.641363 fF per um of side  reference 0.641363

     against the three plates it was not fitted to
       side   2.0 um   fit     9.2863 fF   measured     9.3022 fF   (-0.1717 %)
       side   5.0 um   fit    53.2290 fF   measured    53.2822 fF   (-0.0999 %)
       side  10.0 um   fit   206.5024 fF   measured   206.5822 fF   (-0.0386 %)
   ```

3. Look at the signs of those three residuals. Before you read the checker's one-line comment under them, decide what they tell you.

   Answer: A2, in Answers at the end.
4. Find the checker's three-term fit:

   ```
   three plates, three terms:  C[fF] = area*s^2 + edge*s + const
     area    2.000000    edge   0.659991    const  -0.017742
     against all five plates
       side   1.0 um   fit     2.6422 fF   measured     2.6422 fF   (+0.0000 %)
       side   2.0 um   fit     9.3022 fF   measured     9.3022 fF   (-0.0001 %)
       side   5.0 um   fit    53.2822 fF   measured    53.2822 fF   (-0.0001 %)
   ```

```
      side  10.0 um   fit   206.5822 fF   measured   206.5822 fF   (+0.0000 %)
      side  20.0 um   fit   813.1822 fF   measured   813.1822 fF   (+0.0000 %)
```

You should see: three coefficients fitted to three plates reproduce all five plates to one part in a million, and the area term comes out as 2.000000, which is the model card's `camimc = 2.00e-15` to seven figures.

This fit describes the device better than the headline density does, and you can invert it to answer the question you'll actually be asking when you design: what side length gives me 250 fF?

The card says the perimeter term is `cpmimc = 0.19e-15` farads per micrometre of edge, and a square of side s has $4s$ of edge, so you would expect $\beta = 0.76$. You measured 0.659991. Where did the difference go?

Answer: A3, in Answers at the end.

Part 3: Size one picofarad

How far off is a picofarad sized from the area term alone?

1. Work out the naive size: $A = 1000\,\text{fF}/2.00 = 500\ \mu\text{m}^2$, a square of side $\sqrt{500} = 22.3607$ µm. The deck contains it as `c_naive`.
2. Solve $2.000000\,s^2 + 0.659991\,s - 0.017742 = 1000$ for the corrected side. The deck contains it as `c_fixed`.
3. Predict how far from 1 pF the naive plate lands. Then find block B.

You should see:

```
--- B. one picofarad, naive vs fixed (farads) ---
c_naive = 1.014742e-12
c_fixed = 1.000001e-12
```

The naive plate gives 1.0147 pF, 1.47 % high. The corrected side, $s = 22.1965$ µm, measures 1.000001 pF, which is within one part in a million.

An error of 1.47 % on a single capacitor is usually something a circuit can live with. When isn't it?

Answer: A4, in Answers at the end.

Part 4: Three ways to build a capacitor

Which construction gives the most capacitance per square micrometre, and what does it cost you?

Block C measures the other two kinds of capacitor from Capacitors on a chip: MOS (an nFET gate and a pFET gate, biased at 1.8 V) and VPP (interleaved metal fingers). Block D then takes the bias away from one of the MOS capacitors.

1. Before reading the output, guess which construction is densest.
2. Find block C:

```
--- C. three technologies (farads) ---
c_nmos10 = 7.876883e-13
c_nmos11 = 9.999739e-13
c_pmos10 = 8.429616e-13
c_vpp = 1.164030e-13
```

   `c_nmos10` is a 10 µm × 10 µm nFET gate, and `c_nmos11` is the 11.2699 µm square that makes one picofarad.

3. Find the checker's price list, which turns each density into the area one picofarad needs:

```
  what one picofarad costs
    recipe                              fF/um^2   um^2 for 1 pF   adders
    MIM  cap_mim_m3_1                    2.0297          492.68     4.47
    MOS  nfet_01v8 gate at 1.8 V         7.8732          127.01     1.15
    MOS  pfet_01v8 gate at 1.8 V         8.4296          118.63     1.08
    VPP  m1-m4 wafflecap                 0.9116         1096.96     9.96
```

The last column is the area in DD103 4-bit ripple-carry adders, at 110.11 µm² each, as in Lab 01, Part 5.

4. Find block D, the 10 µm × 10 µm nFET capacitor with the gate at 0 V instead of 1.8 V.

You should see: one picofarad of MIM, drawn the way you would actually draw it, costs about four and a half ripple-carry adders. That's the 22.1965 µm square from Part 3, edges included, which is why its density is 2.0297 rather than 2.00. Block D reads:

```
--- D. the same MOS cap with the gate at 0 V (farads) ---
c_nmos_off = 2.207181e-13
```

That's 220.72 fF instead of the 787.69 fF of `c_nmos10`, a factor of 3.57 (the checker prints it too), and the only thing that changed was the gate voltage; every shape in the layout is the same.

The MOS capacitor is the densest by far. Why doesn't it go in a filter?
Answer: A5, in Answers at the end.

Part 5: The MOS capacitor's C–V curve

How does a MOS capacitor's value move with its gate voltage?

The numbers on the guide pages come from a second set of decks, in `labs/passives-decks`. Its `spice/c_area.spice` measures MIM, VPP and MOS capacitors at 1 kHz (the frequency cancels out of $C = |I|/(2\pi f)$, so any one works; see ngspice deck recipes, section 2), and its `spice/c_moscap_cv.spice` sweeps one MOS capacitor's gate from 0 V to 1.8 V in 100 mV steps.

1. Go to the `passives-decks` package and run both:

```
cd ../passives-decks
make capacitor
```

2. Divide each MIM capacitance in the first block, `--- capacitance in farads ---`, by its drawn area (1, 100 and 900 µm²).
3. Read `c_000mv` through `c_180mv` in the second block, `--- MOS cap, 10 um x 10 um gate ...`, and find the voltage range where the capacitance climbs fastest.
4. Go back to the lab folder:

```
cd ../lab-02-how-big-is-a-picofarad
```

You should see: MIM densities of 2.64, 2.07 and 2.02 fF/µm², falling as the plate grows. The sweep starts at 220.7182 fF at 0 V (`c_000mv = 2.207182e-13`), climbs steeply between 0.3 V and 0.7 V, and flattens out toward 787.6883 fF at 1.8 V (`c_180mv = 7.876883e-13`). This is the curve plotted on Capacitors on a chip.

What sets the voltage where the curve is steepest?
Answer: A6, in Answers at the end.

Part 6: Size your own capacitors

Can you size two MIM plates from your own three-term fit so that each lands within 1 % of its target?

`spice/my_cap.spice` has three capacitors, and you size the two MIM plates:

	device	target
C1	`cap_mim_m3_1`, square	250 fF
C2	`cap_mim_m3_1`, square	20 fF
C3	`nfet_01v8` gate at 1.8 V, square	250 fF

The file ships with all three sized from the area term alone, so the check in Part 1 failed on two of them:

```
  C1  cap_mim_m3_1, square
    target    250.00 fF
    got       257.36 fF   (+2.94 %)   FAIL
  C2  cap_mim_m3_1, square
```

```
    target     20.00 fF
    got        22.07 fF   (+10.35 %)   FAIL
  C3  nfet_01v8 gate at 1.8 V, square
    target    250.00 fF
    got       250.79 fF   (+0.31 %)   PASS
```

1. Before you change anything, work out why C3 passed with the naive sizing and the other two did not. C3 is in the file so that you'll ask this question.

   Answer: A7, in Answers at the end.

2. Solve for the side lengths of C1 and C2 using the three-term fit from Part 2 ($2.000000\ s^2 + 0.659991\ s - 0.017742$ fF), and write them down. Leave C3 as it is. The checker's explanation of each failure gives the side it would use, but compare yours with it only after you've worked out your own.
3. Edit the sizes in `spice/my_cap.spice`. Write them as plain microns, with no `u` suffix.
4. Re-run your file and the checker:

   ```
   make mine check
   ```

You should see: all three capacitors marked `PASS`, then

```
PASS -- every number matches the reference run.
```

Part 7: The three inductors

What inductors does SKY130 give you, and what do they cost in area?

`make` already ran `make henry`, which sweeps the three SKY130 inductor models from 100 MHz to 100 GHz.

1. Find out how many inductors SKY130 offers:

   ```
   ls /foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/ | grep "__ind_"
   ```

   ```
   sky130_fd_pr__ind_03_90.model.spice
   sky130_fd_pr__ind_05_125.model.spice
   sky130_fd_pr__ind_05_220.model.spice
   ```

   There are three fixed devices, with no width or turn-count parameters, so there is nothing to size. With the resistor in Lab 01, any value you wanted was just a number you typed.

2. Find the inductor table in the checker's output:

   ```
     inductor        L @100 MHz    Q max   at GHz   self-res GHz  inner um^2
     ind_05_220        9.9220 nH   12.718   1.3717         3.5236       48400
     ind_05_125        5.7857 nH   13.083   2.2539         6.5212       15625
     ind_03_90         1.5208 nH   21.126   6.3668        16.5015        8100
   ```

3. The last column comes from the PDK's own LVS deck, which describes these devices in a comment. Check it:

   ```
   grep "# # RF Inductor" /foss/pdks/sky130A/libs.tech/klayout/lvs/sky130.lvs
   ```

   ```
   # # RF Inductor with 3 turns and internal diameter of 90um
   # # RF Inductor with 5 turns and internal diameter of 125um
   # # RF Inductor with 5 turns and internal diameter of 220um
   ```

A hole 220 µm across in the middle is 48,400 µm², and the five turns of metal go around the outside of it, so 48,400 µm² is only a lower bound on the inductor's footprint.

4. Now look at the low-frequency end, which is what the guide page quotes, by running the `passives-decks` inductor deck:

```
cd ../passives-decks
make inductor
cd ../lab-02-how-big-is-a-picofarad
```

You should see: `q_1khz = 1.50516e-05` for `ind_05_220` in the `make inductor` output, which means that at 1 kHz the largest inductor is almost entirely resistance, as Inductors on a chip explains. The best Q for `ind_03_90` is about 21.1, near 6.4 GHz, which matches the `henry` table.

Here are the three passives side by side, all from your own runs in the same process:

you drew	area	in ripple adders
1 kΩ resistor (Lab 01, strip only, no contact heads)	1.96 µm²	0.018
1 pF MIM capacitor	492.68 µm²	4.5
9.92 nH inductor	≥ 48,400 µm²	≥ 440

On a schematic the three symbols are the same size, but in silicon their areas span a factor of twenty-five thousand.

A switching regulator needs inductors measured in microhenries. Could you put 10 µH on the chip?
Answer: A8, in Answers at the end.

Part 8: A 2.4 GHz resonator

Can you land a resonator on 2.4 GHz using only numbers you measured?

`spice/tank.spice` puts `ind_03_90` in parallel with a MIM capacitor and looks for the peak in its impedance. The capacitor is sized from numbers you already have:

$$L(2.4\,\mathrm{GHz}) = 1.554019\mathrm{nH} \Rightarrow C = \frac{1}{(2\pi f)^2 L} = 2829.85\mathrm{fF} \Rightarrow s = 37.4508\mu\mathrm{m} \tag{11.2}$$

using the three-term fit from Part 2.

1. Find the `ind_03_90` lines in the `henry` output from Part 1:

```
--- B. ind_03_90 at 2.4 GHz ---
l_090_2g4 = 1.554019e-09
q_090_2g4 = 1.086413e+01
r_090_2g4 = 2.157013e+00
```

Notice that 1.554019 nH is a little more than the 1.5208 nH the inductor shows at 100 MHz. At 2.4 GHz you are already at a seventh of its self-resonant frequency, and its apparent inductance has risen by 2.2 %.

2. Do the sizing arithmetic above yourself, from that inductance and the three-term fit.
3. Find the tank result in the output from Part 1. `fpeak` is the frequency of the impedance peak, and the number after `with=` is the peak impedance in ohms:

```
fpeak              =  2.40000e+09 with=  2.50130e+02
```

You should see: resonance at 2.4000 GHz, exactly where you aimed, with a peak impedance of 250.13 Ω. Since the resonance lands on the digit, your capacitor model, your inductor reading and your arithmetic must all agree. At 2.4 GHz the inductor has a Q of 10.86 and 2.157 Ω of series resistance.

The peak impedance is 250.13 Ω and the inductor's Q is 10.86. Work out how the two are related. Could a better capacitor have made the resonator better?

Answer: A9, in Answers at the end.

The whole resonator takes 8,100 µm² of inductor (a lower bound) plus 1,403 µm² of capacitor, the area of 86 ripple adders for one tuned circuit.

Finishing

You're done when `make mine check`, run on the C1 and C2 sides you worked out in Part 6, prints `PASS -- every number matches the reference run.`

An AC analysis has no randomness in it, so on the pinned image (ngspice-47, `sky130A`, `tt` corner, 27 °C) your numbers will match the ones on this page exactly.

Question: C1 and C2 were sized with the same mistake. Why did C2 miss by 10.35 % and C1 by only 2.94 %?

Answer: A10, in Answers at the end.

After yours pass, read `solutions/README.md` for the reference sizes, why C3 passed, and a check of what inductor geometry the PDK's GDS actually contains.

Submit: nothing for this lab.

Going further (optional)

None of these is in the time estimate, and no later part depends on them. The last line of this page says what to do next on the Course path.

Plot the inductor sweep

`make henry` also leaves the whole 100 MHz to 100 GHz sweep in `results/ind_sweep.txt`. To see it as a plot (there's nothing to install), run:

```
python3 src/plot_inductors.py      # -> results/inductors.svg
```

It prints `wrote results/inductors.svg`. The plot marks each self-resonance, where the apparent inductance crosses zero. Above that frequency the device behaves as a capacitor, even though it is still called an inductor.

Part 9 (optional): Parasitic capacitance in SiliWiz

Parasitic capacitance is easier to believe once you see that a wire is a bar hanging over a plane, and SiliWiz draws that cross-section live as you edit the layout.

1. Open **https://app.siliwiz.com/?preset=blank**.
2. Draw a **metal2** rectangle. Draw a second one beside it, not touching.
3. Label one `in` and the other `out` (click, then **Set Label** or press `S`).
4. Tick **Show SPICE** at the bottom and find the line starting `C0 out in`. That line is the coupling capacitance between two wires you never meant to connect.
5. Drag the rectangles closer together and watch the number.
6. Now put one on **metal1** and the other on **metal2**, overlapping.

You should see: the `C0 out in` value climbs as the two metal2 wires get closer. With one wire on metal1 and the other on metal2, overlapping, the capacitance is lower than for the two metal2 wires at their closest, because the vertical gap between metal layers is bigger than the horizontal gap you can draw between two wires on the same layer.

There is a guided walkthrough at https://tinytapeout.com/siliwiz/parasitics/, and the capacitor tutorial at https://tinytapeout.com/siliwiz/capacitors/ builds the same three kinds of capacitor as Capacitors on a chip.

> **Warning:** SiliWiz isn't SKY130. Its layer stack, permittivities and numbers are simplified for teaching, so use it for the shape of the answer (closer means more, a thicker gap means less, and edges matter), and use the PDK's `parameters/typical.spice` (see Parasitic capacitance) for the value.

Next on the Course path: read What to build instead of an inductor, then do Lab 03: A time constant in silicon, which puts your resistor and capacitor into the same circuit.

Answers

A1. Part 1: The area ladder

The small one comes out proportionally bigger, so the ratio you built differs from the ratio you designed, and measuring either capacitor on its own won't show you that.

A2. Part 2: Fit the area and edge terms

All three are negative, and they shrink as the plate grows. If the fit were only noisy, it would miss in both directions. A miss that always goes the same way and fades with size means the model is missing something small and constant, a term that doesn't scale with the plate at all. The checker's comment says the same thing:

```
-- every residual is negative. Something small and constant
   is missing, and it does not scale with the plate at all.
```

A3. Part 2: Fit the area and edge terms

The plate comes out smaller than the size you drew. The model computes with `wc = w + m3_dw`, where `m3_dw` is the **metal-3 width bias**, how much narrower the etched metal comes out than the mask. Substituting $s \to s - d$ into $2s^2 + 0.76s$ gives $2s^2 + (0.76 - 4d)s + (2d^2 - 0.76d)$. Setting $0.76 - 4d = 0.659991$ gives $d = 0.025002$ µm, and that same d predicts a constant term of −0.017751 fF, against the −0.017742 you measured. The checker prints both:

```
the edge term implies a metal etch bias of 25.00 nm per side,
which predicts a constant term of -0.017751 fF against the -0.017742 you measured.
```

So from five capacitance numbers you've measured a 25 nm etch bias, without anyone telling you it existed. It's the same thing you saw with `weff`, the effective width, in Lab 01, Part 4: the shapes on the silicon differ from the shapes on the mask, and the model accounts for the difference even when the datasheet's headline number doesn't mention it.

A4. Part 3: Size one picofarad

When the error lands on a ratio of two capacitors. A 1.47 % error in the ratio gives you a switched-capacitor filter with the wrong corner frequency, or an ADC that is a quarter of a bit non-linear. Capacitor ratios are the most precise thing you can get on a chip (better than 0.1 % is routine), and ignoring the fringe term throws that precision away.

A5. Part 4: Three ways to build a capacitor

It uses the gate oxide, the thinnest insulator in the process, which is why it is the densest. But it is a transistor, so it only works as a capacitor while there is a channel under the gate to act as the other plate. Remove the bias and 72 % of the capacitance goes with it. A capacitance that depends on the voltage across it causes distortion, so MOS capacitors go on the supply rails by the million, where the voltage never moves, and stay out of signal paths. Density, linearity and mask cost (MIM needs extra masks; see the table on Capacitors on a chip) are separate properties, and no device is best at all three.

A6. Part 5: The MOS capacitor's C–V curve

The transistor's threshold voltage sets it. As you found in Part 4, a MOS capacitor only has its full value while there is a channel under the gate to act as the other plate, and the channel forms around the threshold.

A7. Part 6: Size your own capacitors

All three naive sizings dropped the edge term. The edge term is a perimeter effect, so what matters is the ratio of perimeter to area, which for a square of side s is $4/s$ and grows as the plate shrinks. C3′s dielectric is gate oxide, so its area term is about four times the MIM's, while its edge term comes mostly from the small gate-to-source and gate-to-drain overlaps. With an area term that large, the edges add comparatively little. The lab's `solutions/README.md` explains this at more length.

A8. Part 7: The three inductors

No. An inductance of 9.92 nH is no use for anything but radio. Even if inductance scaled linearly with area (it doesn't; a spiral does considerably worse), 10 µH would need 4.9×10^7 µm², which is 49 mm², larger than most entire chips. That's why every switching regulator you've seen has an inductor soldered next to the chip rather than inside it. It also answers the question on the course's home page, how you fabricate an inductor on a chip: mostly, you don't.

A9. Part 8: A 2.4 GHz resonator

$250.13/(2\pi \times 2.4\ \text{GHz} \times 1.554\ \text{nH}) = 10.7$, close to the inductor's own Q at 2.4 GHz, 10.86. So the capacitor contributed almost no loss, and nearly all of the resonator's loss comes from the inductor, which

leaves a better capacitor almost nothing to improve. That is the usual case on silicon, and it is why people quote on-chip Q for inductors and never for capacitors.

A10. Finishing

They're the same type of plate, but C2′s is much smaller, and the edge term's share of a square plate grows as $4/s$ as the plate shrinks (Part 6, step 1). That's why the same mistake costs C2 3.5 times as much.

III 5–6. RC on a chip

12 What to build instead of an inductor . 65

12 1. Use R and C, at a different frequency 65

12 2. Switched capacitors 65

12 3. Gyrators 66

12 4. Put the inductor off the die 66

12 When to use an on-chip spiral 66

12 What the four answers have in common 66

13 Lab 03: A time constant in silicon . . 67

13 Before you start 67

13 Part 1: Measure two time constants you can predict 67

13 Part 2: Swap in your own resistor and capacitor . 68

13 Part 3: Sizing without Labs 01 and 02 68

13 Part 4: Temperature 69

13 Part 5: A one-microsecond RC 69

13 Part 6: Your own 20 ns time constant 69

13 Part 7: The same RC in the frequency domain . . 70

13 Finishing 71

13 Going further (optional) 72

13 Answers 74

12. What to build instead of an inductor

Without a usable inductor, you might wonder how anyone builds a filter, an oscillator or a power supply on a chip. The answer is to work out what the inductor was doing in the circuit and get that some other way. There are four standard ways to do it, and the first three are used all the time.

1. Use R and C, at a different frequency

Most of what an inductor does in a discrete design, such as setting a corner frequency or shaping a response, an RC network can also do, with one fewer pole per stage. The catch is that an RC corner depends on R × C, and you already know what each of those costs in area:

$$f_{-3\ \mathrm{dB}} = \frac{1}{2\pi RC} \tag{12.1}$$

A 1 kHz corner needs $RC = 159$ µs. If you use the 100 kΩ resistor from What a resistor costs in area (314 µm² of poly), the capacitor has to be 1.6 nF, and that capacitor alone is 791 300 µm², or 0.79 mm². That's about 9 % of a 3 mm × 3 mm die, or the area of 210 800 inverters, for a single pole, so nobody builds one.

So on a chip, the designer moves the corner instead. The corners deck in `labs/passives-decks` builds an RC low-pass from one 3550 Ω resistor and one 206.5822 fF MIM capacitor:

```
--- tt : RC corner frequency, Hz ---
f3db                =  2.15410e+08
```

That's 215.410 MHz. With a resistor and a capacitor of a size you can afford, the corner lands five orders of magnitude above the 1 kHz you wanted, so anything slower than a few megahertz is filtered off-chip, or with one of the next two techniques. Lab 03 builds an RC from the resistor and capacitor you designed and measures its time constant in the time domain.

2. Switched capacitors

Switched capacitors are the most important trick in analog IC design, and they exist because of the two facts you have just learned about what resistors and capacitors cost in area.

Take a capacitor C and a two-phase clock at frequency f_{clk}. Connect C alternately to node A and to node B. Each cycle it carries a charge $Q = C\,\Delta V$ from one node to the other, so the average current is $I = f_{clk}\,C\,\Delta V$. That is Ohm's law with

$$R_{\mathrm{eq}} = \frac{1}{f_{clk}\,C} \tag{12.2}$$

So a capacitor plus a switch acts as a resistor, and its value is set by a clock frequency rather than by a length of poly.

To put numbers on it, take a 100 fF capacitor (about 49 µm², the area of thirteen inverters) and clock it at 1 MHz:

$$R_{\mathrm{eq}} = \frac{1}{10^{6} \times 100 \times 10^{-15}} = 10\ \mathrm{M}\ \Omega \tag{12.3}$$

So you get 10 MΩ in 49 µm². Building the same 10 MΩ as a 1 µm wide `res_xhigh_po` resistor would take a strip $(10^{7} - 34.2111)/2118.7619 = 4719.7$ µm long, which is 4.72 millimetres (wider than most dies) and 4720 µm² of poly. The switched capacitor uses about ninety-six times less area, and unlike a 4.72 mm strip, it's something you can actually place.

Switched capacitors have two more advantages:

- **The value is tunable.** Changing the clock changes the resistance, so you can retune it without new masks.

- **The time constant becomes a capacitor ratio.** An RC made from a switched capacitor C_1 and an integrating capacitor C_2 has $\tau = R_{eq}C_2 = C_2/(f_{clk}C_1)$, which is a ratio of capacitors times a clock period. A ratio of matched devices is the most accurate thing you can make on a chip (see Matching and Pelgrom's law), and that's why almost every ADC you will meet is built from switched capacitors.

The price is that the circuit is now sampled, so it only works below $f_{clk}/2$, and it folds noise. AD202 covers both of those properly.

3. Gyrators

An inductor obeys $V = L\,dI/dt$ and a capacitor obeys $I = C\,dV/dt$: the same equation with current and voltage swapped. A circuit that swaps current and voltage, called a **gyrator** and built from two transconductors, turns a capacitor at one port into an inductance at the other:

$$L_{\text{eq}} = \frac{C}{g_{m1}g_{m2}} \tag{12.4}$$

With $g_m = 100$ µS and $C = 1$ pF you get $L_{eq} = 100$ µH, which is ten thousand times the biggest spiral in the PDK, from about 500 µm² of MIM plus two small amplifiers.

The catch is that a gyrator is *active*: it needs supply current, adds noise, has a limited signal swing, and stops working above the transconductors' bandwidth. So what you get is a circuit that behaves like an inductor over a limited range, which is usually enough. Filters built this way, called "active inductor" or "simulated inductor" filters, are standard practice at audio and low RF.

4. Put the inductor off the die

If you really need an inductor (a switching regulator, an RF matching network, a high-current supply), you buy one and put it on the board. Two on-die pads and a package pin cost a few thousand µm² and get you a component with a Q of 40 and a value in microhenries.

Designers use the package itself, too. A bond wire has roughly 1 nH of inductance per millimetre, and RF designers deliberately make that inductance part of the matching network instead of fighting it.

When to use an on-chip spiral

The three SKY130 spirals still have their uses. Above about a gigahertz, where Q climbs past 10, an on-die inductor is the right answer, and often the only one, for jobs like these:

- the **LC tank** of a voltage-controlled oscillator, where Q sets the phase noise;
- **inductive peaking** on a wideband amplifier, where a couple of nanohenries extends the bandwidth by tens of percent;
- **impedance matching** to a 50 Ω antenna.

Those are the jobs the spiral cells are for. It's a narrow niche, though a real and well-paid one, and neither AD102 nor AD103 works in it.

What the four answers have in common

All four answers rest on the same principle:

> **On a chip, absolute values are expensive and ratios are cheap. So build what you need out of ratios, and get the absolute scale from something that is not silicon: a clock, a bias current, or a component on the board.**

A switched-capacitor filter's corner is a capacitor ratio times a clock. A gyrator's inductance is a capacitance over two transconductances, which are themselves set by a bias current. A bandgap reference's output is a resistor ratio times a physical constant.

The principle follows directly from a measurement you'll see on the next two guide pages, which show that on-chip absolute values are poor and on-chip ratios are essentially exact.

Next on the Course path: Lab 03: A time constant in silicon. After that, read Process corners.

13. Lab 03: A time constant in silicon

You designed a resistor in Lab 01 and a capacitor in Lab 02. In this lab you'll put them in one circuit and measure its time constant, learn to tell a measurement artefact from a real device effect, and find the split between R and C that gives a target τ in the least area. Then you'll measure the same RC in the frequency domain, and see why chips time long intervals by counting clock ticks rather than by charging a capacitor.

Package: `labs/lab-03-a-time-constant-in-silicon/`

Before you start

- Finish Lab 01 and Lab 02. This lab uses the resistor ($L = 30.3315$ µm) and capacitor ($s = 22.1965$ µm) you sized there, and Lab 01's `solutions/README.md`.
- Read What to build instead of an inductor.
- Open a terminal in the workbench and go to the lab folder:

```
cd /foss/designs/modules/ad102/labs/lab-03-a-time-constant-in-silicon
```

The decks don't need any environment setup. Only the XSchem window in the optional Part 8 needs the noVNC desktop and `$PDK` set to `sky130A` (Lab 01, Part 1).

Parts 1 to 7 are required. Part 8, under Going further at the end, is optional.

Time: about 2 hours, not counting the optional part. `make` runs two ngspice decks, and each one prints nothing while the SKY130 model cards load, which takes about a minute per deck on the pinned image (see Warnings and delays).

A resistor charging a capacitor from a step follows

$$v(t) = V_f\left(1 - e^{-t/RC}\right) \qquad \tau = RC \tag{13.1}$$

At $t = \tau$ the output has covered $1 - e^{-1} = 0.632120$ of the way to its final value. So to measure τ you step the input and find the moment the output crosses 63.2 % of its final value; that time is τ by definition. Every `meas` in this lab is a single line that does exactly this.

Part 1: Measure two time constants you can predict

Does a `meas` on a simple RC return the τ you predict?

Section A of `spice/rc.spice` is ECE334's lab-1 RC deck, unchanged: a 1 kΩ/2 kΩ divider with 0.7 pF on the tap, driven by a 1.8 V pulse. If you're a UofT student, it's probably a version of this circuit you have already run. Section B is an ideal 10 kΩ and an ideal 1 pF, so its τ is 10 ns by construction.

1. Predict both time constants. In section A the capacitor charges through the Thévenin resistance of the divider rather than through the 1 kΩ alone:

$$R_{\text{th}} = 1\ \text{k} \parallel 2\ \text{k} = 666.67\ \Omega \qquad \tau = 666.67 \times 0.7\,\text{pF} = 466.67\,\text{ps} \tag{13.2}$$

   and it charges toward $1.8 \times \frac{2}{3} = 1.2$ V.

2. Run the lab:

```
make
```

   It ends in `FAIL` because of area, which you'll fix in Part 6. Leave the output in your terminal.

3. Read the raw `meas` lines near the top of the output, rather than the checker's table:

```
t_ece334            =  1.47167e-09
t_ideal             =  1.10050e-08
v_ece334            =  1.20000e+00
```

4. The step happens at $t = 1$ ns, so subtract 1 ns from each time and compare with your predictions.

You should see: a final value of 1.2000 V, as Thévenin predicts. The times are slightly off, though: the ECE334 circuit took 471.67 ps against a predicted 466.67, and the ideal RC took 10.0050 ns against a predicted 10.0000. Both are 5 ps late, even though their time constants differ by a factor of twenty-one.

An error that stays the same size whatever you measure can't be coming from the thing you're measuring, so look at what the two circuits have in common. The pulse source in the deck is

```
Va ai 0 PULSE(0 1.8 1n 10p 10p 200n 400n)
```

Where do the 5 ps come from?

Answer: A1, in Answers at the end.

Part 2: Swap in your own resistor and capacitor

Does the RC still give 10 ns when the parts are the ones you designed?

Sections C, D and E of the deck replace the ideal 10 kΩ with your Lab 01 resistor ($L = 30.3315$ µm), then the ideal 1 pF with your Lab 02 capacitor ($s = 22.1965$ µm), then both.

1. Predict all three. You designed both devices to hit their targets to better than one part in ten thousand, so your prediction should be 10.0000 ns each time.
2. Find lines C, D and E in the checker's table.
3. Find section H in the raw output, the block headed `--- H. the resistor's own capacitance to substrate`. It leaves the far end of each resistor unconnected and measures the current the source still has to supply, and from that current it works out the resistor's own capacitance to the substrate.

You should see:

```
  C  YOUR Lab 01 resistor + ideal 1 pF                  10.0361 ns
  D  ideal 10 kohm + YOUR Lab 02 capacitor              10.0000 ns
  E  both of yours, no ideal parts anywhere             10.0362 ns
```

```
--- H. the resistor's own capacitance to substrate (farads) ---
cpar_10k = 7.233642e-15
cpar_1m = 9.527372e-14
```

Your capacitor makes no difference: replacing the ideal 1 pF with a real MIM plate changes nothing to five figures. Your resistor adds 36.1 ps, and it adds the same 36.1 ps whether the capacitor next to it is real or ideal. The checker fits a line through the two strips from section H and prints the result below its table:

```
    fitting both strips: 0.1994 fF per um of 1 um-wide poly, plus 1.1869 fF of ends.
```

You'll use that 0.1994 fF per µm in Parts 5 and 6.

How does section H account for the 36 ps?

Answer: A2, in Answers at the end.

Part 3: Sizing without Labs 01 and 02

What would this circuit have done if you had sized it from the model cards alone?

Section F sizes the same circuit with the two naive rules: the resistor's length comes straight from the sheet resistance, and the capacitor's area straight from the capacitance density.

1. In Lab 01 the end resistance was worth 3.7 % on a 10 kΩ resistor, and in Lab 02 the fringe was worth 1.5 % on a picofarad. Predict the timing error of the naively sized circuit.
2. Find line F and the two summary lines below the table.

You should see:

```
F  the same circuit sized without Labs 01 and 02       10.5647 ns

designing with Labs 01 and 02 : +0.36 % from the 10.0000 ns you asked for
designing without them        : +5.65 %
```

Together with the resistor's own 0.36 % from Part 2, the two sizing errors add up to 5.65 % of timing error in one circuit. Neither of them shows up as a warning or in DRC, and neither averages out across a wafer, because every chip gets the same 5.65 %. That's the difference Lab 01, Part 3 drew between a **tolerance**, which you budget for, and a **mistake**, which you don't know is there.

Part 4: Temperature

How much does the same silicon drift between −40 °C and +125 °C?

The deck runs section E again at −40 °C and at +125 °C, with no change to the layout.

1. Predict the direction before you look. Lab 01's `solutions/README.md` gave `res_high_po`'s temperature coefficient as `tc1 = 0.514e-3`.
2. Find the two "E again" lines and the span line.

You should see:

```
   E again at -40 C                                    9.7849 ns
   E again at +125 C                                  10.6544 ns

and the SAME silicon over -40 C to +125 C spans 8.66 %
```

You've spent this lab chasing a 5.65 % sizing error, and temperature on its own is worth 8.66 % over an industrial range. The MIM capacitor's model card has `tc1 = 0`. What does that tell you about where the drift comes from, and about using an on-chip RC as a clock?

Answer: A3, in Answers at the end.

Part 5: A one-microsecond RC

What happens to a real RC when the resistor gets big?

Section G of the deck builds a 1 µs RC twice: once with an ideal 1 MΩ, and once with a real one made of 471.96 µm of `res_xhigh_po`, sized with your Lab 01 numbers. The checker's table calls these two G and H.

1. Using the 0.1994 fF per µm from Part 2, predict the error before you look.
2. Find lines G and H in the checker's table, and the lines about the microsecond below it. (This H is the checker's line, not section H of the raw output, which you read in Part 2.)

You should see:

```
G  ideal 1 Mohm + ideal 1 pF                          999.9950 ns
H  real 1 Mohm of xhigh poly + your 1 pF             1049.8950 ns

the microsecond ran 49.90 ns long.
  a 471.96 um strip of poly carries 95.274 fF to the substrate, and half of it
  adds to the time constant: R*Cpar/2 = 47.64 ns, which is 95 % of what you measured.
```

The microsecond runs 5 % long without any capacitor being mis-sized, because the strip that makes the megohm carries 95 fF of its own, about a tenth of the capacitor it is charging.

A millisecond needs a thousand times this RC, say 100 MΩ and 10 pF. Could you build it?

Answer: A4, in Answers at the end.

Part 6: Your own 20 ns time constant

How small can you make a 20 ns RC?

`spice/my_rc.spice` has one resistor length and one capacitor side for you to choose. Your job is to build a 20.0 ns time constant, within 1 %, in under 400 µm² of drawn area.

The file ships with 2 kΩ and 10 pF. That gives a perfectly good 20 ns, but it costs 4981.84 µm², twelve times the budget, which is why the check in Part 1 failed on area rather than on timing:

```
resistor  L =   5.1124 um at W = 1 um   ->       5.11 um^2
capacitor s =  70.5459 um square       ->    4976.72 um^2
total drawn area                         ->    4981.84 um^2   (budget 400)   FAIL
tau = 20.0020 ns   target 20.0000 ns   (+0.01 %)   PASS
```

1. Write down the two sizing formulas you already have, and the area:

$$R = 378.2448 + 317.2198\,L \qquad C[\text{fF}] = 2.000000\,s^2 + 0.659991\,s - 0.017742 \tag{13.3}$$

   Area is $L + s^2$.

2. Work out which way to move R and C to reduce the area while keeping $RC = 20$ ns, and find the optimum.

3. Account for the resistor's own parasitic, which from Part 2 is 0.1994 fF per micrometre of length plus 1.187 fF for the ends. As the capacitor gets smaller, the parasitic becomes a visible fraction of it and τ runs long, because half of the parasitic adds to the capacitor: $\tau \approx R\,(C + C_{\text{par}}/2)$. Make the capacitor you draw smaller by enough to make up for it.

4. In `spice/my_rc.spice`, change the resistor's `L=` and the capacitor's `W=` and `L=` (keeping the plate square). Write them as plain microns with no `u` suffix, then re-run:

```
make mine check
```

You should see: both the area line and the `tau` line marked `PASS`, then

```
PASS -- every number matches the reference run.
```

Part 7: The same RC in the frequency domain

Does the corner frequency from an `.ac` analysis agree with the τ you measured?

So far you have measured the RC the way you would with a stopwatch, by stepping the input and timing the output. AD101 taught the other way, which is to drive the input at every frequency and find where the output stops keeping up. The two answers are tied together by

$$f_{-3\text{ dB}} = \frac{1}{2\pi\tau} \tag{13.4}$$

`spice/bode.spice` uses the same two devices as section E, and also simulates an ideal 10 kΩ and an ideal 1 pF, the same pair as section B of `rc.spice`, for comparison.

1. Take your τ from Part 2, 10.0362 ns, and predict the corner frequency.

2. Run the frequency-domain deck and draw its plot:

```
make bode
```

   It loads the same model library as the other decks, so expect the same wait.

You should see:

```
f3db_real           =  1.58580e+07
f3db_ideal          =  1.59155e+07
ph_real             =  -7.85359e-01
--- decade of roll-off: the real filter, dB at 10x the corner ---
g_10x               =  -2.00426e+01
  wrote results/ad102-rc-bode.png
```

That PNG is your own copy of the figure below, drawn from your own `results/bode.txt`.

Your prediction was

$$\frac{1}{2\pi \times 10.0362\,\text{ns}} = 15.858088\,\text{MHz} \tag{13.5}$$

and `.ac` measured 15.8580 MHz. They agree to five significant figures, about six parts per million apart, from two analyses that share nothing but the two device lines. Agreement between two independent measurements is the only evidence you get that a deck is right, and checking one against the other costs nothing, so make a habit of it. With any RC you have three numbers, R, C and τ, and any two of them give you the third. If they disagree, one of your decks is wrong, and you've found out cheaply.

The other two lines confirm the shape of a first-order filter:

- `ph_real = -7.85359e-01` is in **radians**. It's −44.998°, the textbook −45° at the corner.
- `g_10x = -2.00426e+01` is −20.0426 dB one decade past the corner. That's the −20 dB/decade slope you read off Bode plots in AD101, measured this time on a resistor and capacitor you designed.

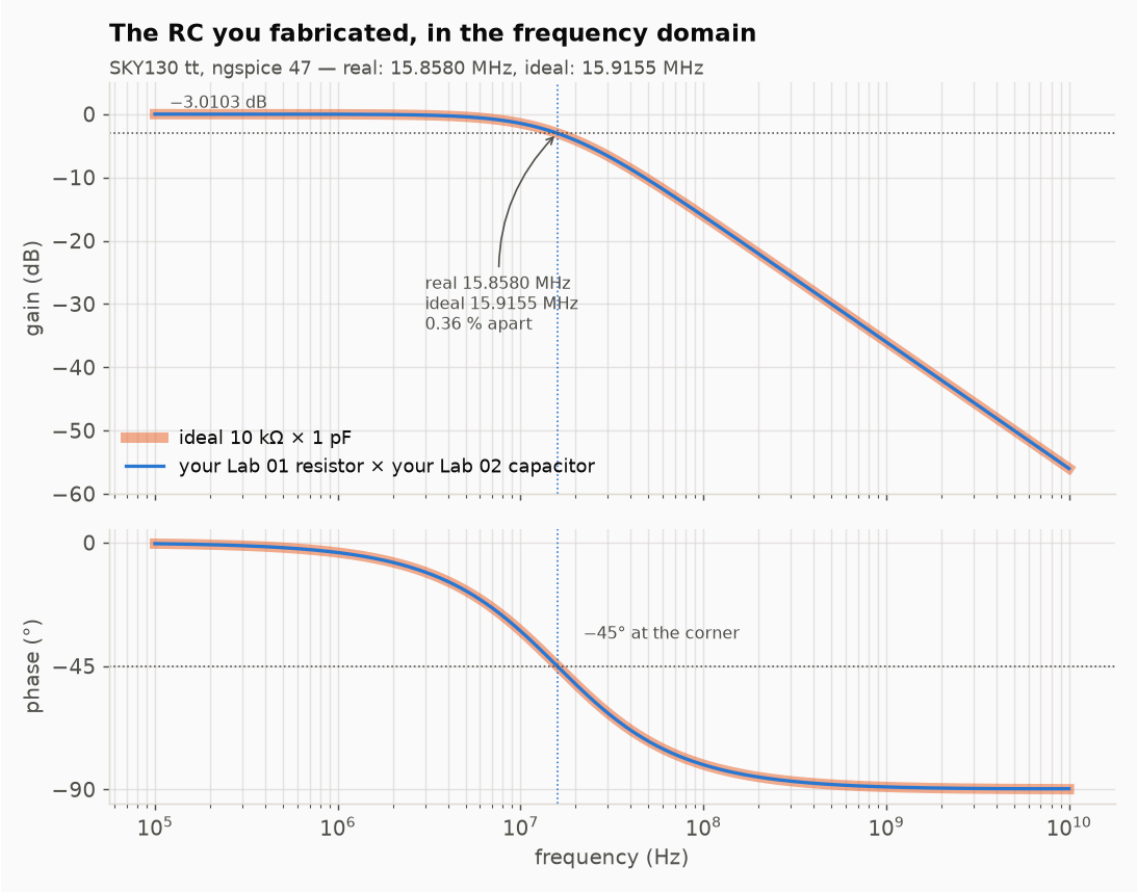


Figure 13.1: Bode plot: gain and phase of the fabricated RC overlaid on the ideal 10 kΩ × 1 pF, corner at 15.8580 MHz vs 15.9155 MHz

How far is your fabricated corner from the ideal one, and how far would the Part 3 design have been?

Answer: A5, in Answers at the end.

Finishing

You're done when `make mine check`, run on the resistor length and capacitor side you chose in Part 6, prints `PASS -- every number matches the reference run.` Your design doesn't have to match anyone else's; it passes when τ is within 1 % of 20 ns and the area is under 400 µm². The "matches the reference run" part refers to the `rc.spice` numbers, which the checker also compares.

A transient analysis of a linear circuit has no randomness in it, so on the pinned image (ngspice-47, `sky130A`, `tt` corner) your numbers will match the ones on this page.

Question: in your Part 6 design, compare the resistor's area with the capacitor's. Why should they come out roughly equal?

Answer: A6, in Answers at the end.

Once yours passes, read the same `solutions/README.md`. It gives one good answer, explains why the two halves of the area bill come out equal, and covers the millisecond that can't be built.

Submit: nothing for this lab.

Going further (optional)

Part 8 isn't in the time estimate, and nothing later in AD102 needs it. What to do next on the Course path is in the last line of this page.

Part 8 (optional): The same circuit in XSchem

So far you have typed device lines and ngspice has taken them as written. The rest of the analog track describes circuits the other way: you **draw** them, and the tool writes the device lines for you.

`xschem/rc.sch` is section E of `spice/rc.spice` as a drawing, with the same resistor (30.3315 µm of high-sheet poly), the same capacitor (a 22.1965 µm square of MIM) and the same pulse.

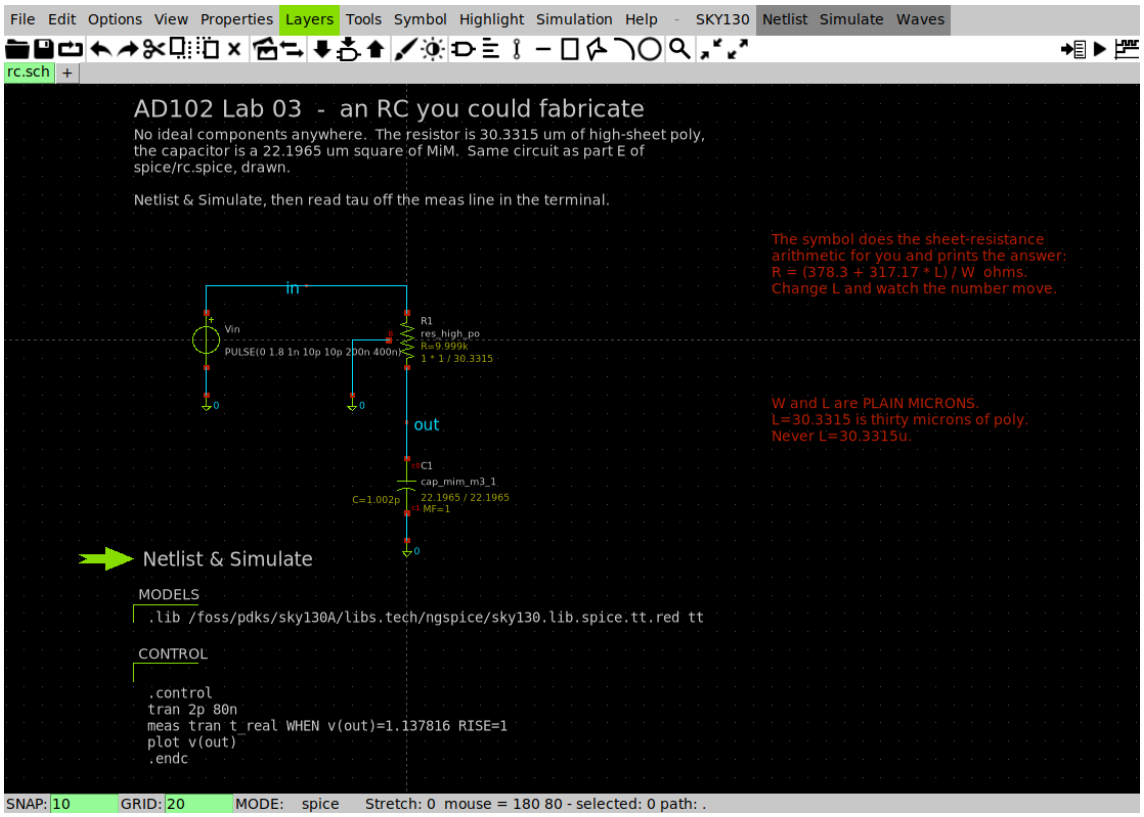


Figure 13.2: XSchem showing rc.sch: a pulse source, a res_high_po symbol annotated R=9.999k, a cap_mim_m3_1 symbol annotated C=1.002p, and the MODELS and CONTROL text blocks below

The **SKY130** badge in the top right shows which parts catalogue XSchem loaded. If yours says **IHP**, stop and redo the PDK steps in Lab 01, Part 1. The resistor symbol prints `R=9.999k` beside itself, because the SKY130 symbols compute the resistance live from the `W` and `L` you give them, and the capacitor shows `C=1.002p` in the same way.

1. Turn the drawing into a netlist. You don't need the desktop for this:

```
make netlist
```

```
== netlisting xschem/rc.sch
   wrote xschem/simulation/rc.spice
XR1 out in 0 sky130_fd_pr__res_high_po W=1 L=30.3315 mult=1
XC1 out 0 sky130_fd_pr__cap_mim_m3_1 W=22.1965 L=22.1965 MF=1
```

Those are the two device lines XSchem wrote from the picture, and they're the same two lines that section E of `spice/rc.spice` has typed by hand. A schematic is a netlist you can see.

2. Simulate what XSchem wrote:

```
cd xschem/simulation && ngspice -b rc.spice; cd ../..
```

It loads the same model library, so there's the same wait. The output includes a warning you can ignore, which appears because the schematic asks for a plot that batch mode can't draw:

```
Warning: command 'plot' is not available during batch simulation, ignored!
    You may use Gnuplot instead.
```

The `meas` still ran, and its answer is the line below the warning.

3. If you have the noVNC desktop up, open the schematic:

```
echo $PDK        # must say sky130A
make edit
```

XSchem opens on the drawing. Click **Netlist & Simulate** in the top left of the canvas, and the waveform appears in its own window.

4. Double-click the resistor symbol:

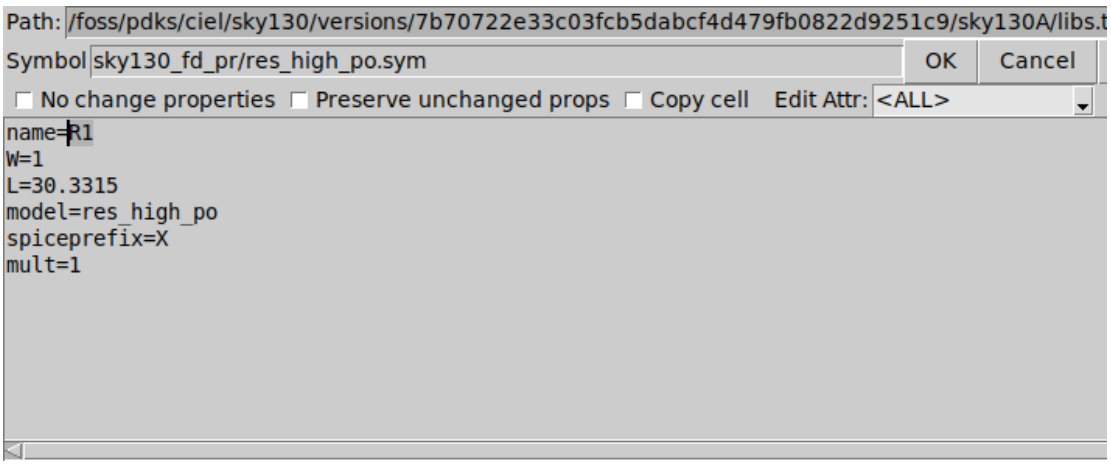


Figure 13.3: The XSchem properties dialog for R1, listing name=R1, W=1, L=30.3315, model=res_high_po, spiceprefix=X, mult=1

That's your device line, laid out as a form. Change `L`, press **OK**, and watch the printed `R=` on the symbol change. Then re-simulate and watch τ follow it.

Notice that the dialog has no units anywhere. `L=30.3315` is thirty micrometres, and `L=30.3315u` would be thirty millionths of a micrometre, which the schematic editor will accept just as happily as a text deck would.

You should see: after step 2,

```
t_real             =  1.10412e-08
```

Subtract the same 1.005 ns offset as in Part 1 and you get $\tau = 10.0362$ ns, the same five figures as section E. The picture and the deck agree because they describe the same circuit.

AD103 is done entirely in XSchem (you'll draw a transistor, a diode and then a CMOS inverter). Its Lab 01 starts from an empty canvas and doesn't assume you have used XSchem before, so treat this part as a preview rather than a prerequisite.

Next on the Course path: read Process corners and Matching and Pelgrom's law, then do Lab 04: Two resistors that disagree.

Answers

A1. Part 1: Measure two time constants you can predict

The rise time is 10 ps, so the step is really a ramp, and a first-order system responds as though the step arrived at the ramp's midpoint, which is 5 ps after the ramp starts. `check_rc.py` subtracts 1.005 ns instead of 1 ns, and then both circuits read exactly right:

```
A  ECE334 deck: 1k/2k divider + 0.7 pF, ideal         466.6700 ps    466.6700 ps
B  ideal 10 kohm + ideal 1 pF                          10.0000 ns     10.0000 ns
```

Telling a **measurement artefact** from a **device effect** is most of what analog verification is, and the clue here was that the error didn't scale with τ.

A2. Part 2: Swap in your own resistor and capacitor

A 30 µm strip of poly is also a 7.2 fF capacitor, whether you wanted one or not. For a distributed RC line, half of that capacitance adds to the time constant: $10\,\mathrm{k}\,\Omega \times 7.234\,\mathrm{fF}/2 = 36.2$ ps, against the 36.1 ps you measured. At 10 kΩ this costs you 0.36 %, and Part 5 finds the size of resistor where you can no longer ignore it.

A3. Part 4: Temperature

The MIM capacitor contributes almost none of the drift, so nearly all of it comes from the resistor. An on-chip RC is a fine way to set a filter corner that is allowed to wander, and a bad way to make a clock.

A4. Part 5: A one-microsecond RC

No. At 1 µm wide, 100 MΩ of `res_xhigh_po` is 47,197 µm of poly, which is 4.7 centimetres folded into a serpentine, and it carries about 9.4 pF to the substrate by itself. The resistor's parasitic is then about the same size as the capacitor it was supposed to charge, and no choice of either value fixes that.

So you can't build a millisecond from an RC on a chip. What chips do instead is count: an oscillator makes a short, repeatable interval, and a digital counter multiplies it. That's why DD103's elevator controller measures its door dwell in clock ticks. It's one of the cleanest examples anywhere of a digital answer to an analog question, and this measurement shows you why the question comes up.

A5. Part 7: The same RC in the frequency domain

Your corner is 0.36 % from the ideal one you drew on paper, and that's what Labs 01 and 02 (the end resistance, the perimeter capacitance, the sizing you got wrong on the first try) bought you. The same filter designed without those two labs, in Part 3, lands 5.65 % out, which is sixteen times worse.

A6. Finishing

Resistor area is roughly $R/317.22$ µm² and capacitor area roughly $C[\mathrm{fF}]/2$ µm², and $C = \tau/R$, so the total has the form $aR + b/R$. Whenever a cost has that form, its minimum is where the two terms are the same size. The reference answer in `solutions/README.md` has 176.36 µm² of resistor and 164.43 µm² of capacitor, and works through why it isn't exactly 50/50.

IV 7–8. Variation and matching

14 Process corners 77
14 Global variation and local mismatch 77
14 The corner files 77
14 One circuit at three corners 78
14 R and C errors together 78
14 Local mismatch 79
14 A rule for specifications 79

15 Matching and Pelgrom's law 81
15 Ratios across corners 81
15 Why the ratio holds 81
15 Mismatch between identical devices 82
15 Pelgrom's law 83
15 Four rules of analog layout 84
15 How analog circuits use matching 84

16 Lab 04: Two resistors that disagree . 87
16 Before you start 87
16 Part 1: Process corners 87
16 Part 2: Run the Monte Carlo 88
16 Part 3: Pelgrom's law 88
16 Part 4: The same divider at 48 times the area . . 89
16 Part 5: One ratio drawn two ways 89
16 Part 6: Design a 0.45 V divider 90
16 Finishing 91
16 Going further (optional) 91
16 Answers 91

14. Process corners

Suppose you size a resistor to 100 000.0 Ω, and the simulator agrees with you to seven digits. The resistor that comes back from the fab will be somewhere between 87.5 kΩ and 112.5 kΩ, and nobody, SkyWater included, can tell you where in that range until the wafer has been measured.

You might take that for a defect, but it's the normal, documented behaviour of a semiconductor process, and the PDK ships the numbers so that you can design around them. This page is about how large the variation is, and the next page is about the technique that makes it manageable.

Global variation and local mismatch

A drawn value can be wrong in two different ways. Both look like "error", and people often confuse them.

Global variation, also called **process corners.** Perhaps the furnace ran three degrees warm, the implant dose was at the top of its window, or the poly etch was slightly aggressive. Changes like these shift **every device on the wafer in the same direction, by roughly the same amount**. If your 100 kΩ became 112 kΩ, the 10 kΩ next to it moved by the same proportion.

Local mismatch. Dopant atoms land at random, so two resistors drawn identically, five microns apart on the same die, still differ. A 1 µm × 1 µm strip contains a countable number of impurity atoms, and the count isn't the same in both. This variation is uncorrelated, which means it's different for every pair, and no amount of process control removes it, because it comes from thermodynamics.

You handle corners with design margin, and mismatch with area and layout. If you confuse the two, you get circuits that pass every corner in simulation and then fail on silicon.

The corner files

SKY130 ships the corners as plain parameter files. `libs.tech/ngspice/r+c/` contains `res_low__cap_low.spice`, `res_typical__cap_typical.spice` and `res_high__cap_high.spice`, and `sky130.lib.spice` wires them into library sections named `ll`, `tt` and `hh`. Here are the sheet resistances from the three files side by side, in Ω/□ except where the row says otherwise:

layer	`ll`	`tt`	`hh`	span
n+ diffusion (`rdn`)	111.6	120	128.4	±7.0 %
p+ diffusion (`rdp`)	175.3	197	218.7	±11.0 %
poly (`rp1`)	44	48.2	53.52	−8.7 / +11.0 %
n-well (`rnw`)	1378	1700	2022	±18.9 %
met1 (`rm1`)	0.111	0.125	0.139	−11.2 / +11.2 %
poly contact (`rcp1`, Ω each)	61.28	145.28	213.88	−58 / +47 %

For the precision poly resistor, the corner files set a single explicit knob:

```
+ sky130_fd_pr__res_high_po__var = -0.125      (res_low__cap_low__lin.spice)
+ sky130_fd_pr__res_high_po__var = 0.0         (res_typical__cap_typical__lin.spice)
+ sky130_fd_pr__res_high_po__var = 0.125       (res_high__cap_high__lin.spice)
```

That is **±12.5 %**, written down by the foundry as a multiplier on `rsheet`. The MIM capacitor gets the same treatment: `camimc` is `1.778e-15`, `2.00e-15` and `2.231e-15` F/µm², or −11.1 % / +11.6 %.

The same files also hold a second mechanism, a list of *width* tolerances:

```
+ tol_poly = 0.0287u
+ tol_nfom = 0.0483u
+ tol_m1   = 0.0175u
+ tol_m5   = 0.119u
```

This means your poly can come out 28.7 nm narrower or wider than you drew it. On a 1 µm-wide resistor that's ±2.9 % of the width. What a resistor costs in area recommended 0.35 µm, because area goes as W^2, and on a resistor that narrow the same 28.7 nm is ±8.2 %. That gives you your first real design trade-off: a narrow resistor is cheap in area, but it's also less accurate.

One circuit at three corners

`labs/passives-decks/spice/corners.spice.in` describes a single circuit. Its Makefile writes `corners_tt.spice`, `corners_ll.spice` and `corners_hh.spice` from it, so the three runs differ only in one word on the `.lib` line. Together they print:

```
--- tt : absolute (moves) ---
r_a = 3.550443e+03
r_b = 1.306704e+04
--- ll : absolute (moves) ---
r_a = 3.106637e+03
r_b = 1.143366e+04
--- hh : absolute (moves) ---
r_a = 3.994248e+03
r_b = 1.470042e+04
```

	ll	tt	hh
R_A (drawn 1 × 10 µm)	3106.637 Ω	3550.443 Ω	3994.248 Ω
relative to typical	**0.875000**	1.000000	**1.125000**

The resistor moves by exactly ∓12.5 %, because that's exactly what the corner file told it to do. The MIM capacitor in the same deck gives:

```
--- ll : MIM capacitance, farads ---   c_mim = 1.797322e-13
--- tt : MIM capacitance, farads ---   c_mim = 2.065822e-13
--- hh : MIM capacitance, farads ---   c_mim = 2.338667e-13
```

That's 179.7322 fF, 206.5822 fF and 233.8667 fF, or 0.87003× to 1.13208× the typical value. You'll run this deck in Lab 04, Part 1.

R and C errors together

The same deck also builds a low-pass filter from one resistor and one capacitor, and prints its corner frequency at each corner:

```
--- ll : RC corner frequency, Hz ---   f3db =  2.82935e+08
--- tt : RC corner frequency, Hz ---   f3db =  2.15410e+08
--- hh : RC corner frequency, Hz ---   f3db =  1.69052e+08
```

corner	$f_{-3\ \mathrm{dB}}$	relative
`ll`	282.935 MHz	1.3135
`tt`	215.410 MHz	1.0000
`hh`	169.052 MHz	0.7848

The filter's corner frequency spans 169 MHz to 283 MHz, a 1.67 : 1 range. The corner frequency is $f = 1/2\pi RC$, and R and C move together (the low corner is low R *and* low C, by construction), so their errors multiply instead of partly cancelling. You can check this by hand: $1.125 \times 1.13207 = 1.2736$, and $1/1.2736 = 0.7852$, against the simulated `hh` ratio of 0.7848. Two ±12 % errors, one in R and one in C,

multiply to shift the answer by −21.5 %, and the arithmetic agrees with the simulation to three decimal places.

This is why nobody builds a precision on-chip filter from a bare R and a bare C. If your specification says "corner at 200 MHz ±5 %", this topology fails it before you've drawn anything, and no amount of careful layout will fix that; you need a different topology.

Local mismatch

Everything above applies one number to the whole wafer. When you turn on the *local* mismatch models as well, even two resistors on the same die disagree with each other. `labs/passives-decks/spice/mismatch.spice` runs 200 Monte Carlo samples and prints:

```
Monte Carlo: 200 runs, seed 12345
ok       sigma/mean A (1x1)       2.7930 %   (reference  2.7930 %)
ok       sigma/mean B (1x1)       2.8775 %   (reference  2.8775 %)
ok       sigma/mean C (10x10)     0.3045 %   (reference  0.3045 %)
ok       sigma/mean D (10x10)     0.3479 %   (reference  0.3479 %)
```

(It also prints two ratio rows and an area line, which Matching and Pelgrom's law uses.)

A and B are drawn identically, 1 µm × 1 µm, in the same simulation on the same wafer, and yet each carries about 2.8 % of its own random variation on top of whatever the corner did. Lab 04: Two resistors that disagree is about that number.

A rule for specifications

> **Never write an absolute on-chip passive value into a specification you have to meet.** Assume ±20 % on any single resistance or capacitance and ±30 % on any product of the two. If the design doesn't survive that, it needs to stop depending on the absolute value.

If absolute values are this poor, you might wonder how anyone builds a 12-bit ADC, whose job is to be accurate to one part in four thousand. The answer is in the same three corner runs, and the next page shows you where.

Next: Matching and Pelgrom's law.

15. Matching and Pelgrom's law

A chip can't hold a resistor to better than ±12.5 %, and yet people build 12-bit converters on chips like that. They manage it by never asking the chip for a particular value, only for a **ratio** between two values.

This is the most important idea in analog IC design. The arithmetic behind it fits on one line, but it's rarely taught in a first circuits course, which is a shame, because it explains why most analog schematics look the way they do.

Ratios across corners

The corners deck from Process corners also prints a second set of numbers for each corner: two resistors drawn in a 1:4 ratio, the ratio of their values, and the output of a divider made from them:

```
--- ll : absolute (moves) ---
r_a = 3.106637e+03
r_b = 1.143366e+04
--- ll : ratio and divider (do not move) ---
ratio = 3.680396e+00
divider = 3.845828e-01
```

```
--- tt : absolute (moves) ---
r_a = 3.550443e+03
r_b = 1.306704e+04
--- tt : ratio and divider (do not move) ---
ratio = 3.680396e+00
divider = 3.845828e-01
```

```
--- hh : absolute (moves) ---
r_a = 3.994248e+03
r_b = 1.470042e+04
--- hh : ratio and divider (do not move) ---
ratio = 3.680396e+00
divider = 3.845828e-01
```

Here are the same numbers in one table:

	`ll`	`tt`	`hh`	spread
R_A	3106.637 Ω	3550.443 Ω	3994.248 Ω	**±12.5 %**
R_B	11 433.66 Ω	13 067.04 Ω	14 700.42 Ω	**±12.5 %**
R_B/R_A	3.680396	3.680396	3.680396	**0**
divider out	0.3845828 V	0.3845828 V	0.3845828 V	**0**

The ratio and the divider output agree to every printed digit. Between the `ll` and `hh` runs each resistor swung by a quarter of its value, yet the divider built from the two of them didn't move at all. You'll run this deck yourself in Lab 04, Part 1.

Why the ratio holds

A corner doesn't add a fixed offset to a resistor's value; it applies a **multiplier**, and every device made of that material gets the same factor k:

$$\frac{R_B}{R_A} = \frac{k\,R_\square\,L_B/W}{k\,R_\square\,L_A/W} = \frac{L_B}{L_A} \tag{15.1}$$

Both k and $R_\square$ cancel, and so does any error in the width, as long as you draw both resistors the same width. What's left is the ratio of the two lengths you drew, and lengths on a mask are held to within nanometres.

The same arithmetic explains why the printed ratio is 3.680396 rather than the 4.0 you drew. The *bodies* are in the ratio $L_B/L_A = 40/10 = 4$, but each device also carries 378.2448 Ω of end resistance (Ohms per square):

$$\frac{R_B}{R_A} = \frac{378.2448 + 317.2198 \times 40}{378.2448 + 317.2198 \times 10} = \frac{13067.04}{3550.443} = 3.680397 \tag{15.2}$$

ngspice printed `3.680396`, so the ends account for the whole difference from 4. The end resistance is a systematic error in the ratio, and changing the corner doesn't remove it. That's why real layouts build a 4:1 ratio from four identical unit resistors in series next to a single unit, rather than from one long strip beside one short one. The four units in series have four sets of ends and the single unit has one, so the ends are in the same 4:1 proportion as the bodies, and the ratio comes out 4.000000 at every corner.

That's the first rule of matched layout, and you can derive it from a printed number:

If you need a ratio, build both sides from repeated copies of the same unit device, never from one big device and one small one.

Mismatch between identical devices

A corner is global: it moves every device made of the same material by the same factor, which is why a ratio cancels it completely. Mismatch is local. Two devices drawn identically still come out slightly different from each other, and a ratio can't cancel that difference at all, so mismatch sets the floor on how well any ratio can be held.

`labs/passives-decks/spice/mismatch.spice` runs two hundred Monte Carlo samples with `mc_mm_switch=1`, which switches on the mismatch terms in the model card (they are multiplied by `MC_MM_SWITCH`, as you'll see under Pelgrom's law below), and uses `setseed 12345` so that the numbers are reproducible. (Lab 04 runs a similar experiment starting from `setseed 1` and gets 2.5852 % for the 1 µm² device. That's a different draw from the same distribution rather than a different answer.) The deck has four devices, all of the same material from the same foundry: **A** and **B** are drawn 1 µm × 1 µm, and **C** and **D** are drawn 10 µm × 10 µm, which is one hundred times the area.

```
== 5/5  mismatch: 200 wafers, four resistors ==================
   800 measurements recorded

  Monte Carlo: 200 runs, seed 12345
  ok       sigma/mean A (1x1)       2.7930 %   (reference  2.7930 %)
  ok       sigma/mean B (1x1)       2.8775 %   (reference  2.8775 %)
  ok       sigma/mean C (10x10)     0.3045 %   (reference  0.3045 %)
  ok       sigma/mean D (10x10)     0.3479 %   (reference  0.3479 %)
  ok       sigma/mean A/B ratio     3.7695 %   (reference  3.7695 %)
  ok       sigma/mean C/D ratio     0.4496 %   (reference  0.4496 %)
           100x the area bought 8.38x better matching   (Pelgrom's square root of 100
would be 10x)
```

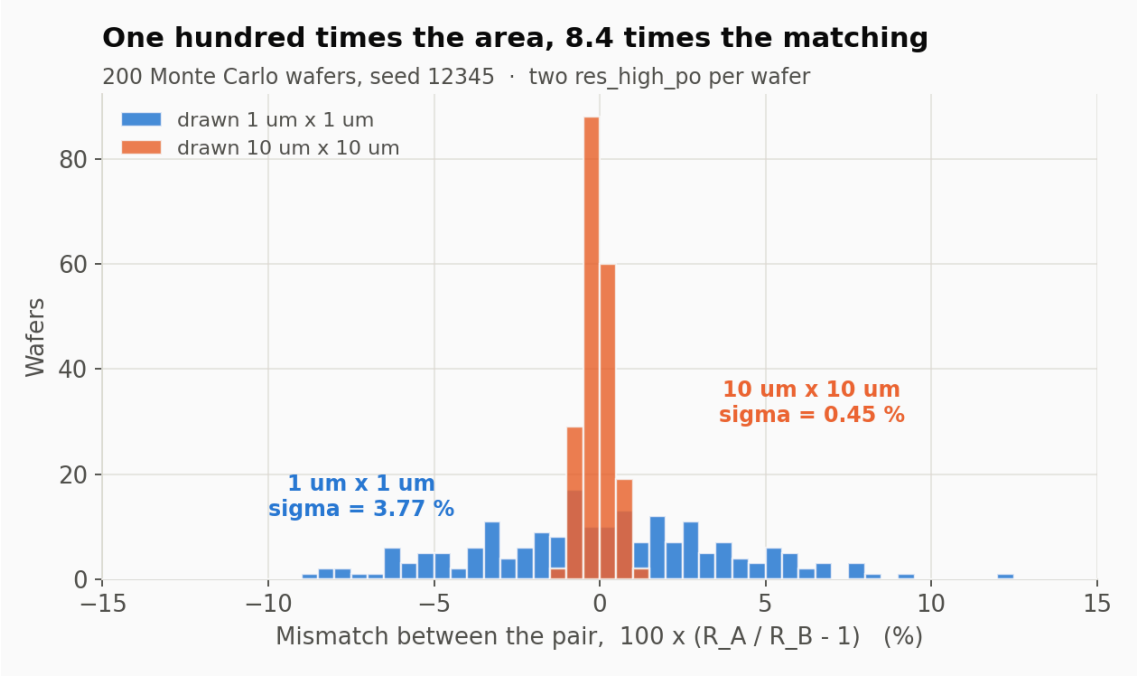


Figure 15.1: Mismatch between a drawn-identical pair, 1x1 vs 10x10

The plot shows two hundred wafers, each with two resistors drawn from the same rectangle. The blue histogram is the 1 µm × 1 µm pair, which routinely comes out 5 % apart and sometimes 12 %. The orange histogram is the 10 µm × 10 µm pair, and almost all of those are within one percent of each other.

Pelgrom's law

The rule behind that plot is **Pelgrom's law**: the mismatch between two identically drawn devices scales as one over the square root of their area.

$$\frac{\sigma_{\Delta R}}{R} = \frac{A_R}{\sqrt{WL}} \tag{15.3}$$

A_R is a constant for each process, and SkyWater prints it on the model card, in `sky130_fd_pr__res_high_po.model.spice`:

```
+ body_pelgrom = 0.03552
+ head_pelgrom = 0.0761
+ num_con_row = {max(floor(0.5+(w-0.33)/0.36),1)}
+ rbody_match = {body_pelgrom/sqrt(w*l*mult)*MC_MM_SWITCH*AGAUSS(0,1.0,1)}
+                          rend_match                                    =
{head_pelgrom/sqrt((w+0.525)*num_con_row*mult)*MC_MM_SWITCH*AGAUSS(0,1.0,1)}
```

So the body of the resistor matches as $0.03552/\sqrt{WL}$. The contact heads follow a different law, $0.0761/\sqrt{(W+0.525)\,n_{\text{contacts}}}$, because a contact head is a row of cuts rather than an area.

Worked example: predicting the Monte Carlo

You can predict what the Monte Carlo will report before you look at it. The head and body resistances of both devices come from running `make resistor` in `labs/passives-decks`:

```
head_l1 = 2.998915e+02    body_l1 = 3.955731e+02    r_l1 = 6.954646e+02
head_10x10 = 3.410354e+01 body_10x10 = 3.247625e+02 r_10x10 = 3.588661e+02
```

The 1 µm × 1 µm device. $n_{\text{contacts}} = \lfloor 0.5 + (1 - 0.33)/0.36 \rfloor = 2$.

$$\sigma_{\text{head}} = \frac{0.0761}{\sqrt{1.525 \times 2}} = 4.357\ \% \qquad \sigma_{\text{body}} = \frac{0.03552}{\sqrt{1}} = 3.552\ \% \tag{15.4}$$

$$\sigma_R = \sqrt{(299.8915 \times 0.04357)^2 + (395.5731 \times 0.03552)^2} = \sqrt{13.067^2 + 14.051^2} = 19.188\ \Omega \tag{15.5}$$

$$\frac{\sigma_R}{R} = \frac{19.188}{695.4646} = \boxed{2.759\ \%} \tag{15.6}\ (15.7)$$

Monte Carlo gives 2.7930 % for A and 2.8775 % for B. Two hundred samples pin σ down only to within about $1/\sqrt{2N} = 5$ % of its own value, so the predicted 2.759 and the measured 2.79 are the same number as far as this experiment can tell.

The 10 µm × 10 µm device. $n_{\text{contacts}} = \lfloor 0.5 + 9.67/0.36 \rfloor = 27$.

$$\sigma_{\text{head}} = \frac{0.0761}{\sqrt{10.525 \times 27}} = 0.4514\ \% \qquad \sigma_{\text{body}} = \frac{0.03552}{\sqrt{100}} = 0.3552\ \% \tag{15.8}$$

$$\sigma_R = \sqrt{(34.10354 \times 0.004514)^2 + (324.7625 \times 0.003552)^2} = 1.1638\ \Omega \tag{15.9}$$

$$\frac{\sigma_R}{R} = \frac{1.1638}{358.8661} = \boxed{0.3243\ \%} \tag{15.10}\ (15.11)$$

Monte Carlo gives 0.3045 % for C and 0.3479 % for D.

The predicted improvement is $2.759/0.3243 = 8.51\times$, and the measured one is 8.38×.

Neither of those is the 10× that Pelgrom's $\sqrt{100} = 10$ suggests, because that figure assumes the whole device is body. In the small resistor 43 % of the resistance is contact head, and the heads match as $\sqrt{(W + 0.525)\,n}$, which improved by only 9.7× between the two devices while the body's $\sqrt{WL}$ improved by 10×. That's where the shortfall from 10 to 8.4 comes from. If you had predicted 10×, you would have been wrong, but in a way the model can explain.

Four rules of analog layout

Everything above comes down to four practical rules:

1. **Build ratios from identical unit devices.** A 4:1 ratio is four units against one unit, never a long device against a short one. That removes the end-resistance error, and it also means both sides see the same etch.
2. **Buy matching with area.** For 0.1 % matching, $\sigma = A_R/\sqrt{WL}$ gives $WL \approx (0.03552/0.001)^2 = 1262$ µm² per device. Matching isn't free, and you pay for it in square micrometres.
3. **Surround the array with dummies.** The outermost real unit has a neighbour on one side and empty space on the other, and etch and stress respond to that difference. An extra unit connected to nothing makes every real unit's surroundings identical.
4. **Interdigitate, or use a common centroid.** The wafer has gradients across it, in temperature, oxide thickness and stress, so two devices placed side by side sit at different points on the gradient. Splitting each device into halves and laying them out `A B B A` puts both devices' centres at the same point, so a linear gradient cancels exactly.

Rules 3 and 4 are things you draw rather than things you calculate, so they're taught in AD104, but the reasons for them are all on this page.

How analog circuits use matching

Now go back to the converter at the top of the page. A 12-bit ADC needs an accuracy of 1 part in 4096 from a process that can't hold a resistor to better than 1 part in 8, and it gets there because none of its accuracy depends on an absolute value:

- a **capacitor DAC** is a binary-weighted array of identical unit capacitors, so its linearity is a ratio, and the unit is made large enough that Pelgrom's law gives the required σ;
- a **current mirror** copies a current by making two transistors identical, without needing to know what the current is;
- a **bandgap reference** produces a voltage from a resistor ratio times kT/q, which depends only on physical constants and the temperature, so the absolute sheet resistance drops out;

- a **differential pair**, the basic analog building block, only needs its two halves to be the same as each other, whatever their actual values turn out to be.

All four rely on the same idea. Silicon is very good at making two things the same and poor at making one thing a particular value, so analog circuits ask it for copies rather than numbers.

Next on the Course path: Lab 04: Two resistors that disagree. It runs its own Monte Carlo, checks Pelgrom's law across four device sizes, and ends with a divider specification that a naive two-resistor layout cannot meet. After that comes From schematic to cross-section, which turns the chip on its side and takes your measured 317.2198 Ω/□ back to a material property.

Stuck on anything here? Ask in the team Discord.

16. Lab 04: Two resistors that disagree

In this lab you'll watch the same resistor change value at every process corner while a ratio of two resistors stays fixed. Then you'll run a Monte Carlo mismatch simulation that anyone can reproduce, check Pelgrom's law against your own data and find where it stops working, and learn to tell **accuracy** (the nominal value is wrong) apart from **precision** (the spread is wide). The lab ends with a divider that you have to make accurate, precise and small at the same time.

Package: `labs/lab-04-two-resistors-that-disagree/`

Before you start

- Finish Lab 01; you'll need its 378.2448 Ω of end resistance. Lab 02 and Lab 03 help, but you don't need them.
- Read Process corners and Matching and Pelgrom's law.
- Open a terminal in the workbench and go to the lab folder:

```
cd /foss/designs/modules/ad102/labs/lab-04-two-resistors-that-disagree
```

This lab needs no environment setup.

Parts 1 to 6 are required. Everything under Going further, at the end, is optional.

Time: about 2 hours, not counting the optional part. The Monte Carlo runs print nothing while they work, so don't mistake the quiet for a hang. A full `make` takes between about 30 seconds and two minutes, depending on your machine (the Makefile records 106 s and 128 s on the pinned image; see Warnings and delays).

Labs 01–03 assumed that a device is worth what its geometry says: you sized a resistor to 2200 Ω and the simulator returned 2200.006 Ω. That settles the design, but the factory still has to build it. The strip you drew is 1 µm wide on the mask, and the finished silicon is 1 µm wide plus or minus whatever the lithography, etch, implant dose and anneal did that day, which is slightly different for every device on the die. So a resistor's value is a random variable, and so is every other device parameter on a chip. You can't design that out, because it comes from building a million things at once, and much of analog design is about arranging for the randomness not to matter.

Part 1: Process corners

What moves between process corners, and what stays put?

The corners deck in `labs/passives-decks` is the one the two guide pages quote. Its circuit has two resistors, R_A drawn 1 × 10 µm (`r_a`) and R_B drawn 1 × 40 µm (`r_b`), a divider built from the same two shapes, and an RC made of a 1 × 10 µm resistor and a 10 × 10 µm MIM capacitor. The deck builds that circuit three times, once per corner (`ll`, `tt`, `hh`), and the only thing it changes between them is one word on the `.lib` line. `ratio` is R_B/R_A, which is drawn as 4 to 1.

1. Go to the `passives-decks` package and run the three corners:

```
cd ../passives-decks
make corners
```

2. Read the three `absolute (moves)` blocks. Divide the `ll` and `hh` values of `r_a` by the `tt` value.
3. Read the three `RC corner frequency` lines. Before you compare them, work out what the `hh` corner should do to the corner frequency: R goes up by 1.125 and the MIM capacitance (the `MIM capacitance` lines) by 1.13207, so f should scale by $1/(1.125 \times 1.13207)$.
4. Read the three `ratio and divider (do not move)` blocks.

5. Go back to the lab folder:

```
cd ../lab-04-two-resistors-that-disagree
```

You should see: `r_a` at 3106.637, 3550.443 and 3994.248 Ω, so `ll` and `hh` are 0.875000 and 1.125000 of typical. The corner frequencies are 282.935, 215.410 and 169.052 MHz, and your prediction for `hh`, $1/1.2736 = 0.7852$, compares with a simulated 0.7848. In all three corners you should also see:

```
ratio = 3.680396e+00
divider = 3.845828e-01
```

The absolute values moved by ±12.5 % and the RC corner frequency over a 1.67 : 1 range, yet the ratio and the divider didn't change in any printed digit. Why not?

Answer: A1, in Answers at the end.

Part 2: Run the Monte Carlo

Can you run a Monte Carlo that someone else can reproduce exactly?

A corner shifts every device together, whereas mismatch moves devices apart from each other, so it needs a different switch. `spice/mismatch.spice` turns the SKY130 mismatch models on with one line:

```
.param mc_mm_switch=1
```

With it on, every device instance draws its own `AGAUSS()` sample when the netlist is evaluated. The control block repeats two steps two hundred times: it calls `mc_source`, which re-evaluates the netlist (in effect, it fabricates a different die), and then it runs one `op`. After that, `alterparam mc_mm_switch=0` followed by one more `mc_source` gives the nominal die, with no mismatch at all. `setseed 1` fixes the random sequence, so your two hundred dies are the same two hundred dies as on this page, in the same order.

1. Run the lab:

```
make
```

   It ends in `FAIL`, which you'll fix in Part 6. Leave the output in your terminal.

2. Before you trust any statistic, check the count at the top of the output. If it says 2 instead of 201, the loop stopped after one die, and Warnings and delays explains why.

You should see: the count, then the nominal die. Section A of the deck forces 1 µA through four single resistors, so dividing `v(s1)` to `v(s4)` by 1 µA gives their nominal resistances: 695.4646, 1647.124, 1468.119 and 1370.854 Ω. `v(ta)` to `v(td)` are the divider taps you'll read in Parts 4 and 5.

```
operating points (200 dies + 1 nominal) : 201
--- NOMINAL (mismatch off) ---
v(s1) = 6.954646e-04
v(s2) = 1.647124e-03
v(s3) = 1.468119e-03
v(s4) = 1.370854e-03
v(ta) = 9.000000e-01
v(tb) = 9.000000e-01
v(tc) = 1.309327e+00
v(td) = 1.350000e+00
```

A Monte Carlo result that nobody can reproduce is an anecdote rather than a measurement. Seeding the run costs one word, and it turns "about 5.6 mV" (a spread you'll meet in Part 5) into a number you can put in a table and have someone else check.

Part 3: Pelgrom's law

Does σ halve when the area quadruples?

Section A of the deck puts 1 µA through four single resistors with areas of 1, 4, 16 and 64 µm².

1. Read the mismatch coefficients off the model card:

```
grep               pelgrom               /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__res_high_po.model.spice
```

```
+ body_pelgrom = 0.03552
+ head_pelgrom = 0.0761
+ rbody_match = {body_pelgrom/sqrt(w*l*mult)*MC_MM_SWITCH*AGAUSS(0,1.0,1)}
+                                  rend_match                                  =
{head_pelgrom/sqrt((w+0.525)*num_con_row*mult)*MC_MM_SWITCH*AGAUSS(0,1.0,1)}
+ res_match = {(body_pelgrom/sqrt(w*l*mult))*MC_MM_SWITCH*AGAUSS(0,1.0,1)}
```

The first two lines are the coefficients, and the last three are the model that uses them. There are two coefficients because a resistor has two parts that mismatch independently of each other: the body and the contact head.

2. Predict all four spreads from `body_pelgrom / sqrt(W*L)`.
3. Find table A in the checker's output (the checker's tables use the deck's section letters).

You should see:

```
A. one resistor, four sizes -- does sigma halve when area quadruples?
  geometry          area    sigma/R   0.03552/sqrt(A)   ratio to previous
  W=1  L=1             1     2.5852%           3.5520%                  --
  W=1  L=4             4     1.6046%           1.7760%               1.611
  W=2  L=8            16     0.8391%           0.8880%               1.912
  W=4  L=16           64     0.4007%           0.4440%               2.094
```

The last column tests the law, and for the three larger devices it holds: 1.912 and 2.094 against a predicted 2.000, from 200 samples that can only pin σ down to ±5 % anyway.

The 1 µm² device doesn't follow the law. It should have scattered by 3.55 %, but it scattered by 2.59 %, which is 27 % less than predicted, so it does *better* than the law allows. Why? (Look back at Lab 01, Part 6.)

Answer: A2, in Answers at the end.

Part 4: The same divider at 48 times the area

What does 48 times the area buy you in precision?

Section B builds a 1:1 divider twice, once from two 0.84 µm² devices and once from two 40 µm² ones. The schematic and the ratio are the same in both, and only the size changes.

1. From area alone, predict how much more precise the big pair should be: $\sqrt{40/0.84}$.
2. Find the two `1:1` lines in the checker's `B and C` table.

You should see:

```
    1:1  two devices W=0.42 L=2    0.84 um^2 each   0.900000 V  25.0787 mV
    1:1  two devices W=2 L=20        40 um^2 each   0.900000 V   3.2321 mV
```

Both nominals are exactly 0.900000 V, so both dividers are perfectly accurate, but one is eight times less precise than the other. 25 mV of scatter on a 900 mV node is 2.8 %, which is roughly five bits' worth of reference error in an ADC, and it comes from a circuit whose schematic is correct. The fix is the least clever thing in engineering: draw it bigger.

47.6 times the area bought 7.76 times the precision, but area alone predicts $\sqrt{47.6} = 6.90$. Why is the small pair about 12 % worse than its area explains?

Answer: A3, in Answers at the end.

Precision is paid for in square micrometres, and that is the main reason analog blocks look enormous next to digital ones.

Part 5: One ratio drawn two ways

Does building a ratio from identical units help its accuracy, its precision, or both?

Section C builds a 3:1 divider twice, each time from exactly 24 µm² of poly, with the tap meant to sit at three quarters of the 1.8 V supply:

- `tc`: one device of $L = 6$ from `vdd` to the tap and one of $L = 18$ from the tap to ground. This is the way most people would draw it.
- `td`: four identical $L = 6$ units in series, tapped after the first one down from `vdd`.

On paper the two have the same ratio, and they use the same area and the same material.

1. Predict the nominal and the spread for each, and decide which will be better and why.
2. Find the two `3:1` lines and the summary below them.

You should see:

```
3:1  one L=6 and one L=18        24 um^2 total  1.309327 V   5.6433 mV
3:1  four identical L=6 units    24 um^2 total  1.350000 V   5.7290 mV

the 3:1 ratio you asked for is 1.350000 V.
  two devices  : 1.309327 V  (-3.01 %)
  four units   : 1.350000 V  (+0.00 %)
and the spread is 5.6433 mV against 5.7290 mV -- a 1.5 % difference,
where 200 samples only pin sigma down to +/- 5.0 %. Those are the same number.
```

The nominals differ by 3.01 %, and the two-device version is wrong in the same direction, by the same amount, on every die. Why?

Answer: A4, in Answers at the end.

The spreads, 5.6433 against 5.7290 mV, are 1.5 % apart. Did the unit cells buy any precision?

Answer: A5, in Answers at the end.

So "matching" usually means two different things, and you pay for each of them differently:

	what fixes it	what it costs
accuracy: the nominal is wrong	identical unit devices	nothing
precision: the spread is wide	area	area, at 4× per factor of 2

Build every ratio you care about from identical unit cells. It costs nothing, and it removes a whole class of error, including errors that nobody has told you about yet.

What this simulation cannot show. Every analog textbook tells you to lay matched devices out as interleaved unit cells in a common centroid, and yet you have just measured that unit cells do nothing for σ. Both are true, because this simulation contains only random, per-device mismatch. Real wafers also have **gradients**: oxide thickness, implant dose and temperature vary smoothly across a die, so two devices ten micrometres apart are more alike than two a millimetre apart, and the left member of a pair sits in a systematically different environment from the right one. Interleaved unit cells in a common centroid cancel the linear part of that gradient. It's a large, real effect, and it's the main reason the technique exists.

`AGAUSS` knows nothing about where your devices are, because your netlist doesn't say where they are. Position enters a design at layout, and a SPICE netlist has no coordinates. If you build the interleaved version and run ten thousand dies, you will measure zero benefit, because what you're measuring is the model rather than the silicon. Before you use a simulation to check a design decision, check that the simulator models the effect the decision is about.

Post-layout extraction, which AD104 covers, doesn't fix this either, because it adds parasitic R and C from your actual geometry and says nothing about process gradients. Gradient matching is checked with test chips and layout review instead, and this is one of the real limits of simulation.

Part 6: Design a 0.45 V divider

Can you make one divider accurate, precise and small at the same time?

`spice/my_divider.spice` has to put 0.4500 V on `tap` from a 1.8 V supply, using `sky130_fd_pr__res_high_po`, and meet all three of these:

1. nominal within 0.2 % of 0.4500 V,

2. σ at most 3.0 mV over 200 dies,
3. total drawn area under 150 µm².

You can use as many devices as you like. Keep the node called `tap` and the supply called `vdd`, because the checker reads your geometry back out of the file.

The file ships with the divider everybody draws first: one resistor on top three times as long as the one underneath (`L=18` and `L=6`, both `W=1`). That is Part 5's `tc` turned upside down, but it runs in a separate deck with its own 200 dies, which is why its σ differs from `tc`'s. When you ran `make` in Part 2, it failed two of the three checks:

```
1. nominal tap   0.490673 V    target 0.450000 V ( +9.04 %)      FAIL
2. sigma           5.9285 mV   budget 3.0000 mV                    FAIL
3. drawn area       24.00 um^2 budget    150 um^2                  PASS
```

1. Decide how to fix the nominal (Part 5) and how to fix σ (Parts 3 and 4).
2. Edit `spice/my_divider.spice`. Write `W` and `L` as plain microns, with no `u` suffix.
3. Re-run your divider and the checker:

```
make mine check
```

You should see: all three checks marked `PASS`, then

```
PASS -- every number matches the reference run.
```

Finishing

You're done when `make mine check`, run on the divider you designed in Part 6, prints `PASS -- every number matches the reference run.`

`setseed 1` makes the Monte Carlo reproducible, so on the pinned image (ngspice-47, `sky130A`, `tt` corner, seed 1, 200 dies) your numbers match the ones on this page.

Question: the shipped divider failed two checks. Which failure did area fix, and which did identical units fix?

Answer: A6, in Answers at the end.

After yours passes, read `solutions/README.md` for one good answer and why unit devices do nothing for σ in this simulator.

Submit: nothing for this lab.

Going further (optional)

This part isn't in the time estimate, and nothing later in the course needs it. What to do next on the Course path is on the last line of this page.

The guide page's mismatch run

The comparison of 1 × 1 µm against 10 × 10 µm on Matching and Pelgrom's law uses a different seed, and its deck is in `labs/passives-decks`. To reproduce it:

```
cd ../passives-decks
make mismatch
cd ../lab-04-two-resistors-that-disagree
```

It ends with `100x the area bought 8.38x better matching`.

Next on the Course path: read From schematic to cross-section, then do the Capstone: Defend a floor plan.

Answers

A1. Part 1: Process corners

A corner applies the same multiplier to every device made of that material, and in a ratio of two such devices the multiplier cancels, which leaves a ratio of the lengths you drew. Matching and Pelgrom's law works it through, including why the ratio is 3.680396 rather than 4.

A2. Part 3: Pelgrom's law

At $W = 1$, $L = 1$ the resistor's total is 695.46 Ω, and 299.89 Ω of that, or 43 %, is contact head rather than body. (The rest of Lab 01′s 378.2448 Ω of ends is the body's extra 78 Ω, as Lab 01, Part 6 found.) `body_pelgrom` describes only the body, so the 3.55 % applies only to the body's share of the resistance. The head has its own coefficient (`head_pelgrom`, averaged over a different quantity), and on this device the head contributes about as much to σ as the body does. The worked example gets 13.067 Ω from the head and 14.051 Ω from the body, which combine to a σ of 19.188 Ω, or 2.759 % of 695.4646 Ω. That is much closer to what you measured than 3.55 % is.

So a mismatch coefficient describes one *parameter* of a device rather than the whole device. Once your geometry stops looking like the geometry the coefficient was extracted on, the single-number version of the law no longer applies. It's the same trap as Lab 01′s naive $R = R_\square L/W$, and it has the same fix: measure the thing you are actually going to build.

A3. Part 4: The same divider at 48 times the area

Each small device is 0.42 µm wide, which is below the 0.69 µm point where Lab 01, Part 4 found that the sheet resistance starts to change. It also pays head mismatch, which doesn't scale with WL at all.

A4. Part 5: One ratio drawn two ways

Lab 01 found that every resistor pays 378.2448 Ω of end resistance whatever its length, so

$$\frac{R_6}{R_6 + R_{18}} = \frac{378.24 + 6(317.22)}{8369.76} = 0.27260 \tag{16.1}$$

of the supply drops across the top device instead of 0.25, and the tap sits at 1.309327 V instead of 1.350000 V. If you build both branches from the *same unit*, that overhead appears equally in both, so the ratio becomes a ratio of **counts** (one unit out of four) and the overhead divides out exactly. The measured nominal is `1.350000`.

A5. Part 5: One ratio drawn two ways

No. 200 samples can't resolve σ better than $1/\sqrt{2N} = 5$ %, so the two spreads are the same number. Splitting the resistor into unit cells bought nothing in random mismatch, and the algebra says it never will: when the errors are independent from device to device, the σ of a ratio depends on the total area and on nothing else about how you divided it up.

A6. Finishing

Identical units fixed the nominal, which was a systematic error from the end resistance (Part 5). Area fixed σ, at a cost of four times the area for each factor of two (Parts 3 and 4). The lab's `solutions/README.md` has one design that does both.

V

9–10. Capstone

17 From schematic to cross-section . . . 95
17 The layer stack . . . 95
17 From sheet resistance to resistivity . . . 96
17 The three questions AD102 answers . . . 96
17 Where you go next . . . 97

18 Capstone: Defend a floor plan . . . 99
18 Before you start . . . 99
18 Part 1: Read the layer stack . . . 99
18 Part 2: Look at cross-sections in SiliWiz . . . 99
18 Part 3: Defend a floor plan . . . 100
18 Finishing . . . 101
18 Answers . . . 101

17. From schematic to cross-section

So far in AD102 you have drawn rectangles and called them components. This page shows what those rectangles look like when you slice through the chip.

Everything up to now has been in **plan view**: you drew a shape on a mask layer and read a number off a simulator. That's how layout is done, and it's enough to design with. A chip, though, is a stack of films a few micrometres tall, and every surprise in this course came from that third dimension:

- The resistor had **end resistance** (378.2448 Ω of it) because current has to climb *down* from metal into poly and back up again.
- The capacitor had **fringe** capacitance because field lines leave the *edge* of a plate.
- The wire had a parasitic to the substrate because it is a bar suspended *over* a plane.
- The inductor was impractical because its magnetic field goes *into* the silicon underneath it.

Each of those is a fact about the side view, which is what this page is about.

The layer stack

Ohms per square started from

$$R = \rho \frac{L}{Wt} = \left(\frac{\rho}{t}\right)\frac{L}{W} = R_{\square}\frac{L}{W} \tag{17.1}$$

and treated the film thickness t as a number that the foundry fixes and that you never need to know. The foundry does fix it, but it also writes it down, in the technology file Magic uses to draw cross-sections, and that file has been on your disk since IC101. Its `height` lines give, for each layer, how far the bottom of the layer sits above the wafer surface and how thick the layer is, both in µm. Here are the lines for the main layers (the file also lists fill, contact, via and MIM layers):

```
height dnwell      -0.1    0.1
height nwell,pwell  0.0    0.2062
height alldiff      0.2062 0.12
height allpoly      0.3262 0.18
height allli        0.9361 0.10
height allm1        1.3761 0.36
height allm2        2.0061 0.36
height allm3        2.7861 0.845
height allm4        4.0211 0.845
height allm5        5.3711 1.26
height mrdl         7.2611 3.0
```

Those eleven lines describe the main stack of the chip. It helps to read them like the floors of a building, each with the height it starts at and how tall it is:

Layer	Bottom	Thickness
deep n-well	−0.1 µm	0.1 µm
n-well / p-well	0.0 µm	0.2062 µm
diffusion	0.2062 µm	0.12 µm
poly	**0.3262 µm**	**0.18 µm**
local interconnect (li)	0.9361 µm	0.10 µm
metal1	1.3761 µm	0.36 µm
metal2	2.0061 µm	0.36 µm

Layer	Bottom	Thickness
metal3	2.7861 µm	0.845 µm
metal4	4.0211 µm	0.845 µm
metal5	5.3711 µm	1.26 µm
redistribution metal	7.2611 µm	3.0 µm

The top of metal5 is 6.6311 µm above the silicon, so the wiring stack up to metal5 is about as thick as a red blood cell is wide. (The redistribution metal above it reaches 10.2611 µm, and the substrate underneath isn't counted at all.) Your 100 kΩ resistor was 314.046460 µm long, which is forty-seven times the height of that stack. You'll print this stack yourself in the capstone, Part 1.

From sheet resistance to resistivity

In Lab 01 you measured the sheet resistance of high-sheet poly yourself, 317.2198 Ω/□, and the tech file says that poly is 0.18 µm thick, so its resistivity is

$$\rho = R_\square \, t = 317.2198 \times 0.18 \times 10^{-6} = 5.7100 \times 10^{-5}\ \Omega\,\mathrm{m} = 5.7100\ \mathrm{m}\ \Omega\,\mathrm{cm} \tag{17.2}$$

A sheet resistance depends on how thick the film is, but this number doesn't: it's a **material property**, the resistivity of the p+ doped polysilicon that SkyWater deposits, and you got it from a resistance you measured and a number you read out of a file.

Do the same for `res_xhigh_po`, which measured 2118.7619 Ω/□ on the same 0.18 µm film:

$$\rho = 3.8138 \times 10^{-4}\ \Omega\,\mathrm{m} = 38.1377\ \mathrm{m}\ \Omega\,\mathrm{cm} \tag{17.3}$$

That's the same layer at the same thickness, with 6.7 times the resistivity. Resistors on a chip said that `res_high_po` and `res_xhigh_po` differ only in implant dose, and the factor of 6.7 is what that difference in dose does to the material: less dopant means fewer carriers, and fewer carriers mean a higher resistivity.

To give those numbers a scale, here they are next to two metals:

Material	ρ	vs copper
copper	1.68×10^{-8} Ω m	1×
SKY130 metal1 (0.125 Ω/□ × 0.36 µm)	4.50×10^{-8} Ω m	**2.7×**
`res_high_po`	5.71×10^{-5} Ω m	**3 399×**
`res_xhigh_po`	3.81×10^{-4} Ω m	**22 701×**

This is why on-chip resistors are made of poly rather than metal. A resistor's job is to resist, and the wiring metal conducts more than a thousand times too well for that. To get 10 kΩ from metal1 you would need $10000/0.125 = 80\,000$ squares, which at the 1 µm width of the course's poly resistors is 80 millimetres of wire on a die only three millimetres across.

It's worth making a habit of this conversion. Whenever a table gives you Ω/□, you're one multiplication away from a material property, and material properties tell you whether a number is plausible. A resistivity of 5.7 mΩ·cm is normal for a doped semiconductor, so if your arithmetic had produced 5.7 Ω·cm or 5.7 µΩ·cm, you would know something had gone wrong.

In the capstone, Part 2 you draw a resistor, a capacitor and a wire in SiliWiz and look at the cross-section of each one, and Part 3 then has you size a filter and defend its area.

The three questions AD102 answers

AD102 set out to answer three questions:

- **How do you fabricate a resistor at the micro scale?** You make it from a strip of doped poly, 0.18 µm thick, with $\rho = 5.71$ mΩ·cm. Its value is $R_\square L/W$ plus a fixed end resistance that you can't avoid, so you buy ohms with length.

- **A capacitor?** You make it from two plates and a gap: metal3 with a thin dielectric above it for a MIM, a gate oxide for a MOS capacitor, or interleaved wire fingers for a VPP. A linear capacitor gives you at best about 2 fF per μm², so a picofarad needs a plate 22 μm on a side.
- **An inductor?** Mostly, you don't make one. A metal spiral gets you single-digit nanohenries at a Q in the teens and costs tens of thousands of square micrometres, and below about 1 GHz the alternatives do the job better.

All three answers rest on one idea: **a component value is a geometry, and geometry costs area.** The habit that follows from it is to ask, of any analog schematic anyone shows you, what shape each part is and what that shape costs.

Where you go next

AD103 — Nonlinear Circuits is the next course. It asks the same question (what is this component, physically?) about devices whose behaviour is no longer linear. On Capacitors on a chip you saw the value of the MOS capacitor move by 3.57× with voltage. That was a transistor turning on, and AD103 opens by explaining what you were looking at. AD103 is also where XSchem becomes your main tool, and Lab 03, Part 8 was your introduction to it.

AD104 — Layout is where the cross-section turns into geometry you draw yourself. You draw these devices in Magic, on the real stack in the table above, and then prove two things about them: with DRC, that a fab could make them, and with LVS, that what you drew is the circuit you meant. Two things from this course come back there. The **unit-device** and **common-centroid** rules from Matching and Pelgrom's law become rectangles that you draw, and **parasitic extraction** reads your finished layout and hands back a netlist that includes every capacitor described on Parasitic capacitance.

Next on the Course path: Capstone: Defend a floor plan.

18. Capstone: Defend a floor plan

You'll read the SKY130 layer stack out of the PDK, look at a resistor, a capacitor and a wire in cross-section, and then size a 1 MHz on-chip low-pass filter and defend its area. There's no lab package for this capstone: Part 1 is a single command, Part 2 happens in a browser, and Part 3 is a page of arithmetic with numbers you've already measured.

Before you start

- Finish Labs 01, 02 and 03 first, because Part 3 uses the numbers you measured in them. The corner results from Lab 04 help with the last step of Part 3, where you decide whether you would build the filter at all.
- Read From schematic to cross-section.
- You need a terminal in the workbench for Part 1, a browser tab for Part 2, and paper or a spreadsheet for Part 3.

Time: about 1.5 hours.

Part 1: Read the layer stack

How tall is the SKY130 wiring stack, and what is the resistivity of the poly you used in Lab 01?

1. Print the `height` lines of Magic's technology file for the main layers:

   ```
   sed -n '/^ height/,/^$/p' /foss/pdks/sky130A/libs.tech/magic/sky130A.tech | grep -
   wE 'dnwell|nwell,pwell|alldiff|allpoly|allli|allm1|allm2|allm3|allm4|allm5|mrdl'
   ```

   The first number on each line is how far the bottom of the layer sits above the wafer surface, and the second is its thickness, both in µm. (Without the `grep`, the block also lists fill, contact, via and MIM layers: 30 lines in all.)

2. Add metal5's bottom height and its thickness to find how high the top of the wiring stack sits.
3. Multiply your Lab 01 sheet resistance for `res_high_po` (317.2198 Ω/□) by the poly thickness to get its resistivity.

You should see: eleven `height` lines, from `dnwell` at −0.1 µm to `mrdl` at 7.2611 µm, with `allpoly` at `0.3262 0.18` and `allm5` at `5.3711 1.26`. The top of metal5 is 6.6311 µm above the silicon, and $\rho = 317.2198 \times 0.18 \times 10^{-6} = 5.7100 \times 10^{-5}\ \Omega\,\mathrm{m}$.

Part 2: Look at cross-sections in SiliWiz

Where in the silicon do Lab 01's end resistance, Lab 02's edge term and the capacitor you never drew come from?

A list of heights from a file doesn't give you much feel for what the layers look like, so in this part you'll use SiliWiz, which shows a live cross-section under whatever layout you draw. Open **https://app.siliwiz.com/?preset=blank**.

> **Warning:** SiliWiz doesn't model SKY130. Its layer stack, permittivities and numbers are simplified for teaching, and its cross-section is a cartoon of the table on From schematic to cross-section rather than a rendering of it. Use it for the shape of the answer ("the current turns two corners", "the field bends around the edge", "there is a plate under everything"), and take the values from the tech file and your ngspice runs. There are guided walkthroughs for resistors · capacitors · parasitics.

The resistor you designed

1. Draw a **polyres** rectangle, long and thin: 20 grid units by 2, roughly the shape of your Lab 01 resistor.
2. Put a **metal1 via** (the contact) at each end and a **metal1** pad over each via.
3. Label the two pads `a` and `b` (click a pad, then **Set Label** or press `S`), and tick **Show SPICE** at the bottom. The resistance appears on the line that starts with `R0`.
4. Look at the **cross-section** pane and follow the current path: down the metal1 pad, through the contact, along the poly strip, up the far contact, and out the far pad.
5. Drag the strip longer and watch what the resistance does.

You should see the resistance climb linearly as the strip gets longer, while the two ends stay as they were. Where, in that current path, is the 378.2448 Ω of end resistance you accounted for in Lab 01?
Answer: A1, in Answers at the end.

The capacitor you designed

1. Start a new drawing. Put down a **metal1** rectangle, and a **metal2** rectangle the same size directly over it.
2. Label them and tick **Show SPICE**.
3. Find the capacitance between them, then drag the top plate so it overhangs the bottom one.

As you push the plates out of alignment, does the capacitance fall in proportion to the overlap area?
Answer: A2, in Answers at the end.

The parasitic you did not draw

1. Start a new drawing. Draw a single **metal1** wire and nothing else, and label it.
2. Read the SPICE output.

You should see a capacitor from the wire to the substrate in the SPICE output, even though you didn't draw one. Why is it there?
Answer: A3, in Answers at the end.

Part 3: Defend a floor plan

What is the smallest 1 MHz low-pass filter you can actually build on this chip, and would you build it?

Here is the brief. Build a first-order low-pass filter with a corner at 1 MHz, on-die, in SKY130. You may use `res_high_po`, `res_xhigh_po` and `cap_mim_m3_1`, and you must state the drawn area and defend it.

Work through it in this order:

1. Fix the product. $RC = 1/(2\pi f_c) = 159.15$ ns is your only hard constraint, and it fixes only the product, so you're free to split it between R and C however you like.
2. Price both halves. From Lab 01, `res_high_po` at $W = 1$ µm needs $L = (R - 378.2448)/317.2198$ µm of length, and `res_xhigh_po` needs $L = (R - 34.2111)/2118.7619$ µm. Because the strip is 1 µm wide, the length in µm is also its area in µm². From Lab 02, a MIM plate of side s holds $2.000000\,s^2 + 0.659991\,s - 0.017742$ fF and takes up s^2 µm². Tabulate the total area of the resistor and capacitor together for R = 10 kΩ, 100 kΩ, 1 MΩ and 10 MΩ.
3. Find the minimum, then find the limit. The total area has a minimum somewhere in your table, the same shape you found in Lab 03, Part 6. Then check your answer against What a resistor costs in area, where 1 MΩ of `res_high_po` comes to 3.15 mm of poly. Which of your rows could you actually build on a 3 mm die?
4. Check the resistor against itself. In Lab 03, Part 5, a bare 471.96 µm strip of 1 µm poly carried 95.274 fF to the substrate, and half of that was added to the time constant. Lab 03, Part 2 fitted that capacitance as 0.1994 fF per µm of length plus 1.187 fF for the ends. At what resistor length does the parasitic become a visible fraction of the capacitor you were planning? That length is the limit, and it's the reason the answer isn't simply "use an enormous resistor".
5. Price it in adders. Convert your best area into DD103 ripple-carry adders, at 110.1056 µm² each, and write out the sentence "my one filter pole costs N adders."
6. Decide whether you would build it, or whether you would rather do what What to build instead of an inductor describes and move the pole off-chip or into the digital domain.

You should see: an area table with a minimum in it, at least one row you have ruled out as unbuildable, and a single number of adders for your chosen design.

Finishing

You're done when you have the area table from step 2 with its minimum marked and at least one row ruled out (with the reason), the sentence from step 5 with your number in it, and an answer to step 6 you can defend.

There is no package and no marking scheme for this capstone.

Submit: nothing. If you'd like to argue about a floor plan, bring your area table to the team Discord.

Next on the Course path: this capstone is its last row, so you have finished AD102. The home page lists the next courses.

Answers

A1. Part 2: Look at cross-sections in SiliWiz

The current doesn't travel in a straight line through one material: it makes two right-angle turns through a contact much smaller than the strip. That detour is the end resistance, and in the cross-section you can point at it. That's why $R = R_{\square} L/W$ has a constant added to it, and why the constant doesn't scale with length.

A2. Part 2: Look at cross-sections in SiliWiz

No. It falls more slowly than the overlap does, because the field still bends around the edges of the plates. That edge field is the perimeter term, which is the middle coefficient of the $C = 2.000000\,s^2 + 0.659991\,s - 0.017742$ fF fit from Lab 02, Part 2.

A3. Part 2: Look at cross-sections in SiliWiz

The cross-section shows why: the wire is a bar, the substrate is a plane underneath it, and the field from the wire has nowhere to go but down into that plane. The guide page Parasitic capacitance is about exactly this.

Appendices

A. The SKY130 passive catalogue

This page lists the resistors and capacitors SKY130 gives you, what each one costs in area, and where each number comes from. It's meant to stay open beside your tools while you design.

We measured every resistance and capacitance below in `hpretl/iic-osic-tools:2026.08` with ngspice 47: we instantiated the device at a stated geometry, simulated it and divided, rather than copying the value from a datasheet. Every parameter is quoted along with the file it lives in, so you can check any line on this page with one `grep`.

Units of W and L

`W` and `L` on a SKY130 device are **plain numbers in microns, with no unit suffix**. `W=1` is one micrometre, and `W=1u` asks for a millionth of that. On a resistor the run doesn't stop: a strip written `W=1u L=10u` comes out at 3193.812 Ω where `W=1 L=10` gives 3550.443 Ω, with no error, and ngspice exits with status 0. The same rule applies to a capacitor's `W` and `L` and to a diode's `area` and `perim`. Getting started shows what happens on a resistor, and AD103's Reading a SKY130 device model has the exact error text for transistors.

Resistors

Resistor table

The table has thirteen devices in nine rows, and their sheet resistances span five orders of magnitude. `R measured` is the resistance of one square (a strip drawn W = 1 µm, L = 1 µm), measured at 0.1 V:

Device	What it is physically	$R_\square$ from the PDK	R measured, W=1 L=1
`res_xhigh_po`	poly with the ultra-high-resistance implant	2000 Ω/□	2152.022 Ω
`res_high_po`	poly with the high-resistance implant	317.3885 Ω/□	695.4646 Ω
`res_generic_pd`	p+ diffusion	197 Ω/□	195.4779 Ω
`res_generic_nd`	n+ diffusion	120 Ω/□	117.4914 Ω
`res_generic_po`	ordinary silicided poly, the same material as the gate	48.2 Ω/□	50.92314 Ω
`res_generic_nw`	the n-well itself	1700 Ω/□	1692.556 Ω
`res_iso_pw`	isolated p-well	3816 Ω/□	14969.38 Ω at W=2.65 L=10
`res_generic_l1`	local interconnect	12.2 Ω/□	11.97427 Ω
`res_generic_m1` … `m5`	metal 1–5	0.125 / 0.125 / 0.047 / 0.047 / 0.0285 Ω/□	0.1288327 Ω (m1), 0.04770963 Ω (m3)

The first five devices and `res_iso_pw` are subcircuits, so you instantiate them on `X` lines with `W=` and `L=` in capitals. `res_generic_nw`, `res_generic_l1` and the metal resistors are bare `.model` cards, so they go on `R` lines with lower-case `w=` and `l=`:

```
XR1 a 0 0 sky130_fd_pr__res_high_po W=1 L=10 mult=1
r1  b 0   sky130_fd_pr__res_generic_m1 w=1 l=1
```

If you get that backwards, ngspice says `Error: unknown subckt:` or `warning, can't find model`. The sheet resistances in the table come from these files in the image:

```
/foss/pdks/sky130A/libs.tech/ngspice/r+c/res_typical__cap_typical.spice
        rp1=48.2  rdn=120  rdp=197  rnw=1700  rl1=12.2
        rm1=0.125  rm2=0.125  rm3=0.047  rm4=0.047  rm5=0.0285
        rspwres=3816
/foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/sky130_fd_pr__res_high_po.model.spice
        rsheet = 317.3885     rhead_ps = 345.8312
/foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__res_xhigh_po.model.spice
        rsheet = 2000.0
```

Magic's extraction technology file, `/foss/pdks/sky130A/libs.tech/magic/sky130A.tech`, keeps its own copy of these values in milliohms per square (`xhrpoly 319800`, `uhrpoly 2000000`, `mrp1 48200`), and it doesn't agree with the ngspice file everywhere. Ohms per square explains which one to use when.

End resistance of `res_high_po`

A real resistor has to be contacted, so the device is a strip of resistive poly with a wide head at each end that holds a row of contacts, and those heads add resistance of their own. If you increase the length by 10 µm, the difference tells you what the *body* alone costs:

Device	R at L = 1	R at L = 11	slope (Ω per µm of L)
`res_high_po`, W=1	695.4646 Ω	3867.662 Ω	**317.2198**
`res_xhigh_po`, W=1	2152.022 Ω	—	—

That's 317.2198 Ω per micron of length at W = 1 µm, against `rsheet = 317.3885` on the model card. The two are 0.05 % apart, so the body really is the sheet resistance the card promises. The remaining $695.4646 - 317.2198 = 378.24$ Ω is end resistance: the contact head (299.8915 Ω, see Ohms per square) plus a short body extension that the model adds. The end resistance stays the same when you shorten the resistor, so in a one-square `res_high_po` the ends contribute more than the body does.

The design rule that follows is **never draw a short precision resistor.** The ends are 378 Ω whatever you do, so they make up 54 % of the value at one square, 19 % at five squares and 11 % at ten squares. Below about five squares, a large fraction of your resistor is contact resistance, and contact resistance matches far worse than sheet resistance does.

The fixed-width precision family

`res_high_po` and `res_xhigh_po` also ship in five pre-characterised widths each. These are the versions a layout tool will give you, because their contact head geometry is fixed:

Width (µm)	`res_high_po_*` Ω/µm of length	`res_xhigh_po_*` Ω/µm of length
0.35	1085.483	5713.715
0.69	472.111	2898.261
1.41	219.662	1418.298
2.85	105.534	701.684
5.73	56.460	349.005

Each figure is `(R at L=11 − R at L=1) / 10` from a measured pair of resistors, so the contact heads have already been subtracted out.

If you multiply each figure in the `xhigh` column by its width, you get 1999.80 Ω/□ for all five ($0.35 \times 5713.715 = 1999.80$, $5.73 \times 349.005 = 1999.80$), against `rsheet = 2000.0` in the model file. All five widths are the same material described by the same sheet resistance.

The same calculation on the `high` column gives 379.9, 325.8, 309.7, 300.8 and 323.5 Ω/□, which don't agree with each other. The narrow-width flavours are separately fitted devices, and `res_high_po_0p35`

behaves like a 380 Ω/□ material rather than a 317 Ω/□ one. So if you move a design from one width to another, re-simulate it instead of scaling the old value by hand.

10 kΩ four ways

Here is the same 10 kΩ target in four flavours, with each length tuned until the simulator agreed:

Device	W (µm)	L (µm)	R measured	**area**
`res_xhigh_po_0p35`	0.35	1.647	9998.007 Ω	**0.576 µm²**
`res_high_po_0p35`	0.35	8.351	10000.90 Ω	**2.923 µm²**
`res_generic_nd`	1	85.1	9998.521 Ω	**85.1 µm²**
`res_generic_po`	1	196.4	10001.30 Ω	**196.4 µm²**

That's a factor of 341 in area for the same 10 kΩ, and it's the reason a process bothers to have a high-resistance implant at all: without one, a modest bias resistor is bigger than a logic gate.

For scale, here is the smallest inverter's size, from `/foss/pdks/sky130A/libs.ref/sky130_fd_sc_hd/lef/sky130_fd_sc_hd.lef`:

```
MACRO sky130_fd_sc_hd__inv_1
  SIZE 1.380 BY 2.720 ;
```

That's 3.7536 µm² for the smallest inverter in the standard-cell library, so the 10 kΩ `res_xhigh_po_0p35` is smaller than one inverter, while the `res_generic_po` version takes the area of fifty-two of them. Scale that up to 100 kΩ in ordinary poly (1964 µm², about 520 inverters) and the resistor is bigger than the circuit it biases, so choosing the resistor flavour is a design decision in its own right.

Capacitors

Capacitor table

Each capacitor was driven by a 1 V AC source at 1 kHz, and its value computed as $C = |I|/(2\pi f V)$ with $V = 1$ V:

Device	Geometry	C measured	density
`cap_mim_m3_1`	2 × 2 µm	9.302250 fF	2.326 fF/µm²
`cap_mim_m3_1`	10 × 10 µm	206.5822 fF	2.066 fF/µm²
`cap_mim_m3_1`	22.2 × 22.2 µm	1000.314 fF	2.030 fF/µm²
`cap_mim_m3_2`	10 × 10 µm	206.5822 fF	2.066 fF/µm²
`cap_vpp_11p5x11p7_l1m1m2m3m4_shieldm5`	11.5 × 11.7 µm (fixed)	147.3400 fF	1.095 fF/µm²
`nfet_01v8` as a MOS cap, gate at 1.8 V	10 × 10 µm gate	787.6883 fF	7.877 fF/µm²
the **same** device, gate at 0 V	10 × 10 µm gate	220.7182 fF	2.207 fF/µm²
`cap_var_lvt` at 1.8 V	10 × 10 µm	817.8665 fF	8.179 fF/µm²
the **same** varactor at 0 V	10 × 10 µm	99.51349 fF	0.995 fF/µm²

A few things in the table are worth pointing out:

1. `m3_1` and `m3_2` are the same capacitor built on different metals. `cap_mim_m3_1` sits between met3 and the `capm` plate, and `cap_mim_m3_2` sits between met4 and `cap2m`. Their capacitance is identical to seven digits, so you choose between them by what you need to route underneath.

2. MIM density rises as the plate shrinks, because the perimeter term is fixed per micron of edge and a smaller plate has more edge for its area: 2.326 fF/µm² at 2 × 2 µm against 2.030 at 22.2 × 22.2 µm.
3. A MOS capacitor is 3.8× denser than a MIM, but its value depends on the voltage across it: 787.6883 fF at 1.8 V and 220.7182 fF at 0 V, **a factor of 3.57 apart on the same physical device.** Use it for supply decoupling, where one plate is always at the rail, and never in a filter whose corner frequency you care about.
4. The varactor uses that nonlinearity on purpose. It goes from 99.5 fF to 817.9 fF, a factor of 8.22, over the supply range, which makes it the tuning element in an oscillator.

MIM capacitance from the model parameters

The MIM model comes down to three lines of `/foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/sky130_fd_pr__cap_mim_m3_1.model.spice`:

```
.param wc = 'w+m3_dw*1e6+tol_m3*1e6'
.param carea  = 'camimc*(wc)*(lc)'
.param cperim = 'cpmimc*((wc)+(lc))*2'
```

(`lc` is defined the same way as `wc`, on the line immediately after it.)

and three constants, which live in `/foss/pdks/sky130A/libs.tech/ngspice/r+c/res_typical__cap_typical__lin.spice` and `.../sky130_fd_pr__model__r+c.model.spice`:

```
+ camimc=  2.00e-15  ; Units: farad/micrometer^2
+ cpmimc = 0.19e-15 ; Units: farad/micrometer
+ m3_dw = -0.025u
```

So when you draw a plate 10 µm square, the metal etches back by 0.025 µm and the real plate is 9.975 µm on a side:

$$+0.19 \times 2 \times (9.975 + 9.975) = 199.00125 + 7.581 = \boxed{206.582\ \text{fF}} \tag{1.1}$$

ngspice measured `2.065822e-13`, which matches to six significant figures using only three constants and a subtraction. When a PDK number and your own arithmetic agree that closely, you can be confident you understand how the device is modelled.

What 1 pF costs

Device	Area for 1 pF	Caveat
MOS cap at 1.8 V	127 µm²	only if one plate stays at a rail
MIM	**493 µm²**: 22.2 × 22.2 µm, measured at 1000.314 fF	linear; the usual choice for a signal path
VPP fringe	913 µm²	it uses met3 and met4, the MIM's own layers, so it can't share area with a MIM

493 µm² is about the area of 130 inverters. It's the number to bring to any "just put a 1 nF cap on it" conversation, because a nanofarad of MIM is half a square millimetre, the area of about 130 000 inverters. What a picofarad costs goes through this at length.

Parasitic capacitance to substrate

Every layer has capacitance to the substrate, whether you asked for it or not. These values come from the `defaultareacap` lines in Magic's `/foss/pdks/sky130A/libs.tech/magic/sky130A.tech`, which gives them in aF/µm²:

Layer to substrate	aF/µm²	as fF/µm²
poly	106.13	0.106

Layer to substrate	aF/µm²	as fF/µm²
local interconnect (li)	36.99	0.037
met1	25.78	0.026
met2	17.5	0.018
met3	12.37	0.012
met4	8.42	0.008
met5	6.32	0.006
n-well / deep n-well	120	0.120

So a 100 µm × 1 µm met1 wire carries 2.6 fF to ground, which is more than a quarter of a 10 fF capacitor you might draw on purpose elsewhere. Parasitic capacitance covers this in more detail, including the sidewall terms that this table leaves out.

Inductors

SKY130 has no good inductor. It ships three spiral models (`sky130_fd_pr__ind_03_90`, `ind_05_125` and `ind_05_220`). `ind_05_220` is 9.92 nH, occupies tens of thousands of µm², and has **Q = 0.0000150516 at 1 kHz**. The best Q any of the three ever reaches is **21.1, at 6.38 GHz**, and above 3.52777 GHz the reactance of `ind_05_220` changes sign. Inductors on a chip and What to build instead of an inductor cover why, and what replaces it.

Corners

`tt` is the typical corner, for transistors and passives alike. Passives also have corners of their own, selected by the same word at the end of the `.lib` line: `ll` (low resistance, low capacitance) and `hh` (high resistance, high capacitance), plus `hl` and `lh`.

We measured these spreads rather than reading them off a parameter file, by simulating the same four devices three times and changing only the corner name:

Device	`ll`	`tt`	`hh`	spread about `tt`
`res_high_po` W=1 L=10	3106.637 Ω	3550.443 Ω	3994.248 Ω	**±12.5 %**
`res_xhigh_po` W=1 L=10	18037.95 Ω	21221.12 Ω	24404.29 Ω	**±15.0 %**
`res_generic_po` W=1 L=10	451.1426 Ω	509.2314 Ω	583.1667 Ω	−11.4 % / +14.5 %
`cap_mim_m3_1` 10 × 10	179.7322 fF	206.5822 fF	233.8667 fF	−13.0 % / +13.2 %

The parameters behind those runs are set in `/foss/pdks/sky130A/libs.tech/ngspice/r+c/res_low__cap_low.spice`, `res_typical__cap_typical.spice`, `res_high__cap_high.spice` and their `__lin` siblings:

Parameter	`ll`	`tt`	`hh`
`rp1` (ordinary poly)	44	48.2	53.52
`rdn` (n+ diffusion)	111.6	120	128.4
`rdp` (p+ diffusion)	175.3	197	218.7
`rl1` (local interconnect)	10.31	12.2	14.02
`rm1` (metal 1)	0.111	0.125	0.139

Parameter	ll	tt	hh
rnw (n-well)	1378	1700	2022
sky130_fd_pr__res_high_po__var	−0.125	0.0	+0.125
camimc (MIM)	1.778e-15	2.00e-15	2.231e-15

A device doesn't always move by exactly its parameter's percentage. rp1 shifts by −8.7 % and +11.0 %, but a real res_generic_po moves by −11.4 % and +14.5 %, because the corner also changes the etch bias that sets the strip's effective width, so **simulate the corner rather than scaling the typical answer by hand.**

Every device measured above moves by more than ±10 % between corners. A *ratio* of two of the same device stays put, which is why analog design is built on ratios; see Matching and Pelgrom's law.

Re-running these numbers

The numbers come from two sets of decks in labs/passives-decks:

```
cd labs/passives-decks
make            # the guide pages' decks: head/body, corners, MIM, spirals, mismatch
make catalogue  # this page's extra devices: the seven materials at one square,
               # both fixed-width families, 10 kΩ four ways, and the MIM/varactor rows
```

make compares its results against 45 stored values and ends with PASS or FAIL. make catalogue prints the numbers from spice/catalogue.spice for you to read against the tables above, and it takes about a minute.

You don't need any environment setup, because every deck names the model library by its absolute path, so a bare container works. The related labs are Lab 01: A resistor you designed, Lab 02: How big is a picofarad? and Lab 04: Two resistors that disagree.

B. ngspice deck recipes

This page gives the smallest deck that measures a passive device for each kind of measurement in AD102. It also shows what ngspice says when you get a deck wrong, and lists the warnings each lab prints that you can safely ignore.

AD102 asks five questions of silicon: how many ohms, how many farads, how many henries, how long the time constant is, and how much all of that moves. Each question has a deck shape you can reuse, and this page shows all five, with the exact output each one produced in `hpretl/iic-osic-tools:2026.08`, ngspice 47, PDK `sky130A`.

Every block below is an **excerpt** from a deck that ships in a lab package, and the deck's path is given beside it. None of them is a file for you to create; if you want to try something, run the shipped deck and edit that.

The five parts of a deck

Write the parts in this order. ngspice itself insists on only one position: whatever is on line 1 is the title. It doesn't check that `.end` comes last. On the pinned image a deck with no `.end` still runs, and a device line after `.end` still gets simulated, so putting `.end` last is a convention for whoever reads the deck rather than something ngspice enforces.

Part	Looks like	Notes
1. title	`* AD102 Lab 01 -- the sheet resistance deck`	**The first line is always a comment**, whatever you put in it, so put a real title there. If a device line lands on line 1, ngspice swallows it as the title.
2. models	`.lib /foss/pdks/.../sky130.lib.spice tt`	an absolute path, so the deck runs from any directory
3. devices and sources	`Xa a 0 0 sky130_fd_pr__res_high_po W=1 L=10`	see the catalogue
4. what to do	`.control` … `.endc`, or `.op` / `.ac` / `.tran` cards	
5. `.end`	`.end`	

Run the deck in batch mode, which runs the analysis, prints the results and exits:

```
ngspice -b spice/sheet.spice
```

Without `-b`, ngspice leaves you at an interactive prompt and the terminal looks as though it has hung; type `quit` to get your results.

1. Resistance: `.op` and Ohm's law

ngspice has no "measure the resistance" command, so you push a current of your own choosing through the device and divide the voltage that results by that current.

From `labs/lab-01-a-resistor-you-designed/spice/sheet.spice`:

```
Il1 0 l1 dc 1u
Xl1 l1 0 0 sky130_fd_pr__res_high_po W=1 L=1
```

```
.control
op
let r_L1 = v(l1)/1u
print r_L1
.endc
```

```
r_l1 = 6.954646e+02
```

That's 695.4646 Ω for one drawn square of a device whose model card says 317.3885 Ω/□. The gap between those two numbers is the subject of Ohms per square.

Three details of that pattern are worth knowing:

- **The 1 µA is chosen on purpose.** It's small enough that the resistor's own self-heating and voltage coefficient don't move the answer, and large enough to stay well clear of numerical noise. If you push 1 mA through 100 kΩ, you're asking a different question.
- **Use `let` before `print`.** `v(l1)/1u` is an *expression*, while `r_L1` is a vector. `print` accepts either, but `meas` doesn't accept an expression, so it's worth getting into the habit of naming the result with `let`, which costs nothing.
- **Vector names come back lower-case.** You wrote `r_L1`, and ngspice prints `r_l1`, which matters when you `grep` the log for it.

If you drive the device with a voltage source instead, the current comes out of the source and is negative, so the resistance is `1m/(-i(va))`, as in `labs/passives-decks/spice/corners.spice.in`:

```
let R_A = 1m/(-i(va))
```

2. Capacitance: one `.ac` point

A capacitor has no DC behaviour to measure, so instead you drive it at one frequency, measure the current, and undo $Z = 1/(j\omega C)$ to get the capacitance.

From `labs/lab-02-how-big-is-a-picofarad/spice/picofarad.spice`:

```
V4 a4 0 dc 0 ac 1
X4 a4 0 sky130_fd_pr__cap_mim_m3_1 W=10 L=10
```

```
.control
ac lin 1 1e6 1e6
let k = 1/(2*pi*1e6)
let c_s10 = abs(i(v4))*k
print c_s10
.endc
```

```
c_s10 = 2.065822e-13
```

That's 206.5822 fF for a 10 × 10 µm MIM plate.

- **`ac lin 1 1e6 1e6`** asks for a single point at 1 MHz, because you don't need a sweep to read one value.
- **`ac 1` on the source is required.** With only `dc 0`, you get zero everywhere, and ngspice gives no warning.
- **The 1 MHz is arbitrary, because it cancels out.** Any frequency works, as long as the same number appears in `k`. As a sanity check, try a decade either side: if the answer moves, your device isn't a pure capacitor (see Parasitic capacitance).

3. Inductance: `.ac` and complex arithmetic

An on-chip inductor is a resistor with some inductance attached, so you can't use the capacitor trick unchanged; you have to separate the imaginary part of the response from the real part.

From `labs/lab-02-how-big-is-a-picofarad/spice/henry.spice`:

```
.control
ac lin 1 1e8 1e8
let l_090_100M = real(imag(v(c))/(2*pi*1e8))
print l_090_100M
.endc
```

```
l_090_100M = 1.520796e-09
```

At 2.4 GHz the same deck works out the quality factor as well:

```
let q_090_2G4 = real(imag(v(c))/real(v(c)))
let r_090_2G4 = real(real(v(c)))
```

```
q_090_2g4 = 1.086413e+01
r_090_2g4 = 2.157013e+00
```

That's 1.5208 nH at 100 MHz and, at 2.4 GHz, Q = 10.86 and 2.157 Ω of series resistance you didn't ask for.

You need the outer `real(...)`, because as far as ngspice's type system is concerned, `imag(v(c))` is still a complex vector, one whose imaginary part happens to be zero. Wrapping it in `real()` gives you a number you can print without a warning.

4. Time constant: `.tran` and `meas`

From `labs/lab-03-a-time-constant-in-silicon/spice/rc.spice`:

```
Vb bi 0 PULSE(0 1.8 1n 10p 10p 200n 400n)
```

```
.tran 2p 80n

.control
run
meas tran t_ideal  WHEN v(bo)=1.137816 RISE=1
.endc
```

The threshold, 1.137816 V, is 0.632120×1.8, so the time at which `v(bo)` crosses it is τ by definition.

The `.tran` is a card outside `.control`, and `run` executes it. That looks like extra work until you want the same analysis at another temperature. `set temp` takes effect only on the next `run`, whereas a `tran` typed inside `.control` runs immediately and ignores any temperature you set afterwards. That's how `rc.spice` runs −40 °C and +125 °C as well as 27 °C from this one card. If you restructure the deck and your cold and hot numbers come out identical to the 27 °C one, this is the reason.

That `meas` line has two traps in it, and together they add 1.005 ns to the answer:

- `meas ... when` returns an **absolute time** rather than an interval. The pulse starts at $t = 1$ ns, so you subtract 1 ns.
- The pulse has a **10 ps rise time**, and a first-order system behaves as though the step arrived at the ramp's midpoint, so you subtract another 5 ps.

The total offset is 1.005 ns, and `src/check_rc.py` subtracts exactly that. If you forget it, a 10 ns time constant reads 11.005 ns, which is 10 % high, and the error is *constant*: it stays 1.005 ns whatever the time constant is. That's the clue to look for, because an error that doesn't scale with the thing you're measuring comes from the measurement rather than from the device.

5. Variation: corners and Monte Carlo

Corners and Monte Carlo answer two different questions, using two different mechanisms, and confusing them is the most common mistake in this half of the course.

Corners: the whole wafer lot is high or low

You choose a corner with one extra word at the end of the `.lib` line, which picks a section of the model file:

```
.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt
```

Section	Means
`tt`	typical, the middle of the distribution
`ll`	low resistance, low capacitance
`hh`	high resistance, high capacitance

`labs/passives-decks/` builds one deck per corner from `spice/corners.spice.in`, and the three decks give:

	`ll`	`tt`	`hh`
R_A, drawn 1 × 10 µm	3106.637 Ω	3550.443 Ω	3994.248 Ω
R_B, drawn 1 × 40 µm	11433.66 Ω	13067.04 Ω	14700.42 Ω
MIM, 10 × 10 µm	179.7322 fF	206.5822 fF	233.8667 fF
RC corner frequency	282.935 MHz	215.410 MHz	169.052 MHz
R_B/R_A	**3.680396**	**3.680396**	**3.680396**
divider tap	**0.3845828 V**	**0.3845828 V**	**0.3845828 V**

Compare the last two rows with the first four. The absolute values move by ±12.5 % and the corner frequency by +31 % and −22 %, but **the ratio doesn't change even in the seventh significant figure.** That table is Matching and Pelgrom's law in miniature, and it's the reason analog designers build ratios.

Mismatch: two devices on the same die differ

A corner moves every device together. Mismatch moves devices on the same die apart from each other, so it needs a different switch.

From `labs/lab-04-two-resistors-that-disagree/spice/mismatch.spice`:

```
.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.param mc_mm_switch=1
```

```
.control
setseed 1
op
* ... measure ...
mc_source
op
* ... measure again: a different imaginary die ...
.endc
```

- **`mc_mm_switch=1`** turns on the per-instance `AGAUSS` term in SkyWater's model cards. Without it, every instance of the same device is identical, so there is no mismatch to see.
- **`mc_source` re-evaluates the netlist**, drawing a fresh sample for every instance. If you loop it 200 times, you have 200 imaginary dies.
- **`setseed 1`** makes the sequence reproducible, which is why your Lab 04 numbers match the writeup to the digit.

Why the decks are slow

Every AD102 deck spends a while apparently doing nothing before it prints anything. What it's doing is reading model cards: `.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt` walks a

tree of `.includes` that covers every device in SKY130, about 12 MB in all, and nearly all of it is MOSFET models that a passive deck never uses.

`.lib` line	wall clock, `spice/sheet.spice`
`sky130.lib.spice tt`	55–65 s on the pinned image; about 10 s on the machine this page was last checked on

Those are wall-clock seconds, so they change with how busy the machine is, whereas the resistances don't change at all.

Earlier workbench images also shipped pre-flattened files such as `sky130.lib.spice.tt.red`, which loaded about thirty times faster. The 2026.08 image doesn't include them, so every deck in this course, including `spice/bode.spice` in Lab 03, now loads `sky130.lib.spice`. If a deck or a tutorial you find elsewhere names a `.red` file, ngspice on this image stops with `Could not find library file`. If you're editing a lab's shipped deck, leave the `.lib` line alone, because the reference numbers were taken with it.

Warnings and delays in each lab

This section lists what each lab prints that isn't an error, and how long each one stays silent while it works. A line that starts with `Error` is a different matter: it stops the run, and Error messages explains what it means.

Lab 01

Every ngspice run loads `.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt`, which is 12 MB of model cards for every device in SKY130, mostly MOSFETs you aren't using, and it prints nothing while it loads. That takes anywhere from about 10 seconds to just over a minute per run, depending on your machine (the comment at the top of the Makefile records 55–65 s). `make` runs the simulator twice, so allow up to about two minutes. The silence doesn't mean it has hung.

`results/sheet.log` holds 170 `unrecognized parameter` lines, in 34 blocks. `make` writes them to the log rather than the terminal, so from this deck your terminal shows only the `---` and `r_` lines. A `rhead_model` block has four of these lines and a `rbody_model` block has six. Here is a head block:

```
Warning: Model issue on line 4842 :
  .model xl1:rhead_model r sw_et=0 isnoisy=0 rsh=    3.458312000000000e+02 ...
unrecognized parameter (sw_et) - ignored
unrecognized parameter (isnoisy) - ignored
unrecognized parameter (p2) - ignored
unrecognized parameter (q2) - ignored
```

SkyWater writes one set of model cards to serve several simulators. `sw_et`, `isnoisy` and `p2`/`q2`/`p3`/`q3` are Spectre-flavoured keywords for self-heating, noise and voltage coefficients, which ngspice doesn't implement. ngspice says so for each one and carries on with the parameters it does understand, which are the ones the lab measures. There are thirty-four blocks for twenty-two resistors because most of these devices expand into more than one `.model` card.

`make resistor` in `passives-decks` (Lab 01, Part 6) prints fewer of these lines, because its deck has fewer resistors: `grep -c 'unrecognized parameter' results/r_head_and_body.log` prints `100`.

If a line starts with `Error`, the run has stopped, and your log will contain no `r_` lines at all. The checker, `src/check_res.py`, tells you to look for exactly that.

Lab 02

`pf`, `mine` and `tank` each load SKY130's model cards before printing anything. `henry` doesn't, because the inductor models live in three small files outside the corner library, so it finishes almost immediately. The Makefile budgets about three and a half minutes for the whole `make` on the pinned image, so a long silence is normal.

Lab 03

Each of the two runs in `make` is silent while ngspice loads 12 MB of SKY130 model cards.

`Doing analysis at TEMP = ... and TNOM = 27.000000` appears five times in `results/rc.log`, at 27 °C, −40 °C and +125 °C. That comes from Part 4, where the deck runs the same `.tran` card at three temperatures. `TNOM` stays at 27, and that's correct: it's the default temperature that model parameters are referred to, not the one you're simulating. (A model card can set its own; `res_high_po`'s says `tnom = 30`, which is the value Lab 01's Going further uses.)

`Warning: command 'plot' is not available during batch simulation, ignored!` comes from the schematic in the lab's XSchem part, which asks for a plot that batch mode can't draw. The `meas` line under the warning still gives you the answer.

Lab 04

Each of the two decks loads the SKY130 model cards (about 55 s on the pinned image) and then evaluates 200 operating points (about 23 s for the bigger of the two), printing nothing while it works. Most of that time goes on loading the models rather than on the Monte Carlo itself. On the machine this page was last checked on, where the models load in about 10 s, each deck took 14 to 16 s in all and a full `make` about 30 s, the low end of the lab's estimate.

`results/mismatch.log` holds 28,280 `unrecognized parameter (...) - ignored` lines, of which 5,656 are `(sw_et)`. They're the same harmless lines as in Lab 01. Each of the two hundred re-evaluations of the netlist repeats the whole block, which is why the log is 1.8 MB. `make` sends all of it to the log rather than to your terminal.

The loop counter must exist before the first `mc_source`. `mc_source` throws away the vectors from the previous analysis, and a counter created after an analysis has run is thrown away with them. If you get this wrong, you'll see the following, buried among the usual model warnings:

```
Warning from checkvalid: vector run is not available or has zero length.
Error: RHS "run + 1" invalid
```

After that the loop stops at one die, and every σ you compute comes out as zero, with no further message and an exit status of 0. That's why `make` prints the operating-point count first: you're looking for 201, and if you see 2, this is the cause. The shipped decks create the counter immediately after `setseed 1`, before anything has been simulated:

```
setseed 1
let run = 0          <- here
dowhile run < 200
  mc_source
  ...
```

`X` lines and `R` lines

Most SKY130 passives are **subcircuits**, so their lines start with `X` and take capital `W=` / `L=`:

```
XR1 a 0 0 sky130_fd_pr__res_high_po W=1 L=10 mult=1
```

A few (`res_generic_nw`, `res_generic_l1`, and all the metal resistors) are bare `.model` cards, so their lines start with `R` and take lower-case `w=` / `l=`:

```
r1 b 0 sky130_fd_pr__res_generic_m1 w=1 l=1
```

If you get it backwards, ngspice tells you so, with a different message for each direction, and both messages are in the table below. The catalogue lists which device takes which.

Units of W and L

In a SKY130 deck, `W`, `L`, `w` and `l` carry **no unit suffix**, so `W=1` means one micron. Every other number on the line takes SI suffixes as usual (`1k`, `0.7p`, `3n`), which is why this mistake is so easy to make.

On a resistor the mistake gives no error at all. Getting started goes through an example: the same device gives 3550.443 Ω when written `W=1 L=10` and 3193.812 Ω when written `W=1u L=10u`. That's an ordinary-looking answer 10 % away from the right one, and ngspice still exits with `0`.

If a resistance is close but not right, look for a stray `u` before you look anywhere else.

Error messages

ngspice says	You did
`Warning: r.xb.rbody: resistance too low or not given, set to 1 mOhm`	put a `u` on `W`/`L` of a resistor. **The run continues, and the answer is wrong.** This is the quiet version of the trap.
`could not find a valid modelname … Simulation interrupted due to error!`	put a `u` on `W`/`L` of a *transistor*, or asked for a size that no model bin covers. This is the loud version, and it's the better outcome, because you can't miss it.
`Error: unknown subckt: xr6 f 0 sky130_fd_pr__res_generic_l1 w=1 l=1`	started the line with `X` for a device that is a plain `.model`. Use `r6 f 0 …`.
`Too many parameters for subcircuit type "sky130_fd_pr__res_generic_po" (instance: xxr1)`	passed `mult=1` to a device whose subcircuit has no `mult`. Not every resistor takes the same parameters.
`Too few parameters for subcircuit type "sky130_fd_pr__cap_var_lvt" (instance: xxcv)`	gave the wrong number of *nodes*. The varactor has three terminals, not two.
`Error: no such vector as -i(vds).`	put an expression where `meas` wanted a vector. Make it a vector first with `let id = -i(Vds)`.
`Error: incomplete or empty netlist / no simulations run!`	nothing, on its own. This is the second message, and the real error is a few lines above it.
`Warning: command 'plot' is not available during batch simulation, ignored!`	asked a batch run for a picture. It's harmless, because the `meas` lines still ran. You'll see it when you netlist a schematic that has `plot` in its control block.
ten `unrecognized parameter (...) - ignored` lines per `res_high_po`, two of them `(sw_et)`	nothing wrong. Every deck with a `res_high_po` in it prints these; see Getting started.

Exit codes

Deck	exit
a correct run	`0`
`W=1u` on a transistor, which gives `could not find a valid modelname`	`1`
`W=1u` on a **resistor**, which comes out 10 % wrong without an error	`0`
a deck whose `meas` failed	`0`

The exit status in `$?` catches a deck that wouldn't build, and nothing else. That's why every AD102 `Makefile` reads the log and prints its own `PASS` or `FAIL`, and any script you write should do the same.

Related: The SKY130 passive catalogue lists which devices exist, which line type each takes, and what each one measures. AD103′s ngspice survival card covers the same tool from the transistor side.