AD103 — Nonlinear Circuits

Simulate diodes and MOSFETs with the SKY130 device models, and finish with a CMOS inverter that you draw in XSchem and size yourself.

UofT ASIC Team

Contents

1 AD103 — Nonlinear Circuits 5

I Start here

2 Getting started 11

3 Lab 01: Your first schematic 19

II The diode

4 Linearity and superposition 31

5 The operating point 35

6 The diode equation 39

7 Three diode models 45

8 Lab 02: The diode I–V curve 49

III The MOSFET

9 How a MOSFET works 63

10 The four regions 67

11 Where saturation starts 73

12 Lab 03: MOSFET regions 77

IV Device parameters

13 What moves the threshold 89

14 Transconductance and output resistance 93

15 Sizing with W and L 97

16 Lab 04: Testing the W/L rule 101

V Capstone

17 The CMOS inverter 113

18 The current mirror 119

19 Capstone: The CMOS inverter 123

Appendices 135

A XSchem cheat sheet 137

B The ngspice survival card 147

C Reading a SKY130 device model 155

1. AD103 — Nonlinear Circuits

In this course you draw transistors in XSchem, simulate them with real SKY130 models, and learn to tell a believable answer from a wrong one. It is the third course in the analog track, and the first one where the circuits stop being linear.

AD103 is part of the **UofT ASIC Team** education materials, and it's published at **https://uoftasic.com/ad103/**. The labs run inside the shared **workspace** you set up in IC101.

At a glance

Who it's for	2nd–3rd year ECE students on the analog track
Track	Analog
Prerequisites	IC101 → AD101 → AD102, in that order
Main tool	**XSchem**, the schematic editor. Lab 01 assumes you have never used it before.
Also	ngspice 47 and the SKY130 device models, both already in the workbench
Math level	Algebra, exponentials, logarithms, and one square root. You won't need calculus.
Time	About 16.5 hours, self-paced

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 lists the pages to read before you start it.

#		What	Time
1	Read	Getting started	30 min
2	Do	Lab 01: Your first schematic. Draw one NMOS in XSchem, simulate it, and break it twice on purpose.	2 h
3	Read	Linearity and superposition · The operating point · The diode equation · Three diode models	1.5 h
4	Do	Lab 02: The diode I–V curve. Sweep a SKY130 diode, extract two of its model parameters, and heat it to 100 °C.	1.75 h

#		What	Time
5	Read	How a MOSFET works · The four regions · Where saturation starts	1.5 h
6	Do	Lab 03: MOSFET regions. Find the four regions on your own NMOS curves, extract its threshold two ways, and add a curve to the SPICE deck.	2.25 h
7	Read	What moves the threshold · Transconductance and output resistance · Sizing with W and L	1.5 h
8	Do	Lab 04: Testing the W/L rule. Predict six currents from W/L, then measure where the rule holds and where it breaks.	2 h
9	Read	The CMOS inverter · The current mirror	1 h
10	Do	Capstone: The CMOS inverter. Draw and size your own inverter, time it, run it as an amplifier, and measure a current mirror.	2.5 h

The times are estimates. Your first lab will probably take longest, because you are learning XSchem at the same time.

Keep these open while you work: XSchem cheat sheet · The ngspice survival card · Reading a SKY130 device model

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

The workbench

This course is pinned to the workbench image **`hpretl/iic-osic-tools:2026.08`**, which ships **XSchem 3.4.8RC** and **ngspice-47**, and Lab 01 has you check both versions.

> **Warning:** the image's default PDK is not SKY130. If you launch XSchem without changing it, you get IHP's 130 nm process instead, which has none of the symbols this course uses, and XSchem gives you no error message. Getting started explains what to look for, and Lab 01 has you check it.

The warning applies when you start XSchem yourself. The lab `make` targets don't depend on the PDK setting, because every lab package pins its own PDK and model paths. Even if Lab 01, Part 1 can't load the course environment for you, `make` in a lab folder still runs to completion in a bare container, and each lab ends in `PASS` or `FAIL` with a reason.

What this course is about

AD101 taught you to read a signal. AD102 taught you that a resistor is a doped strip of silicon whose value is set by its shape. In both, doubling the input doubled the output.

The diode and the MOSFET don't behave that way, and most of what makes a chip work depends on that: a transistor is useful because its response bends. When you amplify with it, the bend gives you gain, and when you switch with it, the bend gives you on and off. If you push it too far either way, the same bend becomes distortion.

The algebra from AD102 can't solve these circuits, so in this course you learn to ask a simulator instead, and to know enough to spot a wrong answer when it gives you one.

What you'll learn

1. Draw a schematic in XSchem from an empty canvas, and read the SPICE netlist it writes
2. Sweep a real SKY130 diode and measure how far it is from the textbook's **60 mV per decade**
3. Map the **four regions** of a MOSFET, including the one most textbooks skip
4. Extract a threshold voltage from your own curve two different ways, and explain why the two answers differ by 86 mV
5. Find out what W and L buy you, and where the W/L rule breaks: five devices with identical W/L carry five different currents
6. Build a **CMOS inverter** from two transistors, measure the voltage at which it switches, and see that sizing it for equal delay and sizing it for a centred threshold give different answers

By the end you'll have a schematic you drew, a set of measured device curves you can defend, and a habit the rest of the analog track depends on: predict the number, run the simulation, and work out which of the two was wrong.

Next courses

- **AD104** — Layout: you draw those transistors as geometry in Magic, and make them pass DRC and LVS
- **AD201** — Analog Signal Processing · **AD202** — Mixed Signal

I Start here

2 **Getting started** 11
2 The PDK setting 11
2 Where XSchem starts 12
2 Units on `W` and `L` 13
2 How wires connect 14
2 The netlist 15
2 Simulation time 17
2 Getting help 17
2 Check yourself 18

3 **Lab 01: Your first schematic** 19
3 Before you start 19
3 Part 1: Load the course 19
3 Part 2: Prove the toolchain works 20
3 Part 3: Look around the reference schematic . . . 20
3 Part 4: Draw your own transistor 22
3 Part 5: Simulate your drawing 25
3 Part 6: Units on `W` and `L` 26
3 Part 7: A gate wire that stops short 26
3 Part 8 (optional): Your device's model bin 27
3 Finishing 27
3 Answers 28

2. Getting started

Nearly everything you do in AD103 happens in **XSchem**, a schematic editor. You draw a circuit, XSchem turns the drawing into a SPICE netlist, and ngspice simulates the netlist and hands you back numbers:

$$\text{schematic} \rightarrow \text{netlist} \rightarrow \text{ngspice} \rightarrow \text{a number} \tag{2.1}$$

You have probably never used XSchem, so this page explains how those pieces fit together, and goes through the handful of mistakes that cost new users the most time. You won't install anything here. AD103 needs one Docker image and a browser, and you already have both from IC101. You'll start the tools and draw your first schematic in Lab 01.

The PDK setting

The workbench image can simulate more than one manufacturing process. The environment variable `PDK` chooses which one, and the image's default is `ihp-sg13g2`, a real 130 nm process from IHP in Germany. We use **SKY130** for everything in AD103, so you'll need to change it.

This matters more in AD103 than anywhere else in the curriculum, because XSchem reads `$PDK` when it starts up to decide which device library to load, and it doesn't tell you which one it picked. If you launch it with the default setting, you get an editor that looks perfectly healthy:

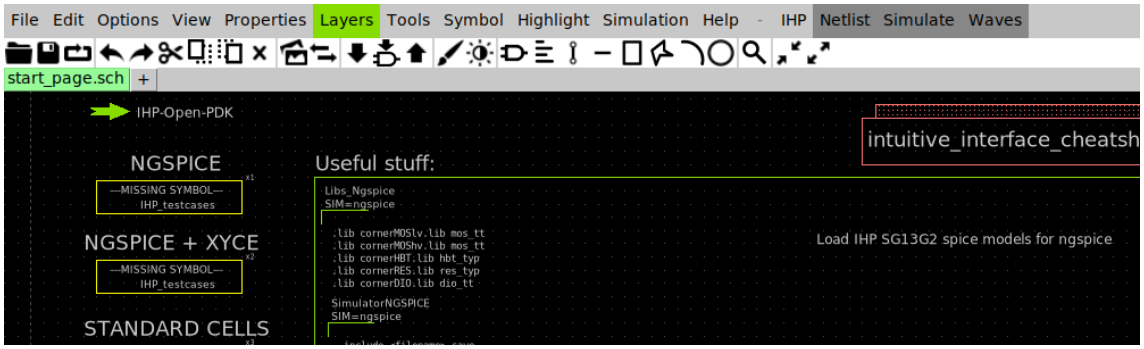


Figure 2.1: XSchem started with the image's default PDK — note IHP in the menu bar

*The menu bar says **IHP**, the tab says `start_page.sch`, and the page offers to load "IHP SG13G2 spice models". XSchem gives no error and no warning.*

None of the `sky130_fd_pr/...` symbols this course uses will be there. If you draw a circuit anyway, the netlist will have `IS MISSING !!!!` next to your transistor, and XSchem will still report that it finished successfully.

The fix is a setup script from the workspace repo, `/foss/designs/common/.designinit`, which sets `PDK=sky130A`. You load it with a leading `.`, which is the shell's `source` command. That runs the file in your current shell rather than in a new one, and it's the only way a script can change your shell's environment. The catch is that the change lasts only as long as that shell, so you have to load the script again in every new terminal you open.

Checking takes a couple of seconds: `echo $PDK` should print `sky130A`, and XSchem's menu bar should say **SKY130**. You'll make the first check in Lab 01, Part 1, and the second in Part 3, and it's worth making a habit of both.

Where XSchem starts

When XSchem starts, it reads a settings file called `xschemrc` from the directory you launched it in. Each AD103 lab has one in its `xschem/` folder. That file is what makes the SKY130 symbols resolve, and it also puts your netlists somewhere you can find them. Three habits follow from this:

- **Launch XSchem from a terminal, not from a desktop icon.** Some of XSchem's messages are printed to the terminal it was started from. If you start it from a menu there is no such terminal, so those messages are lost, and the program can appear to do nothing at all.
- **Launch it from the directory your schematics are in**, so that it reads that directory's `xschemrc`. If symbols that worked yesterday have vanished today, you probably started XSchem from somewhere else.
- **Name the file you want to open**, as in `xschem nmos_probe.sch &`. If you don't give a filename, XSchem opens whatever its settings name as the start page, which is the subject of the next section.

The PDK start page

Every AD103 lab sets the start page to a blank `untitled.sch` in the lab directory, so inside a lab folder a bare `xschem &` does no harm. Anywhere else (a scratch folder of your own with no `xschemrc`, say) you get the SKY130 PDK's own start page instead:

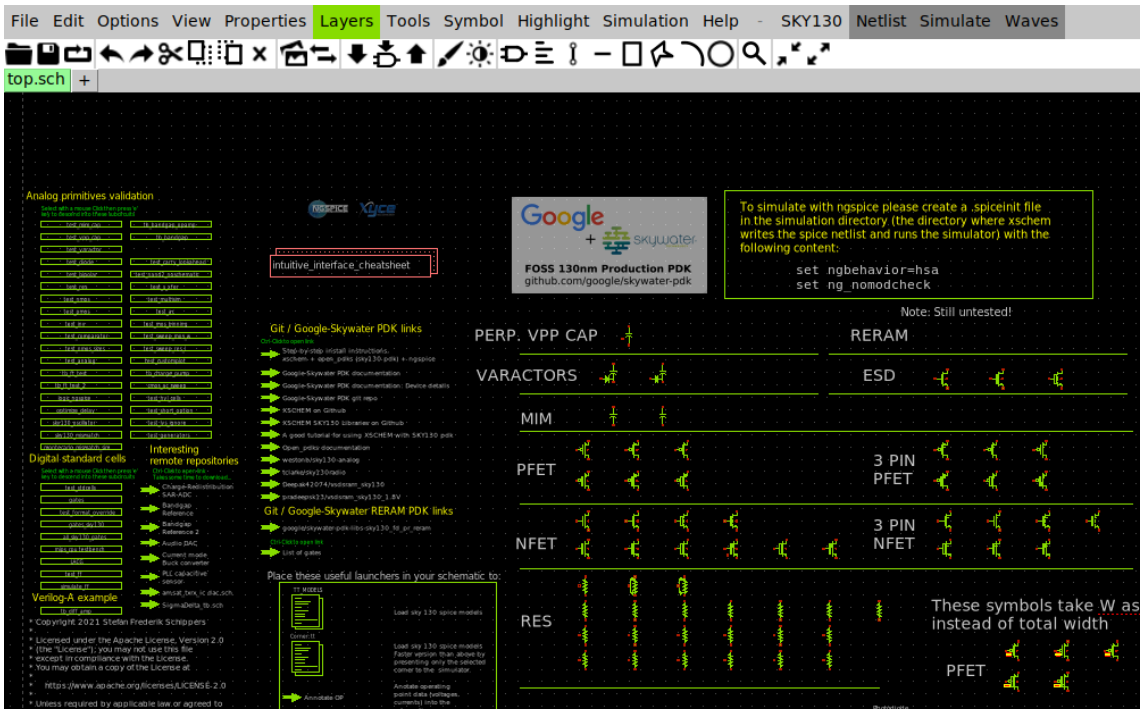


Figure 2.2: The SKY130 start page XSchem opens on when no filename is given

*The menu bar reads **SKY130**, and the canvas is a live schematic: the NFET, PFET, RES, MIM and VARACTOR rows are real symbols. It looks like any other page you could work on, but you can't save anything to it.*

That page is a real file, `/foss/pdks/sky130A/libs.tech/xschem/sky130_tests/top.sch`, and it sits inside the PDK, which is read-only. Two things go wrong if you work on it, and XSchem explains neither of them:

- `Ctrl-S` can't write the file. XSchem asks `save file?`, you answer **Yes**, and it replies with a box saying **`file opening for write failed!`**. The tab keeps its `*`, the file on disk doesn't change, and nothing is printed in the terminal.
- **Netlist** writes the netlist into the same directory as the schematic, so it tries to create a `simulation/` folder inside the PDK and stops with this box:

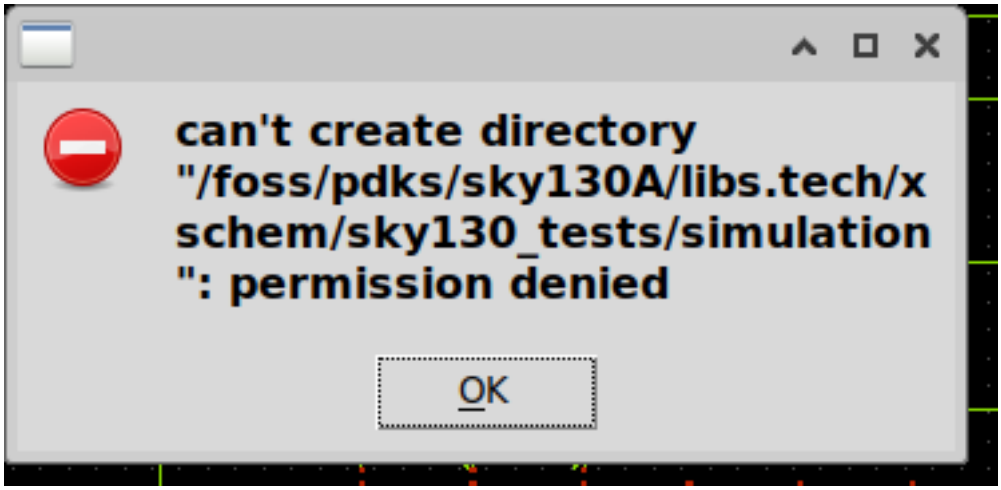


Figure 2.3: XSchem's error box: can't create directory "/foss/pdks/sky130A/libs.tech/xschem/sky130_tests/simulation": permission denied

```
can't create directory
"/foss/pdks/sky130A/libs.tech/xschem/sky130_tests/simulation": permission denied
```

Nothing is wrong with your installation, the PDK or your drawing; you're simply working in a read-only folder. Open a schematic that lives in your own directory and the error goes away. If you've already drawn something on the start page, **File → Save as** into your own folder will keep it.

Units on `W` and `L`

On a SKY130 transistor, **`W` and `L` are plain numbers in microns, with no unit suffix.** `W=1` means one micrometre. Adding a `u` is the most common SKY130 mistake there is, and nearly everyone makes it at least once.

It's an easy mistake to make, because everywhere else in SPICE you would write `1u` for a micron, so `W=1u` looks right. The SKY130 models already treat the number as microns, though, so `W=1u` asks for a millionth of a micron. XSchem accepts this without a murmur. Here is the device line it writes when both `W` and `L` have the suffix:

```
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15u W=1u nf=1 ad=2.9e-07 as=2.9e-07 pd=0.580002
ps=0.580002 nrd=290000 nrs=290000 sa=0 sb=0
+ sd=0 mult=1
```

The symbol calculates the diffusion areas and resistances from `W`, so the mistake spreads to other parameters: `nrd` went from `0.29` to `290000`, and `ad` from `0.29` to `2.9e-07`. Those two `u`s changed six other numbers, and XSchem didn't mention any of it.

It's ngspice that finally objects:

```
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!
```

`could not find a valid modelname` almost always means a unit suffix on `W` or `L`. The SKY130 models are **binned**: each `.model` card covers a range of widths and lengths, and a device has to fall inside one of those ranges to get a model. A width of a millionth of a micron is outside every bin, so no model matches and ngspice stops. The message mentions neither `W` nor units, so when you see it, check `W` and `L` first instead of trying to work out what the message means. Reading a SKY130 device model explains binning in full.

In Lab 01, Part 6 we have you cause this error on purpose, so that you'll recognise it when it happens by accident.

How wires connect

In XSchem, a wire connects to something only where one of its **endpoints** lands on a pin or on another wire. Two wires that cross each other are not connected. XSchem draws a small solid dot at a real junction, and nothing at all where two wires just pass over each other:

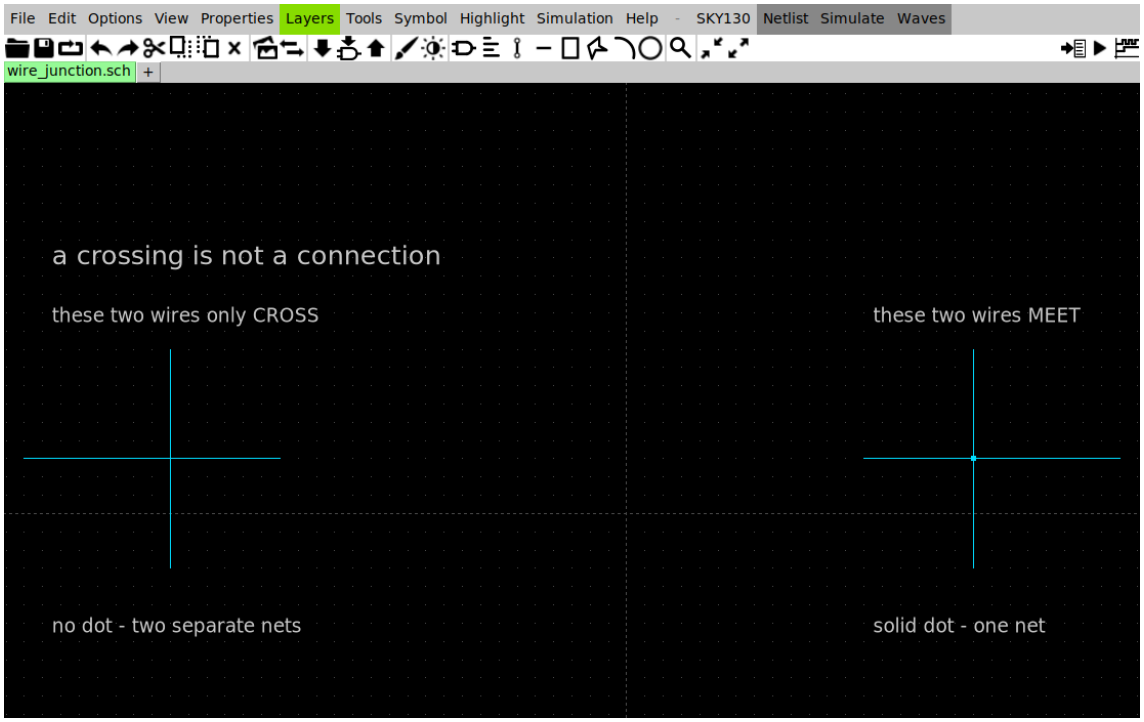


Figure 2.4: Two pairs of XSchem wires side by side: on the left they cross with nothing drawn at the crossing, on the right they meet at a small solid dot

The same pair of wires drawn twice. On the left, one wire runs over the other and XSchem draws nothing at the crossing: these are two separate nets that happen to overlap on the screen. On the right, four wire ends land on the same grid point and XSchem draws a dot, so they form one net. The dot is the only visible difference, and it's only a few pixels across, so zoom in on every junction before you netlist.

XSchem also snaps every click to a grid, so a wire can end one grid step short of a pin, and at normal zoom it will still look connected. Lab 01 shows you how to land wire ends on pins with `Shift-W`, and the netlist check later on this page finds the gaps you miss.

For long connections you don't need to draw a wire at all. Instead you **name the net**: put a net label (`lab_pin.sym`) at each end and give both labels the same name. Any two labels with the same name are the same net, wherever they sit on the page, and every supply and ground connection in this course is made this way.

Here is the reference schematic from Lab 01, which is the first one you'll netlist:

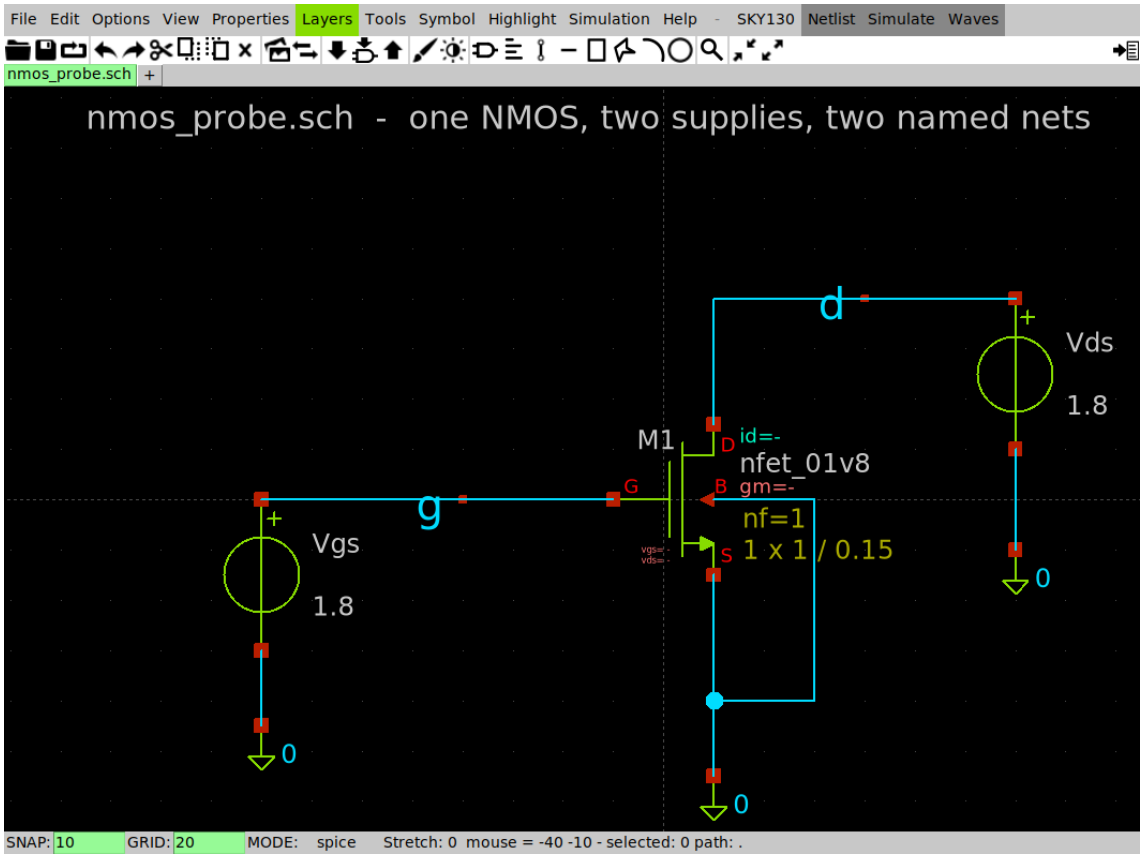


Figure 2.5: nmos_probe.sch: one NMOS, two supplies, two named nets

One `nfet_01v8` with `W=1` and `L=0.15`. `Vgs` drives the gate through a net labelled `g`, and `Vds` drives the drain through `d`. The source and body both go to ground, and the filled cyan dot where they meet is a real junction. The status bar reads `SNAP: 10 GRID: 20 MODE: spice`; the snap setting is why wires land exactly on pins instead of next to them.

XSchem labels each device with its size, here `nfet_01v8`, `nf=1` and `1 x 1 / 0.15`. The last one means `mult × W / L`, and it lets you check the sizes on a whole page of transistors at a glance.

The netlist

When you click **Netlist**, XSchem writes a SPICE file and tells you almost nothing about it, so it's worth knowing where the file goes and what's in it.

Where it goes. By default the netlist is written to `~/.xschem/simulations/`, a folder inside the container, so it's deleted along with the container even though your schematic is safe on your own disk. The setting **Simulation → Use 'simulation' dir in schematic dir** changes this, so that netlists go into a `simulation/` folder next to the `.sch` file. Each lab's `xschemrc` turns it on with `set local_netlist_dir 1`; a scratch directory of your own won't have that line.

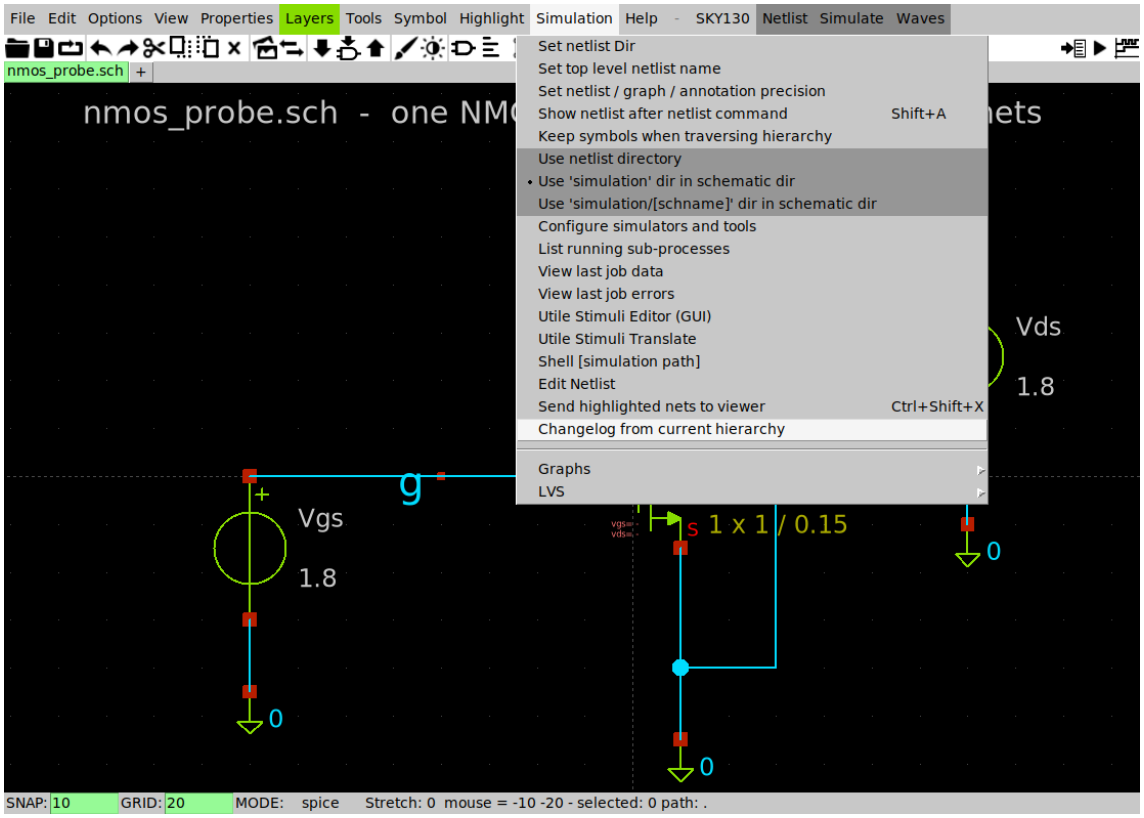


Figure 2.6: The XSchem Simulation menu open, with the radio bullet on "Use 'simulation' dir in schematic dir"

The three netlist-directory entries are radio buttons, so exactly one of them is on. Here it's the middle one, which is what `set local_netlist_dir 1` *in a lab's* `xschemrc` *selects.*

What's in it. This is the netlist of `nmos_probe.sch`:

```
**      sch_path:      /foss/designs/modules/ad103/labs/lab-01-first-schematic/xschem/
nmos_probe.sch
**.subckt nmos_probe
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad=0.29 as=0.29 pd=2.58 ps=2.58
nrd=0.29 nrs=0.29 sa=0 sb=0 sd=0 mult=1
Vgs g 0 1.8
Vds d 0 1.8
**.ends
.end
```

Your whole drawing is in those seven lines. `XM1 d g 0 0` is the transistor, followed by its four terminals in a fixed order: **drain, gate, source, body**. The names are the labels you placed, and `0` is ground, which SPICE has always called node zero.

Notice that there are no device models and no analysis command. A netlist describes only the circuit. The `.lib` line that loads the SKY130 models and the `.control` block that says what to measure belong in the testbench, and you'll meet your first testbench in Lab 02.

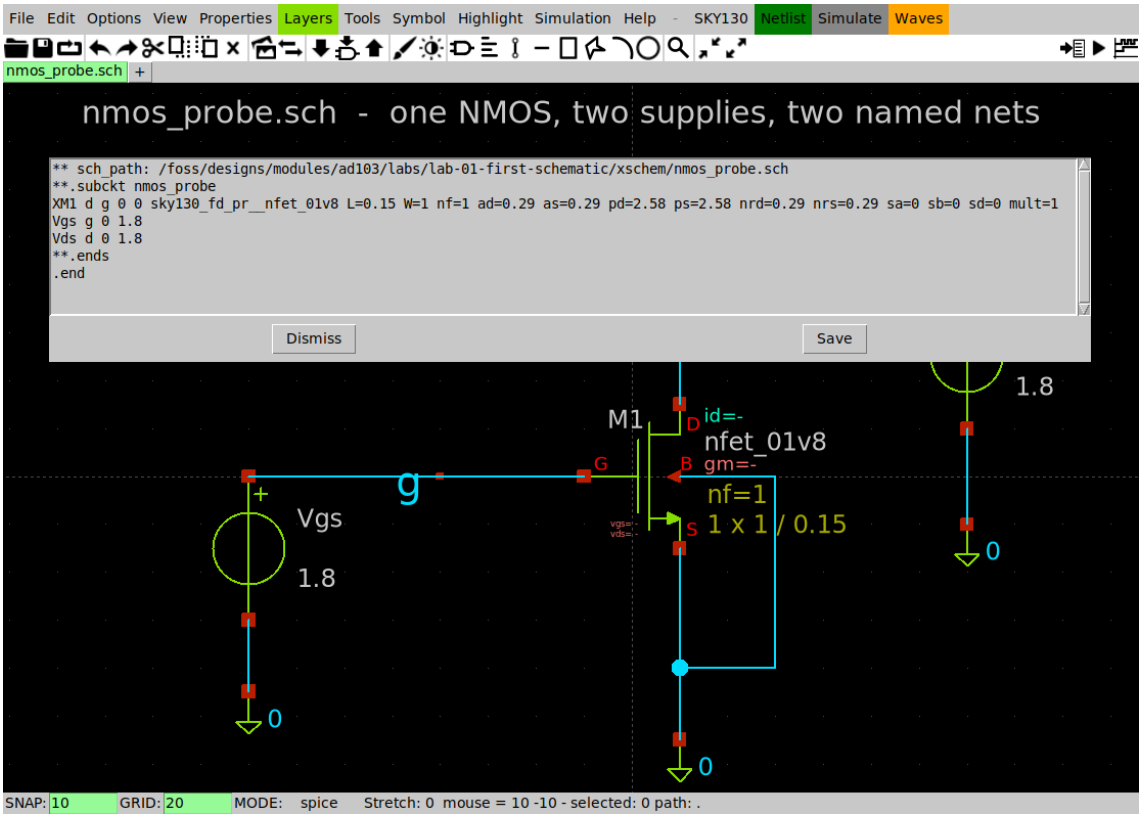


Figure 2.7: XSchem showing the generated netlist in its own window, over the schematic it came from

*With **Simulation → Show netlist after netlist command** (`Shift-A`) ticked, each press of **Netlist** opens the file in its own window, with the schematic still visible behind it.*

Invented net names

Read the four node names on the device line; they should be exactly the nets you labelled. If the gate wire stops one grid step short of the gate pin, XSchem writes this instead, with no error, no warning and an exit status of 0:

```
XM1 d net1 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ...
```

The gate's net `g` has become `net1`. XSchem invents a name for every pin that isn't connected to anything, and all the invented names look like `net` followed by a number. So after every netlist, search for them with `grep 'net[0-9]' simulation/*.spice`. Any match is a wire you think is connected but isn't. This is quicker and more reliable than inspecting the canvas by eye, and it finds nine out of ten wiring mistakes in this course. Lab 01, Part 7 shows you what ngspice makes of a circuit like this.

Simulation time

Every SPICE deck in this course loads the SKY130 models with this line:

```
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
```

The `tt` at the end picks the typical corner. Loading the model library is slow: a single-deck run such as Lab 01's `make` takes about ten seconds, and the terminal shows nothing while ngspice reads the models. It hasn't hung, so don't press `Ctrl-C`.

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. In an EDA tool, much of what looks like your own mistake is really the tool being unhelpful, and XSchem in particular tends to fail without saying anything.
- **Quote the exact error, and say where you saw it.** Some of XSchem's messages go to the terminal it was launched from, and others, such as the two error boxes on this page, appear only in the XSchem window, with nothing in the terminal. Either way, paste the text, along with the command you ran and the output of `xschem --version`.
- **Post a screenshot of the schematic.** A wiring problem that's obvious in a screenshot can be very hard to describe in words.
- **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 and copied every number from a real run, so a mismatch means something has changed, and we want to know.

Check yourself

Answer these on paper before you start Lab 01.

1. XSchem's menu bar says **IHP**. What's wrong, and which two checks tell you it's fixed?
2. You typed `W=2u` on a transistor. Which tool complains, and what does the message say?
3. Two wires cross on the canvas, with no dot at the crossing. Are they connected?
4. A device line reads `XM1 d net3 0 0 ...`. Which terminal is unconnected?
5. You netlisted a schematic in a folder with no `xschemrc`. Where did the netlist go, and what happens to it when the container is removed?

Next: Lab 01: Your first schematic.

3. Lab 01: Your first schematic

You'll load the course, draw one transistor in XSchem, watch it become seven lines of SPICE, and get a current out of it. Then you'll break the same circuit twice on purpose. The first mistake is loud enough that ngspice refuses to run, and the second is so quiet that ngspice hands you a wrong answer without any error.

Package: `labs/lab-01-first-schematic/`

Before you start

- Read Getting started.
- Keep the XSchem cheat sheet open in another tab.

Time: about 2 hours, most of it learning XSchem, not counting the optional Part 8. Each `make` target takes about ten seconds, and almost all of that is ngspice loading the SKY130 models.

Part 1: Load the course

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

1. Start the desktop from your `workspace` clone (`./scripts/start_vnc.sh`, or `scripts\start_vnc.bat` on Windows), open **http://localhost/**, and open a terminal on that desktop. Every command from here on goes in that terminal, not on your laptop.
2. Load the course environment. Do this in every new terminal:

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

   ```
   ASIC-EDU: PDK=sky130A  PDK_ROOT=/foss/pdks  designs=/foss/designs
             run 'mod' to list course modules, or 'mod <name>' to enter one.
   ```

   Check that the banner says `PDK=sky130A` (`echo $PDK` prints the same thing). The image's own default is `ihp-sg13g2`, which is the wrong process for this course, so this is the line that tells you the script worked.
3. Get the course files:

   ```
   mod add ad103      # first time only: clones github.com/uoftasic/ad103
   mod ad103          # every time after that
   ```

   ```
   Cloning https://github.com/uoftasic/ad103.git
        into /foss/designs/modules/ad103
   Cloning into '/foss/designs/modules/ad103'...
   OK  module ready: /foss/designs/modules/ad103
       run: mod ad103
   cwd: /foss/designs/modules/ad103
   ```

What you see	What it means
`bash: mod: command not found`	You did not run step 2 in this terminal.
`mod: no such module: ad103`	You ran `mod ad103` before `mod add ad103`.

What you see	What it means
`fatal: could not read Username for 'https://github.com'`	git couldn't read the repository without logging in. Check the spelling of the module name, and if it's right, ask in Discord. Don't type your password.

4. Check the tool versions:

```
xschem --version
ngspice -v
```

You should see `PDK=sky130A` in the banner from step 2, then `XSCHEM V3.4.8RC` and `ngspice-47` at the top of the two version outputs. The course is pinned to these versions because XSchem's keys and menus have changed between releases, and with a different version the screenshots may not match your window. For problems with the desktop or the container, see IC101's Troubleshooting page.

Part 2: Prove the toolchain works

Does your workbench give the reference answer, and what does a SPICE device line say?

1. Run the reference check:

```
cd /foss/designs/modules/ad103/labs/lab-01-first-schematic
make
```

```
== netlisting xschem/nmos_probe.sch
   wrote xschem/simulation/nmos_probe.spice
== simulating spice/nmos_op.spice
   wrote results/nmos_op.log
== checking
  device line : XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad=0.29 as=0.29
pd=2.58 ps=2.58 nrd=0.29 nrs=0.29 sa=0 sb=0 sd=0 mult=1
  drain current : 501.046 uA  (reference 501.046 uA)

PASS  netlist and operating point match the reference run
```

2. Read the device line first, before you look at the current. `make` netlists `xschem/nmos_probe.sch` and prints the device line from that netlist, but the drain current comes from simulating a separate, hand-written deck, `spice/nmos_op.spice`. The four node names, `d g 0 0`, are in the order **drain, gate, source, body**, and you'll find each of them on the drawing in Part 3.
3. Open `results/nmos_op.log` and find `i(vds) = -5.01046e-04`. On some builds of the image the log opens with several `Error opening osdi lib` lines; they are harmless, and Harmless messages says why.

   The transistor is conducting, so why is the current negative?

   Answer: A1, in Answers at the end.

You should see `PASS` and **501.046 µA**. That is a real SKY130 NMOS, one micrometre wide and 150 nanometres long, with 1.8 V on both the gate and the drain: half a milliamp through a device you could fit two thousand of across the width of a human hair. Lab 03 explains why it carries that particular current.

`make` doesn't need any environment setup, because the `Makefile` sets `PDK=sky130A` itself, so it works in a bare container even if Part 1 went wrong. That makes it a useful check for the rest of the course: if a page seems to disagree with your screen, come back here and run `make`. If it prints `PASS`, your toolchain is fine and the problem is in your drawing.

Part 3: Look around the reference schematic

What does XSchem write when you press **Netlist**, and where does each line come from on the drawing?

1. Open the reference schematic from its own folder, so XSchem reads that folder's `xschemrc`:

```
cd xschem
xschem nmos_probe.sch &
```

2. Check the menu bar says **SKY130**. If it says **IHP**, close XSchem, run step 2 of Part 1 in this terminal, and launch it again. Other ways XSchem can fail, most of them without any error, are listed under Failures that exit 0.

3. Learn the keys you need. XSchem has 154 key bindings, but the ones in this table cover everything in the course. They're taken from XSchem's own list, which you can open with **Help → Keys**, or with `/` for a full-screen poster.

Do this	Key
Zoom to fit	`f`
Zoom in / out	mouse wheel, or `Shift-Z` / `Ctrl-Z`
Pan	hold `Space` and drag, or drag with the middle button
Insert a symbol	`Shift-I` (or `Ins`)
Draw a wire	`w`: press it at one end, then click the other
Draw a wire whose **ends** snap onto the nearest pin	`Shift-W`, and hold `Shift` for the closing click too
Place a net label	`Alt-L`
Edit the selected thing's properties	`q`
Duplicate / move / delete selection	`c` / `m` / `Delete`
Undo	`u`
Save	`Ctrl-S`
Abort whatever you started, and unselect	`Escape`
Help	`?`

   Three of these will catch you out, because your text editor uses the same keys for something else. `c` duplicates the selection (plain `Ctrl-C` is a different, less useful copy), `u` is undo (`Ctrl-Z` zooms out), and `q` opens the properties editor rather than quitting. The full list, with menu equivalents, is in the XSchem cheat sheet.

4. Pan and zoom around `nmos_probe.sch` until it stops looking random. Find the two supplies, the nets `g` and `d`, and the junction dot where the source and body wires meet. Compare it with the picture in Getting started.

   > **Warning:** `nmos_probe.sch` is the lab's reference, and `make` checks against it. Practise on it freely (`u` undoes any number of steps), but don't save over it. If you already did, run `git checkout -- xschem/nmos_probe.sch` from the lab folder to put it back.

   The status bar at the bottom of the window is where XSchem tells you what mode you're in. If it's hidden behind the taskbar, see Windows that open off screen.

5. Open the **Simulation** menu and check the bullet is on **Use 'simulation' dir in schematic dir**. The lab's `xschemrc` sets this, so it should already be on. While the menu is open, tick **Show netlist after netlist command** (`Shift-A`).

6. Click **Netlist** in the menu bar; it's the grey button to the right of `SKY130`. The netlist pops up in its own window.

7. In the terminal, still in the `xschem/` folder, print the file XSchem wrote:

```
cat simulation/nmos_probe.spice
```

```
**    sch_path:    /foss/designs/modules/ad103/labs/lab-01-first-schematic/xschem/
nmos_probe.sch
**.subckt nmos_probe
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad=0.29 as=0.29 pd=2.58 ps=2.58
nrd=0.29 nrs=0.29 sa=0 sb=0 sd=0 mult=1
Vgs g 0 1.8
Vds d 0 1.8
**.ends
.end
```

You should see the seven-line netlist above, with `d g 0 0` on the device line matching the nets on the drawing.

Part 4: Draw your own transistor

Can you draw the reference circuit yourself, with a transistor twice as wide? You start from `xschem/my_probe.sch`, which is the same circuit with the transistor left out.

1. Go back to the lab folder and open it:

```
cd ..
make edit
```

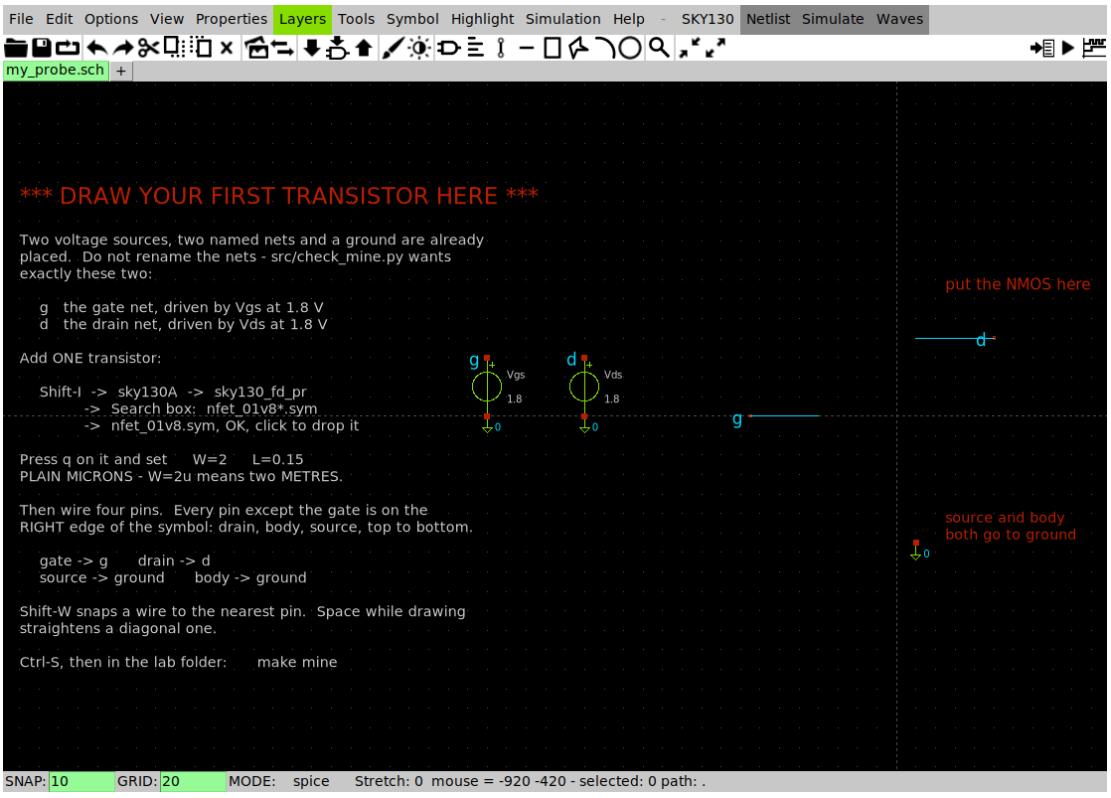


Figure 3.1: xschem/my_probe.sch — the two supplies, the two nets, and a gap where the transistor goes

Everything except the transistor is already placed and wired. `Vgs` and `Vds` are both 1.8 V, and they're connected by net name rather than by wire: any two `lab_pin`s with the same label are the same net, wherever they sit on the page. The status bar reads `SNAP: 10  GRID: 20  MODE: spice`.

2. Press `Shift-I`. The symbol chooser opens:

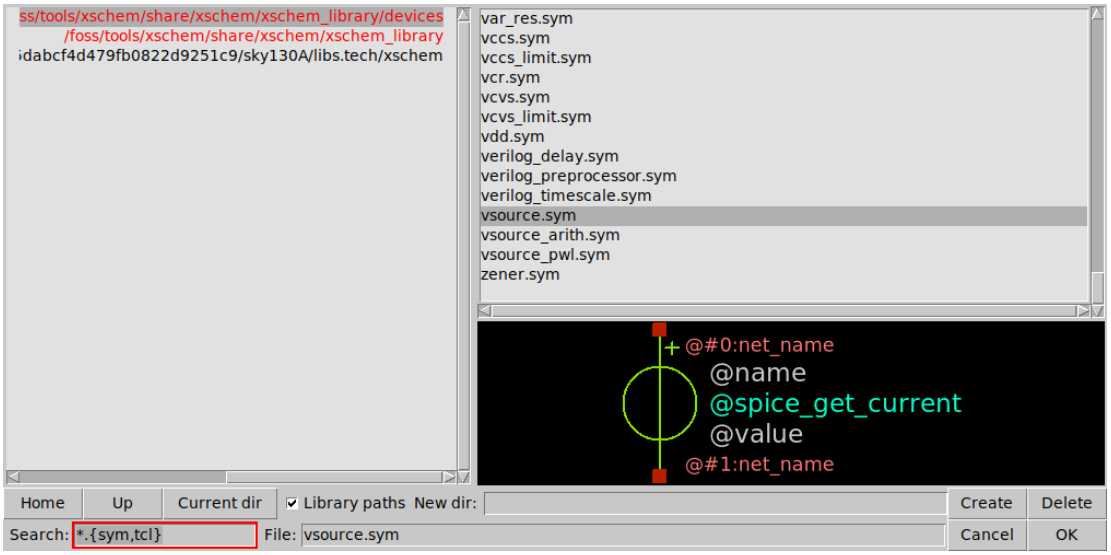


Figure 3.2: XSchem's symbol chooser, opened with Shift-I

*The left pane lists the symbol libraries XSchem knows about, and the right pane shows the current directory. **Search** filters the file list by a pattern. When you pick a file (`vsource.sym` here), the pane underneath draws the symbol, so you can check you have the right one before you place it. Press **OK**, or double-click the file, and the symbol attaches to your cursor; click to drop it.*

Look at the left pane first, because it gives you a second check on the PDK. Three paths are listed, and the third is `…/sky130A/libs.tech/xschem`. If that line is missing or says `ihp-sg13g2`, close XSchem and go back to Part 1, step 2.

The right pane opens on XSchem's built-in **`devices`** library. You need four things from it during the course: `vsource.sym` (a voltage source), `gnd.sym` (ground), `lab_pin.sym` (a net label), and `capa.sym` (a capacitor).

On a desktop 1280 pixels wide, the chooser opens mostly off the right edge of the screen, with **OK** out of view. Drag it back by the part of the title bar you can see; Windows that open off screen has the details.

3. Click the `sky130A` line on the left, open **`sky130_fd_pr`**, and replace the **Search** text with `nfet_01v8*.sym`. That cuts the library's 77 symbols down to five. Search takes a glob pattern rather than a substring, so `nfet` on its own matches nothing.

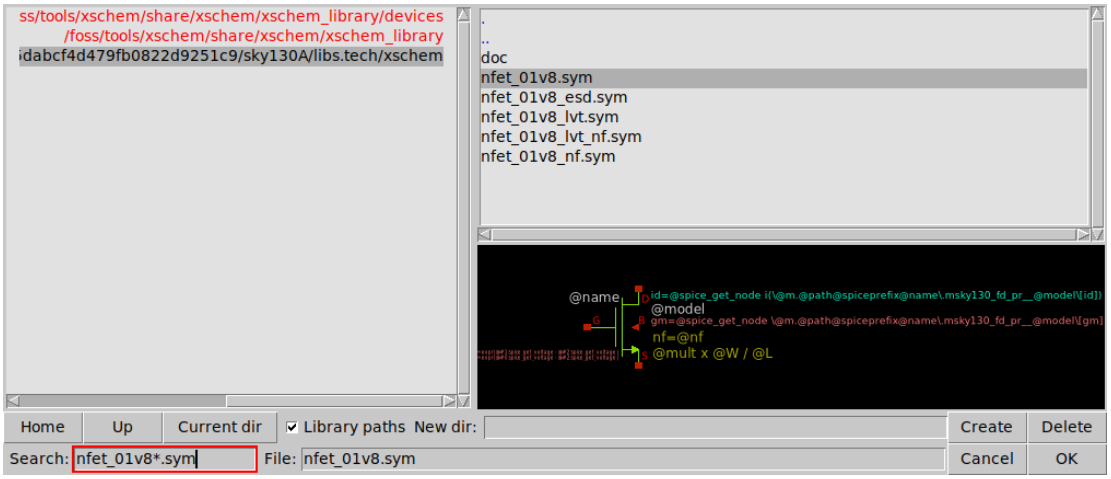


Figure 3.3: The symbol chooser in sky130_fd_pr with the Search box reading nfet_01v8*.sym and five files listed

The search has cut the whole `sky130_fd_pr` library down to five files. The preview shows the symbol with its attributes still as placeholders (`@name`, `@model`, `@mult x @W / @L`), because they're filled in separately for each instance.

The three symbols you'll use in this course are **`nfet_01v8.sym`** (the 1.8 V n-channel MOSFET, the workhorse of this process), **`pfet_01v8.sym`** (its p-channel twin) and **`diode.sym`** (Lab 02's subject). The other seventy-odd devices are described in Reading a SKY130 device model.

4. Pick `nfet_01v8.sym`, press **OK**, and click to drop it in the gap.
5. Select the transistor and press `q`:

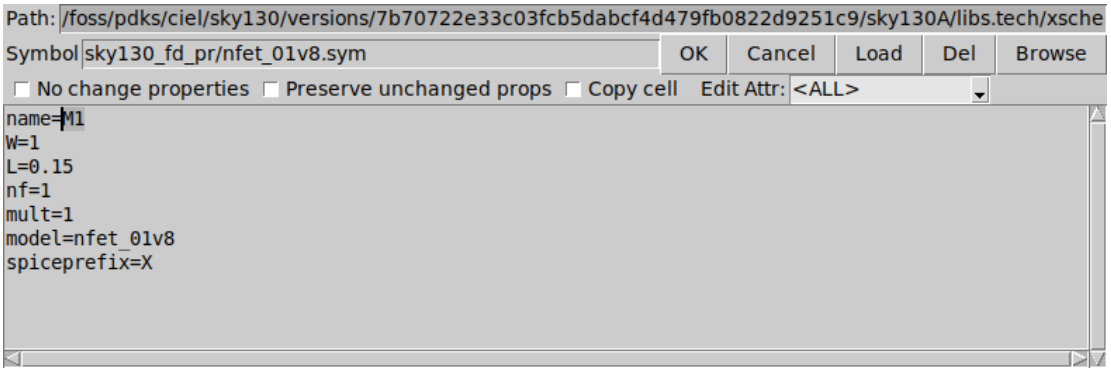


Figure 3.4: The XSchem property editor on an nfet_01v8 instance

The editor shows one attribute per line. `W=1` and `L=0.15` are the only two you'll normally change; `nf`, `mult`, `model` and `spiceprefix` come from the symbol, so leave them alone.

Set **`W=2`** and **`L=0.15`**, as plain numbers with no `u` after them.

6. Wire the four pins: drain to `d`, gate to `g`, source and body to ground. To draw a wire, put the pointer on the first end, press `w`, move to the second end, and click. It's the key, not a click, that sets the start of the wire. So if you press `w` and *then* click in two places, you get a wire from wherever the pointer was sitting when you pressed `w` to your first click, and nothing between the two places you clicked. Each wire is one straight segment, so for an L-shape you draw two.

   Two things often cost new users their first half hour:

- **Your wires may come out diagonal.** XSchem's routing mode has an oblique setting, and it isn't always switched off. Press `Space` while a wire is in progress to cycle between horizontal-then-vertical, vertical-then-horizontal, and straight diagonal. `Escape` throws away a wire you started by mistake.

- **A wire end can miss a pin by one grid step.** If you're not sure a wire will land on a pin, use `Shift-W` instead of `w`, and hold `Shift` down for the closing click as well. `Shift-W` snaps the end you *start* from onto the nearest pin or net endpoint, however far away it is. Unless you hold `Shift`, the click that finishes the wire is an ordinary click: it lands on the snap grid, and it only pulls onto a pin that's already within about one snap step of it, which is why the mistake comes and goes. We measured it on a pin 28 units away: a plain closing click landed on the grid 28 units short of the pin, and with `Shift` held it landed exactly on it.

  Zoom in on every junction and check for the dot.

7. Save with `Ctrl-S`. XSchem always answers with a small window titled **Ask Save**, which asks `save file?` and offers **Yes** / **Cancel** / **No**, and it puts that window wherever the pointer happens to be. Nothing reaches the disk until you press **Yes** (or `Return`), and the tab title keeps its `*` until you do. XSchem has no autosave, so save often.

> **Warning:** if you press `Ctrl-S`, miss the **Ask Save** window, and run `make mine`, you check the *last* version you saved, which is usually the one with no transistor in it. That gives you a confident `FAIL - the netlist has no nfet_01v8`, which sends you hunting for a placement bug that isn't there. Check that the `*` has gone from the tab before you run `make mine`.

You should see an `nfet_01v8` in the gap with `W=2` and `L=0.15`, no hollow square on any wire end or pin, a dot where the source and body wires meet, and no `*` in the tab title.

Part 5: Simulate your drawing

Does a transistor twice as wide carry twice the current?

1. Predict the current before you simulate. The reference device has W = 1 µm and draws 501.046 µA, and yours is twice as wide. Write your number down.
2. Netlist, check and simulate your drawing:

```
make mine
```

   It netlists what you drew and checks the wiring before it simulates anything:

```
  WHAT YOU DREW
     XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=2 nf=1 ad=0.58 as=0.58 pd=4.58
ps=4.58 nrd=0.145 nrs=0.145 sa=0 sb=0 sd=0 mult=1

  Wired correctly. Simulating it.

  YOUR TRANSISTOR
    W = 2 um, L = 0.15 um, V_GS = V_DS = 1.8 V
    drain current    1030.4200 uA
    reference        1030.4200 uA

PASS  you drew a transistor, XSchem turned it into SPICE, and ngspice
      answered 1030.4200 uA. That is the whole chain, and it is the
      chain every other lab in this course runs on.

  One more thing before you leave. The reference schematic is W=1 and
  draws 500.9410 uA. Yours is twice as wide. Twice 500.9410 is
  1001.8820, and you measured 1030.4200 - +2.85 %.
  Small, and not zero, and not noise. Lab 04 is entirely about that gap.
```

   If it says `FAIL`, it tells you which of five things went wrong (a missing device, a unit suffix on `W`, a `net7` where a wire didn't land, the source on the wrong node, or the body left floating) and what to do about it.

3. Compare the result with your prediction. The note at the end compares your current with twice the reference schematic's 500.9410 µA. What does the +2.85 % say about the rule "current is proportional to *W*"?

   Answer: A2, in Answers at the end.

4. The hand-written deck in Part 2 gave 501.046 µA for the reference device, and the XSchem netlist gives 500.941 µA. Both are correct. Compare the device line in Part 2's `make` output with the one in `spice/nmos_op.spice`. What makes them differ by 0.02 %?

   Answer: A3, in Answers at the end.

You should see `PASS` and **1030.4200 µA** from `make mine`.

Part 6: Units on `W` and `L`

What does ngspice say when `W` and `L` carry a unit suffix?

1. Run the deck with units on `W` and `L`:

```
make wrong-units
```

```
Error on line 56 or its substitute:
   m.xm1.msky130_fd_pr__nfet_01v8 d g 0 0 xm1:sky130_fd_pr__nfet_01v8__model l=
1.500000000000000e-07     w=    1.000000000000000e-06 ...
could not find a valid modelname
    Simulation interrupted due to error!
```

2. Find what changed:

```
diff spice/nmos_op.spice spice/nmos_op_wrong_units.spice
```

You should see that the two decks differ by two characters: `L=0.15u W=1u` instead of `L=0.15 W=1`. Getting started explains what the `u` does.

The error message doesn't mention `W` or units. So when you see `could not find a valid modelname`, what should you check first?

Answer: A4, in Answers at the end.

Part 7: A gate wire that stops short

What does ngspice do with a gate wire that stops short of the pin?

1. Predict: will ngspice stop with an error, as it did in Part 6? Write down your guess.
2. Run the schematic with one short wire:

```
make unconnected
```

```
   XSchem exit status: 0.  No error, no warning.  Here is the device line:
     XM1 d net1 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ...

     Note: Starting dynamic gmin stepping
     Warning: Dynamic gmin stepping failed
     Note: Starting true gmin stepping
     Warning: True gmin stepping failed
     Note: Starting source stepping
     Warning: source stepping failed
     Note: Transient op started
     Note: Transient op finished successfully
     i(vds) = -7.34593e-08
     v(net1) = 5.017472e-01
```

3. Open the broken schematic in its own window and compare it with `nmos_probe.sch` (open that too if you closed it after Part 3):

```
cd xschem
xschem broken_probe.sch &
```

   Find the difference. It's a single wire that ends 20 units short of the gate pin, which is about a millimetre on screen.

4. Compare the current with Part 2's 501.046 µA. By what factor is it wrong?

5. `v(net1) = 5.017472e-01` is the gate. `Vgs` is still in the circuit and still set to 1.8 V. Why is the gate at 0.50 V?

   Answer: A5, in Answers at the end.

6. Go back to the lab folder and search the netlists for invented names:

```
cd ..
grep 'net[0-9]' xschem/simulation/*.spice
```

You should see ngspice print an answer, **73.4593 nA**, with no error anywhere, even though the answer is wrong by a factor of **6821**. The `grep` finds the `broken_probe` device line with `net1` on it. Apart from that, the only warning you got was three lines of solver complaints, which look exactly like the noise you learn to scroll past. That's why we ask you to run that `grep` after every netlist for the rest of the course: any match is a wire you believe is connected but isn't.

Part 8 (optional): Your device's model bin

1. Ask ngspice which model card it chose for `L=0.15 W=1`:

```
make bins
```

```
      model xm1:sky130_fd_pr__nfe
       toxe             4.148e-09
       vth0              0.411231
       lmin               1.5e-07
       wmin               8.4e-07
      model xm1:sky130_fd_pr__nfe
          l               1.5e-07
          w                 1e-06
        vth              0.769267
       weff           9.56282e-07
       leff           1.26136e-07
```

You should see two `model` headers. Everything above the second one is the **bin**, and everything below it is **your device**. Reading a SKY130 device model explains each line, including why `leff` is 16 % shorter than the `L` you drew.

Finishing

You're done when `make mine` prints `PASS` and `1030.4200 uA` for the transistor you drew in Part 4.

Questions

Answer these without looking anything up. Each one points to where the lab answered it.

1. How do you set `W` and `L` on an `nfet_01v8`, and what goes wrong if you write `W=1u`? (Part 4, step 5, and Part 6.)
2. How do you draw a wire that isn't diagonal, and make sure it lands on a pin? (Part 4, step 6.)
3. How do you connect two things by net name alone? (Part 4, step 1, and the key table in Part 3.)

4. Where does XSchem write the netlist when you press **Netlist**? (Part 3, steps 5 to 7.)
5. What are the four node names on a device line, in order, and what does a `net7` on it mean? (Part 2, step 2, and Part 7.)

Submit

Nothing to submit.

Next on the Course path: read Linearity and superposition.

Answers

A1. Part 2: Prove the toolchain works

ngspice reports the current flowing *into* the positive terminal of a source. `Vds` is pushing current out of that terminal, so the number comes out negative.

A2. Part 5: Simulate your drawing

Doubling the width gave slightly more than double the current. The difference is too small to be a mistake and too large to be noise, so the rule holds only approximately. This is the first crack in "current is proportional to W", and Lab 04 is about what's behind it.

A3. Part 5: Simulate your drawing

The symbol computes the source and drain diffusion areas and resistances from W (`ad`, `as`, `pd`, `ps`, `nrd`, `nrs`) and writes them onto the device line, while the hand-written deck leaves them at zero. This is the first time in the course that layout shows up in an electrical answer, and here the effect is small.

A4. Part 6: Units on W and L

Check `W` and `L` first, before you try to make sense of the error. The message doesn't point at them, so checking them has to become a habit.

A5. Part 7: A gate wire that stops short

Because the wire stops short, the gate is floating, so its voltage settles wherever the solver happened to leave it. `Vgs` is still there and still drives the net `g`, but `g` no longer reaches the gate.

II

The diode

4 **Linearity and superposition 31**
4 What linear means 31
4 The additivity test 31
4 The diode on linear axes 32
4 What still works 32
4 From procedure to search 33
4 Check yourself 33

5 **The operating point 35**
5 A resistor and a diode in series 35
5 Load lines 35
5 Worked example: guess, correct, repeat 36
5 What ngspice gets 36
5 Two independent calculations 37
5 Small-signal resistance 37
5 Check yourself 37

6 **The diode equation 39**
6 What a junction is 39
6 Why the barrier gives an exponential 39
6 Where 60 mV per decade comes from 40
6 What the SKY130 diode measures 40
6 Reading a log axis 41
6 The ideality factor 41
6 Where the straight line ends 42
6 Driving a diode with a current 43
6 Check yourself 43

7 **Three diode models 45**
7 Model 1: the ideal switch 45
7 Model 2: the constant drop 45
7 Model 3: the exponential 45
7 What each model costs 45
7 Which one to use 46
7 The small-signal model 46
7 What junction area buys 46
7 Check yourself 47

8 **Lab 02: The diode I–V curve 49**
8 Before you start 49
8 Part 1: Run the lab 49
8 Part 2: Test superposition 50
8 Part 3: Read the curve on two axes 51
8 Part 4: Extract n and r_s, and check the model card 53
8 Part 5: Find the operating point 53
8 Part 6: The reverse floor 55
8 Part 7: Junction area and size mistakes 55
8 Part 8: Read a FAIL you caused 57
8 Part 9: Heat the diode 57
8 Finishing 58
8 Going further 58
8 Answers 59

4. Linearity and superposition

Between ECE110 and AD102 you learned methods that solve any circuit you were shown. This page explains why adding a single diode breaks those methods, and what still works once it does.

Superposition, Thévenin equivalents, node-voltage analysis with a matrix and the impedance algebra you do in your head are **theorems** rather than facts about wires, and every one of them rests on the same hypothesis: *every element in this circuit is linear.* You have never had to check that hypothesis, because until now it has always been true. When it fails, the theorem doesn't apply at all, and the answer it gives you is wrong rather than approximately right.

What linear means

An element is **linear** if the relationship between the voltage across it and the current through it keeps two promises:

- **Scaling.** If you double the voltage, the current doubles too.
- **Additivity.** The response to $v_1 + v_2$ is the response to v_1 plus the response to v_2.

A resistor, including the poly-silicon strips you sized in AD102, keeps both, because $i = v/R$ and division distributes over addition. Capacitors and inductors keep both too, because differentiation and integration are linear operations. Superposition is just additivity applied to a whole circuit at once.

That gives you a direct test for any component: drive it with A, then with B, then with A+B, and see whether the third answer is the sum of the first two.

The additivity test

Lab 02's `spice/straight_line.spice` runs exactly that test. It builds six one-ports, three copies each of a 100 kΩ resistor and a 1 µm² SKY130 diode, and drives the three copies of each part at 0.35 V, at 0.35 V again, and at 0.70 V, which is the sum of the first two. These are its results:

	A (0.35 V)	B (0.35 V)	A + B, predicted	A + B, measured at 0.70 V
100 kΩ resistor	3.500000 µA	3.500000 µA	7.000000 µA	**7.000000 µA**
1 µm² diode	125.4090 pA	125.4090 pA	250.8180 pA	**3.917437 µA**

For the resistor, 3.500000 µA + 3.500000 µA = 7.000000 µA, and the measured answer is 7.000000 µA, to seven digits, which is what you'd expect from a theorem whose hypothesis holds.

For the diode, superposition predicts **250.8180 pA**, but the circuit actually delivers **3.917437 µA**, which is out by a factor of **15,618.6**.

Nothing about the wrong answer looks broken. It's a clean, plausible-looking number with seven significant figures and no noise on it, and nothing in the output says "superposition does not apply", so it's up to you to know when the theorem holds and when it doesn't.

> **Check before you reach for a linear tool.** Before you write down a Thévenin equivalent, a superposition sum or an impedance divider, ask whether every element in the loop is linear. If the loop has even one diode, one transistor or anything else with a curve in it, the answer is no.

You'll run this test yourself in Lab 02, Part 2.

The diode on linear axes

Here is the same diode swept from −1 V to +0.9 V and plotted the way you have plotted everything so far, as current against voltage with both axes linear.

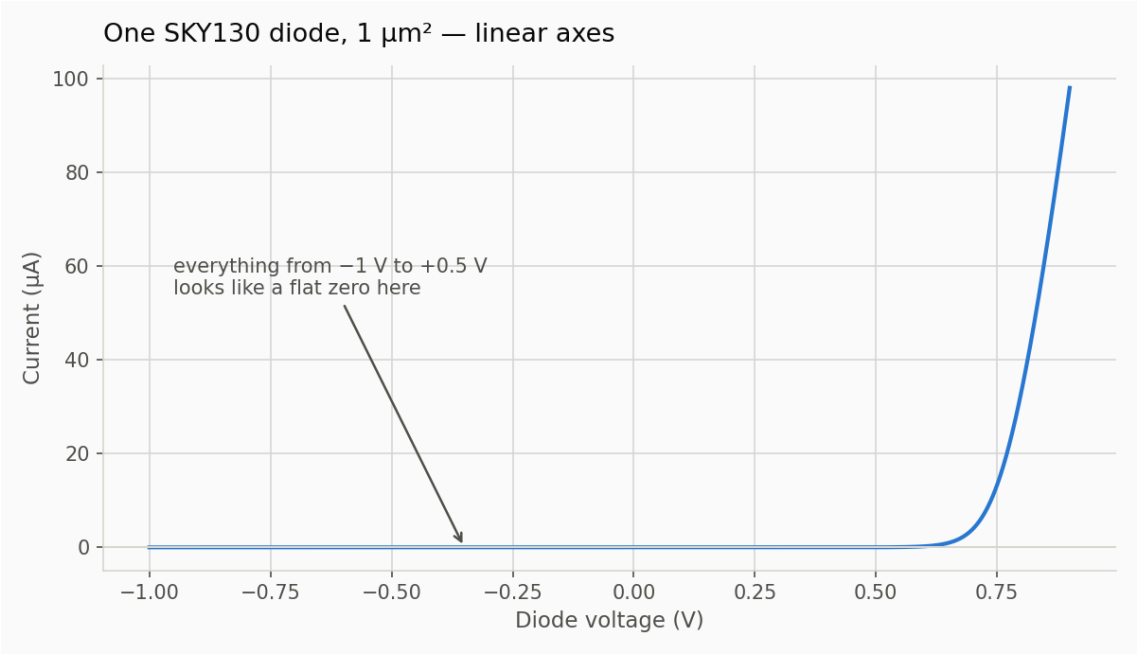


Figure 4.1: Current against voltage for a 1 µm² SKY130 diode on linear axes: a flat line until about 0.6 V, then a near-vertical wall

Produced by `src/plot_iv.py` in the Lab 02 package, from the 1901 rows that `spice/diode_iv.spice` writes.

On these axes a resistor is a straight line through the origin with slope $1/R$. The diode is instead a flat stretch followed by a wall. The slope is the conductance, so this component's resistance is anywhere between "infinite" and "almost zero" depending on where you are on the curve, which means R is no longer a property of the part.

The current along the flat stretch isn't zero, though. At −1.000 V the sweep reads **−1.003559 pA**, and at +0.350 V it reads **125.4090 pA**, which is 125 times more current, yet on this plot both are indistinguishable from the axis. Linear axes hide the part of the curve that carries most of the physics, and The diode equation shows it on a better axis.

What still works

Kirchhoff's laws still hold, because KCL is charge conservation and KVL is energy conservation. Neither law says anything about what the elements are made of, so it makes no difference that one of them is a diode, and you can still write every node equation you know how to write.

Tool	Still valid with a diode in the loop?
KCL, KVL	**Yes.** They are conservation laws, not linearity results.
Ohm's law on the *resistors*	**Yes.** The resistors are still resistors.
Superposition	**No.** It was off by 15,618.6× in the test above.
Thévenin / Norton equivalent	**No** for the part of the circuit containing the diode. It still works for the linear part around the diode, and that turns out to be useful.
Series/parallel combination	**No** for the diode, because it has no R to combine.

Tool	Still valid with a diode in the loop?
Solving the system by algebra	**Usually no.** You get an equation with v both inside and outside an exponential, and no rearrangement gets v on its own.

The last row is the hard one, and it's a problem with the equations rather than with diodes. The next page deals with it.

From procedure to search

Until now you have treated circuit analysis as a procedure: identify the topology, apply the right theorem, turn the crank and get a number. For linear circuits that's the right way to work, and it will serve you for the rest of your career.

With a nonlinear element in the circuit, analysis becomes a **search**. Instead of asking "what is the answer?", you ask "which pair of numbers, one voltage and one current, satisfies both the element and the rest of the circuit at the same time?" Instead of a crank to turn, you have a target, and you need a way of closing in on it.

Check yourself

1. A circuit contains only resistors, capacitors and inductors. Can you use superposition on it? Why?
2. Using the table above, how would you show that a diode fails the scaling promise as well as the additivity one?
3. Why can't you combine a diode in series with a resistor into one equivalent resistance?

Next: The operating point.

5. The operating point

When you can't use superposition on a circuit and you can't solve its equation by hand either, the only way left to get a number out of it is to search for one. You can run that search yourself on paper in about ninety seconds, and it's worth doing once, because it's what every SPICE simulation you launch is doing underneath.

A resistor and a diode in series

The circuit has one supply, one resistor and one diode:

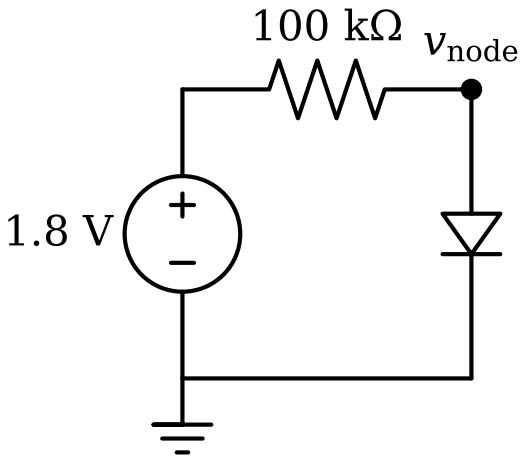


Figure 5.1: The node between the resistor and the diode is the one unknown, v_{node}.

There is only one unknown, `v_node`, and KCL at that node gives you one equation for it: the current flowing down through the resistor has to equal the current through the diode.

$$\frac{1.8 - v}{100\,\text{k}\,\Omega} = I_{\text{diode}}(v) \tag{5.1}$$

The left side is a straight line in v. The right side is the diode's curve, which the next page shows is an exponential in v, so written out in full the equation is

$$\frac{1.8 - v}{100\,000} = (I_0 e^{v/nV_T} - 1) \tag{5.2}$$

with v appearing both outside an exponential and inside one. **No rearrangement of this equation leaves v alone on one side.** That's a property of the equation itself, whatever algebra you try, and mathematicians call an equation like this **transcendental**. The usual way to deal with one is to stop trying for an exact answer and get close to it instead, then closer still.

Load lines

Plot the diode's curve, and then plot the resistor's equation on the same axes. The resistor's equation, $i = (1.8 - v)/R$, is a straight line with an intercept of $1.8/R$ and a slope of $-1/R$. It's called the **load line**, and drawing one is still the fastest way to *see* what a circuit like this will do.

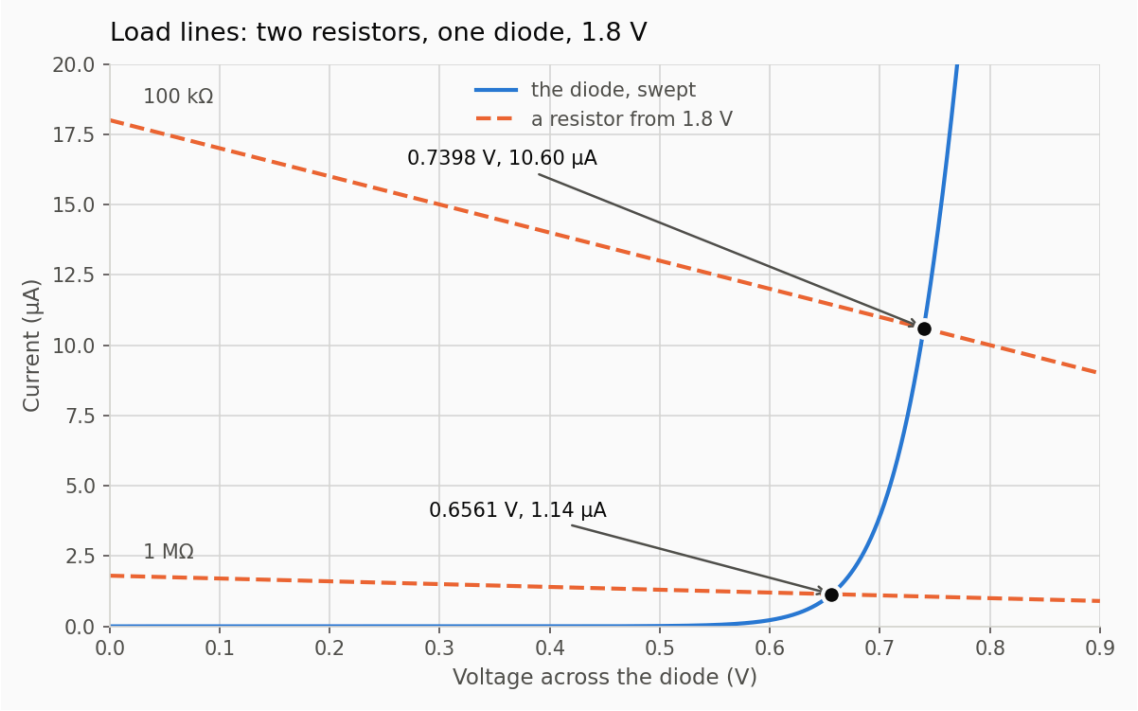


Figure 5.2: The diode's measured curve with two resistor load lines, crossing at marked operating points

Produced by `src/plot_iv.py` in the Lab 02 package. The blue curve is plotted from the same 1901 rows as every other diode plot in this course.

Every point on the blue curve satisfies the diode's equation, and every point on an orange line satisfies the resistor and the supply. The only point that satisfies all of them at once is where the two cross, and that crossing, a single voltage and current together, is the **operating point**.

Two things follow from the picture:

- Changing the resistor moves the operating point **along the diode's curve**; it can't move off it. Going from 1 MΩ to 100 kΩ multiplies the current by 9.27 but moves the voltage by only 83.7 mV, so you get roughly ten times the current for less than a tenth of a volt.
- The diode's curve is so steep near 0.7 V that even a large change in the load line produces only a small change in voltage. That's where the rule of thumb "a diode drops 0.7 V" comes from, and Three diode models puts a number on how good a rule it is.

Worked example: guess, correct, repeat

You can also find the operating point numerically, with a calculator and two facts taken from the measured curve. Start from the rule of thumb, 0.7 V, and use one measured anchor point: the diode carries **3.917437 µA at 0.700 V**, and its current rises by one decade for every **77.04 mV**.

1. **Guess** $v = 0.700$ V. The resistor then delivers $(1.8 - 0.700)/100\,\mathrm{k}\,\Omega = 11.000\,\mu$ A.
2. **Correct.** A diode carrying 11.000 µA wouldn't sit at 0.700 V; it would sit at $0.700 + 0.07704 \log_{10}(11.000/3.917437) = 0.7345$ V.
3. **Repeat.** At 0.7345 V the resistor delivers only 10.655 µA, which the diode holds at 0.7335 V. That voltage gives 10.665 µA, and 10.665 µA gives 0.7335 V again, so the answer has stopped moving.

After three rounds of arithmetic, the hand answer is **0.7335 V and 10.665 µA**.

What ngspice gets

ngspice's operating-point analysis, `op`, solves the same circuit, and it too works by iterating until the answer stops moving. For three values of the resistor it gives:

R	diode voltage	current
10 kΩ	0.8905358 V	90.94642 µA
100 kΩ	**0.7398473 V**	**10.60153 µA**
1 MΩ	0.6561163 V	1.143884 µA

Your hand answer of 0.7335 V and 10.665 µA was **6.3 mV** and **0.60 %** away from the 100 kΩ row, and you got that close with three lines of arithmetic and one measured anchor point.

The 0.60 % error comes from a real property of the device rather than from your arithmetic. You used a constant 77.04 mV per decade, but above about 0.7 V the diode's curve has already started to bend away from that straight line. Lab 02 measures where the bend starts, **0.699 V**, and the next page explains why it happens. If you redo the three steps with 86 mV per decade instead, which is the local value measured in that region, you land on **0.7373 V and 10.627 µA**. That's 2.6 mV and 0.24 % out, less than half the error.

Two independent calculations

`op` and a DC sweep (`.dc`) are different computations. `op` solves the nonlinear circuit once, by iterating. The diode sweep is 1901 separate solves of a *different* circuit, a lone diode driven by a voltage source with no resistor at all. The only reason the two should agree is that they describe the same device.

That gives you a way to check one against the other. In the sweep data, find the voltage at which the diode's current equals the current on the 100 kΩ load line. Interpolating between the two sweep rows on either side of the crossing gives **0.739846 V and 1.060154e-05 A**, and `op` said 7.398473e-01 V and 1.060153e-05 A. The two calculations share nothing but the device model, and they agree to six digits. When two independent routes to a number agree that closely, you can be confident the number isn't an artifact of either route. You'll make this comparison yourself in Lab 02, Part 5.

Every amplifier, current mirror and logic gate is designed by first choosing an operating point and then arranging for the circuit to sit at it. The back half of this course (transconductance, the CMOS inverter, the current mirror) is about picking that point on purpose, rather than discovering where the circuit ended up afterwards.

Small-signal resistance

If you zoom in far enough on any smooth curve, it looks like a straight line. Near the 1 MΩ operating point, the diode's own sweep gives $\Delta V = 2$ mV between the rows at 0.655 V and 0.657 V, and $\Delta I = 65.98$ nA across the same two rows, so $\Delta V/\Delta I$ is **30.3 kΩ**. That's the slope of V against I, the reciprocal of the slope you see on the plot above, where current is drawn against voltage. A small wiggle riding on that operating point sees the diode as a 30.3 kΩ resistor, which means superposition works again, at least for the wiggle.

That is the idea behind every analog circuit you have used: the circuit is nonlinear enough to set a bias point, and everything small that happens near that point is treated as linear. If you move the operating point, the resistance moves with it. At the 100 kΩ point the same measurement gives 4.12 kΩ, about seven times smaller, while the current there is 9.27 times larger.

For a pure junction those two ratios would be the same. Differentiating the diode equation gives a resistance of nV_T/I, and nV_T for this diode is the 77.04 mV per decade divided by $\ln 10$, or 33.5 mV, so the resistance would fall exactly as fast as the current rises. It falls more slowly because the diode also has 979.1 Ω of series resistance in the measurement, and that part doesn't change with current: $33.5\ \mathrm{mV}/I + 0.98\ \mathrm{k}\ \Omega$ gives 30.2 kΩ and 4.13 kΩ, close to both measured values.

> **A small-signal resistance needs an operating point.** If someone quotes you the resistance of a diode without telling you the current through it, they haven't told you anything, because the resistance changes with the current.

Check yourself

1. In the worked example, why does the iteration converge instead of running away? (Think about the slopes of the two curves at the crossing.)

2. With R = 1 MΩ, the load line is ten times flatter. Without calculating anything, would you expect the operating-point voltage to be higher or lower than with 100 kΩ? Check your answer against the table.
3. The small-signal resistance is 30.3 kΩ at the 1 MΩ point and 4.12 kΩ at the 100 kΩ point. What does that tell you about how steep the diode's I–V curve gets as the current rises?

Next: The diode equation.

6. The diode equation

This page explains why a diode's current is an exponential function of its voltage, and where the "60 millivolts per decade" that everyone quotes comes from.

In short, a diode is a **barrier**, and the current through it is carriers **climbing** that barrier. The number of carriers with enough energy to climb a barrier of height E is proportional to $e^{-E/kT}$, and that fact isn't specific to electronics: the same statistics make a chemical reaction go faster when you heat it, and make the atmosphere thinner with altitude. Whenever you put a barrier in front of a thermal population of particles, you get an exponential. The rest of this page turns that idea into numbers you'll measure in Lab 02.

What a junction is

Take one piece of silicon doped so that it has spare electrons to give away (**n-type**), and one doped so that it has spare vacancies for electrons (**p-type**; the vacancies are called **holes**). When you push them together, two things happen, and they work against each other.

Diffusion. Electrons are plentiful on the n side and scarce on the p side, so they spread across, the way a drop of ink spreads through water, and holes spread the other way. Nothing pushes them; diffusion is just the statistics of many particles moving at random.

The field that builds up. Every electron that leaves the n side leaves behind a positively charged donor atom fixed in the crystal, and arrives on the p side as extra negative charge. So the spreading builds an **electric field** across the junction, pointing from the n side to the p side, and that field pushes the next electron back. The region where the field lives has been emptied of mobile carriers, so it is called the **depletion region**.

The junction settles into equilibrium when the two effects cancel, with diffusion pushing carriers across, the field pushing them back, and no net current flowing. The field corresponds to a step in energy, a **built-in barrier** that an electron has to climb to get from the n side to the p side. Nobody applied this barrier from outside; it formed on its own.

Why the barrier gives an exponential

At any temperature above absolute zero, the carriers don't all sit at the same energy. They are spread over a range, with the number at energy E falling off like $e^{-E/kT}$: many carriers with a little energy, exponentially fewer with a lot. That is the **Boltzmann distribution**, and it is the most reused fact in device physics.

Now apply a forward voltage V: plus on the p side, minus on the n side. That voltage opposes the built-in field, so it **lowers the barrier** by qV. The number of carriers that can now get over is multiplied by

$$e^{qV/kT} \tag{6.1}$$

and the current goes up by the same factor. Two consequences of this describe the whole device:

- **Forward bias is exponential**, because you are lowering a barrier that an exponentially distributed population has to climb.
- **Reverse bias saturates.** A reverse voltage makes the barrier *higher*, which stops the climbers entirely. A second, much smaller current remains: the few minority carriers that thermal energy knocks loose near the junction, which the field then sweeps across. Those carriers fall *down* the barrier rather than climbing it, so a taller barrier doesn't produce more of them. Their supply is set by temperature alone, so this current stays at a tiny constant value, called the **saturation current** I_0.

Together these give the Shockley diode equation:

$$I = I_0\left(e^{V/nV_T} - 1\right) \qquad \text{where} \qquad V_T = \frac{kT}{q} \tag{6.2}$$

V_T is the **thermal voltage**, the voltage whose energy qV_T equals the characteristic thermal energy kT. At the 27 °C that ngspice simulates at, $V_T = 25.8649$ mV. The -1 makes the current exactly zero at $V = 0$, and it is also what produces the reverse saturation floor. The n is an experimental correction called the **ideality factor**, which lands between 1 and 2 for real junctions, and you'll extract it from your own data in Lab 02.

Where 60 mV per decade comes from

Rewrite the forward part in terms of *decades* of current instead of factors of e. To multiply the current by 10 you need $e^{V/nV_T} = 10$, so

$$\Delta V = nV_T \ln 10 \tag{6.3}$$

With $n = 1$ and $V_T = 25.8649$ mV:

$$\Delta V = 25.8649\,\text{mV} \times 2.302585 = 59.556\,\text{mV} \tag{6.4}$$

That is the famous **60 mV/decade**. It comes from kT/q times ln 10, which is to say from thermodynamics and arithmetic, so it doesn't depend on silicon, on doping or on the process. Nothing you can do to a diode makes it steeper than that at room temperature, and heating it makes it shallower. Real junctions are shallower than this ideal by the factor n.

What the SKY130 diode measures

Lab 02 sweeps a 1 µm² SKY130 diode and reports, for each decade of current, the forward voltage needed to reach it:

```
current      forward V     mV since previous decade
   1e-12 A   0.181962 V           -
   1e-11 A   0.264643 V       82.68
   1e-10 A   0.342408 V       77.77
   1e-09 A   0.419503 V       77.09
   1e-08 A   0.496518 V       77.02
   1e-07 A   0.573602 V       77.08
   1e-06 A   0.651480 V       77.88
```

Each of the four decades from 100 pA to 1 µA costs between **77.02 and 77.88 mV**. In other words, the device needs a fixed extra voltage for each factor of ten in current, and that is what "exponential" means when you say it in millivolts. A straight-line fit over 1 nA to 100 nA gives **77.037 mV/decade**.

That exchange rate matters for all of analog design. It says that a small voltage change makes a big current change, about a factor of ten for every 77 mV on this diode, and that even a big current change moves the voltage only a little. You're stuck with that trade, because the rate is fixed by physics. Bias networks, references, and every "why is this circuit so sensitive?" question come back to it.

Reading a log axis

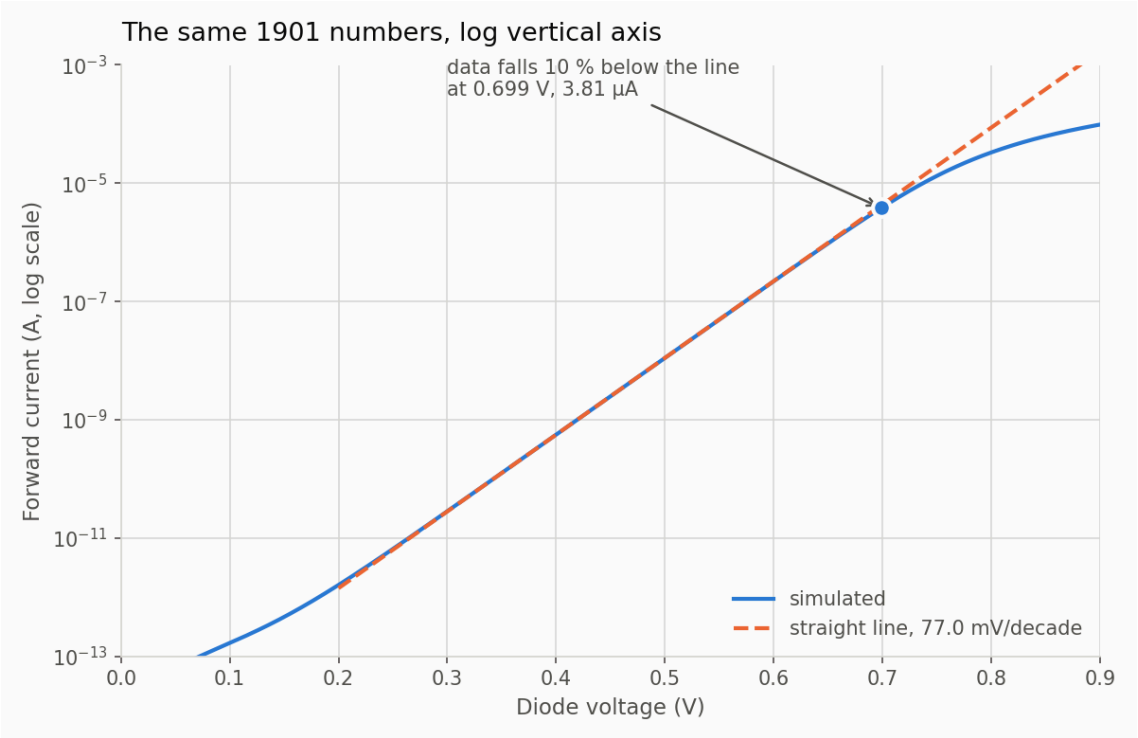


Figure 6.1: The same sweep with a logarithmic vertical axis: six decades of straight line, with a fitted 77.0 mV/decade line

Produced by `src/plot_iv.py` in the Lab 02 package.

This plot uses the same 1901 rows as the flat-and-a-wall plot in Linearity and superposition; only the vertical axis has changed.

A **log axis** labels equal distances with equal **ratios**, not equal differences. The distance from 10^{-9} to 10^{-8} is the same as from 10^{-8} to 10^{-7}, even though the second gap is ten times bigger in amps. That has three consequences, and they apply to every analog plot you'll read:

1. **A straight line on a log-y plot is an exponential**, and only an exponential. Its steepness gives the rate in the exponent, which here is measured in mV per decade.
2. **There is no zero.** Zero is infinitely far down, and a negative number can't be drawn at all, which is why this plot shows only the forward half. To see both sides of zero you need either two plots or the *magnitude* of the current.
3. **Small features at the bottom matter as much as large ones at the top.** The wiggle near 10^{-13} spans as much real information as the wall at 10^{-4}. A linear axis hides six decades of this device, while this one shows all of them.

To read a value, find the gridline below your point, judge how far up towards the next gridline it sits, and multiply the gridline's value by the matching factor. Remember that **halfway up a decade is ×3.16, not ×5.**

The ideality factor

Divide the measured slope by the ideal one:

$$n = \frac{\text{measured mV/decade}}{V_T \ln 10} = \frac{77.0367}{59.556} = 1.2935 \tag{6.5}$$

SKY130's own model card for this diode declares:

```
+ js = 2.75e-015 ; Units: amper/meter^2
+ jsw = 6e-016 ; Units: amper/meter
+ n = 1.2928
+ rs = 981 ; Units: ohm (ohm/meter^2 if area defined)
```

The slope gives **1.2935**, and the card declares **1.2928**, so a foundry parameter pulled out of the slope of a graph agrees with the foundry's own number to five parts in ten thousand. That is what parameter extraction is, and it is most of what a device engineer does: rather than reading a number off a card, you show that you can get it back out of a measurement. You'll find the card yourself in Lab 02, Part 4.

> **Any straight run on a log-current plot gives you an ideality factor.** Its slope in mV/decade divided by 59.6 is n, and a value between 1 and 2 tells you you're looking at a junction. You'll use the same check on a MOSFET in The four regions, where the line comes out at 85.6 mV/decade. The number lands in the same range for a reason: below threshold, the MOSFET's current is also carriers getting over a barrier.

Where the straight line ends

The straight part of the log plot runs from about 0.3 V to about 0.7 V. Outside that range it bends at both ends, and the two bends have different causes: the one at the bottom comes from the simulator, and the one at the top comes from the diode.

The bottom of the curve: GMIN

At the bottom left, the curve flattens into a floor near a picoamp. It's tempting to call that floor the saturation current, but nearly all of it comes from the simulator.

Every SPICE simulator puts a conductance called **GMIN** across every junction, because a branch that conducts nothing makes the matrix it solves singular and the simulation fails to converge. ngspice's default is 10^{-12} S, one picoamp per volt.

Lab 02 measures the current at two reverse voltages and one small forward one, with the default GMIN and again with GMIN set a thousand times smaller:

V	GMIN = 10^{-12} S	GMIN = 10^{-15} S
−1.000 V	−1.00356 pA	−4.55885 fA
−0.500 V	−0.503558 pA	−4.05763 fA
+0.100 V	0.1672571 pA	67.35706 fA

Look at the default column. At −1.000 V the current is 1.00356 pA and at −0.500 V it is 0.503558 pA. The **difference is 0.500002 pA across 0.5 V**, a conductance of 1.000×10^{-12} S to six digits. A diode doesn't have a constant resistance, so this conductance has to come from something other than the diode. It comes from GMIN, which makes up nearly all of the floor.

Subtract GMIN × V from each reverse reading and what's left is constant, which is what a saturation current should be:

V	current	minus GMIN × V
−1.000 V	1.00356 pA	**3.56 fA**
−0.500 V	0.503558 pA	**3.558 fA**
−1.000 V, GMIN = 1e-15	4.55885 fA	**3.559 fA**
−0.500 V, GMIN = 1e-15	4.05763 fA	**3.558 fA**

All four ways give **3.558 fA**. Separately, the straight-line fit to the *forward* curve gives a $V = 0$ intercept of **3.5864 fA**, so two unrelated pieces of arithmetic, done at opposite ends of the plot, agree to 0.8 %. That is I_0, and it's measured in femtoamps: the entire reverse current of this device is about **22,000 electrons per second**.

There's one more check you can make. With GMIN out of the way, the forward point at +0.100 V reads **67.357 fA**. The equation, using only the two numbers extracted from the straight part of the graph, predicts

$$I_0\left(10^{\,V/77.0367\,\mathrm{mV}} - 1\right) = 3.5864\,\mathrm{fA} \times (19.858 - 1) = 67.63\,\mathrm{fA} \tag{6.6}$$

That is **within 0.4 %** of the measurement, and you need the -1 to get there: at 0.1 V the -1 term is still worth 5 % of the answer.

If you report that your diode's reverse leakage is 1 pA, what you've actually reported is ngspice's default setting. The useful exponential range of a real diode is finite at both ends, and the bottom end may be set by your simulator rather than by your circuit.

The top of the curve: series resistance

Above about 0.7 V the measured curve falls **below** the straight line and keeps falling further behind. Lab 02 reports the point where it leaves the line: **at 0.699 V it carries 3.8136 µA, against 4.2487 µA on the line.**

This bend comes from outside the junction, so it doesn't mean the junction is failing. Its cause is the **series resistance** of everything the current has to pass through *besides* the junction: the neutral silicon on either side, the contacts and the metal. You can extract it the same way you extracted n:

```
series resistance, from the top of the curve:
  at 0.900 V the diode carries 98.0354 uA
  the straight line needs only 0.804011 V for that current
  so 95.989 mV is dropped outside the junction
  rs = 95.989 mV / 98.0354 uA = 979.1 ohm
```

The junction itself only needed 0.804011 V, and the other **95.989 mV** was dropped getting the current to it. The model card above says `rs = 981`, so the 979.1 Ω extracted from the shape of the graph agrees with the foundry's number to **0.2 %**.

As the current climbs, the voltage lost to $I \cdot r_s$ climbs *linearly*, while the junction's own requirement climbs only logarithmically, so the resistor term eventually dominates. The local slope shows it taking over (annotations added here):

```
     V        I(A)          mV/decade
 0.450  2.4887e-09        77.01     <- pure junction
 0.600  2.1945e-07        77.50
 0.700  3.9174e-06        85.92     <- rs starting to cost you
 0.800  3.3105e-05       151.64
 0.850  6.2443e-05       217.80     <- mostly a resistor now
```

Driving a diode with a current

Everything above says that a diode's *voltage* is set by its current, rather than the other way round. Thinking of it that way is the hardest habit to build, because for two years you've treated a component as something you put a voltage across.

So you can't say what voltage a diode sits at until you've said what current it carries. If you ask what happens when you put 0.900 V across it, the sweep says it carries **98.0354 µA**, which is **25 times** the current at 0.700 V for only 200 mV more. There is no such thing as slightly too much voltage on a diode, and that is why every diode in a real circuit has something in series with it to limit the current. If you drive it with a current, it settles at a sensible voltage, but if you drive it with a voltage, you had better have the voltage right.

Check yourself

1. Heating a diode from 27 °C to 100 °C raises V_T in proportion to absolute temperature. Does the ideal mV-per-decade figure go up or down?
2. On a log axis, a point sits a quarter of the way from the 10^{-8} gridline to the 10^{-7} one. Roughly what current is it?

3. Why does the GMIN floor make the reverse current look larger than I_0, and how did the second GMIN setting confirm that?
4. The local slope is 77.01 mV/decade at 0.450 V and 151.64 mV/decade at 0.800 V. Which effect is responsible for the change?

Next: Three diode models.

7. Three diode models

You've almost certainly been told that a diode drops 0.7 V, and the last page showed that it's really an exponential, with no fixed voltage at all. You'll use both descriptions in this course, and the point is to choose between them on purpose, once you know what each one costs. A model is a simplification that comes with an error bar, and choosing one is an engineering decision you'll make several times a day.

This page covers the three models you'll use, from the crudest to the exact one.

Model 1: the ideal switch

Forward: a wire, 0 V. Reverse: an open circuit.

This is the model you run in your head while reading a schematic, to answer "which way does the current go?". There's no voltage or current in it, so the only thing it claims to tell you is the **direction** of the current, and it gets that right.

Model 2: the constant drop

Forward: a 0.7 V battery. Reverse: an open circuit.

This is the one everybody quotes, and it works because of the number you saw on the last page, **77.04 mV per decade**. In the decade table, going from 1 nA to 1 µA multiplies the current by a thousand, while the voltage only goes from 0.419503 V to 0.651480 V, a change of 232 mV.

A device whose current can change by three orders of magnitude while its voltage changes by less than a quarter of a volt looks like a battery if you stand far enough back, and Model 2 is the diode seen from that distance.

Model 3: the exponential

$I = I_0\left(e^{V/nV_T} - 1\right)$, *solved together with the rest of the circuit.*

This one is exact, but you can't solve it by rearranging the equation, so it always comes with a numerical method: either your own three lines of iteration or SPICE's.

What each model costs

Take a single resistor R from 1.8 V into a 1 µm² diode. Model 1 puts no voltage across the diode, so it says the current is $1.8/R$. Model 2 takes 0.7 V off the supply and says $(1.8 - 0.7)/R$. Model 3 is ngspice's operating point, from The operating point, and the two error columns compare the other models with it.

R	Model 1: switch	Model 2: 0.7 V	Model 3: ngspice	Model 1 error	Model 2 error
10 kΩ	180.00 µA	110.00 µA	**90.94642 µA**	+97.9 %	+20.9 %
100 kΩ	18.000 µA	11.000 µA	**10.60153 µA**	+69.8 %	+3.8 %
1 MΩ	1.8000 µA	1.1000 µA	**1.143884 µA**	+57.4 %	−3.8 %

Two of Model 2's errors are under 4 %, but "Model 2 is good" is the wrong conclusion to draw from that. Three other things in this table are worth noticing.

Model 2's error changes sign. It's 3.8 % high at 10 µA and 3.8 % low at 1 µA, because 0.7 V is just where this particular junction happens to sit at a few microamps. Above that current the real voltage is higher than 0.7 V, and below it the real voltage is lower. The constant drop is a tangent point rather than a bound: it's right at one current and wrong on either side of it, so it can't tell you which side of the answer you're on. For a diode running at 100 µA, 0.7 V is a bad guess, and for one at 1 nA it's a terrible one, because the decade table puts that diode at 0.419503 V.

Model 2 is worst at the largest current. At that current the diode's series resistance (979.1 Ω) adds a drop of its own on top of the junction's, which Model 2 has no way to include. At 10 kΩ the model is out by a fifth.

Model 1 is at least 57 % wrong at every resistor value in this circuit, and it's still the right model for reading a schematic, because it was never meant to tell you how much current flows, only which way.

Which one to use

You are asking	Use
which way does current flow; is this branch on or off	**Model 1**
roughly what current, roughly what headroom, on the back of an envelope	**Model 2**, and say "about" out loud
what does this actually do; will it still work at 0 °C; how much does it drift	**Model 3**, in a simulator

> **When to stop using 0.7 V.** If a hand calculation used 0.7 V and its answer matters to better than about 10 %, redo it in the simulator. 0.7 V is a one-significant-figure model, and a two-significant-figure result that came out of it isn't worth defending.

The small-signal model

There's also a fourth model, and it's the most useful of all. You replace the diode with a resistor whose value is $\Delta V / \Delta I$ at the operating point (the slope of V against I, which is the reciprocal of the slope of the usual I–V plot), and that resistor is the **small-signal model**. It's valid only for changes small enough that the curve looks straight over them, and it's how every amplifier in this course is analysed. You've already met one of these resistors: 30.3 kΩ at the 1 MΩ operating point, on The operating point.

What junction area buys

In AD102 a resistor's value turned out to be a matter of geometry: so many squares of doped silicon. A diode is drawn the same way, out of the same kinds of shapes, so it's natural to ask what the shape of a diode buys you.

Lab 02 puts four square junctions (1×1, 2×2, 4×4 and 10×10 µm) at the same 0.500 V, and they carry these currents:

area (µm²)	current at 0.500 V
1	11.09698 nA
4	34.57181 nA
16	118.6544 nA
100	667.9656 nA

If current were proportional to area, the 10×10 junction, with a hundred times the area of the 1×1, would carry a hundred times the current. In fact it carries **60.19** times as much.

To see where the rest went, divide each current by its own area and put the result next to the junction's perimeter-to-area ratio:

area (µm²)	perim (µm)	perim / area	current / area (nA/µm²)
1	4	4.0	11.09698
4	8	2.0	8.6429525
16	16	1.0	7.415900

area (µm²)	perim (µm)	perim / area	current / area (nA/µm²)
100	40	0.4	6.679656

Take **only the first and last rows**, leaving the middle two out, and fit a straight line through them:

$$\frac{I}{A} = 6.188848 + 1.2270344 \times \frac{P}{A} \ \text{nA/µm}^2 \tag{7.1}$$

Then use the line to predict the two rows you left out. At $P/A = 2$ it gives **8.642917**, against a measured 8.6429525, and at $P/A = 1$ it gives **7.415882**, against a measured 7.415900. The line matches those two rows to five significant figures, even though it was never fitted to them.

A straight line like that means the diode behaves as two devices in parallel: a **face**, whose current is proportional to area, and an **edge**, whose current is proportional to perimeter. SkyWater's model card describes the diode the same way, with `js` in amps per unit area and `jsw` in amps per unit length, and these four measurements reconstruct both terms without being told either one.

This gives you a fabrication fact worth remembering: **a small junction leaks more per square micron than a big one**, because perimeter shrinks with the linear dimension while area shrinks with its square. For the same reason, two diodes with the same total area don't behave identically if one is a long thin strip and the other is a square.

You can also turn the 60.19× around. At a *fixed current*, the diode with a hundred times the area sits lower by $77.04 \times \log_{10}(60.19) = 137$ mV, so making a junction a hundred times bigger buys you only **about a seventh of a volt**. Area is a very expensive way to move a diode's voltage, and most analog designers learn this once, by drawing something enormous and getting almost nothing for it.

You'll run this measurement yourself in Lab 02, Part 7, along with three ways of getting a diode's size wrong without any warning.

Check yourself

1. A diode carries 100 µA. Is Model 2's 0.7 V likely to be too high or too low? Use the decade table to justify your answer.
2. Using the fitted line above, what would the current per µm² be for a junction with $P/A = 0.1$?
3. You need to lower a diode's voltage by 77 mV at fixed current. Roughly how much bigger must the junction be, if area alone set the current? Why is the real answer bigger still?

Next: Lab 02: The diode I–V curve.

8. Lab 02: The diode I–V curve

In this lab you'll sweep one real SKY130 diode through nine decades of current, show that superposition no longer works once a diode is in the circuit, and pull two of SkyWater's own model parameters back out of the shape of your graph. Then you'll work out which parts of the curve come from the diode and which come from the simulator, and heat the diode to 100 °C to see which parameters move.

Package: `labs/lab-02-diode-iv/`

Before you start

- Finish Lab 01: `make` in `labs/lab-01-first-schematic` prints `PASS` and **501.046 µA**.
- Read Linearity and superposition, The operating point, The diode equation and Three diode models.
- Open a terminal in the workbench and go to the lab folder:

```
cd /foss/designs/modules/ad103/labs/lab-02-diode-iv
```

This lab doesn't need any environment setup, so you don't have to load `.designinit`, run `mod` or set `PDK`. The decks find the models through `PDK_ROOT`, which the image already sets to `/foss/pdks`, and the `Makefile` sets it again anyway. You don't need XSchem in this lab either. If you do open the bench in `xschem/`, XSchem needs `PDK=sky130A` to find its symbols, and `xschem/xschemrc` sets that for you. (On the image's default IHP process, this bench netlists to `D1 - diode IS MISSING !!!!`, and ngspice then simulates it without a single error and writes a column of zeros.)

Time: about 1.75 hours. `make` takes about 25 seconds, and each of the other targets takes about ten; those wall times depend on your machine. Every other number on this page should come out exactly as printed, because you're running the same deck on the same image, and the checker enforces them.

The two equations from the guide pages are all the theory you need for this lab:

$$I = I_0\left(e^{V/nV_T} - 1\right), \qquad V_T = \frac{kT}{q} = 25.8649\ \text{mV at}\ \ 27^\circ\ \text{C} \tag{8.1}$$

$$\text{one decade of current} = nV_T \ln 10 = n \times 59.556\ \text{mV} \tag{8.2}$$

You'll measure the left-hand sides and solve for n and I_0.

Part 1: Run the lab

Does your workbench reproduce the reference sweep?

1. Look at the bench you're about to simulate, in the picture below. `make` simulates the hand-written deck `spice/diode_iv.spice`, and `xschem/diode_tb.sch` is the same circuit drawn as a schematic:

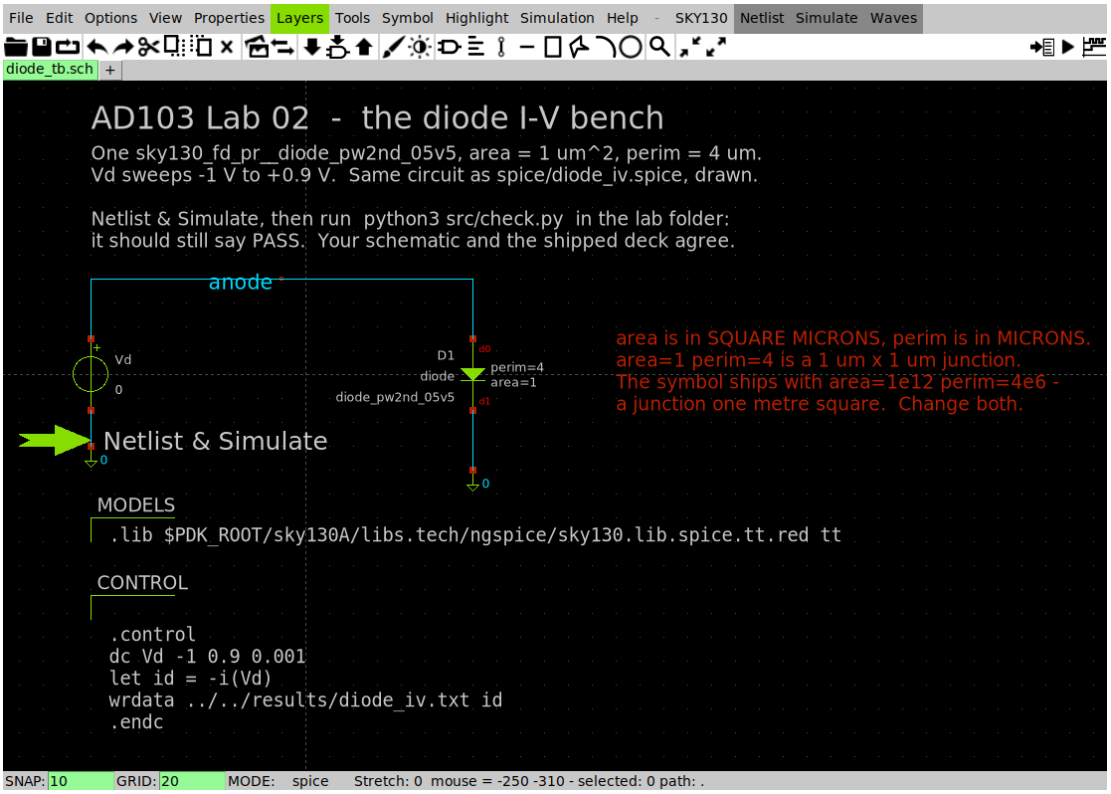


Figure 8.1: diode_tb.sch in XSchem: one SKY130 diode, one swept source, and the model and control text blocks

One `sky130_fd_pr__diode_pw2nd_05v5` with `area=1` and `perim=4`, a source `Vd` sweeping −1 V to +0.9 V, and the two text blocks that carry the `.lib` line and the `.control` block.

2. Run everything: the DC sweep, the additivity test, three plots, and the checker.

```
make
```

It prints 81 lines, and Parts 2 to 4 go through them one block at a time. The output starts and ends like this:

```
== DC sweep: 1901 points, -1.000 V to +0.900 V
   (about 10 s, nearly all of it loading the model library)
   wrote results/diode_iv.txt
== additivity test: a resistor and a diode, driven three ways
...
== checking
...
PASS  every number matches the reference run
```

The checker compares currents to within 1 %, voltages to within 1 mV and the slope to within 0.5 % of the reference run.

You should see `PASS  every number matches the reference run`. Every `results/*.log` file will also contain `IKR too small - model effect disabled!`, on every run. That message is harmless, and the ngspice survival card explains why. If something else goes wrong, look in the card's Error messages table, which lists the diode errors.

Part 2: Test superposition

Does doubling the drive double a diode's current, the way it does a resistor's?

The test drives a 100 kΩ resistor and a 1 µm² diode three times each: once with input A (0.35 V), once with input B (0.35 V), and once with A+B (0.70 V).

1. Predict whether the diode's `i_d_ab` will equal `i_d_a` plus `i_d_b`. Write down yes or no, and if you say no, which way you think it misses.
2. Print the additivity block on its own:

```
make line
```

```
== additivity test: a resistor and a diode, driven three ways
--- 100 kohm resistor (amps) ---
    A alone, B alone, then A+B predicted by superposition, then measured
i_r_a = 3.500000e-06
i_r_b = 3.500000e-06
i_r_sum = 7.000000e-06
i_r_ab = 7.000000e-06
--- 1 um^2 diode (amps) ---
    A alone, B alone, then A+B predicted by superposition, then measured
i_d_a = 1.254090e-10
i_d_b = 1.254090e-10
i_d_sum = 2.508180e-10
i_d_ab = 3.917437e-06
```

3. For the resistor, add `i_r_a` and `i_r_b` and compare the sum with `i_r_ab`. Do the same for the diode.
4. Work out how far off superposition is for the diode, as a factor.

You should see the resistor's sum match to every digit (7.000000 µA). For the diode, superposition predicts 250.8180 pA and the simulator measures 3.917437 µA, so the prediction is short by a factor of **15,618.6**. The checker's lines near the end of `make` report the same result: `resistor: superposition off by 0.0000 %` and `diode    : superposition off by a factor of 15,618.6`.

Part 3: Read the curve on two axes

What does the curve look like on linear and on log axes, and what slope does the log plot have?

1. Open `results/ad103-diode-iv-linear.png`:

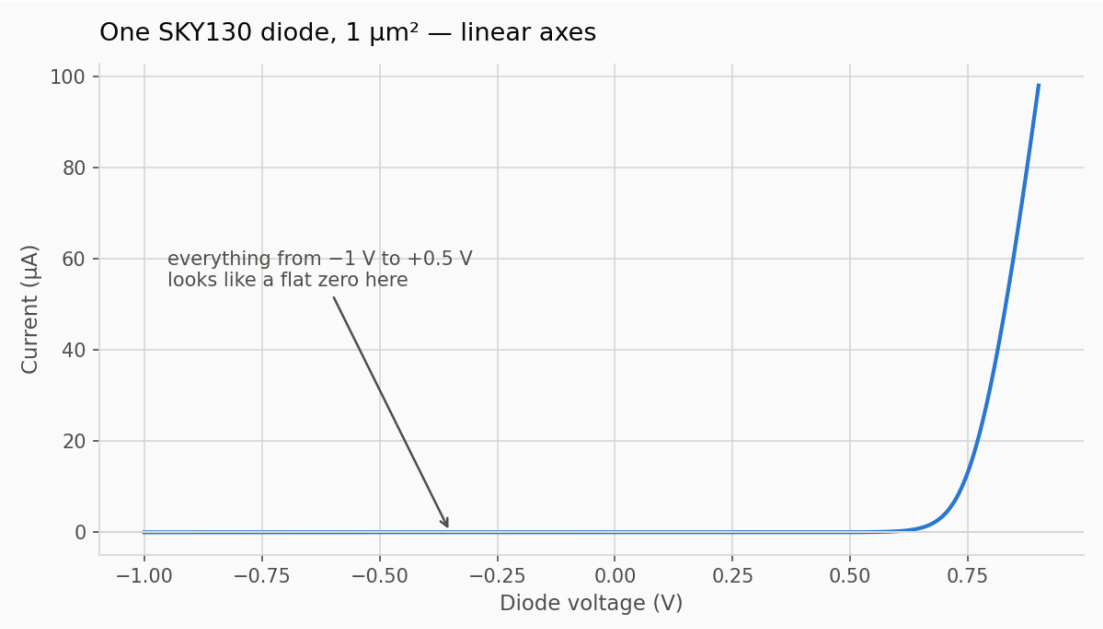


Figure 8.2: Current against voltage on linear axes: flat, then a wall

Try to find the point where the current is 1 nA. Why can't you?

Answer: A1, in Answers at the end.

2. Open `results/ad103-diode-iv-log.png`:

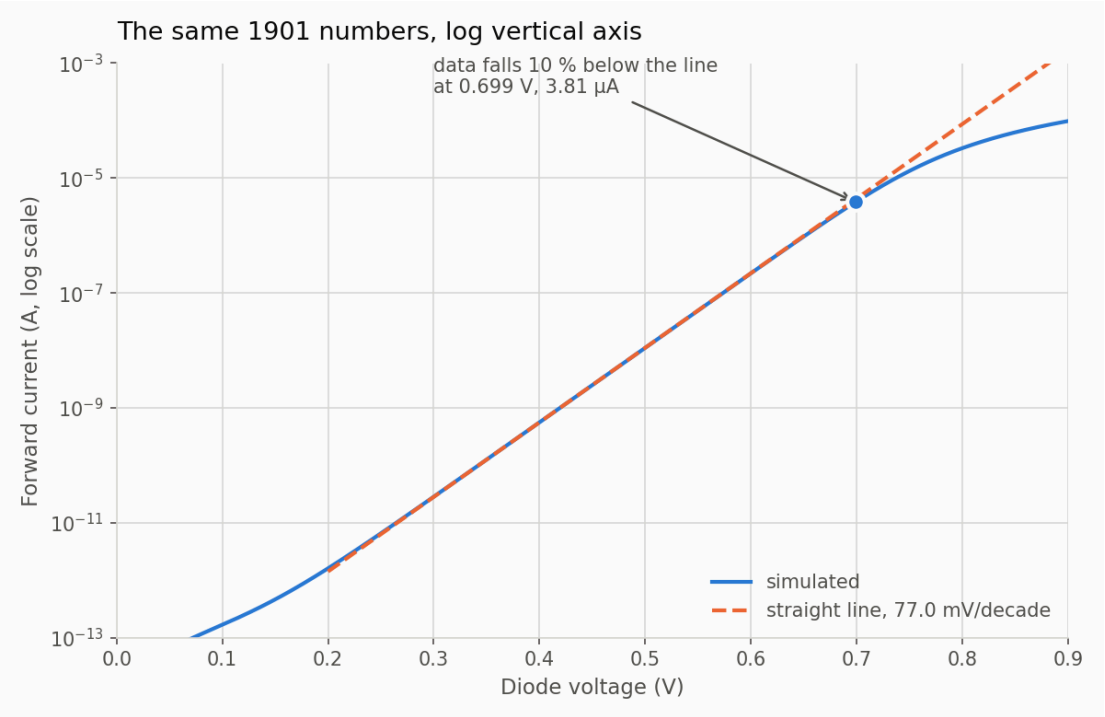


Figure 8.3: The same data with a log current axis: six decades of straight line

Put a ruler on the straight part and read off two points a decade apart. Write down the millivolts between them.

3. Compare your reading with the decade table in the `make` output:

```
current      forward V     mV since previous decade
   1e-12 A   0.181962 V           -
   1e-11 A   0.264643 V       82.68
   1e-10 A   0.342408 V       77.77
   1e-09 A   0.419503 V       77.09
   1e-08 A   0.496518 V       77.02
   1e-07 A   0.573602 V       77.08
   1e-06 A   0.651480 V       77.88
```

Then read the `local slope, measured over a 20 mV window` table further down the output. It gives the slope every 50 mV from 0.050 V to 0.850 V:

```
local slope, measured over a 20 mV window:
   V        I(A)         mV/decade
 0.050  6.2317e-14       96.11
 0.100  1.6726e-13      123.18
 0.150  4.6233e-13      101.91
 0.200  1.6055e-12       85.75
 0.250  6.5318e-12       79.61
 0.300  2.8333e-11       77.72
 0.350  1.2541e-10       77.19
 0.400  5.5825e-10       77.05
 0.450  2.4887e-09       77.01
```

```
  0.500  1.1097e-08       77.02
  0.550  4.9442e-08       77.11
  0.600  2.1945e-07       77.50
  0.650  9.5789e-07       79.19
  0.700  3.9174e-06       85.92
  0.750  1.3278e-05      107.03
  0.800  3.3105e-05      151.64
  0.850  6.2443e-05      217.80
```

You should see a slope of about 77 mV per decade everywhere between about 0.30 V and 0.65 V, and the four middle rows of the decade table agreeing with each other to better than 1 %.

Part 4: Extract n and r_s, and check the model card

What are the ideality factor and series resistance of this diode, and does SkyWater agree?

1. Find the fit block in the `make` output:

   ```
   fit over 154 points from 1e-09 A to 1e-07 A:
     slope           : 77.037 mV/decade
     thermal voltage : 25.8649 mV  (27 degC)
     ideality n      : 1.2935
     I0 (V=0 intercept): 3.5864 fA
     leaves the line : 0.699 V, 3.8136 uA (line says 4.2487 uA)
   ```

   Divide the slope by the ideal 59.556 mV/decade, $77.037/59.556 = 1.2935$, and you have the **ideality factor**. The `leaves the line` row shows where the curve starts to bend away from the straight line, at 0.699 V, and The diode equation explains why it bends.

2. Find the series-resistance block and check its arithmetic, which divides the 95.989 mV dropped outside the junction by the 98.0354 µA flowing through it.

   ```
   series resistance, from the top of the curve:
     at 0.900 V the diode carries 98.0354 uA
     the straight line needs only 0.804011 V for that current
     so 95.989 mV is dropped outside the junction
     rs = 95.989 mV / 98.0354 uA = 979.1 ohm
   ```

3. Write down your n and r_s. Then read SkyWater's own numbers for this diode from its model card:

   ```
   grep -m1 -A20 'model sky130_fd_pr__diode_pw2nd_05v5 d' \
                                    /foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/
   sky130_fd_pr__diode_pw2nd_05v5.model.spice \
     | grep -E '^\+ (js|jsw|n|rs) '
   ```

   ```
   + js = 2.75e-015 ; Units: amper/meter^2
   + jsw = 6e-016 ; Units: amper/meter
   + n = 1.2928
   + rs = 981 ; Units: ohm (ohm/meter^2 if area defined)
   ```

You should see n = 1.2935 from your fit, against `n = 1.2928` on the card, and r_s = 979.1 Ω, against `rs = 981`.

Part 5: Find the operating point

Where does a resistor in series with a diode settle, and how close did the hand iteration get?

1. Solve one resistor and one diode in series, driven from 1.8 V, for three resistor values:

```
make op
```

```
== one resistor and one diode in series, three times
--- diode voltage at the operating point (volts) ---
    R = 10 kohm, 100 kohm, 1 Mohm
v(n1) = 8.905358e-01
v(n2) = 7.398473e-01
v(n3) = 6.561163e-01
--- and the current through it (amps) ---
i_10k = 9.094642e-05
i_100k = 1.060153e-05
i_1meg = 1.143884e-06
```

2. Compare `v(n2)` and `i_100k` with the hand iteration on The operating point (0.7335 V and 10.665 µA). How far off was the hand answer: the voltage in mV, and the current in percent?

   Answer: A2, in Answers at the end.

3. Open `results/ad103-diode-load-line.png` and check the two marked crossings against `v(n2)` and `v(n3)`:

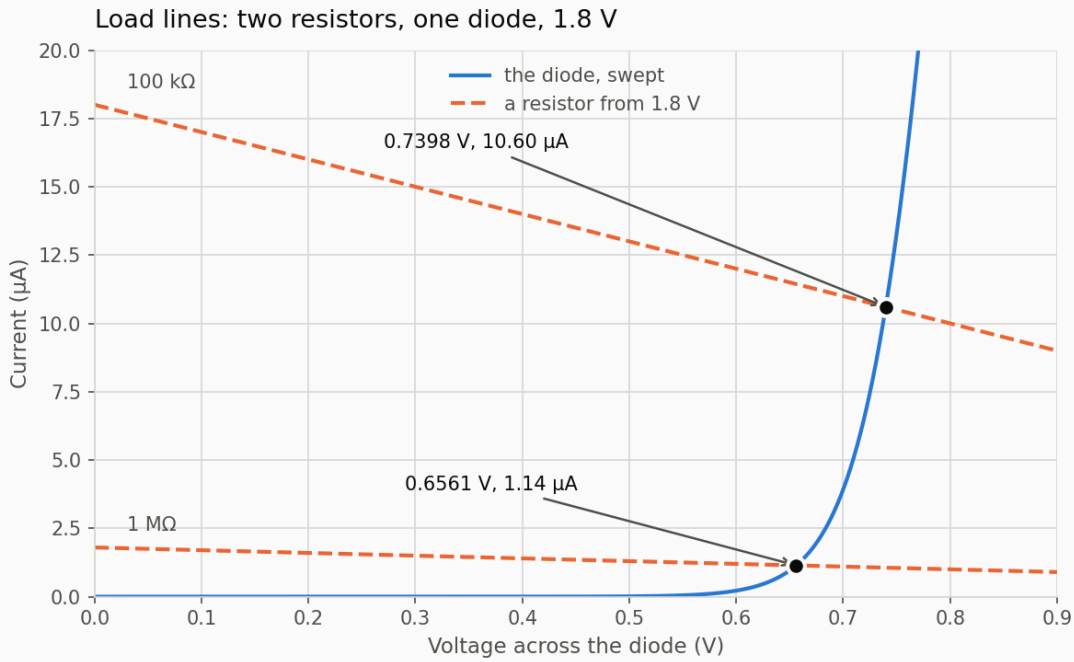


Figure 8.4: The diode curve with two resistor load lines and their crossings marked

4. Now find the same operating point a completely different way, from the DC sweep, which has no resistor in it at all. This script reads `results/diode_iv.txt`, the sweep from Part 1, walks along it until the diode's current crosses the 100 kΩ load line, and then interpolates between the two points on either side:

```
python3 -c "
import sys; sys.path.insert(0,'src')
from iv import read_iv
v, i = read_iv()
prev = None
for a, b in zip(v, i):
    f = b - (1.8 - a)/1e5
    if prev and prev[1] < 0 <= f:
```

```
        v0, f0 = prev
        vx = v0 + (0 - f0)/(f - f0)*(a - v0)
        print(f'{vx:.6f} V   {(1.8 - vx)/1e5:.6e} A'); break
    prev = (a, f)
"
```

```
0.739846 V   1.060154e-05 A
```

You should see the plot's marked crossings at `0.7398 V` and `0.6561 V`, matching `v(n2)` and `v(n3)`, and the script's crossing agreeing with `op`'s 7.398473e-01 V and 1.060153e-05 A to within one in the sixth digit.

The script's crossing came from the sweep of a lone diode, while `op` solved a different circuit, a resistor and a diode together. Why do the two agree so closely?

Answer: A3, in Answers at the end.

Part 6: The reverse floor

Is the flat reverse current on the log plot the diode, or the simulator?

1. Measure the reverse current twice, with two simulator settings:

```
make floor
```

```
--- default GMIN = 1e-12 S (amps; negative = current flowing backwards) ---
    at -1.000 V, at -0.500 V, at +0.100 V
i_m1v0 = -1.00356e-12
i_m0v5 = -5.03558e-13
i_p0v1 = 1.672571e-13
--- GMIN = 1e-15 S: the same diode, a thousand times less simulator ---
    at -1.000 V, at -0.500 V, at +0.100 V
j_m1v0 = -4.55885e-15
j_m0v5 = -4.05763e-15
j_p0v1 = 6.735706e-14
```

2. GMIN is the small conductance that ngspice puts across every junction, and its default is 10^{-12} S. Subtract GMIN $\times V$ from each of the four reverse numbers (the `m1v0` and `m0v5` lines). Use 10^{-12} S for the `i_` lines and 10^{-15} S for the `j_` lines, and keep the signs, since V is negative. Write down what's left.
3. Compare what's left with the `I0` that the forward fit reported in Part 4.

You should see −3.56, −3.558, −3.559 and −3.558 fA (negative because the current flows backwards). That's the same constant four times, from two voltages and two simulator settings, and it's close in size to the **3.5864 fA** you got from the opposite end of the curve.

Suppose you had reported "this diode leaks 1 pA" from the default run. What would you have been reporting?

Answer: A4, in Answers at the end.

Part 7: Junction area and size mistakes

What does a bigger junction buy, and what happens when the size is written wrong?

1. Run the area deck:

```
make area
```

```
--- Part 1: forward current at 0.500 V vs junction area (amps) ---
    area = 1, 4, 16, 100 um^2
i_1um = 1.109698e-08
i_4um = 3.457181e-08
```

```
i_16um = 1.186544e-07
i_100um = 6.679656e-07
```

2. Before you calculate anything, predict what `i_100um` divided by `i_1um` would be if current were proportional to area. Write it down, then do the division. Then repeat the face-and-edge fit from Three diode models using only the first and last rows, and check that it predicts the middle two. The four junctions are squares, 1×1, 2×2, 4×4 and 10×10 µm, so their perimeters are 4, 8, 16 and 40 µm. That page has the numbers to check your fit against.
3. Read the second half of the same output. (The `Part 1` and `Part 2` in it are the deck's own section names, not this lab's parts.) All four of these currents are meant to come from a 1 µm² junction:

```
--- Part 2: three lines that all mean to say 1 um^2, at 0.700 V (amps) ---
    correct, then perim misspelled as pj, then symbol defaults
i_b1um = 3.917437e-06
i_pj = 2.571530e-04
i_def = 2.291258e+06
--- and area=1u, which means a millionth of a square micron (amps) ---
i_su = 4.541414e-12
```

ngspice didn't print an error for any of them. Compare the device lines yourself:

```
grep -n '^X' spice/diode_area.spice
```

For each of the three wrong lines, work out what junction ngspice actually simulated.

Answer: A5, in Answers at the end.

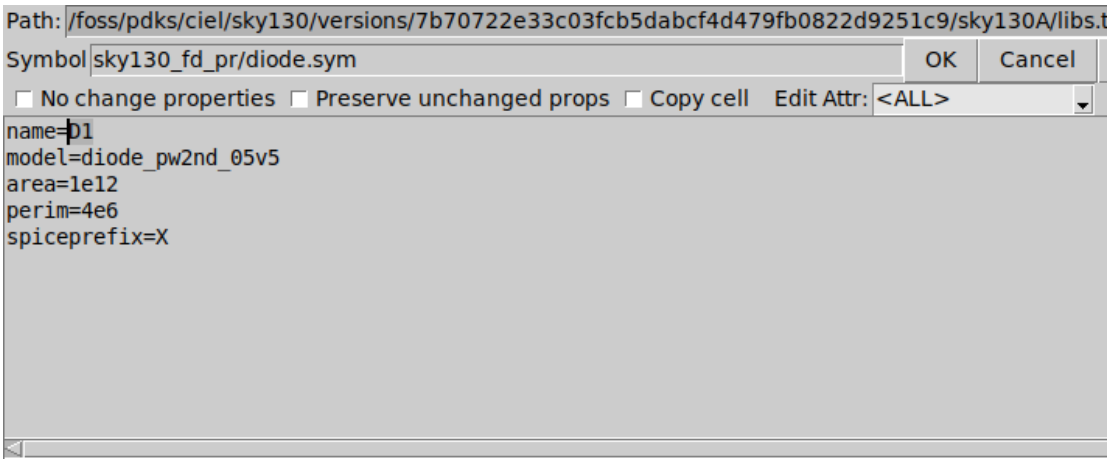


Figure 8.5: The XSchem property editor on a diode.sym instance, showing model=diode_pw2nd_05v5, area=1e12, perim=4e6

*This is the symbol-defaults row as you would meet it in XSchem. A fresh `diode.sym` comes with `area=1e12` and `perim=4e6`, shown here in its property editor (q), so whenever you place one, change **both** before you netlist.*

You should see a ratio of **60.19** rather than 100, and four "1 µm²" diodes whose currents range from picoamps to megaamps without a single error message.

In Lab 01 the same `u` mistake on a MOSFET stopped ngspice with `could not find a valid modelname`. Why doesn't it complain here?

Answer: A6, in Answers at the end.

Check every diode you netlist. Run `grep -n 'diode' <netlist>` and read the `area=` and `perim=` on every line. The numbers should be bare microns, `perim` should be spelled out, and none of the symbol defaults should be left in place.

Part 8: Read a FAIL you caused

Which checks does a 5 % scaling error trip, and which does it leave alone?

1. Predict whether the slope check will fail if every current is 5 % too high, and write down your guess.
2. Hand the checker a sweep that is 5 % off:

   ```
   make broken
   ```

   `src/bend_it.py` copies your sweep and multiplies every current by exactly 1.05, a change far too small to see on either plot.

   ```
   == feeding the checker a sweep that is 5 % off, on purpose
     wrote results/diode_iv_bent.txt (every current multiplied by 1.05)
   ...
   FAIL
     - I(0.450 V) is 2.613143e-09 A, reference is 2.488707e-09 A - off by +5.00 %
     - I(0.600 V) is 2.304189e-07 A, reference is 2.194466e-07 A - off by +5.00 %
     - I(0.700 V) is 4.113309e-06 A, reference is 3.917437e-06 A - off by +5.00 %
     - reverse current -1.0537 pA is not the reference -1.0036 pA
     - V at 1e-09 A is 0.417871 V, reference is 0.419503 V
     - V at 1e-08 A is 0.494886 V, reference is 0.496518 V
     - V at 1e-07 A is 0.571966 V, reference is 0.573602 V
     - I0 is 3.7644 fA, reference is 3.5864 fA - the whole curve is scaled wrong, which
   usually means the junction is not the size you think it is
   ```

3. Now look at what did **not** fail. Scroll up to the checker's lines above `FAIL` and find the `slope` and `ideality n` lines. The checker tests the slope, but it only prints n as a note and doesn't test it.
4. Run `make` again to get back to a clean `PASS`.

You should see the slope still at `77.0349` against a reference of `77.0367`, and n still at `1.29348` against `1.29351`, while the three decade voltages have each moved by about 1.63 mV.
Why didn't the slope change, and what does that tell you about reading a wrong curve?
Answer: A7, in Answers at the end.

Part 9: Heat the diode

What does 100 °C do to the slope, to n and to I_0?

1. Predict the slope at 100 °C. V_T is proportional to absolute temperature, and 27 °C is 300.15 K while 100 °C is 373.15 K. Write your number down, and say whether you expect n and I_0 to move as well.

   Answer: A8, in Answers at the end.

2. Open `spice/diode_iv.spice` in a text editor and add this line directly under the `.lib` line:

   ```
   .options temp=100
   ```

   Save the file.

3. Rerun the sweep and the plots:

   ```
   make sweep plots
   ```

   The `plots` step prints the fit block, which now reads as follows. It uses 191 points instead of 154, because the sweep steps by 1 mV and the two decades now span 191 mV of it:

```
fit over 191 points from 1e-09 A to 1e-07 A:
  slope           : 95.634 mV/decade
  thermal voltage : 25.8649 mV  (27 degC)
  ideality n      : 1.6058
  I0 (V=0 intercept): 8067.1022 fA
  leaves the line : 0.000 V, 0.0000 uA (line says 0.0000 uA)
```

Ignore the `leaves the line` row for this run. The script reports the first forward point that falls 10 % below the fitted line, skipping points where the line itself is below 1 pA. At 27 °C, where I_0 is only 3.5864 fA, that rule skips the bottom of the curve. At 100 °C I_0 is above 1 pA, so nothing gets skipped, and the script reports the very bottom of the curve instead of the bend near the top that Part 4 looked at.

4. The script still divides by the 27 °C thermal voltage, so its `ideality n` line is wrong for this run. Work out V_T at 373.15 K, and then your own n from the slope.

   Answer: A9, in Answers at the end.

5. Compare I_0 with Part 4's 3.5864 fA. Two things moved the curve: the slope, through V_T, and the intercept I_0. Which moved more?

   Answer: A10, in Answers at the end.

6. Run the checker on the hot sweep:

   ```
   make check
   ```

   It prints `FAIL`, and that's correct, because you changed the experiment. Read which lines it objects to and which it doesn't.

   Answer: A11, in Answers at the end.

7. Delete the `.options temp=100` line and run `make`. (Your course folder is the git clone from Lab 01's `mod add`, so `git checkout -- spice/diode_iv.spice` also puts the deck back.)

You should see a slope of **95.634 mV/decade** at 100 °C, within 0.2 % of your prediction, your own n of 1.2916 and an I_0 of **8067.1022 fA**. Once the line is gone, you should see `PASS  every number matches the reference run` again.

Finishing

You're done when your 100 °C sweep in Part 9 gives 95.634 mV/decade, your own n from it comes out at 1.2916, and `make` prints `PASS` again after you take the line out.

Questions

1. What are n and r_s for this diode, from your fit, and what does the model card say? (Part 4.)
2. Why is the reverse current at −1 V not the saturation current I_0? (Part 6.)
3. Why did a hundred times the area give only 60.19 times the current? (Part 7, step 2: the face-and-edge fit from Three diode models.)

Submit

Nothing to submit.

Going further

None of this is graded.

1. **Sweep the other diode.** `sky130_fd_pr__diode_pd2nw_05v5` is the p-diffusion-into-n-well junction, the same idea built the other way up. Change the model name in the deck and see whether n comes out the same.
2. **Two in series.** Put two 1 µm² diodes in series across the sweep source, and predict the decade slope before you run it.

If you get stuck, ask in the team Discord. Quote the exact line you're stuck on, and say which `make` target you ran.

Next on the Course path: read How a MOSFET works.

Answers

A1. Part 3: Read the curve on two axes

1 nA is 0.001 % of the top of the axis, far too small to see. The decade table in step 3 says the current reaches 1 nA at **0.419503 V**, which on this plot is somewhere in the flat part. Most analog circuits work in the part of the curve that linear axes can't show.

A2. Part 5: Find the operating point

The hand answer was **6.3 mV** and **0.60 %** off. The hand iteration assumed a constant 77.04 mV per decade, but above about 0.7 V the curve has already started to bend away from that straight line. The operating point works through the difference.

A3. Part 5: Find the operating point

`op` solves the resistor and diode once, by iterating, and the sweep is 1901 separate solves of a lone diode with no resistor, so the only thing the two calculations have in common is the device model. Agreement that close means the number comes from the model and not from the way either calculation reached it. That crossing is the bias point, and finding it is the first step in designing any analog circuit.

A4. Part 6: The reverse floor

You'd have been reporting ngspice's default GMIN rather than anything about the silicon, because 10^{-12} S across 1 V is 1 pA. The diode equation works through the rest of the arithmetic.

A5. Part 7: Junction area and size mistakes

What you wrote	What you got at 0.700 V	Why
`area=1 perim=4`	**3.917437 µA**	correct
`area=1 pj=4`	257.1530 µA, **66× too big**	`pj` is the raw model card's name for the perimeter, but the subcircuit parameter is called `perim`. So `pj=4` is an unknown parameter and is silently ignored, and `perim` keeps its default of **1e6**, a junction one metre around.
(no parameters at all)	2,291,258 A	The subcircuit falls back on its own defaults, `area=1e12 perim=1e6`, which describe a junction **one metre square** and give you two and a quarter **mega**amps. An XSchem `diode.sym` that you never edit writes the same area, with `perim=4e6`.
`area=1u perim=4u`	4.541414 pA	Micron numbers take **no unit suffix**, exactly as with `W` and `L` in Lab 01, so `1u` is a millionth of a square micron.

A6. Part 7: Junction area and size mistakes

MOSFET models are binned, and a device that small falls outside every bin, so ngspice can't find a model for it. Diode models aren't binned, so here the same mistake gives you a clean-looking answer that is completely wrong, and nothing but your own checking will catch it.

A7. Part 8: Read a FAIL you caused

A 5 % scaling error multiplies every point by the same factor, which on a log-current axis moves every point up by the same distance, so the slope doesn't change; the fit's last digits move by only 0.002 %. Seen along the voltage axis, the same shift slides the whole line sideways by $77.04 \times \log_{10}(1.05) = 1.63$ mV, which is exactly the shift in the three decade voltages.

This is the diagnostic habit we want you to take from the lab. On a log-current plot, **the slope tells you about n and the offset tells you about I_0, and the two can go wrong independently.** A wrong slope means the physics is wrong. A wrong offset with the right slope means the *size* is wrong: the wrong `area`, the wrong `perim`, or the wrong number of devices in parallel.

A8. Part 9: Heat the diode

If only V_T changes, 77.04 mV/decade becomes $77.04 \times (373.15/300.15) = 95.8$ mV/decade.

A9. Part 9: Heat the diode

$V_T = 25.8649\ \text{mV} \times 373.15/300.15 = 32.156$ mV, so one decade is $n \times 32.156 \times \ln 10 = n \times 74.04$ mV, and $n = 95.634/74.04 = 1.2916$. At 27 °C it was 1.2935, so the ideality factor has barely moved.

A10. Part 9: Heat the diode

The change in slope is all due to V_T, because n stayed put. I_0 moves far more, from 3.5864 fA to 8067.1 fA, which is about 2,250 times larger. I_0 is carried by the minority carriers that thermal energy knocks loose, and The diode equation explains that their supply is set by temperature alone.

A11. Part 9: Heat the diode

It objects to the three forward currents, the reverse current (−9.1362 pA), the three decade voltages, the slope and I_0. It doesn't object to the two superposition lines, because they come from `spice/straight_line.spice`, which still ran at 27 °C. Nor does it object to `ideality n`, which it prints as a note and doesn't check.

III The MOSFET

9 How a MOSFET works 63
9 The four terminals 63
9 The channel at four bias points 63
9 The squeeze, in numbers 65
9 Units on `W` and `L` 65
9 What a MOSFET is used for 65
9 Check yourself 66

10 The four regions 67
10 The output curves 67
10 1. Cutoff 67
10 2. Triode: a resistor the gate sets 68
10 3. Saturation: a current source the gate sets 68
10 The boundary between triode and saturation . . 69
10 4. Subthreshold: the region most textbooks skip 69
10 The four regions on one card 70
10 Check yourself 70

11 Where saturation starts 73
11 Measuring V_{TH} 73
11 The measured knee 74
11 Velocity saturation 75
11 Channel-length modulation 75
11 Check yourself 75

12 Lab 03: MOSFET regions 77
12 Before you start 77
12 Part 1: Run the sweeps 77
12 Part 2: Extract the threshold two ways 78
12 Part 3: The output characteristic 79
12 Part 4: Subthreshold conduction 80
12 Part 5: Measure g_m two ways 80
12 Part 6: Four device geometries at one bias 81
12 Part 7: Units on `W` and `L` 82
12 Part 8: Add a sixth gate voltage 83
12 Part 9 (optional): Draw it in XSchem 84
12 Finishing 85
12 Going further 85
12 Answers 85

9. How a MOSFET works

Everyone says a transistor is a switch, and yet your textbook gives you three different equations for it. The reason is that a MOSFET is a switch you can **squeeze**, and this page explains what that means.

Think of a garden hose with your hand around it. If you squeeze hard, nothing gets through. If you let go completely, water flows freely, limited only by the tap. In between, which is where every amplifier works, the hose acts as a resistor whose value your hand sets. So one device gives you three behaviours, depending on how hard you squeeze and how hard the tap pushes, and the three textbook equations describe one channel in three different states rather than three different devices.

The four terminals

A MOSFET has four terminals, and only one of them is the hand:

Terminal	What it does	In the hose picture
Gate (G)	sets how open the channel is	your hand on the hose
Source (S)	where the carriers come from	the tap end
Drain (D)	where they go	the open end
Body / bulk (B)	the silicon everything is built in	the ground the hose lies on

The most important thing to know about the gate is what it's made of. **MOS** stands for **Metal–Oxide–Semiconductor**, and the middle word is an insulator. The gate is a plate of conductor sitting on a few nanometres of silicon dioxide (glass) on top of the silicon, so it forms a capacitor with the silicon rather than making an electrical connection to it. ngspice will tell you how big that capacitor is: for the W = 5 µm, L = 1 µm device this course measures, `@m.xm1.msky130_fd_pr__nfet_01v8[cgg]` comes back as **30.56 fF** at $V_{GS} = V_{DS} = 1.8$ V.

This means that **no steady current flows into the gate.** Instead of injecting anything into the silicon, you are putting charge on a plate, and the electric field from that charge reaches through the glass and pulls mobile electrons up to the surface of the silicon underneath, out of a p-type body that had few to spare. Those electrons are the **channel**.

That is why the terminal is called a gate rather than an input: moving it costs charge, but holding it where it is costs nothing. It's also the reason you'll read that the gate is high impedance.

> **The body is a terminal too.** In this course the body is wired to the source on every NMOS, which makes the four-terminal device behave like the three-terminal one in your textbook. What moves the threshold is about what happens when the two are at different voltages.

The channel at four bias points

Here is the same transistor at four bias points. Nothing changes from one panel to the next except the two voltages, so watch what happens to the blue layer.

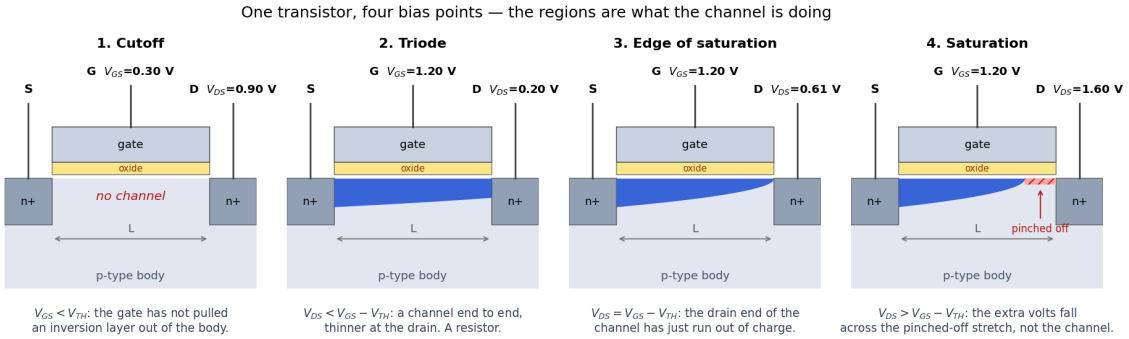


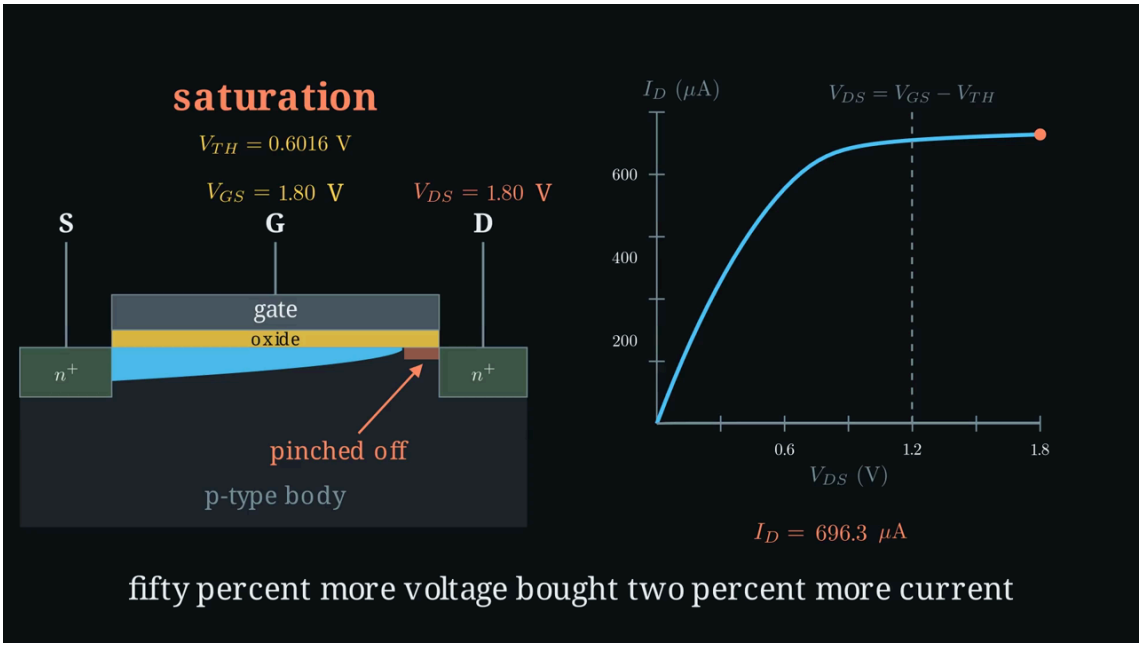
Figure 9.1: Four cross-sections of one n-channel MOSFET: cutoff, triode, edge of saturation, saturation

Produced by `src/channel_picture.py` *in the Lab 03 package. The thickness of the blue layer is proportional to the local inversion charge, and the red hatch marks the pinched-off stretch where there is no channel left at all.*

Read the panels from left to right:

1. **Not squeezing hard enough** (panel 1). The gate voltage is too low to pull a channel out of the body, so there is no conducting path from source to drain. This is **cutoff**.
2. **Squeezing, with a small push** (panel 2). A channel runs from end to end. It is a little thinner at the drain, because the drain voltage partly cancels the gate's pull there, and the current is proportional to the push. This is **triode**, where the device acts as a resistor.
3. **The drain end runs out** (panel 3). If you push harder, the channel gets thinner at the drain end until, at one particular drain voltage, it has exactly zero charge there.
4. **Push harder still** (panel 4). The zero-charge point moves back from the drain, and all the extra volts you added fall across that pinched-off stretch instead of across the channel. The channel itself sees the *same* voltage it did in panel 3, so it carries the *same* current. This is **saturation**, where the device acts as a current source.

The last panel is the one worth understanding properly. In saturation the current stops increasing because **the part of the device that sets the current no longer sees your extra voltage**, not because an equation says so.



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

The four panels above, animated: V_{GS} and V_{DS} move continuously while the I_D–V_{DS} curve traces itself out alongside them. The source is the `MosfetChannel` *scene in* `tooling/manim/scenes/ad103_mosfet_channel.py`*. It's worth watching once before Lab 03 and again after it.*

The squeeze, in numbers

Lab 03 sweeps a W = 5 µm, L = 1 µm `nfet_01v8` at five gate voltages, and this is the first table it prints:

```
== I_D vs V_DS   (W=5, L=1)
   gate voltages read from spice/id_vds.spice: 0.6 0.9 1.2 1.5 1.8
     V_GS  I_D at V_DS=1.8   knee V_DS   channel R at V_DS=10 mV
     0.60         2.005 uA       0.09 V              26648.3 ohm
     0.90        63.876 uA       0.29 V               2529.2 ohm
     1.20       217.505 uA       0.50 V               1288.0 ohm
     1.50       436.080 uA       0.70 V                907.7 ohm
     1.80       696.275 uA       0.89 V                736.9 ohm
```

Look at the last column first. It is the resistance of the channel, measured with only 10 mV across it, which is small enough that the transistor is unambiguously behaving as a resistor. The resistance goes from **26.6 kΩ** to **737 Ω** as the gate goes from 0.6 V to 1.8 V, a 36-fold change with no moving parts, controlled by the voltage on a plate that sits on glass, which makes the transistor a dial rather than an on/off switch.

The current at the off end isn't zero either. With the gate at 0 V and 1.8 V across the device, the same lab measures

```
== below threshold   (V_DS = 1.8 V)
   V_GS = 0.00 V   I_D = 2.1853e-12 A
```

That is **2.19 picoamps**, 320 million times less than the 696 µA the device passes when it is wide open, but still not zero. Much of The four regions is about that number.

Both numbers matter, for different reasons. The 737 Ω is why a digital gate can charge the next gate's input in picoseconds, and that sets your clock speed. The 2.19 pA is why a phone's battery drains while it sits on the table doing nothing. That current is called **leakage**, and on a chip with ten million transistors it has been the dominant design constraint of the last twenty years.

> **On-resistance and off-current.** If someone hands you a transistor and calls it a switch, ask them what its on-resistance and its off-current are. Both are finite, you can measure both in an afternoon, and the word "switch" tells you neither.

You'll produce this table yourself in Lab 03, Part 3.

Units on `W` and `L`

The units mistake described in Getting started applies to every deck from here on. In SKY130, **`W` and `L` are plain numbers in microns, with no unit suffix**, so `W=5` means five micrometres. `W=5u` asks for a width that falls outside every model bin, and ngspice stops with `could not find a valid modelname`, a message that mentions neither `W` nor units. In Lab 03, Part 7 we have you trigger this error once on a MOSFET deck.

What a MOSFET is used for

The regions map onto two jobs:

- **Switching.** You drive the gate all the way to 1.8 V or all the way to 0 V and never linger in between. That uses cutoff and triode, and the numbers you care about are the 737 Ω and the 2.19 pA. This is how all of digital design uses transistors, including every gate in the DD track.
- **Amplifying.** You park the gate somewhere in the middle, keep the drain high enough that the channel stays pinched off, and wiggle the gate. That uses saturation, and what you care about is how much the current moves per volt of wiggle. That number is called g_m, and it is the subject of Transconductance and output resistance.

An amplifier and a logic gate are the same device biased in different regions, which is why The CMOS inverter treats a logic gate as an amplifier later in this course.

Check yourself

1. Why does no steady current flow into the gate? What does it cost to change the gate voltage?
2. In panel 4, the drain voltage is higher than in panel 3. Why is the current the same?
3. From the table, roughly what is the channel resistance at V_{GS} = 1.2 V, and how does it compare with the value at 1.8 V?
4. Is a transistor used as a digital switch mainly in saturation, or in cutoff and triode?

Next: The four regions.

10. The four regions

Your textbook gives you cutoff, triode and saturation. This page draws the boundaries between them on measured curves, shows how to tell which region a device is in, and then adds a fourth region that most textbooks leave out. Each region is one of the channel pictures from How a MOSFET works, seen this time on a graph.

The output curves

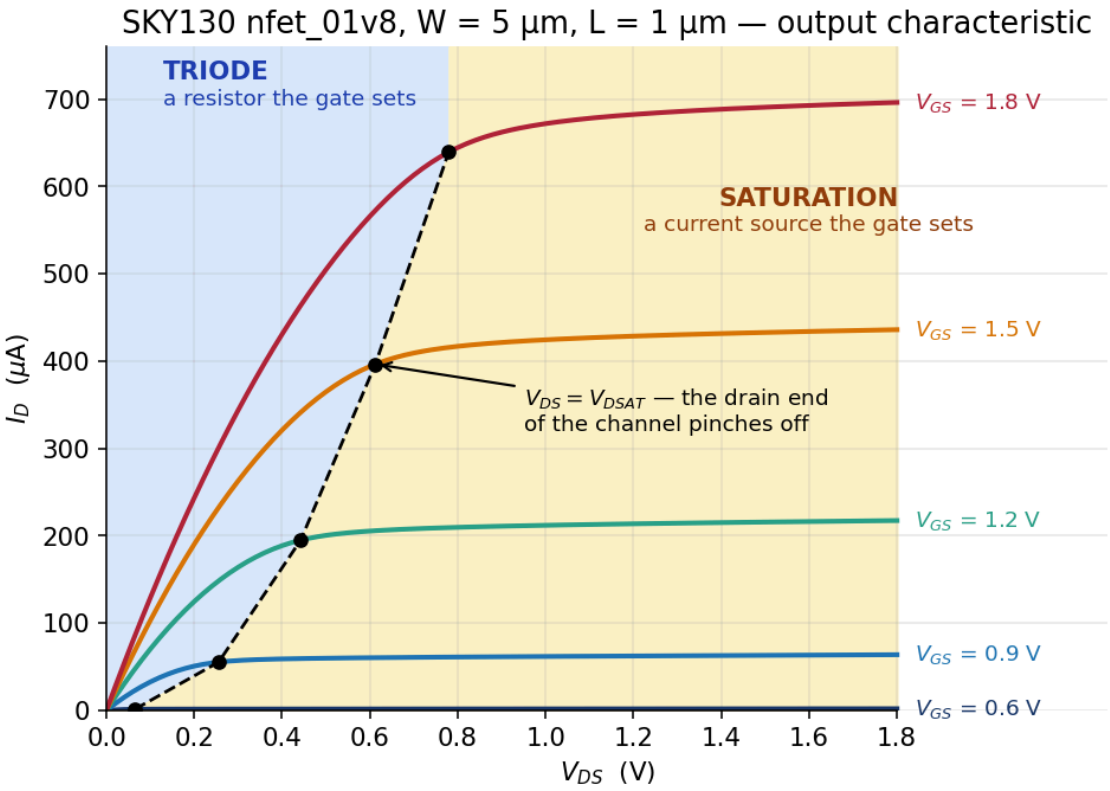


Figure 10.1: I_D vs V_DS for five gate voltages, with the triode and saturation regions shaded

SKY130 `nfet_01v8`, W = 5 µm, L = 1 µm. Five sweeps of V_{DS}, one per gate voltage, from `spice/id_vds.spice` in the Lab 03 package, and drawn by `src/plot_curves.py`. The dashed locus and its dots are the `vdsat` that the model itself reports at each bias, from `spice/op_params.spice`, so they show the simulator's own estimate of where the channel pinched off rather than a fit or a guess of ours.

Almost everything worth knowing about a MOSFET is somewhere in that picture, so we'll go through it one region at a time and match each region to a panel of the cross-section figure on How a MOSFET works.

1. Cutoff

Condition: $V_{GS} < V_{TH}$, where V_{TH} is the **threshold voltage**.

What the channel is doing: there is no channel yet. The gate hasn't pulled enough electrons to the surface to form a conducting layer, so the only path between source and drain is two back-to-back diodes, and one of them is reverse-biased.

On the plot: none of the five curves is in cutoff. The lowest one, $V_{GS} = 0.6$ V, sits right at threshold (the next page measures V_{TH} = 0.6016 V for this device), and even there the current never gets above 2.005 µA over the whole sweep from 0 to 1.8 V. On a plot scaled for 700 µA that is indistinguishable from the axis, and any gate voltage below threshold gives less current still, so a curve in cutoff would be a flat line along the bottom.

$$I_D \approx 0 \tag{10.1}$$

Keep the "≈" in mind, because Section 4 comes back to it.

2. Triode: a resistor the gate sets

Condition: $V_{GS} > V_{TH}$ **and** $V_{DS} < V_{GS} - V_{TH}$.

What the channel is doing: the channel runs all the way from source to drain (panel 2 of the cross-section). It is thinner at the drain end, because the drain's own voltage partly cancels the gate's pull there, but it is still continuous, so if you push harder on the drain, more current flows, almost as it would through a wire.

On the plot: triode is the rising left-hand part of every curve, shaded blue. Near the origin the curves are nearly straight lines through zero, which is the signature of a resistor.

$$I_D = \mu_n C_{ox} \frac{W}{L} \left[(V_{GS} - V_{TH}) V_{DS} - \frac{V_{DS}^2}{2} \right] \tag{10.2}$$

You don't need to memorise that equation, but it's worth reading. It has a term proportional to V_{DS} (the resistor), minus a term that grows as V_{DS}^2 (the channel thinning at the drain end). At small V_{DS} the second term vanishes, and what's left is Ohm's law, with a conductance that the gate controls:

$$R_{ch} \approx \frac{1}{\mu_n C_{ox} \frac{W}{L} (V_{GS} - V_{TH})} \tag{10.3}$$

Lab 03 measures exactly that resistance, by putting 10 mV across the device and dividing by the current that flows:

V_{GS}	channel resistance
0.60 V	26 648.3 Ω
0.90 V	2 529.2 Ω
1.20 V	1 288.0 Ω
1.50 V	907.7 Ω
1.80 V	736.9 Ω

How to recognise triode: look at the curve near the origin. If it's a straight line through zero, the device is in triode and you can treat it as a resistor. The check takes one glance, and it's right far more often than plugging numbers into the quadratic.

3. Saturation: a current source the gate sets

Condition: $V_{GS} > V_{TH}$ **and** $V_{DS} > V_{GS} - V_{TH}$.

What the channel is doing: the channel has run out of charge at the drain end (panel 3), and the pinch-off point has moved back up the channel (panel 4). Any extra drain voltage falls across the pinched-off stretch rather than across the channel, so the channel keeps carrying the current it was already carrying.

On the plot: saturation is the flat right-hand part of every curve, shaded amber. The height of each flat part is set by V_{GS} alone, at 2.005, 63.876, 217.505, 436.080 and 696.275 µA for the five gate voltages. Once the device is in saturation the drain voltage has stopped mattering, but the gate voltage still does.

$$I_D = \frac{1}{2} \mu_n C_{ox} \frac{W}{L} (V_{GS} - V_{TH})^2 \tag{10.4}$$

How to recognise saturation: nudge V_{DS}. If I_D barely moves, the device is in saturation. That is the definition of saturation, and it's also how bias checks are done in real designs: you perturb the drain and see whether the current changes.

The boundary between triode and saturation

The two conditions above meet where $V_{DS} = V_{GS} - V_{TH}$. That quantity is called the **overdrive voltage**, $V_{ov} = V_{GS} - V_{TH}$, and it's the number analog designers think in. It's the drain voltage at which the device stops behaving like a resistor and starts behaving like a current source, so it's also the headroom the device needs before it will act as one.

Where saturation starts shows where the boundary really sits on these curves, and why at high gate voltage the measured knee is 26 % lower than V_{ov}.

4. Subthreshold: the region most textbooks skip

Now go back to the "≈" we asked you to keep in mind in cutoff.

Predict before you read on. The threshold of this device is about 0.6 V. At $V_{GS} = 0.30$ V, half the threshold and comfortably "off", how much current flows with 1.8 V on the drain? Write down an order of magnitude. Zero is an acceptable answer, as long as you commit to it.

Now put the same device on a log axis:

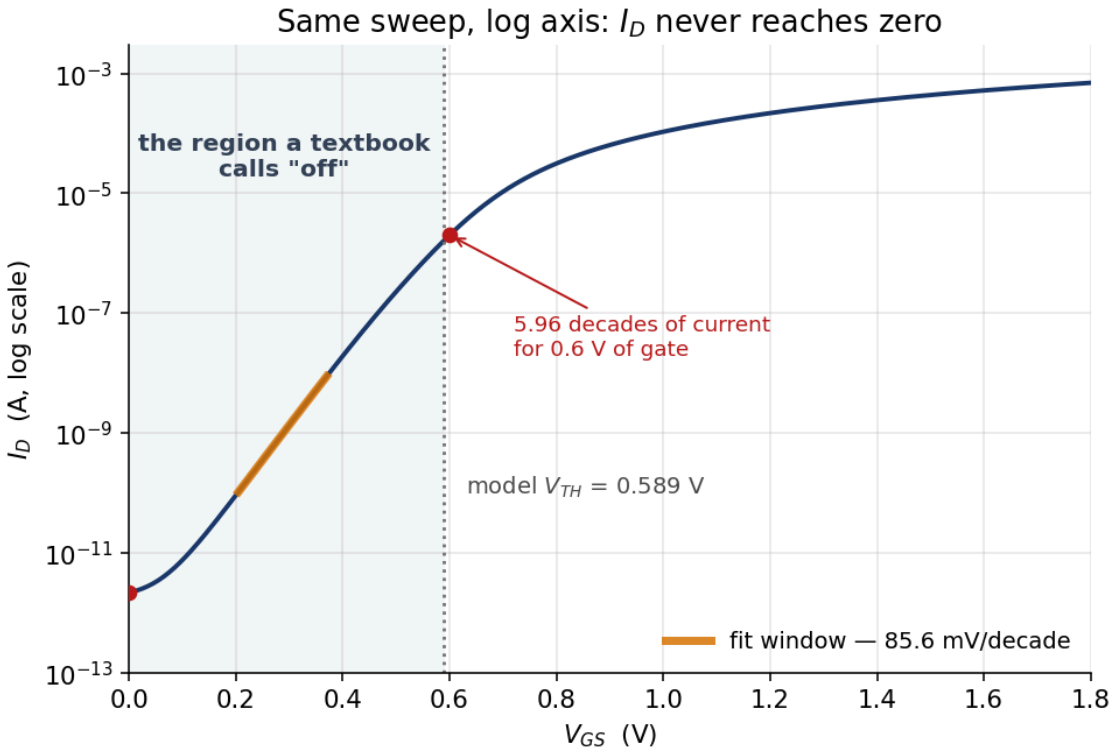


Figure 10.2: I_D vs V_GS on a log axis, showing six decades of current below threshold

From `spice/id_vgs_log.spice`. The same device with the drain at 1.8 V, but this time the gate voltage is swept instead of the drain voltage, and the current is on a log axis.

```
== below threshold   (V_DS = 1.8 V)
   V_GS = 0.00 V   I_D = 2.1853e-12 A
   V_GS = 0.30 V   I_D = 1.3645e-09 A
   V_GS = 0.60 V   I_D = 2.0054e-06 A
   decades from 0.0 V to 0.6 V : 5.96
   subthreshold slope          : 85.6 mV/decade
```

At 0.30 V the device carries 1.36 nanoamps. That's small, but it's a smooth, repeatable, exponential function of gate voltage, so you can't write it off as zero or as noise. On the linear plot this whole region is a flat line on the axis, but on the log plot it's a straight line six decades tall, which is what an exponential looks like on a log axis.

This is **subthreshold conduction**, also called **weak inversion**. The gate hasn't formed a proper channel, but it has lowered the barrier enough that some electrons get across by thermal energy, and the number that do is exponential in the barrier height, exactly as in the diode from Lab 02. The physics is the same, so below threshold a MOSFET behaves like a diode.

The slope of that line is the parameter everyone quotes, the **subthreshold slope**, and on this device it is 85.6 mV per decade: it takes 85.6 mV of gate voltage to change the drain current by a factor of ten. At room temperature no conventional MOSFET can do better than about 60 mV/decade, which is the same $kT/q \ln 10 = 59.6$ mV you met on the diode page, and real devices land above that limit.

Why leakage matters

Suppose you build a chip with ten million of these transistors and switch it off, with every gate at ground:

$$10^7 \times 2.1853\ \text{pA} = 21.9\ \mu\text{A} \tag{10.5}$$

That's tolerable. Now suppose your threshold came out 0.3 V lower than you designed for, which is well within the spread of a real process. In subthreshold the current depends on $V_{GS} - V_{TH}$ and not on either one alone, so a device with V_{TH} 0.3 V lower and its gate at ground carries what this device carries with its gate at 0.3 V, which is the 0.30 V row of the output above:

$$10^7 \times 1.3645\ \text{nA} = 13.6\ \text{mA} \tag{10.6}$$

That's six hundred times as much current, from a 0.3 V shift in one parameter, on a chip that isn't doing anything. This is why threshold voltage is controlled so tightly, why "low power" and "high speed" pull in opposite directions (a low V_{TH} gives you speed and leakage together), and why subthreshold slope has its own line in every process datasheet.

> **A MOSFET is never fully off.** When you hear "the transistor is off", read it as "the transistor is passing a current I have decided to ignore", and then ask how many of them there are.

The four regions on one card

Region	Condition	Channel	I_D behaves like
Subthreshold	$V_{GS} < V_{TH}$	none, but carriers get over the barrier	an exponential in V_{GS}
Cutoff	$V_{GS} \ll V_{TH}$	none	zero, for accounting purposes
Triode	$V_{GS} > V_{TH}$, $V_{DS} < V_{ov}$	continuous, thinner at the drain	a resistor $R(V_{GS})$
Saturation	$V_{GS} > V_{TH}$, $V_{DS} > V_{ov}$	pinched off at the drain end	a current source $I(V_{GS})$

Cutoff and subthreshold are the same region seen on two different axes, which is why textbooks merge them. It's still worth keeping them apart in your head, because cutoff is what you assume when you do digital logic, and subthreshold is what you count when you do a power budget.

You'll draw these plots and read these numbers yourself in Lab 03, Parts 1, 3 and 4.

Check yourself

1. A device has V_{GS} = 1.2 V, V_{TH} = 0.6 V and V_{DS} = 0.3 V. Which region is it in? What if V_{DS} = 1.0 V?
2. Using the subthreshold slope of 85.6 mV/decade, by roughly what factor does I_D change when V_{GS} goes from 0.30 V to 0.39 V?

3. Why does a lower threshold voltage make a chip faster and leakier at the same time?

Next: Where saturation starts.

11. Where saturation starts

The textbook puts the boundary between triode and saturation at $V_{DS} = V_{GS} - V_{TH}$, but the knee in a measured curve lands somewhere else. This page shows why, names the effect responsible, and puts a number on it.

Along the way you'll pick up the two most useful habits in device work: **extract a parameter from a curve instead of looking it up**, and **compare your extraction with the model's own value**, so that you can see how much your choice of definition mattered.

Measuring V_{TH}

Nothing on the device tells you V_{TH}, and the model card doesn't hold it as a single number either, because the BSIM model computes it afresh at every operating point from the geometry and the bias. So engineers *measure* it, by drawing a construction on a measured curve. Here are two common constructions.

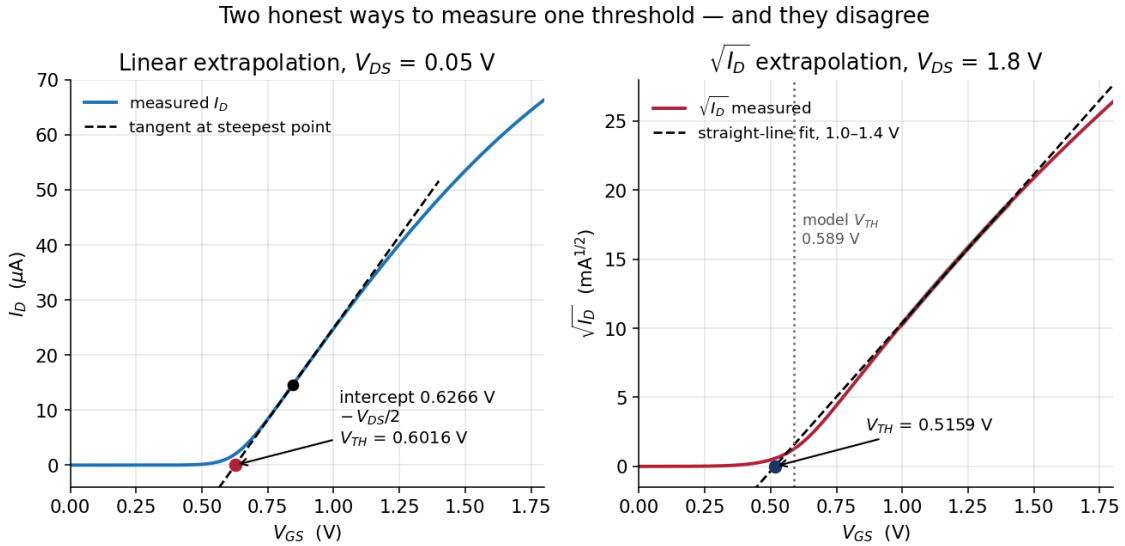


Figure 11.1: Two threshold extractions side by side: linear extrapolation and sqrt(I_D) extrapolation

Both panels show a SKY130 `nfet_01v8` *with W = 5 µm and L = 1 µm, simulated with* `spice/id_vgs.spice` *in the Lab 03 package.*

Method 1: linear extrapolation at small V_{DS}

Hold the drain at 0.05 V, which keeps the device firmly in triode where it behaves like a resistor, and sweep the gate. In triode with a small V_{DS}, the current is very nearly a straight line in V_{GS}:

$$I_D \approx \mu_n C_{ox} \frac{W}{L} \left(V_{GS} - V_{TH} - \frac{V_{DS}}{2} \right) V_{DS} \tag{11.1}$$

Find the steepest point of the measured curve, draw the tangent there, extend it down to $I_D = 0$, and read off where it crosses. According to the equation, that intercept is $V_{TH} + V_{DS}/2$, so you subtract half the drain voltage to get V_{TH}. Lab 03's extraction prints each step of the working:

```
== V_TH by linear extrapolation   (V_DS = 0.05 V)
   steepest point      V_GS = 0.845 V
   tangent slope       66.716 uA/V
   tangent hits zero   V_GS = 0.6266 V
   minus V_DS/2        V_TH = 0.6016 V
```

That gives **0.6016 V** from four lines of arithmetic on 361 sampled points, without a datasheet.

Method 2: $\sqrt{I_D}$ extrapolation in saturation

The saturation law says $I_D \propto (V_{GS} - V_{TH})^2$, so $\sqrt{I_D}$ should be a straight line whose x-intercept is V_{TH}. Sweep the gate with the drain at 1.8 V, take the square root of the current, fit a line over a sensible window, and extrapolate it to zero:

```
== V_TH by sqrt(I_D) extrapolation   (V_DS = 1.8 V, fit 1.0-1.4 V)
   sqrt(I_D) slope     21.4889 mA^0.5 / V
                       V_TH = 0.5159 V
```

This method gives **0.5159 V**.

Two methods, 86 mV apart

```
== your number vs the model's own number   (device A)
   quantity                              yours    ngspice   diff
   V_TH  (linear extrapolation)         0.6016     0.5895   V   +2.07 %
   V_TH  (sqrt extrapolation)           0.5159     0.5895   V  -12.49 %
```

The `ngspice` column is `@m.xma.msky130_fd_pr__nfet_01v8[vth]`, the threshold that BSIM itself reports at that operating point, read from `spice/op_params.spice`. So two reasonable methods and the model, all looking at one device, give three answers spread over 86 mV.

All three are legitimate, because **V_{TH} is defined by its extraction method**. The methods disagree because the square law they both rely on isn't exactly true for this device. Method 1 relies on it only very near threshold, where it is nearly right, and lands within about 2 % of the model. Method 2 relies on it across a whole volt of overdrive, where it is well off, and comes out 12 % low.

> **Ask how a threshold was extracted.** "0.52 V" and "0.60 V" can describe the same transistor, so a datasheet threshold with no method beside it has a tolerance you can't see.

For the rest of this course we take $V_{TH} = 0.6016$ V for this device, because that is what the linear extrapolation gives, and every prediction below is checked against it.

The measured knee

Take the five output curves and find where each one stops rising. Lab 03's `src/mosfit.py` uses one explicit rule, and says which rule it is: **the knee is the first V_{DS} at which the slope has fallen to 10 % of the slope at the origin.** The 10 % is our choice rather than anything physical, and what matters is that the same rule is applied to every curve.

```
   knee of each curve vs the model's vdsat
     V_GS  V_GS - V_TH  your knee     vdsat
     0.60       -0.002       0.09     0.065
     0.90        0.298       0.29     0.256
     1.20        0.598       0.50     0.442
     1.50        0.898       0.70     0.612
     1.80        1.198       0.89     0.779
```

Read the second and third columns together, from top to bottom.

At $V_{GS} = 0.9$ V the textbook boundary predicts a knee at 0.298 V, and the curve knees at **0.29 V**, so at low overdrive the simple theory is right to two digits.

At $V_{GS} = 1.8$ V the boundary predicts 1.198 V, but the curve knees at **0.89 V**, which makes the theory 35 % higher than the measured knee. Taken the other way round, with V_{ov} as the base, the knee is 26 % lower than the theory. The error grows steadily the harder you drive the gate.

Velocity saturation

The square law rests on an assumption that is rarely stated: that the drift velocity of the electrons is proportional to the electric field, without limit, so that doubling the field doubles their speed.

In silicon that holds only up to roughly 10^4 V/cm. Above that the carriers stop speeding up, because they scatter off the crystal lattice as fast as the field can accelerate them, and they settle at a **saturation velocity** of about 10^7 cm/s. Once the carriers in the channel are moving as fast as they can, more drain voltage can't push any more current through. This is **velocity saturation**.

As a result, the current doesn't wait for V_{DS} to reach the overdrive $V_{ov} = V_{GS} - V_{TH}$ before it stops responding. It stops as soon as the field along the channel is high enough, and for $L = 1$ μm and a volt of overdrive, that happens at a lower drain voltage. The model reports the drain voltage where it actually happens as **`vdsat`**, and `vdsat` is below V_{ov} at every one of the five bias points above.

There are two consequences you'll use constantly:

- **A short device saturates earlier**, because the same overdrive across a shorter channel makes a higher field. At $V_{GS} = 1.8$ V, `spice/op_params.spice` gives `vdsat = 0.912` V for L = 2 µm and **`vdsat = 0.362` V** for L = 0.15 µm. The minimum-length device pinches off at 40 % of the drain voltage the long one needs.
- **The square law drifts toward a straight line.** When the device is fully velocity-saturated, I_D becomes proportional to V_{ov} rather than V_{ov}^2. That is why Method 2 under-read the threshold by 12 %: it fitted a square root to a curve that was drifting toward linear.

> **When a square-law calculation is tens of percent out** on a short modern device, suspect the square law before your arithmetic. Simulate the circuit, and use the hand calculation for *direction* ("wider means more current") and the simulator for *magnitude*.

Channel-length modulation

The right-hand end of the $V_{GS} = 1.8$ V curve isn't horizontal. The current rises from **682.502 µA** at $V_{DS} = 1.2$ V to **696.275 µA** at 1.8 V, which is 2.0 % more current for 0.6 V more on the drain.

You can see the reason in panel 4 of the channel cross-section. As V_{DS} grows, the pinch-off point moves further from the drain, so the conducting part of the channel gets slightly **shorter**, and a shorter channel carries more current. This is **channel-length modulation**, and it is what the $(1 + \lambda V_{DS})$ factor in your textbook approximates.

The slope of the curve in this region is called the **output conductance** g_{ds}, and its reciprocal is the **output resistance** r_o:

Device	g_{ds} at $V_{GS} = V_{DS} = 1.8$ V	$r_o = 1/g_{ds}$
W = 5 µm, **L = 1 µm**	15.593 µS	**64.13 kΩ**
W = 1 µm, **L = 0.15 µm**	51.415 µS	**19.45 kΩ**

Both rows are from `spice/op_params.spice`. They mean that a saturated transistor isn't an ideal current source: it behaves like one with 64 kΩ in parallel, or 19 kΩ if you use the minimum-length device.

That resistance sets how much voltage gain one transistor can give, and it is the first thing Transconductance and output resistance uses.

You'll reproduce every number on this page in Lab 03, Parts 2 and 3.

Check yourself

1. Method 1 subtracts $V_{DS}/2$ from the tangent's intercept. Where does that term come from?
2. Why does Method 2 give a lower V_{TH} than Method 1 on this device?
3. At V_{GS} = 1.5 V, how far is the measured knee below $V_{GS} - V_{TH}$, in percent?
4. You shorten a device from L = 1 µm to L = 0.15 µm at the same bias. Does `vdsat` go up or down, and does r_o go up or down?

Next: Lab 03: MOSFET regions.

12. Lab 03: MOSFET regions

You'll sweep a real SKY130 `nfet_01v8` and find cutoff, triode, saturation and subthreshold on your own curves. You'll extract its threshold voltage two ways and see them disagree by 86 mV, measure g_m as the slope of a curve, and test whether current really scales with W/L. Then you'll add a sixth gate voltage to the sweep deck yourself.

Package: `labs/lab-03-mosfet-regions/`

Before you start

- Finish Lab 02.
- Read How a MOSFET works, The four regions and Where saturation starts.
- Open a terminal in the workbench and go to the lab folder:

```
cd /foss/designs/modules/ad103/labs/lab-03-mosfet-regions
```

The `Makefile` sets `PDK=sky130A` itself, so no other setup is needed. For the optional XSchem part at the end, load the environment first (`. /foss/designs/common/.designinit`).

Time: about 2.25 hours, not counting the optional Part 9. `make` takes about 50 seconds; the other targets take a few seconds each once `make` has run, or about ten seconds when they run a deck of their own.

The equations from the guide pages, for reference:

$$I_D^{\text{tri}} = \mu_n C_{ox} \frac{W}{L}\left[(V_{GS} - V_{TH})V_{DS} - \frac{1}{2}V_{DS}^2\right] \qquad I_D^{\text{sat}} = \frac{1}{2}\mu_n C_{ox}\frac{W}{L}(V_{GS} - V_{TH})^2 \tag{12.1}$$

with the boundary at $V_{DS} = V_{ov} = V_{GS} - V_{TH}$. The two slopes are

$$g_m = \frac{\partial I_D}{\partial V_{GS}} \qquad r_o = \left(\frac{\partial I_D}{\partial V_{DS}}\right)^{-1} \tag{12.2}$$

and below threshold, where the square law doesn't apply, the subthreshold slope $S = \partial V_{GS}/\partial(\log_{10} I_D)$, in mV/decade, describes the exponential.

Part 1: Run the sweeps

Does your workbench reproduce the reference sweeps?

1. Run the four sweeps and the operating-point dump, and check the results:

```
make
```

```
== sweeping spice/id_vds.spice
   wrote results/id_vds.txt
== sweeping spice/id_vgs.spice
   wrote results/id_vgs.txt
== sweeping spice/id_vgs_log.spice
   wrote results/id_vgs_log.txt
== sweeping spice/wl_sweep.spice
   wrote results/wl_sweep.txt
== operating point   spice/op_params.spice
   wrote results/op_params.log
== checking
```

```
  YOUR NUMBERS
  ok  I_D  from id_vds.txt, V_GS=1.8 V     696.2755 uA      (reference 696.2750)
  ok  I_D  W=5 L=1, V_GS=V_DS=1.8 V        696.2755 uA      (reference 696.2750)
  ok  I_D  W=10 L=1, same bias            1395.3199 uA      (reference 1395.3200)
  ok  I_D  W=5 L=2, same bias              380.7284 uA      (reference 380.7280)
  ok  I_D  W=1 L=0.15, same bias           501.0462 uA      (reference 501.0460)
  ok  V_TH by linear extrapolation           0.6016 V       (reference 0.6016)
  ok  V_TH by sqrt(I_D) extrapolation        0.5159 V       (reference 0.5159)
  ok  g_m at V_GS = 1.8 V                  914.6500 uS      (reference 914.6500)
  ok  subthreshold slope                    85.5629 mV/dec  (reference 85.6000)

PASS  all nine extracted parameters match the reference run
```

If any line says `XX`, `check.py` tells you which one and lists the four things that most often cause it, most frequent first.

2. Draw the plots into `results/`:

```
make figures
```

It writes six PNGs plus the channel cross-section figure. Open them as you go through the parts below.

3. Print every extracted parameter with its working. You'll read this output block by block in Parts 2 to 6:

```
make extract
```

You should see `PASS  all nine extracted parameters match the reference run`, and seven PNG files in `results/`.

Part 2: Extract the threshold two ways

What is V_{TH} for this device?

1. Open `results/ad103-vth-extraction.png`:

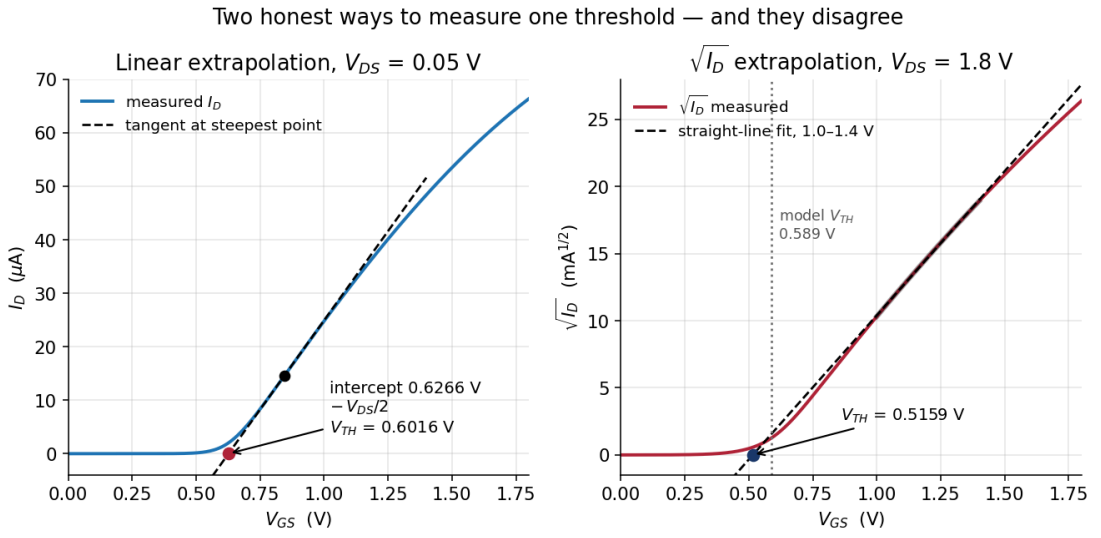


Figure 12.1: Two threshold extractions side by side

2. Find the two `V_TH` blocks in the `make extract` output: linear extrapolation at V_{DS} = 0.05 V, and $\sqrt{I_D}$ extrapolation at V_{DS} = 1.8 V.

3. Find the model's own threshold for the same device in the operating-point log:

```
grep -i 'xma.*\[vth\]' results/op_params.log
```

`xma` is device A in `spice/op_params.spice`, the same W = 5 µm, L = 1 µm transistor. Compare all three with the `your number vs the model's own number` block near the end of the `make extract` output.

You should see 0.6016 V by linear extrapolation, **0.5159 V** by $\sqrt{I_D}$ extrapolation and **0.5895 V** from the model. That's three different answers for one device, spread over 86 mV. Use 0.6016 V as V_{TH} for the rest of this lab.

Which of the three is the right threshold, and why do they disagree?
Answer: A1, in Answers at the end.

Part 3: The output characteristic

Where does each output curve knee, and does it land at $V_{GS} - V_{TH}$?

1. Open `results/ad103-id-vds-family.png`:

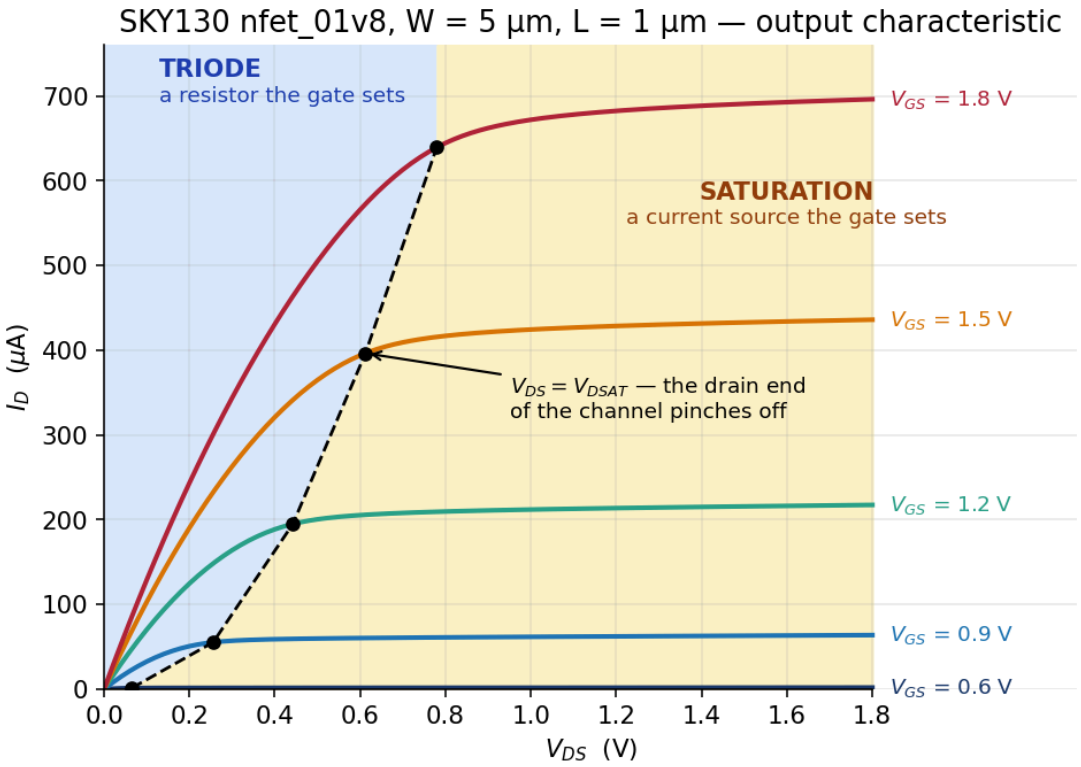


Figure 12.2: I_D vs V_DS for five gate voltages, with the regions shaded

2. Read the first block of `make extract`, `I_D vs V_DS`. Look first at the `channel R at V_DS=10 mV` column, which gives the channel's resistance at each gate voltage.
3. Work out $V_{GS} - V_{TH}$ for the 0.9 V to 1.8 V curves, using your V_{TH} of 0.6016 V, and compare it with the `knee V_DS` column. Then find the `knee of each curve vs the model's vdsat` table at the end of the output.
4. Find the lowest current in the `below threshold` block: V_{GS} = 0 V, with 1.8 V on the drain.

You should see the channel resistance fall from **26.6 kΩ** to **737 Ω** as V_{GS} goes from 0.6 V to 1.8 V. At V_{GS} = 0.9 V the knee lands at **0.29 V** against a predicted 0.298 V, which is close, but at 1.8 V it lands at **0.89 V** against a predicted 1.198 V, so there the prediction is 35 % high. The model's `vdsat` is below $V_{GS} - V_{TH}$ at every gate voltage. Even with the gate at 0 V, the device still passes **2.1853 pA**.

Why does the 1.8 V curve knee so far below $V_{GS} - V_{TH}$?
Answer: A2, in Answers at the end.

Part 4: Subthreshold conduction

Is a transistor below threshold really off?

1. Write down what you predicted for I_D at V_{GS} = 0.30 V when you read The four regions. If you skipped the prediction, make one now.
2. Read the `below threshold` block of `make extract`, and open `results/ad103-subthreshold.png`:

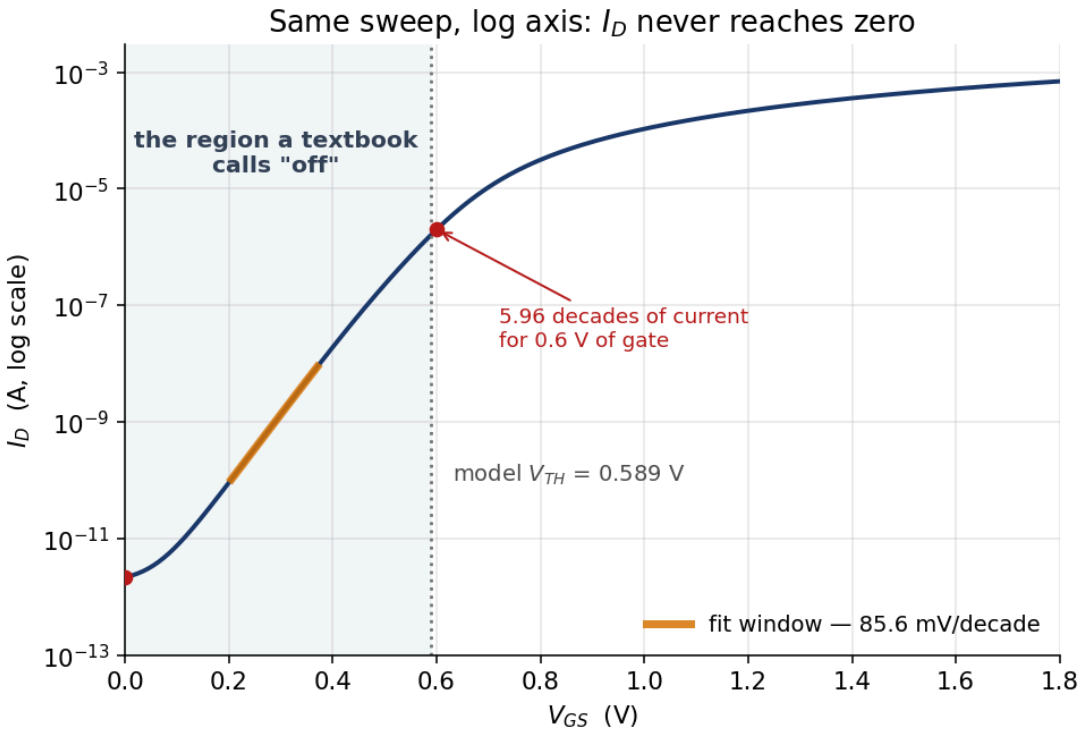


Figure 12.3: I_D vs V_GS on a log axis

3. Multiply the V_{GS} = 0 V current by ten million transistors. Then do the same with the V_{GS} = 0.30 V current, which is what a device with a threshold 0.3 V lower would carry with its gate at ground.

You should see 1.3645 nA at V_{GS} = 0.30 V, which is small but not zero. On the log plot it sits on a straight line **5.96 decades** tall, with a slope of **85.6 mV/decade**. Ten million transistors at 2.1853 pA each add up to 21.9 µA of leakage, and the same ten million with a threshold 0.3 V lower draw **13.6 mA**.

The two leakage totals are a factor of 624 apart. What does that say about how a process has to treat its threshold voltage?

Answer: A3, in Answers at the end.

Part 5: Measure g_m two ways

Does the slope of your sweep agree with the g_m the model reports?

1. Open `results/ad103-gm.png`:

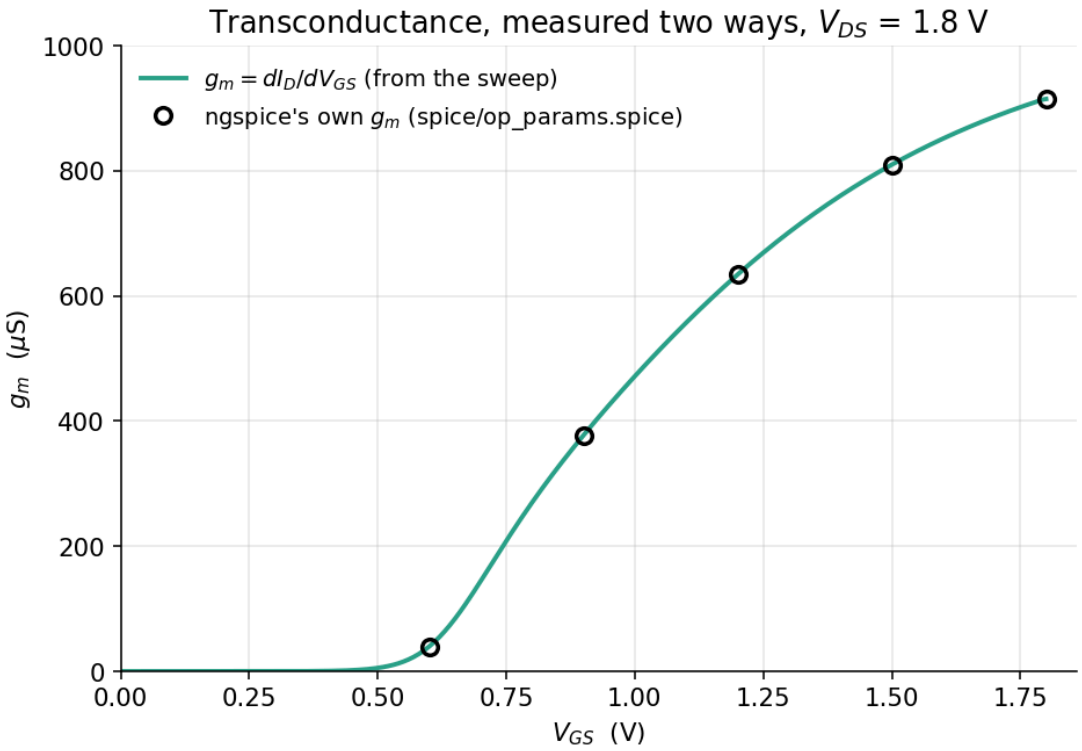


Figure 12.4: g_m from the sweep, with ngspice's own g_m overlaid

The green curve is `numpy.gradient` applied to your sweep. The circles are the g_m that ngspice's model reports at five bias points. The two are computed by completely different means, so each is an independent check on the other.

2. Read the `g_m = dI_D/dV_GS` block of `make extract`, and the `g_m` line in the `your number vs the model's own number` block.

You should see 914.650 µS from the slope of your sweep against **915.312 µS** from the model at V_{GS} = 1.8 V, **0.07 %** apart. In the `g_m/I_D` column, the value falls from 5.90 /V at V_{GS} = 0.9 V to 1.31 /V at 1.8 V.

What does g_m/I_D measure, and what does its trend say about where to bias an amplifier?
Answer: A4, in Answers at the end.

Part 6: Four device geometries at one bias

Does current scale with W/L?

The `W/L is a knob` block puts four devices at $V_{GS} = V_{DS}$ = 1.8 V:

	W	L	W/L
A	5 µm	1 µm	5
B	10 µm	1 µm	10
C	5 µm	2 µm	2.5
D	1 µm	0.15 µm	6.67

1. Before you look, use W/L alone to predict the currents of B, C and D relative to A, which carries 696.275 µA. Write the three numbers down.
2. Read the `W/L is a knob` block of `make extract`, and open `results/ad103-wl-sweep.png`:

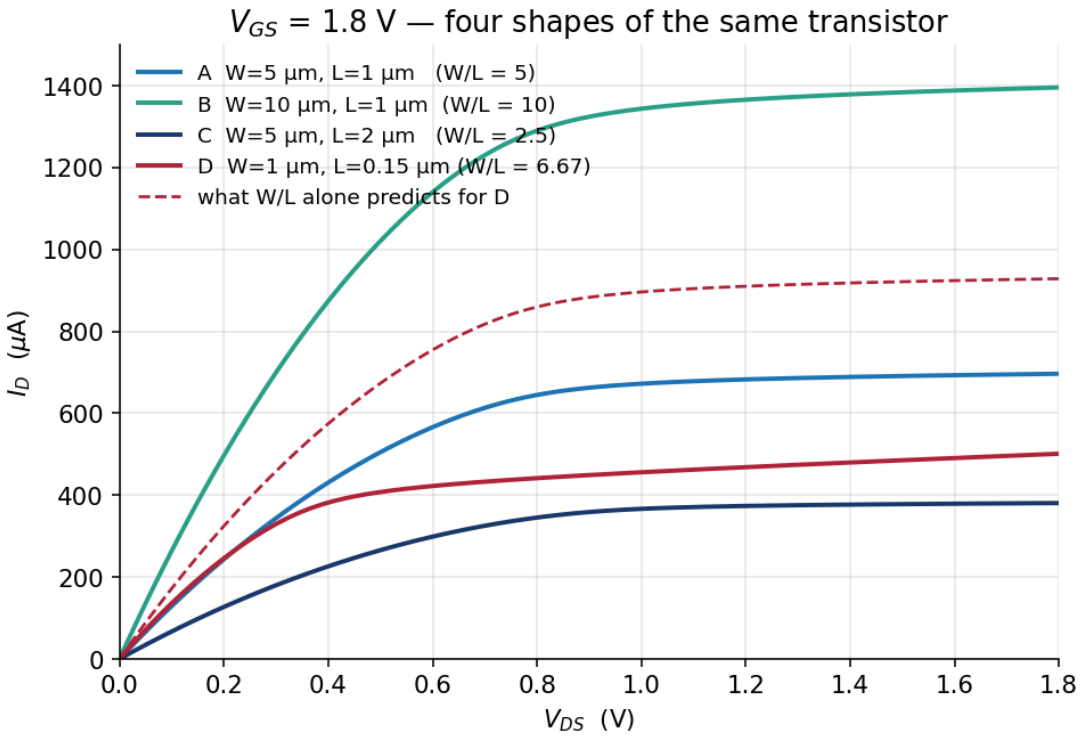


Figure 12.5: Four device geometries at the same bias

3. For each device, work out how far the measurement is from your prediction, in percent.

You should see B within **0.20 %** of twice A's current, so width scales the way W/L says it should. Doubling L gives 9.4 % more current than $1/L$ predicts, and the minimum-length device D delivers **501.046 µA** against a predicted 928.367 µA, only **54 %** of the prediction. Sizing with W and L, which you'll read before Lab 04, closes that gap in two steps.

Part 7: Units on `W` and `L`

What does a unit suffix on `W` do to this deck?

1. Predict what ngspice will print. You made the same mistake on purpose in Lab 01, Part 6.
2. Run the deck with units on `W` and `L`:

```
make wrong-units
```

```
Error on line 56 or its substitute:
   m.xm1.msky130_fd_pr__nfet_01v8 d g 0 0 xm1:sky130_fd_pr__nfet_01v8__model l=
1.000000000000000e-06     w=    5.000000000000000e-06 ...
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!
```

3. Find what changed:

```
diff spice/id_vds.spice spice/id_vds_wrong_units.spice
```

You should see that the two decks differ by two characters, the two `u`s in `L=1u W=5u` where the working deck has `L=1 W=5`, and that the error message names neither `W` nor units.

Why does a `u` make ngspice stop here, when the same `u` on a diode in Lab 02 ran silently?
Answer: A5, in Answers at the end.

Part 8: Add a sixth gate voltage

Can you add a curve to `spice/id_vds.spice` so that the analysis picks it up?

The deck sweeps V_{DS} once for each gate voltage in its `foreach` list. You'll add a curve at V_{GS} = 1.05 V, which sits between the 0.9 V and 1.2 V curves. It takes four edits.

1. Predict I_D at V_{GS} = 1.05 V and V_{DS} = 1.8 V from the 0.9 V and 1.2 V rows of the first `make extract` table. Write it down.
2. Open `spice/id_vds.spice` in a text editor and add `1.05` to the `foreach` line as the second entry, right after `0.6`. We have you put it there rather than at the end on purpose: at the end, the existing `let` lines would keep their numbers, and renumbering them in step 3 is the part worth practising.
3. Each `dc` command makes its own plot, named `dc1`, `dc2`, … in the order the sweeps ran. Add a `let` line for the new curve (call it `id_vgs105`), and fix the `dcN` numbers on the other `let` lines to match the new order.

   Answer: A6, in Answers at the end.
4. Add `id_vgs105` to the `wrdata` line, and save the file. Its position in that list doesn't matter, because `extract.py` matches each column to its gate voltage through the vector's name.
5. Run the lab and print the extraction again:

```
make
make extract
```

The first table of `make extract` now has six rows. The `gate voltages read` line lists the gates in the order of your `wrdata` line, so yours may differ from the one below, which had `id_vgs105` second. The table under it is sorted by gate voltage either way:

```
== I_D vs V_DS   (W=5, L=1)
   gate voltages read from spice/id_vds.spice: 0.6 1.05 0.9 1.2 1.5 1.8
     V_GS  I_D at V_DS=1.8   knee V_DS   channel R at V_DS=10 mV
     0.60         2.005 uA       0.09 V              26648.3 ohm
     0.90        63.876 uA       0.29 V               2529.2 ohm
     1.05       131.053 uA       0.39 V               1690.0 ohm
     1.20       217.505 uA       0.50 V               1288.0 ohm
     1.50       436.080 uA       0.70 V                907.7 ohm
     1.80       696.275 uA       0.89 V                736.9 ohm
```

You should see a sixth row, **131.053 µA at V_{GS} = 1.05 V**, sorted into gate-voltage order, and `make` still printing `PASS`. That works because nothing in `src/` assumes five curves; `extract.py` reads the gate list back out of the deck. The reference deck is `solutions/id_vds_six.spice`, and `solutions/README.md` discusses it.

Part 9 (optional): Draw it in XSchem

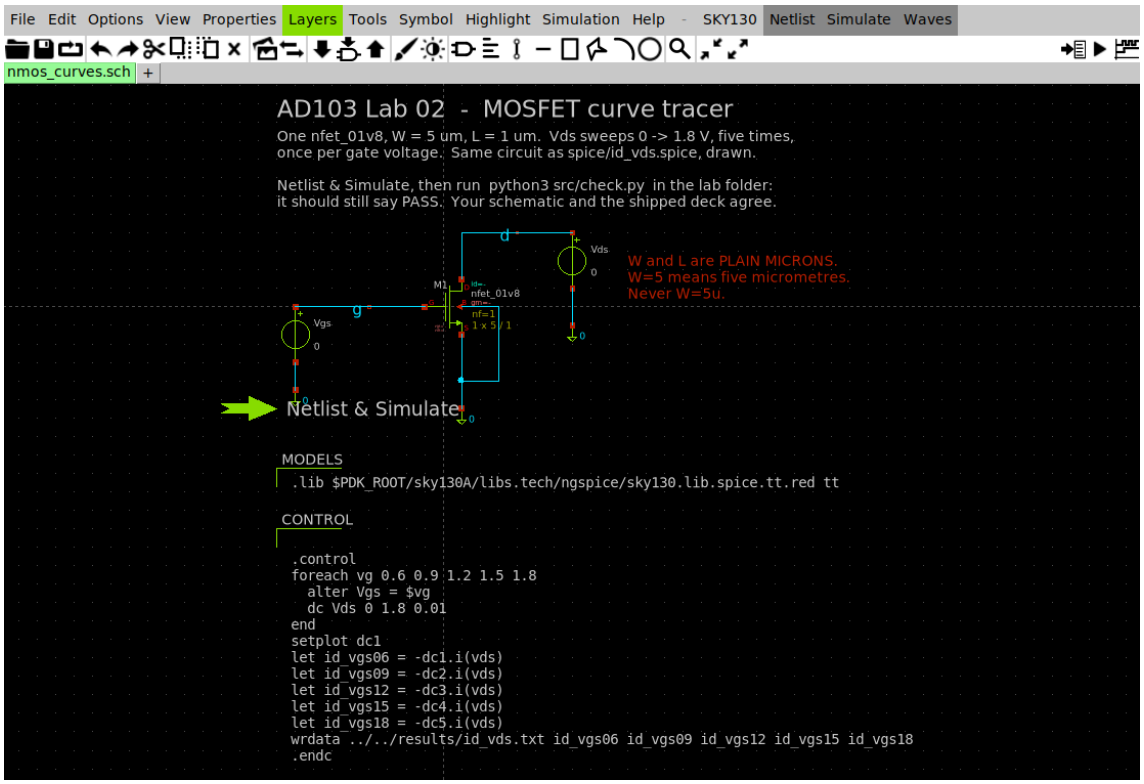


Figure 12.6: The Lab 03 curve tracer open in XSchem

`xschem/nmos_curves.sch`: the same circuit as the shipped `spice/id_vds.spice`, drawn. The `.lib` line and the `.control` block are carried in two `code_shown` symbols, which is how a testbench keeps its simulation commands next to its circuit.

1. Do this part after you've finished Part 8 and checked its line in Finishing, because this step throws your Part 8 edit away. The schematic writes five columns, and `check.py` reads the gate list from the deck, so with your six-curve deck in place it reports `the results file is stale`. Put `spice/id_vds.spice` back to five curves (your course folder is the git clone from Lab 01's `mod add`):

```
git checkout -- spice/id_vds.spice
make
```

2. Open the schematic:

```
cd xschem
PDK=sky130A xschem nmos_curves.sch &
```

3. Find the green **Netlist & Simulate** launcher arrow on the canvas. It isn't a toolbar button, and a plain left-click on it only selects it, without netlisting, simulating or printing any message. **Click the arrow once to select it, then press `Ctrl-H`** to fire it. (Ctrl-click on the arrow also works, but it misses if your hand drifts between press and release. The XSchem cheat sheet has the same note.)

4. Back in the lab folder, rerun the checker on the data the schematic wrote:

```
cd ..
python3 src/check.py
```

You should see `PASS` again, because the schematic writes the same `results/id_vds.txt`. The drawn device carries **696.226 µA** rather than 696.275 µA, because the XSchem symbol adds the diffusion resis-

tances `nrd` and `nrs` that the hand-written deck leaves out. Those resistances come from how the device is laid out, so the 50 nA difference is layout showing up in an electrical answer.

Finishing

You're done when `make extract` shows your sixth row, 131.053 µA at V_{GS} = 1.05 V, and `make` still prints `PASS` with your edited deck (Part 8).

Questions

1. What are the three threshold values for this device, and why do they differ? (Part 2.)
2. At V_{GS} = 1.8 V, where does the curve knee, and why is it below $V_{GS} - V_{TH}$? (Part 3.)
3. Which of B, C and D in Part 6 matched your W/L prediction, and which missed?

   Answer: A7, in Answers at the end.

Submit

Nothing to submit.

Going further

Two more changes to the decks, with reference answers in `labs/lab-03-mosfet-regions/solutions/`. Try them before you open that folder.

1. **Repeat it for a PMOS.** Do the same measurements for `pfet_01v8` and find out how much of this lab is about *n*-channel devices specifically.
2. **Lengthen the short device.** Set device D in `spice/wl_sweep.spice` to `L=0.5`, rerun `make extract`, and watch `vth`, `vdsat` and I_D move together. Check whether the two-step arithmetic in Sizing with W and L still closes.

Next on the Course path: read What moves the threshold.

Answers

A1. Part 2: Extract the threshold two ways

None of them is wrong, because V_{TH} is defined by the method you use to extract it. The two extraction methods disagree because both rely on the square law, and the square law isn't exactly true for this device. Linear extrapolation relies on it only very near threshold, so it lands 2.07 % from the model. $\sqrt{I_D}$ extrapolation relies on it across a whole volt of overdrive, so it ends up 12.49 % out. Where saturation starts goes through this in more detail.

A2. Part 3: The output characteristic

The cause is velocity saturation. The square law assumes that carrier speed keeps rising with the electric field, but above roughly 10^4 V/cm the carriers stop speeding up. Once they're moving as fast as they can, more drain voltage adds nothing, so the current stops responding before V_{DS} reaches V_{ov}. The harder you drive the gate, the bigger the error, which is why the 0.9 V curve lands close to its prediction and the 1.8 V curve doesn't. See Where saturation starts.

A3. Part 4: Subthreshold conduction

A threshold 0.3 V lower multiplies the leakage of those ten million transistors by 624, from 21.9 µA to 13.6 mA. Because a shift that small changes a chip's leakage that much, a process has to hold its threshold voltage tightly, and threshold control is a discipline of its own.

A4. Part 5: Measure g_m two ways

g_m/I_D is the transconductance you get for each unit of current you burn, so it measures how efficiently the device uses its bias current. It's highest near threshold and lowest with the gate wide open, which is the opposite of where a beginner tends to bias an amplifier. Transconductance and output resistance, which you'll read before Lab 04, picks this up.

A5. Part 7: Units on W and L

SKY130′s MOSFET models are binned in microns: each model covers a range of widths and lengths, so a width with a `u` on it matches no bin, and ngspice stops. The diode models aren't binned, so there's no range for a wrong size to fall outside, and the diode deck runs without complaint.

A6. Part 8: Add a sixth gate voltage

The 1.05 V sweep now runs second, so it is `dc2`, and every curve after it moves down by one:

```
let id_vgs06 = -dc1.i(vds)
let id_vgs105 = -dc2.i(vds)
let id_vgs09 = -dc3.i(vds)
let id_vgs12 = -dc4.i(vds)
let id_vgs15 = -dc5.i(vds)
let id_vgs18 = -dc6.i(vds)
```

The `N` in `dcN` counts sweeps in the order they ran, and has nothing to do with the gate voltage.

A7. Finishing

B, the wider device, matched to within 0.20 %. C, the longer one, carried 9.4 % more than $1/L$ predicts, and D, the minimum-length device, carried only 54 % of its prediction.

IV Device parameters

13 What moves the threshold 89
13 The fourth terminal 89
13 Body effect 89
13 The body-effect formula 89
13 Threshold and drawn length 90
13 Threshold and temperature 91
13 Check yourself 91

14 Transconductance and output resistance 93
14 Measuring g_m 93
14 The square-law estimate 93
14 g_m/I_D: how designers size devices 94
14 r_o and intrinsic gain 95
14 Where these numbers go next 95
14 Check yourself 96

15 Sizing with W and L 97
15 Four shapes at one bias 97
15 Device B: width 98
15 Device C: length 98
15 Device D: minimum length 99
15 What geometry costs 100
15 Check yourself 100

16 Lab 04: Testing the W/L rule 101
16 Before you start 101
16 Part 1: Write your predictions 101
16 Part 2: Move the threshold 102
16 Part 3: Efficiency and intrinsic gain 103
16 Part 4: Run the lab and score your predictions . 103
16 Part 5: Width 104
16 Part 6: Length 106
16 Part 7: Five devices with $W/L = 5$ 107
16 Part 8: Read a FAIL you caused 107
16 Finishing 108
16 Going further 108
16 Answers 108

13. What moves the threshold

In Lab 03 you extracted $V_{TH} = 0.6016$ V for the W = 5 µm, L = 1 µm transistor. A second method gave 0.5159 V, and BSIM's own value was 0.5895 V, so the number you call V_{TH} **depends on the method you used to extract it**.

This page covers the other half of the story. Even with the method fixed, the *same transistor* has a different threshold depending on three things you have been holding still so far: the voltage between its source and bulk, the length you drew, and the temperature.

The fourth terminal

Look at any device line in this course:

```
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=1 W=5
```

It names four nodes, in the order **drain, gate, source, bulk**. On every deck since Lab 01 you have wired the last two to the same place, node 0, so the voltage between them has always been zero.

The **bulk** (also called the *body* or *substrate*) is the silicon the transistor is built in. For an NMOS in this process that's the p-type wafer itself, which every other NMOS on the chip shares. You don't usually get to choose its voltage, because it's tied to ground.

What you *can* change, sometimes without meaning to, is the **source** voltage. If you stack two NMOS transistors in series, as in the bottom half of a two-input NAND gate, the upper one's source sits on the lower one's drain, somewhere above ground. Its bulk is still at 0, so for that device $V_{SB} > 0$.

Body effect

The Lab 03 deck `spice/vth_body.spice` uses the same W = 5 µm, L = 1 µm device with the source lifted onto its own node. V_{GS} and V_{DS} are held fixed *relative to the source* (V_{GS} = 1.8 V, V_{DS} = 0.05 V), so the only thing that changes is V_{SB}:

V_{SB}	V_{TH}	shift	I_D
0 V	0.5894596 V	—	66.3374 µA
0.3 V	0.6607482 V	**+71.289 mV**	61.6210 µA
0.6 V	0.7213725 V	**+131.913 mV**	57.7533 µA
0.9 V	0.7745817 V	**+185.122 mV**	54.4634 µA
1.2 V	0.8222507 V	**+232.791 mV**	51.5967 µA

Lifting the source 1.2 V above the bulk moved the threshold by **232.791 mV**, which is nearly a quarter of a volt on a supply of only 1.8 V, and it cost **22.2 %** of the drain current at the same gate drive, even though nothing was rewired and no dimension changed.

This is the **body effect**. It's the reason a two-input NAND gate is slower than an inverter by more than the extra transistor's capacitance accounts for. Because the upper device in the stack doesn't have its source at ground, it works against a threshold a hundred-odd millivolts higher than the one on its datasheet.

The body-effect formula

The standard formula, which your device-physics course will give you, is

$$V_{TH}(V_{SB}) = V_{TH0} + \gamma\left(\sqrt{2\phi_F + V_{SB}} - \sqrt{2\phi_F}\right) \tag{13.1}$$

Here γ is the **body-effect coefficient**, in $\sqrt{\text{V}}$, and $2\phi_F$ is roughly twice the Fermi potential of the substrate. Both are set by how heavily the silicon is doped, and neither is printed anywhere easy to read, so instead you can fit them to the five measured points, which also tells you whether the formula is any good.

Take $V_{TH0} = 0.5894596$ V from the $V_{SB} = 0$ row, then do a least-squares fit of the other four points for γ and $2\phi_F$:

$$\gamma = 0.4108\ \sqrt{\text{V}} \qquad 2\phi_F = 0.600\ \text{V} \tag{13.2}$$

V_{SB}	measured	formula	difference
0 V	0.5895 V	0.5895 V	—
0.3 V	0.6607 V	0.6610 V	**−0.23 mV**
0.6 V	0.7214 V	0.7213 V	**+0.11 mV**
0.9 V	0.7746 V	0.7744 V	**+0.20 mV**
1.2 V	0.8223 V	0.8224 V	**−0.15 mV**

With two parameters fitted to five points, the worst error is **0.23 mV**. That's four significant figures of agreement between a formula from the 1960s and a BSIM4 model with several hundred parameters.

Compare that with the last two pages, where the square law missed the saturation knee by 35 % and the $\sqrt{I_D}$ extraction missed the threshold by 12 %. Here a formula from the same era lands within a quarter of a millivolt. The difference is that the square law relies on physics that short channels changed, while the body-effect formula rests on the depletion charge under the channel, which doesn't depend on how long the channel is. That's why the square-root law survived where the square law didn't.

> **When a hand formula disagrees with a simulator,** ask which assumption of the formula the device violates. If you can't name one, suspect your arithmetic.

Threshold and drawn length

Shortening a device raises its current, and it also moves its threshold, in a direction few people predict. The Lab 03 deck `spice/vth_vs_l.spice` has nine copies of the same W = 5 µm transistor at nine lengths, all at V_{DS} = 50 mV so that the drain doesn't influence the answer:

```
L        model vth
0.15 um    7.078517e-01 V
0.2 um     6.872240e-01 V
0.3 um     6.553824e-01 V
0.5 um     6.300670e-01 V
0.8 um     6.003750e-01 V
1 um       5.894596e-01 V
1.5 um     5.627056e-01 V
2 um       5.495175e-01 V
4 um       5.364102e-01 V
```

Most textbooks introduce **threshold roll-off**: as a channel gets shorter, the source and drain depletion regions take over some of the charge the gate would otherwise have to support, the gate's job gets easier, and V_{TH} **falls**. These nine devices go the other way. Their threshold rises monotonically as the channel gets shorter, from **0.5364102 V** at L = 4 µm to **0.7078517 V** at L = 0.15 µm, a span of **171.44 mV**, so the shortest device has the *highest* threshold.

This is the **reverse short-channel effect**, and it comes from a manufacturing decision rather than from unavoidable physics. Processes at this node implant extra p-type dopant in pockets right beside the source and drain, called **halo** or **pocket** implants, specifically to fight roll-off, because a short device whose threshold collapses also leaks. In SKY130 the halo over-corrects: the average doping under a short channel ends up *higher* than under a long one, so the gate has more charge to support rather than less.

This has two practical consequences. The first is that you can't shorten a transistor and keep its bias point, because 171 mV of threshold is most of an overdrive voltage. The second is that analog designers

who need two transistors to behave identically always give them **the same length**, even when the equations say only W/L matters. In Lab 04 you'll see five devices with identical W/L carry five different currents, and this is one of the reasons.

Threshold and temperature

The same `vth_body.spice` run ends by simulating the device at $V_{SB} = 0$ at three temperatures, spanning the automotive range that real parts are specified over:

T	V_{TH}	I_D
−40 °C	0.6499250 V	93.8168 µA
27 °C	0.5894596 V	66.3374 µA
125 °C	0.5010177 V	45.0650 µA

V_{TH} falls by **148.9 mV** across the range, a slope of **−0.9025 mV/°C**, which is close enough to the −1 mV/°C rule of thumb that you can simply remember the rule of thumb.

Before you read the current column, predict which way it goes. A lower threshold means more overdrive at the same gate voltage, so you might expect the hot device to carry *more* current.

In fact the hot device carries **32.1 % less** current than at 27 °C, and the cold one carries **41.4 % more**.

That's because temperature moves more than the threshold. Carrier **mobility** falls as the lattice heats up and scatters carriers more, roughly as $T^{-1.5}$, and at this overdrive the mobility effect wins comfortably. The two effects have opposite signs, and which one dominates depends on the bias. At very low overdrive, near threshold, the threshold term wins and the current goes *up* with temperature. The gate voltage at which the two cancel is called the **zero-temperature-coefficient point**, and circuits that must not drift are sometimes deliberately biased there.

> **"Does this get better or worse when it's hot?"** has no general answer for a MOSFET, so simulate both ends. This is the corner discipline AD102 drilled into you with resistors, and it matters more here, because a resistor's temperature coefficient doesn't change sign on you.

You'll run `vth_body.spice` and `vth_vs_l.spice` yourself in Lab 04, Part 2.

Check yourself

1. In a stack of two NMOS transistors, which one sees $V_{SB} > 0$, and does that make it stronger or weaker?
2. Using γ = 0.4108 √V and $2\phi_F$ = 0.600 V, estimate the threshold shift at V_{SB} = 0.6 V and compare it with the table.
3. Why does the halo implant raise the threshold of a short device more than a long one?
4. At this overdrive, the drain current falls when the device heats up even though V_{TH} falls. Which effect wins, and would the answer be the same just above threshold?

Next: Transconductance and output resistance.

14. Transconductance and output resistance

You'll often hear that a transistor amplifies. This page puts a number on how much it can amplify, and shows that what sets the number is the product $g_m r_o$.

So far you've measured currents, and each one answered the question "what does this device do at this bias?". An amplifier cares about something else: **how much the answer changes when you nudge the input.** That's a slope rather than a value, and a transistor has two slopes that matter, one for each terminal you can move.

Symbol	Name	It is the slope of	In words
g_m	**transconductance**	I_D vs V_{GS}	how hard the gate pushes current through the device
g_{ds}	**output conductance**	I_D vs V_{DS}	how much the drain voltage leaks through to the current
$r_o = 1/g_{ds}$	**output resistance**	—	the same slope, written as its reciprocal

g_m is measured in siemens because it's amps per volt, and it's called *trans*-conductance because the volts are at one terminal and the amps at another. That's the reason a transistor can have gain at all. A two-terminal device's conductance can only turn a voltage across it into the current through *itself*, but a transistor lets a voltage on the gate control the current through the drain.

Measuring g_m

g_m is the slope of the I_D vs V_{GS} curve you swept in Lab 03. When you differentiated that sweep in Lab 03, Part 5, you got:

```
== g_m = dI_D/dV_GS   (V_DS = 1.8 V)
   V_GS = 0.90 V   I_D =    63.876 uA   g_m =  376.932 uS   g_m/I_D =   5.90 /V
   V_GS = 1.20 V   I_D =   217.505 uA   g_m =  634.008 uS   g_m/I_D =   2.91 /V
   V_GS = 1.50 V   I_D =   436.080 uA   g_m =  809.693 uS   g_m/I_D =   1.86 /V
   V_GS = 1.80 V   I_D =   696.275 uA   g_m =  914.650 uS   g_m/I_D =   1.31 /V
```

and the same lab printed it next to the model's own value:

```
   g_m at V_GS = 1.8 V               914.6500   915.3117  uS   -0.07 %
```

Your 361-point sweep gives **914.650 µS**, and BSIM's analytic derivative gives **915.3117 µS**, so the two are only 0.07 % apart. Compare that with the threshold voltage, where two reasonable methods disagreed by 86 mV. The difference is that g_m is simply the slope of a measured curve, so there's no choice of definition to argue about.

The square-law estimate

Differentiating the square law gives a tidy expression for the slope:

$$I_D = \frac{1}{2}\mu_n C_{ox}\frac{W}{L}V_{ov}^2 \quad \Rightarrow \quad g_m = \frac{2I_D}{V_{ov}} \tag{14.1}$$

At $V_{GS} = 1.8$ V, with $V_{TH} = 0.6016$ V from Where saturation starts, the overdrive is 1.1984 V and I_D is 696.275 µA:

$$g_m = \frac{2 \times 696.275\ \mu\ \text{A}}{1.1984\ \text{V}} = 1162.0\ \mu\ \text{S} \tag{14.2}$$

The model gives **915.3 µS**, so the formula comes out **27 % high**.

Lower down, at $V_{GS} = 0.9$ V, the overdrive is only 0.2984 V:

$$g_m = \frac{2 \times 63.876\ \mu\ \text{A}}{0.2984\ \text{V}} = 428.1\ \mu\ \text{S} \tag{14.3}$$

The measured value there is **376.9 µS**, so at this lower bias the formula is only **13.6 %** high.

This is the same way the square law failed at the saturation knee, and it has the same cause, velocity saturation. As you push the overdrive up, the current stops growing quadratically, so its derivative stops being $2I_D/V_{ov}$. In the fully velocity-saturated limit $g_m = I_D/V_{ov}$, which at 1.8 V would predict 581 µS, 36 % low. The real value lies between those two limits, in a region that no closed-form formula covers.

> **Use $2I_D/V_{ov}$ as a sanity bracket rather than as an answer.** If your simulator says g_m is half or double that, look for a mistake. If it says 27 % less, that's a 130 nm transistor behaving normally.

g_m/I_D: how designers size devices

Look again at the last column of the g_m printout, set out here as a table:

V_{GS}	I_D	g_m	g_m/I_D
0.90 V	63.876 µA	376.932 µS	**5.90 /V**
1.20 V	217.505 µA	634.008 µS	**2.91 /V**
1.50 V	436.080 µA	809.693 µS	**1.86 /V**
1.80 V	696.275 µA	914.650 µS	**1.31 /V**

Between the first row and the last, the current goes up by a factor of **10.9**, but the g_m you bought with it goes up by a factor of only **2.4**. You can think of g_m/I_D as an exchange rate: it says how much transconductance one ampere of bias current buys you, and it falls steeply as you push the gate harder.

In the other direction, toward lower V_{GS}, it climbs. The Lab 03 deck `spice/gm_ro.spice` steps the same device down to $V_{GS} = 0.7$ V:

```
V_GS = 0.7 V
i(vdc) = -1.06419e-05
@m.xmc.msky130_fd_pr__nfet_01v8[gm] = 1.437123e-04
```

143.7123 µS from 10.6419 µA works out to **13.50 /V**, which is ten times the efficiency of the 1.8 V bias, using a sixty-fifth of the current.

The climb has a ceiling, though. Deep below threshold the drain current is exponential in V_{GS}, and the derivative of an exponential is proportional to the exponential itself, so g_m/I_D levels off at

$$\frac{g_m}{I_D}\Big|_{\max} = \frac{1}{nV_T} \tag{14.4}$$

Here $V_T = kT/q = 25.8649$ mV at 27 °C, and n comes from the subthreshold slope you measured in Lab 03, **85.6 mV/decade**, so $n = 85.6/(2.3026 \times 25.8649) = 1.437$. That puts the ceiling for this device at

$$\frac{1}{1.437 \times 0.0258649} = 26.9\ /\text{V} \tag{14.5}$$

At $V_{GS} = 0.7$ V you're at 13.50, **half the ceiling**, which is the textbook definition of **moderate inversion**. That's the region between the square law and the exponential, where most low-power analog circuits operate and where neither hand formula works.

The single number g_m/I_D captures the whole design trade. If you bias the device hard, you get speed and small devices, but poor current efficiency. If you bias it softly, you get more g_m per microamp, at the cost of area and bandwidth. Professional analog designers choose g_m/I_D **first** (10 to 15 /V for a low-noise amplifier, 5 for something that has to be fast) and let the geometry follow from it.

r_o and intrinsic gain

g_m tells you how much signal current you get for a given input. To turn that current back into a voltage you have to push it through a resistance, and the largest one available is the transistor's own r_o. It comes from the slope of the saturation region, which looks flat but, as Where saturation starts showed, isn't quite.

When you multiply the two, the units of current cancel and you're left with a voltage gain:

$$A_{v,\max} = g_m r_o = \frac{g_m}{g_{ds}} \tag{14.6}$$

This is the **intrinsic gain**, the most voltage gain one transistor can produce at a given bias, whatever you build around it. It's an upper limit because it assumes you loaded the transistor with something infinitely better than itself. `spice/gm_ro.spice` measures it at four lengths, each at $V_{GS} = V_{DS}$ = 1.8 V:

L	I_D	g_m	$r_o = 1/g_{ds}$	$g_m r_o$	in dB
0.15 µm	2644.190 µA	2468.806 µS	3.588 kΩ	**8.86**	18.9 dB
0.5 µm	1196.020 µA	1460.824 µS	24.008 kΩ	**35.07**	30.9 dB
1 µm	696.275 µA	915.312 µS	64.130 kΩ	**58.70**	35.4 dB
4 µm	197.974 µA	274.342 µS	386.764 kΩ	**106.11**	40.5 dB

Compare the two ends of the table. The minimum-length device has **9.0× the transconductance** of the 4 µm one, yet it gives only **one twelfth of the gain**, because its output resistance is 108 times lower. Channel-length modulation is the reason: when the drain voltage shortens a short channel slightly, that's a *large* fractional change in its length, so g_{ds} is large.

This is the most important trade in analog IC design, and it runs in the opposite direction to digital design. The digital track, DD101 through DD104, wants every transistor as short as the process allows, because short transistors are fast. An analog designer who needs gain deliberately draws long devices and pays for them in area and speed. Both use the same process and the same transistor, with opposite instincts about its length, and when the two teams share a die, this is what they argue about.

> **Intrinsic gain in the tens is normal** for a modern process. If a hand analysis promises a gain of 500 from one transistor, it used a value of λ from a textbook written when channels were microns long.

You'll run `spice/gm_ro.spice` yourself in Lab 04, Part 3.

Where these numbers go next

The inverter. Take an NMOS and a PMOS, tie their gates together and their drains together, and you have two transconductances driving one node, loaded by two output resistances in parallel:

$$A_v = -(g_{mn} + g_{mp})\left(r_{on} \parallel r_{op}\right) \tag{14.7}$$

Both devices push on the output, so their g_m values add, but their output conductances add as well, so the result is a weighted average of the two devices' own $g_m r_o$ rather than a sum. The CMOS inverter measures a gain of **−116.0341** at L = 0.5 µm, well above the 35.07 in the table above, because it works at a different bias from the table's, where each transistor's own $g_m r_o$ is about 116. That page has the numbers.

The mirror. A saturated transistor is very nearly a current source, and r_o tells you how nearly. An L = 1 µm device holding 50 µA changes its current by **1.46 %** when its drain moves from 0.9 V to 1.2 V. That device is the heart of the current mirror, and this page's r_o column sets the mirror's error budget.

Check yourself

1. Why can a transistor have voltage gain when a resistor can't?
2. At V_{GS} = 1.2 V, what does $2I_D/V_{ov}$ predict for g_m, using V_{TH} = 0.6016 V? Is the measured value above or below it?
3. You need a g_m of 1 mS from as little current as possible. Should you bias near threshold or with a large overdrive? What do you give up?
4. Going from L = 0.15 µm to L = 4 µm, g_m falls and $g_m r_o$ rises. Explain each in one sentence.

Next: Sizing with W and L.

15. Sizing with W and L

When you place a resistor you give it a resistance, and when you place a capacitor you give it a capacitance, but a transistor has no single value like that. Instead you choose two lengths, its width W and its length L.

Every equation in the last few pages has a factor of W/L in it, and that is the only factor in them you control. μ_n, C_{ox} and V_{TH} are set by the foundry and come fixed in the model file, and V_{GS} and V_{DS} are set by the rest of the circuit. W and L are the two numbers you type in, so designing a transistor comes down to choosing them. It's the lesson AD102 taught you about resistors, that on a chip a value is a geometry, applied to the device that matters most.

Four shapes at one bias

In Lab 03, Part 6, `spice/wl_sweep.spice` put four differently shaped transistors at the same bias, $V_{GS} = V_{DS} = 1.8$ V:

	W	L	W/L	why it is there
A	5 µm	1 µm	5	the reference device from the last few pages
B	10 µm	1 µm	10	twice as wide
C	5 µm	2 µm	2.5	twice as long
D	1 µm	0.15 µm	6.67	minimum length

If current scaled with W/L, then relative to A, B would carry 2.00× as much current, C would carry 0.50×, and D (whose W/L is 6.67 against A's 5) would carry 1.33×, or **928 µA**. Here is what the simulation measured:

```
== W/L is a knob   (V_GS = 1.8 V, V_DS = 1.8 V)
   device              W/L          I_D  I_D/I_D(A)  (W/L)/(W/L)_A
   A  W=5   L=1       5.00   696.275 uA      1.0000         1.0000
   B  W=10  L=1      10.00  1395.320 uA      2.0040         2.0000
   C  W=5   L=2       2.50   380.728 uA      0.5468         0.5000
   D  W=1   L=0.15    6.67   501.046 uA      0.7196         1.3333
```

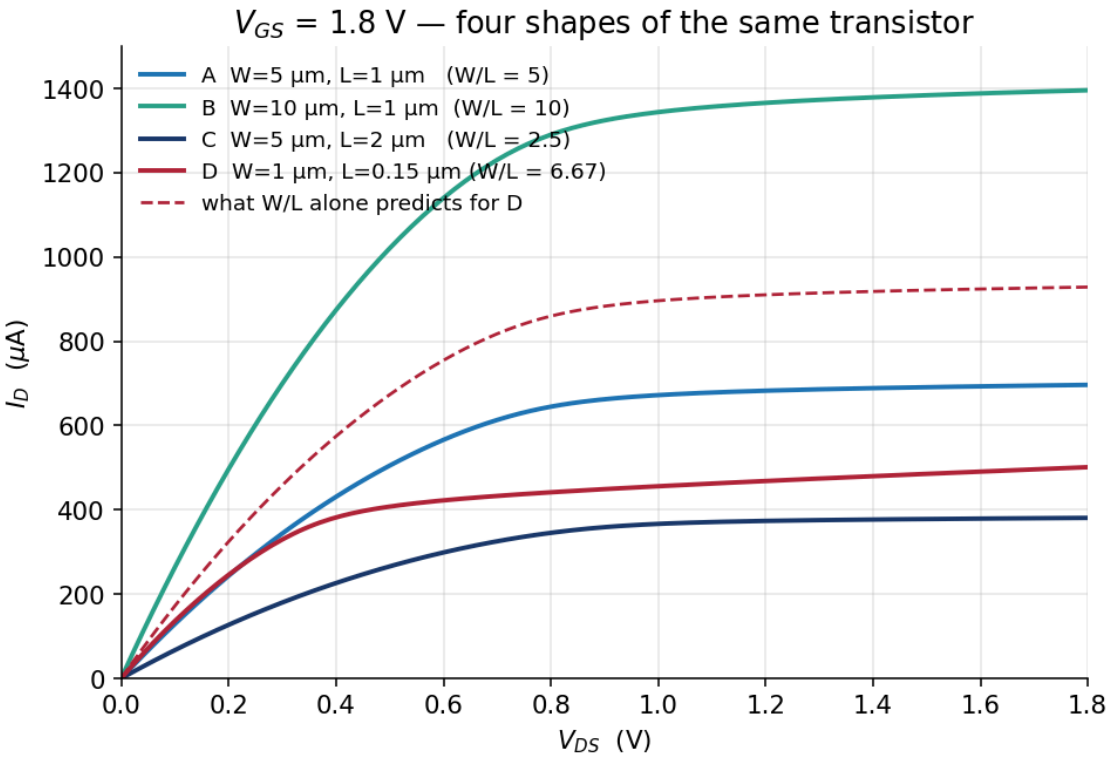


Figure 15.1: Four device geometries at the same bias, with the W/L prediction for the minimum-length device drawn as a dashed line

The data are from `spice/wl_sweep.spice`, plotted by `src/plot_curves.py`. The dashed line is what W/L alone predicts for device D, and the solid red curve is what device D actually carries.

The prediction holds for B and misses for C and D, and the two misses have different causes. We'll take the devices in order of how far off the prediction is, so B comes first.

Device B: width

The prediction for B was 1 392.551 µA and the simulation measured **1 395.320 µA**, so the two agree to two parts in a thousand.

That's because width really is a simple multiplier. Doubling W is physically the same as putting two of the original transistors side by side and wiring them in parallel, and that is how a wide device is laid out on a real chip anyway, as `nf` fingers of a narrower device. Nothing about the physics along the channel changes, so the current per micron of width doesn't change either.

The 0.20 % that's left over comes from the source and drain diffusion resistance, which doesn't scale quite in step with W. You'll meet it properly in AD104, when you draw one of these devices.

So current is proportional to W, and that's a rule you can trust in hand calculations.

Device C: length

For C the prediction was 348.138 µA and the simulation measured **380.728 µA**, so the long device carries 9.4 % *more* current than $1/L$ says it should. Two effects add current here.

The threshold. `spice/op_params.spice` gives C a model `vth` of 0.5495 V against A's 0.5895 V (the table under Device D below lists both), so at the same gate voltage C has 40 mV more overdrive. That's the same trend What moves the threshold measured over nine lengths, seen from the long end. If you put each device's own threshold into the square law, you get

$$\frac{I_D(\mathrm{C})}{I_D(\mathrm{A})} = \frac{(W/L)_\mathrm{C}\,(1.8-0.5495)^2}{(W/L)_\mathrm{A}\,(1.8-0.5895)^2} = \frac{2.5 \times 1.5638}{5 \times 1.4654} = 0.5336 \tag{15.1}$$

$$696.275\ \mu\text{A} \times 0.5336 = 371.5\ \mu\text{A} \tag{15.2}$$

which closes about 72 % of the gap between the 348 µA prediction and the measured 381 µA.

The field. As Where saturation starts explained, a longer channel with the same voltage across it has a **lower field**, so its carriers are further from their speed limit. Expecting the current to halve when you double L assumes that carrier velocity is proportional to the field without limit, so that half the field gives half the speed. Carriers near their speed limit don't slow down in proportion when the field drops, so doubling L costs less than half the current. The measured current is 380.728 / 371.5 = 1.025 times the corrected prediction, and that remaining 2.5 % is this effect.

The model's own numbers agree. `vdsat` is **0.912 V** for the L = 2 µm device and **0.779 V** for L = 1 µm, which means the longer device needs more drain voltage before its carriers stop responding, and that's another way of saying it was less limited by velocity all along.

The rule you can trust here is that current goes *up* when you shorten the channel, but by less than $1/L$ predicts. For quick reasoning, $1/L$ gets the direction right and overestimates the size of the change.

Device D: minimum length

For D the prediction was 928.367 µA and the simulation measured **501.046 µA**, only 54 % of the prediction.

The prediction assumed that V_{TH} and μ_n stay the same when you change the device's shape, and neither of them does. We can account for the gap in two steps.

Step 1: the threshold moved. `spice/op_params.spice` gives each device's threshold at the same bias:

	L	model `vth`
A	1 µm	0.5895 V
C	2 µm	0.5495 V
D	**0.15 µm**	**0.7693 V**

The minimum-length device's threshold is **180 mV higher** than A's. That is the reverse short-channel effect from What moves the threshold, which showed the same trend over nine lengths at low drain voltage, with the threshold rising at every step to a total of 171 mV. The halo implants that cause it take up a small fraction of a long channel but most of a 0.15 µm one, which is why `nfet_01v8` at minimum length is deliberately harder to turn on.

A higher threshold uses up a lot of the overdrive. If you redo the prediction with each device's own threshold and the square law, you get:

$$\frac{I_D(\text{D})}{I_D(\text{A})} = \frac{(W/L)_\text{D}\,(1.8 - 0.7693)^2}{(W/L)_\text{A}\,(1.8 - 0.5895)^2} = \frac{6.667 \times 1.0624}{5 \times 1.4654} = 0.9666 \tag{15.3}$$

$$696.275\ \mu\text{A} \times 0.9666 = 673.06\ \mu\text{A} \tag{15.4}$$

That brings the prediction down from 928 µA to 673 µA, which is much closer to the measured 501 µA.

Step 2: the carriers hit their speed limit. The measured current is 501.046 / 673.06 = **0.744** of the corrected prediction, and the remaining 26 % is velocity saturation. The model shows it: `vdsat` for device D is **0.362 V**, while its overdrive $V_{GS} - V_{TH}$ is 1.031 V, so the device stopped responding to drain voltage at barely a third of the overdrive the square law assumed it would use.

So making the transistor 6.7× shorter gives you 0.72× the current instead of 1.33×, because you paid 180 mV of threshold for it and the carriers were already at their speed limit.

This is the most common failure in a first analog design: a hand calculation says the circuit works, a simulation says it doesn't, and a minimum-length device turns out to be behind the difference. Analog designers usually pick L **two to four times the minimum**, and you can now see all three reasons for it: the square law still roughly holds, V_{TH} is closer to nominal, and r_o is larger (64.13 kΩ against 19.45 kΩ, or 3.3×, from Where saturation starts). Digital designers use minimum length for the opposite reason, because they want the smallest, fastest switch and they don't care about r_o.

When a measured current disagrees with W/L scaling, look at the model's `vth` and `vdsat` for that device before you look at your algebra. Two `print` statements are usually enough to settle it.

What geometry costs

You can build $W/L = 5$ as W = 5 µm, L = 1 µm, or as W = 0.75 µm, L = 0.15 µm. The ratio is the same, so every equation on these pages treats the two identically, but the first one takes up **forty-four times the area** (5 µm² against 0.1125 µm²).

Device	$W \times L$	area, relative to D
A	5 × 1	33.3×
B	10 × 1	66.7×
C	5 × 2	66.7×
D	1 × 0.15	1×

Area costs money. A wafer costs what it costs, and you are billed for the square millimetres you occupy, so every micron of W you spend buying current is a micron you can't spend on something else. Area isn't the only cost, either, because gate capacitance also scales with $W \times L$. The 30.56 fF from How a MOSFET works is for device A, and every femtofarad of gate has to be charged by whatever drives it, which costs time and energy elsewhere in the circuit.

Designing a transistor means making that trade: more current, with more area and more capacitance, or less of all three. There's no correct answer to it in general, so it's a choice you make for each design.

Lab 04 tests the W/L rule over a much wider range, with seventeen devices, five of which all have $W/L = 5$. The second Lab 03 extension (device D at `L=0.5`) checks whether the two-step arithmetic above still accounts for the whole gap in a longer device.

Check yourself

1. Why is doubling W almost exactly the same as using two devices in parallel?
2. Device C is twice as long as A. Why does it carry *more* than half of A's current?
3. In the two-step calculation for device D, which step closed more of the gap?
4. Two designs have $W/L = 5$, one at minimum length and one at L = 1 µm. Name one property the long one wins on and one it loses on.

Next: Lab 04: Testing the W/L rule.

16. Lab 04: Testing the W/L rule

The square law says that only the ratio W/L matters:

$$I_D^{\text{sat}} = \frac{1}{2}\,\mu_n C_{ox}\,\frac{W}{L}\,(V_{GS} - V_{TH})^2 \tag{16.1}$$

You'll test that claim on seventeen transistors at one bias point, after first writing down your own predictions for six of them. The rule gets two of those six right to within 3 % and misses the other four by up to 91 %, and the signs of those errors follow a pattern you can read straight off the scoring table. Five of the devices have a W/L of exactly 5, and they carry five different currents. Between writing the predictions and scoring them, you'll use the Lab 03 package to watch the threshold move and to measure how much gain one transistor can give.

Package: `labs/lab-04-wl-knob/`, plus three targets in `labs/lab-03-mosfet-regions/`

Before you start

- Finish Lab 03. This lab uses its reference device, W = 5 µm, L = 1 µm at **696.2755 µA**, as the anchor for every prediction.
- Read What moves the threshold, Transconductance and output resistance and Sizing with W and L.
- Open a terminal in the workbench. Both `Makefile`s set `PDK=sky130A` themselves, so no other setup is needed.

Time: about 2 hours. `make` in this lab takes about 40 seconds; each Lab 03 target in Parts 2 and 3 takes about ten. Those wall times depend on your machine, but every other number on this page is fixed, and the checker enforces it.

Part 1: Write your predictions

What does the W/L rule predict for six devices? Do this before you run anything. It takes ninety seconds, and it's the only part of the lab you can't redo afterwards, because once you've seen the measured currents you can't predict them honestly any more.

1. Go to the lab folder, then open `predictions.txt` in a text editor:

```
cd /foss/designs/modules/ad103/labs/lab-04-wl-knob
```

2. Read the header. Your one measured anchor is the Lab 03 device: W = 5 µm, L = 1 µm, $V_{GS} = V_{DS}$ = 1.8 V, I_D = 696.2755 µA.
3. For each of the six devices, write the current the W/L rule predicts, in microamps, to one decimal place, after the `=`. Write only the number, because `make predict` doesn't read a line that ends in `uA`. Don't run anything first, and don't go back and change a prediction once you've seen the answer.

```
P1  W=10   L=1      =
P2  W=50   L=1      =
P3  W=5    L=2      =
P4  W=5    L=0.15   =
P5  W=0.75 L=0.15   =
P6  W=20   L=4      =
```

4. Save the file.

You should see six numbers in `predictions.txt`. Leave the file alone until Part 4, where you score them.

Part 2: Move the threshold

How far do body bias, temperature and channel length move V_{TH}? The two decks that answer this are in the Lab 03 package.

1. Go to the Lab 03 folder and lift the source above the bulk:

```
cd ../lab-03-mosfet-regions
make vth-body
```

```
== spice/vth_body.spice
--- vth and I_D vs V_SB, W = 5 um, L = 1 um, V_GS = 1.8 V, V_DS = 0.05 V ---
V_SB = 0 V
@m.xm1.msky130_fd_pr__nfet_01v8[vth] = 5.894596e-01
i(vds) = -6.63374e-05
...
V_SB = 1.2 V
@m.xm1.msky130_fd_pr__nfet_01v8[vth] = 8.222507e-01
i(vds) = -5.15967e-05
--- and the same device, V_SB = 0, at three temperatures ---
T = -40 C
@m.xm1.msky130_fd_pr__nfet_01v8[vth] = 6.499250e-01
i(vds) = -9.38168e-05
...
T = 125 C
@m.xm1.msky130_fd_pr__nfet_01v8[vth] = 5.010177e-01
i(vds) = -4.50650e-05
```

   The rows marked `...` here, the 27 °C row among them, are printed in full in your terminal, and they're also tabulated on What moves the threshold.

2. Work out the threshold shift at each V_{SB}, relative to V_{SB} = 0, and the drop in drain current at V_{SB} = 1.2 V. Check one row against the body-effect formula with γ = 0.4108 √V and $2\phi_F$ = 0.600 V.

3. From the temperature block, work out the threshold's slope in mV/°C, and the change in current from 27 °C to 125 °C. Predict the sign of the current change before you calculate it.

4. Measure the threshold at nine channel lengths:

```
make vth-l
```

```
== spice/vth_vs_l.spice
   L        model vth
   0.15 um    7.078517e-01 V
   0.2 um     6.872240e-01 V
   0.3 um     6.553824e-01 V
   0.5 um     6.300670e-01 V
   0.8 um     6.003750e-01 V
   1 um       5.894596e-01 V
   1.5 um     5.627056e-01 V
   2 um       5.495175e-01 V
   4 um       5.364102e-01 V
```

You should see a threshold shift of **+232.791 mV** at V_{SB} = 1.2 V, with **22.2 %** less current, and the body-effect formula should agree with the measurement to within a fraction of a millivolt. The threshold's temperature slope is about **–0.9 mV/°C**, and the hot device carries **32.1 % less** current than the one at 27 °C. As the length shrinks from 4 µm to 0.15 µm, the threshold *rises* by **171.44 mV**. What moves the threshold explains each of these.

Part 3: Efficiency and intrinsic gain

How much voltage gain can one transistor give, and how does that depend on L?

1. Predict: as L goes from 0.15 µm to 4 µm, does $g_m r_o$ rise or fall? Write it down.
2. Still in the Lab 03 folder, measure g_m and g_{ds} at four lengths and six gate voltages:

   ```
   make gm-ro
   ```

   The first block covers four lengths at $V_{GS} = V_{DS}$ = 1.8 V, and the first of them reads:

   ```
   --- W = 5 um, V_GS = V_DS = 1.8 V, four channel lengths ---
   L = 0.15 um
   i(vda) = -2.64419e-03
   @m.xma.msky130_fd_pr__nfet_01v8[gm] = 2.468806e-03
   @m.xma.msky130_fd_pr__nfet_01v8[gds] = 2.787163e-04
   ```

   The second block steps the gate of the L = 1 µm device from 0.7 V to 1.8 V.
3. For each length, compute $r_o = 1/g_{ds}$ and the intrinsic gain $g_m r_o = g_m/g_{ds}$.
4. In the second block, compute g_m/I_D at V_{GS} = 0.7 V and at 1.8 V. Compare both with the ceiling of 26.9 /V from Transconductance and output resistance.

You should see an intrinsic gain of **8.86** at L = 0.15 µm rising to **106.11** at L = 4 µm, even though g_m falls by a factor of nine over the same range. In the second block, g_m/I_D is **13.50 /V** at V_{GS} = 0.7 V, about half the ceiling, against only 1.31 /V with the gate wide open.

Part 4: Run the lab and score your predictions

Which of your six predictions land, and is there a pattern in the misses?

1. Go back to the Lab 04 folder and run the four decks:

   ```
   cd ../lab-04-wl-knob
   make
   ```

   ```
   == spice/w_ladder.spice
      wrote results/w_ladder.log
   == spice/l_ladder.spice
      wrote results/l_ladder.log
   == spice/same_ratio.spice
      wrote results/same_ratio.log
   == spice/double_w.spice
      wrote results/double_w.log
   == checking
     YOUR NUMBERS
     ok  I_D  W=1  L=1                 127.1470 uA  (reference 127.1470)
     ok  I_D  W=5  L=1                 696.2755 uA  (reference 696.2755)
     ok  I_D  W=10 L=1                1395.3200 uA  (reference 1395.3200)
     ok  I_D  W=50 L=1                6762.1370 uA  (reference 6762.1370)
     ok  I_D  W=5  L=0.15             2644.1910 uA  (reference 2644.1910)
     ok  I_D  W=5  L=4                 197.9738 uA  (reference 197.9738)
     ok  I_D  W=0.75 L=0.15            364.9284 uA  (reference 364.9284)
     ok  I_D  W=20 L=4                 798.0193 uA  (reference 798.0193)
     ok  two W=5 in parallel          1392.5510 uA  (reference 1392.5510)
     ok  vth(W=10) - vth(W=5)          -12.8354 mV  (reference -12.8354)

     2 x I_D(W=5)           =    1392.5510 uA
     two W=5 in parallel    =    1392.5510 uA   (difference +0.000000 uA)
   ```

```
PASS  all ten measured values match the reference run
```

Every run also leaves a three-line `bsim4v5.out` in the lab folder. It's harmless, and Eta0 is negative on the ngspice survival card has the details.

2. Score your predictions:

```
make predict
```

```
  device            W/L    the W/L rule       measured       rule err   your guess
your err
 ------------------------------------------------------------------------------------------------
  W=10   L=1      10.00    1392.551 uA   1395.3200 uA     -0.2 %          --        --
  W=50   L=1      50.00    6962.755 uA   6762.1370 uA     +3.0 %          --        --
  W=5    L=2       2.50     348.138 uA    380.7284 uA     -8.6 %          --        --
  W=5    L=0.15   33.33    4641.837 uA   2644.1910 uA    +75.5 %          --        --
  W=0.75 L=0.15    5.00     696.275 uA    364.9284 uA    +90.8 %          --        --
  W=20   L=4       5.00     696.275 uA    798.0193 uA    -12.7 %          --        --
```

The sample above was made with an empty `predictions.txt`, which is why its last two columns are `--`. With your file filled in, your own numbers appear there instead. (If `make predict` says `predictions.txt still has 6 blank(s)`, you didn't save the file in Part 1.)

3. Cover the last two columns and read only `rule err`, top to bottom. Each entry is the rule's error as a percentage of the measured current. That's why the L = 2 µm row reads −8.6 % here, while Sizing with W and L calls the same gap 9.4 % more current than $1/L$ predicts: that page takes it as a percentage of the rule's 348.138 µA.

You should see your six predictions in the `your guess` column. In `rule err`, the two rows where only W changed are within 3 %, and the four where L changed are out by 8 to 91 %. That error is positive every time L got shorter and negative every time it got longer.

Which half of W/L can you use in a hand calculation, and which half can't you?
Answer: A1, in Answers at the end.

Part 5: Width

Is current proportional to W, and is "twice as wide" the same as "two of them"?

1. Print every table this lab produces, with its working:

```
make extract
```

2. Read block 1, the W ladder, and open `results/ad103-w-ladder.png`:

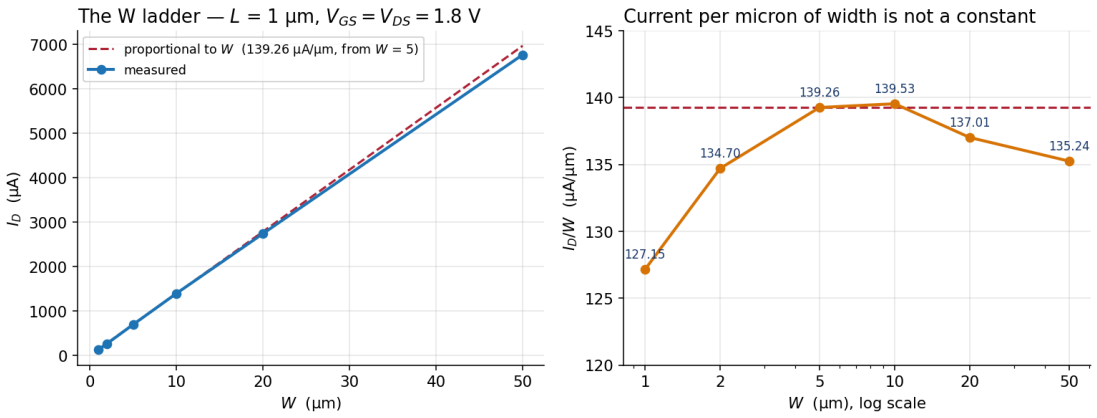


Figure 16.1: The W ladder: measured current against the proportional-to-W line, and current per micron of width

```
W (um)      I_D           I_D / W        W x I(W=1)     model vth
1         127.1470 uA  127.1470 uA/um     +0.00 %     615.2253 mV
2         269.4032 uA  134.7016 uA/um     -5.61 %     603.4546 mV
5         696.2755 uA  139.2551 uA/um     -8.69 %     589.4596 mV
10       1395.3200 uA  139.5320 uA/um     -8.88 %     576.6242 mV
20       2740.1490 uA  137.0075 uA/um     -7.20 %     564.0813 mV
50       6762.1370 uA  135.2427 uA/um     -5.99 %     556.5815 mV
```

On the left plot, the measurement and the proportional-to-W line lie almost on top of each other, so the difference is hard to see there. Look at the right plot instead, which shows the same data divided by W.

3. Read block 2, six ways of asking for "twice as wide":

```
A   W=5            696.2755 uA   1.0000 x A   the reference
B   W=10          1395.3200 uA   2.0040 x A   drawn twice as wide
C   W=5 + W=5     1392.5510 uA   2.0000 x A   two of A, wired in parallel
D   W=5 m=2       1392.5510 uA   2.0000 x A   ngspice's own multiplier
E   W=5 mult=2     696.2755 uA   1.0000 x A   the subckt's 'multiplier'
F   W=10 nf=2     1458.1360 uA   2.0942 x A   folded into two fingers

A doubled by hand    1392.5510 uA
```

Compare C and D with the last line, digit by digit.

You should see current per micron of width climb 9.7 % from W = 1 to W = 10 and then fall 3.1 % by W = 50, while the model's `vth` falls at every step of the ladder, by 58.6438 mV in all. In block 2, C and D both match the hand-doubled **1392.5510** to every digit, so two 5 µm transistors in parallel carry exactly twice the current of one, and so does `m=2`.

Over this hundredfold range of width, is "current is proportional to W" good enough to use? And can the falling `vth` explain both the rise and the fall in current per micron?

Answer: A2, in Answers at the end.

Now work out what each of the other rows in block 2 says.

- B is 2.0040× A rather than 2.0000×. Why isn't one 10 µm device the same as two 5 µm ones?

  Answer: A3, in Answers at the end.

- E says `mult=2` and carries exactly what A does. Why?

  Answer: A4, in Answers at the end.

- F has the same W, L and bias as B, and 4.50 % more current. What is different?

  Answer: A5, in Answers at the end.

Don't push `nf` higher to see how far the effect goes. `W=10 nf=5` aborts the entire run with `Fatal: Drout = -1.76308 is negative` and `doAnalyses: no such parameter on this device or parameter is missing`. That is a BSIM parameter check failing inside one of the model bins, not a mistake you made, and two fingers are enough to show the effect.

Part 6: Length

How badly does $1/L$ fail, and why?

1. Read block 3 of `make extract`, the L ladder, and open `results/ad103-l-ladder.png`:

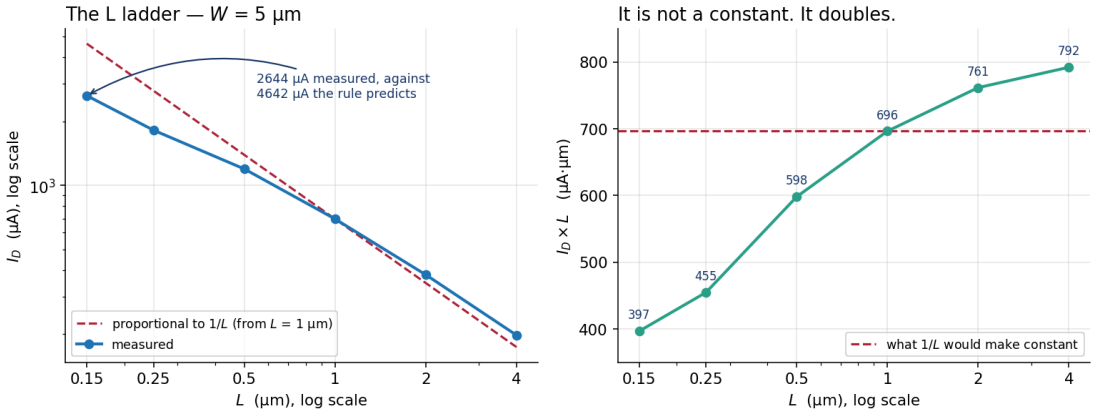


Figure 16.2: The L ladder on log axes, and I_D times L, which 1/L says should be constant

```
L (um)      I_D          I_D x L      vs 1/L from L=1   vth      vdsat
0.15     2644.1910 uA   396.6287        +75.55 %     707.44   351.92 mV
0.25     1818.6050 uA   454.6513        +53.14 %     658.63   426.29 mV
0.5      1196.0170 uA   598.0085        +16.43 %     626.23   628.45 mV
1         696.2755 uA   696.2755         +0.00 %     589.46   779.46 mV
2         380.7284 uA   761.4568         -8.56 %     549.52   912.45 mV
4         197.9738 uA   791.8952        -12.07 %     536.41  1001.27 mV
```

2. The $I_D \times L$ column is the one that tests the rule: if current really went as $1/L$, that column would show the same number in all six rows.
3. Compare `vdsat` with the overdrive $V_{GS} - V_{TH}$ for the shortest and longest devices (1092.6 mV and 1263.6 mV).
4. Read the end of the block, which closes the gap at L = 0.15 µm one effect at a time:

```
Closing the L = 0.15 um gap, one effect at a time:
  1/L alone                                  4641.837 uA
  1/L, with each device's own vth and (V_ov)^2  3781.114 uA
  measured                                   2644.191 uA
The threshold step closes 43.1 % of the gap.
```

You should see $I_D \times L$ run from **396.63 to 791.90**, so it doubles across the ladder, and `vdsat` run from 351.92 mV to 1001.27 mV. Correcting for each device's threshold closes **less than half** of the gap at 0.15 µm.

The 0.15 µm device stopped responding to its drain at a third of the overdrive the square law assumed. What does that tell you, and why can't more arithmetic on `vth` close the rest of the gap?

Answer: A6, in Answers at the end.

Part 7: Five devices with $W/L = 5$

Five devices share a W/L of 5 but differ in area. What does the extra area buys?

1. Read block 4 of `make extract`, and open `results/ad103-same-ratio.png`:

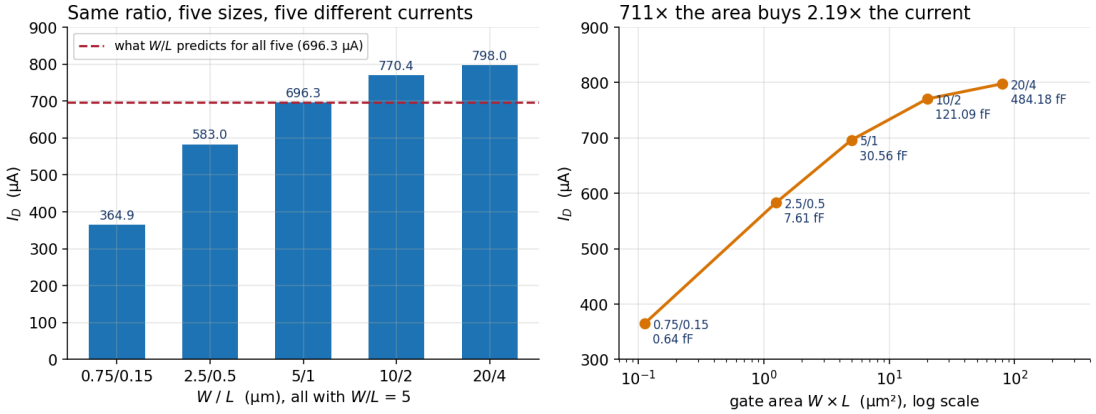


Figure 16.3: Five devices with W/L = 5, and what their area buys

```
W / L        I_D          gate area      cgg        I_D per um^2
 0.75 / 0.15    364.9284 uA    0.1125 um^2     0.6427 fF    3243.81 uA/um^2
  2.5 / 0.5     582.9524 uA    1.2500 um^2     7.6137 fF     466.36 uA/um^2
    5 / 1       696.2755 uA    5.0000 um^2    30.5632 fF     139.26 uA/um^2
   10 / 2       770.4110 uA   20.0000 um^2   121.0871 fF      38.52 uA/um^2
   20 / 4       798.0193 uA   80.0000 um^2   484.1798 fF       9.98 uA/um^2
```

2. Check the ratio in the first column: 0.75/0.15 = 5, 20/4 = 5, and so is every row in between.
3. Work out what the largest device buys over the smallest: the ratio of currents, of gate areas and of gate capacitances.
4. Read the two `gds` lines at the end of the block:

```
gds(0.75/0.15) =   38.855 uS  ->  r_o =   25.74 kohm
gds(20/4)      =    7.435 uS  ->  r_o =  134.50 kohm
```

You should see currents from **364.9284 µA to 798.0193 µA, a factor of 2.1868**, for identical W/L at identical bias. The biggest device gets 2.19× the current of the smallest for **711× the gate area** and **753× the gate capacitance**, so per square micron of gate the small one is **325× more efficient**. The biggest device's output resistance is **5.23×** the small one's.

What does the biggest device buy for its 711× area?
Answer: A7, in Answers at the end.

Part 8: Read a FAIL you caused

What does the checker tell you when one number is 3 % off?

1. Hand the checker a run that is 3 % wrong:

```
make broken
```

It copies your W ladder, moves the W = 10 µm current by 3 %, and hands the result to the checker. A 3 % error is far too small to see by eye, but it's large enough to mean you simulated the wrong device.

```
  XX  I_D  W=10 L=1                1437.1800 uA  (reference 1395.3200)
...
FAIL
  - I_D  W=10 L=1: got 1437.1800, reference 1395.3200 (+3.00 %)

  Three things cause this, in order of how often:
   1. A unit suffix on W or L somewhere in spice/. They are plain
      microns. Check with:  grep -n 'W=[0-9.]*[a-z]' spice/*.spice
```

2. Run `make` again to get back to `PASS`.

You should see one `XX` row among nine `ok` rows. The `FAIL` section names the number and gives both values and the percentage, then lists three causes in order of how often they happen, with a `grep` for the first one.

Finishing

You're done when `make predict` shows your six predictions in the `your guess` column, next to the measurements, with your own errors in `your err` (Part 4).

Questions

1. Which half of W/L can you trust in a hand calculation, and by roughly how much? (Part 4.)
2. Why does one 10 µm device differ from two 5 µm devices in parallel? (Part 5, row B.)
3. Five devices with $W/L = 5$: what does the largest one buy for its 711× area? (Part 7.)

Submit

Nothing to submit.

Going further

Three extensions, with reference answers in `solutions/`. For each one, predict first, then run it, and only then read the answer.

1. **Widen further.** Add `W=100` to the W ladder, then `W=200`. One of them tells you where the model ends, and the error it gives is one you have already met for a completely different reason.
2. **Repeat the ladder in PMOS.** Build the same ladder out of `pfet_01v8`. At $L = 1$ µm the NMOS carries **6.685×** the PMOS; the capstone measures the same ratio at $L = 0.15$ µm and gets **2.496**, so the ratio between the two device types isn't a property of the process alone.
3. **Lower the gate.** Re-run the L ladder at $V_{GS} = 0.9$ V. If velocity saturation is really what breaks $1/L$, the short-channel error should shrink at the lower gate voltage, and it does, from +75.5 % to +29.5 %. Then look at what happens at the long end.

Next on the Course path: read The CMOS inverter.

Answers

A1. Part 4: Run the lab and score your predictions

You can use W, because changing it on its own kept the rule within 3 %. L is the half you can't trust: the rule over-predicts the current of every shorter device and under-predicts every longer one, by 8 to 91 %.

A2. Part 5: Width

Yes, you can use it, because it's wrong by less than 10 % over the whole ladder. The direction of the `vth` trend makes sense (a wider device has a lower threshold, so it turns on harder), but a cause that only ever moves one way can't on its own produce a rise and then a fall. Something else must push the other way beyond $W = 10$ µm, and this course doesn't claim to know what it is.

A3. Part 5: Width

One 10 µm device carries 2.7690 µA (+0.199 %) more than two 5 µm ones. The model reports `vth` = 589.4596 mV for the 5 µm device and 576.6242 mV for the 10 µm one, 12.8354 mV apart even though the

bias and length are identical. So width enters the model itself rather than only multiplying its result, and that makes "twice as wide" and "two of them" different circuits.

A4. Part 5: Width

The SKY130 subcircuit declares a parameter called `mult` and never uses it. You can see this in `/foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/sky130_fd_pr__nfet_01v8__tt.pm3.spice`, where the `.subckt`'s `.param` line lists `mult = 1` and the device line inside carries no `m = {mult}`. So if you write `mult=2` in a deck by hand, it's accepted and ignored, and nothing tells you so; `m=2` is the parameter that works. XSchem's symbol writes `m=` for you whenever `mult` isn't 1. If you set `mult=2` in the property editor, the device line comes out `... sd=0 mult=2 m=2`, and the current doubles. (At `mult=1` the line ends `... sd=0 mult=1` with no `m=`, because there is nothing to multiply.) That's why drawing the schematic gets this right, and copying a device line by hand doesn't.

A5. Part 5: Width

`nf=2` folds one 10 µm channel into two 5 µm fingers that share a drain diffusion. Nothing in W/L describes that, because it's a fact about the layout, and you'll draw it in AD104.

A6. Part 6: Length

It tells you the device is velocity saturated, and that's why $1/L$ over-promises on every short device. No further arithmetic on `vth` can close the rest of the gap, because the square law has no term for a carrier speed limit. At this point hand calculation runs out, and you need the simulator.

A7. Part 7: Five devices with W/L = 5

It doesn't buy much current: 2.19× for 711× the area. Area costs money, and gate capacitance costs time, because whatever drives the gate has to charge it. What the area does buy is 5.23× the output resistance. An amplifier's gain is $g_m r_o$, so that factor is the reason an analog designer pays for a long channel, and the reason a digital designer doesn't.

V

Capstone

17 The CMOS inverter 113
17 What you already know 113
17 Drive strength of each half 113
17 The transfer curve 113
17 The switching threshold V_M 114
17 Sizing the PMOS 114
17 Lengthening the channels 115
17 Current while switching 117
17 Check yourself 117

18 The current mirror 119
18 Why not distribute a voltage 119
18 The circuit 119
18 Limit 1: output headroom 120
18 Limit 2: ratio error 120
18 The fix: unit devices 121
18 After AD103 122
18 Check yourself 122

19 Capstone: The CMOS inverter 123
19 Before you start 123
19 Part 1: Run the reference decks 123
19 Part 2: Draw the inverter 124
19 Part 3: The switching threshold 126
19 Part 4: Size the PMOS from the drive ratio . . . 127
19 Part 5: Time the three inverters 128
19 Part 6: The same circuit as an amplifier 128
19 Part 7: The current while it switches 129
19 Part 8: The PMOS source on the wrong node . . 129
19 Part 9: The current mirror 130
19 Finishing 131
19 Going further 131
19 Answers 133

17. The CMOS inverter

The digital track draws this circuit and calls it a NOT gate. It is also an amplifier, and this page treats it as one, although digital design only ever looks at its two ends (the two flat ends of the transfer curve below).

The circuit is two transistors with their gates tied together and their drains tied together: an NMOS pulling the output down and a PMOS pulling it up. In DD101 this is the smallest circuit that computes, and in this course it is the smallest circuit with gain.

What you already know

The pieces you need all come from earlier pages:

- Roughly speaking, a MOSFET is **off** below threshold and **on** above it (How a MOSFET works).
- In more detail it has four regions, and the interesting one here is saturation (The four regions).
- In saturation the gate moves the current by g_m, and the drain leaks it back through r_o (Transconductance and output resistance).
- Electrons are more mobile than holes, so the same shape drawn as a PMOS is weaker.

Everything the inverter does follows from these four facts, without any new idea.

Drive strength of each half

The capstone lab starts by measuring each transistor on its own, with both drawn at W = 1 µm, L = 0.15 µm and both turned fully on:

```
== 1. What each transistor can do on its own   W = 1 um, L = 0.15 um
   NMOS, gate and drain at 1.8 V          501.0462 uA
   PMOS, gate and drain at 0 V            200.7478 uA
   NMOS / PMOS                              2.4959
```

The NMOS current, **501.0462 µA**, is the same number Lab 01's `make` printed on your first day, from a deck with only that one transistor in it.

The PMOS, drawn the same size and driven just as hard, carries **200.7478 µA**, so the NMOS carries **2.4959 times** as much. Holes drift through silicon more slowly than electrons do, by roughly that factor in this process, and the only thing a designer can do about it is draw the PMOS wider. Every asymmetry on the rest of this page comes from that ratio.

The transfer curve

If you sweep the input from 0 to 1.8 V and plot the output, you get this curve:

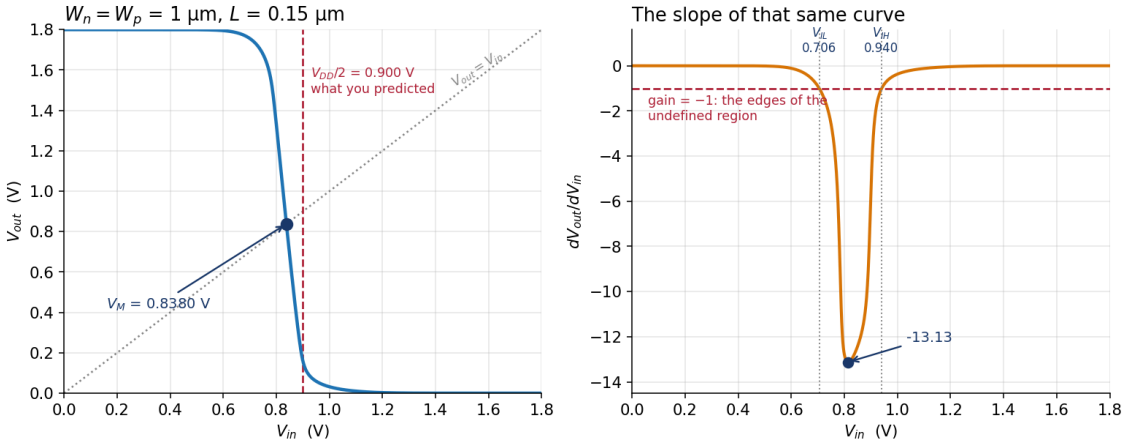


Figure 17.1: The voltage transfer characteristic of a CMOS inverter: flat at 1.8 V, a near-vertical drop near 0.84 V, flat at 0 V

From `spice/vtc.spice` in the capstone package, with $W_n = W_p = 1$ µm and $L = 0.15$ µm, at the SKY130 tt corner.

The curve has three parts, and digital design uses only two of them:

- **Input low.** The NMOS is below threshold and the PMOS is fully on, so the circuit is a closed switch to V_{DD} and the output sits at **1.7506 V**.
- **Input high.** This is the mirror image, with the NMOS on and the PMOS off, and the output sits at **0.0684 V**.
- **In between.** Both transistors are on and both are in saturation, and the output falls steeply.

That steep middle part is where the circuit amplifies, and its slope is the voltage gain:

```
switching threshold V_M       0.838027 V   (V_DD/2 would be 0.900000)
steepest slope                -13.1253     at V_in = 0.814 V
```

The gain is **−13.1253**: if you wiggle the input by 1 mV near 0.814 V, the output moves 13.1 mV the other way. That makes it a real amplifier, if not a good one (the circuit doesn't know it's supposed to be a logic gate).

Digital logic works *because* this gain is bigger than one. A gate with a gain of 0.9 would shrink every signal a little at each stage, and a chain of them would fade to grey. With a gain above one, a logic level **restores** itself as it passes through: a slightly degraded 1.6 V at the input comes out as a clean 1.7506 V. You can think of digital design as analog design with enough gain that nobody has to think about it.

The switching threshold V_M

The **switching threshold** V_M is the input voltage at which the output equals the input, which is where the curve crosses the 45° line. Here it is **0.838027 V**. $V_{DD}/2$ would be 0.900000 V, so V_M sits **62.0 mV low**.

Despite the name, V_M isn't the threshold voltage of either device. Those are 769.27 mV and 510.03 mV, and their average, 0.6396 V, isn't V_M either. V_M is a property of the *circuit*: it's the input voltage at which the NMOS current and the PMOS current are equal, with both devices in saturation. Because the NMOS is 2.4959× stronger, the currents balance at an input **below** the midpoint, so the pull-down wins with less gate voltage than you might expect.

Sizing the PMOS

The obvious fix is to make the PMOS 2.4959× wider so that the two devices are equally strong, but as this sweep of the PMOS width shows, that doesn't quite work:

```
Wp (um)   Wp/Wn        V_M        error vs 0.9 V      gain
1          1.00    0.838027 V      -61.97 mV     -13.1253
```

```
2          2.00    0.869826 V      -30.17 mV     -12.8834
2.5        2.50    0.882739 V      -17.26 mV     -11.8856
3.5        3.50    0.899865 V       -0.13 mV     -11.2590
4          4.00    0.905807 V       +5.81 mV     -11.2846
```

$W_p = 2.5$ µm, the ratio the current measurement suggested, still leaves **17.26 mV** of error, so it removes about two thirds of the problem and leaves a third. The ratio that actually centres V_M is **3.5**.

It's worth understanding why, more than remembering the number. The drive measurement had each gate at the *full* supply. At V_M the input is V_M, so with V_M centred the NMOS sees $V_{GS} = 0.9$ V and the PMOS sees $V_{SG} = 0.9$ V. Neither device is anywhere near full drive, and at that lower overdrive the two devices' currents don't scale in the same way. **Matching currents at 1.8 V is a different condition from matching them at 0.9 V**, and only the second one sets V_M.

Now time all three sizings, each driving a 10 fF load:

```
Wp (um)      t_pHL         t_pLH      pull-up / pull-down
1          27.7001 ps    64.6539 ps         2.334
2.5        30.2508 ps    30.7067 ps         1.015
3.5        31.7807 ps    24.1077 ps         0.759
```

The two sizings answer two different questions, and neither is wrong:

You want	Draw	You get
equal rise and fall time	$W_p = 2.5$ µm	30.2508 / 30.7067 ps, 1.5 % apart
equal noise margins (V_M centred)	$W_p = 3.5$ µm	V_M = 0.899865 V, 0.13 mV out

Delay is charge divided by current, and the load and the voltage swing are the same for both edges, so equal drive currents give equal delays. That is why $W_p = 2.5$ µm, which matches the full-drive currents, gives matched edges. The $W_p = 3.5$ inverter rises **24 % faster than it falls**, while the $W_p = 2.5$ inverter has its switching point 17 mV low. No width does both, so which one you choose depends on whether your concern is timing or noise.

> **When two reasonable criteria give two different answers,** you've found a trade-off rather than a mistake. Name both criteria before you pick a width.

Lengthening the channels

The gain was −13.1253 with $L = 0.15$ µm, the process minimum. Transconductance and output resistance showed that gain is g_m times an output resistance, and that r_o grows quickly with length. So the next step is to lengthen both devices and change nothing else:

```
L = 0.15 um, Wn = Wp = 1 um    gain  -13.1253 at V_in = 0.814 V
L = 0.5  um, Wn = Wp = 1 um    gain -116.0341 at V_in = 0.712 V
                               V_M   0.714056 V
```

Making the channels 3.33× longer gives **8.84× the gain**, from the same two transistors on the same supply, with nothing changed but `L`.

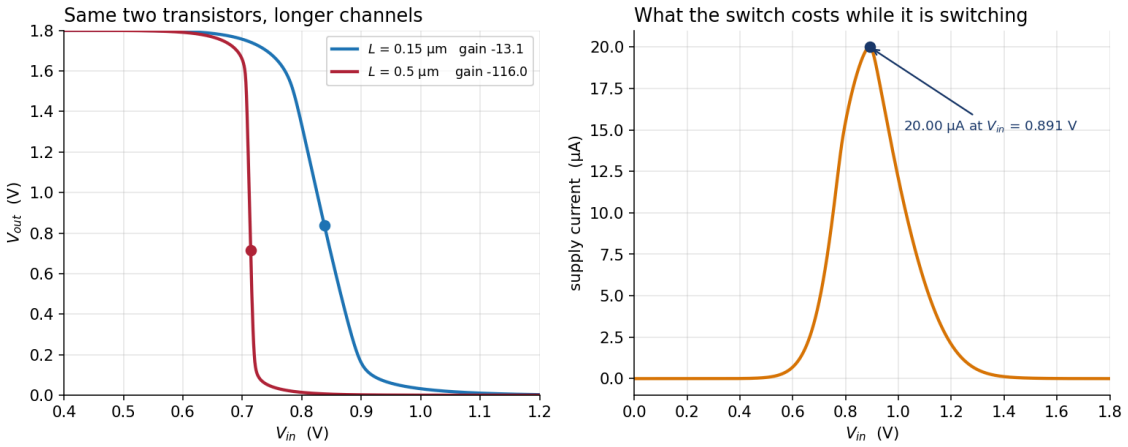


Figure 17.2: Two transfer curves overlaid: the L = 0.15 µm inverter and the much steeper L = 0.5 µm one

It's worth checking this against Transconductance and output resistance, where the intrinsic gain $g_m r_o$ of a single $L = 0.5$ µm device was **35.07**. The inverter gets **116.03**, more than three times that.

Both transistors amplify at once. The gates are tied together, so a wiggle on the input moves the NMOS current and the PMOS current in the same direction through the output node. The transconductances therefore add, and the output resistances combine in parallel:

$$A_v = -(g_{mn} + g_{mp})(r_{on} \parallel r_{op}) \tag{17.1}$$

That alone doesn't explain the factor of three over one device, because the output conductances add as well. The formula is a weighted average of the two devices' own $g_m r_o$, so the pair can't beat what each transistor manages by itself at the same bias, and the difference is in the bias. The 35.07 was measured with the gate and drain at 1.8 V, but at the inverter's steepest point the input is 0.712 V, both transistors are only just on, and the supply current through the pair is 2.3618 µA. Block 5 of the capstone's `make extract` gets each device's g_m and g_{ds} at that point from `spice/gain_op.spice`:

```
                 g_m          g_ds       g_m / g_ds
NMOS       36.3178 uS      0.3136 uS      115.81
PMOS       22.2154 uS      0.1894 uS      117.31
(g_mn + g_mp) / (g_dsn + g_dsp)            116.37
the sweep's steepest slope                 116.03
```

At this bias each transistor has a $g_m r_o$ of about 116 on its own, and the formula gives 116.37 against the 116.03 the sweep measured.

Driving the PMOS gate does pay off against a different baseline: a PMOS used only as a load. If its gate sat at a fixed DC voltage, the NMOS alone would amplify, into the same two output conductances, and give $36.3178/(0.3136 + 0.1894) = 72.2$. Because the PMOS amplifies too, the pair gets 116. Two devices pushing into one node is the cheapest gain in analog design. The arrangement is called the **CMOS inverting amplifier**, or push-pull stage, and you'll meet it again in AD201. It's the same schematic your logic library calls `inv_1`.

The bias problem. To use this as an amplifier you have to hold the input at 0.712 V, where the slope is 116. A 10 mV drift in that bias then moves the output by 1.16 V, which slams it into a rail. Real amplifiers spend most of their transistors on holding the operating point still, and the first tool for that is the current mirror, on the next page.

Current while switching

```
supply current at V_in = 0 V             0.000002 uA
supply current at V_in = 1.8 V           0.000320 uA
peak supply current                       20.0048 uA   at V_in = 0.891 V
```

At either end of the sweep one transistor is off, so the gate draws almost nothing, 2 pA with the input low and 320 pA with it high. A CMOS gate that isn't switching costs essentially nothing, and that is why CMOS replaced every logic family before it.

Halfway through a transition both devices are on at once, and **20.0048 µA** flows straight from V_{DD} to ground without doing any useful work. This is **short-circuit current**, one of the two reasons a chip heats up when it computes (the other is charging all that gate capacitance). Multiply it by a hundred million gates switching a billion times a second and you have the thermal design problem of a processor.

The same current explains a rule the digital track gives without proof: **never leave an input floating.** A floating gate drifts to somewhere near V_M, which parks the inverter in the steep middle of its curve, and there it burns 20 µA indefinitely in a gate that is doing nothing.

You'll measure every number on this page yourself in the capstone lab.

Check yourself

1. Why does a gain greater than one let a chain of inverters restore a degraded logic level?
2. With $W_n = W_p$, V_M is below $V_{DD}/2$. Which device is stronger, and why does that pull V_M down?
3. Why doesn't W_p = 2.5 µm, the measured drive ratio, centre V_M? Which criterion does it satisfy instead?
4. Lengthening both devices from 0.15 µm to 0.5 µm raised the gain from 13 to 116. Which quantity in $A_v = -(g_{mn} + g_{mp})(r_{on} \parallel r_{op})$ changed most?

Next: The current mirror.

18. The current mirror

The CMOS inverter works as an amplifier only if its input is held at 0.712 V, where it sits on a slope with a gain of 116. This page is about what holds a bias point like that in place, and the answer is a **current** rather than a voltage.

It's the last new idea in AD103, and it's the one that turns the previous pages into a circuit you could build. You have already met the idea behind it in AD102, applied to resistors.

Why not distribute a voltage

Every amplifier on a chip needs to be biased at the right point. The obvious plan is to generate one good bias voltage and wire it to every amplifier. Before you do, count what that voltage would have to survive, using numbers from earlier pages:

It has to survive	measured
threshold moving with temperature	**−0.9025 mV/°C**, 148.9 mV across the automotive range
threshold moving with channel length	**171.44 mV** from L = 4 µm to L = 0.15 µm
threshold moving with source-to-bulk voltage	**232.791 mV** at V_{SB} = 1.2 V
process corners	Part 1 of AD102 Lab 04

A gate voltage that is 100 mV wrong, on a device with g_m = 334 µS, gives a current that is **33 µA** wrong, so the plan doesn't survive a real die.

AD102's Lab 04 reached a conclusion that points the way out: **absolute values wander, but ratios hold.** Two devices side by side on the same die are at the same temperature and in the same corner, and they have the same threshold, so whatever moved one has moved the other too.

So instead of distributing a voltage, you distribute a *current*, using a circuit whose output is a **ratio** of its input.

The circuit

Take a transistor and tie its gate to its drain. It is now a two-terminal device, and it is in saturation automatically: $V_{DS} = V_{GS}$, so V_{DS} always exceeds $V_{GS} - V_{TH}$. This is called a **diode-connected** transistor.

If you push a known current into it, it settles at whatever gate voltage carries that current. In effect the device solves its own equation for you, which is useful because you don't know $\mu_n C_{ox}$ and it does.

Now connect a second transistor's gate to the same node. The second device has the same V_{GS}, is made of the same silicon and sits at the same temperature, so it carries the same current. That pair is a **current mirror**.

The capstone package's `spice/mirror.spice` forces 50 µA into the reference branch and prints the gate voltage it settles at:

```
--- part 1: the reference branch sets a gate voltage ---
v(ng) = 8.672678e-01
@m.xmref.msky130_fd_pr__nfet_01v8[vdsat] = 2.336319e-01
```

The diode-connected device chose **0.8672678 V** to carry 50 µA. Nobody calculated that number; the transistor found it for itself. If the die is hot or slow, or the threshold has moved, the number moves with it, which is what you want, because the copy has shifted in the same way, so the gate voltage that is right for the reference is right for the copy too.

With the copy's output held at 0.9 V, the deck measures the copied current:

```
i(vo2) = -5.00878e-05
```

That's **50.0878 µA against a 50 µA reference, a 0.176 % error**, from a circuit with no resistors, no feedback and no trimming.

Limit 1: output headroom

A saturated transistor is very nearly a current source, but if you drop it out of saturation it becomes a resistor. Sweeping the output node shows where that happens:

V_{out}	I_{out}	error vs 50 µA
0.1 V	28.4076 µA	**−43.18 %**
0.2 V	42.7022 µA	−14.60 %
0.4 V	48.1163 µA	−3.77 %
0.6 V	49.1572 µA	−1.69 %
0.9 V	50.0878 µA	+0.18 %
1.2 V	50.8174 µA	+1.63 %
1.8 V	52.0797 µA	+4.16 %

The error column contains two separate effects, and you need to be able to tell them apart.

Below about 0.25 V the copy collapses. The reference branch already printed the reason: `vdsat = 2.336319e-01`. With less than **0.2336 V** on its drain, the output device is in triode rather than saturation, and a triode device's current depends on its drain voltage. Every current mirror has a minimum output voltage like this, called its **compliance**, and for this one it's V_{DSAT}, about 234 mV. That's why a 1.8 V supply feels tight: every current source in a stack uses up about a quarter of a volt just to work.

Above that, the copy drifts upward by about 1.6 % per 0.3 V. That drift is r_o, the finite slope of saturation, and you can estimate it from the 0.6 V and 1.8 V rows, which are both clear of the collapse:

$$r_o \approx \frac{1.8 - 0.6}{(52.0797 - 49.1572)\ \mu\ \mathrm{A}} = 410.6\ \mathrm{k}\ \Omega \tag{18.1}$$

So this "current source" has 410 kΩ in parallel with it. That finite resistance is the reason the next circuit in every analog textbook is the **cascode**, which stacks a second transistor on top to shield the first from the output swing. It buys back a factor of $g_m r_o$ in output resistance, at the cost of another V_{DSAT} of compliance.

> **Before you call a mirror broken, ask what V_{out} was.** When half the current is missing, the cause is almost always a device in triode rather than a wiring error.

Limit 2: ratio error

If you widen the output device, the copy scales with it, so a mirror is also a *multiplier*. The deck draws 1:1, 1:2 and 1:4 mirrors by width, all from one gate voltage and with the outputs at 0.9 V, so widths of W = 5, 10 and 20 µm should give 1×, 2× and 4×.

drawn	W	I_{out}	actual ratio	error
1×	5 µm	50.0878 µA	1.0000	—
2×	10 µm	112.313 µA	**2.2423**	**+12.12 %**
4×	20 µm	239.347 µA	**4.7785**	**+19.46 %**

The 2× branch is twelve percent high and the 4× branch nearly twenty, which is a lot for a circuit whose whole selling point is exact ratios. The deck prints the cause on the next lines:

```
  and why the ratio is not what you drew:
@m.xmw1.msky130_fd_pr__nfet_01v8[vth] = 5.894596e-01
@m.xmw2.msky130_fd_pr__nfet_01v8[vth] = 5.766246e-01
```

```
@m.xmw1.msky130_fd_pr__nfet_01v8[gm] = 3.340839e-04
@m.xmw2.msky130_fd_pr__nfet_01v8[gm] = 7.187672e-04
```

The W = 10 µm device has a threshold **12.835 mV lower** than the W = 5 µm one. This course's Lab 04 measured the same shift independently (`vth(W=10) - vth(W=5) = -12.8354 mV`), so two different decks agree to five figures. The cause is the **narrow-width effect**: the edges of a channel behave differently from its middle, and in a wider device the edges are a smaller fraction of the whole channel.

Does 12.8 mV explain 12 %? The wide device's g_m is 718.7672 µS, so a threshold 12.835 mV lower is worth about

$$\Delta I \approx g_m \Delta V_{TH} = 718.7672\,\mu\text{ S} \times 12.835\,\text{mV} = 9.23\,\mu\text{ A} \tag{18.2}$$

on a branch that should carry $2 \times 50.0878 = 100.1756$ µA. The measured excess is **12.14 µA**, which is the same size and the same sign. The straight-line estimate reads low because at g_m/I_D = 6.40 /V (718.7672 µS on 112.313 µA) this device is in moderate inversion, where current rises faster than linearly with gate voltage. So the measurement confirms the mechanism even though the linear estimate is loose, and it's worth keeping those two claims separate: you've shown what causes the error, not that the straight-line formula predicts its size.

The fix: unit devices

Matching and Pelgrom's law gave you a rule for resistors:

> *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.*

A 4:1 resistor ratio is built from four identical units and one unit, never from a long strip beside a short one, because the ends of a strip don't scale with its length. Transistors follow the same rule for the same reason: **the edges don't scale.**

So don't draw one W = 10 µm device. Draw **two W = 5 µm devices in parallel**, which have the same total width and the same total area as the single device, and in theory carry the same current.

```
--- part 4: one W=10 device vs two W=5 devices in parallel ---
i(vo3) = -1.12313e-04
i(vo5) = -1.00176e-04
```

output branch	I_{out}	ideal 2 ×	error
one W = 10 µm device	112.313 µA	100.1756 µA	**+12.12 %**
two W = 5 µm devices in parallel	**100.176 µA**	100.1756 µA	**+0.0004 %**

The parallel pair matches the ideal to six figures. Two copies of the unit device carry exactly twice the unit device's current because they *are* two copies of it, with the same width, the same edges and the same threshold, so there's nothing left to differ.

This is why real analog layout is full of arrays of identical transistors with their sources and drains strapped together, and why a schematic that says `m=4` means "four of these" rather than "one four times as wide". It's the rule AD102 showed on resistors (where it made a 3:1 divider land at 1.350000 V instead of 1.309327 V), reached here from a different direction.

> **When a schematic asks for a ratio of two devices,** check whether both sides are built from the same unit. If they aren't, the ratio has a systematic error that no amount of care will remove.

You'll run `spice/mirror.spice` yourself in the capstone lab.

After AD103

AD104 takes the inverter you draw in the capstone and has you build it as geometry. The capstone lab's last section shows what that looks like and which ideas from this course come back there.

Check yourself

1. Why is a diode-connected transistor always in saturation?
2. A mirror's output sits at 0.15 V. Which limit is it hitting, and what would you check first?
3. Using r_o = 410.6 kΩ, how much would the copied current change if V_{out} moved by 0.5 V?
4. Why do two W = 5 µm devices in parallel copy a current more accurately than one W = 10 µm device?

Next: Capstone: The CMOS inverter.

19. Capstone: The CMOS inverter

You'll draw a CMOS inverter, measure the voltage at which it switches, and size its PMOS two different ways for two different goals. You'll also run the same circuit as an amplifier, and finish with the current mirror that would bias it.

At the end you'll have a schematic of a real logic gate, sized by you and verified against a simulation, and it's the file that **AD104** asks you to draw as geometry.

Package: `labs/capstone-inverter/`

Before you start

- Finish Lab 04.
- Read The CMOS inverter and The current mirror.
- Open a terminal in the workbench, load the environment (you'll use XSchem in Part 2), and go to the lab folder:

```
. /foss/designs/common/.designinit
cd /foss/designs/modules/ad103/labs/capstone-inverter
```

The `Makefile` sets `PDK=sky130A` itself, so the `make` targets work even without the first line.

Time: about 2.5 hours. `make` takes about 55 seconds; `make mirror` and `make broken` take about ten seconds each. Once `make` has run, `make figures` takes a few seconds, and so does `make extract`, except the first time, when it also runs `spice/gain_op.spice` and takes about ten.

At the switching threshold the input and output are the same voltage, so both transistors are saturated and carrying the same current:

$$I_{Dn}(V_{GS} = V_M) = I_{Dp}(V_{SG} = V_{DD} - V_M) \tag{19.1}$$

That condition says the currents are equal at V_M, which is not the same as saying the two devices are equally strong at full drive, and the difference between those two statements is what most of this lab is about. The gain at the switching point has the same form as any amplifier's gain:

$$A_v = -\left(g_{mn} + g_{mp}\right)\left(r_{on} \parallel r_{op}\right) \tag{19.2}$$

Part 1: Run the reference decks

Does your workbench reproduce the reference inverter?

1. Run the five decks and the checker:

```
make
```

```
== spice/vtc.spice
   wrote results/vtc.txt
== spice/vtc_ratio.spice
   wrote results/vtc_ratio.txt
== spice/vtc_long.spice
   wrote results/vtc_long.txt
== spice/drive.spice
   wrote results/drive.log
== spice/delay.spice
```

```
   wrote results/delay.log
== checking
  YOUR NUMBERS
  ok  I_D  NMOS W=1 L=0.15          501.0462 uA  (reference 501.0462)
  ok  I_D  PMOS W=1 L=0.15          200.7478 uA  (reference 200.7478)
  ok  NMOS / PMOS drive ratio         2.4959     (reference 2.4959)
  ok  V_M   Wn = Wp = 1               0.8380 V   (reference 0.8380)
  ok  gain at V_M, L = 0.15         -13.1253     (reference -13.1253)
  ok  V_M   Wp = 2.5                  0.8827 V   (reference 0.8827)
  ok  V_M   Wp = 3.5                  0.8999 V   (reference 0.8999)
  ok  V_M   L = 0.5, Wn = Wp          0.7141 V   (reference 0.7141)
  ok  gain at V_M, L = 0.5         -116.0341     (reference -116.0341)
  ok  t_pHL  Wp = 2.5                30.2508 ps  (reference 30.2508)
  ok  t_pLH  Wp = 2.5                30.7067 ps  (reference 30.7067)
  ok  t_pLH / t_pHL  Wp = 1           2.3341     (reference 2.3341)

  V_out(0.0 V) = 1.800000 V   V_out(1.8 V) = 2.129252e-07 V

PASS  all twelve measured values match the reference run
```

2. Draw the figures into `results/`:

```
make figures
```

You should see `PASS  all twelve measured values match the reference run`.

Part 2: Draw the inverter

Can you draw an inverter that behaves exactly like the shipped deck?

1. Open your scaffold:

```
make edit
```

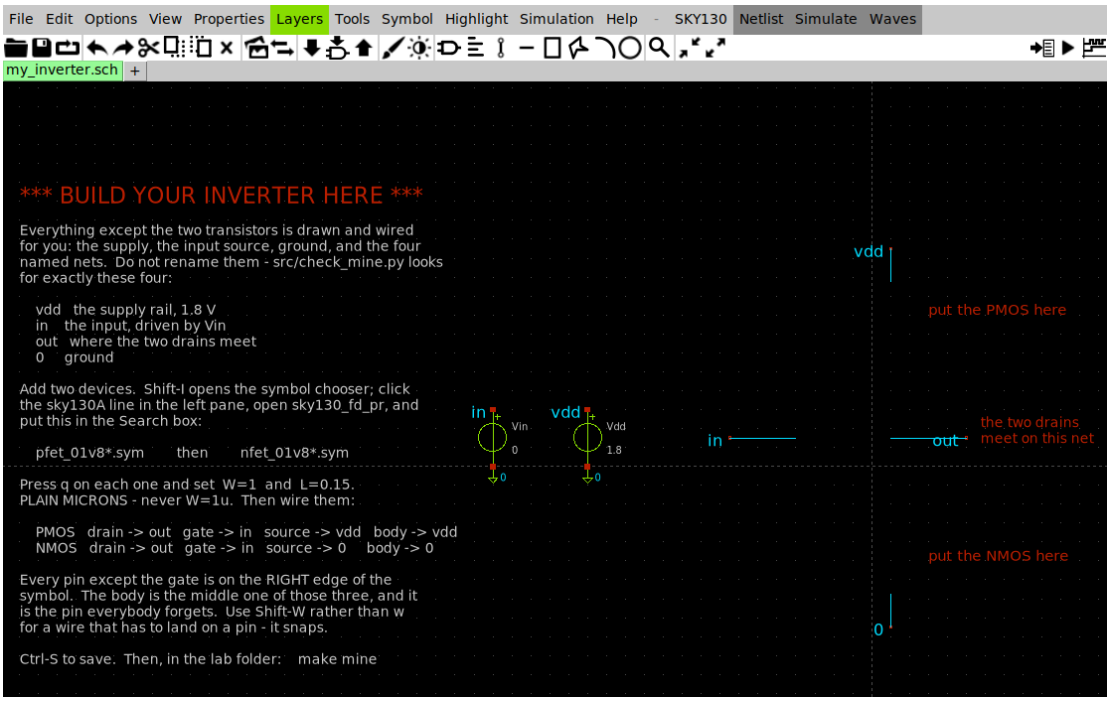


Figure 19.1: xschem/my_inverter.sch — the supply, the input, the four named nets, and two gaps

The scaffold has everything except the two transistors. The nets `vdd`, `in`, `out` and `0` are already named and wired. Don't rename them, because `src/check_mine.py` looks for exactly those four names.

2. Add a `pfet_01v8` and an `nfet_01v8` (`Shift-I`, as in Lab 01), both with `W=1` and `L=0.15`.
3. Wire the eight pins:

```
    PMOS   drain -> out    gate -> in    source -> vdd    body -> vdd
    NMOS   drain -> out    gate -> in    source -> 0      body -> 0
```

   The first time through, wire the PMOS **body** to `0` instead of `vdd`, on purpose. It's the pin everyone forgets, so it's worth seeing what the checker says about it.

4. Save (`Ctrl-S`, then **Yes**) and check your drawing:

```
make mine
```

   Read what it says about the PMOS body, then fix the wire, save, and run `make mine` again.

   `make mine` checks the things a first inverter usually gets wrong (a missing device, a unit suffix on `W`, an invented `net7` where a wire didn't land, the PMOS body left on ground) and reports each one in words.

5. When it passes, compare your switching threshold with the reference.

You should see, once the wiring is right:

```
  YOUR INVERTER
    V_out(0.0 V)              1.800000 V
    V_out(1.8 V)          2.131046e-07 V
    switching threshold       0.838029 V
    gain at V_M               -13.1252
    noise margins H / L     0.8106 / 0.6376 V

PASS  Wn = Wp = 1 um, and your V_M is 0.838029 V against the
      reference 0.838027 V. You drew the same inverter the
      shipped deck describes, and it behaves identically.
```

Your 0.838029 V and the deck's 0.838027 V differ by two microvolts, which shows that your drawing and the hand-written deck in `spice/vtc.spice` are the same circuit. Every schematic you draw is really a claim about a netlist, and comparing it against a deck you already trust is how you check that claim.

Here is the finished version, to compare against once yours works:

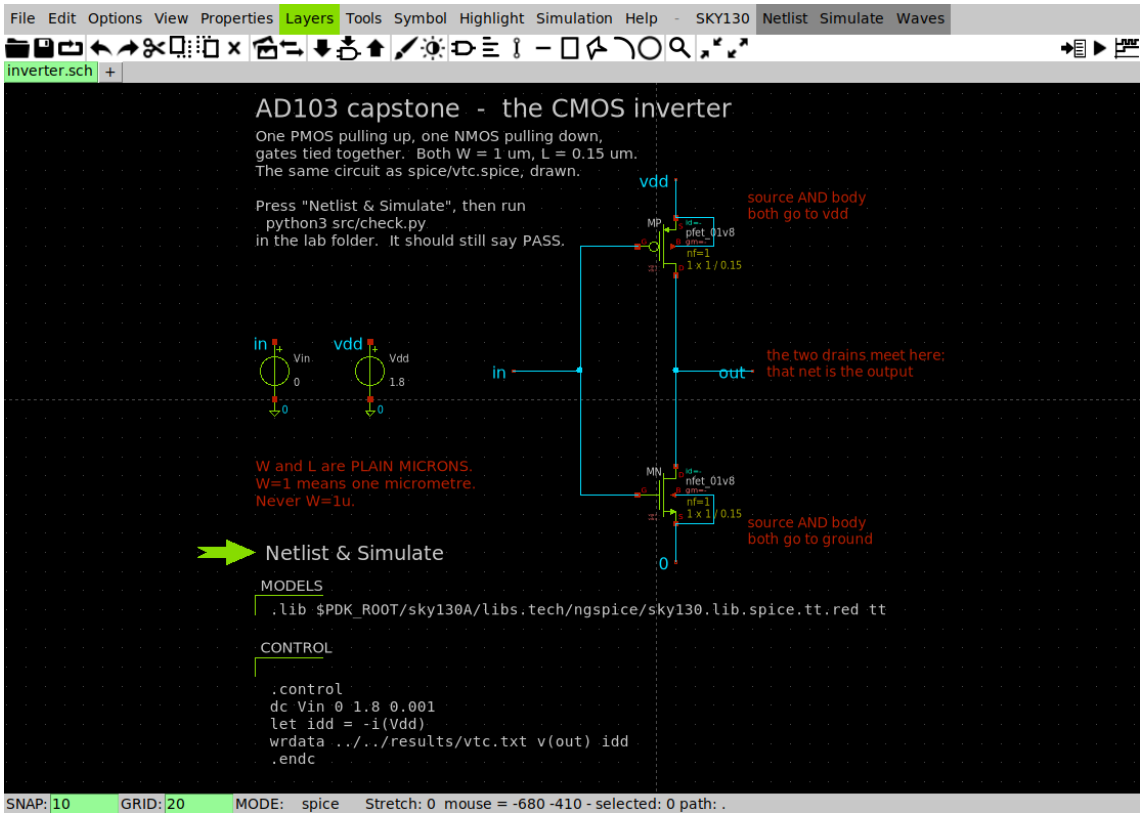


Figure 19.2: xschem/inverter.sch — the completed capstone schematic

One `pfet_01v8` above one `nfet_01v8`, with the drains tied to `out` and the gates tied to `in`. The `.lib` line and the `.control` block are carried on the same sheet in two `code_shown` symbols.

Part 3: The switching threshold

Where does an inverter with $W_n = W_p$ switch?

1. Before you look, write down where you think an inverter with $W_n = W_p$ switches. There's only one plausible guess, but write it down anyway so that you can compare it with the result.
2. Print the analysis, and read block 2:

```
make extract
```

```
switching threshold V_M       0.838027 V   (V_DD/2 would be 0.900000)
steepest slope               -13.1253     at V_in = 0.814 V
V_IL / V_IH                    0.706 / 0.940 V
V_OH / V_OL                   1.7506 / 0.0684 V
noise margin high / low        0.8106 / 0.6376 V
```

3. Open `results/ad103-inverter-vtc.png`:

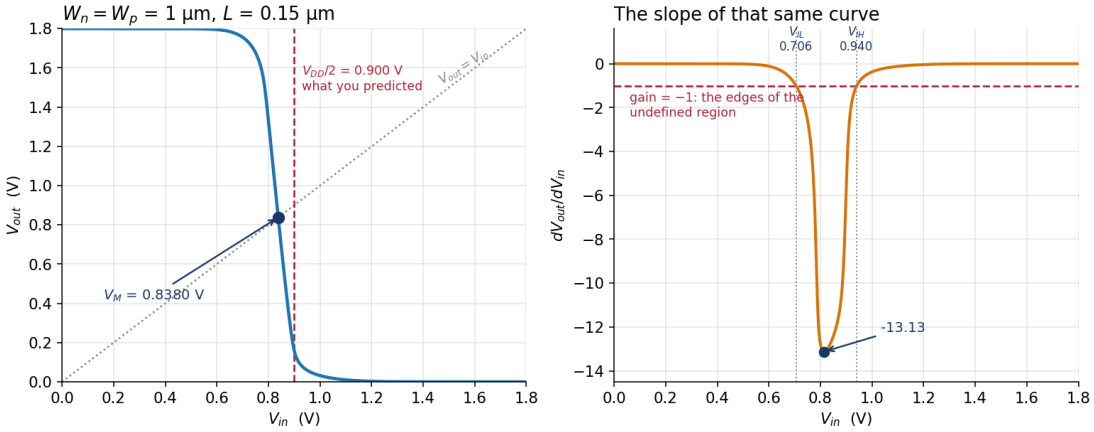


Figure 19.3: The VTC of the Wn = Wp inverter, and its slope

V_{IL} and V_{IH} are the two inputs at which the slope passes through −1, marked on the right-hand plot. Between those two inputs the gate amplifies noise instead of rejecting it, and the noise margins are the room left on either side of that region.

You should see V_M = 0.838027 V, which is **62.0 mV** below the 0.900 V of $V_{DD}/2$.
Why does the $W_n = W_p$ inverter switch below $V_{DD}/2$?
Answer: A1, in Answers at the end.

Part 4: Size the PMOS from the drive ratio

If you widen the PMOS by the drive ratio, does V_M land at 0.9 V?

1. Read block 1 of `make extract`, the drive of each transistor on its own:

```
NMOS, gate and drain at 1.8 V          501.0462 uA
PMOS, gate and drain at 0 V            200.7478 uA
NMOS / PMOS                              2.4959
```

   The 501.0462 µA is the same number Lab 01 gave you, because it's exactly the same device at exactly the same bias.

2. The PMOS is the weaker device, so the obvious fix is to widen it to match. Using the drive ratio, predict V_M for W_p = 2.5 µm.

3. Read block 3, five pull-up widths, and open `results/ad103-inverter-ratio.png`:

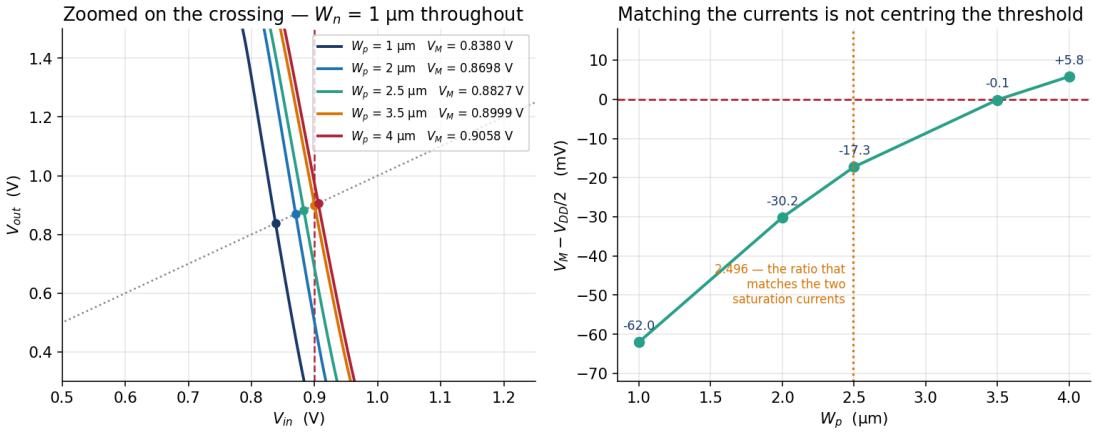


Figure 19.4: Five pull-up widths, and the error in V_M against W_p

```
Wp (um)   Wp/Wn        V_M        error vs 0.9 V      gain
1          1.00    0.838027 V      -61.97 mV     -13.1253
2          2.00    0.869826 V      -30.17 mV     -12.8834
2.5        2.50    0.882739 V      -17.26 mV     -11.8856
3.5        3.50    0.899865 V       -0.13 mV     -11.2590
4          4.00    0.905807 V       +5.81 mV     -11.2846
```

You should see 0.882739 V for W_p = 2.5 µm, which is 17.3 mV below centre instead of 62.0 mV, so it's better but still not centred. The width that does centre it is **3.5 µm, giving V_M = 0.899865 V**, 0.13 mV from the middle.

Why didn't matching the drive ratio centre V_M?
Answer: A2, in Answers at the end.

Part 5: Time the three inverters

Which pull-up width gives equal rise and fall delays?

1. Before you read it, guess which of the three inverters (W_p = 1, 2.5 and 3.5 µm) has the most equal pair of delays.
2. Read block 4 of `make extract`. The delays come from `spice/delay.spice`: a real edge, a 10 fF load, and `.meas` for rise and fall delay.

```
Wp (um)      t_pHL         t_pLH      pull-up / pull-down
1          27.7001 ps    64.6539 ps         2.334
2.5        30.2508 ps    30.7067 ps         1.015
3.5        31.7807 ps    24.1077 ps         0.759
```

You should see the W_p = 2.5 µm inverter with the most equal pair: **30.2508 ps for the output to fall against 30.7067 ps for it to rise, 1.5 % apart**. The W_p = 3.5 µm inverter, which is the one with the centred threshold, rises 24 % faster than it falls.

Why does the drive-ratio width equalise the delays, when it didn't centre V_M?
Answer: A3, in Answers at the end.

In the same table, t_{pHL} rises from 27.70 to 31.78 ps as the pull-up gets wider, although nothing about the pull-down changed. Why?
Answer: A4, in Answers at the end.

Part 6: The same circuit as an amplifier

How much voltage gain do the same two transistors give at a longer L?

1. Find what changed between the minimum-length and long-channel decks:

```
diff spice/vtc.spice spice/vtc_long.spice
```

2. Predict: will the L = 0.5 µm inverter have more gain at V_M or less? Use what Lab 04, Part 3 measured about $g_m r_o$.
3. Read block 5 of `make extract`, and open `results/ad103-inverter-gain.png`:

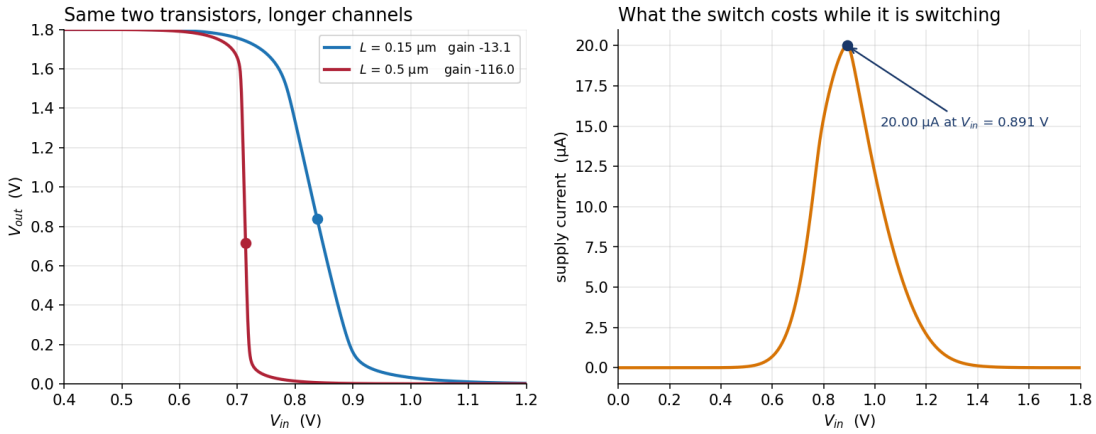


Figure 19.5: The minimum-length and long-channel inverters, and the supply current during a switch

```
L = 0.15 um, Wn = Wp = 1 um    gain  -13.1253 at V_in = 0.814 V
L = 0.5  um, Wn = Wp = 1 um    gain -116.0341 at V_in = 0.712 V
```

You should see that the two device lines differ only in `L`, 0.15 against 0.5 (the rest of the diff is comments and the output file name). The long-channel inverter has **8.84× the voltage gain for 3.33× the channel length**, from the same two transistors on the same supply.

Why does a longer channel give so much more gain, and which inverter would you use for which job?

Answer: A5, in Answers at the end.

Part 7: The current while it switches

How much supply current does an inverter draw, and when?

1. Read block 6 of `make extract`, the supply current of the $W_n = W_p$ = 1 µm inverter across its input sweep:

```
supply current at V_in = 0 V            0.000002 uA
supply current at V_in = 1.8 V          0.000320 uA
peak supply current                      20.0048 uA   at V_in = 0.891 V
```

You should see almost nothing at either end and **20.0048 µA** at V_{in} = 0.891 V.

Why is the current almost zero at both ends and largest near the middle, and what does that mean for power?

Answer: A6, in Answers at the end.

Part 8: The PMOS source on the wrong node

What happens when the PMOS source and body go to the wrong node?

1. Predict: will ngspice or the checker warn you about a PMOS whose source is on ground? Write down your guess.
2. Run the broken deck:

```
make broken
```

It is `spice/vtc.spice` with one node name changed: the PMOS source and body go to `0` instead of `vdd`.

```
The sweep ran. 1801 rows, no errors, no warnings.
Here is what it says the output does:
```

```
    V_in       V_out          supply current
    0.000 V     0.000000000 V      -0.000000 uA
    0.300 V     0.000000000 V      -0.000000 uA
    0.600 V     0.000000000 V      -0.000000 uA
    0.900 V     0.000000000 V      -0.000000 uA
    1.200 V     0.000000000 V      -0.000000 uA
    1.500 V     0.000000000 V      -0.000000 uA
    1.800 V     0.000000000 V      -0.000000 uA

  this curve never gets steeper than -1. Its steepest slope is -4.794e-15, which
means the output barely responds to the input at all. A circuit with no gain above
1 cannot restore a logic level, so it is not a usable inverter whatever else it is.

FAIL  this circuit is not an inverter
```

3. Find the change: `diff spice/vtc.spice spice/vtc_broken.spice`. Apart from comments, spacing and the output file name, the only change is on the `XP` line, where `vdd vdd` became `0 0`.

You should see an output pinned at exactly zero for all 1801 points, a supply that delivers nothing, and no complaint from ngspice. ngspice has nothing to complain about, because the netlist is perfectly valid; a PMOS with its source on ground just can't pull the output up. The check that catches this in one line, on any inverter, is the one the checker runs before anything else:

```
  V_out(0.0 V) = 1.800000 V   V_out(1.8 V) = 2.129252e-07 V
```

Read both rails first. If the output doesn't swing from one supply to the other, nothing else in the measurement means anything, and no amount of staring at V_M will tell you why.

Part 9: The current mirror

How accurately does a mirror copy a current, and what limits it?

1. Force 50 µA into a diode-connected reference and copy it four ways:

```
make mirror
```

```
== spice/mirror.spice   a 50 uA reference, copied four ways
--- part 1: the reference branch sets a gate voltage ---
v(ng) = 8.672678e-01
@m.xmref.msky130_fd_pr__nfet_01v8[vdsat] = 2.336319e-01
--- part 2: one gate voltage, three widths, V_out = 0.9 V ---
i(vo2) = -5.00878e-05
i(vo3) = -1.12313e-04
i(vo4) = -2.39347e-04
  and why the ratio is not what you drew:
@m.xmw1.msky130_fd_pr__nfet_01v8[vth] = 5.894596e-01
@m.xmw2.msky130_fd_pr__nfet_01v8[vth] = 5.766246e-01
@m.xmw1.msky130_fd_pr__nfet_01v8[gm] = 3.340839e-04
@m.xmw2.msky130_fd_pr__nfet_01v8[gm] = 7.187672e-04
--- part 4: one W=10 device vs two W=5 devices in parallel ---
i(vo3) = -1.12313e-04
i(vo5) = -1.00176e-04
--- part 3: the same 1:1 copy at several output voltages ---
V_out = 0.1 V
i(vo1) = -2.84076e-05
...
V_out = 1.8 V
i(vo1) = -5.20797e-05
```

2. Work out the 1:1 copy's error at V_{out} = 0.9 V, from `i(vo2)`.
3. In the `part 3` block (it comes after `part 4` in the output), find the output voltage below which the copy collapses, and compare it with the reference branch's `vdsat`.
4. Work out the actual ratios for the 2× and 4× outputs (`i(vo3)` and `i(vo4)` against `i(vo2)`), and the threshold difference between the 5 µm and 10 µm devices.
5. Compare one W = 10 µm device (`i(vo3)` in `part 4`) with two W = 5 µm devices in parallel (`i(vo5)`), against twice the 1:1 current.

You should see a 1:1 copy good to **0.176 %**, a collapse below about **0.2336 V**, ratios of **2.2423** and **4.7785** instead of 2 and 4, a threshold **12.835 mV** lower for the wide device, and the two parallel unit devices within **0.0004 %** of twice the unit current.

Why does one wide device copy a current less accurately than two unit devices, and why does the copy collapse below 0.2336 V?

Answer: A7, in Answers at the end.

Finishing

You're done when `make mine` passes with your own `xschem/my_inverter.sch` and reports a switching threshold of 0.838029 V (Part 2).

Questions

1. Why does the $W_n = W_p$ inverter switch below $V_{DD}/2$? (Part 3.)
2. Which W_p gives equal delays, which centres V_M, and why are they different? (Part 5.)
3. Why does one wide device copy a current less accurately than two unit devices? (Part 9.)

Submit

Nothing to submit. Keep `xschem/my_inverter.sch`, because AD104 starts from it.

Going further

Extensions

There are three, with reference answers in `solutions/`:

1. **Why the foundry's inverter isn't symmetric.** `sky130_fd_sc_hd__inv_1`, the inverter LibreLane puts in your digital designs, has W_n = 0.65 µm and W_p = 1.0 µm, a ratio of **1.54**, which is neither of this lab's two answers. You can read the sizes yourself:

   ```
   grep -A3 'subckt sky130_fd_sc_hd__inv_1 ' \
     /foss/pdks/sky130A/libs.ref/sky130_fd_sc_hd/spice/sky130_fd_sc_hd.spice
   ```

   It switches at **0.790872 V**, 109 mV below centre, and that is deliberate. Work out why before you read the reference answer.
2. **The delay arithmetic.** Write out why equal drive currents give equal delays in Part 5, and check that the arithmetic closes.
3. **More gain.** Take `spice/vtc_long.spice` to longer L and find out how much gain two transistors can give.

SiliWiz

SiliWiz, which AD102 used to look at a resistor in cross-section, is worth one more visit: draw an NMOS in it, sweep the gate, and watch the channel you've been calculating about appear under the oxide.

Your inverter in AD104

You now have a schematic of a real logic gate, sized by you against a criterion you chose and verified against a simulation to five digits. **AD104: Layout** takes this file and asks you to draw it as geometry in Magic. Every transistor becomes a rectangle of diffusion under a stripe of poly, and every net becomes metal with a width and a spacing. Then Magic runs **DRC** to check that a fab could make what you drew, and netgen runs **LVS** to show that what you drew is the same circuit as your schematic, node for node and device for device.

This is the foundry's own inverter, `sky130_fd_sc_hd__inv_1`, opened in Magic:

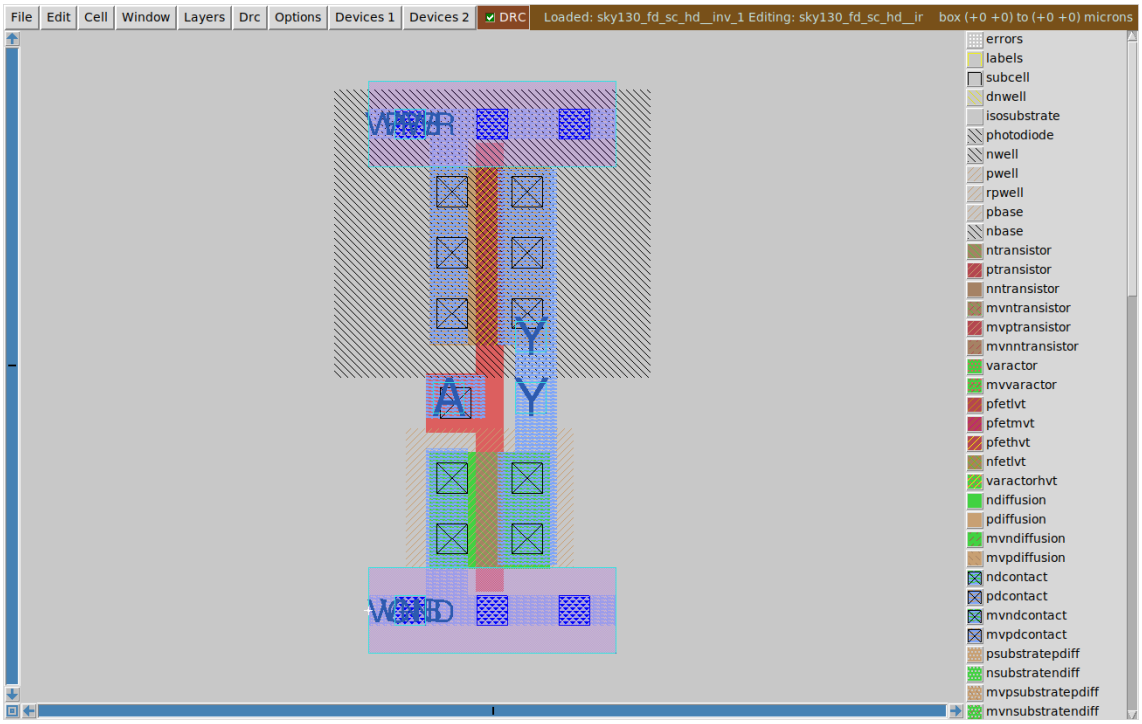


Figure 19.6: The SKY130 standard-cell inverter open in Magic: nwell and p-diffusion at the top, n-diffusion at the bottom, one poly stripe crossing both, VPWR and VGND as metal1 rails, and the full layer palette down the right-hand side

Every hatch pattern in that window is a mask layer, listed by name on the right. `A` enters on local interconnect, crosses both channels as the single poly stripe in the middle, and `Y` leaves on local interconnect. These are the nets on your schematic: `A` is `in`, `Y` is `out`, and the VPWR and VGND rails along the top and bottom are `vdd` and `0`, all drawn here as rectangles.

The geometry has to obey design rules, and Magic checks them continuously while you paint. If you drop a metal1 shape 0.04 µm from the power rail, Magic tells you before you've even finished the wire:

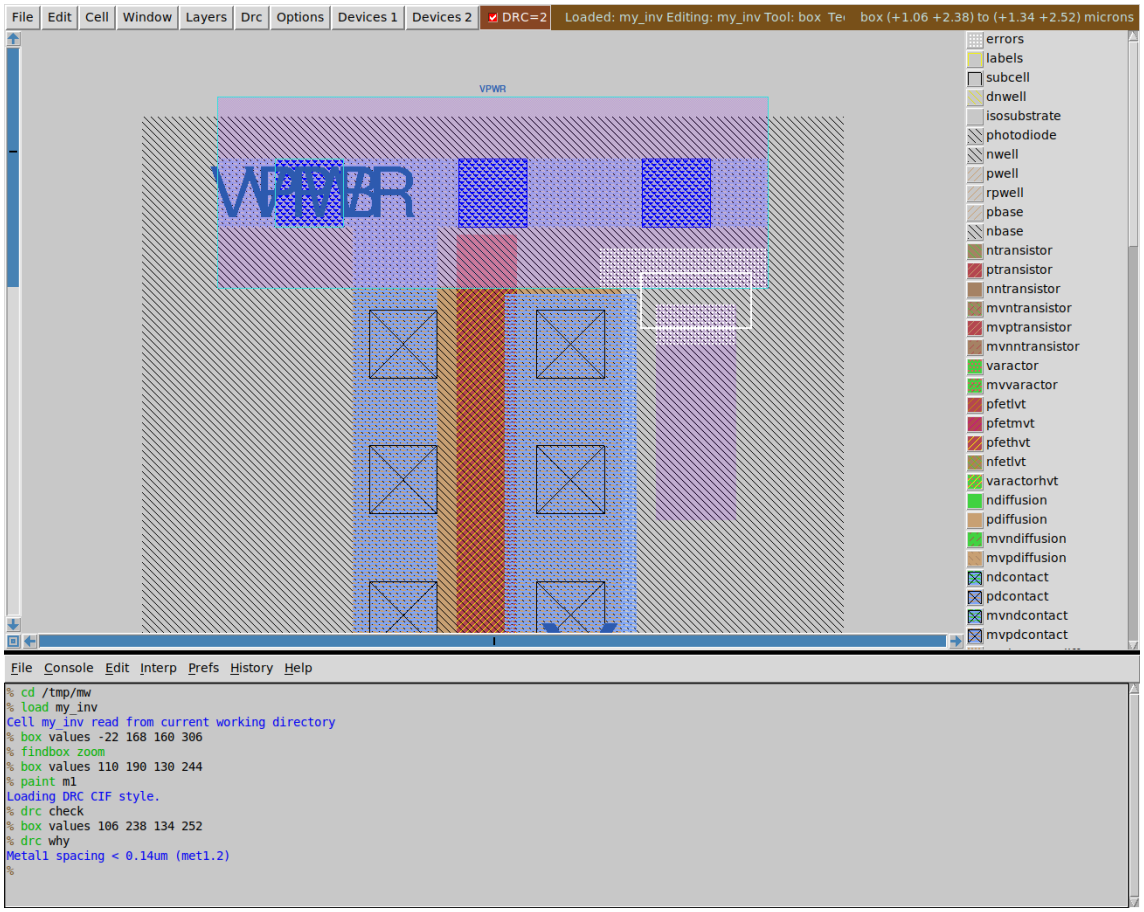


Figure 19.7: Magic showing a design-rule violation: a metal1 stub painted too close to the VPWR rail, the violation hatched in white, DRC=2 in the toolbar, and `drc why` in the console printing "Metal1 spacing < 0.14um (met1.2)"

The white hatch is the error region, `DRC=2` in the toolbar counts errors live as you edit, and `drc why` names the rule that was broken: `Metal1 spacing < 0.14um (met1.2)`.

Three things from AD103 come back in AD104:

- ***W* and *L* are geometry.** In Magic you draw them with a mouse instead of typing them, so the `W=1` versus `W=1u` trap disappears, because there's no number to put a unit on.
- **`nf=2`**, the multi-finger device from Lab 04 that changed the current by 4.5 % with no change to W or L, becomes a picture: two gate stripes sharing a drain.
- **Unit devices in an array**, from the current mirror. AD104 is where you see what a mirror looks like when it is drawn properly, dummies and all.

Stuck, or want to argue about any of this? Ask in the team Discord, and bring the number you got and the number you expected.

Next on the Course path: you've finished AD103. The next course is AD104: Layout, and it starts from your `xschem/my_inverter.sch`.

Answers

A1. Part 3: The switching threshold

The pull-down is the stronger device (Part 4 measures by how much: 2.4959×), so it takes less input voltage than you might expect for it to drag the output down. V_M is the input at which the two currents are equal. It isn't the threshold voltage of either device (769.27 mV for the NMOS, the `vth` Lab 01's `make bins` printed, and 510.03 mV for the PMOS), and it isn't their average either. That makes it the first parameter in this course that belongs to a circuit rather than to a single transistor.

A2. Part 4: Size the PMOS from the drive ratio

You matched the currents at full drive, but V_M is set by where the currents match at the crossing. At V_M the NMOS has $V_{GS} = V_M$ and the PMOS has $V_{SG} = 1.8 - V_M$, so neither device is at 1.8 V, and currents that are equal at full drive aren't equal there.

A3. Part 5: Time the three inverters

Delay is charge over current, and both edges have the same 10 fF load and the same 0.9 V swing, so making the two saturation currents equal is exactly what makes the two delays equal. Neither sizing is wrong; each one is the right answer for a different goal:

you want	set W_p to	because
equal rise and fall delay	**2.5 µm**, the current ratio	delay is charge over current
V_M at $V_{DD}/2$	**3.5 µm**	that is where the currents match *at the crossing*

Your Part 4 prediction was the right answer to a different question from the one Part 4 asked, and that is the most common way an engineering calculation goes wrong.

A4. Part 5: Time the three inverters

While the NMOS pulls the output down, the PMOS is still partly on and fights it, and a wider PMOS fights harder.

A5. Part 6: The same circuit as an amplifier

Lab 04 measured the reason: r_o rises with L, and the gain is $(g_{mn} + g_{mp})(r_{on} \parallel r_{op})$. A logic gate needs only enough gain to restore a degraded signal, and more than that doesn't help it, whereas an amplifier wants every decibel it can get. So the same two transistors do a different job depending on L: the minimum-length inverter is the logic gate, and the long-channel one is the amplifier.

A6. Part 7: The current while it switches

At either end of the sweep one of the two devices is off, so there is no path from `vdd` to ground. Near the middle both are on at once, and current runs straight from `vdd` to ground without doing any useful work. A CMOS gate therefore draws almost nothing while it holds a level and draws its current while it switches. That is why CMOS replaced everything before it, and why a processor gets hot when it computes and stays cool when it idles.

A7. Part 9: The current mirror

The wide device's threshold is 12.835 mV lower, so at the same gate voltage it carries more current than its width alone would give. The two unit devices, on the other hand, are each the same 5 µm device as the reference, so together they carry twice its current. Below about 0.2336 V, which is the reference branch's `vdsat`, the output device leaves saturation. The current mirror works through each of these.

Appendices

A. XSchem cheat sheet

This page collects the XSchem keys you need, sorted by when you'll need them.

XSchem ships 154 key bindings and 16 mouse bindings (`keys.help` in the image lists them), and most of them are a single keystroke. You'll use only eleven of them for almost all of AD103, so this page lists those eleven first, then the rest grouped by task, and then the four keys that don't do what you'd expect.

Where these keys come from

Every key on this page comes from two files inside the course image, and a key appears here only if both files agree on it. This command prints the XSchem version and then the first file, `keys.help`:

```
docker run --rm --user "$(id -u):$(id -g)" hpretl/iic-osic-tools:2026.08 --skip \
  bash -c 'xschem --version; cat /foss/tools/xschem/share/xschem/keys.help'
```

```
XSCHEM V3.4.8RC
Copyright (C) 1998-2026 Stefan Schippers
```

- `/foss/tools/xschem/share/xschem/keys.help`: the binding list XSchem itself shows.
- `/foss/tools/xschem/share/xschem/xschem.tcl`: the menu definitions, each carrying an `-accelerator` label.

The two files disagree in exactly one place: **Delete files** is `Ctrl-D` in `keys.help` but `Shift-D` in the menu, so we left that key off this sheet. Inside XSchem, ? opens the help window and / shows the whole binding list full-screen.

The XSchem window

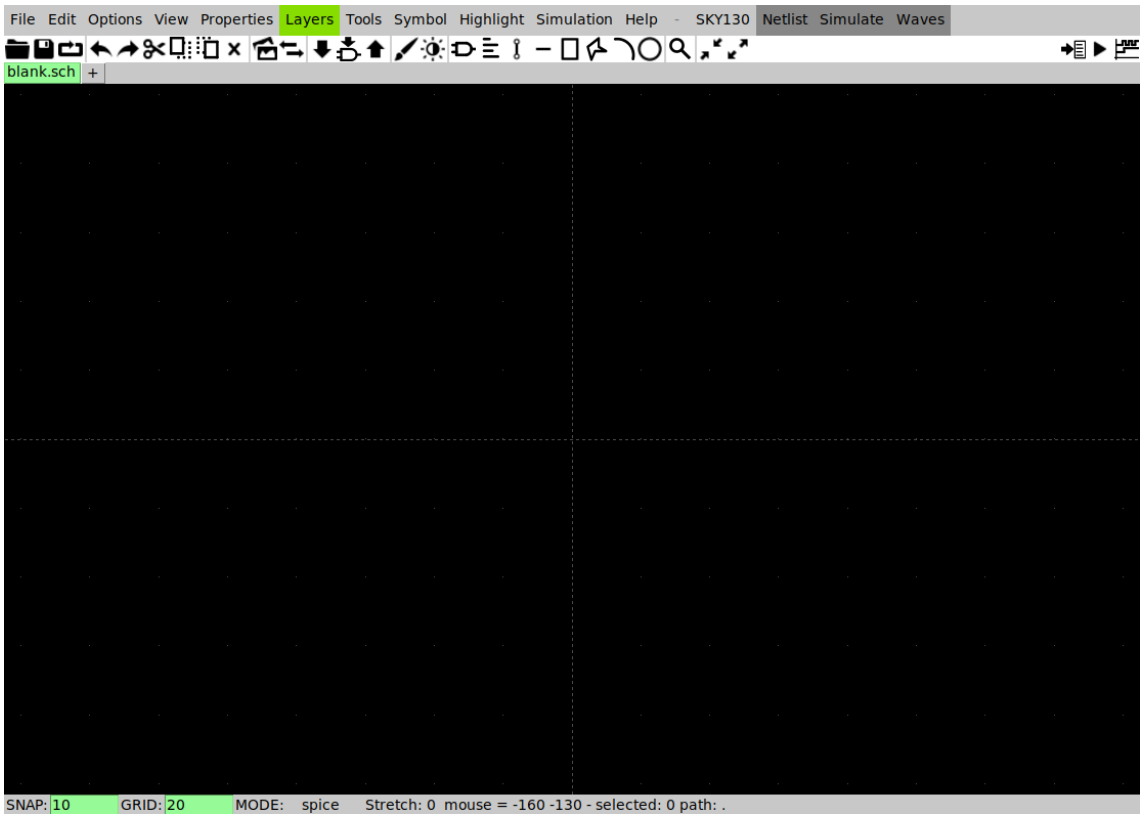


Figure A.1: The XSchem window: menu bar reading File Edit Options View Properties Layers Tools Symbol Highlight Simulation Help, then SKY130 Netlist Simulate Waves; a toolbar; a black canvas with a dotted grid; a status bar reading SNAP 10 GRID 20 MODE spice

Three parts of that window are worth knowing:

Where	What it tells you
`SKY130` in the menu bar	which PDK `xschemrc` loaded. If it says `IHP`, stop and see Wrong PDK below.
`Netlist` / **`Simulate`** / **`Waves`**	the three buttons that let you netlist, simulate and look at results without going to a terminal. The menu shows no accelerator label for them, but the keys are `n` and `s`, and for waves you use the `Waves` menu.
status bar	`SNAP: 10 GRID: 20 MODE: spice`. `MODE: spice` means the netlister will write SPICE rather than Verilog. `SNAP` is the grid your clicks land on, and a wire end that misses that grid point by one step is the most common reason a wire looks connected when it isn't.

Windows that open off screen

- **The status bar is hidden.** On the default 1280×800 desktop, XSchem opens its window 764 px tall and 29 px down from the top of the screen, which puts the bottom 24 px of it behind the Xfce taskbar. Drag the window's bottom edge up a little, or start the desktop bigger (`VNC_RESOLUTION=1920x1080 ./scripts/start_vnc.sh`).

- **The symbol chooser (Shift+I) opens partly off screen.** XSchem positions it relative to the main window, and because that window is nearly as wide as the screen, on a 1280-wide desktop the chooser lands mostly off the right edge. The library pane, the file list, the Search box and **OK** can all be out of view. Drag it back by the part of the title bar you can see (or Alt-drag anywhere in it), and it will stay where you put it for the rest of the session. The chooser also remembers the directory you were last in, so click **Home** to jump back to the top of the tree.
- The **Edit Properties** window that q opens behaves the same way.

The eleven keys you need

Do this	Key
Zoom to fit the whole schematic	f
Place a symbol from the library	Shift+I
Draw a wire	w
Edit the selected object's properties (`W`, `L`, `value`, `name`)	q
Move the selection	m
Duplicate the selection	c
Rotate the selection	Shift+R
Get out of whatever mode you are in	Escape
Undo	u
Save	Ctrl+S, then **Yes** in the `save file?` box it always raises
Write the SPICE netlist	n

Those eleven are enough to draw every schematic in AD103. The rest of the page is detail you can look up when you need it.

View

Action	Key
Zoom to fit	f
Zoom in / out	Shift+Z / Ctrl+Z, or the scroll wheel
Zoom to a box you drag	z
Pan	hold Space and drag, or drag with the middle button
Redraw / abort / unselect	Escape
Light ↔ dark colour scheme	Shift+O
Halve / double the snap grid	g / Shift+G
Help window	?
Full-screen picture of every binding	/

Place

Action	Key
Insert symbol	Shift+I, or Insert, or **Tools → Insert symbol**
Insert symbol, browser stays open	Ctrl+I (also Shift+Insert)
Wire	w
Wire whose **start** snaps to the nearest pin	Shift+W
Wire whose **far end** snaps too	Shift+W, then hold Shift for the closing click
Text label (a comment, not a net name)	t
Line / rectangle / polygon (drawing, not circuitry)	l / r / p
Net label (name a wire)	Alt+l places `lab_pin.sym`
Schematic input port	Ctrl+P
Schematic output port	Ctrl+Shift+P

Most laptops have no Insert key, so use Shift+I instead.

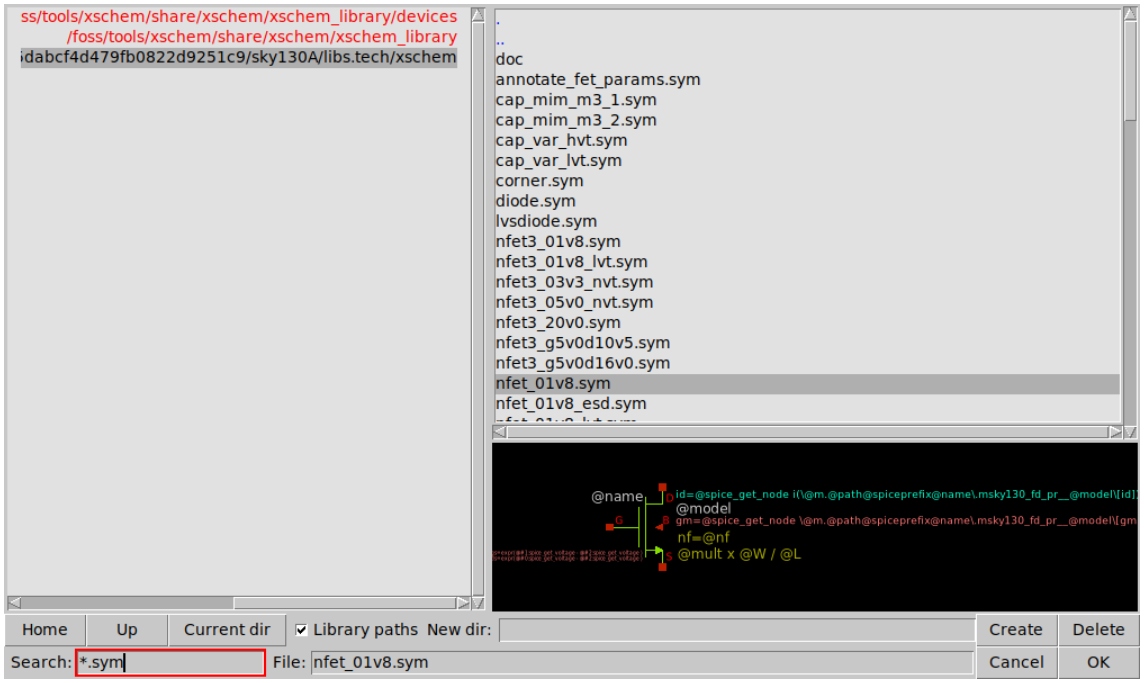


Figure A.2: The XSchem symbol chooser listing the sky130_fd_pr library: annotate_fet_params, cap_mim, cap_var, corner, diode, lvsdiode, nfet3 and nfet variants

*Shift+I opens this chooser. Here we have clicked the `sky130A` line on the left and opened `sky130_fd_pr`, so it shows the PDK's whole symbol library with no filter applied. The **Search** box takes a glob rather than a substring (`nfet_01v8*.sym`, not `nfet`), and the pane at the bottom previews whichever file is selected.*

Select and edit

Action	Key
Select	click; Shift-click adds to the selection

Action	Key
Select by area	drag with the left button
Select every connected wire, label and pin	Shift+right-click
Select all	Ctrl+A
Unselect the object under the pointer	d
Edit properties	q
Edit properties in a text editor	Shift+Q
Move	m
Move, stretching the wires that are attached	Ctrl+M
Duplicate	c
Delete	Delete
Rotate 90°	Shift+R
Flip left–right / top–bottom	Shift+F / Shift+V
Rotate or flip **in place**, each object about its own anchor	Alt+r / Alt+f / Alt+v
Undo / redo	u / Shift+U
Descend into a symbol's schematic	e
Descend into a symbol's *drawing*	i
Come back up	Ctrl+E or Backspace
Find by name or regexp	Ctrl+F
Tcl console	=

Keys that surprise you

Your hands learned what these four keys do in other editors, but in XSchem each of them does something else.

You press	You expect	XSchem does
r	rotate	starts drawing a **rectangle**
x	flip in x	**starts a second XSchem process**, with a whole new window
c	copy to clipboard	**duplicates** the selection immediately, attached to the pointer
p	plot this net	starts drawing a **polygon**

x is the worst of the four, because the second window looks exactly like your first, and you can spend a minute editing the wrong copy. We checked this with a scripted run under Xvfb that starts one XSchem, sends it a single key, and then counts the XSchem processes:

```
### rect    : xschem processes now = 1     (key 'r')
### poly    : xschem processes now = 1     (key 'p')
### copy    : xschem processes now = 1     (keys ctrl+a, 'c')
### newsess : xschem processes now = 2     (key 'x')
```

A single press of x started one extra process. The other three rows are the control, and they show that none of those keys starts a second editor.

The rotate you want is Shift+R, the flip is Shift+F, the clipboard copy is Ctrl+C, and the way to look at a waveform is the **Waves** menu. If you land in rectangle or polygon mode by accident, Escape gets you out and u removes anything you drew.

Checking wire connections

How XSchem draws a wire end or a symbol pin tells you how many connections it has:

What you see	What it means
hollow square	connected to nothing
nothing drawn	exactly one connection
solid dot	two or more connections

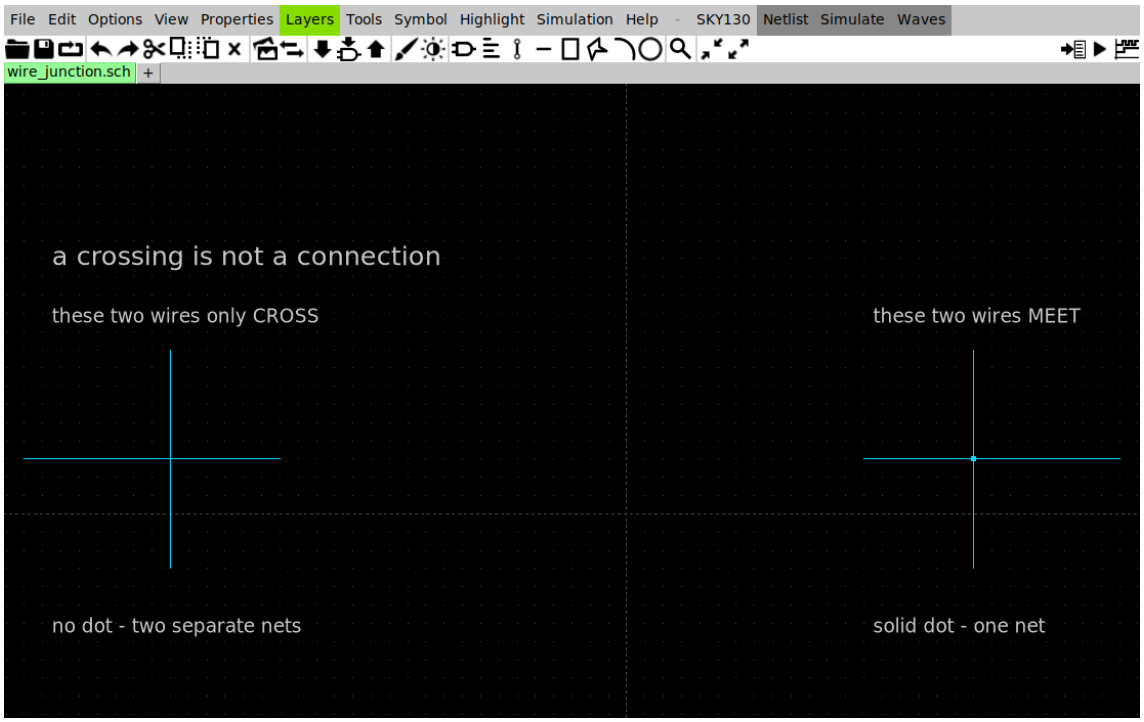


Figure A.3: Two pairs of XSchem wires side by side: on the left they cross with nothing drawn at the crossing, on the right they meet at a small solid dot

On the left, two wires cross and XSchem draws nothing at the crossing. On the right, four wire ends land on one grid point and XSchem draws the dot. The dot is the only difference between the two pictures.

Crossing wires do not connect. Two wires connect only where one wire's *endpoint* lands on the other wire or on a pin, and "lands on" means on the same snap grid point, which is why the `SNAP: 10` in the status bar matters. A wire end that stops one grid step short looks identical to a connected one at normal zoom, so press f, zoom in on every junction, and look for hollow squares.

You can also connect by name: two `lab_pin` symbols carrying the same label are one net, however far apart they are drawn. AD103's testbenches use this for supplies and gates.

Netlist and simulate

Action	How
Write the SPICE netlist for this schematic and everything under it	n, or the **Netlist** button
Netlist only this level, no hierarchy	Shift+N
Run the simulator	s (it asks `Run circuit simulation?` first), or the **Simulate** button (which runs without asking)
Look at results	the **Waves** menu: `Op`, `Dc`, `Ac`, `Tran`
Annotate DC operating point onto the schematic	**Waves → Op Annotate**
Fire a launcher (one of the green arrows on a testbench)	click it once, then Ctrl+H

XSchem documents launchers as Ctrl-click, and Ctrl-click does work, but it misses if your hand drifts, because the launcher fires only if the selection is exactly one object *and* the pointer did not move between press and release. If the pointer drifts by three pixels, nothing happens and XSchem gives no message. Clicking the arrow to select it and then pressing Ctrl+H doesn't depend on a steady hand, so the labs use that instead.

Netlisting from a terminal

```
export PDK=sky130A
cd labs/lab-01-first-schematic/xschem
xschem -n -q -x nmos_probe.sch
```

The flags are `-n` to write the netlist, `-q` to quit when it's done, and `-x` to run without graphics, and this is the command every AD103 `Makefile` runs. **The `export` is required**: if you leave it out, you get the silent failure described below. The command writes `simulation/nmos_probe.spice` next to the schematic:

```
** sch_path: /work/xschem/nmos_probe.sch
**.subckt nmos_probe
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 ad=0.29 as=0.29 pd=2.58 ps=2.58
nrd=0.29 nrs=0.29 sa=0 sb=0 sd=0 mult=1
Vgs g 0 1.8
Vds d 0 1.8
**.ends
.end
```

Notice the parameters the symbol added that you never typed: `ad`, `as`, `pd`, `ps`, `nrd` and `nrs`. They describe the source and drain diffusion geometry, which the symbol computes from `W` and `nf`. The junction capacitances that come from that geometry are part of the delay, so a deck you write by hand without them simulates faster than the real circuit, and its delays are optimistic.

`-o OUTDIR` is ignored when `xschemrc` contains `set local_netlist_dir 1`, and every AD103 lab's `xschemrc` does, as this run shows:

```
$ xschem -n -s -q -o out nmos_probe.sch ; echo "exit=$?"
exit=0
$ ls out
$ ls simulation
nmos_probe.spice
```

XSchem exits with status 0 and leaves `out/` empty, because the netlist went somewhere else. Look for it in `simulation/`, next to the schematic.

Failures that exit 0

XSchem exits with status 0 even when the netlist it wrote is broken, so its exit status doesn't tell you whether it worked. Check the netlist itself instead of `$?`.

Wrong PDK

```
docker run --rm ... hpretl/iic-osic-tools:2026.08 --skip bash -c 'echo "PDK=$PDK"'
```

```
PDK=ihp-sg13g2
```

The image ships with the IHP SG13G2 PDK selected rather than SKY130. XSchem reads `$PDK` at launch to decide which symbol library to load, and if it picks the wrong one, every `sky130_fd_pr` symbol resolves to nothing. XSchem still exits with status 0, and the netlist looks like this:

```
** sch_path: /work/xschem/nmos_probe.sch
**.subckt nmos_probe
*  M1 -  nfet_01v8  IS MISSING !!!!
Vgs g 0 1.8
Vds d 0 1.8
**.ends
.end
```

The menu bar tells you before the netlist does, showing `IHP` instead of `SKY130`:

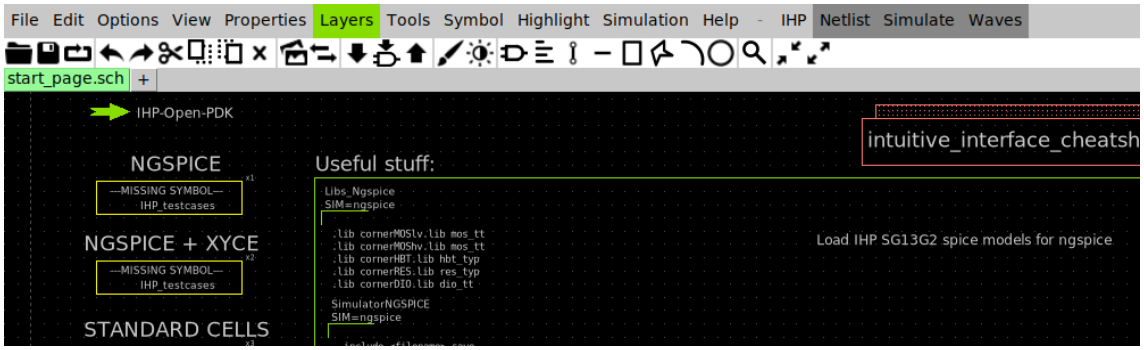


Figure A.4: The XSchem menu bar reading IHP instead of SKY130, above the IHP-Open-PDK start page

`docker run -e PDK=sky130A` does not fix it, because the image's entrypoint sets `PDK` after Docker does and overwrites the value you passed with `-e`. We checked:

```
docker run --rm -e PDK=sky130A ... --skip bash -c 'echo "PDK=$PDK"'
```

```
PDK=ihp-sg13g2
```

Instead, set it *inside* the container's command: run `export PDK=sky130A`, or set it on the command itself (`PDK=sky130A xschem -n -q -x nmos_probe.sch`), or let the lab `Makefile` do it. Every AD103 lab that ships a schematic already does this, with `export PDK := sky130A` near the top of the `Makefile` in `lab-01-first-schematic`, `lab-02-diode-iv`, `lab-03-mosfet-regions` and `capstone-inverter`. `lab-02-diode-iv` goes further and pins `PDK` inside its own `xschem/xschemrc`, because that lab promises to run with no environment set up at all and its bench has to keep that promise when you open it by hand. (`lab-04-wl-knob` and `survival-card` ship no schematic and never launch XSchem; `survival-card` pins only `PDK_ROOT`, which is all its `.lib` lines need.)

Make this check a reflex: `grep -c 'IS MISSING' xschem/simulation/*.spice` must print `0`.

The others

Symptom	Cause
`IS MISSING !!!!` and no devices in the netlist	XSchem can't find the symbol, either because `$PDK` is wrong or because the symbol is not on `XSCHEM_LIBRARY_PATH`.
netlist looks right, ngspice says `could not find a valid modelname`	a `u` suffix on `W` or `L`. See Reading a SKY130 device model.
a net you were sure you drew is missing from the netlist	a wire endpoint that stops one snap step from the pin. Zoom in and look for a hollow square.
ports wired to the wrong nets in a hierarchical cell	the `.sym` pin order does not match the order the `ipin`/`opin`/`iopin` symbols appear in the schematic.
`can't create directory ".../sky130_tests/simulation": permission denied` on **Netlist**	the open schematic is the PDK's own start page, `sky130_tests/top.sch`, which lives inside the read-only PDK. With `local_netlist_dir 1` the netlist goes next to the schematic, and XSchem can't write anything there. Open a schematic in your own directory. The AD103 lab `xschemrc` files also set `XSCHEM_START_WINDOW {}` so a bare `xschem &` never lands there.
`file opening for write failed!` on save	the same cause: you're editing a file in a read-only directory. Your drawing is still in the window, so use **File → Save as** to save it into your own folder.
`net1`, `net2`, … in the netlist	a pin that touches nothing. XSchem invents a name for it and exits 0, and Lab 01, Part 7 shows what ngspice does with one. You can check for them with `grep 'net[0-9]' simulation/*.spice`.
a blank rectangle with a name instead of a circuit	an unexpanded subcell. Descend into it with e, come back with Ctrl+E.
XSchem opens the wrong start page, or symbols vanish after it worked yesterday	you started it from a different directory, so it read a different `./xschemrc`.
`make mine` says `FAIL - the netlist has no nfet_01v8` right after you drew one	you didn't press **Yes** in the **Ask Save** window, so `make mine` netlisted the last version you saved.

Where files land

Thing	Path
Netlists, with `set local_netlist_dir 1`	`simulation/` beside the schematic
Netlists, without it	`~/.xschem/simulations/`, inside the container, so it is gone when the container is removed. This default is documented on line 62 of the PDK's own `xschemrc`.
XSchem's own config	XSchem looks for `--rcfile FILE`, then `./xschemrc` in the launch directory, then `~/.xschem/xschemrc`, and reads only the first one

Thing	Path
	it finds, so a project `./xschemrc` replaces the home one rather than adding to it
The window XSchem opens with no filename	`XSCHEM_START_WINDOW`, which the PDK's `xschemrc` sets to `sky130_tests/top.sch` (read-only). If you set it to `{}`, you get a blank `untitled.sch` in the launch directory
SKY130 symbols	`/foss/pdks/sky130A/libs.tech/xschem/sky130_fd_pr/`
Generic symbols (`gnd`, `vsource`, `lab_pin`, `ipin`)	`/foss/tools/xschem/share/xschem/xschem_library/devices/`

There is no `XSCHEMRC` environment variable (the string doesn't appear once in XSchem's `xschem.tcl`), so setting one does nothing.

Next comes the other half of the loop, The ngspice survival card, which covers what happens after you press n.

B. The ngspice survival card

This page covers the analyses you can ask ngspice to run on a netlist, and what ngspice says when you get one wrong.

Four analyses, and the `meas` command that turns their results into numbers, cover everything in AD103 and most of second year. Each deck below is a complete file that ships in a lab package, so you never have to retype one, and every output shown on this page came from running that file in `hpretl/iic-osic-tools:2026.08` with ngspice 47.

```
cd labs/survival-card
make
```

runs the six decks in that package in about a minute and checks every number those decks print on this page against what ngspice has just printed, so the page and the decks can't drift apart without the check failing:

```
PASS  every number on the survival card reproduces
```

To run a deck of your own:

```
ngspice -b mydeck.spice
```

`-b` is batch mode, in which ngspice runs the deck and then exits. Without it you land in ngspice's interactive prompt, which waits for you to type a command, so the terminal looks as if it has hung.

The `.lib` line

Every deck starts the same way:

```
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
```

That line loads the typical-typical (`tt`) corner of SKY130. There are two things worth knowing about it:

- **Loading it is slow.** The `tt` section of `sky130.lib.spice` pulls in a tree of 169 model files. One single-deck run, such as Lab 01's `make`, takes about ten seconds on the reference machine, while a deck with no `.lib` line at all runs in about 0.01 s. Almost all of those ten seconds go on loading models, so if the terminal stays silent that long, ngspice hasn't hung.
- **`$PDK_ROOT` is already set** to `/foss/pdks` inside the image, so the line works with no setup. `$PDK` is *not* set to what you want (see the XSchem cheat sheet), but this line doesn't use it, so that doesn't matter here.

1. `.op`: one bias point

"With these DC voltages applied, what is every node voltage and every current?"

The deck is **`labs/lab-01-first-schematic/spice/nmos_op.spice`**, and you have already run it, because it's the deck Lab 01's `make` uses. Leaving out the `nf=1 m=1` that XSchem writes, it reads:

```
* .op -- one bias point of one NMOS
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1
Vgs g 0 1.8
Vds d 0 1.8
.control
op
print i(Vds)
```

```
.endc
.end
```

```
i(vds) = -5.01046e-04
```

That is 501.046 µA, and it's negative because SPICE measures the current flowing *into* a voltage source's positive terminal. The current leaves `Vds`, goes down through the transistor, and comes back into the source the other way, so every drain current you print through a supply comes out negative. That's the sign convention working as intended, and it's the most common cause of "my answer has the wrong sign" in first year. `make` in that lab prints:

```
  drain current : 501.046 uA  (reference 501.046 uA)

PASS  netlist and operating point match the reference run
```

`.op` also gives you the small-signal parameters that the device-parameter pages use. You ask for them by name:

```
print    @m.XM1.msky130_fd_pr__nfet_01v8[vth]    @m.XM1.msky130_fd_pr__nfet_01v8[gm]
@m.XM1.msky130_fd_pr__nfet_01v8[gds] @m.XM1.msky130_fd_pr__nfet_01v8[cgg]
```

```
@m.xm1.msky130_fd_pr__nfet_01v8[vth] = 7.692674e-01
@m.xm1.msky130_fd_pr__nfet_01v8[gm] = 5.322242e-04
@m.xm1.msky130_fd_pr__nfet_01v8[gds] = 5.141539e-05
@m.xm1.msky130_fd_pr__nfet_01v8[cgg] = 8.568632e-16
```

Those four lines come from `labs/survival-card/spice/op_nmos.spice`, which is the deck above with that `print` line added, and `make op` in the survival-card package runs it.

The name is ugly, and you can't shorten it. `XM1` is *your* instance, and inside it the PDK's subcircuit builds a real MOSFET called `msky130_fd_pr__nfet_01v8`; `@m.XM1.<that>[...]` is the path down to that inner device. If you get any part of the path wrong, ngspice says `Error: no such vector`.

2. `.dc`: sweep a source

"Sweep this voltage and give me a curve."

The deck is **`labs/survival-card/spice/dc_id_vgs.spice`**, and `make dc` runs it:

```
* AD103 survival card, analysis 2 -- .dc, sweep a source
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=1 W=5
Vgs g 0 0
Vds d 0 1.8
.control
dc Vgs 0 1.8 0.01
let id = -i(Vds)
meas dc id_1v8 find id at=1.8
meas dc id_0v9 find id at=0.9
meas dc vg_at_1ua when id=1u rise=1
.endc
.end
```

```
No. of Data Rows : 181
id_1v8              =  6.96275e-04
id_0v9              =  6.38760e-05
vg_at_1ua           =  5.65815e-01
```

There are 181 rows because 0 to 1.8 in steps of 0.01 is 181 points, counting both ends. It's worth checking the row count every time, because if it isn't the number you expect, the sweep isn't the one you think you asked for.

The `dc` command takes `source start stop step`. If you add four more arguments for a second, outer sweep, you get a whole family of curves in one run:

```
dc Vds 0 1.8 0.01 Vgs 0.6 1.8 0.3
```

(That's `labs/survival-card/spice/dc_family.spice`, which `make dc` also runs.)

```
No. of Data Rows : 905
id[180] = 2.005399e-06
id[181] = 1.881519e-42
```

905 rows is 5×181: ngspice ran five gate voltages, each with a full V_{DS} sweep, and laid them end to end in a single vector. Index 180 is the last point of the first curve, and index 181 is the first point of the second, back at $V_{DS} = 0$. That's why `labs/lab-03-mosfet-regions/spice/id_vds.spice` uses `foreach` and `alter` instead. Five separate `dc` commands give you five separate plots, which you can address as `dc1.i(vds)` … `dc5.i(vds)`, and that's what you want when each curve is going to become its own column in a file.

To get data out of ngspice and into a plotting program, use `wrdata`:

```
wrdata ../results/id_vds.txt id_vgs06 id_vgs09 id_vgs12 id_vgs15 id_vgs18
```

It writes plain text with one x,y pair per vector on every row, so five vectors give you ten columns, with the x column repeated five times. Here are the first two rows of the file that command writes:

```
  0.00000000e+00   1.78502380e-42   0.00000000e+00   1.88151889e-42   0.00000000e+00
1.97801397e-42  ...
  1.00000000e-02   3.75258957e-07   1.00000000e-02   3.95382562e-06   1.00000000e-02
7.76418855e-06  ...
```

Make the vector with `let` before you `meas` it. If you write `-i(Vds)` directly inside `print` or `meas`, you get:

```
Error: no such vector as -i(vds).
 meas dc id_at_1v8 find -i(vds) at=1.8 failed!
```

`i(Vds)` is a vector, but `-i(Vds)` is an *expression*, and `meas` wants the name of a vector. So make the vector yourself with `let id = -i(Vds)`, and then give `meas` the name `id`.

3. `.tran`: the time domain

"Step the input and watch the output move."

The deck is **`labs/survival-card/spice/tran_rc.spice`**, and `make tran` runs it:

```
* AD103 survival card, analysis 3 -- .tran, the time domain
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
Vin in 0 pulse(0 1.8 1n 10p 10p 50n 100n)
XR1 in out sky130_fd_pr__res_generic_po W=1 L=200
XC1 out 0 sky130_fd_pr__cap_mim_m3_1 W=10 L=10 MF=1 m=1
.control
tran 5p 40n
meas tran tau when v(out)=1.1376 rise=1
meas tran vfinal find v(out) at=39n
.endc
.end
```

```
No. of Data Rows : 8014
tau                =  3.10826e-09
vfinal             =  1.80000e+00
```

The command is `tran <step> <stop>`, and ngspice treats the step as a suggestion. It takes smaller steps where the waveform is moving quickly, which is why 40 ns at 5 ps produced 8014 rows rather than 8000.

The source's arguments are `pulse(v1 v2 delay risetime falltime width period)`, so here the edge starts at 1 ns.

`meas ... when` returns an absolute time, measured from zero, rather than an interval. 1.1376 V is 0.632×1.8, so `tau` should mark one time constant, and 3.10826 ns is one time constant *after the edge*

at 1 ns. The time constant itself is $3.10826 - 1 = 2.10826$ ns. If you forget to subtract the delay, your RC comes out 47 % too big.

4. `.ac`: the frequency domain

"Same circuit. Where does it stop responding?"

The deck is **`labs/survival-card/spice/ac_rc.spice`**, which `make ac` runs, and it uses the same two devices as the `.tran` deck:

```
* AD103 survival card, analysis 4 -- .ac, the frequency domain
.lib $PDK_ROOT/sky130A/libs.tech/ngspice/sky130.lib.spice tt
Vin in 0 dc 0 ac 1
XR1 in out sky130_fd_pr__res_generic_po W=1 L=200
XC1 out 0 sky130_fd_pr__cap_mim_m3_1 W=10 L=10 MF=1 m=1
.control
ac dec 200 10k 1G
meas ac f3db when vdb(out)=-3.0103 fall=1
.endc
.end
```

```
No. of Data Rows : 1001
f3db                =  7.56449e+07
```

The command is `ac dec <points-per-decade> <fstart> <fstop>`, and five decades at 200 points each gives 1001 rows. The source needs `ac 1` on it. With only `dc 0`, the source has no AC amplitude, so you get zero everywhere, with no warning, and a flat line at $-\infty$ dB.

`vdb(out)` is $20\log_{10}|v(\text{out})|$. Half power is -3.0103 dB rather than -3, and using -3 shifts your answer by about 0.1 %. That's fine until you compare the result against the hand calculation.

The two analyses have to agree

You can measure the resistor and the capacitor in those two decks on their own, with **`labs/survival-card/spice/rc_parts.spice`**, which `make parts` runs:

```
r = 1.018463e+04
cmim = 2.065822e-13
```

$$RC = 10184.63 \times 206.5822 \times 10^{-15} = 2.10396\ \text{ns} \tag{2.1}$$

$$f_{-3\ \text{dB}} = \frac{1}{2\pi RC} = 75.6453\ \text{MHz} \tag{2.2}$$

`.ac` measured 75.6449 MHz, which is 0.0005 % from the arithmetic. `.tran` measured 2.10826 ns, 0.20 % from the arithmetic, and it's high rather than randomly off, because a 5 ps step can't land exactly on the crossing. When two analyses and a hand calculation agree this closely, you can be confident the deck is correct. It's worth making this a habit: whenever you have an RC, the three numbers are tied together by $f_{-3\ \text{dB}} = 1/(2\pi RC)$, so one of them comes free as a check on the others. `make` in that package prints this comparison at the end.

5. `.meas`: a number instead of a picture

`meas` gets a section number here, but it isn't a fifth analysis. It's the command the other four use to produce numbers instead of plots. It runs *after* an analysis, in the same `.control` block, and prints one line.

Form	Means
`meas dc <name> find <vec> at=<x>`	the value of `<vec>` at sweep point `<x>`
`meas tran <name> find <vec> at=<t>`	the value at time `<t>`
`meas tran <name> when <vec>=<val> rise=1`	the **time** of the first rising crossing

Form	Means
`meas ac <name> when vdb(out)=-3.0103 fall=1`	the **frequency** of the first falling crossing
`meas tran <name> avg <vec> from=<t1> to=<t2>`	the average over a window
`meas tran <name> trig v(a) val=0.9 rise=1 targ v(b) val=0.9 fall=1`	a propagation delay

Each of these three rules has cost people an evening:

1. The analysis type in `meas dc` / `meas tran` / `meas ac` must match the analysis you just ran, or the measurement fails without a clear message.
2. `when` gives you the x-axis value (time, frequency, or the swept voltage), and `find` gives you the y-axis value.
3. A failed `meas` prints `meas <name> ... failed!`, and **ngspice keeps going and exits 0**. So read the output rather than trusting the exit code.

Harmless messages

OSDI `Error:` lines

On some builds of the workbench image, every run opens with a block like this, printed before ngspice has read a single character of your circuit:

```
Error opening osdi lib "/foss/pdks/sky130A/libs.tech/ngspice/osdi/psp103.osdi": No
such file or directory!
Error: Library /foss/pdks/sky130A/libs.tech/ngspice/osdi/psp103.osdi couldn't be
loaded!
Error opening osdi lib "/foss/pdks/sky130A/libs.tech/ngspice/osdi/psp103_nqs.osdi": No
such file or directory!
Error: Library /foss/pdks/sky130A/libs.tech/ngspice/osdi/psp103_nqs.osdi couldn't be
loaded!
...
Warning: OSDI libs have not been loaded successfully.
    Any of the following steps may fail, if Verilog A models are involved!.
```

It looks alarming, but you can ignore it. OSDI is ngspice's plug-in interface for compiled Verilog-A models. SKY130's `nfet_01v8`, `pfet_01v8`, resistors and diodes all use *built-in* BSIM and diode models, so none of those libraries is ever needed, and the image doesn't ship them. Runs that print this block go on to produce numbers quoted to six figures.

These lines say `Error`, but they are **not** followed by `Simulation interrupted due to error!`, and the run exits `0`. A line starting with `Error:` isn't enough on its own to mean the run failed, so look for the pair. Every AD103 `Makefile` sends ngspice's output to a log, so these lines reach your terminal only when you run `ngspice -b` yourself.

The warning block from resistors and diodes

Whatever you run against this PDK, stderr fills with warnings, and they are harmless. This is the complete stderr, 18 lines of it, from a deck that instantiates eight resistors and a capacitor and gets every number right:

```
Warning: Model issue on line 4842 :
  .model xr2: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: sky130_fd_pr__model__parasitic__diode_ps2nw: IKR too small - model effect
```

```
disabled!
Warning: sky130_fd_pr__diode_pw2nd_05v5: IKR too small - model effect disabled!
```

- **`unrecognized parameter (sw_et) - ignored`** (10 of them): the PDK's resistor model cards carry parameters written for a different simulator. ngspice drops them and still models the resistor correctly.
- **`IKR too small - model effect disabled!`** (2): IKR is a high-injection knee current, and the models set it to zero. The warning appears whenever a junction diode is in the deck, which is always, and it has nothing to do with your circuit. Every Lab 02 `results/*.log` contains it, and `grep -c 'Simulation interrupted' results/*.log` still prints `0` for every file.
- **`Note: No compatibility mode selected!`** prints on every run, including perfect ones. It's ngspice telling you that it isn't pretending to be HSPICE.

Warnings go to stderr, and results go to stdout. So `ngspice -b deck.spice > results.log` gives you a clean log and leaves the noise on your terminal. A real problem looks different from these harmless messages: it says `Error on line …`, followed by `Simulation interrupted due to error!` and then `no simulations run!`. If you see numbers after the warnings, the warnings didn't matter.

Eta0 is negative

```
Warning: Eta0 = -0.0310679 is negative.
```

SkyWater's model card has a negative value for this fitting parameter, which BSIM allows but comments on. The value is legal and deliberate, and the run is unaffected.

Every Lab 04 run leaves the same note behind in a three-line file, `bsim4v5.out`, in the lab folder:

```
Checking parameters for BSIM 4.5 model xf:sky130_fd_pr__nfet_01v8__model.5
Warning: Eta0 = -0.0310679 is negative.
```

The deck that writes it is `spice/double_w.spice`, and `xf` is that deck's `nf=2` device. BSIM's parameter checker writes this file whenever a model card carries a value it wants to comment on, which here is `nfet_01v8` bin 5. `make clean` removes the file.

Error messages

ngspice says	You did
`could not find a valid modelname`	put a `u` on `W` or `L`, or asked for a size no bin covers. Full explanation: Reading a SKY130 device model.
`Error: no such vector as -i(vds).`	put an expression where `print`/`meas` wanted a vector. Make the vector with `let id = -i(Vds)` first.
`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` rather than a `.subckt`. `res_generic_l1` and the metal resistors are `R` lines: `r6 f 0 sky130_fd_pr__res_generic_l1 w=1 l=1`.
`warning, can't find model 'sky130_fd_pr__nfet_01v8' from line`	started the line with `M` for a device that *is* a `.subckt`. Transistors are `X` lines.
`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 SKY130 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*, despite what the message says. The varactor has three terminals, not two.

ngspice says	You did
`Error: incomplete or empty netlist / no simulations run!`	nothing by itself. This is the second message, and the real error is a few lines above it.
`Error: Library …/osdi/psp103.osdi couldn't be loaded!`	nothing at all. Some image builds print several of these at the start of every run; see above.
`Error: unknown subckt: xd1 a 0 sky130_fd_pr__diode_pw2nd_05v5 area=1 perim=4`, then `Error: incomplete or empty netlist`	one of two things: there is **no `.lib` line**, or the **model name is misspelled**. The error quotes your device line rather than the cause. Check the `.lib` line first, because that's the more common one.
`make: ngspice: No such file or directory`	ran `make` on your own machine instead of in the workbench container. The labs need `hpretl/iic-osic-tools:2026.08`.

Diode sizes that give no error

ngspice accepts all of these without a word and gives you the wrong current. Lab 02, Part 7 has you run each one.

You see	You did
a current about **66×** larger than expected	wrote `pj=` instead of `perim=`. `pj` is ignored and `perim` keeps its default of 1e6.
a current about **a million times** smaller (picoamps where you expected microamps)	put a `u` on `area` or `perim`. Diode sizes are bare micron numbers.
megaamps	gave no size at all, so the subcircuit's defaults, `area=1e12 perim=1e6`, apply, which describe a junction one metre square. An unedited XSchem `diode.sym` writes the same area, with `perim=4e6`.

In Lab 02, if the checker says `no sweep data at …`, either `make sweep` hasn't run yet or `make clean` removed its output. Run `make` to fix it.

Exit codes

These are the exit codes we measured on the decks on this page:

Deck	exit
a correct `.op`	`0`
`W=1u`: `could not find a valid modelname`	`1`
a deck whose `meas` failed	`0`
a deck with no `.control` block at all	`0`
any of the above, with the OSDI `Error:` lines	unchanged

So `$?` catches a deck that wouldn't build, but not a failed measurement or a deck that never asked for one. That's why every AD103 lab `Makefile` reads the log and prints its own `PASS` or `FAIL` instead of trusting ngspice's exit status, and any script of yours should do the same.

Next: Reading a SKY130 device model, where `could not find a valid modelname` gets its full explanation.

C. Reading a SKY130 device model

This page is about where a transistor's numbers come from, and why `W=1u` breaks everything.

You have been writing `sky130_fd_pr__nfet_01v8 L=0.15 W=1` since Lab 01 without anyone telling you what is on the other end of that name, so here we open it up and look inside. Along the way we explain the error message that every AD103 student meets sooner or later:

```
could not find a valid modelname
```

Where the models live, and which one you are using

```
ls /foss/pdks/sky130A/libs.tech/ngspice/
ls /foss/pdks/sky130A/libs.ref/sky130_fd_pr/spice/ | wc -l
```

Path	What is in it	You use it
`libs.tech/ngspice/sky130.lib.spice`	A 16 541-byte table of contents. It has 51 `.lib` sections, one per corner, and each pulls in a tree of `.includes` (169 files for `tt`).	Always, in AD103: every deck loads its `tt` section.
`libs.ref/sky130_fd_pr/spice/`	675 files, each one device family and small enough to read, such as `sky130_fd_pr__nfet_01v8__tt.pm3.spice` and `sky130_fd_pr__res_high_po.model.spice`. These are the files the `.lib` sections include.	When you want to *read* a device rather than simulate it.
`libs.tech/combined/`	The **continuous** models: the same devices, recharacterised so that their parameters vary smoothly with size.	Not directly. This is the directory XSchem's own `xschemrc` points `SKYWATER_MODELS` at.

The last row marks a real fork in the PDK, which the foundry's own README explains:

> The original devices models were characterized at specific device width and length, and are considered "micro-binned"; that is, device models are accurate for devices with width and length equal to the devices that were characterized, and can diverge considerably for other device sizes. The "continuous" models have been recharacterized to ensure that parameters are continuous across device width and length.
>
> — `/foss/pdks/sky130A/libs.tech/combined/README`

AD103's decks load `libs.tech/ngspice/sky130.lib.spice`, whose `tt` section includes the binned models, so every number in this course comes from the binned models. Both kinds of model take `W` and `L` as plain numbers in microns.

Your device line becomes a subcircuit

```
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1
```

The leading `X` means "instance of a subcircuit". In the `tt` corner, this subcircuit comes from `libs.ref/sky130_fd_pr/spice/sky130_fd_pr__nfet_01v8__tt.pm3.spice`, and you can find its definition with:

```
grep -m1 -n -A3 '^\.subckt  sky130_fd_pr__nfet_01v8 ' \
                                              /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__nfet_01v8__tt.pm3.spice
```

```
35:.subckt  sky130_fd_pr__nfet_01v8 d g s b
36-+
37-.param  l = 1 w = 1 nf = 1.0 ad = 0 as = 0 pd = 0 ps = 0 nrd = 0 nrs = 0 sa = 0 sb
= 0 sd = 0 mult = 1
38-msky130_fd_pr__nfet_01v8 d g s b sky130_fd_pr__nfet_01v8__model l = {l} w = {w} nf
= {nf} ...
```

The subcircuit has four terminals in the order **d g s b** and thirteen parameters with defaults, and inside it is one real MOSFET whose model is called `sky130_fd_pr__nfet_01v8__model`. That trailing `__model` is the name the bins are attached to. The MOSFET inside the subcircuit is also why your `.op` printout is full of names like `@m.XM1.msky130_fd_pr__nfet_01v8[gm]`: the transistor you placed is `XM1`, and the MOSFET inside it is `msky130_fd_pr__nfet_01v8`.

Not every device is a subcircuit. `sky130_fd_pr__res_generic_m1` and the other metal and local-interconnect resistors are bare `.model` cards, so their instance lines start with `R` rather than `X`:

```
r7 g 0 sky130_fd_pr__res_generic_m1 w=1 l=1
```

With `X` on one of those, ngspice says `Error: unknown subckt:`, and with `M` on a transistor it says `warning, can't find model 'sky130_fd_pr__nfet_01v8'`. The first letter has to match how the PDK defines the device, so look at the PDK file instead of guessing.

What a bin is

There is no single `nfet_01v8` model. The tt corner has **180** of them, and each one is fitted to its own rectangle of the (W, L) plane:

```
grep -c '^\.model sky130_fd_pr__nfet_01v8__model\.' \
                                              /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__nfet_01v8__tt.pm3.spice
```

```
180
```

Each model states its rectangle on its first line. This is the one ngspice will choose for `L=0.15 W=1`:

```
.model sky130_fd_pr__nfet_01v8__model.71 nmos
+ lmin = 1.5e-07 lmax = 1.8e-07 wmin = 8.4e-07 wmax = 1.0e-6
```

If you sort the edges of all 180 rectangles, you get the ladder of sizes the foundry actually characterised:

axis	edges (µm)
L	0.15, 0.18, 0.25, 0.5, 1, 2, 4, 8, 20, 100
W	0.36, 0.39, 0.42, 0.52, 0.54, 0.55, 0.58, 0.60, 0.61, 0.64, 0.65, 0.74, 0.84, 1.0, 1.26, 1.68, 2.0, 3.0, 5.0, 7.0, 100

Ten L edges make nine bands and twenty-one W edges make twenty, and $9 \times 20 = 180$, which is exactly the bin count. So a SKY130 NMOS is defined only for **L from 0.15 µm to 100 µm and W from 0.36 µm to 100 µm**. If you ask for a size outside that range, there is no model for ngspice to run.

The other devices in the same corner are binned too. These counts are taken across all the files that the corner's `.lib tt` section includes:

device	bins
`sky130_fd_pr__nfet_01v8`	180
`sky130_fd_pr__pfet_01v8`	108
`sky130_fd_pr__pfet_01v8_hvt`	198

device	bins
sky130_fd_pr__nfet_01v8_lvt	88
sky130_fd_pr__pfet_01v8_lvt	48
sky130_fd_pr__nfet_g5v0d10v5	88
sky130_fd_pr__pfet_g5v0d10v5	88

The units, and why they look wrong

Look again at the first line of that bin:

```
+ lmin = 1.5e-07 lmax = 1.8e-07 wmin = 8.4e-07 wmax = 1.0e-6
```

Those limits are in metres, but you wrote `L=0.15 W=1`, so something has to convert one into the other before ngspice can compare them. The conversion comes from a line that every bin also carries:

```
+ binunit = 2.0
```

`binunit = 2` tells ngspice that the *instance's* `L` and `W` are micron numbers, so it scales them before choosing a bin. `L=0.15` becomes 1.5×10^{-7} m, which is exactly `lmin`.

Lab 01 ships a deck, `labs/lab-01-first-schematic/spice/bins.spice`, that lets you watch the conversion happen. It is the same transistor as `nmos_op.spice` with one deliberate difference: it has **no `.control` block at all**. Without one, batch ngspice dumps the whole operating point, including the model card it chose and the geometry it used, rather than only the values you asked it to print.

That dump runs to several hundred lines, so `make bins` pulls out the eleven that matter here:

```
cd labs/lab-01-first-schematic
make bins
```

```
      model xm1:sky130_fd_pr__nfe
       toxe             4.148e-09
       vth0              0.411231
       lmin               1.5e-07
       wmin               8.4e-07
      model xm1:sky130_fd_pr__nfe
          l               1.5e-07
          w                 1e-06
        vth              0.769267
       weff           9.56282e-07
       leff           1.26136e-07
```

(The `make bins` target is just four lines of `grep`. If you would rather run it yourself, the command is `ngspice -b spice/bins.spice 2>/dev/null | grep -E '^ +model xm1|^ +(toxe|vth0|lmin|wmin|l|w|vth|weff|leff) '`.)

The model name is cut short to `sky130_fd_pr__nfe` because ngspice's output column is only that wide, and the header appears twice because ngspice prints the model card first and the instance second. Everything above the second header describes the **bin**, and everything below it describes **your device**.

That block tells you four separate things:

- `l = 1.5e-07` and `w = 1e-06`: your plain `0.15` and `1` arrived as metres, so the conversion really did happen.
- `lmin = 1.5e-07` and `wmin = 8.4e-07`: this is bin 71, the one ngspice chose. Your device sits on its **lower** L edge (0.15 µm) and its **upper** W edge (1.0 µm), so if you nudge either size past that edge, a different model card with different fitted parameters takes over.
- `vth0 = 0.411231` is the model card's threshold parameter, and `vth = 0.769267` is the threshold this device actually has at this bias. The two numbers differ, and they are meant to; What moves the threshold explains why.

- `weff = 9.56282e-07` and `leff = 1.26136e-07`: **you drew 1 µm and got 0.956 µm, and you drew 0.15 µm and got 0.126 µm.** The gate is 16 % shorter than the mask because the source and drain diffuse sideways underneath it, and in general nothing comes out the size you drew.

The `W=1` versus `W=1u` trap

Everywhere else in SPICE, a bare number is in metres and you write `1u` for a micron. On a SKY130 device you write **`W=1`** instead, and `W=1u` fails. Lab 01 ships two decks in `labs/lab-01-first-schematic/spice/` that differ only in those suffixes:

```
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15 W=1 nf=1 m=1     ← nmos_op.spice
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.15u W=1u nf=1 m=1   ← nmos_op_wrong_units.spice
```

```
make wrong-units
```

```
Error on line 56 or its substitute:
    m.xm1.msky130_fd_pr__nfet_01v8  d  g  0  0  xm1:sky130_fd_pr__nfet_01v8__model  l=
1.500000000000000e-07     w=    1.000000000000000e-06     nf=    1.000000000000000e+00
ad= ...
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!
```

The exit status is `1`.

This trap is hard to spot because the error prints `l = 1.500000000000000e-07`, which is 150 nm, the number you wanted. It looks correct, so students stare at that line and conclude that the problem must be somewhere else.

That isn't the value ngspice ends up using, though, because it is printed *before* `binunit` scaling, and ngspice is about to multiply it by 10^{-6} again: `0.15u` = 1.5×10^{-7} $\longrightarrow$ 1.5×10^{-13} m = 0.00000015 µm, which is below `lmin` in all 180 bins. What the error message shows you is the value halfway through the conversion.

So if you get `could not find a valid modelname` on a line that looks fine, look for a `u`. Then compare the two `l=` values: the good deck's `.op` shows `l 1.5e-07` *after* scaling, while the failing deck's error shows `l= 1.5e-07` *before* it.

The same error, a different cause

```
XM1 d g 0 0 sky130_fd_pr__nfet_01v8 L=0.1 W=1
```

```
Error on line 56 or its substitute:
    m.xm1.msky130_fd_pr__nfet_01v8  d  g  0  0  xm1:sky130_fd_pr__nfet_01v8__model  l=
1.000000000000000e-01     w=    1.000000000000000e+00 ...
could not find a valid modelname
```

The message is identical, but there is no `u` anywhere on the line. This time the cause is that SKY130's shortest characterised gate is **0.15 µm** and you asked for 0.1 µm, so again no bin claims the device. The expanded line is what tells the two cases apart: `l= 1.0e-01` is a plain micron number that is simply too small, while `l= 1.5e-07` is a value in metres that should have been in microns.

So the message can mean two things, and the expanded `l=` tells you which:

the expanded `l=` shows	what happened
a number that looks like microns (`1.0e-01`, `5.0e+00`)	The size is outside the ladder; check it against the L and W edges above.

the expanded `l=` shows	what happened
a number that looks like metres (`1.5e-07`, `1.0e-06`)	You wrote a `u`, so delete it.

Every other device takes plain micron numbers too

The rule applies to more than transistors: no device in the SKY130 PDK takes a unit suffix on its geometry.

Device	Line	Parameters	Units
MOSFET	`XM1 d g s b sky130_fd_pr__nfet_01v8 W=1 L=0.15 nf=1 m=1`	`W, L, nf, m`	µm
Precision resistor	`XR1 r0 r1 b sky130_fd_pr__res_high_po W=1 L=10 mult=1`	`W, L, mult`	µm
Generic poly / diffusion resistor	`XR1 t1 t2 sky130_fd_pr__res_generic_po W=1 L=200`	`W, L, mult`	µm
Metal / li resistor	`r1 a b sky130_fd_pr__res_generic_m1 w=1 l=1`	`R, w, l, mult, m`	µm
MIM capacitor	`XC1 c0 c1 sky130_fd_pr__cap_mim_m3_1 W=10 L=10 MF=1 m=1`	`W, L, MF, m`	µm
Varactor	`XCV c0 c1 b sky130_fd_pr__cap_var_lvt W=10 L=10 VM=1`	three terminals	µm
Diode	`XD1 n p sky130_fd_pr__diode_pw2nd_05v5 area=1 perim=4`	`area, perim`	µm², µm

Two of those rows hide a trap of their own. Passing `mult=1` to `res_generic_po`, which has no such parameter, makes ngspice report `Too many parameters for subcircuit type "sky130_fd_pr__res_generic_po" (instance: xxr1)`, and giving the varactor two nodes instead of three makes it report `Too few parameters for subcircuit type "sky130_fd_pr__cap_var_lvt" (instance: xxcv)`. When in doubt, read the device's `.subckt` line, which lists both the terminals and the parameters and is only one `grep` away:

```
grep -m1 -A2 'subckt.*sky130_fd_pr__res_high_po ' \
                                        /foss/pdks/sky130A/libs.ref/sky130_fd_pr/
spice/sky130_fd_pr__res_high_po.model.spice
```

The diode's `area` and `perim` deserve a warning of their own, because getting them wrong produces no error at all. The deliberate failures in Lab 02: The diode I–V curve show what happens instead.

Corners

The `tt` is the second word on the `.lib` line, and it is one of 51 corners:

```
tt  sf  ff  ss  fs  ll  hh  hl  lh  sf_ll  sf_hh  …  tt_mm  …  mc
```

- The first five are the **transistor** corners: `tt` typical, `ff` fast-fast, `ss` slow-slow, and the mixed `sf` and `fs`.
- `ll`, `hh`, `hl` and `lh` are the **passive** corners, with the resistors and capacitors at their low or high values. AD102's corner decks use these.
- A `_mm` suffix adds mismatch, and `mc` is Monte Carlo. Both need extra setup, and AD103 does not use them.

Every corner is a `.lib` section of the same `sky130.lib.spice`, so you switch corners by changing that second word.

Three checks

1. **Before you believe a simulation**, check that the geometry landed where you meant it to: run a bare `.op` and read `l`, `w`, `weff` and `leff`.

2. **When you see `could not find a valid modelname`**, read the expanded `l=` in the error before you change anything. If it looks like metres, delete a `u`; if it looks like microns, your size is off the ladder.
3. **When a device behaves oddly across a size sweep**, remember that you may have crossed a bin edge. The parameters change discontinuously there. That is the price of "micro-binned" models, and it is the reason the continuous models exist.

Next: Sizing with W and L, which is about what to do with the two numbers now that you know what happens to them. For the passive side of the same PDK, see AD102′s SKY130 passive quick-reference.