ngspice deck recipes

This page gives the smallest deck that measures a passive device for each kind of measurement in AD102. It also shows what ngspice says when you get a deck wrong, and lists the warnings each lab prints that you can safely ignore.

AD102 asks five questions of silicon: how many ohms, how many farads, how many henries, how long the time constant is, and how much all of that moves. Each question has a deck shape you can reuse, and this page shows all five, with the exact output each one produced in `hpretl/iic-osic-tools:2026.08`, ngspice 47, PDK `sky130A`.

Every block below is an **excerpt** from a deck that ships in a lab package, and the deck's path is given beside it. None of them is a file for you to create; if you want to try something, run the shipped deck and edit that.

The five parts of a deck

Write the parts in this order. ngspice itself insists on only one position: whatever is on line 1 is the title. It doesn't check that `.end` comes last. On the pinned image a deck with no `.end` still runs, and a device line after `.end` still gets simulated, so putting `.end` last is a convention for whoever reads the deck rather than something ngspice enforces.

Part	Looks like	Notes
1. title	`* AD102 Lab 01 -- the sheet resistance deck`	**The first line is always a comment**, whatever you put in it, so put a real title there. If a device line lands on line 1, ngspice swallows it as the title.
2. models	`.lib /foss/pdks/.../sky130.lib.spice tt`	an absolute path, so the deck runs from any directory
3. devices and sources	`Xa a 0 0 sky130_fd_pr__res_high_po W=1 L=10`	see the catalogue
4. what to do	`.control` … `.endc`, or `.op` / `.ac` / `.tran` cards	
5. `.end`	`.end`	

Run the deck in batch mode, which runs the analysis, prints the results and exits:

```
ngspice -b spice/sheet.spice
```

Without `-b`, ngspice leaves you at an interactive prompt and the terminal looks as though it has hung; type `quit` to get your results.

1. Resistance: `.op` and Ohm's law

ngspice has no "measure the resistance" command, so you push a current of your own choosing through the device and divide the voltage that results by that current.

From `labs/lab-01-a-resistor-you-designed/spice/sheet.spice`:

```
Il1 0 l1 dc 1u
Xl1 l1 0 0 sky130_fd_pr__res_high_po W=1 L=1
```

```
.control
op
let r_L1 = v(l1)/1u
print r_L1
.endc
```

```
r_l1 = 6.954646e+02
```

That's 695.4646 Ω for one drawn square of a device whose model card says 317.3885 Ω/□. The gap between those two numbers is the subject of Ohms per square.

Three details of that pattern are worth knowing:

- **The 1 µA is chosen on purpose.** It's small enough that the resistor's own self-heating and voltage coefficient don't move the answer, and large enough to stay well clear of numerical noise. If you push 1 mA through 100 kΩ, you're asking a different question.
- **Use `let` before `print`.** `v(l1)/1u` is an *expression*, while `r_L1` is a vector. `print` accepts either, but `meas` doesn't accept an expression, so it's worth getting into the habit of naming the result with `let`, which costs nothing.
- **Vector names come back lower-case.** You wrote `r_L1`, and ngspice prints `r_l1`, which matters when you `grep` the log for it.

If you drive the device with a voltage source instead, the current comes out of the source and is negative, so the resistance is `1m/(-i(va))`, as in `labs/passives-decks/spice/corners.spice.in`:

```
let R_A = 1m/(-i(va))
```

2. Capacitance: one `.ac` point

A capacitor has no DC behaviour to measure, so instead you drive it at one frequency, measure the current, and undo $Z = 1/(j\omega C)$ to get the capacitance.

From `labs/lab-02-how-big-is-a-picofarad/spice/picofarad.spice`:

```
V4 a4 0 dc 0 ac 1
X4 a4 0 sky130_fd_pr__cap_mim_m3_1 W=10 L=10
```

```
.control
ac lin 1 1e6 1e6
let k = 1/(2*pi*1e6)
let c_s10 = abs(i(v4))*k
print c_s10
.endc
```

```
c_s10 = 2.065822e-13
```

That's 206.5822 fF for a 10 × 10 µm MIM plate.

- **`ac lin 1 1e6 1e6`** asks for a single point at 1 MHz, because you don't need a sweep to read one value.
- **`ac 1` on the source is required.** With only `dc 0`, you get zero everywhere, and ngspice gives no warning.

- **The 1 MHz is arbitrary, because it cancels out.** Any frequency works, as long as the same number appears in `k`. As a sanity check, try a decade either side: if the answer moves, your device isn't a pure capacitor (see Parasitic capacitance).

3. Inductance: `.ac` and complex arithmetic

An on-chip inductor is a resistor with some inductance attached, so you can't use the capacitor trick unchanged; you have to separate the imaginary part of the response from the real part.

From `labs/lab-02-how-big-is-a-picofarad/spice/henry.spice`:

```
.control
ac lin 1 1e8 1e8
let l_090_100M = real(imag(v(c))/(2*pi*1e8))
print l_090_100M
.endc
```

```
l_090_100M = 1.520796e-09
```

At 2.4 GHz the same deck works out the quality factor as well:

```
let q_090_2G4 = real(imag(v(c))/real(v(c)))
let r_090_2G4 = real(real(v(c)))
```

```
q_090_2g4 = 1.086413e+01
r_090_2g4 = 2.157013e+00
```

That's 1.5208 nH at 100 MHz and, at 2.4 GHz, Q = 10.86 and 2.157 Ω of series resistance you didn't ask for.

You need the outer `real(...)`, because as far as ngspice's type system is concerned, `imag(v(c))` is still a complex vector, one whose imaginary part happens to be zero. Wrapping it in `real()` gives you a number you can print without a warning.

4. Time constant: `.tran` and `meas`

From `labs/lab-03-a-time-constant-in-silicon/spice/rc.spice`:

```
Vb bi 0 PULSE(0 1.8 1n 10p 10p 200n 400n)
```

```
.tran 2p 80n

.control
run
meas tran t_ideal  WHEN v(bo)=1.137816 RISE=1
.endc
```

The threshold, 1.137816 V, is 0.632120×1.8, so the time at which `v(bo)` crosses it is τ by definition.

The `.tran` is a card outside `.control`, and `run` executes it. That looks like extra work until you want the same analysis at another temperature. `set temp` takes effect only on the next `run`, whereas a `tran` typed inside `.control` runs immediately and ignores any temperature you set afterwards. That's how `rc.spice` runs −40 °C and +125 °C

as well as 27 °C from this one card. If you restructure the deck and your cold and hot numbers come out identical to the 27 °C one, this is the reason.

That `meas` line has two traps in it, and together they add 1.005 ns to the answer:

- `meas ... when` returns an **absolute time** rather than an interval. The pulse starts at $t = 1$ ns, so you subtract 1 ns.
- The pulse has a **10 ps rise time**, and a first-order system behaves as though the step arrived at the ramp's midpoint, so you subtract another 5 ps.

The total offset is 1.005 ns, and `src/check_rc.py` subtracts exactly that. If you forget it, a 10 ns time constant reads 11.005 ns, which is 10 % high, and the error is *constant*: it stays 1.005 ns whatever the time constant is. That's the clue to look for, because an error that doesn't scale with the thing you're measuring comes from the measurement rather than from the device.

5. Variation: corners and Monte Carlo

Corners and Monte Carlo answer two different questions, using two different mechanisms, and confusing them is the most common mistake in this half of the course.

Corners: the whole wafer lot is high or low

You choose a corner with one extra word at the end of the `.lib` line, which picks a section of the model file:

```
.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt
```

Section	Means
`tt`	typical, the middle of the distribution
`ll`	low resistance, low capacitance
`hh`	high resistance, high capacitance

`labs/passives-decks/` builds one deck per corner from `spice/corners.spice.in`, and the three decks give:

	`ll`	`tt`	`hh`
R_A, drawn 1 × 10 µm	3106.637 Ω	3550.443 Ω	3994.248 Ω
R_B, drawn 1 × 40 µm	11433.66 Ω	13067.04 Ω	14700.42 Ω
MIM, 10 × 10 µm	179.7322 fF	206.5822 fF	233.8667 fF
RC corner frequency	282.935 MHz	215.410 MHz	169.052 MHz
$\boldsymbol{R_B/R_A}$	**3.680396**	**3.680396**	**3.680396**
divider tap	**0.3845828 V**	**0.3845828 V**	**0.3845828 V**

Compare the last two rows with the first four. The absolute values move by ±12.5 % and the corner frequency by +31 % and −22 %, but **the ratio doesn't change even in the seventh significant figure.** That table is Matching and Pelgrom's law in miniature, and it's the reason analog designers build ratios.

Mismatch: two devices on the same die differ

A corner moves every device together. Mismatch moves devices on the same die apart from each other, so it needs a different switch.

From `labs/lab-04-two-resistors-that-disagree/spice/mismatch.spice`:

```
.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.param mc_mm_switch=1
```

```
.control
setseed 1
op
* ... measure ...
mc_source
op
* ... measure again: a different imaginary die ...
.endc
```

- **`mc_mm_switch=1`** turns on the per-instance `AGAUSS` term in SkyWater's model cards. Without it, every instance of the same device is identical, so there is no mismatch to see.
- **`mc_source` re-evaluates the netlist**, drawing a fresh sample for every instance. If you loop it 200 times, you have 200 imaginary dies.
- **`setseed 1`** makes the sequence reproducible, which is why your Lab 04 numbers match the writeup to the digit.

Why the decks are slow

Every AD102 deck spends a while apparently doing nothing before it prints anything. What it's doing is reading model cards: `.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt` walks a tree of `.includes` that covers every device in SKY130, about 12 MB in all, and nearly all of it is MOSFET models that a passive deck never uses.

`.lib` line	wall clock, `spice/sheet.spice`
`sky130.lib.spice tt`	55–65 s on the pinned image; about 10 s on the machine this page was last checked on

Those are wall-clock seconds, so they change with how busy the machine is, whereas the resistances don't change at all.

Earlier workbench images also shipped pre-flattened files such as `sky130.lib.spice.tt.red`, which loaded about thirty times faster. The 2026.08 image doesn't include them, so every deck in this course, including `spice/bode.spice` in Lab 03, now loads `sky130.lib.spice`. If a deck or a tutorial you find elsewhere names a `.red` file, ngspice on this image stops with `Could not find library file`. If you're editing a lab's shipped deck, leave the `.lib` line alone, because the reference numbers were taken with it.

Warnings and delays in each lab

This section lists what each lab prints that isn't an error, and how long each one stays silent while it works. A line that starts with `Error` is a different matter: it stops the run, and Error messages explains what it means.

Lab 01

Every ngspice run loads `.lib /foss/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt`, which is 12 MB of model cards for every device in SKY130, mostly MOSFETs you aren't using, and it prints nothing while it

loads. That takes anywhere from about 10 seconds to just over a minute per run, depending on your machine (the comment at the top of the Makefile records 55–65 s). `make` runs the simulator twice, so allow up to about two minutes. The silence doesn't mean it has hung.

`results/sheet.log` holds 170 `unrecognized parameter` lines, in 34 blocks. `make` writes them to the log rather than the terminal, so from this deck your terminal shows only the `---` and `r_` lines. A `rhead_model` block has four of these lines and a `rbody_model` block has six. Here is a head block:

```
Warning: Model issue on line 4842 :
  .model xl1:rhead_model r sw_et=0 isnoisy=0 rsh=    3.458312000000000e+02 ...
unrecognized parameter (sw_et) - ignored
unrecognized parameter (isnoisy) - ignored
unrecognized parameter (p2) - ignored
unrecognized parameter (q2) - ignored
```

SkyWater writes one set of model cards to serve several simulators. `sw_et`, `isnoisy` and `p2`/`q2`/`p3`/`q3` are Spectre-flavoured keywords for self-heating, noise and voltage coefficients, which ngspice doesn't implement. ngspice says so for each one and carries on with the parameters it does understand, which are the ones the lab measures. There are thirty-four blocks for twenty-two resistors because most of these devices expand into more than one `.model` card.

`make resistor` in `passives-decks` (Lab 01, Part 6) prints fewer of these lines, because its deck has fewer resistors: `grep -c 'unrecognized parameter' results/r_head_and_body.log` prints `100`.

If a line starts with `Error`, the run has stopped, and your log will contain no `r_` lines at all. The checker, `src/check_res.py`, tells you to look for exactly that.

Lab 02

`pf`, `mine` and `tank` each load SKY130's model cards before printing anything. `henry` doesn't, because the inductor models live in three small files outside the corner library, so it finishes almost immediately. The Makefile budgets about three and a half minutes for the whole `make` on the pinned image, so a long silence is normal.

Lab 03

Each of the two runs in `make` is silent while ngspice loads 12 MB of SKY130 model cards.

`Doing analysis at TEMP = ... and TNOM = 27.000000` appears five times in `results/rc.log`, at 27 °C, −40 °C and +125 °C. That comes from Part 4, where the deck runs the same `.tran` card at three temperatures. `TNOM` stays at 27, and that's correct: it's the default temperature that model parameters are referred to, not the one you're simulating. (A model card can set its own; `res_high_po`'s says `tnom = 30`, which is the value Lab 01's Going further uses.)

`Warning: command 'plot' is not available during batch simulation, ignored!` comes from the schematic in the lab's XSchem part, which asks for a plot that batch mode can't draw. The `meas` line under the warning still gives you the answer.

Lab 04

Each of the two decks loads the SKY130 model cards (about 55 s on the pinned image) and then evaluates 200 operating points (about 23 s for the bigger of the two), printing nothing while it works. Most of that time goes on loading the models rather than on the Monte Carlo itself. On the machine this page was last checked on, where the models load in about 10 s, each deck took 14 to 16 s in all and a full `make` about 30 s, the low end of the lab's estimate.

`results/mismatch.log` holds 28,280 `unrecognized parameter (...) - ignored` lines, of which 5,656 are `(sw_et)`. They're the same harmless lines as in Lab 01. Each of the two hundred re-evaluations of the netlist repeats the whole block, which is why the log is 1.8 MB. `make` sends all of it to the log rather than to your terminal.

The loop counter must exist before the first `mc_source`. `mc_source` throws away the vectors from the previous analysis, and a counter created after an analysis has run is thrown away with them. If you get this wrong, you'll see the following, buried among the usual model warnings:

```
Warning from checkvalid: vector run is not available or has zero length.
Error: RHS "run + 1" invalid
```

After that the loop stops at one die, and every σ you compute comes out as zero, with no further message and an exit status of 0. That's why `make` prints the operating-point count first: you're looking for 201, and if you see 2, this is the cause. The shipped decks create the counter immediately after `setseed 1`, before anything has been simulated:

```
setseed 1
let run = 0          <- here
dowhile run < 200
  mc_source
  ...
```

`X` lines and `R` lines

Most SKY130 passives are **subcircuits**, so their lines start with `X` and take capital `W=` / `L=`:

```
XR1 a 0 0 sky130_fd_pr__res_high_po W=1 L=10 mult=1
```

A few (`res_generic_nw`, `res_generic_l1`, and all the metal resistors) are bare `.model` cards, so their lines start with `R` and take lower-case `w=` / `l=`:

```
r1 b 0 sky130_fd_pr__res_generic_m1 w=1 l=1
```

If you get it backwards, ngspice tells you so, with a different message for each direction, and both messages are in the table below. The catalogue lists which device takes which.

Units of W and L

In a SKY130 deck, `W`, `L`, `w` and `l` carry **no unit suffix**, so `W=1` means one micron. Every other number on the line takes SI suffixes as usual (`1k`, `0.7p`, `3n`), which is why this mistake is so easy to make.

On a resistor the mistake gives no error at all. Getting started goes through an example: the same device gives 3550.443 Ω when written `W=1 L=10` and 3193.812 Ω when written `W=1u L=10u`. That's an ordinary-looking answer 10 % away from the right one, and ngspice still exits with `0`.

If a resistance is close but not right, look for a stray `u` before you look anywhere else.

Error messages

ngspice says	You did
`Warning: r.xb.rbody: resistance too low or not given, set to 1 mOhm`	put a `u` on `W`/`L` of a resistor. **The run continues, and the answer is wrong.** This is the quiet version of the trap.
`could not find a valid modelname` … `Simulation interrupted due to error!`	put a `u` on `W`/`L` of a *transistor*, or asked for a size that no model bin covers. This is the loud version, and it's the better outcome, because you can't miss it.
`Error: unknown subckt: xr6 f 0 sky130_fd_pr__res_generic_l1 w=1 l=1`	started the line with `X` for a device that is a plain `.model`. Use `r6 f 0 …`.
`Too many parameters for subcircuit type "sky130_fd_pr__res_generic_po" (instance: xxr1)`	passed `mult=1` to a device whose subcircuit has no `mult`. Not every resistor takes the same parameters.
`Too few parameters for subcircuit type "sky130_fd_pr__cap_var_lvt" (instance: xxcv)`	gave the wrong number of *nodes*. The varactor has three terminals, not two.
`Error: no such vector as -i(vds).`	put an expression where `meas` wanted a vector. Make it a vector first with `let id = -i(Vds)`.
`Error: incomplete or empty netlist / no simulations run!`	nothing, on its own. This is the second message, and the real error is a few lines above it.
`Warning: command 'plot' is not available during batch simulation, ignored!`	asked a batch run for a picture. It's harmless, because the `meas` lines still ran. You'll see it when you netlist a schematic that has `plot` in its control block.
ten `unrecognized parameter (...) - ignored` lines per `res_high_po`, two of them `(sw_et)`	nothing wrong. Every deck with a `res_high_po` in it prints these; see Getting started.

Exit codes

Deck	exit
a correct run	`0`
`W=1u` on a transistor, which gives `could not find a valid modelname`	`1`
`W=1u` on a **resistor**, which comes out 10 % wrong without an error	`0`
a deck whose `meas` failed	`0`

The exit status in `$?` catches a deck that wouldn't build, and nothing else. That's why every AD102 `Makefile` reads the log and prints its own `PASS` or `FAIL`, and any script you write should do the same.

Related: The SKY130 passive catalogue lists which devices exist, which line type each takes, and what each one measures. AD103′s ngspice survival card covers the same tool from the transistor side.