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
```

$$RC = 10184.63 \times 206.5822 \times 10^{-15} = 2.10396\ \text{ns}$$

$$f_{-3\ \text{dB}} = \frac{1}{2\pi RC} = 75.6453\ \text{MHz}$$

`.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
`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.
`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.

ngspice says	You did
`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.