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
`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=
1.5000000000000000e-07    w=   1.0000000000000000e-06    nf=   1.0000000000000000e+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 = 1.5000000000000000e-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` = $1.5 \times 10^{-7} \rightarrow 1.5 \times 10^{-13}$ 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.
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`	`w`, `l`	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, `W`, `L`, `VM`	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.