UNIVERSITY OF CALIFORNIA, BERKELEY
Department of Electrical Engineering and Computer Sciences

EE251B Advanced Digital Circuits and Systems

Spring 2024, Prof. Borivoje Nikolic
Homework 2

Issued: Friday, Monday, Febuary 16th, 2024
Due: Friday, March 8th 11:59pm

---

1 Transistor IV Models (50 %)

(a) Use simulator of your choice (Spectre or HSpice) to simulate the VGS/IDS curve for the following device

Cell	`nfet_01v8`
Width	$.64\mu m$
Length	$.15\mu m$
Process Corner	tt

Ground the body of the device ($V_{BS} = 0$). The device should be in saturation ($V_{DS} > V_{GS} - V_{th}$).

Provide your simulated V_{GS}/I_{DS} curve. What is the device's V_{th}?

Hints:

- Set the drain of the device to V_{DD}(1.8v) and the source to a 0ohm resistor, with the other terminal of the resistor ground. You can measure the current through the resistor as your device's I_{DS}.
- Here is some starter code for a testbench that may be useful:

```
.lib '/home/ff/eecs251b/sky130/sky130_cds/sky130_release_0.0.1/models/sky130.lib.spice'
tt

.include '/home/ff/eecs251b/sky130/sky130_conv.spice'

xnfet drain gate source body nfet_01v8 w=0.64u l=0.15u
```

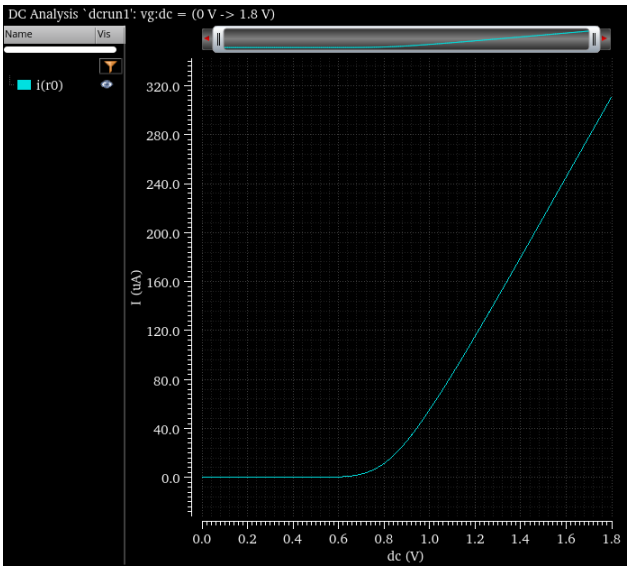

Figure 1: $I_{DS}V_{GS}$ curve for `nfet_01v8 w=0.64u l=0.15u`

```
* HW2 Q1 testbench

.lib '/home/ff/eecs251b/sky130/sky130_cds/sky130_release_0.0.1/models/sky130.lib.spice' tt
.include '/home/ff/eecs251b/sky130/sky130_conv.spice'

vvd vdd 0 1.8
vgnd vss 0 0

# Define a DC source for the gate voltage of the device
vg g 0 1.8

# Hook up the device: source to a 0ohm resistor (to measure current),
# drain to vdd, gate to gate voltage, and base to ground
xnfet vdd g s vss nfet_01v8 w=.64u l=.15u
r0 s vss 0

# Define a DC sweep
.dc vg 0 1.8 0.01

# Measure the drain current vs. the gate voltage
.probe v(g) i(r0)
.plot tran v(g) i(r0)
```

See 1 for the IV plot. The V_{Th} is approximately when the current begins to grow nonlinearly, $\approx .74V$. Any answer that lead to a reasonable fit was accepted.

(b) For the I_{DSat} model used in class ($I_{DSat} = \frac{W}{L}\frac{\mu_{eff}E_CL}{2}\frac{(V_{GS}-V_{Th})^2}{V_{GS}-V_{Th}+E_CL}$), find the values of E_CL and μ_{eff} that best fits your device's curve. Use the V_{th} value you found in part a).

Note that you may want to modify the $I_{Dsat}c$ equation to account for finite output resistance.

Hint: The `lsqcurvefit` function in MATLAB[1] may be useful.

We consider only currents above our threshold, as the I_{DSat} model only models current in the saturation (not sub-threshold) region.

```
% Load data
iv = csvread("q1a.csv",1);
v = iv(:,1);
i = iv(:,2);

% Pick vth and define parameters
Vth = .74; % v
W = 0.64e-6; % m
L = 0.15e-6; % m

% Define our idsat model and initial guess
IDSat_model = @(x, Vgs) (1/2 * (W/L) * ...
    (x(1)*x(2)*L) * (Vgs-Vth) .^ 2 ./ (Vgs-Vth+x(2)*L));
x0 = [485e-6, 1e6]; % Initial guesses for ueff and Ec, from an edaboard post

% Only consider data with v > vth
valid_indices = find(v > Vth);
v = v(valid_indices);
i_sat = i(valid_indices);

% Setup guess and bounds

% Fit curve
x = lsqcurvefit(IDSat_model, x0, v, i_sat);
ueff = x(1);
Ec = x(2);
i_sat_fitted = IDSat_model(x, v);

% Plot results
figure;
hold on;
plot(v,i_sat, 'LineWidth', 2, 'DisplayName', "Simulated");
plot(v, i_sat_fitted, 'LineWidth', 2, 'DisplayName', "Fitted");
legend;
title("V_{gs} I_{ds} Curve for nfet\_01v8 w=0.64u l=0.15u");
ylabel("I_{DS} (A)");
xlabel("V_{gs} (V)")
hold off;
```

The measured/fitted curves can be seen in 2. We have our fitted parameters $E_cL = .15V$ and $\mu_{eff} = .0010479\frac{m^2}{V \cdot s}$. Credit was given to fits and fitted parameters that followed the general shape of the curve. Note that credit was given for giving the units of μ_{eff} as mobility ($\frac{m^2}{V \cdot s}$) or mobility times capacitance ($\frac{A}{V^2}$) since its definition was not clear.

[1] You can find an example for how to export Spectre data to MATLAB at `https://courses.grainger.illinois.edu/ece483/sp2024/data/cadence/ECE483_MATLAB_export.pdf`

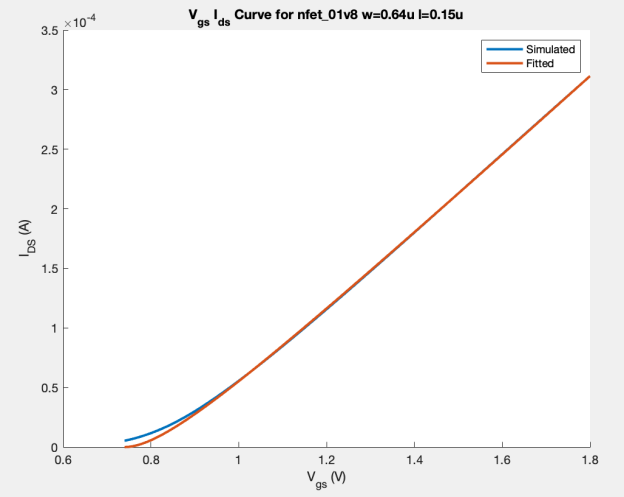

Figure 2: Fitted and Measured Curves

(c) Fit the simulated IV curve to the alpha-power-law model $I_D = K(V_{GS} - V_{Th})^{\alpha}$. Report K, V_{Th}, and α; Attach an image of the fitted curve superimposed on top of the simulated one.

i. is your fitted V_{th} different from the device model's? Why might this occur?

ii. We fit this curve for a device in saturation. when might this model be useful when the I_{DSat} one is not?

```
% same code as above
% Alpha power law model
Alpha_ID_model = @(x, Vgs) x(1) * (abs(Vgs - x(2))).^x(3);
x0 = [9e-4, Vth, 1.6]; % Had to just play around with the parameters

x = lsqcurvefit(Alpha_ID_model, x0, v, i_sat);

i_sat_fitted = Alpha_ID_model(x, v);

figure;
hold on;
plot(v,i_sat, 'LineWidth', 2, 'DisplayName', "Simulated");
plot(v, i_sat_fitted, 'LineWidth', 2, 'DisplayName', "Fitted, Alpha Power Law");
legend;
title("V_{gs} I_{ds} Curve for nfet\_01v8 w=0.64u l=0.15u");
ylabel("I_{DS} (A)");
xlabel("V_{gs} (V)")
hold off;
```

The fitted and simulated curves can be seen in 3. The fitted parameters are as follows: $K = 0.0003\frac{A}{V^{\alpha}}, V_{th} = 0.7445V, \alpha = 1.1451$

The fitted V_{th} may vary from the physical one as this is not a physical model, but an empirical one. This is a simple model that, when used carefully, can be derived into efficient and analytically tractable expressions for power, delay, and other quantities useful in circuit

design. It can also theoretically account for subthreshold behavior, while the I_{DSat} model cannot.

Unfortunately the fitted curve doesn't look amazing for this model. Credit was given for reasonable attempts.

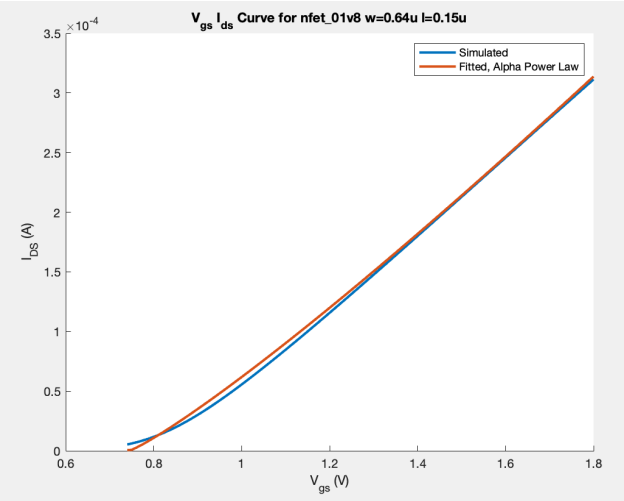

Figure 3: Alpha Power Law Fit

Hint: the `lsqcurvefit` function in MATLAB may also be useful here.

2 Standard Cells (50 %)

You have been tasked with providing useful delay modeling abstractions to a digital design team. Your responses should reference the following Sky130 LIB file: `sky130_fd_sc_hd__tt_025C_1v80.lib`. It can be found in `/home/ff/eecs251b/sky130/sky130A/libs.ref/sky130_fd_sc_hd/lib/`

(a) Using the lib file for `sky130_fd_sc_hd__inv_1`, plot (for both rising and falling outputs) the fanout (load) versus delay of using the tool of your choice.

Hint: the device has two 2d "delay" arrays, one for rise time (`cell_rise`) and one for fall time (`cell_fall`). One of the dimensions is the fanout (load) that device is driving.

The second index (columns) is the output load, while the first (rows) is the input net transition time. We choose the last row for both rise and fall timing, fixing the (80%-20%, as defined in the lib with the variables `slew_[lower/upper]_threshold_pct_[rise/fall]` input net transition time at 1.5ns, and sweep the output load.

Plotting these against the load (values of index_2), we have the blue line in figure 4.

(b) Simulate and plot the fanout (load) versus delay of `sky130_fd_sc_hd__inv_1` using SPICE simulations. Does it differ from the .lib defined behavior?

Hint: what is the second column of the .delay array? What must you change in your simulation setup to ensure you are matching the environment the .lib parameters were extracted from?

We define delay, as it is defined in the LIB file, as the time it takes the output to reach 80% of the final value from 20% of the final value.

Note that it appears that overall, slower input transition times lead to closer matching between simulation and lib files, but credit was given to students with reasonable simulation setups and simulation curves that matched the shape of the LIB curves.

```
* HW2 Q2 Testbench

.lib '/home/ff/eecs251b/sky130/sky130_cds/sky130_release_0.0.1/models/sky130.lib.spice' tt

# Units are broken in the spice netlist, define our own inverter
.subckt sky130_fd_sc_hd__inv_1 A VGND VNB VPB VPWR Y
X0 VGND A Y VNB sky130_fd_pr__nfet_01v8 w=0.65u l=0.15u
X1 VPWR A Y VPB sky130_fd_pr__pfet_01v8_hvt w=1u l=0.15u
.ends

# Define supplies and a piecewise-linear function
# that steps from initial to final value in
# (1/(80% - 20%)) * 1.5ns = 3ns
vvd vdd 0 1.8
vgnd vss 0 0
vpwl a 0 PWL(0 0 5n 0 8n 1.8 10n 1.8)
# Swap for
# vpwl a 0 PWL(0 1.8 5n 1.8 8n 0 10n 0)
# for rise time

# Add a load capacitance, sweep this manually since .step doesn't seem to work
```

```
c0 y vss 0.0005000000E-12


# Define our DUT
xnfet a vss vss vdd vdd y sky130_fd_sc_hd__inv_1

# Measure our rise/fall time
.probe v(a) v(y)
.meas tran {all trig v(y) val='1.8*0.8' fall=1 targ v(y) val='1.8*0.2' fall=1
# Swap for
# .meas tran trise trig v(y) val='1.8*0.2' rise=1 targ v(y) val='1.8*0.8' rise=1
# for rise time

.plot tran v(a) v(y)
.tran .01n 10n
```

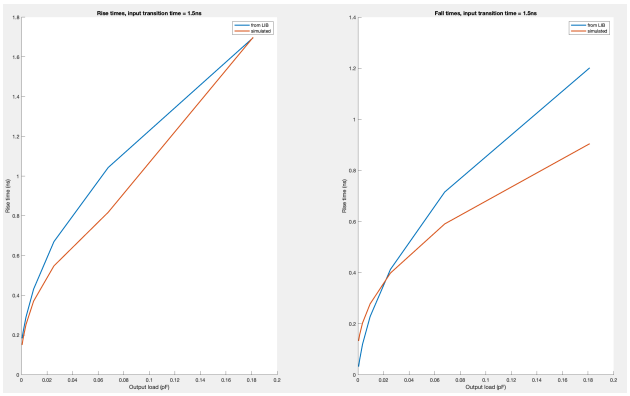

Figure 4: Timings From LIB Versus SPICE simulation (simulation using 80%-20% threshold)

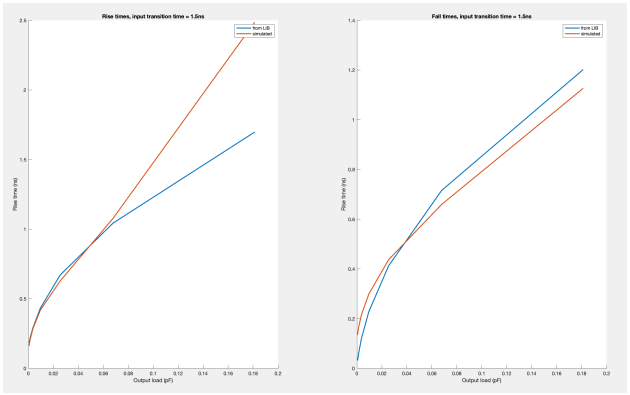

Figure 5: Timings From LIB Versus SPICE simulation (simulation using 90%-10% threshold)

Notice that the lines largely track, but seem to diverge as the load capacitance gets large for fall times. This is likely due to higher-order effects not included in the lib file (including mismatch in simulation/.lib characterization setup) that show themselves as we push the device further into its non-idealities.

Interestingly enough, (incorrectly) taking the delay timings to be between 90% and 10% of the final/initial values visually seems to achieve a better fit 5. This may be a coincidence due to the lib file being too conservative.

(c) Draw the schematic you could simulate to characterize the FO4 delay of an inverter `inv_custom` with size $\frac{W}{L}$. See Figure 6

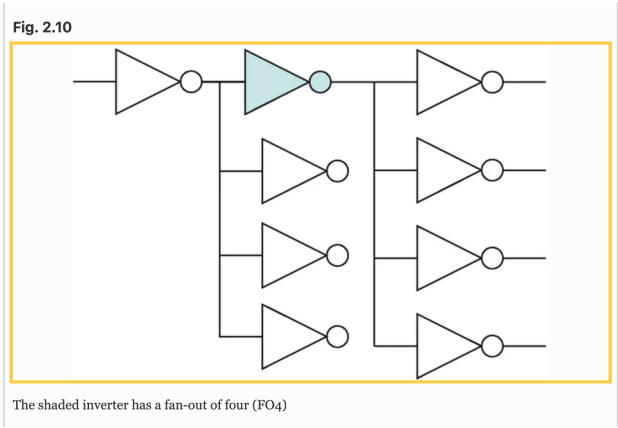

Figure 6: The shaded inverter has a fanout of 4. A more accurate setup might include an additional fan-out of 4 on each end to ensure loading/driving inverter input rise/fall times are accurate. soruce: `https://link.springer.com/chapter/10.1007/978-3-031-35605-6_2`

(d) Given the inverter `sky130_fd_sc_hd__inv_1` as your reference device, what is the logical effort of `sky130_fd_sc_hd__and2_0`? You may reference the `.lib` values.

Here we compare the input capacitances of the and2 and our reference inverter. From the lib file, the inverter has an input capacitance of .0023pF and the 2 inputs (A and B) of the AND gate have capacitances of .0016pF and .001636pF respectively. We divide input capacitances and by the inverter capacitance to get logical efforts of .6951 and .7111 for inputs A and B respectively. Note that this logical effort is less than 1, but that is because we are normalizing to a non-min-size-inverter.