Lab 2: g_m/I_D Design Methodology

EE140/240A – Linear Integrated Circuits

Fall 2025

1 Introduction

We used the square-law model in the previous lab to obtain the small signal parameters and bias points of transistors, and we observed that it fits *relatively* well with simulated results when long-channel transistors are used. The square-law model, however, falls short in at modeling the behavior of short-channel devices. Therefore it cannot be used for analysis and design of integrated circuits fabricated in advanced technology nodes. Many second-order effects are not captured by the square-law model in sub-micron devices, such as subthreshold conduction, drain-induced barrier lowering and velocity saturation. In the early 90s, a number of other more complicated models were proposed to capture such second-order effects. The EKV model [1] is perhaps the most accurate and widely used model. These models, however, require a multitude of parameters and lack the simplicity of the square-law model, severely limiting their application in practice. In 1995, Flandre *et al.* devised a powerful design approach bypassing any device model by relying on pre-computed/simulated transistor-width-independent parameters (g_m/I_D, I_D/W, g_m/c_{gg} and g_m/g_{ds}) known as g_m/I_D design methodology [2]. Transconductance efficiency, g_m/I_D, is a unique parameter with a finite range ($\sim$0–30 for MOS devices) that can be used as a knob to adjust the inversion level (subthreshold vs. strong inversion) and ultimately the speed or energy efficiency of the MOS transistors.

Over the course of this lab, you will learn the fundamentals of the g_m/I_D method by designing simple amplifiers. There are a number of design examples in this lab to make you comfortable with this method and prepare you for the design project. We will also practice scripting the design process. Redesigning with a different set of specifications would take only a matter of seconds. In Lab 0, you simulated the performance of a resistively-loaded common-source amplifier whose design was given to you. Just as a demonstration how powerful the g_m/I_D method is, towards the end of this lab, you will redesign the same amplifier by the knowledge you gain from this lab and simultaneously improve its gain and unity-gain bandwidth by respectively 5 dB and 10x!

The original g_m/I_D method has been widely used and further developed by many designers. The essence of all this development is how the look-up tables (LUTs) are generated and accessed. Here, we use a MATLAB script written by Boris Murmann [3] for LUT generation and readout. In the next section, you will learn what these LUTs are, how they are generated and accessed.

This lab spans over four lab sessions. The lab handout for each session is posted weekly. All the deliverables of this lab for write up are shown in Purple. There are a total of 2 check-offs and 4 write-ups to be submitted as part of the requirements of lab 2. Please be mindful of the due dates for these check-offs and the write up found on the course calendar.

2 Design Flow

The design flow of any circuit starts with a given set of specifications and constraints (*e.g.,* gain, bandwidth and output load). The designer first needs to identify the design variables (*e.g.,* g_m, W/L, etc.) and find their relationships with the given specifications and constraints (*e.g.,* $f_u = g_m/2\pi C_L$). The design space can usually be tightened by reducing the number of variables, that is, by elegant choice of topology (simplest topology with the fewest number of variables considered first) or by describing variables in terms of other variables. The last step in the design is to search within the design space to find the best set of variables that meet the design specs.

Previously, in Lab 1, we intentionally used long-channel devices so that we can leverage the square-law model in our hand calculations to describe the relationship between the variables. For example, for an intrinsic gain stage (IGS)

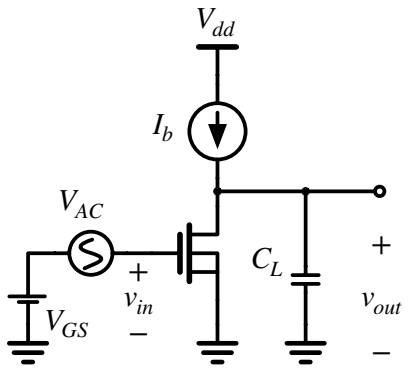

Figure 1: Intrinsic gain stage (IGS).

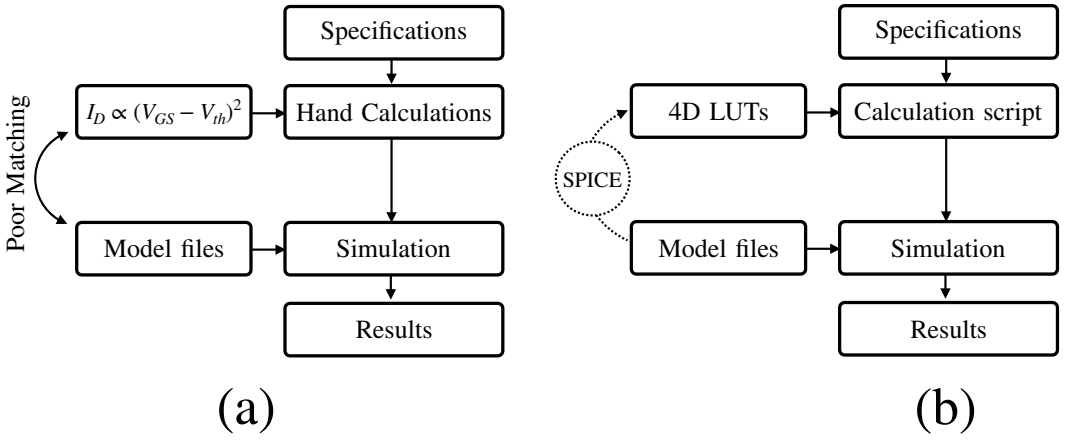

Figure 2: (a) Classic design flow using the square-low model. (b) LUT-based design flow.

shown in Figure 1 the relationship between variables is as follows:

$$g_m = \sqrt{2\mu C_{ox}\frac{W}{L}I_D} \tag{1}$$

$$r_o = \frac{1}{\lambda I_D} \tag{2}$$

$$A = g_m \cdot r_o = \frac{1}{\lambda}\sqrt{\frac{2\mu C_{ox}W/L}{I_D}} \tag{3}$$

$$f_u = \frac{g_m}{2\pi C_L} = \frac{\sqrt{2\mu C_{ox}W/LI_D}}{2\pi C_L} \tag{4}$$

Therefore, for given A_o, f_u and C_L, (3) and (4) can be simultaneously solved for W/L and I_D and the design is complete. Often due to approximations made in the square-law model, *e.g.,* channel-length modulation and body-effect, the design parameters need to be manually fine tuned during simulations to get the required A_o and f_u. This design flow is shown in Figure 2(a).

For short-channel devices, as mentioned earlier, the square-law model does not hold valid and the classic design flow needs to be revised to the one shown in Figure 2(b). The broken square-law model in this regime is replaced by pre-computed look-up tables. These look-up tables store all the information about our short-channel transistors (such as small-signal parameters g_m, c_{gs}, g_{ds}, etc.) across all the possible bias points (various V_{GS}, V_{DS} and V_{SB}) and for various device lengths. That was a lot to take in, so read that last sentence again! In short, any small-signal parameter of the device is stored as a function of four variables: V_{GS}, V_{DS}, V_{SB} and L. These look-up tables are generated for each device only once and by using a 4-dimensional parametric dc simulations whose setup is shown in Figure 3. As you have noticed, these look-up tables are generated using a single and arbitrary value for the width of the device W, say 1 μm. Then, you may wonder how we can use these LUTs as these LUTs store small-signal parameters of a device with say $W = 1$ μm. What if my transistor has a different W than the one these look-up tables were generated for? Well, the answer lies in the power of g_m/I_D design method that relies on W-independent variables such as g_m/I_D, I_D/W, g_m/c_{gg}, g_m/g_{ds}. It can be shown that these variables are (to the first order) independent of the width of the transistor (curious? Use the square-law model to demonstrate it). Therefore, it does not matter what W is used for the test device in Figure 3 when generating the LUTs as long as you use the LUTs for accessing W-independent variables.

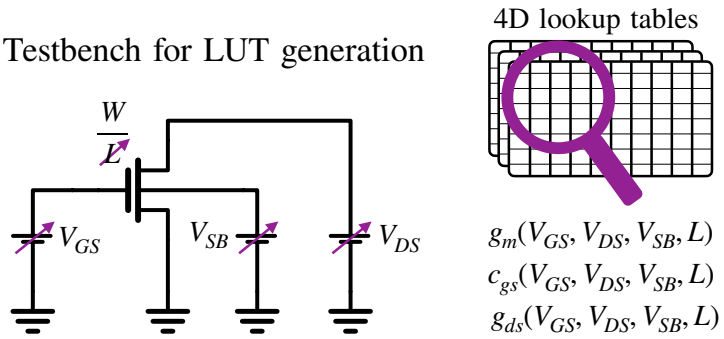

Figure 3: Testbench used for the simulation and generation of 4-D look-up tables

In the interest of time, you are provided with these look-up tables for nmos and pmos devices devices in the `gpdk045`, that is `nmos1v/pmos1v`. These lookup tables can be accessed using `python` or `MATLAB` and you are free to run whichever you are more comfortable. The lab computers do not support python and can be slow when running `MATLAB`. For that reason, we have provided the data at this link: `https://tinyurl.com/EE140Lab2Data`, so you may work with the data on your personal computer as well. If you choose to use `MATLAB` on the lab computers: Run the following commands in your terminal:

```
mkdir gpdk045_LUTs
cd gpdk045_LUTs
cp -r /home/ff/ee140/fa22/lab2/LUTs/. .
export PATH="$PATH:/share/b/bin"
matlab &
```

If you run these commands after sourcing `cadence_setup`, you need to add `MATLAB`'s path to your $PATH environment variable again by entering:

```
export PATH="$PATH:/share/b/bin"
matlab &
```

If you are choose to develop on your personal device using `MATLAB`, place `nch_1v.mat`, `pch_1v.mat`, `look_upVGS.m` and `look_up.m` in the same folder as your scripts. If you choose to develop with `python`, place `nch_1v.mat`, `pch_1v.mat`, and `look_up.py` in the same folder as your code.

Inside the `LUTs` directory you copied earlier, in addition to the LUTs, there are two important functions `look_up` and `look_upVGS` written by Boris Murmann. Open these functions' script and carefully read their description. Now, create a new `matlab` script and load the look-up tables of the `nmos1v` and `pmos1v` devices:

```
% MATLAB
nch = importdata('nch_1v.mat');
pch = importdata('pch_1v.mat');
```

```python
# python
import numpy as np
import matplotlib.pyplot as plt
from look_up import *
nch = importdata("nch_1v.mat")
pch = importdata("pch_1v.mat")
```

Notice: if you copy code from this PDF to MATLAB, the single quote symbol ' might be incompatible with MATLAB's editor, so you have to type the single quote symbols manually.

Explore the data stored in `nch` and `pch`. For example, you can find the length and width of the `nmos1v` test device used for generating these LUTs by entering `nch.L` or `nch['L']` and `nch.W` or `nch['W']`, respectively for `MATLAB` or `python`. These dimensions are stored in micrometers. The sweep range for V_{GS} and V_{DS} are respectively obtained by `nch.VGS` and `nch.VDS`. Moreover, you can obtain I_D–V_{GS} curves of the test device (remember with W of `nch.W`) for various L values by the following commands:

```matlab
% MATLAB
L = 0.1:0.2:1;    % Lengths of interest
semilogy(nch.VGS, look_up(nch, 'ID', 'VGS', nch.VGS, 'L', L));
```

```python
# python
L = np.arange(0.1, 1, 0.2)
plt.semilogy(nch['VGS'], look_up_basic(
    nch, 'ID', vgs=nch['VGS'], l=L).T)
plt.show()
```

Notice we did not specify V_{DS} in the above plots, instead the script specified a default V_{DS}. What is V_{DS} in the I_D–V_{GS} curves you just plotted? Can you estimate the threshold voltage for any of the test devices? The threshold voltage is the V_{GS} when the drain current exceeds $1\mu A$ for a $1\mu m$ width device.

Similarly, you can obtain I_D–V_{DS} of the test devices for various V_{GS} values using the following commands:

```
% MATLAB
VGS = 0.1:0.3:max(nch.VGS);
plot(nch.VDS, look_up(nch, 'ID', 'VGS', VGS, 'VDS', nch.VDS))
```

```
# python
VGS = np.arange(0.1, max(nch['VGS']), 0.3)
plt.plot(nch['VDS'], look_up_basic(nch, 'ID', vgs=VGS, vds=nch['
    VDS']).T)
```

What is the length of the device in the I_D–V_{DS} curves you just plotted? The I_D–V_{GS} and I_D–V_{DS} curves of the test `nmos1v` are shown in Figure 4 for your reference.

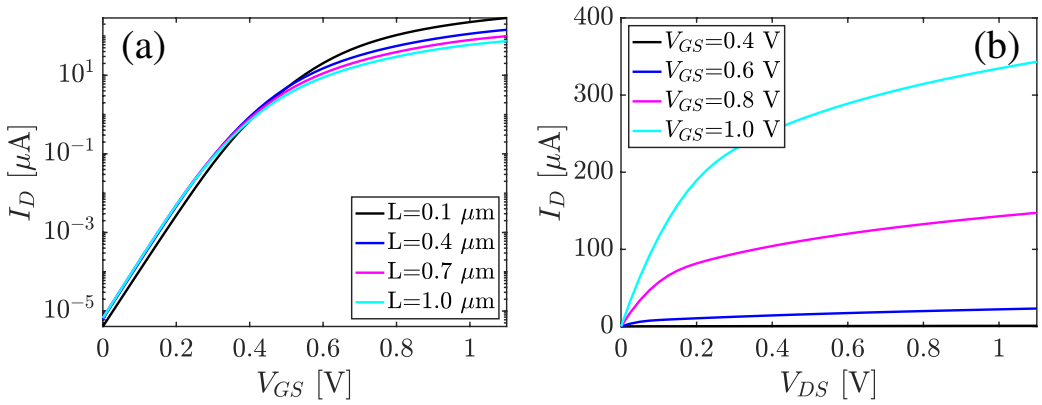

Figure 4: (a) I_D–V_{GS}, and (b) I_D–V_{DS} curves of the test `nmos1v` device plotted using the LUTs.

As mentioned earlier, almost never do we use LUTs to obtain absolute values of the small signal parameters. Instead, we obtain the ratios of small-signal parameters that are width-independent. You will see later on, how these width-independent ratios are effectively used during the design phase as variables. For now, let's plot three of these width-independent variables to uncover an inherent trade-off in MOS devices.

The transit frequency f_T of a transistor is the frequency at which the magnitude of the current gain of the device falls to unity. This current gain is caclulated when the drain of the transistor is at AC ground and the input has a small signal current input i_{in}. Draw the small signal model of an NMOS and prove that:

$$f_T \sim \frac{g_m}{2\pi c_{gg}} \tag{5}$$

$$c_{gg} = c_{gs} + c_{gd} + c_{gb} \tag{6}$$

The classic small signal model of the transistor you used to derive (7) falls short of correctly modeling the transistor's second order effects at frequencies greater than roughly $f_T/10$. Therefore, we limit the operating frequency range of our circuits to be $\leq f_T/10$. Keep in mind that f_T is a measure of a device's speed. That is, devices with a larger f_T allow us to operate at higher frequencies. Both g_m and c_{gg} scale linearly with W, and therefore f_T is a width-independent variable. So are the intrinsic gain $A_v = g_m/g_{ds}$, transconductance efficiency g_m/I_D, and the current density $J_D = I_D/W$ of the MOS devices. In the g_m/I_D design methodology, as the name suggests, g_m/I_D is used as a knob to set the speed or the intrinsic gain of the devices. Let's have a demonstration. Plot f_T and $A_v = g_m/g_{ds}$ of an intrinsic gain stage as a function of g_m/I_D by running the following commands:

```
% MATLAB
L = [0.05 0.2 0.5 1];
gm_ID = 5:0.1:30
fT = look_up(nch, 'GM_CGG', 'GM_ID', gm_ID, 'L', L)/2/pi;
Av = look_up(nch, 'GM_GDS', 'GM_ID', gm_ID, 'L', L);
semilogy(gm_ID, fT, '-')
hold on
yyaxis right
plot(gm_ID, Av, ':')
```

```
# Python
L = np.array([0.05, 0.2, 0.5, 1])
gm_ID = np.arange(4, 30, 0.1)
fT = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, l=L)/2/np.pi
Av = look_up_vs_gm_id(nch, 'GM_GDS', gm_ID, l=L)
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.semilogy(gm_ID, fT.T, '-')
ax2.plot(gm_ID, Av.T, ':')
plt.show()
```

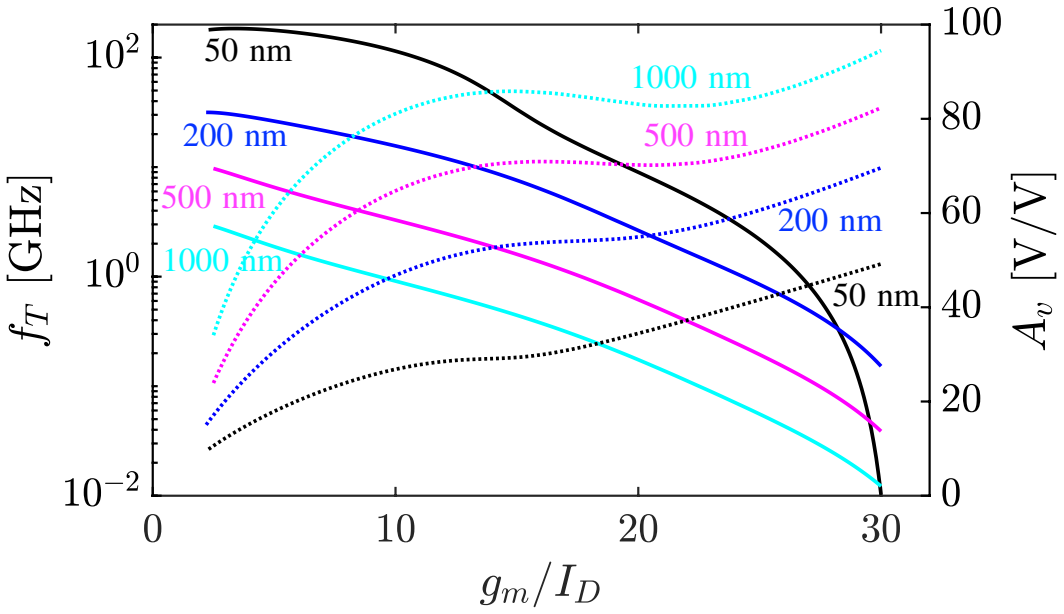

Figure 5: g_m/I_D used as a knob for adjusting the speed and the intrinsic gain of a device. Solid and dotted lines are respectively f_T and A_v.

Four main observations can be made in regards to Figure 5: 1) A_v is proportional to the length of the device. This is expected as increasing the length decreases the channel length modulation and ultimately g_{ds}. 2) f_T is inversely proportional to the length of the device. Again, this is an expected result because reducing the length simultaneously decreases c_{gg} and increases g_m. 3) A_v roughly monotonically increases with g_m/I_D, while at the same time 4) f_T degrades with increasing g_m/I_D. Therefore, g_m/I_D and L can be used as proxies to adjust the gain and speed, but remember that either the speed or the gain of the device can be maximized, not both by these proxies.

Now that we reviewed the basics, it is time to introduce g_m/I_D systematic design methodology [4]:

1. From design specifications and constraints, determine g_m
2. Pick L:
   (a) short-channel → high f_T, small area
   (b) long-channel → high A_v, large area
3. Pick g_m/I_D:
   (a) large g_m/I_D → low-power, high A_v
   (b) small g_m/I_D → high f_T, small area
4. determine I_D using steps 1. and 3.

5. determine W using I_D/W denormalization.

The next section has a series of five design exercises to walk you through the design steps introduced above. In each exercise, a list of specifications are provided along with a semi-complete Matlab script. You need to complete the Matlab script and verify your design in *Cadence Virtuoso*. For each exercise, provide a table of comparison, comparing your simulated results with the specifications listed for that exercise.

3 Lab Exercises

Exercise 1: IGS with known g_m/I_D and L

Consider the intrinsic gain stage shown in Figure 6. Calculate the intrinsic gain, f_T, W and required V_{GS} of the `nmos1v` device with the following assumptions: $g_m/I_D = 15$ S/A, $L = 100$ nm, $C_L = 1$ pF, $f_u = 1$ GHz and $V_{DS} = 0.55$ V. The feedback network shown in gray in Figure 6 is used to properly set the gate-source voltage of the nmos such that the output DC value is at $V_{DS} = 0.55$ V. The V_{DD} of the circuit should be 1.1V. For AC signals, you can ignore the feedback loop. You can use an ideal voltage-controlled voltage source, `vcvs`, from the `analogLib` library to implement the DC setting feedback amplifier. To avoid simulation convergence issues you may need to set `Maximum Output Voltage` and `Minimum Output Voltage` of the `vcvs` source to 1.1 V and 0 V, respectively. The `Voltage gain` of the `vcvs` should be ≥ 1000. The NMOS device used here is the `nmos1v` device from the `gpdk045` library. When creating a library for lab 2, please select "`Attach to an existing technology library`" and select `gpdk045` like in lab 0.

Answer:

In this exercise, g_m/I_D and L, V_{DS} and $V_{SB} = 0$ of the transistor are known. Therefore, we can directly use the `look_up` function to obtain f_T and the intrinsic gain of the device (remember the lookup tables are 4 dimensional). Note that both f_T and A_v are width independent and therefore we don't need to know the width of the device to be able to calculate these. The following Matlab script calculates the intrinsic gain of the stage. Complete the script to obtain f_T and W of the device.

```matlab
% MATLAB
gm_ID = 15;
L = 0.1;
fu = 1e9;
CL = 1e-12;
vds = 0.55;
fT = look_up(nch, 'GM_CGG', 'GM_ID', gm_ID, 'L', L)/2/pi;
%% Your code to obtain the intrinsic gain
%% Your code to obtain the current density JD = ID/W
%% Your code to calculate 'gm' from 'fu' and 'CL'
ID = gm/gm_ID;
W = ID/JD;
VGS = look_upVGS(nch, 'GM_ID', gm_ID, 'L', L);
```

```python
# Python
gm_ID = 15
L = 0.1
fu = 1e9
CL =1e-12
```

```
vds = 0.55
fT = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, l=L, vds=vds)/2/np.pi
# Your code to calculate intrinsic gain
# Your code to calculate current density JD = ID/W
# Your code to calculate gm from fu and CL
ID = gm / gm_ID
W = ID/JD
VGS = look_up_vgs_vs_gm_id(nch, gm_ID, l=L, vds=vds)
```

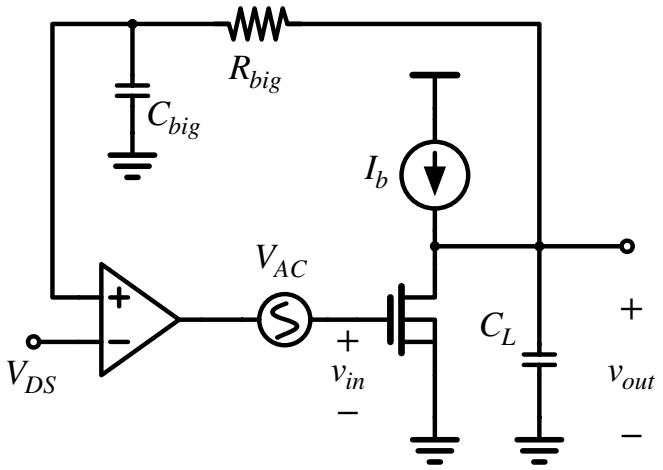

Figure 6: Intrinsic gain stage (IGS) used for simulation. The feedback loop is placed to set the gate-source voltage of the nmos for a given V_{DS}

The last `VGS = look_upVGS(...)` call in the above script returns the required V_{GS} of the device. In the simulation, we use this value as a sanity check. Now, simulate the performance of your IGS with the parameter values calculated above. You should run both DC and AC simulations to confirm the bias points of the amplifier as well as its AC response. Save a copy of the bode-plot of the transfer function of your amplifier and create a table listing its designed and simulated gain, unity gain bandwidth, g_m/I_D and V_{GS}.

Week 1: Introduction + Exercise 1 Deliverables (Report)

For week 1 (introduction + exercise 1) report, you must complete all the tasks in the following list and compile all the screenshots, codes, calculations, and everything else relevant into **one single document**. The items in the following list are from the purple texts in introduction and exercise 1.

Introduction

- (p. 5) Notice we did not specify V_{DS} in the above plots, instead, the script specified a default V_{DS}. What is V_{DS} in the I_D–V_{GS} curves you just plotted? Can you estimate the threshold voltage for any of the test devices? The threshold voltage is the V_{GS} when the drain current exceeds $1\mu A$ for a $1\mu m$ width device.
- (p. 6) What is the length of the device in the I_D–V_{DS} curves you just plotted?
- (p. 7) Draw the small signal model of an NMOS and prove that:

$$f_T \sim \frac{g_m}{2\pi c_{gg}} \tag{7}$$

$$c_{gg} = c_{gs} + c_{gd} + c_{gb} \tag{8}$$

- (p. 7) Plot f_T and $A_v = g_m/g_{ds}$ of an intrinsic gain stage as a function of g_m/I_D.

Exercise 1

- (p. 10) Complete the script to obtain f_T and W of the device.
- (p. 11) Now, simulate the performance of your IGS with the parameter values calculated above. You should run both DC and AC simulations to confirm the bias points of the amplifier as well as its AC response. Save a copy of the Bode plot of the transfer function of your amplifier and create a table listing its designed and simulated gain, unity gain bandwidth, g_m/I_D and V_{GS}.

Exercise 2: IGS with known g_m/I_D, Variable L

The previous exercise was fully constrained with known g_m/I_D and L. There was no design involved, and we merely analyzed the given IGS. In this exercise, we relax the constraints by setting L as a design variable. Consider the same intrinsic gain stage shown in Figure 6. Assuming a constant $g_m/I_D = 5$ V^{-1}, sweep over `nch.L`, find the achievable f_T and A_v by the IGS as well as the corresponding W for the `nmos1v` device. As in Exercise 1: $C_L = 1$ pF ,$V_{DD} = 1.1$ V and $V_{DS} = 0.55$ V, but $f_u = 500$ MHz.

Answer:

Complete the following script to complete this exercise:

```
% MATLAB
gm_ID = 5;
L = nch.L;
fu = 5e8;
CL = 1e-12;
VDS = 0.55;
%% Your code to calculate 'gm' from 'fu' and 'CL'
ID = gm/gm_ID;
JD = look_up(nch, 'ID_W', 'GM_ID', gm_ID, 'L', L, 'VDS', VDS);
W = ID./JD;
%% Your code to obtain fT = gm/2picgg
Av = look_up(nch, 'GM_GDS', 'GM_ID', gm_ID, 'L', L, 'VDS', VDS);
loglog(L, fT/1e9, '-k', L, Av, ':k');
```

```
# Python
gm_ID = 5
L = nch['L']
fu = 5e8
CL = 1e-12
vds = 0.55
# Your code to calculate gm from fu and CL
ID = gm/gm_ID
JD = look_up_vs_gm_id(nch, 'ID_W', gm_ID, l=L, vds=vds)
W = ID/JD
# Your code to obtain fT = gm/2picgg
Av = look_up_vs_gm_id(nch, 'GM_GDS', gm_ID, l=L, vds=vds)
plt.semilogy(L, fT/1e9, '-k')
plt.plot(L, Av, ':k')
plt.show()
```

Find the maximum value of L that satisfies $f_u \leq f_T/10$. As you can see, there are many solutions $\{L,\ W,\ I_D\}$ that satisfy $f_u = 500$ MHz constraint (with different intrinsic gains obviously). Choose two of these solutions and confirm with simulations both of them obtain $f_u = 500$ MHz. Save the simulated AC response of your designs and compare their performance in a table.

Exercise 3: IGS with variable g_m/I_D and L, Max A_v optimization

The previous exercise demonstrated that for a fixed g_m/I_D, f_u and C_L, the only way to increase the intrinsic gain of the stage is to increase the length of the device. But, because of the $f_T \geq 10 \times f_u$ constraint, L (and subsequently A_v) can only be increased up to a certain limit. Referring to the trade off shown in Figure 5, an attentive designer might think that there is a possibility to increase the gain beyond that obtained in the previous exercise by increasing the g_m/I_D of the transistor. This is in indeed true as shown in Figure 7. For small values of g_m/I_D according to Figure 5, the intrinsic gain of the device is at its minimum. Slightly increasing the g_m/I_D (to mid values of say ∼15) significantly increases the intrinsic gain. Increasing the g_m/I_D further pushes the device into the subthreshold region reducing its speed (f_T). But, because f_T of the device must be at least $10 \times f_u (= 5$ [GHz] in the previous exercise), maintaining this minimum required f_T requires reducing its length to small values (of ∼50 nm), which ultimately results in a heavy penalty in the intrinsic gain of the device. Therefore, there indeed seems to be an optimal value for g_m/I_D and the length of the device to obtain the maximum possible gain for the IGS. Complete the following Matlab script to find out what the optimal design is. As in previous exercise assume: $C_L = 1$ pF, $f_u = 500$ MHz, $V_{DD} = 1.1V$ and $V_{DS} = 0.55$ V.

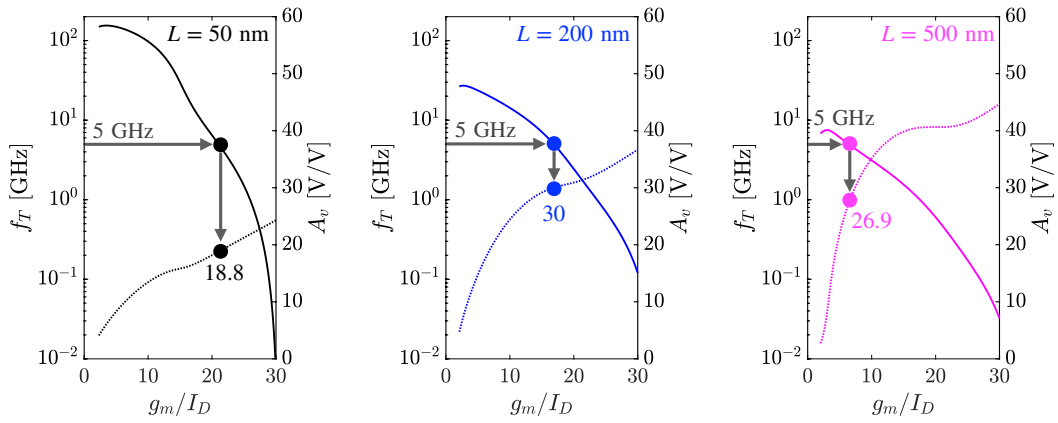

Figure 7: IGS gain optimization.

Answer:

To solve this problem, we first sweep over `nch.L` and calculate the required g_m/I_D value associated with each `nch.L` that obtains $f_T = 10 \times f_u$. Then for each $\{L, g_m/I_D\}$ pair, we compute the intrinsic gain. You do not have to use the provided script below, only the end results matter!

```
% MATLAB
L = nch.L;
fu = 5e8;
CL = 1e-12;
gm_ID_range = linspace(5, 30, 50);
gm_ID = [];
for i = 1:length(L)
  % Your code to obtain fT
  M = fT >= 10*fu;
  if(any(M))
    gm_ID(i) = gm_ID_range(max(find(M==1)));
  else
    gm_ID(i) = NaN;
  end
  %% Your code to obtain Av(i) = gm/gds
end
plot(L, Av, L, gm_ID);
%%%%%%%%%%%%%%%%%%%%%%
gm_ID_opt = X; % replace X based on the results obtained above
L_opt = Y;     % replace Y based on the results obtained above
%% your code to obtain JD = ID/W;
Cdd_W = look_up(nch, 'CDD_W', 'GM_ID', gm_ID_opt, 'L', L_opt);
Cdd(1) = 0;
for m = 1:10
  gm_opt = 2 * pi * fu*(CL+Cdd(m));
  %% Your code to obtain ID(m) %%
  W(m) = ID(m)./JD;
  Cdd(m+1) = W(m)*Cdd_W;
end
```

```
# python
L = nch['L']
fu = 5e8
CL = 1e-12
vds = 0.55
gm_id_range = np.linspace(5, 30, 50)
gm_id = []
av = []
for i, l in enumerate(L):
    # your code to obtain ft
    m = ft >= 10*fu
    if(any(m)):
        gm_id.append(gm_id_range[max(np.where(m==1)[0])])
    else:
        gm_id.append(float('nan'))
    # your code to obtain av[i] = gm/gds

fig, ax = plt.subplots(1)
ax.plot(l, av)
ax.plot(l, gm_id)
plt.show()

gm_id_opt = x # replace x based on the results above
l_opt = y     # replace y based on the results above
```

```
25 # your code to find jd = id/w
26 cdd_w = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt, l=l_opt)
27 cdd = 0
28 for i in range(1, 10):
29     gm_opt = 2 * np.pi * fu*(cl + cdd)
30     # your code to calculate id
31     w = id / jd
32     cdd = w * cdd_w
```

In previous exercises, we ignored c_{dd} (total drain capacitance) of the `nmos` while calculating g_m. In high-frequency applications, ignoring the total drain capacitance, however, decreases the design accuracy, and therefore should be included during the de-normalization phase (finding the widths/currents of the devices). Because transistor width determines c_{dd}, and c_{dd} adds to C_L which determines the g_m and ultimately width, c_{dd} is best estimated in an iterative fashion as shown in the last part of the script above. Initially W is calculated with c_{dd} assumed to be zero. Then, using the calculated W, c_{dd} is estimated and used to update the value of W. This process continues for a few cycles such that c_{dd} and W converge.

Simulate the AC response of your optimal design and compare its performance against the design in the previous exercise.

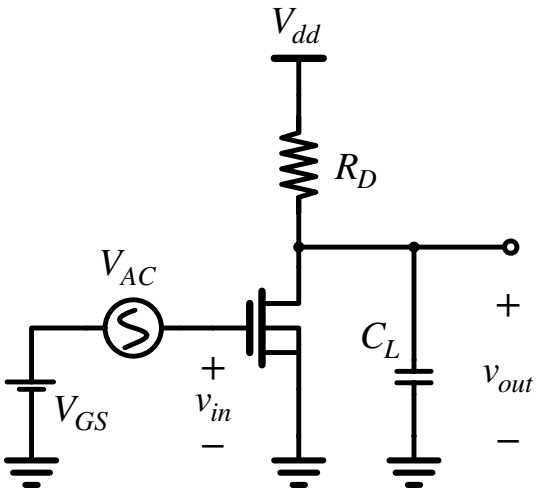

Figure 8: Common-source amplifier with resistive load.

Exercise 4: Common-source with resistive load

In Lab 0, we simulated the transient response of a resistively loaded common source amplifier shown in Figure 8. Simulate the AC response of the same amplifier ($R_D = 7.8$ kΩ, $W = 2$ μm, $L = 100$ nm, $V_{GS} = 630$ mV, $V_{DD} = 1.1$ V) with a load capacitance C_L of 1 pF and find the gain and unity-gain bandwidth f_u of that amplifier. For this part use an ideal resistor `res` from `analogLib`. Now, redesign the amplifier using the LUTs to obtain the maximum voltage gain while achieving $f_u = f_T/10 = 1$ GHz with $C_L = 1$ pF.

Answer:

The low-frequency gain of the resistively loaded common-source amplifier is given by

$$A_v = \frac{g_m}{g_{ds} + 1/R_D} = \frac{g_m/I}{g_{ds}/I + 1/(IR_D)} = \frac{g_m/I}{g_{ds}/I + 1/(V_{dd} - V_{DS})} \tag{9}$$

Therefore, the low-frequency gain of this amplifier can be describe in terms of width-independent parameters, g_m/I and g_{ds}/I and the drain-source voltage V_{DS}. Remember that to use the `look_up` function, we need 4 parameters usually V_{DS}, V_{SB}, L and one other width independent parameter which in this case can be $f_T = g_m/c_{gg} = 10$ GHz. Given $V_{SB} = 0$, L and V_{DS} can be used as design parameters to obtain the gain. Complete the following Matlab script to finish this task.

```
% MATLAB
fu = 1e9;
fT = fu*10;
CL = 1e-12;
```

```
VDD = 1.1;
VDS_range = 0.1:0.05:1.1;
L_range = 0.05:0.01:0.15;
for k = 1:length(VDS_range)
  gm_ID(:,k) = look_up(nch,'GM_ID','GM_CGG',2*pi*fT,'VDS',
   VDS_range(k),'L',L_range);
  % Your code to find gds_ID(:, k)''
  % Your code to find Av(:, k);
end
gain_opt = max(max(Av));
[L_idx, VDS_idx] = find(Av == gain_opt);
VDS_opt = VDS_range(VDS_idx);
L_opt = L_range(L_idx);
% Your code to find gm_ID at the optimal design point
% Your code to find current density (JD) at the optimal design
   point
Cdd_W = look_up(nch,'CDD_W','GM_ID',gm_ID_opt,'VDS',VDS_opt,'L',
   L_opt);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Cdd(1) = 0;
for i = 1:10
  % Your code to obtain gm(i)
  % Your code to obtain I(i)
  % Your code to obtain W(i)
  Cdd(i+1) = W(i)*Cdd_W;
end
% Your code to find RD
% Your code to find VGS
```

```
   # Python
fu = 1e9
fT = fu*10
CL = 1e-12
VDD = 1.1
VDS_range = np.arange(0.1, 1.1, 0.05)
L_range = np.arange(0.05, 0.15, 0.01)
gm_ID = np.zeros((len(L_range), len(VDS_range)))
gds_ID = np.zeros((len(L_range), len(VDS_range)))
Av = np.zeros((len(L_range), len(VDS_range)))
for k, vds in enumerate(VDS_range):
    gm_ID[:, k] = look_up_vs_gm_cgg(nch, 'GM_ID', 2*np.pi*fT, vds
   =vds, l=L_range)
    # Your code to calculate gds_ID
    # Your code to calculate Av

gain_opt_idx = np.argmax(Av.flatten())
l_idx, vds_idx = np.unravel_index(gain_opt_idx, Av.shape)
vds_opt = VDS_range[vds_idx]
l_opt = L_range[l_idx]
# Your code to calculate gm_ID_opt
# Your code to calculate JD_opt
Cdd_W = look_up_vs_gm_id(nch, 'CDD_W', gm_ID_opt, vds=vds_opt, l=
   l_opt)
cdd = 0
```

```
for i in range(10):
    # Your code to calculate gm
    # Your code to calculate I
    # Your code to calculate W
    cdd = W*Cdd_W

# Your code to calculate RD
# Your code to calculate VGS
```

Now, verify the performance of your amplifier in Cadence and report its simulated gain and unity gain bandwidth in a table comparing them against the design goals and those obtained by the circuit in Lab 0. Include W, L, V_{DS}, R_D and the power consumption of both the amplifiers in your table. Do you think the amplifier we used in Lab 0 was an optimal design? (To answer this question you may need to redesign the amplifier with the same unity gain bandwidth as that in Lab 0)

Week 2: Exercises 2-4 Deliverables (Report)

For week 2 (exercises 2-4) report, you must complete all the tasks in the following list and compile all the screenshots, codes, calculations, and everything else relevant into **one single document**. The items in the following list are from the purple texts in exercises 2-4.

Exercise 2

- (p. 13) Complete the script to find the achievable f_T and A_v by the IGS as well as the corresponding W for the `nmos1v` device. Include the script and plot in your report.
- (p. 13) Choose two solutions that satisfy $f_u = 500$ MHz and simulate them in Cadence Virtuoso, comparing their performance (A_v, f_u, I_D) in a table.

Exercise 3

- (p. 16) Simulate the AC response of your optimal design and compare its performance against the design in the previous exercise.

Exercise 4

- (p. 17) Simulate the AC response of the amplifier from Lab 0.
- (p. 17) Complete the script to perform the design of a resistively-loaded CS with a $f_u = 1$ GHz. Include the script in your report.
- (p. 19) Simulate the performance of the designed amplifier in Cadence and report its simulated gain and unity gain bandwidth against the amplifier from Lab 0 in a comparison table. Include W, L, V_{DS}, R_D, and the power consumption of both your amplifiers.
- (p. 19) Do you think the amplifier we used in Lab 0 was an optimal design? Justify, potentially with a redesign of the amplifier with the same f_u specification as the design from lab 0.

References

[1] Enz, Christian C., François Krummenacher, and Eric A. Vittoz. "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications." Analog integrated circuits and signal processing 8, no. 1 (1995): 83–114.

[2] Silveira, Flandre, Denis Flandre, and Paul GA Jespers. "A g_m/I_D based methodology for the design of CMOS analog circuits and its application to the synthesis of a silicon-on-insulator micropower OTA." *IEEE Journal of Solid-State Circuits* 31, no. 9 (1996): 1314–1319.

[3] online: `https://web.stanford.edu/~murmann/gmid`

[4] Jespers, Paul GA, and Boris Murmann. Systematic Design of Analog CMOS Circuits. Cambridge University Press, 2017.