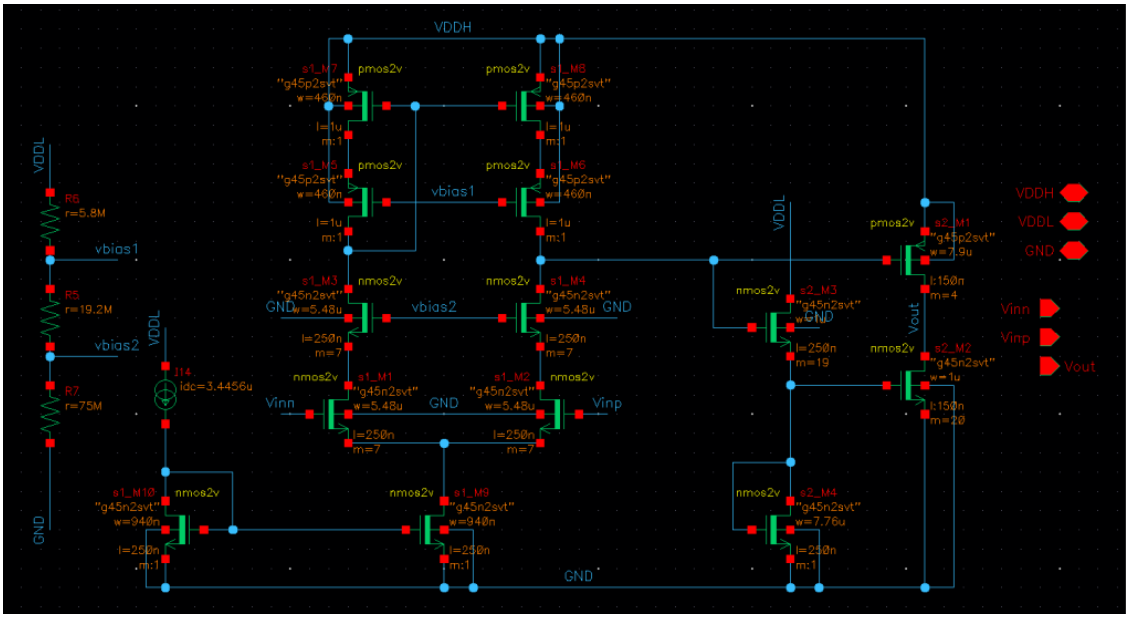
EE140 Project Report

Yu (Jaray) Li
SID: 3039817966

May 1, 2024

Overview

Cadence Schematic



The first stage is a telescope NMOS amplifier. The voltage bias point is set to be vbias1= 1.13V and vbias2 = 0.9V. The second stage is a class AB amplifier with the input to the PMOS. A NMOS source follower is applied to set the vbias of the NMOS and keep the amplified value of stage 1. The first stage contributes to most of the gain of the amplifier. The second stage is designed to have a good slew rate.

Comparison Table

This table shows the performance comparison between the design and the spec.

Performance	**Design Specifications**	**Design Values**
Settling Time	180.22ns	173ns
Power Consumption	$\leq$ 1.1mW	482.41μW
Maximum total capacitance	2pF	50fF
Maximum total resistance	100M Ohm	100M Ohm
CMRR at DC	$\geq$ 65dB	70.5221 dB
PSRR at DC	$\geq$ 50dB	59.78 dB
Phase Margin	$\geq 45°$	53.5°
FOM	$\frac{10^{-9}}{T_{settle}*P_{total}}$	11.98

Design

Consideration

- High Gain: meet static error and allow a greater proportion of 0.2% to go to dynamic, transient error.
- Slew Rate: The current drives a small capacitance (compensation), and the stage has a somewhat small output swing (the second stage applies gain to the signal).
- The standard differential amplifier doesn't provide enough gain; need to cascode and telescopic cascode is a valid option.

Settling time and required gain

My design is based on the idea to set the static and dynamic error each set to be 0.1%. The equation below shows the required DC gain to get a 0.1% static error.

$$\varepsilon_s = \frac{1}{1+\beta A_{DC}}, \quad A_{DC} = \frac{1}{\beta}\left(\frac{1}{\varepsilon_s} - 1\right) = \frac{1}{\frac{1}{3}}\left(\frac{1}{0.1\%} - 1\right) = 2997.$$

The equation below shows the required closed loop gain dominant pole frequency.

$$\varepsilon_d = e^{-\frac{T_{settling}}{\tau_{3dB}}}, \quad \tau_{3dB} = \frac{T_{settling}}{\ln \varepsilon_d} = \frac{180 \text{ ns}}{\ln 0.001} = 26.05 \text{ ns}, \quad f_{3dB} = 38.39 \text{ MHz}$$

2-stage gm design

Due to the presence of the compensation capacitor, the other poles are separated from the main pole and are much larger than the unity-gain bandwidth. Therefore, we can consider the system to be a single-pole system.
For a single-pole system, the formula for loop gain is:

$$\beta H = \frac{\beta A}{1 + s w_p}$$

The unity gain bandwidth of the loop gain is: $\beta A w_p$

And the closed-loop gain of the single-pole system is as follows:

$$\frac{H}{1+\beta H} = \frac{\frac{A}{1+sw_p}}{1+\frac{\beta A}{1+sw_p}} = \frac{1}{1+\beta A} * \frac{A}{1+\frac{s}{w_p(1+\beta A)}}$$

So in the preliminary analysis, we can use a simplified form. The main pole of the closed-loop gain is located at $w_p(1+\beta A) \approx \beta A w_p$

From the formula we can see that the unity gain bandwidth of the loop gain is equal to the main pole of the closed-loop gain: $f_u = f_{3dB} = 38.39$ MHz ,which is also the basis for our subsequent analysis.

For a 45 degree of phase margin, the non-dominant pole $w_{p2} = w_u = 2 \times p_i \times f_u$. From the lecture, the dominant pole is located at C_c, and the non-dominant pole is located at C_L. We can calculate g_{m2}:

$$g_{m2} = w_{p2} \times C_L$$

For g_{m1}, we know that the DC gain is $\beta A_1 A_2$, and the dominant pole is $w_{p1} = \frac{1}{r_{o1} C_C A_2}$. Then we can get:

$$w_u = \beta A_1 A_2 w_{p1} = \frac{\beta A_1 A_2}{r_{o1} C_C A_2} = \frac{\beta A_1}{r_{o1} C_C} = \frac{\beta g_{m1} r_{o1}}{r_{o1} C_C} = \frac{\beta g_{m1}}{C_C}$$

$$g_{m1} = \frac{w_u C_C}{\beta}$$

Transistor sizing design and parameters

From (2) we can get wu. Then we can apply Cc to the circuit. For stage 1, I followed the gm/id script to sweep my gm/id and choose a reasonable length so that the ft of the transistor can meet the fu specification and the gain is maximized. This operation is consistent with the ideas in previous exercises EX5, EX6, etc. In addition, considering the phase margin, I also swept Cc to get a reasonable phase margin. Once I selected gm/id, I then used the script to find CDD/W and JD. By iteratively calculating gm, I can calculate the gain.

For stage 2, I swept the gm/id of the PMOS so that the input voltage bias was the same as the output voltage of stage 1. For the NMOS, I chose a smaller gm/id to reduce power. The parameters for the PMOS and NMOS were then used to size the source follower. The reason for using a source follower design here is from the idea of AB amplifer:

- It's effective DC level-shifter.
- Simple to bias and integrate into a class A common-source approach. Can bias with generally low current and use relative widths to set VGS of NMOS device.

- Overcome slew-rate limit that makes class A very power-hungry and take advantage of increased $g_{m,\text{eff}}$.

Transistor and Bias Summary

In the table below, I have listed the bias and parameters of the MOS in the circuit, where S1 represents the MOS in stage 1 and S2 represents the MOS in stage 2. The figure shows the parameters of I_D, V_{GS}, g_m, g_{ds} corresponding to the MOS.

MOS Number	I_D	V_{GS}	g_m	g_{ds}
$S1\_M1$	1.2351μ	377.02m	33.672μ	1.9146μ
$S1\_M2$	1.2352μ	377.02m	33.675μ	1.9145μ
$S1\_M3$	1.2351μ	395.20m	34.441μ	625.26n
$S1\_M4$	1.2352μ	395.13m	34.479μ	621.79n
$S1\_M5$	1.2350μ	587.14m	14.611μ	329.80n
$S1\_M6$	1.2349μ	587.21m	14.603μ	332.89n
$S1\_M7$	1.2351μ	578.33m	12.635μ	2.2231μ
$S1\_M8$	1.2349μ	578.33m	12.632μ	2.2256μ
$S1\_M9$	2.4703μ	545.56m	39.857μ	3.5274μ
$S1\_M10$	3.4456μ	545.56m	52.831μ	1.9606μ
$S2\_M1$	221.21μ	573.97m	3.3865m	84.092μ
$S2\_M2$	228.63μ	610.17m	2.8979m	84.606μ
$S2\_M3$	57.663μ	615.85m	932.41μ	33.034μ
$S2\_M4$	57.663μ	610.17m	728.40μ	25.284μ

Discussion

Design flow and implementation

In the first stage, I will try to design a large gain amplifier that provides most of the gain required to meet the static error. At the same time, it also needs to have better performance than the stage1 structure in exercise6, because I found its performance very poor when I reproduced ex6. Therefore, I chose the telescope amplifier taught in the course as my first stage. Then for the second stage, I consulted some materials, and after comprehensive consideration, I designed a class AB common source as my second stage to obtain a better slew rate. The gain of the second stage does not have to be very large. However, we should apply high-current, short-length devices to obtain a better slew rate.

Optimization of stage 1 amplifier gain

We know the g_{m1} value from β,C_c, and f_u. Then a Python script similar to the previous exercise can help scan g_m/I_D to try to find the best A_v, and at the same time filter out g_m/I_D that does not pass f_u. By formula derivation, we can know the gain of the telescopic amplifier as follows:

$$
\begin{aligned}
A_V &\approx g_{m2}\left(g_{m6} r_{o6} r_{o8} \,/\!/\, g_{m4} r_{o2} r_{o4}\right) \\
&= g_{m2} \frac{g_{m6} r_{o6} r_{o8} g_{m4} r_{o2} r_{o4}}{g_{m6} r_{o6} r_{o8} + g_{m4} r_{o2} r_{o4}} \\
&= \frac{\left(g_{m2}/I_D \times g_{m6}/I_D \times g_{m4}/I_D\right) \Big/ \left(g_{ds6}/I_D \times g_{ds8}/I_D \times g_{ds2}/I_D \times g_{ds4}/I_D\right)}{\left(g_{m6}/I_D\right) \Big/ \left(g_{ds6}/I_D\right)\left(g_{ds8}/I_D\right) + \left(g_{m4}/I_D\right) \Big/ \left(g_{ds2}/I_D\right)\left(g_{ds4}/I_D\right)}
\end{aligned}
$$

The highest gain obtained in the Python code is 1760 (V/V). The simulation gain is 460 (V/V).

Sizing the Compensate Capacitor

The increase in C_c will provide another discharge path for $S1\_V_{out}$. This change will cause the tail current to increase, thereby increasing the power consumption. In addition, the position of the non-main pole will also have a certain impact on the phase margin. From the formula, we know that the non-main pole is located at $w_{p2} = \frac{g_{m2}}{C_L}$, and the increase in C_c will increase C_L, thereby reducing the position of w_{p2}. This will make the phase margin smaller because the second pole is pulled down below the unity gain bandwidth. According to my experience and some attempts, if I use 1pF C_c, I will not be able to meet the corresponding specifications. However, applying 50fF can easily meet the specifications, so I choose to use 50fF here.

Sizing the gm/id & W/L of stage 2

During the design process, I first designed the first stage amplifier and biased the output voltage to 1.21V. Then for the second stage amplifier, what we need to do is to supplement and improve on the basis of the first stage. The design idea for the second stage is to make the slope large enough and ensure that the input voltage is biased to 1.21V. The gain of the second stage AB class common source amplifier is as follows:

$$A_V \approx (g_{m1} + g_{m2})(r_{o1} /\!/ r_{o2})$$
$$= (g_{m1} + g_{m2}) \frac{\frac{1}{g_{ds1}} \cdot \frac{1}{g_{ds2}}}{\frac{1}{g_{ds1}} + \frac{1}{g_{ds2}}}$$
$$= \frac{g_m/I_D + g_{m2}/I_D}{g_{ds1}/I_D + g_{ds2}/I_D}$$

Based on this, since we need a 45-degree phase margin, we can set w_{p2} to unity gain bandwidth and provide C_L. From the equation above, we can get the overall g_{m2}:

$$G_{M2} = w_p \times C_L$$

$$G_{M2} = g_{m_n} + g_{m_p}$$

Using g_{m_p}/I_D to control the V_{GS} of the PMOS, we need V_{GS} to be 0.59V so that V_{in} is the same as the first stage output. From this, we can sweep g_{m_p} and see if the output matches. g_{m_n}/I_D controls the current of the second stage amplifier, and we need a large current to increase the slope. I swept g_{m_n}/I_D and saw if the slope was within spec.

Another design parameter that needs to be supplemented is the W/L value of the MOS. The larger the length, the greater the parasitic capacitance, which will reduce the charging speed. For this parameter, I used the minimum length to minimize the parasitic capacitance.

Design Issues and Tradeoffs:

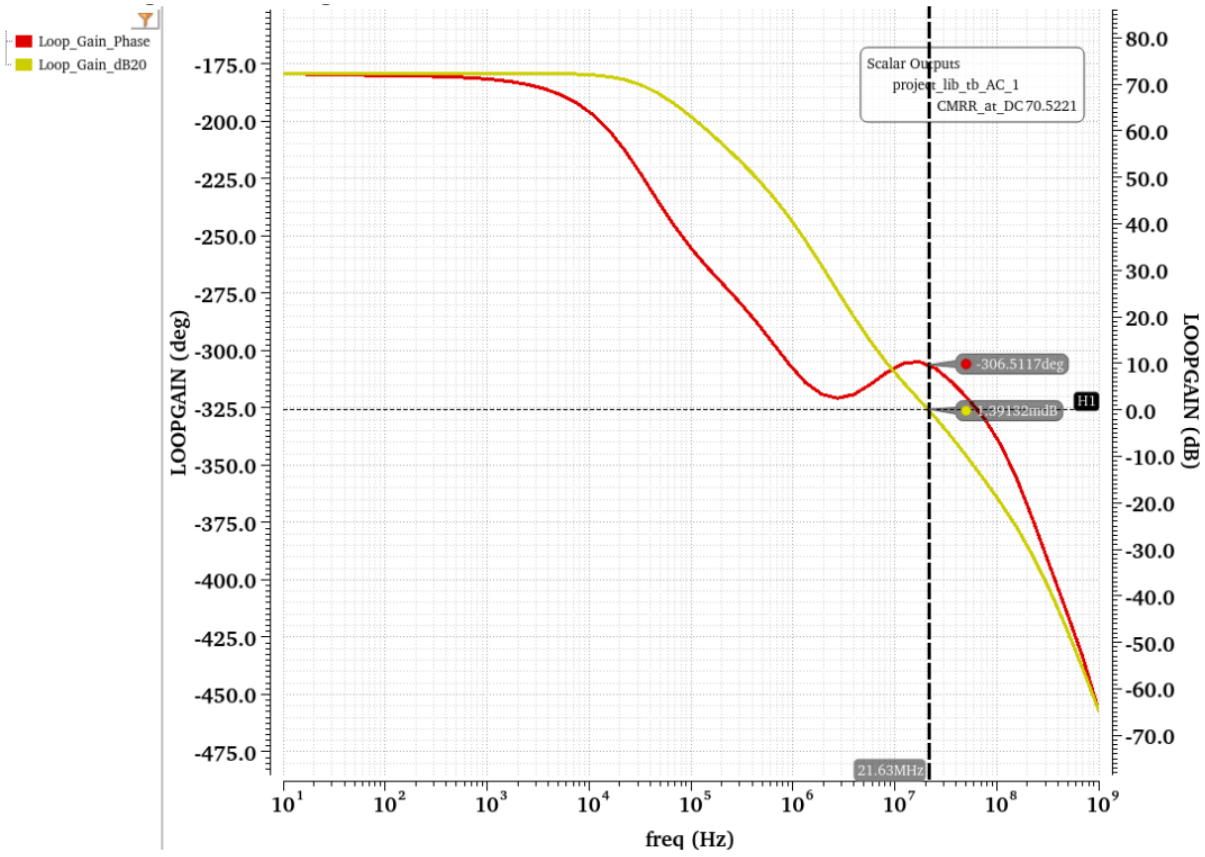
Design Issues:
(a) The output voltage of the first stage and the input voltage of the second stage must match.
(b) The location of the second pole must be greater than unity frequency.
(c) The gain of the loop gain must be large enough to ensure static error.
(d) The 3dB bandwidth of the loop gain must be large enough to keep the dynamic error within specification.

Tradeoffs:
(a) The slope of the second stage is a tradeoff with power. A larger slope requires more power.
(b) The size of the compensation capacitor is a tradeoff with the tail current of the first stage. A larger compensation capacitor requires a larger tail current.

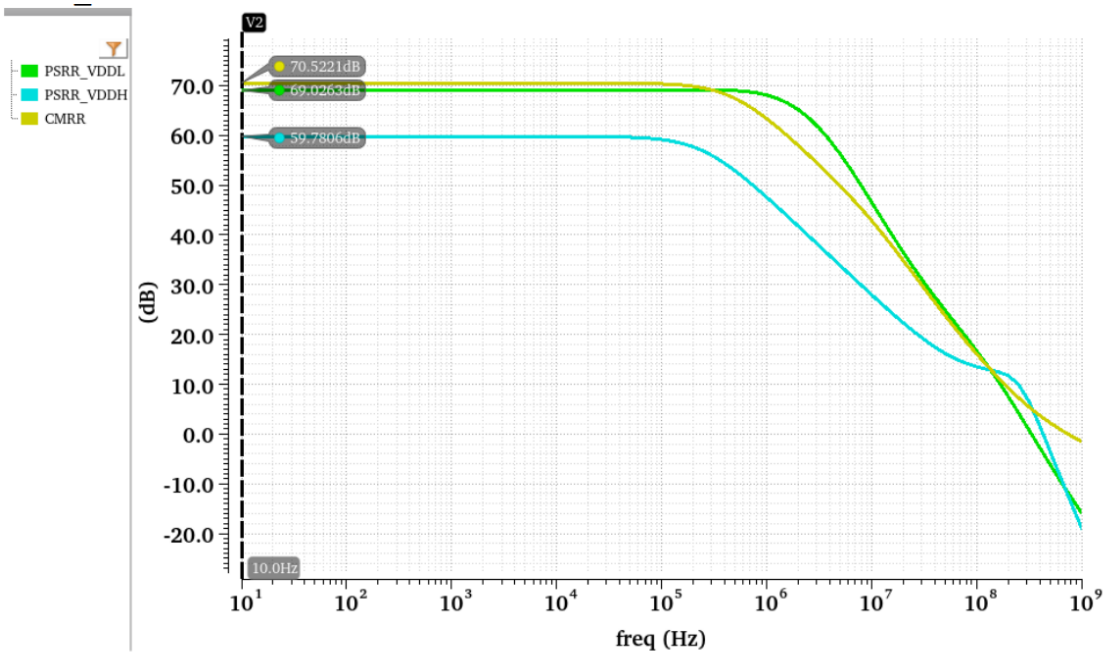
AC Figures

- **Bode plots of the loop gain**

  Unity gain bandwidth = 21.63 MHz.
  Phase Margin = 53.5 degree.

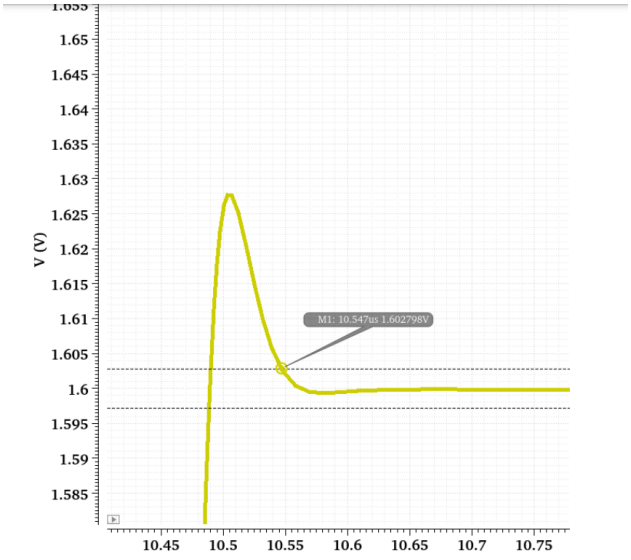


- **CMRR and PSRR vs frequency**
  CMRR at DC = 70.52 dB.
  PSRR_VDDL = 69.04 dB.
  PSRR_VDDH = 59.78 dB.

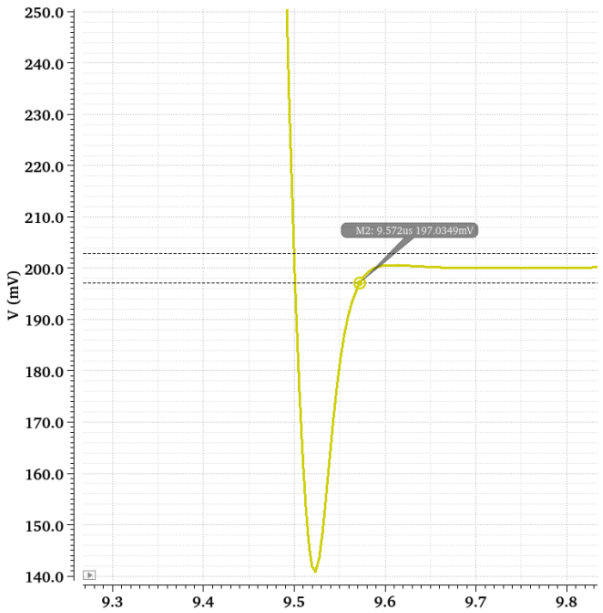


Transient Figures

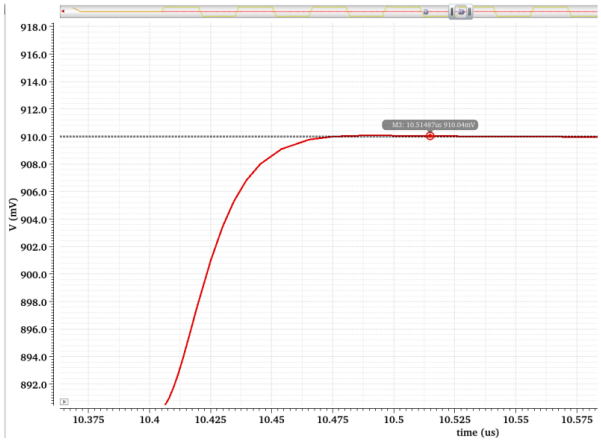
1. Waveform of the voltage at the load capacitor with legible markers for the settling time of the rising and falling 1.4V and 20mV output steps.
   Vout_Load (1.4V rising):
   Settling time $= 10.547\mu - 10.4\mu = 147$ ns



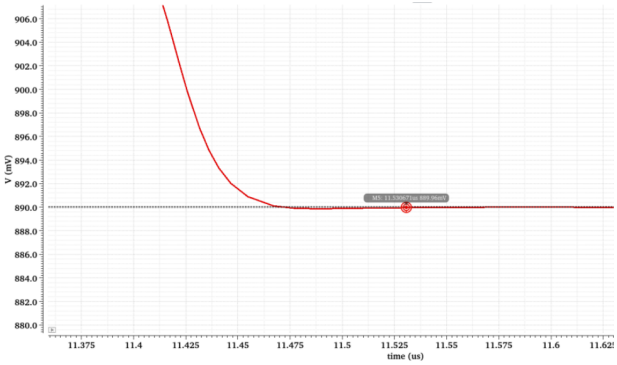
Vout_Load (1.4V falling):
Settling time $= 9.572\mu - 9.4\mu = 172$ ns.



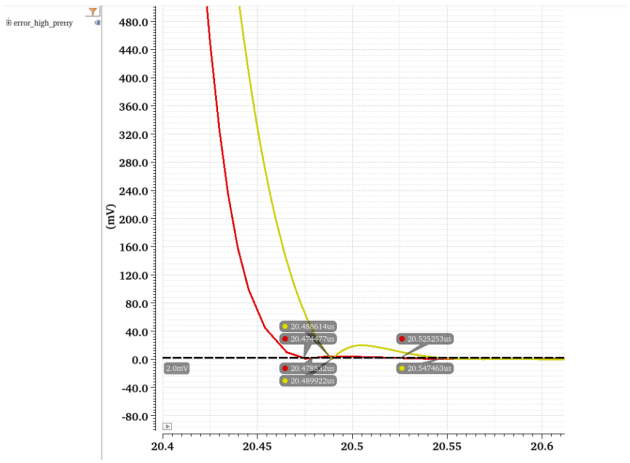
Vout_Load (20mV rising):
Settling time = $10.515\mu - 10.4\mu = 115$ ns.



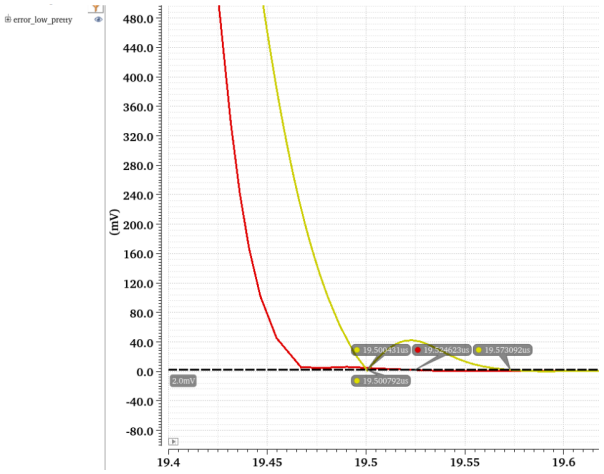
Vout_Load (20mV falling):
Settling time = $11.53\mu - 11.4\mu = 113$ ns.



2. Settling error for rising and falling 1.4V and 20mV output steps at at the load capacitor with legible markers.
   Settling error (high to low):
   Settling error = 0.2% Settling time = $20.547\mu s - 20.4\mu s = 147$ ns

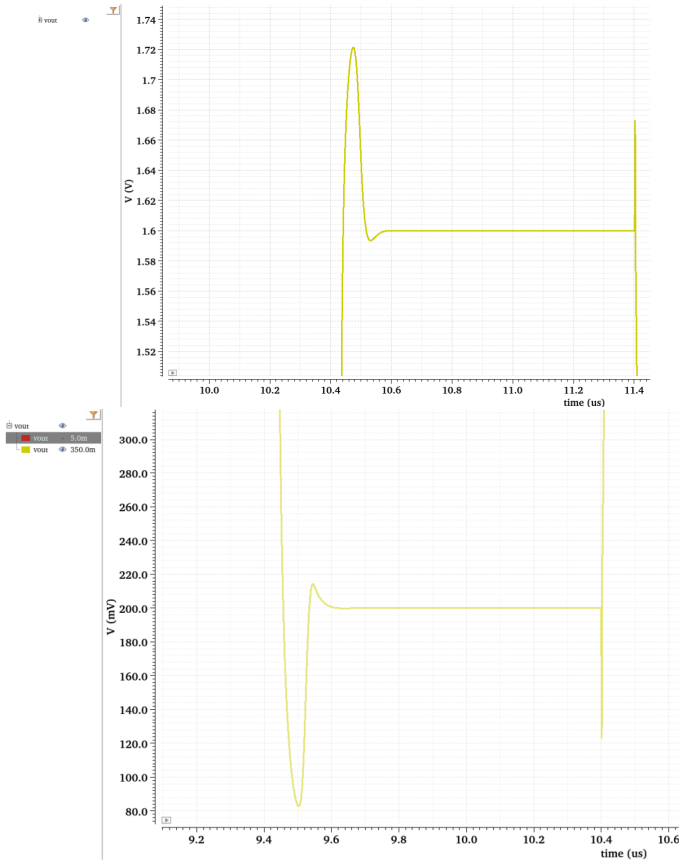


   Settling error (low to high):
   Settling error = 0.2% Settling time = $19.573\mu s - 19.4\mu s = 173$ ns

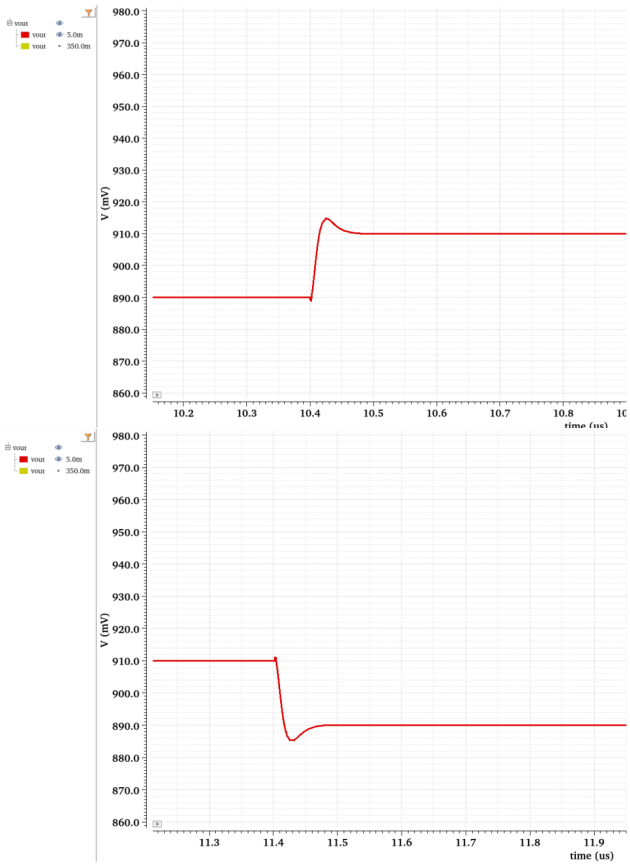


3. Waveform of the votlage at amplifier output for the rising and falling 1.4V and 20mV output steps.

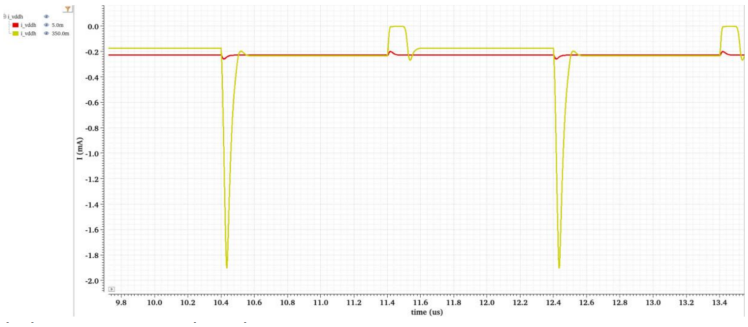
Vout_amplifier (1.4V rising) & Vout_amplifier (1.4V falling)::



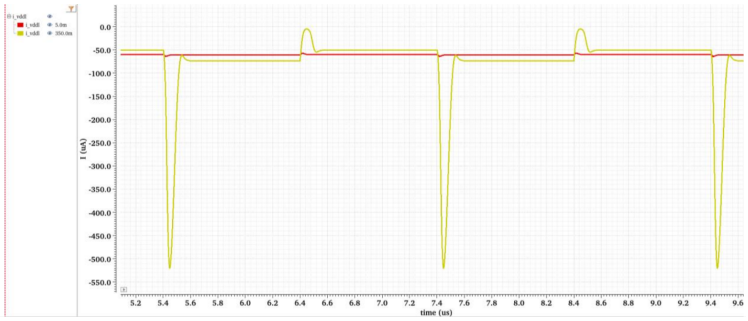
Vout_amplifier (20mV rising) & Vout_amplifier (20mV falling)::



4. Waveform of the currents drawn from VDDH and VDDL.
   Current drawn (VDDH):



Current drawn (VDDL):



Testbench & Summary Table

Testbench Used

Name	Type	
vout	expr	vtime('tran "/out_int")
vout_load	expr	vtime('tran "/out_load")
vin	expr	vtime('tran "/in")
vin_n	expr	vtime('tran "/in_n")
reset	expr	vtime('tran "/reset")
i_vddl	expr	getData("/V1/PLUS" ?result "tran")
i_vddh	expr	getData("/V0/PLUS" ?result "tran")
avg_pow_vddl	expr	((VAR("vddl") * integ(i_vddl 2.4e-06 1.04e-05 nil)) / 8e-06)
avg_pow_vddh	expr	((VAR("vddh") * integ(i_vddh 2.4e-06 1.04e-05 nil)) / 8e-06)
settling_error	expr	((value(vout 2.2e-05) - vout) / value(vout 2.2e-05))
vout_high	expr	value(vout_load 2.14e-05)
vout_low	expr	value(vout_load 2.04e-05)
error_high	expr	(abs((vout_load - vout_high)) / (4 * VAR("vamp")))
error_low	expr	(abs((vout_load - vout_low)) / (4 * VAR("vamp")))
error_high_pretty	expr	clip(error_high 2.04e-05 2.14e-05)
error_low_pretty	expr	clip(error_low 1.94e-05 2.04e-05)

Name	Type	Details
CM_Gain	expr	vfreq('ac "/vcm_out")
DM_Gain	expr	vfreq('ac "/vdm_out")
CMRR	expr	dB20((DM_Gain / CM_Gain))
CMRR_at_DC	expr	value(CMRR 10)
VDDL_Gain	expr	vfreq('ac "/vps_out_vddl")
VDDH_Gain	expr	vfreq('ac "/vps_out_vddh")
PSRR_VDDL	expr	dB20((DM_Gain / VDDL_Gain))
PSRR_VDDH	expr	dB20((DM_Gain / VDDH_Gain))
Loop_Gain_Phase	expr	phaseDegUnwrapped(getData("loopGain" ?result "stb"))
Loop_Gain_dB20	expr	db(mag(getData("loopGain" ?result "stb")))

The test bench shown here is used to help calculate and plot all the SPEC values. I added two parameters `error_high_pretty` and `error_low_pretty` to the test bench to plot the error from high to low and from low to high. The x-axis is time and the y-axis is error.

SPEC Table

Project	SPEC	Designed Value
Settling Time	180 ns	173 ns
Power Consumption	$\leq$ 1.1 mW	482.41 μW
Total Capacitance	4 pF	50 fF
Total Resistance	100 MΩ	100 MΩ
CMRR at DC	$\geq$ 65 dB	70.5221 dB
PSRR at DC	$\geq$ 50 dB	59.78 dB
Phase Margin	$\geq$ 45 degree	53.5 degree
FoM	10^{-9}/(Settling Time $\times Power$)	11.98

Conclusion

Summary & Experience: In this two-stage operational amplifier design project, I employed a telescope structure for the first stage to achieve high gain. The telescope structure is well-suited as a front-end amplifier due to its high input impedance and low output impedance. For the second stage, I opted for a Class-AB amplifier structure to enhance the output slew rate and drive capability. The Class-AB amplifier, consisting of complementary NPN and PNP transistors, forms a push-pull output stage that ensures a large output swing while maintaining low quiescent power consumption.

To meet the settling time specification, I calculated the required 3dB bandwidth of the amplifier and determined the location of the first pole accordingly. During the design process, I utilized frequency compensation techniques, employing compensation capacitors to optimize the location of the second pole, thereby achieving a reasonable phase margin and ensuring system stability. I optimized the bias currents and transistor sizes of each stage to meet the specifications while minimizing chip area and power consumption.

Through this project, I applied the theoretical knowledge learned in the classroom to practical circuit design. By analyzing the specification requirements, selecting appropriate circuit architectures, calculating, and optimizing device parameters, I successfully implemented a two-stage operational amplifier that meets the given specifications. Throughout the design process, I deepened my understanding of various performance metrics of op-amps, such as gain, bandwidth, phase margin, and output swing. Furthermore, I experienced the challenges of transitioning from theory to practice, honing my problem-solving and debugging skills. These valuable experiences have been immensely beneficial to me and will serve as important guidance for my future work in analog circuit design.

Feedback: Regarding checkoff, I had to skip it because of lack of time. I hope the final design result can make some compensation for the checkoff score.

Scripts

```
1
2  // Part1:
3
4  import numpy as np
5  import matplotlib . pyplot as plt
6  from look_up import *
7  nch = importdata ( 'nch_1v.mat')
8  pch = importdata ( 'pch_1v.mat')
9  nch = importdata('nch_2v.mat')
10 pch = importdata('pch_1v.mat')
11 Lp = pch['L']
12 Ln = nch['L']
13 fu = 1e8
14 gm_id_range = np.linspace(5, 30, 50)
15 CL=1e-12
16 VDD=1.8
17 Vout_range=np.linspace(0.1, 1.5, 100)
18
19
20 def telescope_lab(fu=1e9, CL=1e-12, VDD=1.8, Vout_range=np.
      linspace(0.1, 1.7, 100)):
21     pch = importdata('pch_1v.mat')
22     nch = importdata('nch_1v.mat')
23     Lp = pch['L']
24     Ln = nch['L']
25     gm_id_range = np.linspace(5, 30, 50)
26
27
28     av_max_values = []
29     vout_opt_values = []
30     gm_id_opt_m4_values = []
31     gm_id_opt_m6_values = []
32     l_opt_m4_values = []
33     l_opt_m6_values = []
34
35     for Vout in Vout_range:
36         VDS_m4 = Vout / 2
37         VDS_m6 = (VDD - Vout) / 2
38
39         av_values = []
40         gm_id_m4_values = []
41         gm_id_m6_values = []
42         l_m4_values = []
```

```
        l_m6_values = []

        for l_m4 in Ln:
            for l_m6 in Lp:
                gm_id_range = np.linspace(5, 30, 50)

                ft_m4 = look_up_vs_gm_id(nch, 'GM_CGG',
                   gm_id_range, vds=VDS_m4, l=l_m4) / (2 * np.pi)
                m_m4 = ft_m4 >= fu
                if np.any(m_m4):
                    gm_id_m4 = gm_id_range[max(np.where(m_m4 == 1)
                       [0])]
                    gds_id_m4 = look_up_vs_gm_id(nch, 'GDS_ID',
                       gm_id_m4, vds=VDS_m4, l=l_m4)
                else:
                    continue

                ft_m6 = look_up_vs_gm_id(pch, 'GM_CGG',
                   gm_id_range, vds=VDS_m6, l=l_m6) / (2 * np.pi)
                m_m6 = ft_m6 >= fu
                if np.any(m_m6):
                    gm_id_m6 = gm_id_range[max(np.where(m_m6 == 1)
                       [0])]
                    gds_id_m6 = look_up_vs_gm_id(pch, 'GDS_ID',
                       gm_id_m6, vds=VDS_m6, l=l_m6)
                else:
                    continue

                av = (gm_id_m4**2 * gm_id_m6) / (gm_id_m4 *
                   gds_id_m6**2 + gm_id_m6 * gds_id_m4)

                av_values.append(av)
                gm_id_m4_values.append(gm_id_m4)
                gm_id_m6_values.append(gm_id_m6)
                l_m4_values.append(l_m4)
                l_m6_values.append(l_m6)

        max_av_index = np.nanargmax(av_values)
        av_max_values.append(av_values[max_av_index])
        vout_opt_values.append(Vout)
        gm_id_opt_m4_values.append(gm_id_m4_values[max_av_index])
        gm_id_opt_m6_values.append(gm_id_m6_values[max_av_index])
        l_opt_m4_values.append(l_m4_values[max_av_index])
        l_opt_m6_values.append(l_m6_values[max_av_index])

```

```
    max_av_index = np.nanargmax(av_max_values)
    av_max = av_max_values[max_av_index]
    vout_opt = vout_opt_values[max_av_index]
    gm_id_opt_m4 = gm_id_opt_m4_values[max_av_index]
    gm_id_opt_m6 = gm_id_opt_m6_values[max_av_index]
    l_opt_m4 = l_opt_m4_values[max_av_index]
    l_opt_m6 = l_opt_m6_values[max_av_index]

    VDS_m4 = vout_opt / 2
    VDS_m6 = (VDD - vout_opt) / 2
    VGS_m4 = vout_opt
    VGS_m6 = VDD - vout_opt

    jd_m4 = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt_m4, vds=VDS_m4
       , l=l_opt_m4)
    jd_m6 = look_up_vs_gm_id(pch, 'ID_W', gm_id_opt_m6, vds=VDS_m6
       , l=l_opt_m6)
    cdd_w_m4 = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt_m4, vds=
       VDS_m4, l=l_opt_m4)
    cdd_w_m6 = look_up_vs_gm_id(pch, 'CDD_W', gm_id_opt_m6, vds=
       VDS_m6, l=l_opt_m6)
    cdd_m4 = cdd_m6 = 0

    for i in range(1, 10):
        gm_m4 = 2 * np.pi * fu * (CL + cdd_m4 + cdd_m6)
        gm_m6 = gm_m4
        ID_m4 = gm_m4 / gm_id_opt_m4
        ID_m6 = ID_m4
        W_m4 = ID_m4 / jd_m4
        W_m6 = ID_m6 / jd_m6
        cdd_m4 = W_m4 * cdd_w_m4
        cdd_m6 = W_m6 * cdd_w_m6

    result = {
        'fu': fu,
        'cdd_m4': cdd_m4,
        'cdd_m6': cdd_m6,
        'gm_m4': gm_m4,
        'gm_m6': gm_m6,
        'gm_id_opt_m4': gm_id_opt_m4,
        'gm_id_opt_m6': gm_id_opt_m6,
        'l_opt_m4': l_opt_m4,
        'l_opt_m6': l_opt_m6,
        'vout_opt': vout_opt,
        'VGS_m4': VGS_m4,
```

```
        'VGS_m6': VGS_m6,
        'VDS_m4': VDS_m4,
        'VDS_m6': VDS_m6,
        'ID_m4': ID_m4,
        'ID_m6': ID_m6,
        'W_m4': W_m4,
        'W_m6': W_m6,
        'av_max': av_max
    }

    return result


pch = importdata('pch_2v.mat')
nch = importdata('nch_2v.mat')
Lp = pch['L']
Ln = nch['L']

fu = 1e9
CL = 1e-12
VDD = 1.8
Vout_range = np.linspace(0.1, 1.7, 100)

av_max_values = []
vout_opt_values = []
gm_id_opt_m4_values = []
gm_id_opt_m6_values = []
l_opt_m4_values = []
l_opt_m6_values = []

for Vout in Vout_range:
    VDS_m4 = Vout / 2
    VDS_m6 = (VDD - Vout) / 2

    av_values = []
    gm_id_m4_values = []
    gm_id_m6_values = []
    l_m4_values = []
    l_m6_values = []

    for l_m4 in Ln:
        for l_m6 in Lp:
            gm_id_range = np.linspace(5, 30, 50)

```

```
            ft_m4 = look_up_vs_gm_id(nch, 'GM_CGG', gm_id_range,
                vds=VDS_m4, l=l_m4) / (2 * np.pi)
            m_m4 = ft_m4 >= fu
            if np.any(m_m4):
                gm_id_m4 = gm_id_range[max(np.where(m_m4 == 1)[0])
                    ]
                gds_id_m4 = look_up_vs_gm_id(nch, 'GDS_ID',
                    gm_id_m4, vds=VDS_m4, l=l_m4)
            else:
                continue

            ft_m6 = look_up_vs_gm_id(pch, 'GM_CGG', gm_id_range,
                vds=VDS_m6, l=l_m6) / (2 * np.pi)
            m_m6 = ft_m6 >= fu
            if np.any(m_m6):
                gm_id_m6 = gm_id_range[max(np.where(m_m6 == 1)[0])
                    ]
                gds_id_m6 = look_up_vs_gm_id(pch, 'GDS_ID',
                    gm_id_m6, vds=VDS_m6, l=l_m6)
            else:
                continue

            av = (gm_id_m4**2 * gm_id_m6) / (gm_id_m4 * gds_id_m6
                **2 + gm_id_m6 * gds_id_m4)

            av_values.append(av)
            gm_id_m4_values.append(gm_id_m4)
            gm_id_m6_values.append(gm_id_m6)
            l_m4_values.append(l_m4)
            l_m6_values.append(l_m6)

    max_av_index = np.nanargmax(av_values)
    av_max_values.append(av_values[max_av_index])
    vout_opt_values.append(Vout)
    gm_id_opt_m4_values.append(gm_id_m4_values[max_av_index])
    gm_id_opt_m6_values.append(gm_id_m6_values[max_av_index])
    l_opt_m4_values.append(l_m4_values[max_av_index])
    l_opt_m6_values.append(l_m6_values[max_av_index])

max_av_index = np.nanargmax(av_max_values)
av_max = av_max_values[max_av_index]
vout_opt = vout_opt_values[max_av_index]
gm_id_opt_m4 = gm_id_opt_m4_values[max_av_index]
gm_id_opt_m6 = gm_id_opt_m6_values[max_av_index]
l_opt_m4 = l_opt_m4_values[max_av_index]
```

```
l_opt_m6 = l_opt_m6_values[max_av_index]

VDS_m4 = vout_opt / 2
VDS_m6 = (VDD - vout_opt) / 2
VGS_m4 = vout_opt
VGS_m6 = VDD - vout_opt

jd_m4 = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt_m4, vds=VDS_m4, l=
    l_opt_m4)
jd_m6 = look_up_vs_gm_id(pch, 'ID_W', gm_id_opt_m6, vds=VDS_m6, l=
    l_opt_m6)
cdd_w_m4 = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt_m4, vds=VDS_m4
    , l=l_opt_m4)
cdd_w_m6 = look_up_vs_gm_id(pch, 'CDD_W', gm_id_opt_m6, vds=VDS_m6
    , l=l_opt_m6)
cdd_m4 = cdd_m6 = 0

for i in range(1, 10):
    gm_m4 = 2 * np.pi * fu * (CL + cdd_m4 + cdd_m6)
    gm_m6 = gm_m4
    ID_m4 = gm_m4 / gm_id_opt_m4
    ID_m6 = ID_m4
    W_m4 = ID_m4 / jd_m4
    W_m6 = ID_m6 / jd_m6
    cdd_m4 = W_m4 * cdd_w_m4
    cdd_m6 = W_m6 * cdd_w_m6

print(f"fu: {fu}")
print(f"cdd_m4: {cdd_m4}")
print(f"cdd_m6: {cdd_m6}")
print(f"gm_m4: {gm_m4}")
print(f"gm_m6: {gm_m6}")
print(f"gm_id_opt_m4: {gm_id_opt_m4}")
print(f"gm_id_opt_m6: {gm_id_opt_m6}")
print(f"l_opt_m4: {l_opt_m4}")
print(f"l_opt_m6: {l_opt_m6}")
print(f"vout_opt: {vout_opt}")
print(f"VGS_m4: {VGS_m4}")
print(f"VGS_m6: {VGS_m6}")
print(f"VDS_m4: {VDS_m4}")
print(f"VDS_m6: {VDS_m6}")
print(f"ID_m4: {ID_m4}")
print(f"ID_m6: {ID_m6}")
print(f"W_m4: {W_m4}")
print(f"W_m6: {W_m6}")
```

```
print(f"av_max: {av_max}")


Part2:
import numpy as np
import matplotlib.pyplot as plt
from look_up import *

# Load data
pch = importdata('pch_2v.mat')
nch = importdata('nch_2v.mat')
Lp = pch['L']
Ln = nch['L']

# Sweep gm/id for stage 1
def stage1_sizing(fu, CL, VDD, Vout_range):
    av_max_values = []
    vout_opt_values = []
    gm_id_opt_m4_values = []
    gm_id_opt_m6_values = []
    l_opt_m4_values = []
    l_opt_m6_values = []

    for Vout in Vout_range:
        VDS_m4 = Vout / 2
        VDS_m6 = (VDD - Vout) / 2

        av_values = []
        gm_id_m4_values = []
        gm_id_m6_values = []
        l_m4_values = []
        l_m6_values = []

        for l_m4 in Ln:
            for l_m6 in Lp:
                gm_id_range = np.linspace(5, 30, 50)

                ft_m4 = look_up_vs_gm_id(nch, 'GM_CGG',
                   gm_id_range, vds=VDS_m4, l=l_m4) / (2 * np.pi)
                m_m4 = ft_m4 >= fu
                if np.any(m_m4):
                    gm_id_m4 = gm_id_range[max(np.where(m_m4 == 1)
                       [0])]
                    gds_id_m4 = look_up_vs_gm_id(nch, 'GDS_ID',
                       gm_id_m4, vds=VDS_m4, l=l_m4)
```

```
                else:
                    continue

                ft_m6 = look_up_vs_gm_id(pch, 'GM_CGG',
                   gm_id_range, vds=VDS_m6, l=l_m6) / (2 * np.pi)
                m_m6 = ft_m6 >= fu
                if np.any(m_m6):
                    gm_id_m6 = gm_id_range[max(np.where(m_m6 == 1)
                       [0])]
                    gds_id_m6 = look_up_vs_gm_id(pch, 'GDS_ID',
                       gm_id_m6, vds=VDS_m6, l=l_m6)
                else:
                    continue

                av = (gm_id_m4**2 * gm_id_m6) / (gm_id_m4 *
                   gds_id_m6**2 + gm_id_m6 * gds_id_m4)

                av_values.append(av)
                gm_id_m4_values.append(gm_id_m4)
                gm_id_m6_values.append(gm_id_m6)
                l_m4_values.append(l_m4)
                l_m6_values.append(l_m6)

        max_av_index = np.nanargmax(av_values)
        av_max_values.append(av_values[max_av_index])
        vout_opt_values.append(Vout)
        gm_id_opt_m4_values.append(gm_id_m4_values[max_av_index])
        gm_id_opt_m6_values.append(gm_id_m6_values[max_av_index])
        l_opt_m4_values.append(l_m4_values[max_av_index])
        l_opt_m6_values.append(l_m6_values[max_av_index])

    max_av_index = np.nanargmax(av_max_values)
    av_max = av_max_values[max_av_index]
    vout_opt = vout_opt_values[max_av_index]
    gm_id_opt_m4 = gm_id_opt_m4_values[max_av_index]
    gm_id_opt_m6 = gm_id_opt_m6_values[max_av_index]
    l_opt_m4 = l_opt_m4_values[max_av_index]
    l_opt_m6 = l_opt_m6_values[max_av_index]

    VDS_m4 = vout_opt / 2
    VDS_m6 = (VDD - vout_opt) / 2
    VGS_m4 = vout_opt
    VGS_m6 = VDD - vout_opt

```

```
    jd_m4 = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt_m4, vds=VDS_m4
        , l=l_opt_m4)
    jd_m6 = look_up_vs_gm_id(pch, 'ID_W', gm_id_opt_m6, vds=VDS_m6
        , l=l_opt_m6)
    cdd_w_m4 = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt_m4, vds=
        VDS_m4, l=l_opt_m4)
    cdd_w_m6 = look_up_vs_gm_id(pch, 'CDD_W', gm_id_opt_m6, vds=
        VDS_m6, l=l_opt_m6)
    cdd_m4 = cdd_m6 = 0

    for i in range(1, 10):
        gm_m4 = 2 * np.pi * fu * (CL + cdd_m4 + cdd_m6)
        gm_m6 = gm_m4
        ID_m4 = gm_m4 / gm_id_opt_m4
        ID_m6 = ID_m4
        W_m4 = ID_m4 / jd_m4
        W_m6 = ID_m6 / jd_m6
        cdd_m4 = W_m4 * cdd_w_m4
        cdd_m6 = W_m6 * cdd_w_m6

    return gm_id_opt_m4, gm_id_opt_m6, l_opt_m4, l_opt_m6, av_max,
        vout_opt, VGS_m4, VDS_m4, gm_m4, ID_m4, W_m4, VGS_m6,
       VDS_m6, gm_m6, ID_m6, W_m6

# Sweep gm/id for stage 2
def stage2_sizing(gm_stage1, ID_stage1, Vout_stage1):
    gm_id_range = np.linspace(6, 20, 50)

    # Sweep gmp to match input voltage
    gmp_values = []
    vgs_values = []
    for gmp in gm_id_range:
        vgs = look_up_vs_gm_id(pch, 'VGS', gmp, vds=1.15, l=0.18)
        if abs(vgs - Vout_stage1) < 0.01:
            gmp_values.append(gmp)
            vgs_values.append(vgs)
    gmp_opt = gmp_values[np.argmin(abs(vgs_values - Vout_stage1))]

    # Sweep gmn for largest current
    gmn_values = []
    id_values = []
    for gmn in gm_id_range:
        id = look_up_vs_gm_id(nch, 'ID_W', gmn, vds=0.65, l=0.18)
           * 10  # assume W=10um
        gmn_values.append(gmn)
```

```
365             id_values.append(id)
366
367     gmn_opt_idx = np.argmax(id_values)
368     gmn_opt = gmn_values[gmn_opt_idx]
369     id_opt = id_values[gmn_opt_idx]
370
371     Wp_min = (gm_stage1/gmp_opt) / look_up_vs_gm_id(pch, 'ID_W',
            gmp_opt, vds=1.15, l=0.18)
372     Wn_min = (gm_stage1/gmn_opt) / look_up_vs_gm_id(nch, 'ID_W',
            gmn_opt, vds=0.65, l=0.18)
373
374     return gmp_opt, gmn_opt, Wp_min, Wn_min, id_opt
375
376 # Testbench
377 def testbench(design):
378     print(f"Stage 1 gm/id (NMOS): {design['gm_id_opt_m4']:.2f} V
            ^-1")
379     print(f"Stage 1 gm/id (PMOS): {design['gm_id_opt_m6']:.2f} V
            ^-1")
380     print(f"Stage 1 L (NMOS): {design['l_opt_m4']*1e6:.2f} um")
381     print(f"Stage 1 L (PMOS): {design['l_opt_m6']*1e6:.2f} um")
382     print(f"Stage 1 Av_max: {design['av_max']:.2f} V/V")
383     print(f"Stage 1 Vout_opt: {design['vout_opt']:.2f} V")
384     print(f"Stage 1 VGS (NMOS): {design['VGS_m4']:.2f} V")
385     print(f"Stage 1 VDS (NMOS): {design['VDS_m4']:.2f} V")
386     print(f"Stage 1 gm (NMOS): {design['gm_m4']*1e3:.2f} mA/V")
387     print(f"Stage 1 ID (NMOS): {design['ID_m4']*1e3:.2f} mA")
388     print(f"Stage 1 W (NMOS): {design['W_m4']*1e6:.2f} um")
389     print(f"Stage 1 VGS (PMOS): {design['VGS_m6']:.2f} V")
390     print(f"Stage 1 VDS (PMOS): {design['VDS_m6']:.2f} V")
391     print(f"Stage 1 gm (PMOS): {design['gm_m6']*1e3:.2f} mA/V")
392     print(f"Stage 1 ID (PMOS): {design['ID_m6']*1e3:.2f} mA")
393     print(f"Stage 1 W (PMOS): {design['W_m6']*1e6:.2f} um")
394
395     print(f"Stage 2 gm/id (PMOS): {design['gmp_opt']:.2f} V^-1")
396     print(f"Stage 2 gm/id (NMOS): {design['gmn_opt']:.2f} V^-1")
397     print(f"Stage 2 W (PMOS): {design['Wp_min']*1e6:.2f} um")
398     print(f"Stage 2 W (NMOS): {design['Wn_min']*1e6:.2f} um")
399     print(f"Stage 2 ID: {design['id_opt']*1e3:.2f} mA")
400
401 # Main
402 if __name__ == '__main__':
403     fu = 1e9
404     CL = 1e-12
405     VDD = 1.8
```

```
    Vout_range = np.linspace(0.1, 1.7, 100)

    gm_id_opt_m4, gm_id_opt_m6, l_opt_m4, l_opt_m6, av_max,
       vout_opt, VGS_m4, VDS_m4, gm_m4, ID_m4, W_m4, VGS_m6,
       VDS_m6, gm_m6, ID_m6, W_m6 = stage1_sizing(fu, CL, VDD,
       Vout_range)

    gmp_opt, gmn_opt, Wp_min, Wn_min, id_opt = stage2_sizing(gm_m4
       , ID_m4, vout_opt)

    design = {
        'gm_id_opt_m4': gm_id_opt_m4,
        'gm_id_opt_m6': gm_id_opt_m6,
        'l_opt_m4': l_opt_m4,
        'l_opt_m6': l_opt_m6,
        'av_max': av_max,
        'vout_opt': vout_opt,
        'VGS_m4': VGS_m4,
        'VDS_m4': VDS_m4,
        'gm_m4': gm_m4,
        'ID_m4': ID_m4,
        'W_m4': W_m4,
        'VGS_m6': VGS_m6,
        'VDS_m6': VDS_m6,
        'gm_m6': gm_m6,
        'ID_m6': ID_m6,
        'W_m6': W_m6,
        'gmp_opt': gmp_opt,
        'gmn_opt': gmn_opt,
        'Wp_min': Wp_min,
        'Wn_min': Wn_min,
        'id_opt': id_opt
    }

    testbench(design)
```