EE140 Lab Report: Lab1

Yu Li
SID: 3039817966

April 2, 2024

Exercise 1

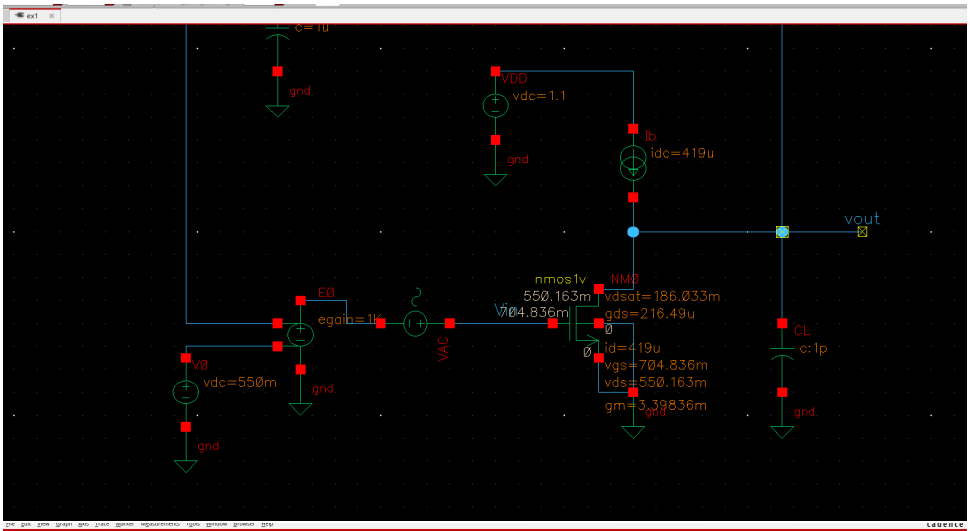
- **Design and Script**
  I set up known parameters like gmID, L, etc. Then I use the lookup function for fT, Av and gds. With fu and CL, gm can be calculated, then get ID with gm and gmID. Next I can get JD with lookup method, and finally W.

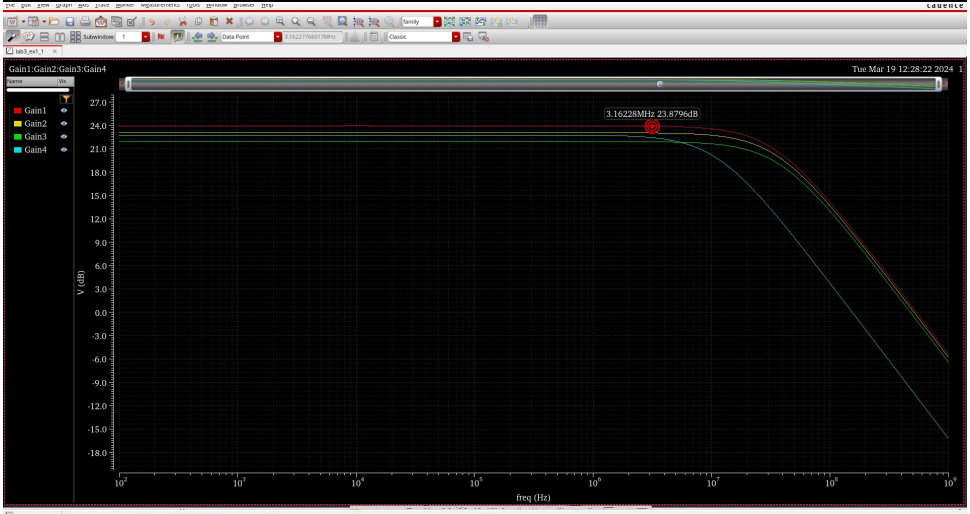
  Script is as follows:

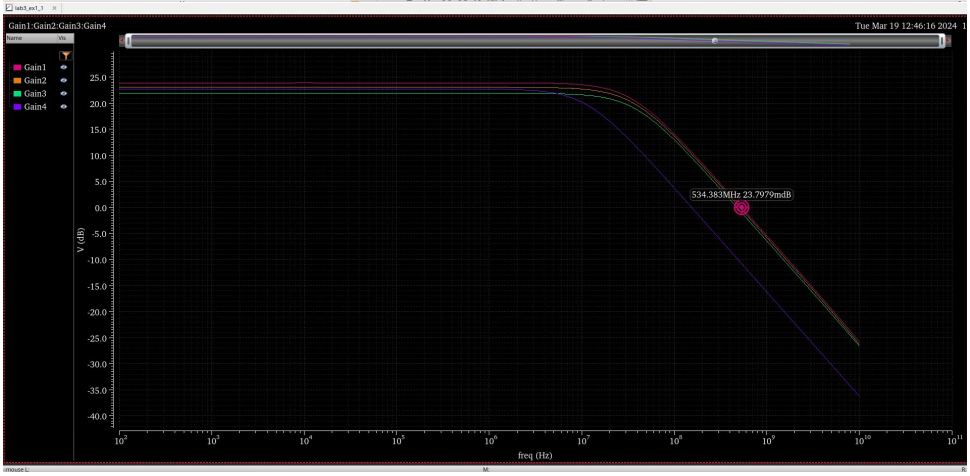
```
# EX1
# assumptions:
gm_ID = 15  # S/A
L = 0.1  # 100nm=0.1um
CL = 1e-12  # 1 pF
fu = 1e9  # 1 GHz
VDS = 0.55  # V
VSB = 0  # V,

fT = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, vds=VDS, l=L) / (2 *
    np.pi)
Av = look_up_vs_gm_id (nch , 'GM_GDS', gm_ID, vds=VDS, l=L)
gds = look_up_vs_gm_id(nch, 'GDS', gm_ID, vds=VDS, l=L)

print(f"gds:{gds}")

gm = 2 * np.pi * fu * CL
print(f"gm:{gm}")

ro = 1 / gds
intrinsic_gain = gm * ro

print(f"gain :{Av}dB")

ID = gm / gm_ID

JD = look_up_vs_gm_id(nch,'ID_W',gm_ID, vds=VDS, l=L)
# print(JD)
W = ID / JD
print(f"ID:{ID}")
VGS = look_up_vgs_vs_gm_id(nch, gm_ID, l=L, vds=VDS)

print(f"Intrinsic Gain: {intrinsic_gain}")
print(f"fT: {fT} Hz")
print(f"Width (W): {W} nm")
```

```
35  print(f"Required VGS: {VGS} V")
```

- **Cadence simulation**
  The cadence simulation schematic and bode plot are as follow:
  Gain1 curve is for exercise1.

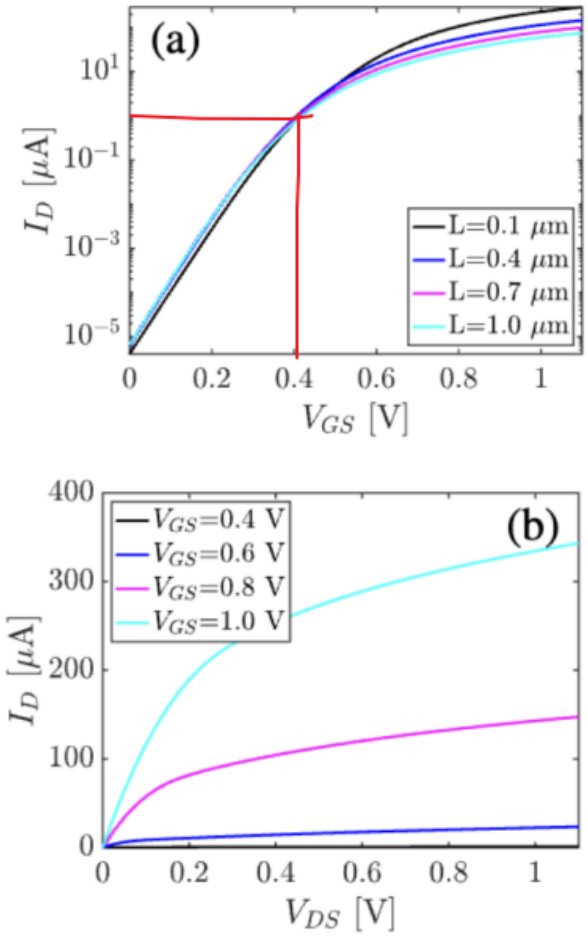




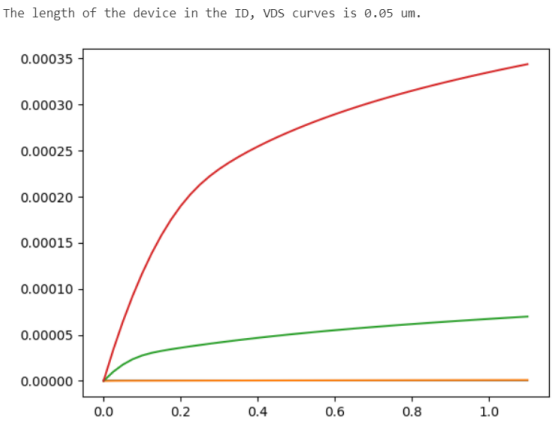


- **Analysis of the results**

1. **Question1**
   The threshold voltage is the VGS when the drain current exceeds 1 µA for a 1 µm width device. So from the figure we can estimate threshold voltage is approximately 0.4V.



2. **Question2**
   According to the code search, the corresponding length of L found is 0.05um.



3. **Results Analysis**
   The curves and results displayed by the simulation and the calculated results are listed in the

table below.

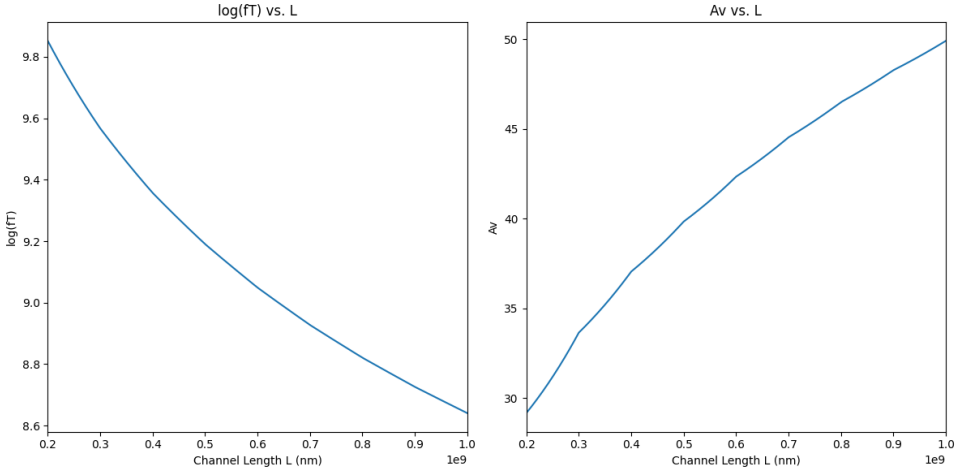
Performance	Design result	Simulation result
Gain(dB)	22.37	23.38
Unity gain bandwidth(Hz)	1G (fu)	534.4M
$gm_{ID}(S/A)$	15	8.11
$VGS(V)$	0.54	0.55

It can be found that all the values are relatively close, and the biggest difference is Unity gain bandwidth.

Exercise 2

- **Design and Script**

For the design of exercise 2, we must first draw the images of ft vs. L and Av vs. L according to the question requirements, as shown in the figure below.



In order to satisfy the condition of fu¡=ft/10, we need to select appropriate segments based on the image, and then select the points that meet the conditions in these segments to be the parameters corresponding to the solution we need.
Then we selected the two solutions and repeated the same operations as exercise1 in the script to obtain two sets of results. The next step is to use the same circuit diagram and change the parameters to obtain different Cadence simulation results.

```
#EX2
gm_ID = 15  # S/A
CL = 1e-12  # 1 pF
fu = 5e8  # 500 MHz
VDS = 0.55  # V
VSB = 0  # V
L_values = np.linspace(0.2, 1.0, 1000)  # Adjusted to a valid
    number of points

# Containers for results
fT_values = []
Av_values = []
W_values = []
```

```
13
14 for L in L_values:
15     # Correct function calls
16     fT = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, vds=VDS, l=L) / (2
       * np.pi)
17     Av = look_up_vs_gm_id(nch, 'GM_GDS', gm_ID, vds=VDS, l=L)  #
       Corrected to L
18     gds = look_up_vs_gm_id(nch, 'GDS', gm_ID, vds=VDS, l=L)
19     gm = 2 * np.pi * fu * CL
20     ID = gm / gm_ID
21     JD = look_up_vs_gm_id(nch, 'ID_W', gm_ID, vds=VDS, l=L)
22     W = ID / JD
23     VGS = look_up_vgs_vs_gm_id(nch, gm_ID, l=L, vds=VDS)
24
25     fT_values.append(fT)
26     Av_values.append(Av)
27     W_values.append(W)
28
29 # Find the maximum value of L that satisfies fu     fT/10
30 fT_values = np.array(fT_values)
31 L_max_index = np.where(fT_values >= 10 * fu)[0][-1]
32 L_max = L_values[L_max_index]
33
34 print(f"Maximum value of L that satisfies fu     fT/10:
       {L_max:.3f}")
35
36 # Plot log(fT) vs. L
37 plt.figure(figsize=(12, 6))
38 L_values_nm = L_values * 1e9
39 plt.subplot(1, 2, 1)
40 plt.plot(L_values_nm, np.log10(fT_values))
41 plt.xlabel('Channel Length L (nm)')
42 plt.ylabel('log(fT)')
43 plt.title('log(fT) vs. L')
44 plt.xlim(L_values_nm.min(), L_values_nm.max())
45
46 # Plot Av vs. L
47 plt.subplot(1, 2, 2)
48 plt.plot(L_values_nm, Av_values)
49 plt.xlabel('Channel Length L (nm)')
50 plt.ylabel('Av')
51 plt.title('Av vs. L')
52 plt.xlim(L_values_nm.min(), L_values_nm.max())
53
54 plt.tight_layout()
55 plt.show()
```

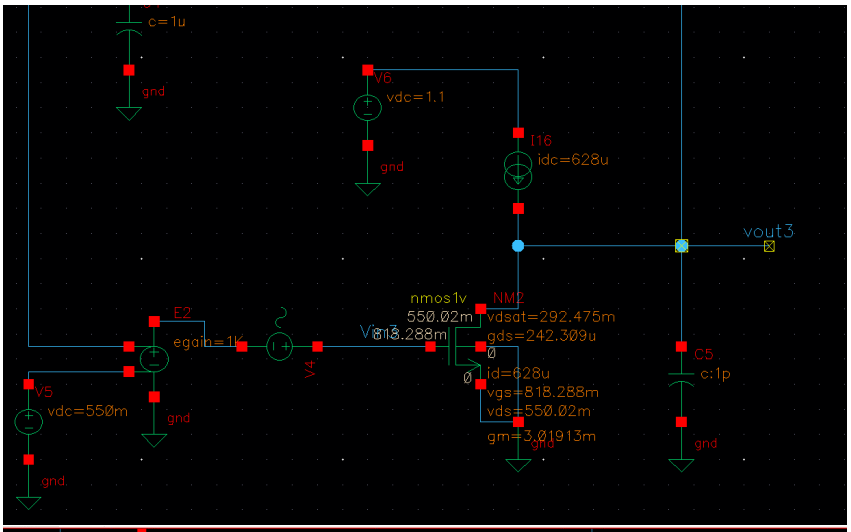
Then we can select two solutions and use script to obtain corresponding parameters.

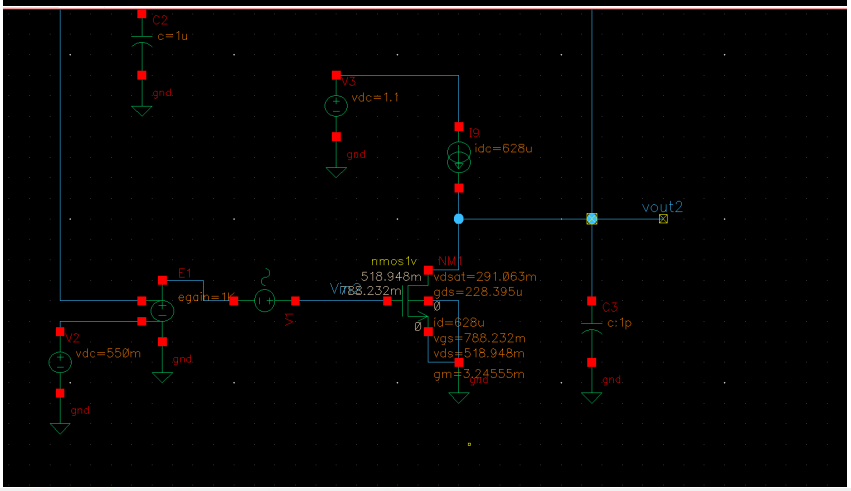
```
1 gm_ID = 5  # V^(-1)
2 CL = 1e-12  # 1 pF
3 fu = 5e8  # 500 MHz
4 VDS = 0.55  # V
5 VSB = 0  # V
6
7 # Solution 1
```

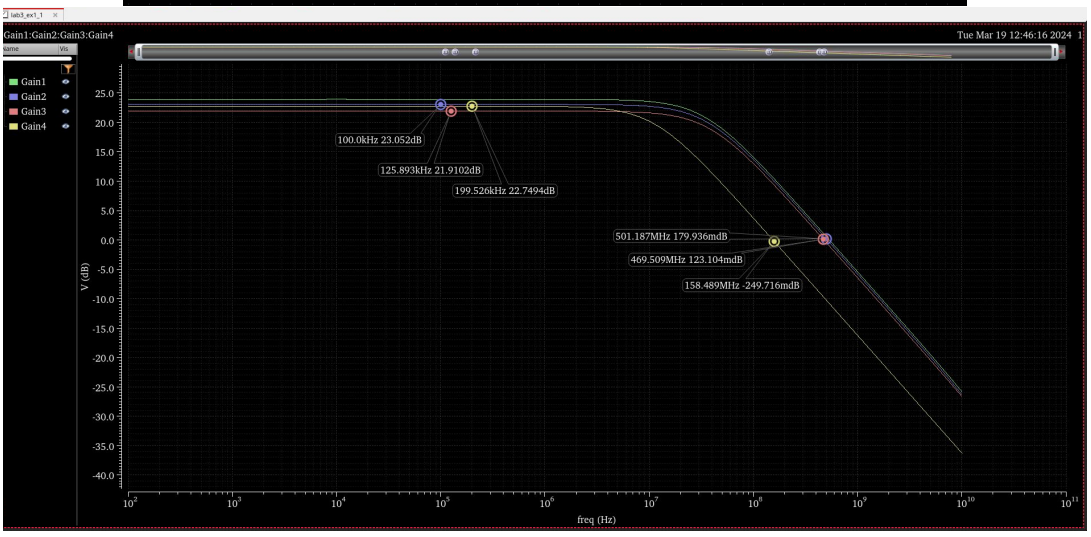
```
L_1 = 0.200  # Maximum value of L that satisfies fu     fT/10

fT_1 = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, vds=VDS, l=L_1) / (2
    * np.pi)
Av_1 = look_up_vs_gm_id(nch, 'GM_GDS', gm_ID, vds=VDS, l=L_1)
gds_1 = look_up_vs_gm_id(nch, 'GDS', gm_ID, vds=VDS, l=L_1)

print(f"Solution 1:")
print(f"gds: {gds_1}")

gm_1 = 2 * np.pi * fu * CL
print(f"gm: {gm_1}")

ro_1 = 1 / gds_1
intrinsic_gain_1 = gm_1 * ro_1
print(f"Av_1:{Av_1}")

ID_1 = gm_1 / gm_ID

JD_1 = look_up_vs_gm_id(nch, 'ID_W', gm_ID, vds=VDS, l=L_1)
W_1 = ID_1 / JD_1
print(f"ID: {ID_1}")
VGS_1 = look_up_vgs_vs_gm_id(nch, gm_ID, l=L_1, vds=VDS)

print(f"Intrinsic Gain: {intrinsic_gain_1}")
print(f"fT: {fT_1} Hz")
print(f"Width (W): {W_1}um")
print(f"Required VGS: {VGS_1} V")

# Solution 2
L_2 = 0.150  # Different L value

fT_2 = look_up_vs_gm_id(nch, 'GM_CGG', gm_ID, vds=VDS, l=L_2) / (2
    * np.pi)
Av_2 = look_up_vs_gm_id(nch, 'GM_GDS', gm_ID, vds=VDS, l=L_2)
gds_2 = look_up_vs_gm_id(nch, 'GDS', gm_ID, vds=VDS, l=L_2)

print(f"\nSolution 2:")
print(f"gds: {gds_2}")

gm_2 = 2 * np.pi * fu * CL
print(f"gm: {gm_2}")

ro_2 = 1 / gds_2
intrinsic_gain_2 = gm_2 * ro_2
print(f"Av_2:{Av_2}")

ID_2 = gm_2 / gm_ID

JD_2 = look_up_vs_gm_id(nch, 'ID_W', gm_ID, vds=VDS, l=L_2)
W_2 = ID_2 / JD_2
print(f"ID: {ID_2}")
VGS_2 = look_up_vgs_vs_gm_id(nch, gm_ID, l=L_2, vds=VDS)

print(f"Intrinsic Gain: {intrinsic_gain_2}")
print(f"fT: {fT_2} Hz")
```

```
print(f"Width (W): {W_2} um")
print(f"Required VGS: {VGS_2} V")
```

- **Cadence simulation**
  Gain2 and Gain3 curves are for exericise2.







- **Analysis of the results**

The corresponding simulation results for the two sets of parameters selected from the script drawing that meet the conditions are as follows:

Solution1

Performance	Design result	Simulation result
Gain(dB)	24.74	23.05
Gain bandwidth(Hz)	-	501M
$gm_{ID}(S/A)$	5	5.17
$VGS(V)$	0.788	0.788

Solution2

Performance	Design result	Simulation result
Gain(dB)	22.92	22.91
Gain bandwidth(Hz)	-	469M
$gm_{ID}(S/A)$	5	4.81
$VGS(V)$	0.798	0.818

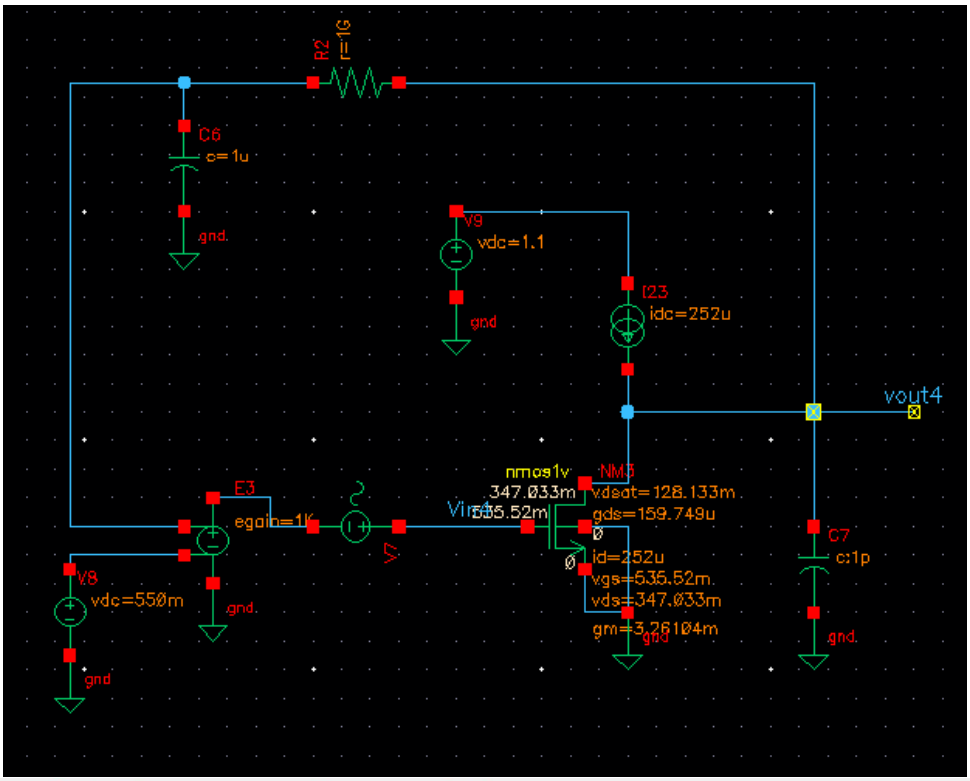
Exercise 3

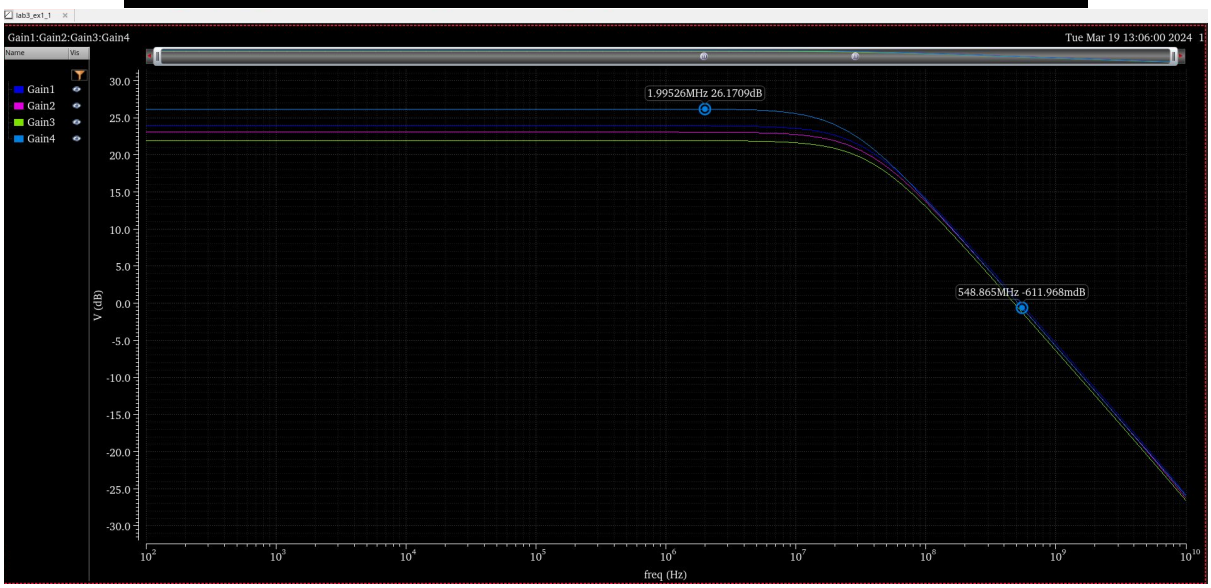
- **Design and Script**
  The purpose of exercise three is mainly to design the maximum gain model. We only need to supplement the corresponding purpose code according to the idea of template code. The detailed code is below:
  1) Find ft: Use lookup to find ft, and then use a judgment condition to filter the list of ft that meets the condition.
  2) Obtain av [i] = gm / gds: This step is to find out gm and gds respectively through lookup and then directly find the av list that meets the conditions.
  3) Lookup to find JD: The last step is to find JD through lookup, and iterate W in order to consider the impact of cdd.

```
import numpy as np
import matplotlib.pyplot as plt
from look_up import *
nch = importdata ( 'nch_1v.mat')

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):
    # TODO: Find ft
    ft = look_up_vs_gm_id(nch, 'GM_CGG', gm_id_range, vds=VDS,
        l=l) / (2 * np.pi)
    m = ft >= 10 * fu
    if np.any(m):
        gm_id.append(gm_id_range[max(np.where(m == 1)[0])])
    else:
```

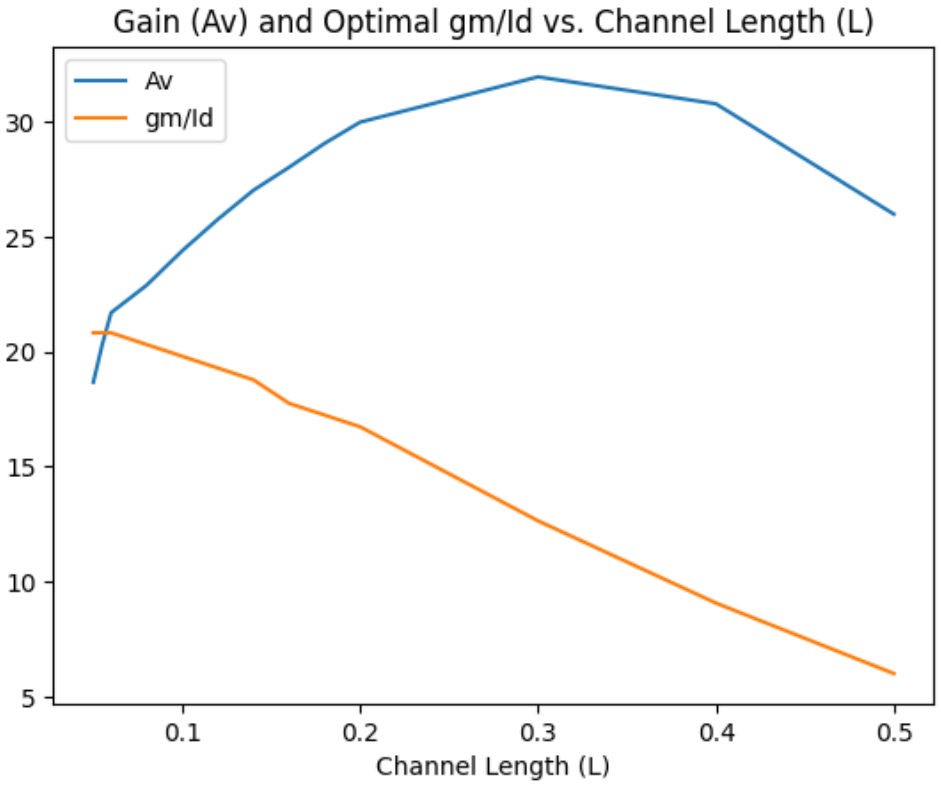
```
            gm_id.append(float('nan'))

        # TODO: Obtain av[i] = gm / gds
        gm = look_up_vs_gm_id(nch, 'GM', gm_id[i], vds=VDS, l=l)
        gds = look_up_vs_gm_id(nch, 'GDS', gm_id[i], vds=VDS, l=l)
        av.append(gm / gds)

fig, ax = plt.subplots(1)
ax.plot(L, av, label='Av')

ax.plot(L, gm_id, label='gm/Id')
ax.legend()
plt.xlabel('Channel Length (L)')
plt.title('Gain (Av) and Optimal gm/Id vs. Channel Length (L)')
plt.show()

gm_id_opt = gm_id[np.nanargmax(av)]
l_opt = L[np.nanargmax(av)]

# TODO: Lookup to find JD
jd = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt, vds=VDS, l=l_opt)

cdd_w = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt, vds=VDS, l=l_opt)
cdd = 0
for i in range(1, 10):
    gm_opt = 2 * np.pi * fu * (CL + cdd)
    ID = gm_opt / gm_id_opt
    W = ID / jd
    cdd = W * cdd_w

print(f"cdd:{cdd}")
print(f"gm_opt:{gm_opt}")
print(f"gm_id_opt: {gm_id_opt}")
print(f"l_opt:{l_opt}")


# Simulation parameter
fT_3 = look_up_vs_gm_id(nch, 'GM_CGG', gm_id_opt, vds=VDS,
   l=l_opt) / (2 * np.pi)
Av_3 = look_up_vs_gm_id(nch, 'GM_GDS', gm_id_opt, vds=VDS, l=l_opt)
gds_3 = look_up_vs_gm_id(nch, 'GDS', gm_id_opt, vds=VDS, l=l_opt)

ro = 1 / gds_3
intrinsic_gain_3 = gm_opt * ro

ID_3 = gm_opt / gm_id_opt

JD_3 = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt, vds=VDS, l=l_opt)
W_3 = ID_3 / JD_3
print(f"ID: {ID_3}")
VGS_3 = look_up_vgs_vs_gm_id(nch, gm_id_opt, l=l_opt, vds=VDS)

print(f"Intrinsic Gain: {intrinsic_gain_3}")
print(f"fT: {fT_3} Hz")
print(f"Width (W): {W_3} m")
print(f"Required VGS: {VGS_3} V")
```

- **Cadence simulation**
  Gain4 is for exercise3





- **Analysis of the results**
  The parameters designed in this exercise are simulated, and the simulation result is the Gain4 curve. It can be seen that the Gain4 design is significantly higher than other designs in the 20dB curve. The specific performance results comparison is shown in the table below. It can be found that the gain and cutoff frequency have been improved compared to the previous design.



Performance	Exercise3 Design	Exercise2 Design1	Exercise2 Design2
Gain(dB)	26.17	24.74	22.92
Gain bandwidth(Hz)	548M	501M	470M
$gm_{ID}(S/A)$	12.94	5.17	4.81
$VGS(V)$	0.535	0.818	0.788

Exercise 4

- **Design and Script**

For exercise 4, the design of lab1 must be re-simulated first, because the device is replaced with the original part in gpdk045. Regarding the function design part, we need to design according to the conditions of known fufirst, and then encapsulate it into a function.

The specific script implementation is consistent with the idea of exericse3. First, find the ft that meets the fu condition and its corresponding Av through the loop of the gmid list and the L list. By selecting the L corresponding to the maximum Av, the various parameters of the component are further determined. The specific The implementation steps remain the same as the previous exercise.

Script:

```
1 def lab1(fu, CL=1e-12, gm_id_range=np.linspace(5, 30, 50),
    vds_range=np.linspace(0.35, 1.05, 200)):
2     nch = importdata('nch_1v.mat')
3     L = nch['L']
```

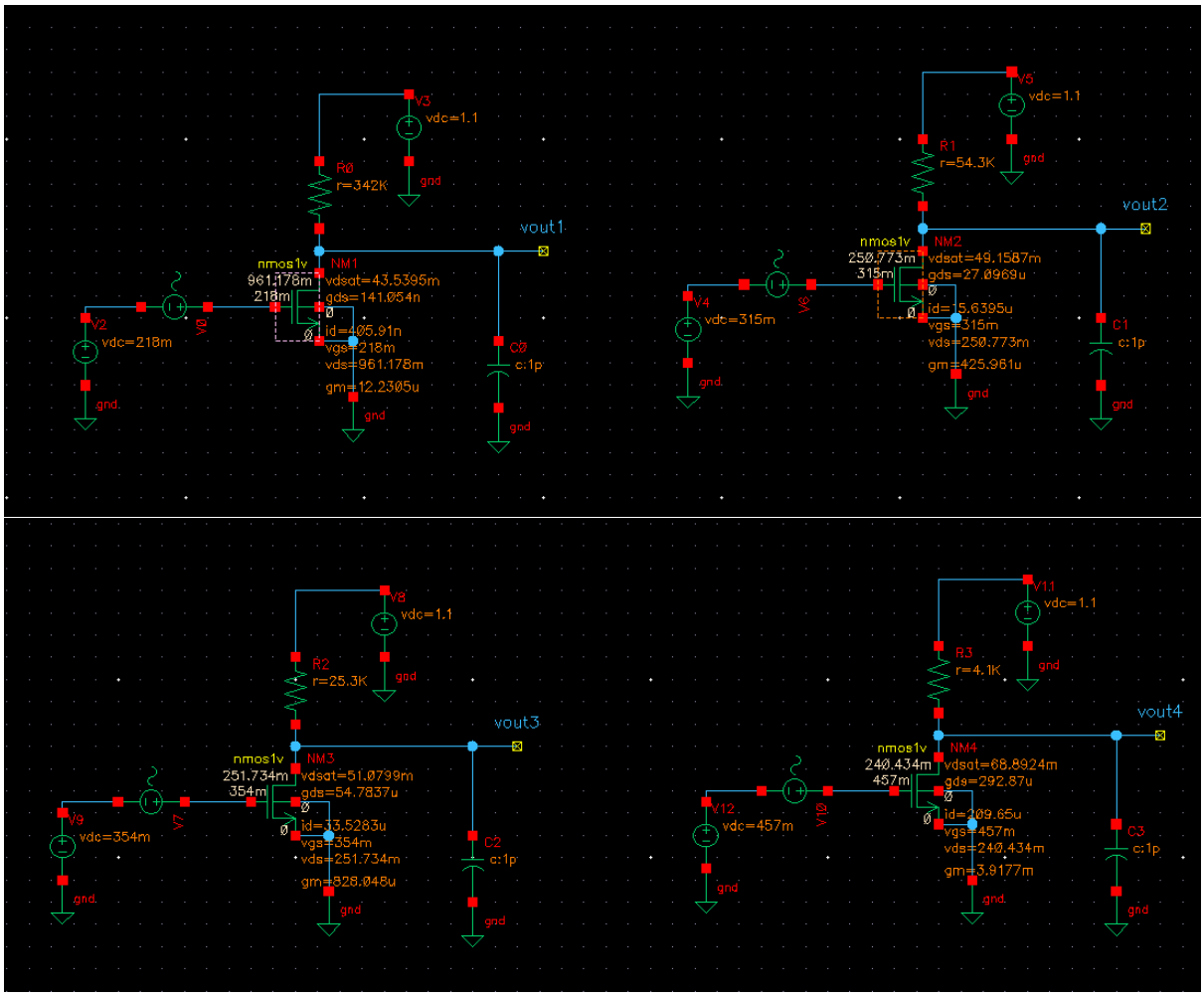
```
    av_matrix = np.zeros((len(gm_id_range), len(L)))
    vds_opt_matrix = np.zeros((len(gm_id_range), len(L)))

    for i, gm_id in enumerate(gm_id_range):
        for j, l in enumerate(L):
            ft = look_up_vs_gm_id(nch, 'GM_CGG', gm_id, l=l) / (2
                * np.pi)
            if ft < 10 * fu:
                av_matrix[i, j] = float('nan')
                vds_opt_matrix[i, j] = float('nan')
                continue

            av_values = []
            for vds in vds_range:
                gds_id = look_up_vs_gm_id(nch, 'GDS_ID', gm_id,
                    vds=vds, l=l)
                av = gm_id / (gds_id + 1/(1.1-vds))
                av_values.append(av)

            max_av_index = np.argmax(av_values)
            av_matrix[i, j] = av_values[max_av_index]
            vds_opt_matrix[i, j] = vds_range[max_av_index]

    max_indices = np.unravel_index(np.nanargmax(av_matrix),
        av_matrix.shape)
    max_gm_id_index, max_l_index = max_indices
    gm_id_opt = gm_id_range[max_gm_id_index]
    l_opt = L[max_l_index]
    vds_opt = vds_opt_matrix[max_indices]

    jd = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt, vds=vds_opt,
        l=l_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(1, 10):
        gm_opt = 2 * np.pi * fu * (CL + cdd)
        ID = gm_opt / gm_id_opt
        W = ID / jd
        cdd = W * cdd_w

    result = {
        'fu': fu,
        'cdd': cdd,
        'gm_opt': gm_opt,
        'gm_id_opt': gm_id_opt,
        'l_opt': l_opt,
        'vds_opt': vds_opt,
        'ID': ID,
        'VGS': look_up_vgs_vs_gm_id(nch, gm_id_opt, l=l_opt,
            vds=vds_opt),
        'RD': (1.1 - vds_opt) / ID,
        'W': W
    }
```

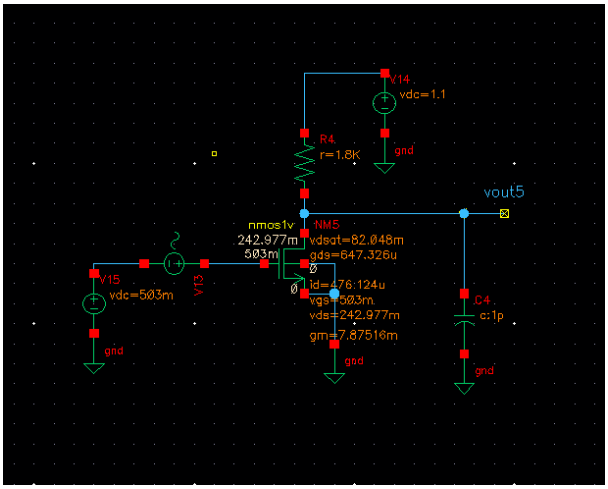
```
54
55      return result
```

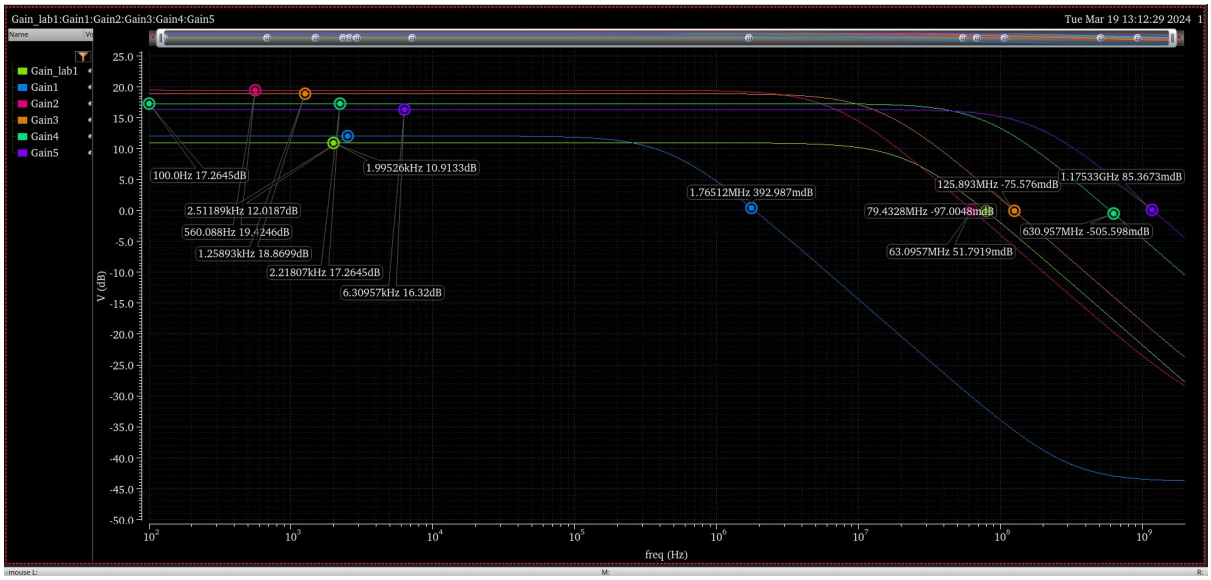
The following is a test of the lab1 function script based on the fu list given in the question, and the return value is obtained.

```
fu_list = [1e6, 50e6, 100e6, 500e6, 1e9]
results = [lab1(fu) for fu in fu_list]

for result in results:
    print(f"fu: {result['fu']/1e6} MHz")
    print(f"cdd: {result['cdd']}")
    print(f"gm_opt: {result['gm_opt']}")
    print(f"gm_id_opt: {result['gm_id_opt']}")
    print(f"l_opt: {result['l_opt']}")
    print(f"vds_opt: {result['vds_opt']}")
    print(f"ID: {result['ID']}")
    print(f"VGS: {result['VGS']}")
    print(f"RD: {result['RD']}")
    print(f"W:{result['W']}")
```

- **Cadence simulation**
  Gain1-Gain5 are the five frequencies required by the question. We use the parameters obtained from the script to simulate the circuit diagram.







- **Analysis of the results**

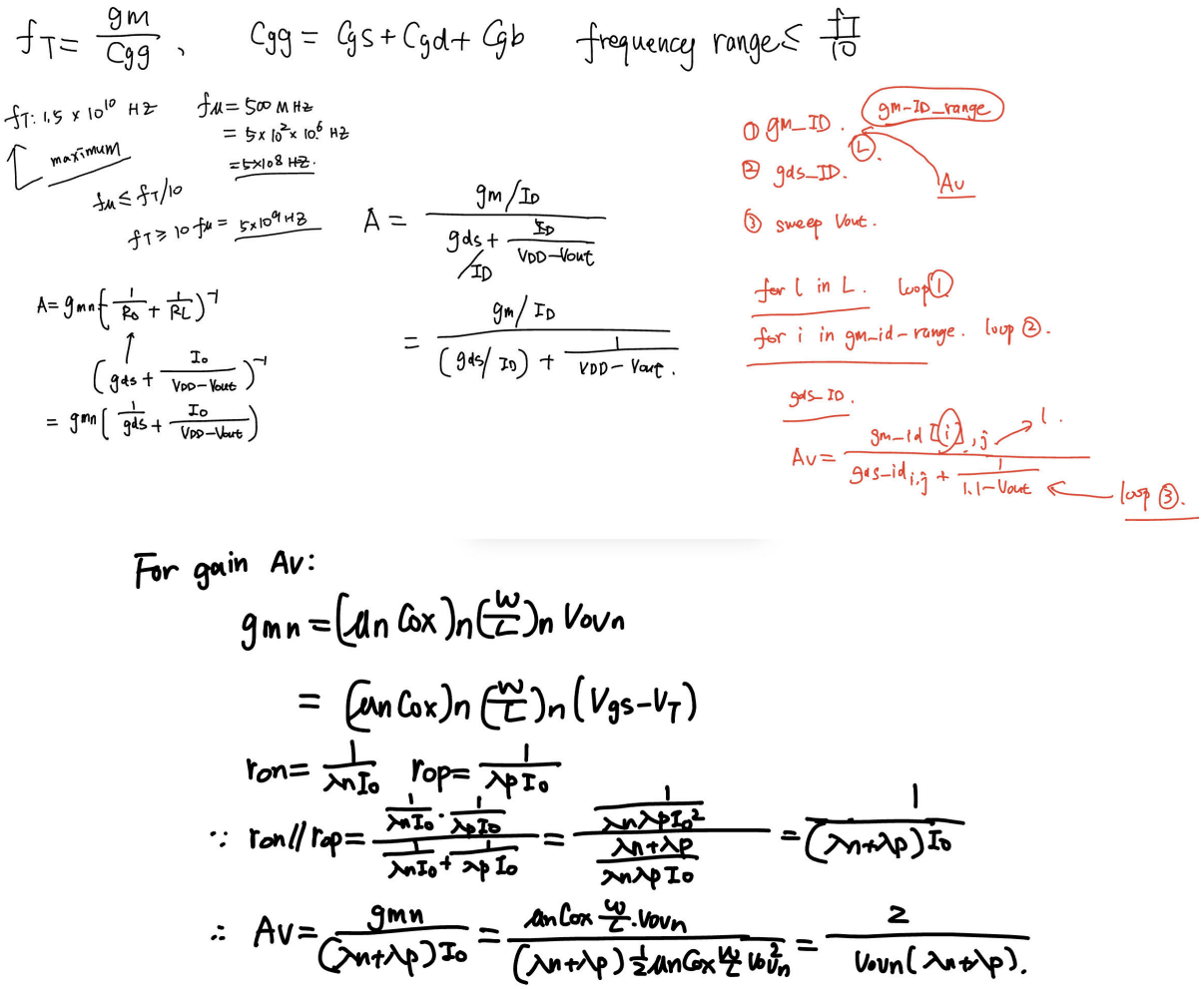
  The simulation results obtained according to the requirements are drawn into a table, and the performance of each result is displayed as follows:

Parameter	**1 MHz**	**50 MHz**	**100 MHz**	**500 MHz**	**1 GHz**
$VGS(V)$	218m	315m	354m	457m	503m
$RD(\Omega)$	342k	54.3k	25.3k	4.1k	1.8k
$W(\mu m)$	27.66	144.55	115.82	83.71	86.38
$L(\mu nm)$	1.0	0.14	0.14	0.14	0.12
$I(A)$	405.91n	15.64u	33.53u	209.65u	472.16u
$gm/ID(S/A)$	30.19	27.24	24.69	18.69	16.54
Gain (dB)	12.02	19.42	18.87	17.26	16.32
Gain bandwidth (Hz)	1.77M	63.10M	125.89M	630.96M	1.17G

Exercise 5

- **Design and Script**
  For the redesign of lab2, we first use the same logic script code to find the corresponding optimal parameters as before.

  My draft derivation is as follows. As mentioned in the lab, first the expression of Av is given, which is related to the three controllable parameters of gmid, gdsid and vout, where vout is the vds of nmos. So we can control The range of vout, construct an outermost loop for vout to find the maximum Av. Inside this loop, we still use the logic from our previous exercise to draw the relationship between Av and L for gmid and gdsid. After finding this L, find the corresponding parameters through lookup. Another thing to note is that PMOS and NMOS appear at the same time in this exercise, so the search for their parameters needs to be done separately. Here, first set up the PMOS cycle, then find the PMOS parameters and vgs that maximize the gain, and then find the NMOS parameters.



Script

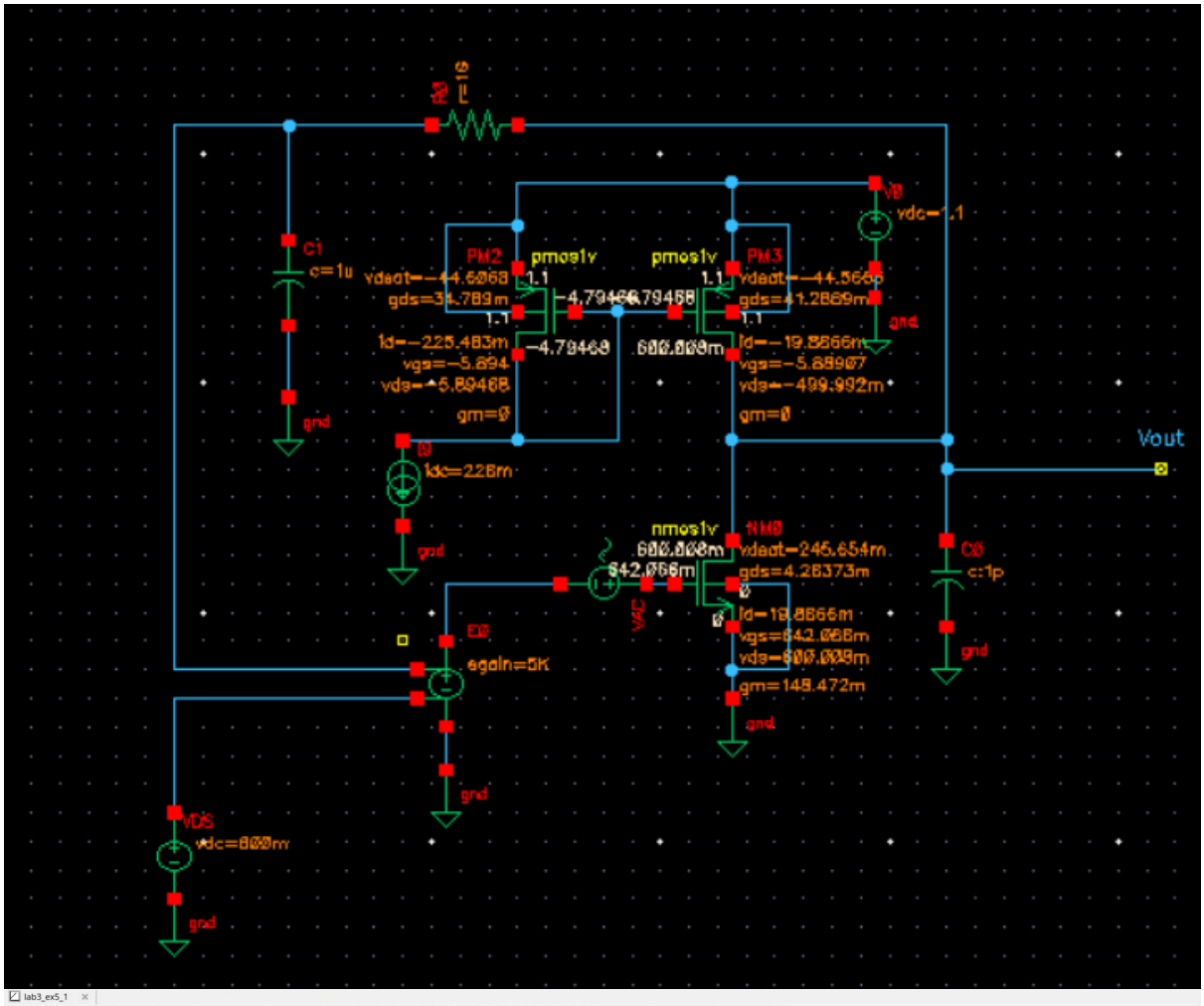
```
    import numpy as np
import matplotlib.pyplot as plt
from look_up import *
```

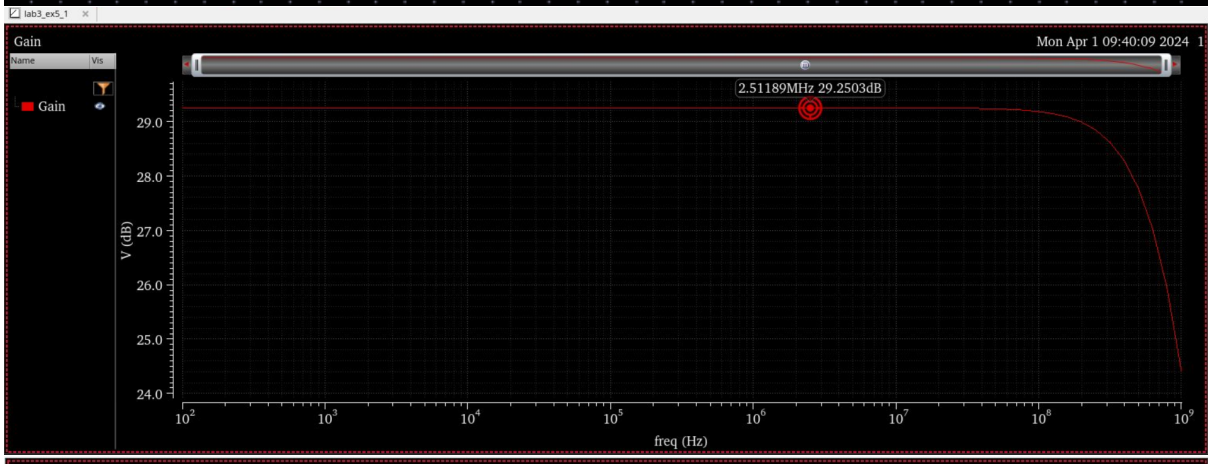
```
fu = 1e9
CL = 1e-12
VDD = 1.1
Vout_range = np.linspace(0.1, 1.0, 100)

pch = importdata('pch_1v.mat')
nch = importdata('nch_1v.mat')
Lp = pch['L']
Ln = nch['L']

best_av_max = -np.inf
best_results = None

for Vout in Vout_range:
    VGS_p = VDS_p = VDD - Vout

    best_av_p = -np.inf
    best_pmos = None

    for l_p in Lp:
        gm_id_range = np.linspace(5, 30, 50)
        ft_p = look_up_vs_gm_id(pch, 'GM_CGG', gm_id_range,
            vds=VDS_p, l=l_p) / (2 * np.pi)
        m = ft_p >= fu

        if np.any(m):
            gm_id_p = gm_id_range[np.max(np.where(m))]
            gds_id_p = look_up_vs_gm_id(pch, 'GDS_ID', gm_id_p,
                vds=VDS_p, l=l_p)

            best_av_n = -np.inf
            best_nmos = None

            for l_n in Ln:
                gm_id_range = np.linspace(5, 30, 50)
                ft_n = look_up_vs_gm_id(nch, 'GM_CGG',
                    gm_id_range, vds=Vout, l=l_n) / (2 * np.pi)
                m = ft_n >= fu

                if np.any(m):
                    gm_id_n = gm_id_range[np.max(np.where(m))]
                    gds_id_n = look_up_vs_gm_id(nch, 'GDS_ID',
                        gm_id_n, vds=Vout, l=l_n)
                    gm_n = look_up_vs_gm_id(nch, 'GM', gm_id_n,
                        vds=Vout, l=l_n)
                    RL = 1 / ((VDD - Vout) / (gm_id_n / gm_n))
                    av = gm_id_n / (gds_id_n + gds_id_p + 1/RL)

                    if av > best_av_n:
                        best_av_n = av
                        best_nmos = {
                            'gm_id_n': gm_id_n,
                            'l_n': l_n
                        }
```

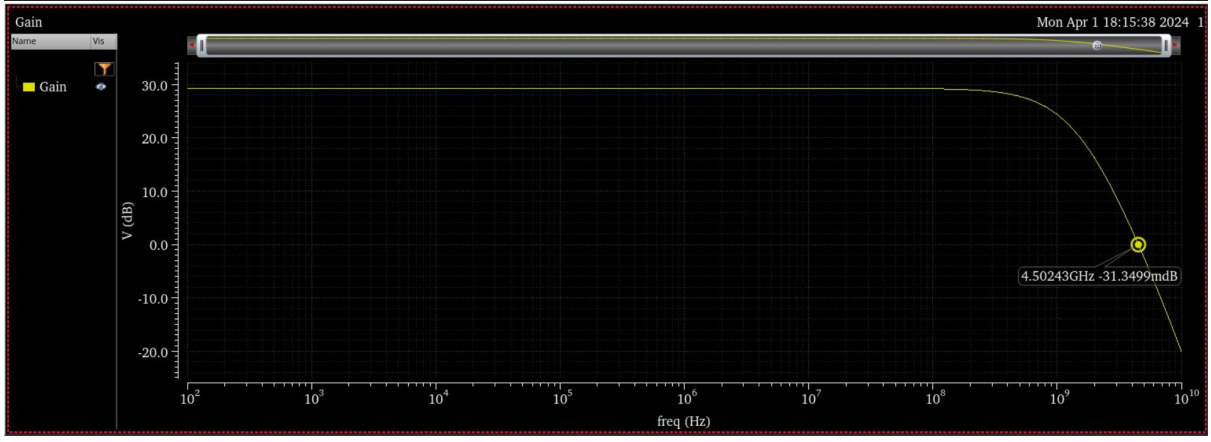
```
            if best_nmos is not None and best_av_n > best_av_p:
                best_av_p = best_av_n
                best_pmos = {
                    'gm_id_p': gm_id_p,
                    'l_p': l_p
                }

    if best_pmos is not None and best_av_p > best_av_max:
        best_av_max = best_av_p
        best_results = {
            'fu': fu,
            'gm_id_opt_p': best_pmos['gm_id_p'],
            'gm_id_opt_n': best_nmos['gm_id_n'],
            'l_opt_p': best_pmos['l_p'],
            'l_opt_n': best_nmos['l_n'],
            'vout_opt': Vout,
            'VGS_p': VGS_p,
            'VDS_p': VDS_p,
            'av_max': best_av_p
        }

if best_results is not None:
    gm_id_opt_p = best_results['gm_id_opt_p']
    gm_id_opt_n = best_results['gm_id_opt_n']
    l_opt_p = best_results['l_opt_p']
    l_opt_n = best_results['l_opt_n']
    vout_opt = best_results['vout_opt']
    VGS_p = best_results['VGS_p']
    VDS_p = best_results['VDS_p']
    VGS_n = vout_opt

    jd_n = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt_n,
       vds=vout_opt, l=l_opt_n)
    jd_p = look_up_vs_gm_id(pch, 'ID_W', gm_id_opt_p, vds=VDS_p,
       l=l_opt_p)

    cdd_w_n = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt_n,
       vds=vout_opt, l=l_opt_n)
    cdd_w_p = look_up_vs_gm_id(pch, 'CDD_W', gm_id_opt_p,
       vds=VDS_p, l=l_opt_p)

    cdd_n = cdd_p = 0
    for i in range(1, 10):
        gm_opt_n = 2 * np.pi * fu * (CL/2 + cdd_n)
        gm_opt_p = 2 * np.pi * fu * (CL/2 + cdd_p)
        ID = gm_opt_n / gm_id_opt_n
        W_n = ID / jd_n
        W_p = ID / jd_p
        cdd_n = W_n * cdd_w_n
        cdd_p = W_p * cdd_w_p

    best_results['cdd_n'] = cdd_n
    best_results['cdd_p'] = cdd_p
    best_results['gm_n'] = gm_opt_n
    best_results['gm_p'] = gm_opt_p
    best_results['VGS_n'] = VGS_n
```

```
        best_results['ID'] = ID
        best_results['W_n'] = W_n
        best_results['W_p'] = W_p

        for key, value in best_results.items():
            print(f"{key}: {value}")
    else:
        print("No valid results found.")
```

- **Cadence simulation**


R0
1k
C1
c=1u
gnd
PM2
pmos1v
vdsat=-44.6068
gds=31.789m
Id=-225.483m
vgs=-5.894
vds=-5.89468
gm=0
PM3
pmos1v
vdsat=-44.568
gds=41.2869m
Id=-19.8866m
vgs=-5.88907
vds=-499.992m
gm=0
V0
vdc=1.1
Vout
I0
Idc=226m
nmos1v
NM0
vdsat=245.654m
gds=4.28373m
Id=19.8866m
vgs=642.068m
gm=148.472m
C0
c=1p
E0
egain=5K
VAC
VDS
vdc=800m


Gain
Mon Apr 1 09:40:09 2024
Name
Vis
Gain
2.51189MHz 29.2503dB
V (dB)
freq (Hz)


Gain
Mon Apr 1 18:15:38 2024
Name
Vis
Gain
4.50243GHz -31.3499mdB
V (dB)
freq (Hz)

- **Analysis of the results**

  For hand calculation, the gain can be calculated directly using this formula: Av=gmn(ron//rop).

  In lab2, the results obtained by manual calculation and experiment are not much different, but in exercise 4, the simulation results obtained are obviously very different from lab2. This is because the devices used are different and the parameters set are completely different.

Performance	Handout result in lab2	Simulation result for ex4
Gain(dB)	44.372	29.25
Gain bandwidth(Hz)	1.95M	4.5G

Exercise 6

- **Design and Script**

  For ex6 design and script improvement:

- For NMOS, we use `VGS_n` = `VICM` as the gate voltage since the source is now connected to the tail current source.
- When calculating `RL`, we use `RL` = $\frac{1}{(\text{VDD}-\text{Vout})/(2*\text{gm\_id\_n}/\text{gm\_n})}$ because in differential mode, the load resistance is twice the single-ended value.
- When calculating the gain `av`, we use `av` = $\frac{\text{gm\_id\_n}}{2*(\text{gds\_id\_n}+\text{gds\_id\_p})+1/\text{RL}}$, because in differential mode, the transconductance is twice the single-ended value, and the output conductance is the single-ended value doubled.
- In the `best_results` dictionary, I added `'VICM_opt': VICM` and `'VGS_n': VGS_n` key-value pairs to store the best input common mode voltage and gate voltage of the NMOS.

  Script:

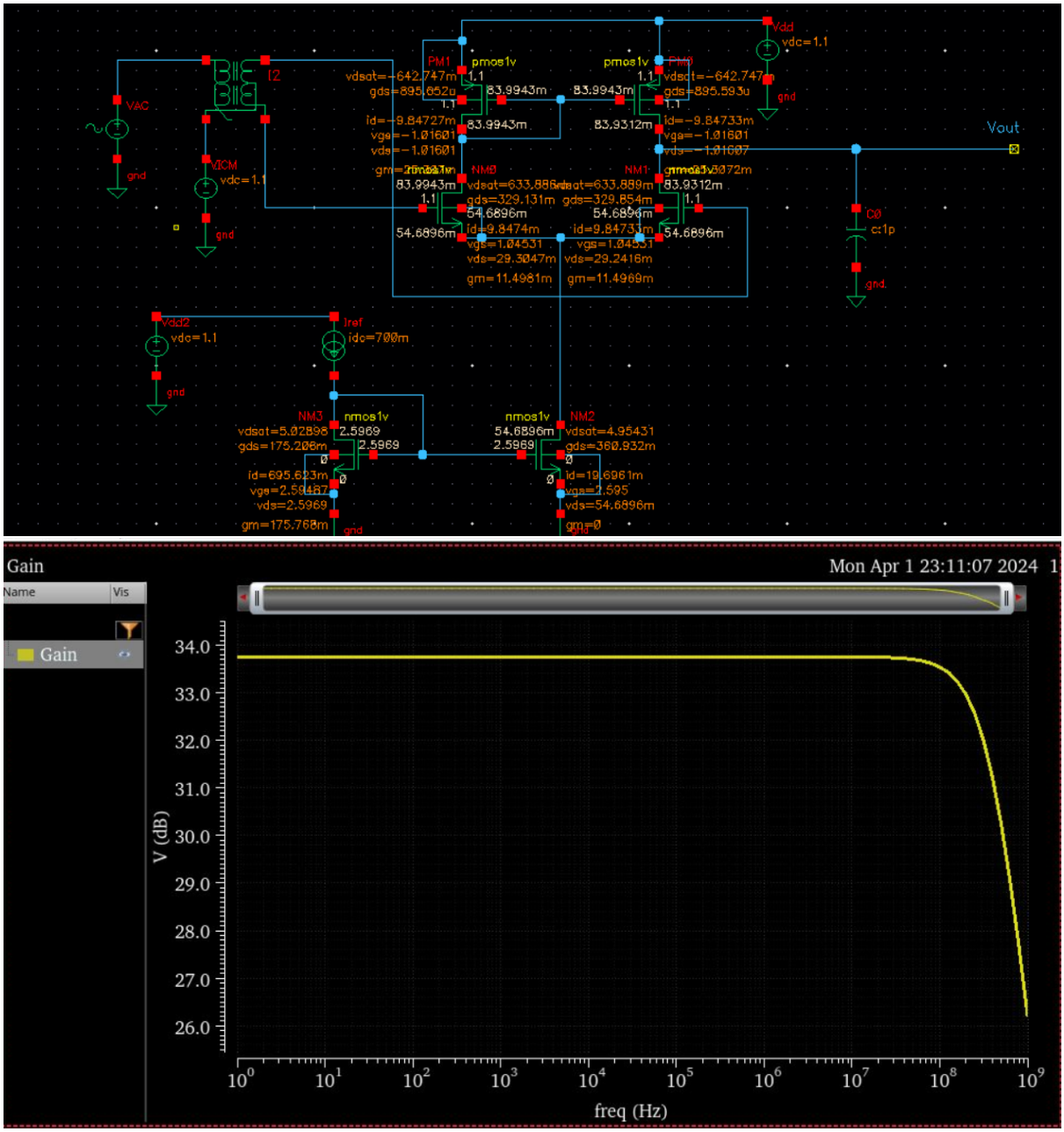
```
import numpy as np
import matplotlib.pyplot as plt
from look_up import *

fu = 1e9
CL = 1e-12
VDD = 1.1
Vout_range = np.linspace(0.1, 1.0, 100)
VICM = 0.6  #Set a suitable input common mode voltage

pch = importdata('pch_1v.mat')
nch = importdata('nch_1v.mat')
Lp = pch['L']
Ln = nch['L']

# Step 1: Search for the optimal parameters of the differential
    amplifier circuit
best_av_max = -np.inf
best_results = None

for Vout in Vout_range:
    VGS_p = VDS_p = VDD - Vout
    VGS_n = VICM

```

```
    best_av_p = -np.inf
    best_pmos = None

    for l_p in Lp:
        gm_id_range = np.linspace(5, 30, 50)
        ft_p = look_up_vs_gm_id(pch, 'GM_CGG', gm_id_range,
            vds=VDS_p, l=l_p) / (2 * np.pi)
        m = ft_p >= fu

        if np.any(m):
            gm_id_p = gm_id_range[np.max(np.where(m))]
            gds_id_p = look_up_vs_gm_id(pch, 'GDS_ID', gm_id_p,
                vds=VDS_p, l=l_p)

            best_av_n = -np.inf
            best_nmos = None

            for l_n in Ln:
                gm_id_range = np.linspace(5, 30, 50)
                ft_n = look_up_vs_gm_id(nch, 'GM_CGG',
                    gm_id_range, vds=VGS_n, l=l_n) / (2 * np.pi)
                m = ft_n >= fu

                if np.any(m):
                    gm_id_n = gm_id_range[np.max(np.where(m))]
                    gds_id_n = look_up_vs_gm_id(nch, 'GDS_ID',
                        gm_id_n, vds=VGS_n, l=l_n)
                    gm_n = look_up_vs_gm_id(nch, 'GM', gm_id_n,
                        vds=VGS_n, l=l_n)
                    RL = 1 / ((VDD - Vout) / (2 * gm_id_n / gm_n))
                    av = gm_id_n / (2 * (gds_id_n + gds_id_p) +
                        1/RL)

                    if av > best_av_n:
                        best_av_n = av
                        best_nmos = {
                            'gm_id_n': gm_id_n,
                            'l_n': l_n
                        }

            if best_nmos is not None and best_av_n > best_av_p:
                best_av_p = best_av_n
                best_pmos = {
                    'gm_id_p': gm_id_p,
                    'l_p': l_p
                }

    if best_pmos is not None and best_av_p > best_av_max:
        best_av_max = best_av_p
        best_results = {
            'fu': fu,
            'gm_id_opt_p': best_pmos['gm_id_p'],
            'gm_id_opt_n': best_nmos['gm_id_n'],
            'l_opt_p': best_pmos['l_p'],
            'l_opt_n': best_nmos['l_n'],
            'vout_opt': Vout,
```

```
            'VGS_p': VGS_p,
            'VDS_p': VDS_p,
            'VGS_n': VGS_n,
            'av_max': best_av_p
        }

# Step 2: Use the optimal parameters of the differential amplifier circuit to search for the optimal parameters of the current mirror
if best_results is not None:
    gm_id_opt_p = best_results['gm_id_opt_p']
    gm_id_opt_n = best_results['gm_id_opt_n']
    l_opt_p = best_results['l_opt_p']
    l_opt_n = best_results['l_opt_n']
    vout_opt = best_results['vout_opt']
    VGS_p = best_results['VGS_p']
    VDS_p = best_results['VDS_p']
    VGS_n = best_results['VGS_n']

    jd_n = look_up_vs_gm_id(nch, 'ID_W', gm_id_opt_n, vds=VGS_n, l=l_opt_n)
    jd_p = look_up_vs_gm_id(pch, 'ID_W', gm_id_opt_p, vds=VDS_p, l=l_opt_p)

    cdd_w_n = look_up_vs_gm_id(nch, 'CDD_W', gm_id_opt_n, vds=VGS_n, l=l_opt_n)
    cdd_w_p = look_up_vs_gm_id(pch, 'CDD_W', gm_id_opt_p, vds=VDS_p, l=l_opt_p)

    best_IM_max = -np.inf
    best_IM_results = None

    for l_IM in Lp:
        gm_id_range = np.linspace(5, 30, 50)
        ft_IM = look_up_vs_gm_id(pch, 'GM_CGG', gm_id_range, vds=VDS_p, l=l_IM) / (2 * np.pi)
        m = ft_IM >= fu

        if np.any(m):
            gm_id_IM = gm_id_range[np.max(np.where(m))]
            gds_id_IM = look_up_vs_gm_id(pch, 'GDS_ID', gm_id_IM, vds=VDS_p, l=l_IM)
            IM_av = gm_id_IM / gds_id_IM

            if IM_av > best_IM_max:
                best_IM_max = IM_av
                best_IM_results = {
                    'gm_id_IM': gm_id_IM,
                    'l_IM': l_IM,
                    'IM_av': IM_av
                }

    if best_IM_results is not None:
        gm_id_IM = best_IM_results['gm_id_IM']
        l_IM = best_IM_results['l_IM']
        IM_av = best_IM_results['IM_av']
```

```
122
123         jd_IM = look_up_vs_gm_id(pch, 'ID_W', gm_id_IM, vds=VDS_p,
                l=l_IM)
124         cdd_w_IM = look_up_vs_gm_id(pch, 'CDD_W', gm_id_IM,
                vds=VDS_p, l=l_IM)
125
126         cdd_n = cdd_p = cdd_IM = 0
127         for i in range(1, 10):
128             gm_opt_n = 2 * np.pi * fu * (CL/2 + cdd_n)
129             gm_opt_p = 2 * np.pi * fu * (CL/2 + cdd_p)
130             ID = gm_opt_n / gm_id_opt_n
131             W_n = ID / jd_n
132             W_p = ID / jd_p
133             W_IM = ID / jd_IM
134             cdd_n = W_n * cdd_w_n
135             cdd_p = W_p * cdd_w_p
136             cdd_IM = W_IM * cdd_w_IM
137
138         best_results['cdd_n'] = cdd_n
139         best_results['cdd_p'] = cdd_p
140         best_results['cdd_IM'] = cdd_IM
141         best_results['gm_n'] = gm_opt_n
142         best_results['gm_p'] = gm_opt_p
143         best_results['ID'] = ID
144         best_results['W_n'] = W_n
145         best_results['W_p'] = W_p
146         best_results['W_IM'] = W_IM
147         best_results['gm_id_IM'] = gm_id_IM
148         best_results['l_IM'] = l_IM
149         best_results['IM_av'] = IM_av
150
151         for key, value in best_results.items():
152             print(f"{key}: {value}")
153     else:
154         print("No valid results found for current mirror.")
155 else:
156     print("No valid results found for differential amplifier.")
```

- **Cadence simulation**



- **Analysis of the results**

  This is a simple design, and better implementation will be improved in the project. Due to the time limit of the experimental report, the simulation has not been perfectly realized, that is, it has not yet reached the computational requirements.