EENG304L Lab 4 Report

February 22, 2025

1 MOSFET Linear Amplifier Analysis

By: Gabriel DiMartino, Michael Baker, Elijah Chin

1.1 Initial Parameters

Parameter	Purpose	Value
W	Channel Width	40 μm
L	Channel Length	1.6 μm
K_n'	Process Transconductance	50 μA/V^2
V_{to}	Threshold Voltage	0.5 V
C_{ox}	Gate Oxide Capacitance	2.3 mF/m^2
λ	Channel Length Modulation	0.0625 V^{-1}
γ	Body Effect Parameter	0.6 V$^{1/2}$
ϕ	Surface Potential	0.8 V

1.2 Circuit Conditions

Parameter	Purpose	Value
I_{BB}	DC Bias Current	250 μA
V_{DD}	DC Supply Voltage	5 V
V_{BB}	DC Bias Voltage	2.5 V
R	Resistance	10 kΩ

Before executing any code, make sure to have all necesarry packages installed, This can be run from within using the "Run All" button.

[11]:
```
%pip install -q numpy matplotlib PyLTSpice==3.1.0 pandas;
```

```
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
W = 40e-6
L = 1.6e-6
KN_PRIME = 50e-6
V_TO = 0.5
C_OX = 2.3e-3

I_BB = 250e-6
V_DD = 5
```

```
V_BB = 2.5
V_DS = 2.5
R = 10e3
RI = 20e3
```

1.3 Equations

1.3.1 Channel Length Modulation Parameter

$$\lambda(L) = \frac{0.1\mu m \cdot V^{-1}}{L}$$

1.3.2 MOSFET Saturation Region Current

$$I_D = \frac{1}{2}K_n'\frac{W}{L}(V_{GS} - V_{to})^2(1 + \lambda V_{DS})$$

Where: - I_D is biased by I_{BB} at 250 A - We need to solve for V_{GS}

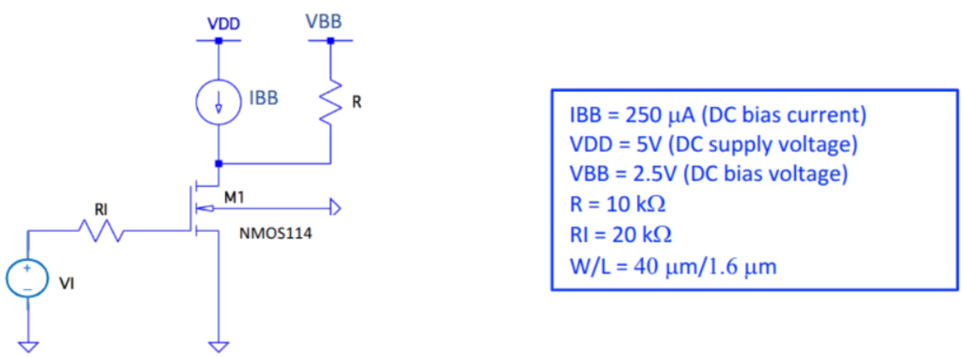


Figure 1.

1.4 Calculations

1.4.1 Solving for Channel Length Modulation

[2]:
```
lambda_val = 0.1e-6/L
print(lambda_val)
```

```
0.0625
```

1.4.2 Threshold Voltage (V_{TH})

1.4.3 Given Parameters

Parameter	Purpose	Value
V_{to}	Threshold Voltage	0.5 V
γ (Gamma)	Body Effect Parameter	0.6 $V^{1/2}$
ϕ (phi)	Surface Potential	0.8 V
V_{SB}	Source-Bulk Voltage	0 V

1.4.4 Circuit Analysis

Since both the Source and Bulk are connected to ground: - Source voltage = 0V - Bulk voltage = 0V - Therefore, $V_{SB} = V_S - V_B = 0$ V

Similarly, to solve the Drain and Bulk - Drain voltage = 2.5V - Bulk voltage = 0V - Therefore, $V_{DB} = V_D - V_B = 2.5$ V

1.4.5 Threshold Voltage Equation

$$V_{TH} = V_{to} + \gamma(\sqrt{\phi + V_{SB}} - \sqrt{\phi})$$

[3]:
```python
import math

GAMMA = 0.6     # V^(1/2)
PHI = 0.8       # V
V_SB = 0        # V
V_DB = 2.5      # V

# Calculate Vth
Vth = V_TO + GAMMA * (math.sqrt(PHI + V_SB) - math.sqrt(PHI))

print(f"VTH = {Vth:.1f} V")
```

```
VTH = 0.5 V
```

Given that $\lambda = 0.0625$ V^{-1}, and $V_{TH} = 0.5V$ we can solve for V_{GS} using the saturation region equation:

1.4.6 Solving for V_{GS} using the saturation region equation

[4]:
```python
# Calculate beta term
beta = KN_PRIME * (W/L)
Id = I_BB

# Solve for Vgs using saturation equation
# ID = (1/2) * beta * (Vgs - Vto)^2 * (1 + lambda * Vds)
term1 = 2 * Id / (beta * (1 + lambda_val * V_DS))
Vgs = math.sqrt(term1) + Vth
```

```python
print(f"VGS = {Vgs:.3f} V")
```

```
VGS = 1.088 V
```

1.4.7 Small Signal Parameters Calculation

Given Values:

Parameter	Purpose	Value
V_{GS}	Gate-Source Voltage	1.088 V
V_{TH}	Threshold Voltage	0.5 V

Small Signal Equations Transconductance:

$$g_m = \frac{2I_D}{V_{GS} - V_{TH}}$$

Output Conductance:

$$g_{ds} = \lambda \cdot I_D$$

Gate-to-Source Capacitance:

$$C_{gs} = \frac{2}{3} \cdot C_{ox} \cdot W \cdot L$$

[5]:
```python
# Calculate gm
gm = (2 * Id) / (Vgs - Vth)

# Calculate gds
gds = lambda_val * Id

# Calculate Cgs
Cgs = (2/3) * C_OX * W * L

print(f"gm = {gm*1e3:.3f} mS")
print(f"gds = {gds*1e6:.3f} µS")
print(f"Cgs = {Cgs*1e15:.3f} fF")
```

```
gm = 0.850 mS
gds = 15.625 µS
Cgs = 98.133 fF
```

As expected, the the analysis of the MOSFET parameters yielded the following results:

Parameter	Purpose	Value
g_m	Transconductance	850 μS
g_{ds}	Output Conductance	15.625 μS
C_{gs}	Gate-Source Capacitance	98.133 fF

1.5 Capacitance Analysis

1.5.1 Device Parameters

The following parameters are required for extrinsic capacitance calculations:

Parameter	Purpose	Value
CGDO, CGSO	Gate-Drain/Source Overlap Capacitance	0.5 fF/μm
CJ	Zero-Bias Area Capacitance	0.1 fF/μm^2
CJSW	Zero-Bias Sidewall Capacitance	0.5 fF/μm
PB	Junction Potential	0.95 V
PBSW	Junction Potential Sidewall	0.95 V
MJ	Area Junction Grading Coefficient	0.5
MJSW	Sidewall Junction Grading Coefficient	0.33
HDIF	Half Length of S/D Diffusion	1.5 μm

The following equations are used to determine the intermediate values:

1. Diffusion Length:
   - $L_{DIF} = 2 \times HDIF = 2 \times 1.5\ \mu\text{m} = 3\ \mu\text{m}$
2. Source/Drain Areas:
   - $A_d = A_s = 2 \times HDIF \times W = 2 \times 1.5\ \mu\text{m} \times 40\ \mu\text{m} = 120\ \mu\text{m}^2$
3. Source/Drain Perimeters:
   - $P_d = P_s = 2 \times L_{DIF} + W = 2 \times 3\ \mu\text{m} + 40\ \mu\text{m} = 46\ \mu\text{m}$

[6]:
```python
HDIF = 1.5e-6  # m
CGDO = CGSO = 2e-14  # F/m
CJ = 0.1e-15 * 1e12  # F/m²
CJSW = 0.5e-15 * 1e6  # F/m
PB = PBSW = 0.95  # V
MJ = 0.5
MJSW = 0.33

# Calculate intermediate values
LDIF = 2 * HDIF  # meters
Ad = As = 2 * HDIF * W  # Area of drain/source in m²
Pd = Ps = 2 * LDIF + W  # Perimeter of drain/source in m

print(f"LDIF = {LDIF*1e6:.1f}  m")
print(f"Ad = As = {Ad*1e12:.1f}  m²")
print(f"Pd = Ps = {Pd*1e6:.1f}  m")
```

```
LDIF = 3.0  m
Ad = As = 120.0  m²
Pd = Ps = 46.0  m
```

Now knowing that the drain and source areas (A_d, A_s) are both 120 m² and their perimeters (P_d, P_s) are both 46 m, we can solve for the drain-bulk (C_{db}) and source-bulk (C_{sb})

1.5.2 Junction Equations

The drain-bulk and source-bulk junction capacitances are given by:

1. Drain-Bulk Capacitance:

$$C_{db} = \frac{A_D \cdot CJ}{(1 + \frac{V_{DB}}{PB})^{MJ}} + \frac{P_D \cdot CJSW}{(1 + \frac{V_{DB}}{PBSW})^{MJSW}}$$

2. Source-Bulk Capacitance:

$$C_{sb} = \frac{A_S \cdot CJ}{(1 + \frac{V_{SB}}{PB})^{MJ}} + \frac{P_S \cdot CJSW}{(1 + \frac{V_{SB}}{PBSW})^{MJSW}}$$

1.5.3 Known Values

Parameter	Purpose	Value
A_D, A_S	Drain/Source Area	120 μm^2
P_D, P_S	Drain/Source Perimeter	46 μm
V_{DB}	Drain-Bulk Voltage	2.5V
V_{SB}	Source-Bulk Voltage	0 V

[7]:
```
# Calculate source-bulk capacitance components
Csb_area = As * CJ / (1 + V_SB/PB)**MJ
Csb_perim = Ps * CJSW / (1 + V_SB/PBSW)**MJSW
Csb = Csb_area + Csb_perim

# Calculate drain-bulk capacitance components
Cdb_area = Ad * CJ / (1 + V_DB/PB)**MJ
Cdb_perim = Pd * CJSW / (1 + V_DB/PBSW)**MJSW
Cdb = Cdb_area + Cdb_perim

print("Junction Capacitance Results:")
print("----------------------------")
print(f"Csb = {Csb*1e15:.3f} fF")
print(f"Cdb = {Cdb*1e15:.3f} fF")
```

```
Junction Capacitance Results:
----------------------------
Csb = 35.000 fF
Cdb = 21.325 fF
```

1.6 Amplifier DC Gain and Bandwidth Analysis

After calculating the small-signal parameters and extrinsic capacitances, we can now determine the amplifier's DC gain and bandwidth.

Key equations: - Output resistance: $r_o = \frac{1}{\lambda I_D}$ - DC Gain: $A_v = \frac{V_{out}}{V_{in}} = -g_m(r_o||R_L)$ - Unity gain frequency: $f_T = \frac{g_m}{2\pi C_{total}}$ - Bandwidth (Miller Approximation): $f_{-3dB} = \frac{1}{2\pi r_o(C_{gs}+C_{gd}(1+gmr_o))}$

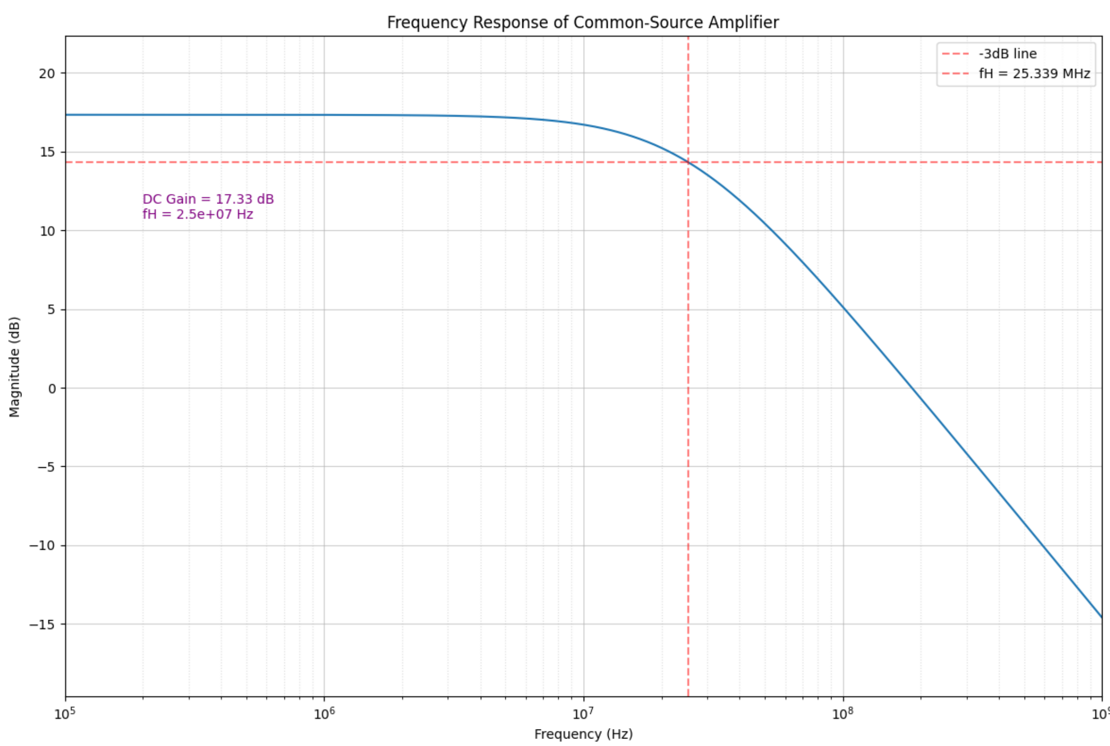
Where: - $C_{total} = C_{db} + C_{gd} + C_{gs} $ - $R_{eq} = r_o||R$

[8]:

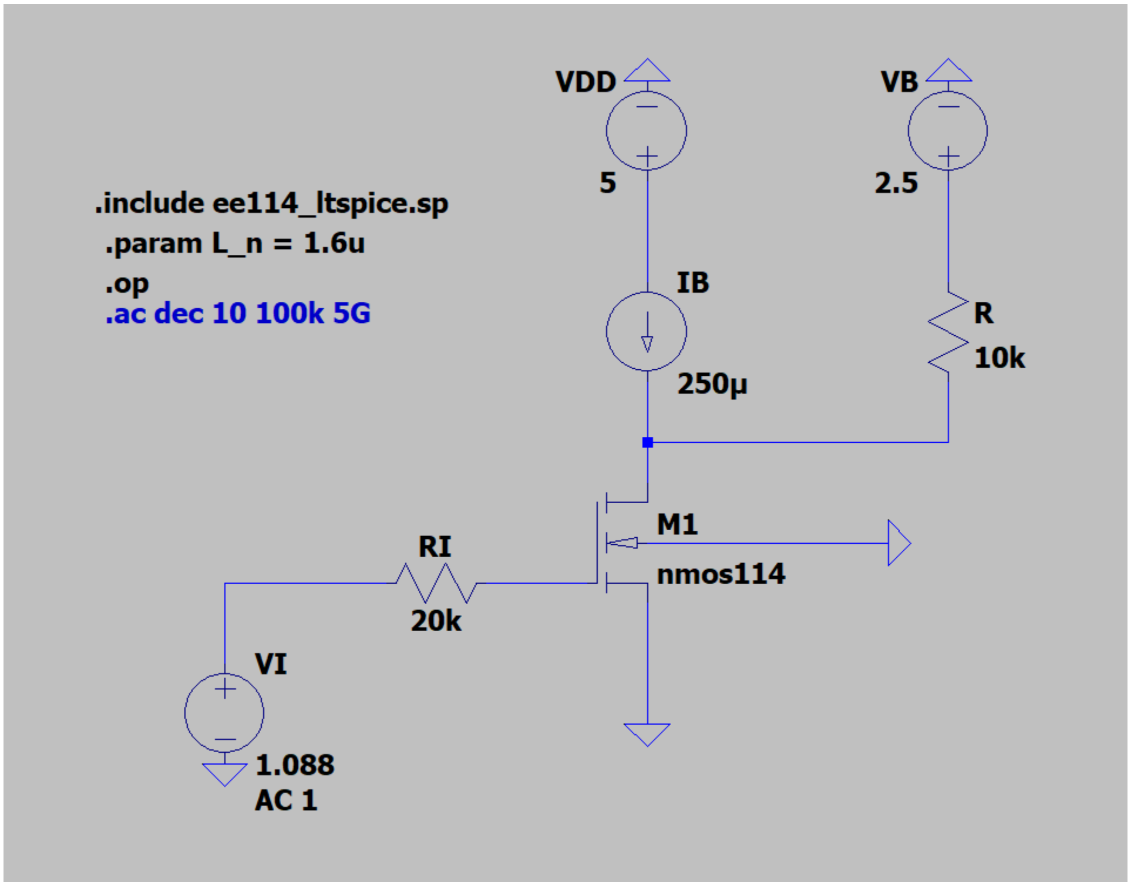
```python
import numpy as np
import matplotlib.pyplot as plt


# Calculate output resistance
ro = 1 / (lambda_val * I_BB)

# Calculate equivalent resistance
R_eq = (ro * R) / (ro + R)

# Calculate DC gain
gain_dc = -gm * R_eq
gain_db = 20 * np.log10(abs(gain_dc))

# Calculate capacitances
Cgd_ov = CGDO * W  # Gate-drain overlap capacitance
Cgd = Cgd_ov       # In saturation, intrinsic Cgd is negligible

C_total = Cdb + Cgd + Cgs  # Total capacitance affecting frequency response

# Updated -3dB bandwidth formula
fH_hand = 1 / (2 * np.pi * ro * (Cgs + Cgd*(1+abs(gain_dc))))  # Cutoff␣
 ↪frequency

# Calculate unity gain frequency
f_T = gm / (2 * np.pi * C_total)

# Print results
print(f"Output resistance (ro) = {ro/1e3:.2f} kΩ")
print(f"DC Gain = {gain_dc:.2f} V/V ({gain_db:.2f} dB)")
print(f"Total output capacitance = {C_total*1e15:.2f} fF")
print(f"-3dB Bandwidth = {fH_hand/1e6:.3f} MHz")
print(f"Unity gain frequency = {f_T/1e6:.2f} MHz")

# Create frequency response plot
f = np.logspace(5, 9, 1000)  # 100kHz to 1GHz
w = 2 * np.pi * f

# Calculate frequency response using transfer function
# H(s) = Av / (1 + s/wp) where wp = 2 * fH
H = gain_dc / (1 + 1j * f/fH_hand)
mag_db = 20 * np.log10(np.abs(H))

plt.figure(figsize=(12, 8))
plt.semilogx(f, mag_db)
plt.grid(True, which='both')
plt.xlabel('Frequency (Hz)')
```

```python
plt.ylabel('Magnitude (dB)')
plt.title('Frequency Response of Common-Source Amplifier')
plt.axhline(y=gain_db - 3, color='r', linestyle='--', alpha=0.5, label='-3dB␣
↪line')
plt.axvline(x=fH_hand, color='r', linestyle='--', alpha=0.5, label=f'fH =␣
↪{fH_hand/1e6:.3f} MHz')
plt.xlim(1e5, 1e9)
plt.ylim(min(mag_db) - 5, gain_db + 5)

# Add annotation
plt.annotate(f'DC Gain = {gain_db:.2f} dB\nfH = {fH_hand:.1e} Hz',
             xy=(2e5, gain_db - 5), ha='left', va='top', color='purple')

plt.legend()
plt.grid(True, which='minor', linestyle=':', alpha=0.4)
plt.grid(True, which='major', linestyle='-', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
Output resistance (ro) = 64.00 kΩ
DC Gain = -7.35 V/V (17.33 dB)
Total output capacitance = 119.46 fF
-3dB Bandwidth = 25.339 MHz
Unity gain frequency = 1132.58 MHz
```



1.7 SPICE SIMULATION ANLYSIS



[9]:

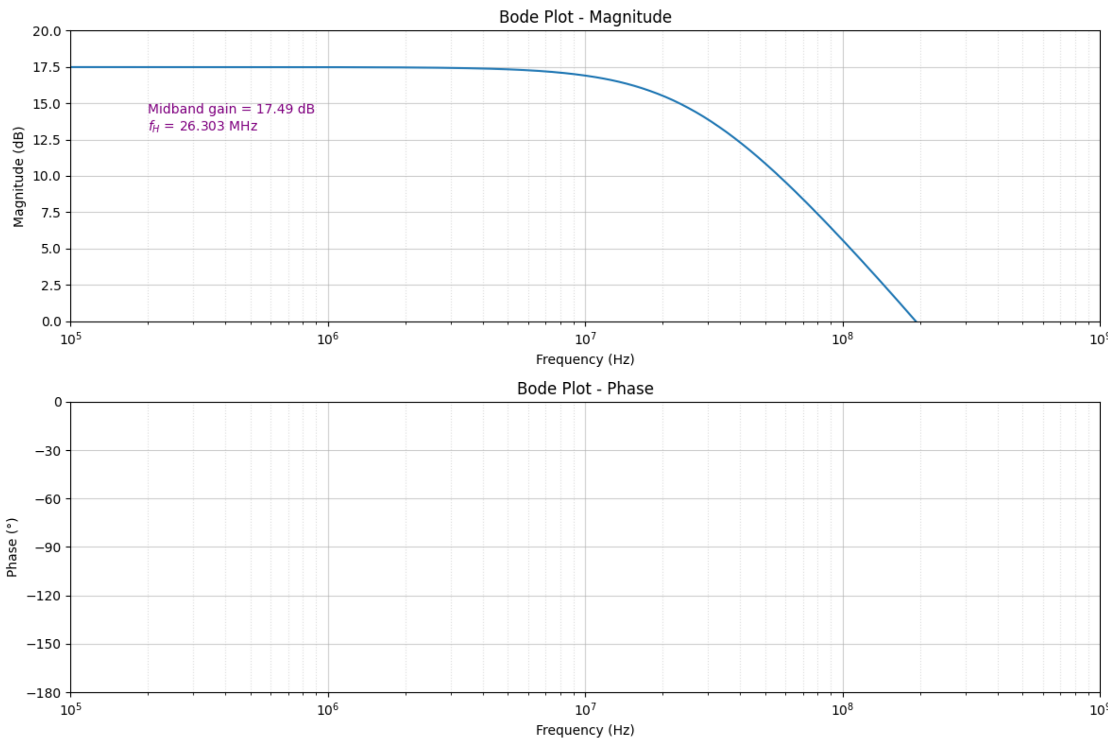
```python
from PyLTSpice import RawRead

# Read the simulation data
LTR = RawRead("./EENG304L Lab 4.raw")

# Get frequency and voltage data
f = LTR.get_trace('frequency')
Vo = LTR.get_trace('V(n003)')
freq = f.get_wave(0)
Vout = np.array(Vo.get_wave(0), dtype=complex)

# Calculate magnitude in dB
mag = 20*np.log10(np.abs(Vout))
gain_db_spice = max(mag)

# Find -3dB frequency (cutoff)
cutoff_indices = np.where(mag <= gain_db_spice-3)
fH_spice = abs(freq[cutoff_indices[0][0]]) / 1e6  # Convert to MHz
```

```python
# Print results
print(f"Midband gain: {gain_db_spice:.2f} dB")
print(f"High frequency cutoff: {fH_spice:.3f} MHz")

# Create Bode plot with adjusted figure size
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# Magnitude plot with improved scaling
ax1.semilogx(freq, mag)
ax1.grid(True, which="both")
ax1.set_title("Bode Plot - Magnitude")
ax1.set_ylabel("Magnitude (dB)")
ax1.set_xlabel("Frequency (Hz)")
ax1.set_xlim(1e5, 1e9)
ax1.set_ylim(0, 20)
ax1.grid(True, which='minor', linestyle=':', alpha=0.4)
ax1.grid(True, which='major', linestyle='-', alpha=0.6)

# Add annotations in a better position
ax1.annotate(f'Midband gain = {gain_db_spice:.2f} dB\n$f_H$ = {fH_spice:.3f}␣
↪MHz',
             xy=(2e5, 15), ha='left', va='top', color='purple')

# Phase plot with improved scaling
phase = np.angle(Vout, deg=True)
ax2.semilogx(freq, phase)
ax2.grid(True, which="both")
ax2.set_title("Bode Plot - Phase")
ax2.set_ylabel("Phase (°)")
ax2.set_xlabel("Frequency (Hz)")
ax2.set_xlim(1e5, 1e9)
ax2.set_ylim(-180, 0)
ax2.set_yticks(np.arange(-180, 15, 30))
ax2.grid(True, which='minor', linestyle=':', alpha=0.4)
ax2.grid(True, which='major', linestyle='-', alpha=0.6)

plt.tight_layout()
plt.show()
```

```
Reading file with encoding  utf_16_le
File contains 18 traces, reading 18
Binary RAW file with Normal access
Midband gain: 17.49 dB
High frequency cutoff: 26.303 MHz
```

```
/home/macbee280/anaconda3/envs/EENG304L/lib/python3.10/site-
packages/matplotlib/cbook.py:1709: ComplexWarning: Casting complex values to
```

```
real discards the imaginary part
  return math.isfinite(val)
/home/macbee280/anaconda3/envs/EENG304L/lib/python3.10/site-
packages/matplotlib/cbook.py:1345: ComplexWarning: Casting complex values to
real discards the imaginary part
  return np.asarray(x, float)
```



From our hand calculations we got a Cdb of 21.325fF which matches our SPICE analysis of Cbd

Name:	m1
Model:	nmos114
Id:	2.50e-04
Vgs:	1.09e+00
Vds:	2.50e+00
Vbs:	0.00e+00
Vth:	5.00e-01
Vdsat:	5.88e-01
Gm:	8.50e-04
Gds:	1.35e-05
Gmb:	2.85e-04
Cbd:	2.13e-14
Cbs:	3.50e-14
Cgsov:	2.00e-14
Cgdov:	2.00e-14
Cgbov:	0.00e+00
Cgs:	9.82e-14
Cgd:	0.00e+00
Cgb:	0.00e+00

1.8 Error

[ ]:
```
import pandas as pd

# Hand Calculations
gain_db_hand = 20 * np.log10(abs(gain_dc))
fH_hand = 1 / (2 * np.pi * ro * (Cgs + Cgd*(1+abs(gain_dc))))
C_total_hand = Cdb + Cgd + Cgs

C_total_spice = (2.13e-14 + 0 + 9.82e-14)

# SPICE Model Values
comparison = pd.DataFrame({
    'Parameter': ['Midband Gain (dB)', 'Cutoff Frequency (MHz)', 'Total␣
 ↪Capacitance (fF)'],
    'Hand Calculation': [gain_db_hand, fH_hand/1e6, C_total_hand*1e15],
    'SPICE': [gain_db_spice, fH_spice/1e6, C_total_spice*1e15]
})

# Calculate errors
comparison['Absolute Error'] = abs(comparison['Hand Calculation'] -␣
 ↪comparison['SPICE'])
comparison['Relative Error (%)'] = (comparison['Absolute Error'] /␣
 ↪comparison['SPICE']) * 100

# Format the numbers to be more readable
pd.options.display.float_format = '{:.3f}'.format

print("Comparison between Hand Calculations and SPICE Results:\n")
```

```python
print(comparison.to_string(index=False))

print("\nCapacitance Breakdown:")
print(f"Hand Calculations:")
print(f"   - Cdb = {Cdb*1e15:.2f} fF")
print(f"   - Cgd = {Cgd*1e15:.2f} fF")
print(f"   - Cgs = {Cgs*1e15:.2f} fF")
print(f"   Total = {C_total_hand*1e15:.2f} fF")
print(f"\nSPICE:")
print(f"   - Cbd = {2.13e-14*1e15:.2f} fF")
print(f"   - Cgd = {0:.2f} fF")
print(f"   - Cgs = {9.82e-14*1e15:.2f} fF")
print(f"   Total = {C_total_spice*1e15:.2f} fF")
```

```
Comparison between Hand Calculations and SPICE Results:

             Parameter  Hand Calculation   SPICE  Absolute Error  Relative Error
(%)
     Midband Gain (dB)            17.328  17.490           0.162
0.925
Cutoff Frequency (MHz)            25.339  26.303           0.964
3.664
Total Capacitance (fF)           119.459 119.500           0.041
0.034

Capacitance Breakdown:
Hand Calculations:
   - Cdb = 21.32 fF
   - Cgd = 0.00 fF
   - Cgs = 98.13 fF
   Total = 119.46 fF

SPICE:
   - Cbd = 21.30 fF
   - Cgd = 0.00 fF
   - Cgs = 98.20 fF
   Total = 119.50 fF
```

1.8.1 Comparison between Hand Calculations and SPICE Results

Parameter	Hand Calculation	SPICE	Absolute Error	Relative Error (%)
Midband Gain (dB)	17.328	17.490	0.162	0.925
Cutoff Frequency (MHz)	25.339	26.303	0.964	3.664
Total Capacitance (fF)	119.459	119.500	0.041	0.034

1.8.2 Capacitance Breakdown

Hand Calculations

- **Cdb** = 21.32 fF
- **Cgd** = 0.00 fF
- **Cgs** = 98.13 fF
- **Total** = 119.46 fF

SPICE

- **Cbd** = 21.30 fF
- **Cgd** = 0.00 fF
- **Cgs** = 98.20 fF
- **Total** = 119.50 fF

1.8.3 Explanation of Discrepancy

The largest discrepancy is in the cutoff frequency, where the hand calculation predicts **25.339 MHz**, whereas SPICE simulation gives **26.303 MHz**. The relative error is **3.664%**, which is significant. The difference is most likely the result of using the Miller Approximation over OCT.

1.8.4 Calculation of SPICE Capacitance

The **total capacitance in SPICE** was obtained by summing the individual capacitances:

`C_total,SPICE = C_bd + C_gd + C_gs`

where: * C__bd = 2.13 × 10^-14 F (21.30 fF) * C__gd = 0.00 F (0.00 fF) * C__gs = 9.82 × 10^-14 F (98.20 fF)

Thus,

`C_total,SPICE = 2.13 × 10^-14 + 0 + 9.82 × 10^-14 = 1.195 × 10^-13 F (119.50 fF)`

This closely matches the hand calculation, confirming that the capacitance assumptions were reasonable.