Assessing the feasibility of developing and taping-out an analog block for LED driver control using IHP-Open-PDK and IIC-OSIC-TOOLS

Priyanka Toyni Hrishikesh Pangavhane

2025-11-19

Table of contents

List of Figures

List of Tables

1. Abstract

Open-source EDA has reached a point where its practical viability for real mixed-signal IC fabrication can be examined on modern semiconductor processes. This thesis evaluates that potential through the design and tape-out of an analog block for LED-driver control using the **IHP OpenPDK SG13G2 a 130 nm SiGe BiCMOS PDK** and the implementation using the **open-source IIC-OSIC-TOOLS** developed by *Prof. Pretl (JKU Linz)*

The project follows a complete analog design cycle. System requirements are derived from an ideal reference schematic of a DC-DC LED-driver PMIC, modeled following the methodology from Prof. Bernhard Wicht's book on design of power managemant ICs. From these constraints, we design an operational transconductance amplifier (OTA) intended to drive a specific load extracted from the system-level PMIC model. The electrical design is implemented in Xschem for schematic capture, Ngspice for pre-layout simulation, corner and Monte-Carlo analysis using CACE, and iterative refinement to ensure gain, UGB, phase margin, and output swing meet the load-driving requirements under process and temperature variation.

The physical design phase is performed entirely in KLayout, following industry-aligned analog layout practices: matching-aware device placement, common-centroid arrays, dummy structures, guard rings, routing strategies, density compliance, pad-frame integration, and the use of IHP-provided seal-ring and ESD p-cells. Parasitics are extracted using KLayout-PEX (KPEX), and post-layout simulations confirm the impact of capacitor/resistor parasitics on stability and dynamic response, validating the design's robustness across PVT.

A key contribution of this work is practical verification: both tape-outs were accepted and fabricated in the **IHP Open MPW run of July 2025**. To strengthen cross-disciplinary understanding, the authors executed two complete tape-outs, one for a design of NMOS foldedcascode structure and other for PMOS foldedcascode structure, switching roles one serving as **design engineer** while the other served as **layout engineer** in the first run, and vice versa in the second. This approach revealed workflow bottlenecks, toolchain strengths, and realistic guidance for analog teams adopting open-source EDA.

The results demonstrate that a fully open-source toolchain, combined with an industry-grade SiGe BiCMOS open PDK, can reliably produce manufacturable analog ICs with predictable post-layout performance. This work serves as a practical reference for engineers, researchers, and SMEs evaluating open-source flows for mixed-signal IC development, and provides evidence that democratized, open analog IC design is not only feasible but tape-out proven.

2. Introduction

In recent years, the **open-source hardware ecosystem** has matured to a point where advanced *analog* and *mixed-signal* integrated circuit (IC) design has become increasingly feasible outside proprietary EDA environments.
Building on this trend, the present work contributes to the **preparatory phase of a larger collaborative initiative** that aims to strengthen open-source-based chip design methodologies for small and medium-sized enterprises (SMEs), universities, and research institutes.

Using the design of an **LED driver IC** as a demonstrator, the overarching project seeks to **identify and close gaps** in the open-source design flow and to establish a reproducible, validated toolchain for mixed-signal system design.

> **i** Project Context
>
> This thesis forms the **technical pre-work** of a multi-partner initiative that extends the *IIC-OSIC-TOOLS* environment with the *IHP SG13G2 OpenPDK*, enabling future open-source mixed-signal tapeouts for SMEs and universities.

The long-term objective of the parent project is to extend the **open-source IIC-OSIC-TOOLS** developed by *Prof. Pretl (JKU Linz)* with the **IHP OpenPDK SG13G2**, a 130 nm SiGe BiCMOS process offering analog and RF capabilities suited for power management, sensor, and communication ICs.
By integrating the PDK into the IIC-OSIC-TOOLS flow, the project aims to make **open-source chip design accessible for SMEs**, who have so far been restricted to board-level development due to the prohibitive cost and complexity of commercial EDA ecosystems.
This initiative aligns with the global movement toward **democratizing IC design**, exemplified by the formation of the *IEEE SSCS Technical Committee on Open Source Ecosystem (TC-OSE)*.

2.1. Scope of the Thesis

Within this framework, the present thesis establishes the **proof of concept for the analog front-end** of the LED driver IC.
The work focuses on developing, simulating, and taping-out an **operational amplifier (OTA)** using the **IHP SG13G2 PDK** and the **IIC-OSIC-TOOLS**.
The design was submitted to the **IHP Open Multi-Project Wafer (MPW) Run in July 2025**, demonstrating a

complete, fabrication-ready analog layout realized entirely with open-source tools such as **Xschem**, **Ngspice**, **KLayout**, **Klayout-PEX**, and **CACE**.

> ❗ Technical Milestone
>
> The successful **tape-out of the OTA** represents one of the earliest demonstrations of the SG13G2 Open-PDK using the IIC-OSIC toolchain and serves as a validation of the *open-source analog flow*.

2.2. Motivation and Challenges

The analog subsystem of an LED driver in **aircraft cabin lighting** must regulate voltages between **18 V and 32 V**, ensuring efficiency, low noise, and individual seat controllability.
Such power-management circuits are particularly sensitive to **layout-dependent parasitic effects**, which can degrade performance. Hence, a major emphasis of this work lies in the **DRC- and LVS-clean** design and its reliability in open-source flows.

> 💡 Core Research Focus
>
> Improvement of **PEX precision** and **post-layout verification** within the open-source flow is the **central contribution** of this thesis and the broader collaborative project.

2.3. Methodological Contributions

Beyond the circuit-level design, this research establishes a **validated analog verification workflow** under fully open conditions.
The complete flow — from schematic capture, device sizing, and layout to LVS, PEX, and post-layout simulation — was developed within the *IIC-OSIC-TOOLS* environment.
Insights gained from the OTA design, including *device matching*, *guard-ring implementation*, *fill density*, and *PEX consistency*, are being integrated into improvements of the toolchain for future mixed-signal designs.

3. Declaration of Authorship

https://www.hs-bremen.de/assets/hsb/de/Dokumente/ZLL/MMCC/declaration_indep_work_Nov_2024_Form.pdf

I hereby certify that I have written this thesis independently and have not used any sources or aids other than those specified. Sources and aids other than those indicated. The passages in the thesis that have been taken from other works in terms of wording or meaning are from other works are identified as such.

This statement also applies to the graphics, sketches and illustrations contained in the thesis. I have not already submitted the work in the same or a similar form, including extracts, as part of an examination or coursework. submitted as part of an examination or coursework.

I hereby certify that the electronic version of the dissertation submitted is a complete copy of the printed version. and agree to an electronic check of the thesis using plagiarism software.

Please tick the box: One of the two options must be chosen in consultation between the assessor and the assessee.

- ☐ Option 1: Use of AI-based tools without mandatory labelling I confirm that I have not used any AI-based tools whose use has not been agreed in writing with the verifier. I understand that the use of text or other content and products generated by AI-based tools is not a guarantee of their quality. I accept responsibility for the work submitted and the content contained therein. I also assure you that my creative influence predominates in this work.
- ☐ Option 2: Use of AI-based tools with mandatory labelling I confirm that I have not used any AI-based tools whose use has not been agreed in writing with the verifier. I understand that the use of text or other content and products generated by AI-based tools is not a guarantee of their quality. I accept responsibility for the work submitted and the content contained therein. I also assure you that my creative influence predominates in this work. All verbatim or analogous adaptations and quotations, and all sections designed, written and/or edited using AI-based tools, will be marked and checked by me. The form of marking will be agreed between the examiner and the examinee.

I am aware that failure to comply with the above may have consequences in terms of examination law, and in particular may result in the examination being marked 'insufficient' or the coursework being marked 'failed' and, in the case of multiple or serious attempts to cheat, may result in my being de-registered.

First name and surname:

Matriculation number:

Date:

Signature:

3. Declaration of Authorship

First name and surname:

Matriculation number:

Date:

Signature:

Part I.

System

4. Requirements

A switch-mode LED driver IC needs to be designed to operate within an 18–36 V input range, support high-resolution (16-bit) PWM dimming for at least four RGBW channels with adjustable current control (1–40 mA), integrate protection features (ESD, over/under-voltage, over-temperature), and enable robust data communication with upstream and downstream ICs. The IC must also include an internal oscillator (≥25 MHz), internal generation of data line supply (3–5 VDC), and advanced diagnostics such as LED error detection.

4.1. Topology Selection:

To meet the defined performance and protection requirements efficiently, a buck converter topology has been selected for the LED driver IC. This choice is based on the analysis and recommendations from Bernhard Wicht's 2020 paper **Analog Building Blocks for DC-DC Converters,** [1] which highlights the buck converter as a suitable and energy-efficient architecture for step-down voltage regulation in integrated power management systems. Given the input voltage range of 18–36 V and the need for tightly regulated current control per channel (1–40 mA), the buck converter enables high efficiency and compact implementation. Its compatibility with mixed-signal integration and the ability to support precise current regulation and fast transient response makes it well aligned with the requirements of our LED driver IC, particularly in ensuring reliable performance across varying input conditions and thermal loads.

4.2. Block Diagram

Block	Function
Power Stage Interface	Inductor-Capacitor Buck output stage
PWM Generator	Ramp + Comparator
Error Amplifier	Feedback loop, V_ref vs V_out
Current Control	Per channel analog regulation (DAC + current sink)
Thermal + Voltage protections	Shutdown to over-temperature, over and under voltages
Biasing, reference circuits, reference sources	Bandgap or bias current

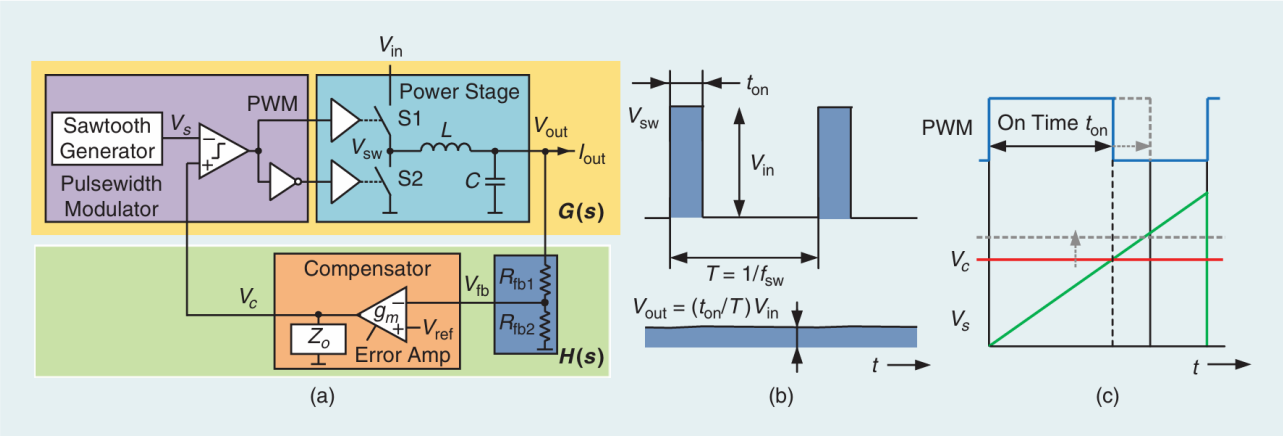


Figure 4.1.: Block Diagram

Part II.

Circuit Design

5. Thesis Focus

The focus of our thesis is designing and implementating error amplifier block within the selected buck converter topology using Open Source Tools. This involves developing the block from initial specification and schematic design through to layout, verification (including LVS and PEX), and final tapeout using exclusively open-source tools and IHP-SG13G2 PDK. The goal is to create a high-performance, low-power error amplifier tailored to the requirements of the mixed-signal LED driver IC, ensuring accurate regulation and stability of the control loop. This work will contribute a critical building block to the overall ASIC design while also serving as a case study in open-source analog IC design workflows.

The amplifier is tailored to meet the feedback stability and performance specifications of the LED driver system operating within a 5 V output and 18–36 V input range.

In the section detailing buck converter control design (especially the voltage-mode control loop), [1] focuses on:

- A transconductance amplifier used as the error amplifier
- High open-loop gain
- A clear emphasis on stability, bandwidth, and linear range
- In most cases, the paper assumes low voltage domain
- The architectures shown are two-stage OTAs or folded cascode for high PSRR

Final Take:

Folded Cascode OTA (see Figure 5.1) is very much aligned with [1] methodology, especially if our focus is on:

- A high-gain, linear, feedback-stage amplifier
- Working within 1.5V analog domain
- Dealing with moderate capacitive loads (e.g. compensation network / VCOMP node)
- [1] doesn't enforce one architecture rigidly, but the characteristics described match folded cascode best.

Table 5.1.: Finalised Specifications

Parameter	**Target Value**
Bandwidth	> 1 MHz
Phase Margin	> 60°
Power Supply	1.5 V (AVDD)

Parameter	Target Value
Load Capacitance	~1-2 pF
Power Consumption	< 1 mW

5.1. Into the design

As shown in the Figure 5.1, M_1 and M_2 are the input differential pair mosfets with $M_{5,6}$ and $M_{7,8}$ as their cascoded mosfets.

> **!** Cascode Bias Voltage Generation
>
> It is critically import for a stable performance across PVT that the bias voltages for the cascode gates are created in a manner that tracks variations with process, temperature, and supply voltage!

The current mirror constructed out of $M_{3,7}$ and $M_{4,8}$ is a special kind of **cascode current mirror for low-voltage operation**, also referred to as high-swing cascode current mirror [2]. This type is very often used, as it forces the V_{GS} and V_{DS} of $M_{3,4}$ to be equal, so the current mirror ratio is independent of g_{ds}.

5.2. Sizing using g_m over I_d method

In nanometer CMOS, the MOSFET behavior is much more complex than these simple models. Also, this highly simplified derivations introduce concepts like the threshold voltage or the overdrive voltage, which are interesting from a theoretical viewpoint, but bear little practical use. Modern compact MOSFET models (like the PSP model used in SG13G2) use hundreds of parameters and fairly complex equations to somewhat properly describe MOSFET behavior over a wide range of parameters like width, length and temperature. A modern approach to MOSFET sizing is thus based on the thought to use exactly these MOSFET models, characterize them, put the resulting data into tables and charts, and thus learn about the complex MOSFET behavior and use it for MOSFET sizing.

The gm over Id methodology has the huge advantage that it catches MOSFET behavior quite accurately over a wide range of operating conditions, and the curves look very similar for pretty much all CMOS technologies, from micrometer bulk CMOS down to nanometer FinFET devices. Of course the absolute values change, but the method applies universally.

We will be using the pygmid tool which is included in IIC-OSIC-TOOLS and is basically a python version of the gm/Id starter kit from Boris Murmann.

A brief implementation of this method is available here

Following is the python notebook designed for the sizing of Foldedcascode OTA with nmos differential input pair using g_m over I_d method.

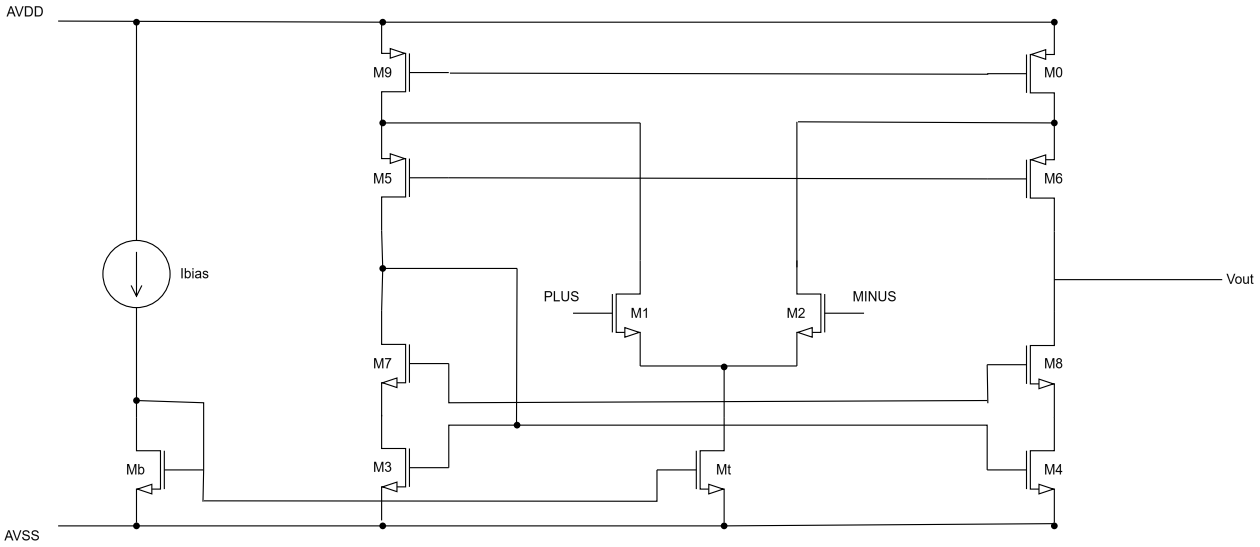


Figure 5.1.: Topology

OTA Sizing

```
# read table data
from pygmid import Lookup as lk
import numpy as np
lv_nmos = lk('sg13_lv_nmos.mat')
lv_pmos = lk('sg13_lv_pmos.mat')
# list of parameters: VGS, VDS, VSB, L, W, NFING, ID, VT, GM, GMB, GDS, CGG, CGB, CGD, CGS, CDD
# if not specified, minimum L, VDS=max(vgs)/2=0.9 and VSB=0 are used

# define the given parameters as taken from the specification table or initial guesses
c_load = 1e-12
gm_id_m12 = 18
gm_id_m34 = 12
gm_id_m56 = 13
gm_id_m78 = 13
gm_id_m90 = 10
gm_id_mt = 6
l_12 = 1
l_34 = 2
l_56 = 2
l_78 = 2
l_90 = 2
l_t = 5
f_bw = 1e6 # -3dB bandwidth of the voltage buffer
i_total_limit = 12e-6 # we plan 2x5uA in addition for additional bias voltage generation
```

```
i_bias_in = 14e-6
output_voltage = 1.3
vin_min = 0.7
vin_max = 0.9
vdd_min = 1.45
vdd_max = 1.55
vds_headroom = 0.2

# we get the required gm of M1/2 from the -3dB bandwidth requirement of the voltage buffer s
# note that the -3dB bandwidth of the voltage buffer with gain Av=1 is equal to the unity ga
# of the ota, hence we wet them equal here
# we add a factor of 3 to allow for PVT variation plus additional MOSFET parasitic loading
# we also add an additional factor of 2 to get more dc gain (and there is power still in the
gm_m12 = f_bw * 3 * 4*np.pi*c_load * 3

print('gm12 =', round(gm_m12/1e-3, 4), 'mS')

gm12 = 0.1131 mS

# since we know gm12 and the gmid we can calculate the bias current
id_m12 = gm_m12 / gm_id_m12
i_total = 2*id_m12
print('i_total (exact) =', round(i_total/1e-6, 1), 'µA')
# we round to 0.5µA bias currents
i_total = max(round(i_total / 1e-6 * 2) / 2 * 1e-6, 0.5e-6)
# here is a manual override to set the current; we keep a reserve of 2µA for bias branch
#i_total = 8e-6
id_m12 = i_total/2

print('i_total (rounded) =', i_total/1e-6, 'µA')
if i_total < i_total_limit:
    print('[info] power consumption target is met!')
else:
    print('[info] power consumption target is NOT met!')

i_total (exact) = 12.6 µA
i_total (rounded) = 12.5 µA
[info] power consumption target is NOT met!

# we calculate the dc gain
gm_gds_m12 = lv_nmos.lookup('GM_GDS', GM_ID=gm_id_m12, L=l_12, VDS=vds_headroom, VSB=2*vds_h
gm_gds_m34 = lv_nmos.lookup('GM_GDS', GM_ID=gm_id_m34, L=l_34, VDS=vds_headroom, VSB=0)
gm_gds_m56 = lv_pmos.lookup('GM_GDS', GM_ID=gm_id_m56, L=l_56, VDS=vds_headroom, VSB=2*vds_h
gm_gds_m78 = lv_nmos.lookup('GM_GDS', GM_ID=gm_id_m78, L=l_78, VDS=vds_headroom, VSB=vds_hea
gm_gds_m90 = lv_pmos.lookup('GM_GDS', GM_ID=gm_id_m90, L=l_90, VDS=vds_headroom, VSB=0)
```

```
# conductance of lower cascoded differential pair
gds_m12 = gm_m12 / gm_gds_m12
gds_m56 = gds_m12 / gm_gds_m56
# conductance of upper cascoded current mirror
gm_m34 = gm_id_m34 * i_total/2
gds_m34 = gm_m34 / gm_gds_m34
gds_m78 = gds_m34 / gm_gds_m78

print('gds_12 =', round(gds_m12/1e-6, 3), 'µs')
print('gm_56/gds_56 =',round(float(gm_gds_m56), 1))
print('gds_34 =', round(gds_m34/1e-6, 3), 'µs')
print('gm_78/gds_78 =', round(float(gm_gds_m78), 1))

a0 = gm_m12 / (gds_m56 + gds_m78)
print('a0 =', round(20*np.log10(a0), 1), 'dB')

gds_12 = 11.675 µs
gm_56/gds_56 = 43.6
gds_34 = 7.731 µs
gm_78/gds_78 = 19.7
a0 = 50.7 dB

# we calculate the MOSFET capacitance which adds to Cload, to see the impact on the BW
gm_cgs_m12 = lv_nmos.lookup('GM_CGS', GM_ID=gm_id_m12, L=l_12, VDS=vds_headroom, VSB=2*vds_head
gm_cdd_m56 = lv_pmos.lookup('GM_CDD', GM_ID=gm_id_m56, L=l_56, VDS=vds_headroom, VSB=2*vds_head
gm_cdd_m78 = lv_nmos.lookup('GM_CDD', GM_ID=gm_id_m78, L=l_78, VDS=vds_headroom, VSB=vds_headro


c_load_parasitic = abs(gm_m12/gm_cgs_m12) + abs(gm_m12/gm_cdd_m56) + abs(gm_m34/gm_cdd_m78)
print('additional load capacitance =', round(c_load_parasitic/1e-15, 1), 'fF')

f_bw = gm_m12 / (4*np.pi * (c_load + c_load_parasitic))
print('unity gain bandwidth incl. parasitics =', round(f_bw/1e6, 2), 'MHz')

additional load capacitance = 80.6 fF
unity gain bandwidth incl. parasitics = 8.65 MHz

# we can now look up the VGS of the MOSFET
vgs_m12 = lv_nmos.look_upVGS(GM_ID=gm_id_m12, L=l_12, VDS=vds_headroom, VSB=2*vds_headroom)
vgs_m56 = lv_pmos.look_upVGS(GM_ID=gm_id_m56, L=l_56, VDS=vds_headroom, VSB=2*vds_headroom)
vgs_m34 = lv_nmos.look_upVGS(GM_ID=gm_id_m34, L=l_34, VDS=vds_headroom, VSB=0.0)
vgs_m78 = lv_nmos.look_upVGS(GM_ID=gm_id_m78, L=l_78, VDS=vds_headroom, VSB=vds_headroom)

vgs_m90 = lv_pmos.look_upVGS(GM_ID=gm_id_m90, L=l_90, VDS=vds_headroom, VSB=0)
```

```
vgs_mt = lv_nmos.look_upVGS(GM_ID=gm_id_m56, L=l_t, VDS=vds_headroom, VSB=0.0)

print('vgs_12  =', round(float(vgs_m12), 3), 'V')
print('vgs_56 =', round(float(vgs_m56), 3), 'V')
print('vgs_34  =', round(float(vgs_m34), 3), 'V')
print('vgs_78 =', round(float(vgs_m78), 3), 'V')

print('vgs_90 =', round(float(vgs_m90), 3), 'V')

print('vgs_t  =', round(float(vgs_mt), 3), 'V')

vgs_12  = 0.343 V
vgs_56 = 0.529 V
vgs_34  = 0.365 V
vgs_78 = 0.375 V
vgs_90 = 0.518 V
vgs_t  = 0.318 V

# calculate settling time due to slewing with the calculated bias current
t_slew = (c_load + c_load_parasitic) * output_voltage / i_total
print('slewing time  =', round(t_slew/1e-6, 3), 'µs')
t_settle = 5/(2*np.pi*f_bw)
print('settling time =', round(t_settle/1e-6, 3), 'µs')

slewing time  = 0.108 µs
settling time = 0.092 µs

# calculate voltage gain error
gain_error = a0 / (1 + a0)
print('voltage gain error =', round((gain_error-1)*100, 1), '%')

voltage gain error = -0.3 %

# calculate total rms output noise
sth_m12 = lv_pmos.lookup('STH_GM', VGS=vgs_m12, L=l_12, VDS=vds_headroom, VSB=2*vds_headroom
gamma_m12 = sth_m12/(4*1.38e-23*300*gm_m12)

sth_m34 = lv_pmos.lookup('STH_GM', VGS=vgs_m34, L=l_34, VDS=vds_headroom, VSB=0) * gm_m34
gamma_m34 = sth_m34/(4*1.38e-23*300*gm_m34)

output_noise_rms = np.sqrt(1.38e-23*300 / (c_load + c_load_parasitic) * (2*gamma_m12 + 2*gam
print('output noise =', round(output_noise_rms/1e-6, 1), 'µVrms')

output noise = 89.0 µVrms
```

```
# calculate all widths
id_w_m12 = lv_nmos.lookup('ID_W', GM_ID=gm_id_m12, L=l_12, VDS=vds_headroom, VSB=2*vds_headroom
w_12 = id_m12 / id_w_m12
w_12_round = max(round(w_12*2)/2, 0.5)
print('M1/2  W =', round(w_12, 2), 'um, rounded W =', w_12_round, 'um')

id_m56 = id_m12
id_w_m56 = lv_pmos.lookup('ID_W', GM_ID=gm_id_m56, L=l_56, VDS=vds_headroom, VSB=2*vds_headroom
w_56 = id_m56 / id_w_m56
w_56_round = max(round(w_56*2)/2, 0.5)
print('M5/6 W =', round(w_56, 2), 'um, rounded W =', w_56_round, 'um')

id_m34 = id_m12
id_w_m34 = lv_nmos.lookup('ID_W', GM_ID=gm_id_m34, L=l_34, VDS=vds_headroom, VSB=0)
w_34 = id_m34 / id_w_m34
w_34_round = max(round(w_34*2)/2, 0.5)
print('M3/4  W =', round(w_34, 2), 'um, rounded W =', w_34_round, 'um')

id_m78 = id_m12
id_w_m78 = lv_nmos.lookup('ID_W', GM_ID=gm_id_m78, L=l_78, VDS=vds_headroom, VSB=vds_headroom)
w_78 = id_m78 / id_w_m78
w_78_round = max(round(w_78*2)/2, 0.5)
print('M7/8 W =', round(w_78, 2), 'um, rounded W =', w_78_round, 'um')

id_m90 = id_m12*2
id_w_m90 = lv_pmos.lookup('ID_W', GM_ID=gm_id_m90, L=l_90, VDS=vds_headroom, VSB=0)
w_90 = id_m90 / id_w_m90
w_90_round = max(round(w_90*2)/2, 0.5)
print('M9/0 W =', round(w_90, 2), 'um, rounded W =', w_90_round, 'um')









id_w_mt = lv_nmos.lookup('ID_W', GM_ID=gm_id_mt, L=l_t, VDS=vds_headroom, VSB=0)
w_t = i_total / id_w_mt
w_t_round = max(round(w_t*2)/2, 0.5)
print('Mt    W =', round(w_t, 2), 'um, rounded W =', w_t_round, 'um')
```

```
w_b = w_t_round * i_bias_in / i_total
print('Mb    W =', round(w_b, 2), 'um')

M1/2  W = 12.31 um, rounded W = 12.5 um
M5/6 W = 25.16 um, rounded W = 25.0 um
M3/4  W = 5.17 um, rounded W = 5.0 um
M7/8 W = 6.02 um, rounded W = 6.0 um
M9/0 W = 25.81 um, rounded W = 26.0 um
Mt    W = 9.72 um, rounded W = 9.5 um
Mb    W = 5.32 um

# Print out final design values
print('Improved OTA dimensioning:')
print('--------------------------')
print('M1/2  W=', w_12_round, ', L=', l_12)
print('M5/6 W=', w_56_round, ', L=', l_56)
print('M3/4  W=', w_34_round, ', L=', l_34)
print('M7/8 W=', w_78_round, ', L=', l_78)
print('M9/0 W=', w_90_round, ', L=', l_90)
print('Mt   W=', w_t_round, ', L=', l_t)
print('Mb   W=', round(w_b, 2), ', L=', l_t)
print()
print('Improved OTA performance summary:')
print('---------------------------------')
print('supply current =', round(i_total/1e-6, 1), 'µA')
print('output noise =', round(output_noise_rms/1e-6, 1), 'µVrms')
print('voltage gain error =', round((gain_error-1)*100, 1), '%')
print('unity gain bandwidth incl. parasitics =', round(f_bw/1e6, 2), 'MHz')
print('turn-on time (slewing+settling) =', round((t_slew+t_settle)/1e-6, 3), 'µs')
print()
print('Improved OTA bias point check:')
print('------------------------------')
print('headroom M1+M5 =', round(vdd_min-vgs_m34+vgs_m12-vin_max, 3), 'V')
print('headroom M4+M8 =', round(vdd_min-vin_max, 3), 'V')
print('headroom Mt     =', round(vin_min-vgs_m12, 3), 'V')

Improved OTA dimensioning:
--------------------------
M1/2  W= 12.5 , L= 1
M5/6 W= 25.0 , L= 2
M3/4  W= 5.0 , L= 2
M7/8 W= 6.0 , L= 2
M9/0 W= 26.0 , L= 2
Mt   W= 9.5 , L= 5
```

```
Mb   W= 5.32 , L= 5

Improved OTA performance summary:
---------------------------------
supply current = 25.0 µA
output noise = 89.0 µVrms
voltage gain error = -0.3 %
unity gain bandwidth incl. parasitics = 8.65 MHz
turn-on time (slewing+settling) = 0.2 µs

Improved OTA bias point check:
------------------------------
headroom M1+M5 = 0.527 V
headroom M4+M8 = 0.55 V
headroom Mt     = 0.357 V
```

5.3. Complete Design and constant g_m biasing using current mirrors

Based on the sizing procedure in Section 5.2, we are ready with all the W/L ratios and able to design the complete OTA (see Figure 5.2)

Find the schematic design here

5.3.1. Discussion of the OTA Design

We will now do an analysis of the circuit design of the OTA including all the complications which make this design practical.

1. For easier navigation, the device identifier are consistent with the circuit sketch in Figure 5.2.
2. Some MOSFET dimensions are rounded to make a better fit in the IC layout. Please also look carefully at W, L, and ng. The parameter ng defines how the total W of a MOSFET should be split into individual MOSFET fingers with $W_\mathrm{f} = W/\mathrm{ng}$. This is done to arrive at a suitably sized MOSFET physical implementation.
3. As you can (hopefully) see the circuit is carefully drawn to ease readability. Important nets are named, text comments state certain properties like nominal voltage levels, bias currents, etc. Current sensing elements are added to directly see the dc currents in the circuit simulation.
4. The bias voltage generation for the cascodes is included as well. The voltage drop for the bottom transistors is developed by properly scaling the MOSFETs in the reference branch. We reduce the W/L ratio to increase the V_{GS} to create a voltage headroom for the bottom MOSFET.

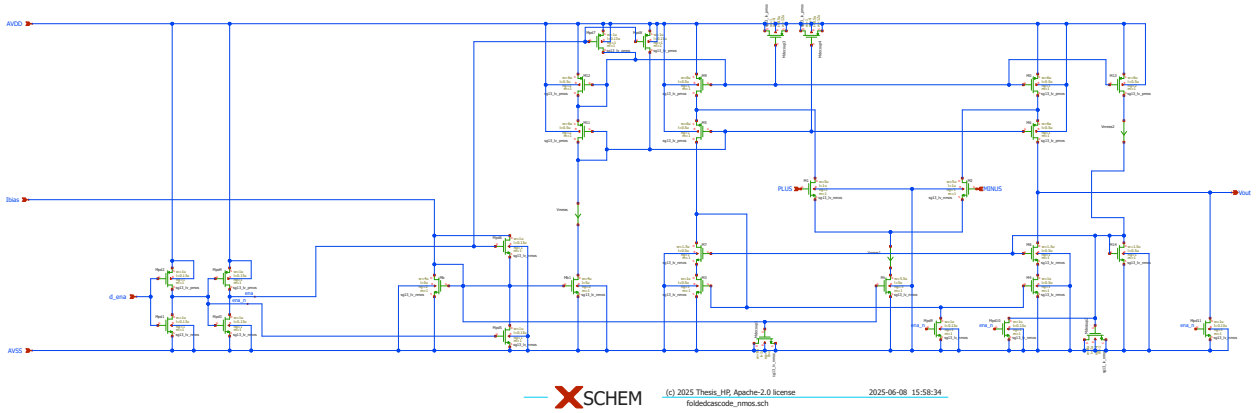


Figure 5.2.: OTA design in Xschem

5. Sensitive bias nodes are buffered with decoupling capacitors. We are using MOSFETs as nonlinear capacitors, which is not an issue in this application, but we value the increased capacitive density. Please note how the MOSFET are connected (some are tied to VDD while others are tied to VSS).

5.4. Simulation of the OTA

Now that circuit is ready we need to test its ac and transient behaviour as well as loop gain using Middlebrook's method and Tian's method as mention in Figure 5.3, Figure 5.5, Figure 5.7

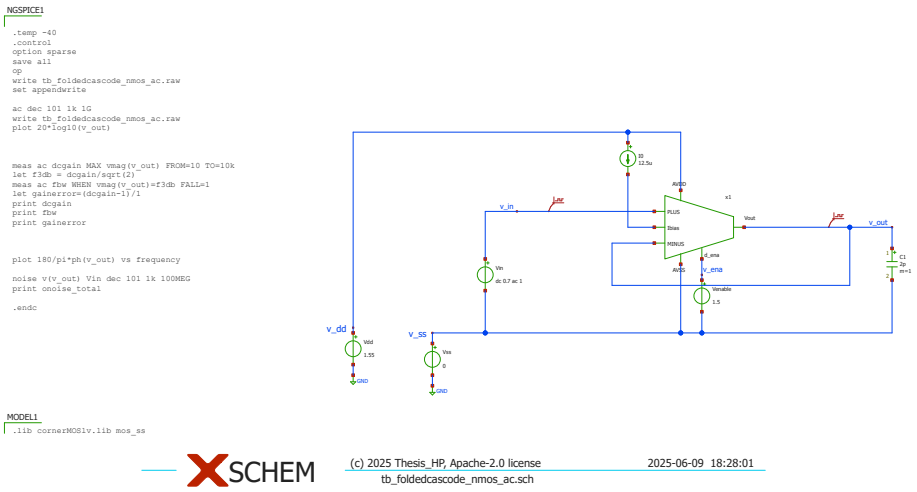


Figure 5.3.: AC Analysis Setup

The loop gain analysis yeilds the following plots which clarifies that our specifications such as gain and phase margin are met successfully. See Figure 5.8 and Figure 5.9 The plots include both Middlebrook's and Tian's method for loop gain.

‘

5.5. Corner Simulations for PVT and Monte-Carlo

As you have seen in Section 5.4, running simulations by hand is tedious. When we want to check the overall performance, we have to run many simulations over various conditions:

1. The supply voltage of the circuit has tolerances, and thus we need to check the performance against this variation.

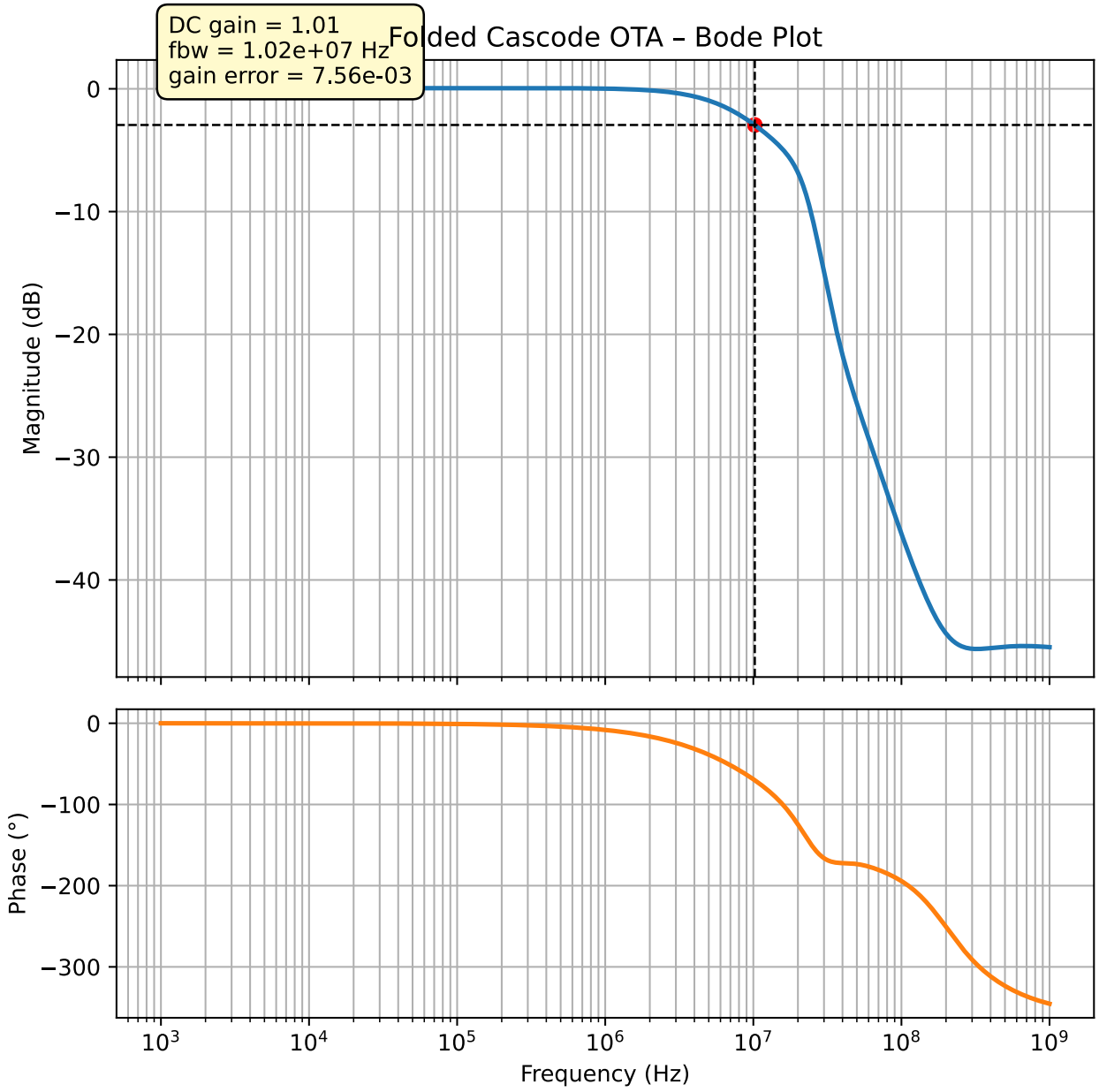


Figure 5.4.: AC Analysis Output

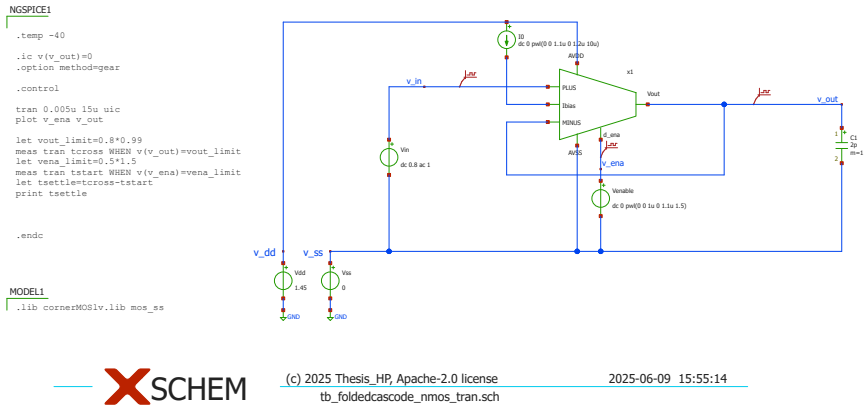


Figure 5.5.: Transient Analysis Setup

2. The temperature at which the circuit is operated is likely changing. Also the performance against this has to be verified.

3. When manufacturing the wafers random variations in various process parameters lead to changed parameters of the integrated circuit components. In order to check for this effect, wafer foundries provide model files which shall cover these manufacturing excursions. Simplified, this leads to a slower or faster MOSFET, and usually NMOS and PMOS are not correlated, so we have the process corners **SS**, **SF**, **TT**, **FS**, and **FF**. So far, we have only used the **TT** models in our simulations.

The variations listed in the previous list are abbreviated as **PVT** (process, voltage, temperature) variations. In order to finalize a circuit all combinations of these (plus the variations in operating conditions like input voltage) have to be simulated. As you can imagine, this leads to a huge number of simulations, and simulation results which have to be evaluated for pass/fail.

There are two options how to tackle this efficiently:

1. As an experienced designer you have a very solid understanding of the circuit, plus based on the analytic equations you can identify which combination of operating conditions will lead to a worst case performance. Thus, you can drastically reduce the number of corners to simulate, and you run them by hand.

2. You are using a framework which highly automates this task of running a plethora of different simulations and evaluating the outcome. These frameworks are called simulation runners.

Luckily, there are open-source versions of simulation runners available, and we will use CACE. CACE is written in Python and allows to setup a datasheet in YAML which defines the simulation problem and the performance

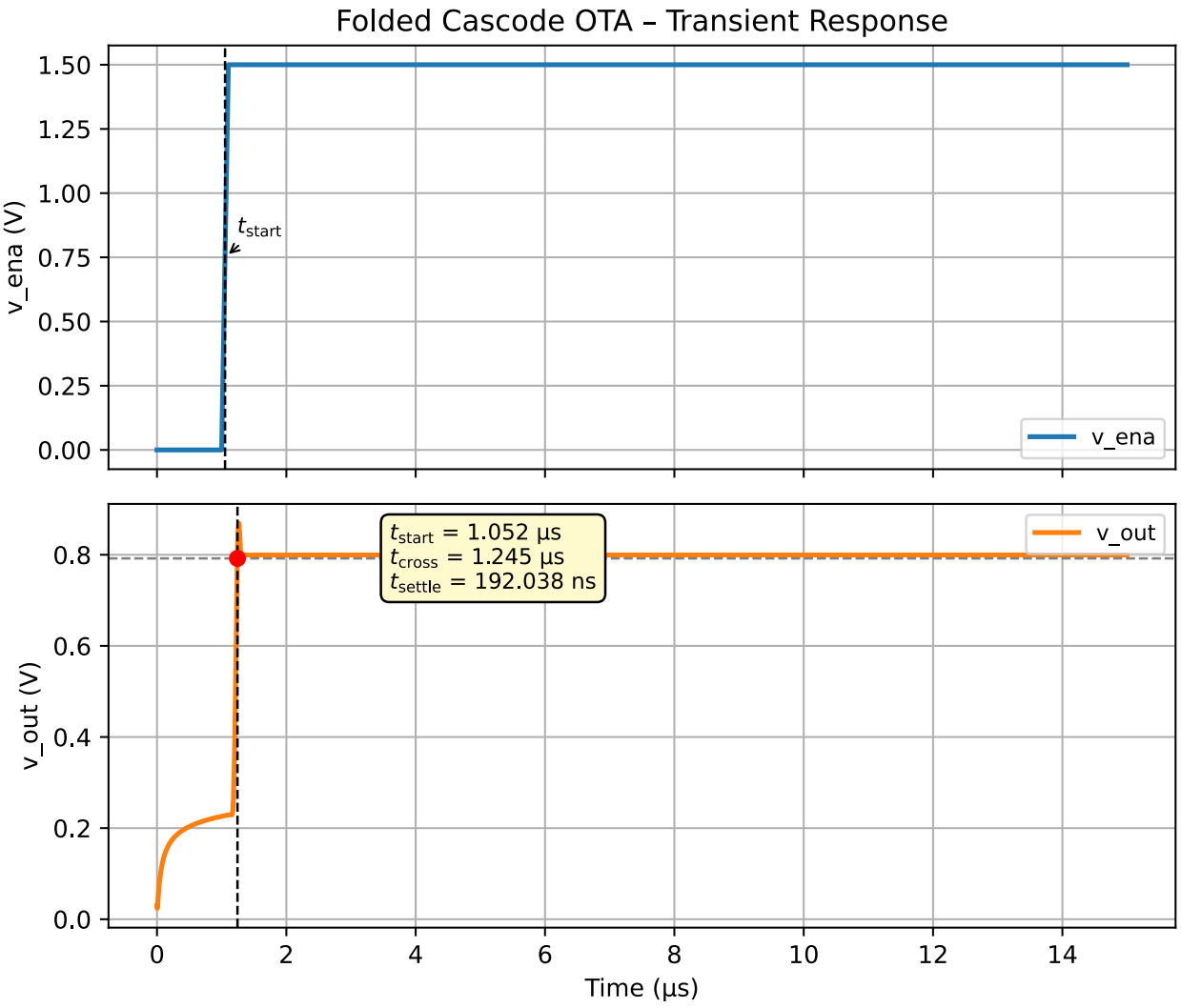


Figure 5.6.: Transient Analysis Output

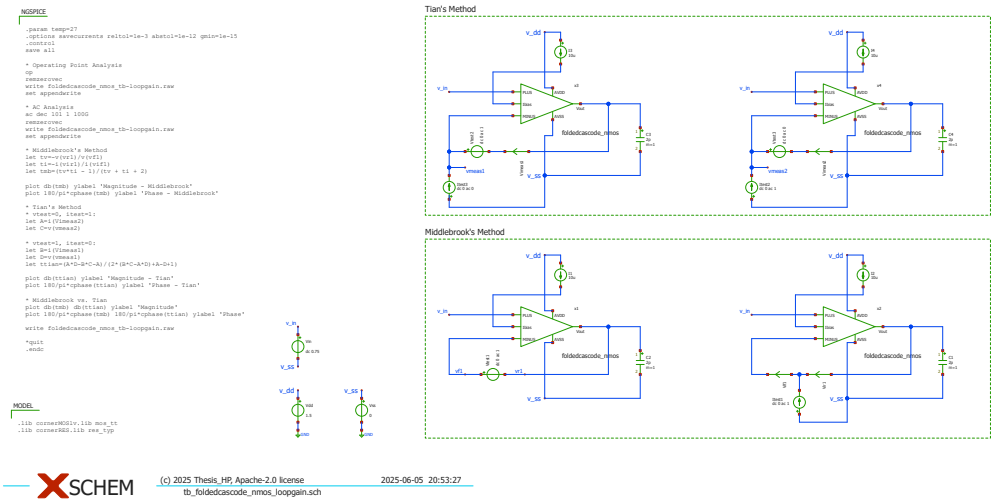


Figure 5.7.: Loop Gain Setup

parameters to evaluate against which limits. The resulting simulations are then run in parallel and the simulation data is evaluated and summarized in various forms.

There is a CACE setup available for our 5T-OTA. The datasheet describes the operating conditions and the simulations tasks. For each simulation a testbench template is needed, this one is used for ac simulations, this one is used for noise simulation, and this one is used for transient simulation.

After a successful run, a documentation is automatically generated. The result of a full run of this is presented below:

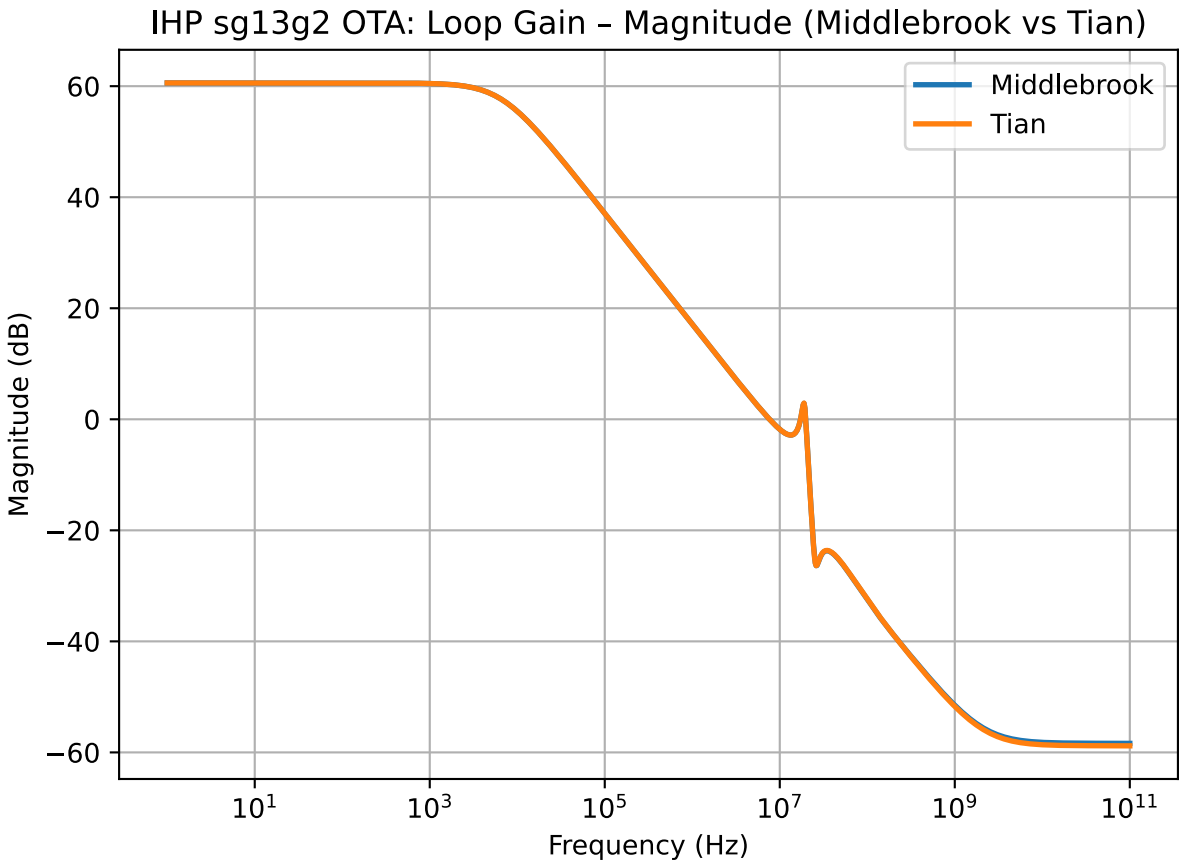


Figure 5.8.: Magnitude middlebrook vs Tian

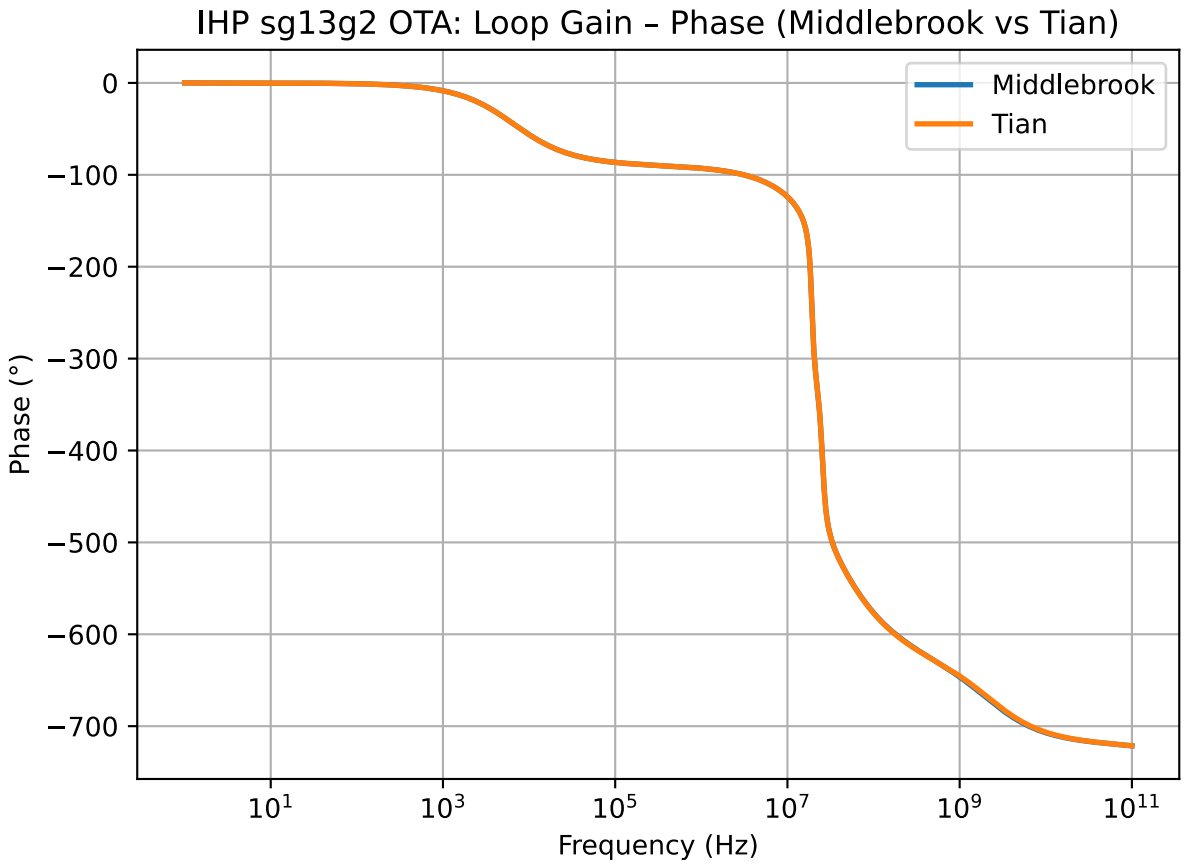


Figure 5.9.: Phase middlebrook vs Tian

CACE Summary for OTA

6. CACE Summary for foldedcascode_nmos

netlist source: schematic

Parameter	Tool	Result	Min Limit	Min Value	Typ Target	Typ Value	Max Limit	Max Value	Status
Output voltage ratio	ngspice	gain	0.98 V/V	0.996 V/V	any	0.999 V/V	1.1 V/V	1.000 V/V	Pass
Bandwidth	ngspice	bw	1e6 Hz	5118320.000 Hz	any	7827360.000 Hz	any	13271000.000 Hz	Pass
Output voltage ratio (MC)	ngspice	gain_mc	any	0.671 V/V	any	0.996 V/V	any	1.502 V/V	Pass
Bandwidth (MC)	ngspice	bw_mc	1e6 Hz	1024950.000 Hz	any	7454465.000 Hz	any	91913200.000 Hz	Pass
Output noise	ngspice	noise	any	0.069 mV	any	0.101 mV	0.2 mV	0.134 mV	Pass
Settling time	ngspice	tsettle	any	0.259 us	any	0.287 us	1.5 us	0.320 us	Pass

The situation with the design is summarized in Table 6.2.

Table 6.2.: Voltage buffer specification

Specification	OTA	Unit
Output voltage error	< 1	%
Total output noise (rms)	< 0.15	mV_{rms}
Supply current (as low as possible)	< 14	µA
Turn-on time	< 0.4	µs
Externally provided bias current (nominal)	12.5	µA

Part III.

Physical Design

7. Introduction to Layout

Once, schematic in **Xschem** is finished, what we have is only the functional intent of the circuit. At this stage the devices are ideal: transistors have exactly the width and length we assign, wires have zero resistance, and parasitics like source/drain capacitances or coupling between metals are invisible. The layout stage is where this abstraction becomes **real silicon geometry**. Here, every transistor must be drawn in the technology's layers, with actual fingers, source/drain contacts, guard rings and no component is ideal anymore.

For analog circuits, this step is critical. Small layout choices — for example, placing two input devices slightly asymmetrically, or routing one input longer than the other — can directly show up as offset voltage or degraded common-mode rejection. In digital design, such effects are often absorbed by logic thresholds, but in analog design they decide whether the OTA meets its simulated gain or not.

In other words, schematic defines *"what we want"*, layout decides *"what we actually get"*. This chapter focuses on showing how a *folded-cascode OTA* in **IHP-SG13G2** was translated from a verified schematic into a manufacturable layout that still respects matching, symmetry, and parasitic limits.

7.1. Layout Toolchain: KLayout

For the layout implementation, we relied on *KLayout*. It is a free, scriptable editor that is already integrated with the **IHP-SG13G2 open-source PDK** and provided in the **IIC-OSIC toolchain**. This makes it possible to perform layout tasks without the need for commercial software.

All necessary files — such as **Design Rule Checks (DRC)**, **LVS comparison setups**, and **Parameterized cells (PCells)** — are directly accessible and supported within KLayout. This allows devices to be generated, verified, and extracted in a reproducible way.

For further details, see the KLayout website.

> **i** Using KLayout: Docker vs. Source Build
>
> Our layout work was carried out using **KLayout v0.30.1**, provided within the **IIC-OSIC Docker image**. Since this image is a prebuilt package, some advanced KLayout or Qt features may not function as expected. If KLayout is build from source on your system, most of these limitations disappear, but in that case **IHP-SG13G2 technology files and libraries** must be manually integrated the into your local setup. For guidance on this process, a useful resource is this Efabless tutorial (second section by Thomas Perry).

8. Device-Level Layout Concepts

Integrated circuit layout translates the schematic into **physical layers** on silicon.
The IHP-SG13G2 process provides numerous layers such as active/diffusion regions, poly gates, contacts, multiple metal levels, and passivation. [3].
Each of these layers plays a role in forming MOSFETs, capacitors, resistors, and interconnects, further layer specific description for the IHP PDK can be found on Layout Rules [3].

Unlike digital design, where the goal is mainly connectivity, analog layout must ensure **matching, symmetry, and parasitic control**. This is particularly critical in our **folded-cascode OTA**, where the differential input pair, current mirrors, and cascode devices dominate the overall gain, offset, and common-mode rejection ratio (CMRR) [[4]][5].

The following subsections describe the major device-level layout techniques used in this work, with examples and figures suggested for illustration.

8.1. Matching & Symmetry

In real silicon, device characteristics are never perfectly uniform across the die.
Process gradients arise during fabrication due to variations in temperature, pressure, and material distribution. These gradients can be visualized using concepts similar to **isobars** (lines of equal pressure) or **isotherms** (lines of equal temperature), where physical properties gradually shift across the wafer surface [5].

Such gradients directly affect threshold voltage, mobility, and oxide thickness. As a result, two transistors with identical schematic dimensions may behave differently if placed at different positions on the chip. For analog circuits like OTAs, these mismatches degrade input offset, gain, and common-mode rejection ratio (CMRR).

To minimize this, layout must enforce **symmetry**:

- **Symmetry around an axis.**
  Devices intended to operate as pairs, such as the OTA differential input transistors, are placed symmetrically about a central axis. This ensures that any linear process gradient affects both devices equally, preserving balance in their electrical behavior.

- **Symmetry in current flow.**
  Symmetry is not only geometrical but also electrical. The orientation of source, drain, and gate terminals is arranged so that current flows through matched devices in identical directions. This prevents systematic mismatches introduced by unequal diffusion edge proximity or channel orientation.

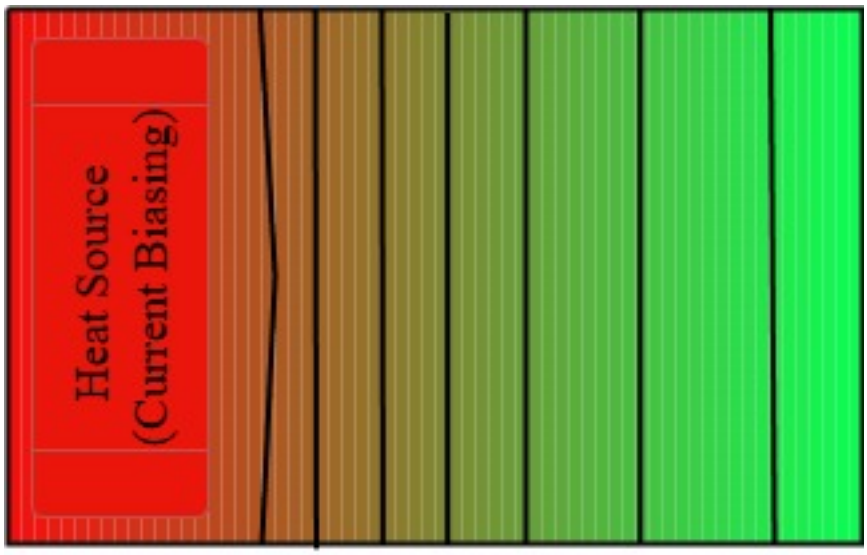


Figure 8.1.: Isotherm Process Gradient due to temperature

Together, these forms of symmetry improve **matching** — the ability of two devices to exhibit nearly identical electrical performance. Accurate matching is critical in the OTA's input stage, where small imbalances directly appear as input offset voltages.

Finally, symmetry must extend to the **routing**. Interconnect lengths, widths, and via stacks must be balanced on both sides of the differential pair. Even with well-matched devices, asymmetric wiring can introduce additional resistance and capacitance, which shows up as systematic offset or degraded bandwidth [4], [5].
In our OTA layout, routing symmetry was not only maintained in terms of path length and via balance, but also extended to the **choice of metal layers**. Long interconnects were assigned to **higher metal layers** with lower sheet resistance, while short local connections remained on **lower metals** for compact access.
This combination of length balancing, via symmetry, and layer assignment ensured that both differential branches exhibited comparable parasitic resistance and capacitance, preserving matching despite unavoidable topological constraints.

8.2. Multi-Fingering

When MOSFETs are designed with very wide channel widths, implementing them as a single continuous device introduces several problems. A long, uninterrupted polysilicon gate line introduces significant **polysilicon gate resistance**, which degrades high-frequency behavior and increases phase delay. At the same time, the enlarged diffusion regions lead to higher source and drain junction capacitances, creating substantial loading on sensitive circuit nodes. Finally, wide monolithic devices suffer in terms of matching: process gradients across the active area are no longer averaged, making these devices more vulnerable to systematic variation and random mismatch effects [4], [5].

The solution is **multi-fingering**: dividing the wide transistor into several narrower, identical fingers that are connected in parallel. This segmentation reduces gate resistance (shorter poly stripes), distributes junction

capacitances more evenly, and improves statistical matching by averaging across multiple smaller units.

In our OTA, the **tail current source** required a total width of 32 µm. Instead of realizing it as a single device, it was split into **eight fingers of 4 µm each**. This segmentation reduced the distributed gate resistance, improved current uniformity, and made gate routing more compact and symmetrical, while maintaining acceptable junction capacitance at the high-impedance tail node. Similarly, majority of transistors were implemented in multi-finger form to balance parasitic capacitances and maintain symmetry across branches.

Despite its benefits, multi-fingering comes with trade-offs:
- **Increased routing overhead**, since all fingers must be tied together with well-balanced interconnect.
- **Extra junction perimeters**, which can increase total parasitic capacitance if the layout is not compacted carefully.
- **Via placement sensitivity**, as each finger requires source/drain contacts and vias; poor via distribution can lead to resistance mismatches or electromigration concerns.

For this reason, multi-fingering is often combined with other layout strategies.
Dummies are placed at the array edges to shield active fingers, while **common-centroid arrangements** may be used for further gradient cancellation. This combined approach was applied in our OTA's diffpair, ensuring both parasitic reduction and high matching fidelity.

Figures to include:
- Comparison of a single wide MOSFET vs. multi-finger implementation (adapted from [4], Ch. 5).
- Annotated OTA tail transistor layout showing finger distribution.

> **i** How to calculate minimum number of fingers
>
> Determining the appropriate number of fingers for a wide MOSFET is not arbitrary; it requires balancing several parasitic and physical effects.
> As excessive fingering increases total diffusion perimeter and further elevating the total source/drain junction capacitance.
> Therefore, selecting the *minimum* number of fingers that satisfies these electrical and physical constraints is essential for achieving optimal analog performance.
> A rigorous methodology for estimating this minimum finger count—considering gate resistance limits, diffusion capacitance contributions, layout symmetry, and manufacturability—is presented in **Section 5.6: Finger Count Estimation**, where the all equations and procedures are discussed in detail.

8.3. Dummy Devices

Even when multi-fingering is used, the **edge fingers** of a device array experience different fabrication conditions compared to the inner fingers. At the boundary between active diffusion and isolation (STI), variations in oxide growth, stress, and implant profiles create **edge effects** that change threshold voltage, carrier mobility, and junction capacitance [4].
If left unaddressed, these systematic differences degrade device matching, particularly in sensitive analog structures such as the OTA input pair or current mirrors.

To eliminate this, **dummy devices** are placed at the ends of transistor arrays. They act as sacrificial neighbors, ensuring that all *active* devices are surrounded by identical environments.
This makes the inner devices more uniform, effectively shielding them from edge-related mismatches.

Connection of dummy devices can vary across design flows:
- In some methodologies, dummy gates are left **floating**, since their only role is geometric.
- In others, dummies are **tied to supply rails** (AVDD or AVSS) to avoid LVS errors and ensure consistent recognition by verification tools.

In our OTA, Dummy devices were placed at the edges of all the MOSFET's. This ensured that all transistors within the OTA operated in well-matched environments, minimizing systematic offset.

> **!** Important Implementation Note
>
> In our OTA layout, dummy devices were implemented as **complete MOSFETs** with their gates explicitly tied to **AVDD** or **AVSS**.
> This was necessary because in **KLayout**, adding an extra finger with an unconnected gate/source/drain is not recognized as a dummy. LVS then fails due to netlist mismatches.
> To ensure **LVS clean results**, we:
> - Added dummy devices explicitly in the schematic.
> - Connected all terminals to valid nets.
> - Replicated the same connections in layout.

8.4. Interdigitated Layout

When two matched transistors are placed adjacent to each other but in separate diffusion regions, each device is exposed to a slightly different portion of the process gradient across the chip. As a result, parameters such as threshold voltage, carrier mobility, and oxide thickness vary between them, leading to a systematic mismatch even though both devices have identical drawn dimensions.

The **interdigitated layout technique** addresses this issue by physically alternating the unit devices of a matched pair or array. Instead of grouping devices as AA–BB, the transistors are arranged in an alternating or mirrored pattern such as **ABAB** or **A–B–B–A**, where *A* and *B* represent identical unit devices belonging to two matched elements.
Through this alternating placement, each device is spatially distributed across the gradient, allowing it to experience regions of both higher and lower process variations. As a result, the local fabrication differences are **averaged** out across all unit fingers, and the two devices become physically and electrically better matched under real process conditions.

This concept can be visualized by imagining a linear gradient across the x-axis of the die. If one device occupies the left region and the other the right, they sample distinct gradient values. However, when their unit fingers are interleaved, each device occupies multiple local regions along the gradient, resulting in matched *average parameters*.

In practice, interdigitation is most effective for **linear process gradients** along one direction and is often used for **matched transistor pairs, current mirrors, or differential input stages**.
For two-dimensional gradients or more complex variations, this technique is typically extended using **common-centroid placement** (discussed in the next section).

In the folded cascode OTA, **interdigitation** was applied to the active loads and bias current mirrors. This layout ensured that each mirrored transistor experienced an equivalent average doping and oxide thickness, resulting in better current matching, symmetrical output swing, and reduced systematic offset in bias generation.

8.5. Common-Centroid Layout

While interdigitation cancels **first-order linear gradients** along a single direction, it cannot fully correct for **two-dimensional or higher-order variations** such as diagonal gradients or wafer-level curvature. For more advanced matching requirements—such as in precision current mirrors or differential pairs in high-gain OTAs **common-centroid layout** is used.

In a common-centroid configuration, matched devices are arranged so that their geometric centers, or **centroids**, coincide. Each device is distributed symmetrically around a central point such that any gradient—whether horizontal, vertical, or diagonal—affects both devices equally when averaged over their area. This approach effectively cancels first-order gradient components in both axes and reduces second-order effects as well [5].

Conceptually, the centroid layout extends the interdigitated approach into two dimensions. For example, four identical transistors can be arranged symmetrically around a central point in an AB/BA pattern, forming what is effectively a square or cross-shaped configuration. Although each individual finger occupies a slightly different position on the chip—and therefore a slightly different local environment—the geometric center (centroid) of all devices coincides. This ensures that the averaged electrical properties of each device are identical, effectively cancelling first-order process gradients across both axes.

In our folded-cascode OTA, the **diff pair** were laid out using a **common-centroid pattern** derived from the AB/BA principle. This was essential to maintain current accuracy and minimize systematic gain variation.
The layout ensured that matched transistors experienced a balanced environment, even in the presence of lateral thermal gradients or stress gradients from nearby metal density variations.

Routing Considerations:
Achieving symmetry in interconnect is as critical as maintaining symmetry in device placement. In principle, identical path lengths, matched metal layers, and mirrored via distributions help ensure that the parasitic resistances and capacitances experienced by paired devices remain equal. However, practical experience reveals that *strict geometric symmetry does not always translate into optimal electrical symmetry*, especially in multi-metal processes such as IHP-SG13G2.

A notable observation from our layout work illustrates this point clearly. In most of tech nodes including IHP-sg13g2 lower metal layers (Metal1, Metal2) exhibit **higher sheet resistance**, they are typically used for short internal routing, while upper metals provide lower resistance and reduced coupling.
In an attempt to maximize matching, the initial differential-pair routing was implemented almost entirely on the lowest metal layers (Metal1 and Metal2), with perfectly mirrored path lengths and identical layer usage

on both branches. Although geometrically symmetric, this approach resulted in **severe performance degradation**: the differential pair failed to operate correctly.
Post-layout analysis revealed that the close proximity of long parallel segments on the same or adjacent lower metal layers introduced **excessive lateral and broadside capacitances**, overwhelming the high-impedance input nodes and collapsing the expected differential behavior.

This led to a revised routing strategy grounded in parasitic-aware symmetry rather than purely geometric symmetry. The final solution employed a combination of **Metal1, Metal2, and Metal3**, arranged such that the two branches remained electrically matched but **did not rely on identical metal layers running in close parallel**. Where parallelism was unavoidable, routing was staggered (e.g., $Metal_n$ with $Metal_{n+2}$) to reduce capacitive coupling.
This parasitic-conscious routing produced a substantial improvement in OTA performance, restoring gain and achieving stable differential operation.

These findings highlight that in precision analog layout, **symmetry must be interpreted electrically, not only geometrically**. Effective matching requires an awareness of sheet-resistance gradients, metal-layer coupling, and the spatial distribution of parasitics.

Meticulous routing decisions—layer staggering, spacing, orthogonality, and controlled via placement—often determine whether a differential structure performs ideally or fails entirely.

8.6. Finger Count Estimation (Minimum Number of Fingers)

This section establishes a systematic method for estimating the **minimum number of fingers** N_f required for a wide MOSFET such that
(i) the **distributed gate resistance** remains below acceptable limits, and
(ii) the **junction capacitances** introduced by segmentation do not dominate the node dynamics.
The methodology follows the classical treatment of **gate resistance**, **thermal/gate noise**, and **diffusion capacitance scaling** described in Razavi's *Design of Analog CMOS Integrated Circuits* and related literature.

For a device with total drawn width W_{tot} and channel length *L*, splitting the device into N_f identical fingers results in an effective per-finger width

$$W_f = \frac{W_{\text{tot}}}{N_f}.$$

Selecting N_f is therefore an exercise in determining how narrow each finger can be made before device performance becomes degraded by source drain capacitence, or minimum-geometry constraints.

The following design principles guide the estimation procedure:

8.6.1. Gate Resistance Requirement (Low-Noise Criterion)

As a practical thumb rule used in low-noise analog design:

> Each finger width W_f must be chosen such that the **gate resistance** of that finger is **less than** the inverse transconductance $1/g_m$ associated with that finger.
> For high-performance or low-noise applications, the gate resistance should preferably be in the range of

$$R_g < \frac{1}{5g_m} \quad \text{to} \quad R_g < \frac{1}{10g_m}$$

This condition ensures that the **gate thermal noise contribution** remains significantly below the **channel thermal noise**, preventing degradation of input-referred noise or gain.[6]

Basically, we compare the **channel thermal noise** with the **gate-resistance thermal noise**, and we solve to get the value of N_f ensuring that gate noise remains well below channel noise.

Channel noise:

$$\overline{v_{n,\text{ch}}^2} = \frac{4kT\,\gamma}{g_m}$$

Gate-resistance noise:

$$\overline{v_{n,\text{g}}^2} = \frac{4kTR_{sheet}W}{3LN_f^2}$$

where,

- *k* is Boltzmann's constant,
- *T* is the absolute temperature,
- γ is the noise factor
- g_m, W & L are the transconductance, width and length of the device.

In our application, Channel noise is 5 times that of Gate-resistance noise.

8.6.2. Noise Model Adjustment Based on Channel Length

The optimization also depends on the **noise coefficient** γ in the MOSFET thermal noise model. The correct value of γ depends on the channel length:

- If $L < 3 * L_{min}$, then $\gamma = 1$
- otherwise, $\gamma = \frac{2}{3}$

Because shorter-channel devices exhibit stronger velocity saturation and increased thermal noise.

8.6.3. Minimum Geometry Constraint

The number of fingers must also satisfy:

$$\frac{W_{\text{tot}}}{N_f} > L_{\text{min}},$$

because excessively narrow fingers approach the **minimum manufacturable width** and exhibit strong proximity effects, increased mismatch, and higher perimeter capacitance.[7] For the IHP-SG13G2 130-nm process used here, as the name suggests the minimum channel length is:

$$L_{\text{min}} = 0.13\,\mu\text{m}.$$

This constraint prevents unrealistic fingering.

8.6.4. Even Number of Fingers (Practical Layout Consideration)

Although electrically permissible, **odd-valued finger counts** typically complicate routing and disrupt symmetry. Although in fewer cases odd numbered fingers are used as well.
But, a practical recommendation particularly in differential or current-mirror structures is:

Choose the **minimum even number of fingers** N_f that satisfies the electrical and geometrical constraints. Even segmentation simplifies mirrored routing, centroid or interdigitated arrangements, and guard ring alignment.

> **i** Number of fingers Calculation using Python Script
>
> To support the design methodology described above, a dedicated **Python script** has been developed to automatically compute the **minimum number of fingers** required for any given MOSFET width.
> The script implements the **thumb-rule constraints** introduced above. By providing required parameters, the script outputs the **minimum feasible even-valued** N_f that satisfies all electrical constraints.

9. Design Rule Checking (DRC)

The transition from an electrical schematic to a manufacturable integrated circuit relies on the integrity of the **layout**, which represents the true physical interface between design and fabrication. While schematic-level verification ensures functional correctness and circuit performance, it is the layout that determines whether the circuit can be fabricated reliably within the limits of the chosen process.
To formalize and automate this verification, the semiconductor industry have **Design Rule Checking (DRC)**—a systematic comparison of the layout geometry against a set of process-specific constraints provided by the foundry or technology provider (in our case IHP-PDK Authors).

9.1. Fundamentals of DRC

Design Rule Checking is a geometric verification step that ensures a layout adheres to the **minimum/maximum feature sizes, spacings, enclosures, and overlaps** defined by the fabrication process.
Each layer in a modern BiCMOS process (e.g. diffusion, polysilicon, metal interconnects, and vias) is subject to numerous rules derived from the physical limitations of photolithography, etching, and deposition steps.
Violating these constraints can lead to a spectrum of manufacturing defects: short-circuits due to over-etching, open connections caused by lithographic line thinning, or parametric variations resulting from disuniform film growth.
Thus, DRC forms the **first line of defense** against non-manufacturable or yield-critical layouts.

The design rules themselves originate from **process specifications**. Foundries experimentally determine the tolerances of their technology through test structures that measure lithographic resolution, etch bias, and alignment accuracy between layers. From these experiments, statistical margins are extracted and translated into deterministic layout constraints—expressed as rules such as *minimum Active width = 0.15 µm* or *minimum spacing between Metal1 lines = 0.18 µm*. [3] As discussed in [5], these rules balance manufacturability with design density: tighter rules enable compact designs but raise the risk of yield loss, while relaxed rules increase process robustness at the expense of silicon area.

9.2. Technology Dependence and Process Node

Every **technology node**—for instance, 130 nm, 65 nm, or 28 nm—defines its own set of DRC parameters. These parameters evolve with advances in lithographic precision, material systems, and transistor architecture. At older nodes such as **0.35 µm or 0.25 µm**, DRC rules mainly governed planar dimensions and spacing; however, in sub-100 nm processes, additional considerations such as **antennas, stress effects, and metal density**

gradients are incorporated. Consequently, the complexity of DRC rules increases exponentially as the process geometry scales down.[4]

The **Process Design Kit (PDK)** serves as the bridge between the foundry and the design environment.
It encapsulates all technology-specific information—design rules, layer definitions, device models, parasitic extraction parameters, and layout-Vs-schematic (LVS) configurations—into a single framework usable by EDA tools.
The DRC deck, which is a formalized file (often written in languages such as SVRF or Python-based scripts for tools like KLayout), encodes these geometric constraints in machine-readable form. When the layout is processed through the DRC engine, each polygon and layer is algorithmically compared against the rule database to identify violations.

In the case of the **IHP SG13G2 130 nm BiCMOS process**, used throughout this work, the *IHP-PDK* provides a comprehensive DRC rule set defining all critical geometrical limits—from diffusion-to-diffusion spacing and via enclosures to hierarchical density checks of upper metal layers, [3]. The *IHP-PDK* also supplies detailed technology definition files that can be manually integrated into layout tools such as KLayout. However, since the *IHP-PDK* is part of the *IIC-OSIC* open-source toolchain, these technology files and rule decks are already pre-configured within the provided Docker image, offering an optimized and reproducible setup for DRC verification.

In addition, the PDK documentation includes a complete layer table defining all process layers, along with descriptions of forbidden layers reserved for internal fabrication use, grid rules, and the full list of DRC constraints illustrations. This document has served as the primary reference for understanding the process and implementing all layout structures in this work. Hence, even within an open-source environment, the design remains **foundry-compliant**, physically realizable, and fully aligned with the manufacturing specifications of the SG13G2 process.

9.3. Importance in the Design Flow

DRC plays a pivotal role in ensuring that the **layout is not only electrically correct but physically manufacturable**.
As mentioned before, an unverified layout might pass all functional and electrical simulations yet still fail in fabrication because certain features violate lithographic limits or introduce yield-limiting defects. Typical examples include insufficient metal spacing leading to shorts, or inadequate enclosure of contacts resulting in open circuits after etching.
By enforcing DRC compliance before tape-out, the designer ensures that the design adheres to the **design-for-manufacturability (DFM)** principles required for reproducible production.

Furthermore, DRC compliance is not merely a binary condition of “pass or fail.” It also acts as a **metric of process awareness**: a DRC-clean layout demonstrates the designer’s adherence to technology constraints and reflects a realistic understanding of fabrication tolerances. In professional design flows, foundries refuse to fabricate any layout that fails DRC, as doing so would waste mask costs and process time.

Therefore, DRC stands as the final checkpoint between design creativity and physical implementation—a gate that ensures the **logical intent of the schematic can manifest in silicon without violation of the underlying physics**.

9.4. Practical Implementation of Design Rule Checking

As discussed in earlier chapters, the **layout is the sole design entity that directly communicates with fabrication**. Neither schematic diagrams nor behavioral models ever reach the foundry—only the geometric database (typically in GDSII or OASIS format) generated from the layout is delivered for mask generation and processing. In this sense, **Design Rule Checking (DRC)** serves as the mechanism through which the *language of design* is translated into the *language of manufacturing*. It guarantees that every transistor, interconnect, and via is drawn within the lithographic and etching tolerances established by the process technology. This alignment between layout geometry and process capability is essential to achieving both functional yield and long-term device reliability.
To borrow the phrasing of [4], *"a perfect schematic means nothing without a correct layout."*
DRC ensures that this correctness is quantifiable and verifiable—it enforces the discipline that transforms circuit theory into a physically reproducible artifact, one capable of moving confidently from layout database to wafer fabrication.

In this work, all DRC verification was performed using **KLayout** as part of the **IIC-OSIC open-source toolchain**, which integrates the **IHP SG13G2 130 nm BiCMOS process**. The PDK provides complete DRC rule decks in the form of Python-based macros that systematically check every process layer and layer combination—diffusion overlaps, via enclosures, contact spacings, metal width and density requirements, and top-metal clearance to the passivation opening.

The DRC engine performs a hierarchical scan of the layout, flags any violations, and displays annotated markers directly within the KLayout interface for review. Each violation encountered during the layout phase was analyzed and corrected at either the device or routing level until the design achieved a fully **DRC-clean status**. This verified layout therefore not only satisfies the intended schematic connectivity but also conforms to every **physical constraint required for successful fabrication**. Through this process, DRC validation in KLayout establishes the final and most critical bridge between the abstract world of circuit design and the tangible reality of silicon manufacturing—where compliance with foundry rules ensures that what is drawn can, indeed, be built.

9.4.1. Types of DRC Scripts in the IHP SG13G2 Process

The **IHP SG13G2 PDK** provides two distinct DRC scripts, each corresponding to a specific level of design compliance and verification rigor. These scripts are referred to as **sg13g2_minimal** and **sg13g2_maximal** rule decks. Both are written as Python macros configured within the KLayout environment and can be executed directly from the *Macros Development* interface.

1. The **sg13_g2_minimal DRC deck** enforces the *standard design rules* of the process: These rules define the **mandatory fabrication constraints** that must be satisfied to ensure the layout is physically manufacturable. Violations of these checks typically indicate errors that could prevent successful chip production or cause fatal defects in the fabricated devices. Examples include insufficient metal spacing, missing via enclosures, or incorrect layer overlaps. List of minimal rules can be found here.

2. The **sg13_g2_maximal DRC deck**, in contrast, applies the *recommended design rules*: These rules contain both **mandatory fabrication constraints** and **non-mandatory** therefore are intended to improve overall

manufacturability, process reliability, and yield consistency. They provide design hints to minimize parametric variability, such as improving parasitics, preventing latch-ups, or avoiding angles that can introduce stress points during fabrication. While designs that only pass the minimal check are still fabricable, running the sg13_g2_maximal script ensures the layout adheres to **best practices** for long-term robustness and high-yield performance. List for maximal rules can be found here.

9.4.2. Performing DRC in KLayout

As mentioned in the *Known Issues* section, DRC cannot be executed in KLayout solely by relying on direct tools or commands. Instead, a specific workflow must be followed. Within the **Macros Development** interface of KLayout, there is a dedicated **DRC section** that lists all available rule scripts for the selected technology. To perform DRC, the user must select the desired rule deck—either *sg13_g2_minimal* or *sg13_g2_maximal*—and then execute the corresponding script.

Once the check is complete, the list of errors appears in the **output console** below the editor window. Unfortunately, due to current tool limitations, KLayout does not automatically generate a separate **DRC report file**. This means that the error list cannot be reloaded, filtered, or revisited directly within the tool after the session ends. Furthermore, some **Qt-based interactive functions** (intended to automatically highlight or open error markers) are not fully supported within the Docker image of the IIC-OSIC toolchain.

To address this limitation, a **custom Python script** was developed in this work to automate the error-handling process. The script parses the raw DRC output copied from the KLayout console and filters the messages to produce a clean, structured list of all raised violations that require manual correction. This greatly simplifies the debugging workflow by identifying unique errors and mapping them to their respective layer categories.

9.4.3. DRC Automation Script

This is the automation script used for processing DRC results:

Script

DRC Errors Check
This script gives 2 functions: 1st needs the the errors copied from KLayout to be pasted in drc_log function inside script and than run the script. It can be tidious to change script again and again. Therefore, not recommended.
For 2nd function just run the script and paste your errors copied from KLayout in the terminal and press enter and DONE. By default, script uses this function.

```
# DRC Error Check
# 1st function

def parse_drc_errors(drc_log: str):
    lines = drc_log.strip().splitlines()
    print(" Rules with errors:")
```

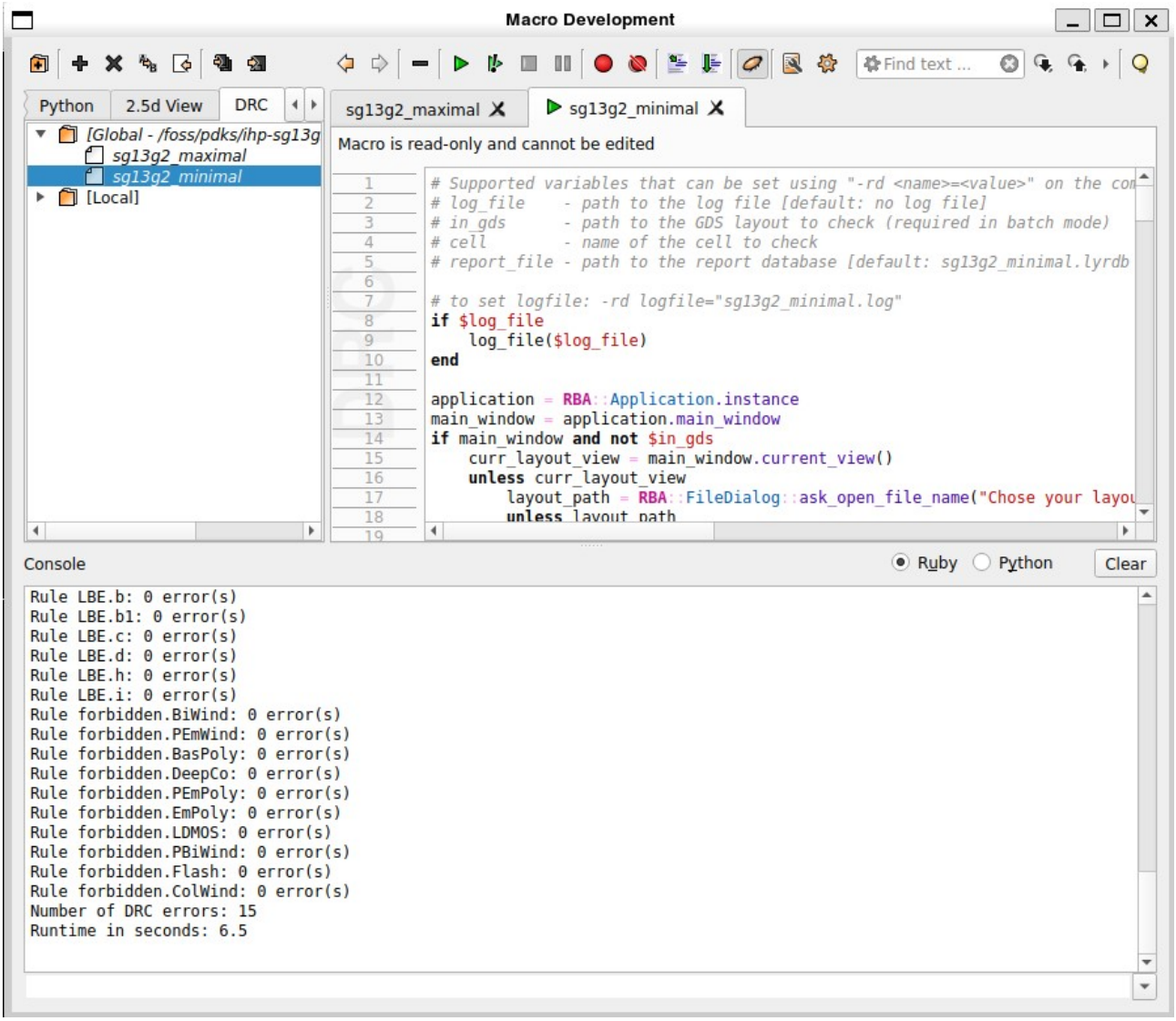


Figure 9.1.: Macro Development interface with DRC Results in Console

```
    for line in lines:
        if "error(s)" in line:
            try:
                rule, error_text = line.split(":")
                error_count = int(error_text.strip().split()[0])
                if error_count >= 1:
                    print(f"{rule.strip()}: {error_count} error(s)")
            except ValueError:
                continue

# Example: Paste your DRC log below READ THE COMMENT BELOW
drc_log = """
Rule forbidden.PEmPoly: 0 error(s)
Rule forbidden.EmPoly: 0 error(s)
Rule forbidden.LDMOS: 0 error(s)
Rule forbidden.PBiWind: 0 error(s)
Rule forbidden.Flash: 0 error(s)
Rule forbidden.ColWind: 0 error(s)
"""
# parse_drc_errors(drc_log)
# remove above comment ==>> If you want to put manually inside script not in terminal than us

# 2nd Funtion

def parse_drc_errors_new():
    print(" Paste your DRC error log below. Press Enter twice to finish:")

    # Collect multiline input
    lines = []
    while True:
        line = input()
        if line.strip() == "":
            break
        lines.append(line)

    print("\n  Rules with errors :")
    i=0
    for line in lines:
        if "error(s)" in line:
            try:
                rule, error_text = line.split(":")
                error_count = int(error_text.strip().split()[0])
                if error_count >= 1:
                    print(f"{rule.strip()}: {error_count} error(s)")
```

```
                    i=1

            except ValueError:
                continue

    if i==0:
        print("Congrats! No error")

# Run the parser
parse_drc_errors_new()
```

```
 Paste your DRC error log below. Press Enter twice to finish:

  Rules with errors :
Congrats! No error
```

To use the script, simply copy the DRC console output from KLayout, paste it into the terminal running the Python program, and the script will generate a concise summary of the errors that remain to be corrected manually in the layout.

> **Pro Tip — Run DRC Early and Often**
>
> It is strongly recommended to run DRC checks **after every 5–6 layout modifications**, even for small routing or placement changes. This practice prevents the accumulation of undetected violations that can take hours to debug later. Running DRC frequently not only saves significant time but also prevents accidental geometry overlaps or spacing errors that could compromise the layout structure and require extensive rework.

Figures to include:
- Illustration of typical DRC violation categories (spacing, enclosure, overlap).
- Screenshot of DRC results window in KLayout for the OTA layout.
- Example cross-section showing how spacing and enclosure rules correspond to physical layers.

10. Layout Vs Schematic (LVS)

After a layout has been verified for physical manufacturability through Design Rule Checking (DRC), the next essential step is to confirm its **electrical correctness** with respect to the original schematic. This verification step is known as **Layout Vs Schematic (LVS)**. Whereas DRC ensures that the physical geometry can be fabricated, LVS ensures that the fabricated geometry will function as intended by the circuit design. Together, DRC and LVS form the core of the **physical verification** process that bridges the logical and geometric domains of IC design.

10.1. Fundamentals of LVS

Layout Vs Schematic comparison is a **netlist-based verification process**. It compares the electrical connectivity extracted from the physical layout with that defined in the schematic. During LVS, the tool first performs **netlist extraction** from the layout (parsing all transistors, polygons, capacitors, and interconnects), and then compares this extracted netlist against the schematic netlist. The goal is to verify that both represent the same electrical circuit — that is, the same devices, same connections, and identical hierarchy.

Typical checks performed during LVS include:
- **Device recognition**: confirming that every transistor or passive component in the layout corresponds to a schematic element.
- **Pin and net matching**: ensuring all connections and signal names align between the two domains.
- **Parameter equivalence**: verifying that critical parameters such as transistor width, length, and multiplicity match (number of fingers).

If all devices and nets correspond correctly, the design is said to be **LVS clean**. Otherwise, the LVS report lists mismatches that must be corrected before fabrication.

10.2. Importance in the Design Flow

While DRC ensures *how* a design can be fabricated, LVS ensures *what* will be fabricated. Even a DRC-clean layout can fail to perform correctly if a wire is connected to the wrong node, a transistor terminal is swapped, or a device is missing. Such errors may lead to functional failure, excessive power consumption, or biasing issues that are difficult to detect without LVS.

Therefore, LVS verification is critical before tape-out because it confirms the **topological equivalence** between the intended circuit (schematic) and its physical realization (layout).

It ensures that all devices, interconnects, and terminals have been represented faithfully, safeguarding against layout-level errors that could otherwise result in costly re-fabrications.

10.3. Practical Implementation

In this work, LVS verification was carried out using **KLayout** in combination with the **IHP SG13G2 PDK** inside the **IIC-OSIC Docker environment**.
However, due to the known limitation that Qt-based LVS functions do not operate within the Dockerized KLayout interface, LVS must be performed **externally in the Docker terminal** using a *Python script provided by the IHP PDK*.

> **LVS Script**
>
> The LVS verification script provided with the **IHP SG13G2 PDK** can be found at:
> `/foss/pdks/ihp-sg13g2/libs.tech/klayout/tech/lvs/`
> The main Python file, `run_lvs.py`, executes the complete LVS procedure.

This LVS script automates the comparison between the schematic netlist and the layout. To execute it, the user provides three key inputs:

- The **schematic netlist** (in SPICE format, generated from schematic in Xschem)
- The **layout GDSII file** (the physical design)
- The **output directory** for output file publication

Once executed, the script performs layout netlist extraction, mapping, and comparison against the schematic. It then produces several output files, typically including:

- `.lvsdb` — the LVS database containing the extracted and compared netlists with in-depth details
- `.cir` — the extracted SPICE-format netlist from the layout containing circuit description
- `.log` — the report of information such as name of files, tool version and error if encountered

> **Important Naming Consistency**
>
> For LVS to run successfully, the **schematic name**, **GDS file name**, **SPICE netlist name**, and the **top-cell name** inside the GDS must all be **identical**.
> If any of these names differ, the Python script will terminate with an unexpected error because it cannot match the top-level hierarchy between schematic and layout.

10.4. Interpreting and Visualizing LVS Results in KLayout

After running the LVS script in the terminal, the generated `.lvsdb` file can be opened in **KLayout** for detailed inspection and mapping. To do this, open the corresponding GDS layout in KLayout, then navigate to:

Tools → Netlist Browser

This opens the Netlist Browser catalog window. Within this interface, load the `.lvsdb` file generated by the script. When the .lvsdb file is opened in the Netlist Database Browser within KLayout, a catalog window appears with four main tabs — **Netlist, Schematic, Cross Reference, and Log**. The Cross Reference tab displays a side-by-side comparison between the layout and reference schematic, listing all pins, nets, and devices in both domains. Matched elements are shown in green, indicating full correspondence between schematic and layout, as illustrated in Figure below (example: analog_inverter).

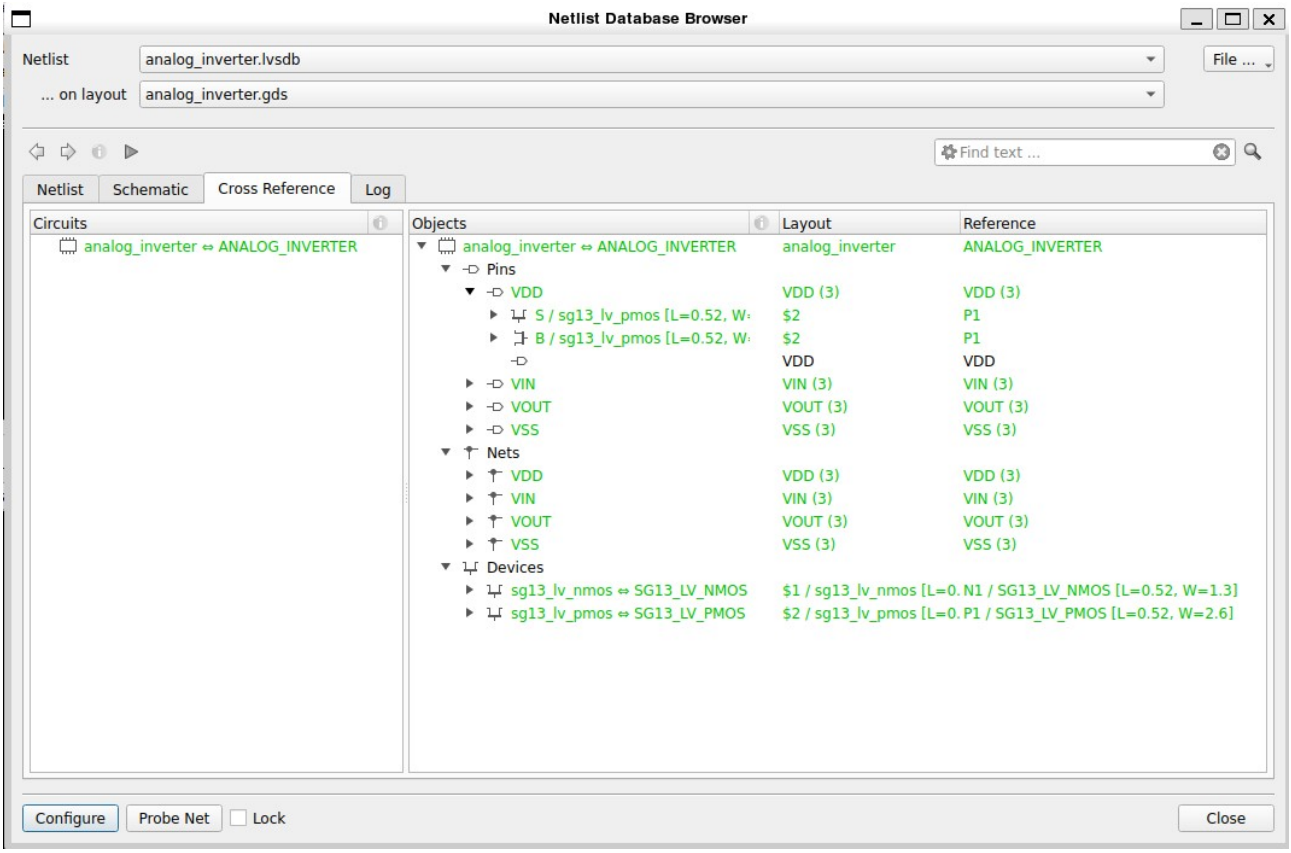


Figure 10.1.: Netlist Browser to view LVSDB files and trace nets

If mismatches occur, the differences are highlighted in red, showing missing devices, incorrect connections, or parameter discrepancies. These issues are further detailed in the Log tab, which provides textual summary of each error and its location in the design. This visualization enables efficient debugging by allowing the designer to inspect each net and device interactively, ensuring complete electrical equivalence before declaring the layout LVS-clean.

Klayout also provides powerful **tracing functionality**, allowing designers to visually select and highlight individual devices or nets within the layout to locate mismatches precisely.
This interactive feedback significantly accelerates debugging and helps ensure that every element is electrically connected as intended.

In summary, LVS is the final and most critical step in confirming that a verified, DRC-clean layout is also **electrically identical** to its schematic representation. By performing LVS through the IHP-provided Python script and analyzing results within KLayout, the design achieves both physical and electrical closure.
The successful completion of this step certifies that the layout accurately implements the intended circuit topology, allowing it to proceed confidently toward **parasitic extraction and post-layout simulation**, and ultimately, **fabrication**.

11. Parasitics and Prevention

Even though a layout is **DRC- and LVS-clean** but can still fail in silicon if parasitics and noise coupling are ignored. This chapter formalizes the main parasitic mechanisms relevant to **analog ICs**, outlines why analog is **more vulnerable than digital**, and documents the concrete prevention measures implemented in our design. The discussion follows the physical reasoning and best-practice guidance in [5] (Ch. 7), adapted to the SG13G2 context.

11.1. Why Analog Suffers More from Parasitics

Digital logic tolerates substantial RC and substrate coupling because functionality is determined by logic thresholds and noise margins. By contrast, **analog circuits** encode information in **amplitude, phase, and continuous bias conditions**. Small parasitic shifts in device parameters or interconnect coupling directly effects gain, offset, bandwidth, and noise.

Key vulnerabilities arise because analog circuits operate on continuous voltage and current levels, where even small parasitic interactions can shift the operating point or distort the signal. 1. High-impedance nodes, such as the inputs of operational transconductance amplifiers (OTAs), are particularly sensitive because they cannot source or sink significant current. Any parasitic capacitance — whether from junction diffusion, metal-to-substrate coupling, or fringe fields from nearby interconnects — directly loads these nodes. This loading introduces unwanted poles, reduces input resistance, and increases thermal noise coupling. In precision amplifiers, such parasitic capacitances can also create signal-dependent charge storage, causing distortion or phase lag at higher frequencies.

2. Bias networks and current mirrors are another vulnerable class. These circuits rely on exact matching of device characteristics and stable well or substrate potentials to maintain bias accuracy. Substrate noise or potential fluctuations, often injected from digital switching activity or supply ripple, modulate the body voltage of transistors. This alters their threshold voltage and transconductance (g_m), leading to current imbalance, gain drift, and degraded common-mode rejection. In deep-submicron processes with thin oxides and shallow wells, such coupling becomes even more significant.

3. Feedback paths in analog systems — particularly those forming loops for gain control, regulation, or filtering — are strongly influenced by parasitic-induced poles and zeros. Capacitances at feedback nodes or unintended inductive loops in routing can introduce additional frequency-dependent elements into the loop transfer function. These parasitics may reduce phase margin, cause peaking in the frequency response, or even lead to oscillations if uncompensated.

Consequently, the layout must be arranged to minimize parasitic capacitance at high-impedance nodes, isolate bias networks from noisy regions, and maintain compact, symmetrical feedback routing to preserve loop stability. Therefore, parasitics must be **anticipated and shaped**—not merely checked—during layout.

11.2. Interconnect and Device-Level Parasitics

11.2.1. Metal and Coupling Capacitances

Parallel conductors form **inter-wire capacitances**; long, co-planar runs also couple substrate noise through capacitance to ground.
In our layout, we **avoid long same-layer parallel routing** on sensitive nets. Where proximity is unavoidable, we prefer **layer staggering** (e.g., route one net on M_n and the other on M_{n+2}), maintain generous spacing, and, when beneficial, add **symmetric ground shields**. Orthogonal routing between adjacent layers (Manhattan style) further reduces broadside coupling [5].

11.2.2. Via and Contact Parasitics

Vias introduce **series resistance** and small fringing capacitances; unbalanced via stacks can create gain/offset asymmetry. We use **redundant via arrays** on low-impedance and high-current paths, keep **via symmetry** across matched branches, and avoid unnecessary layer hops on high-impedance nodes. This also improves electromigration margins.

11.2.3. Junction/Overlap Capacitances

Source/drain junction areas and perimeters add capacitance that **loads high-impedance nodes** and shifts poles.
We mitigate this by **multi-fingering** large devices (controlling diffusion area/perimeter) and by **dummy fingers** to regularize edge effects, as detailed earlier. Device segmentation is paired with **balanced routing and via symmetry** to preserve matching.

11.3. Critical Parasitics Mechanisms and Mitigation Strategies

As discussed above, the physical behavior of the silicon can be compromised by secondary effects. These parasitic mechanisms—though invisible in schematic design—can severely degrade performance or even render an integrated circuit non-functional.
The following subsections discuss the most relevant effects for analog circuits and the preventive measures applied in this work.

11.3.1. Latch-Up

Mechanism.
Latch-up is a destructive parasitic phenomenon that occurs primarily in bulk CMOS and BiCMOS technologies. Within the substrate and well structure, every pair of adjacent p–n junctions inherently forms **parasitic bipolar transistors** (pnp and npn).
When certain layout geometries or biasing conditions bring these devices into proximity, they can create an unintended **silicon-controlled rectifier (SCR)** path between the power rails (AVDD and AVSS).
A transient event—such as an electrostatic discharge, substrate noise spike, or voltage overshoot—can trigger the SCR, producing a **low-impedance short** between supply and ground.
Once triggered, the latch-up state sustains itself through regenerative feedback, often leading to local thermal runaway and irreversible damage.

Prevention.
Latch-up prevention requires **breaking the feedback path** and **lowering substrate resistance**.
This is achieved by surrounding all active devices with **dense guard rings** and inserting **well and substrate contacts** at regular intervals to provide low-resistance current return paths.
Adequate **spacing between n-wells and p-wells** reduces parasitic coupling between complementary transistors.
For highly sensitive analog blocks, the **deep-N-well (isolation well)** option provided in the SG13G2 process was employed. This isolation ensures that the substrate region of the analog core is electrically separated from neighboring digital or I/O structures, effectively suppressing latch-up initiation.
Additionally, I/O pads and high-swing nodes are placed away from precision analog circuits to minimize injection of transient substrate currents [5].

11.3.2. Antenna Effect

Mechanism.
During etching and deposition steps in fabrication, unconnected metal or polysilicon features can act as **charge-collecting antennas**.
If a long interconnect is connected to a transistor gate but not yet to a diffusion region, the charge accumulated on the metal surface can discharge through the thin gate oxide once the diffusion is formed.
This discharge can permanently **damage or weaken the gate dielectric**, leading to gate leakage or early breakdown.
The risk increases with larger metal area and thinner gate oxides, making advanced nodes and precision analog devices particularly susceptible.

Prevention.
To prevent this, **metal hopping** is applied—routing is divided into shorter segments by inserting vias to higher or lower metal layers, thus reducing the effective exposed area.
In this work, **metal segmentation** were used for nets identified by the SG13G2 PDK's antenna check script as having potential plasma-induced charging risk.

11.3.3. Interconnecting Vias (Reliability and Noise)

Mechanism.
Vias are the vertical interconnects that connect one metal layer to another. Each via has a finite **resistance and current-carrying limit**.
If a single via is forced to conduct large currents, it experiences localized heating and **electromigration**, which can lead to open circuits or resistance drift over time. Furthermore, any asymmetry in via count or placement between matched branches creates **mismatch in series resistance**, leading to offset in analog differentials or current mirrors.
Vias also contribute small fringing capacitances, and an uneven via distribution can slightly distort signal paths, particularly at high frequency.

Prevention.
To enhance both **reliability and symmetry**, we employ **redundant via arrays**—multiple vias placed in parallel—to reduce resistance and distribute current density evenly. These are used especially on **low-impedance and high-current paths** such as power rails, bias lines, and OTA output nodes.
In matched differential structures, **via symmetry** is strictly maintained, meaning identical via count, shape, and placement on both branches. This guarantees balanced parasitic resistance and capacitance, preventing differential offset.
These measures collectively improve **electromigration margins**, ensuring the long-term reliability of the metal interconnect network.

11.3.4. Charge Injection and Gate Coupling

Mechanism.
Charge injection refers to the **unintended transfer of charge** from a switching device's channel or overlap capacitances into adjacent high-impedance nodes. When a MOSFET gate or drain experiences a fast voltage transition, the associated capacitances (C_{gd}, C_{gs}, and junction capacitances) couple a portion of the transient charge into nearby sensitive nodes.
In analog amplifiers or sampling circuits, this causes **voltage transients, bias perturbations, or signal distortion**. The effect is more pronounced in narrow-geometry devices and high-speed circuits, where transitions occur rapidly and charge redistribution is incomplete.

Prevention.
The most effective mitigation is **careful routing and shielding**.
High-impedance nodes are kept as short as possible, routed away from noisy or switching signals, and, where feasible, enclosed by **ground shields** connected to quiet references (AVSS).
In differential circuits, balanced and symmetric routing ensures that any residual injected charge appears as a **common-mode disturbance**, which is then rejected by the amplifier's differential nature.
In networks involving switching (e.g., current-steering DACs or sample-and-hold circuits), **bottom-plate switching** and **complementary transistor pairs** are used to cancel charge injection. In static bias networks, minimizing gate–drain overlap and ensuring sufficient separation from high-slew-rate lines further reduces this parasitic coupling.

In summary, understanding these parasitic mechanisms is essential for robust analog design. Each represents a different physical coupling path—through the substrate, interconnect, or process environment—and each must be addressed at layout level.
The preventive strategies applied in this work, derived from the IHP SG13G2 PDK rules and the guidelines in [5], ensure that the layout not only meets design rules but also preserves **electrical integrity and long-term reliability** in fabrication and operation.

11.4. Guard Rings (Local Substrate Sinks) - Saviour

Guard rings are a fundamental structural element for **substrate noise suppression** and **latch-up prevention** in analog and mixed-signal IC layouts.
They consist of **diffusion regions tied to power rails** that form a **low-impedance sink** for substrate currents and injected minority carriers. When nearby transistors switch, charge is injected into the substrate or well regions, creating potential fluctuations that can couple into sensitive nodes. Guard rings intercept these currents and route them to a stable potential (AVDD or AVSS), thereby **reducing parasitic coupling, improving device reliability, and stabilizing substrate potentials** [5].

In physical terms, the substrate and well behave as resistive media represented by **Rsub** (substrate resistance) and **Rwell** (well resistance).
Both parameters define the degree of electrical isolation between neighboring devices.
A high R_{sub} or R_{well} increases the voltage drop caused by substrate or well currents, thereby amplifying potential modulation and noise coupling. Guard rings **effectively lower these resistances** by providing parallel, low-resistance paths to the respective power rails.
This ensures that any injected charge is quickly dissipated, keeping local substrate potentials nearly constant.
Reducing R_{sub} and R_{well} is therefore essential not only for **latch-up prevention**, but also for **matching accuracy** in precision analog circuits, where small potential gradients can alter threshold voltages and current ratios.

In this work, **every sensitive device group**—including the differential input pair, active loads, and current mirrors—is enclosed by **fully closed rectangular guard rings**.

For NMOS devices placed in the p-substrate region, **p^+ guard rings** are tied to **AVSS (analog ground)**, while for PMOS devices inside n-wells, **n^+ guard rings** are connected to **AVDD**.
This configuration ensures that both substrate and well domains have **firm voltage references**, minimizing variations in local body potential and suppressing latch-up triggering paths. Contacts are inserted at **regular pitch** along the ring, effectively reducing the sheet resistance of the ring and enhancing current collection efficiency.

From a design standpoint, several parameters determine guard-ring effectiveness:
- **Ring width** — Wider guard rings exhibit lower resistance and better current collection but consume additional layout area.
- **Contact density** — A higher number of substrate or well contacts per unit length lowers R_{sub} and R_{well}, improving isolation.
- **Continuity** — Continuous guard rings provide stronger confinement of the local substrate potential, while segmented ones trade isolation for area and routing flexibility.

Two topological categories are widely used:

- **Continuous Guard Rings**
  These structures **completely enclose** the protected device region without any breaks.
  They offer **maximum isolation**, **lowest effective substrate resistance**, and **best protection against latch-up and substrate noise**.
  Continuous rings are preferred in **high-precision analog blocks** where low R_{sub} and R_{well} are crucial for stable operation.
  In our OTA layout, every critical block—especially the differential pair—is enclosed by **continuous rectangular guard rings**, chosen for their strong isolation and reliability.

- **Segmented Guard Rings**
  These are composed of **discrete diffusion segments** separated by intentional openings to accommodate routing or area constraints.
  While they exhibit **higher** R_{sub} and **reduced isolation** compared to continuous rings, properly designed segmented rings can still prevent latch-up and effectively attenuate substrate coupling. They are often used in moderately sensitive or space-limited blocks.

In conclusion, the guard-ring network in this layout not only suppresses substrate coupling and latch-up but also **reduces Rsub and Rwell**, anchoring the local substrate and well potentials to fixed references.
This ensures stable device behavior across temperature and process variations and significantly enhances long-term reliability of the analog core.

11.5. Block-Level Practices Implemented in This Work

- **Per-group isolation.** The **differential pair**, **active load**, and **current-mirror groups** are each enclosed by **complete guard rings** and placed inside **deep N-well/isolation** where available, creating local "quiet islands."

- **Layer strategy.** Sensitive differentials avoid **same-layer, long parallel** runs. When proximity is required, we route on M_n **vs.** M_{n+2} with adequate spacing; orthogonal adjacency between Mn and Mn+1 is preferred for crossings.

- **Decoupling. Local decoupling mosfets** (e.g., from AVDD to AVSS) are placed **close to the OTA core**, especially around the **input pair**, with short, wide connections and redundant vias inside the same isolation domain.

- **Symmetry.** Wherever matching matters, we keep **equal lengths**, **matched via stacks**, and **balanced shields** on both branches so parasitics affect each side equally.

11.6. Practical Checklist (Analog-Focused)

1. **Plan isolation first.** Assign deep-N-well/guard-ring envelopes before routing.

2. **Route high-Z nodes last.** Keep them shortest; add shields only if you can mirror them.

3. **Avoid parallelism.** No long, same-layer parallels on sensitive nets; prefer Mn vs. Mn+2.

4. **Use via arrays.** Add redundancy and enforce symmetry across matched paths.

5. **Place decaps locally.** Near the most sensitive block (here, the differential pair).

6. **Re-run checks often.** DRC + antenna + density after small edits; extract and inspect parasitics on critical nodes early.

Pro Tip — Parasitics Are Design Variables

Treat coupling capacitances, substrate returns, and via resistances as **designable quantities**. Early floorplanning of isolation, ring continuity, layer assignment, and decap placement saves orders of magnitude in late-stage debugging and protects analog performance margins [5].

12. Final Layout Steps — Bonding, Seal Ring, and Pad-Frame Integration

After completing device-level layout, parasitic-aware routing, and full DRC/LVS verification, the final stage of physical design is the construction of the **pad frame** and the **seal ring**.
These structures are the physical interface between the silicon die and the outside world. Although electrically simple, they are mechanically and parasitically critical, as they determine whether the fabricated chip can be safely diced, bonded, packaged, and measured.
In the SG13G2 process, the IHP-PDK provides a fully parameterized library of PCells for bond pads, ESD structures, and seal rings, which we used and adapted for our layout.

12.1. The Seal Ring: Mechanical and Electrical Protection

The **seal ring** is a continuous frame of thick metal that surrounds the entire active layout region. Its primary function is **mechanical protection** during wafer dicing: when the wafer is cut, cracks can propagate inward from the die edge. The seal ring absorbs these stresses and prevents them from reaching active devices.
Beyond its mechanical role, the seal ring also acts as a **low-impedance electrical shield**, tied to AVSS, providing:

- protection against environmental contamination and moisture diffusion,
- suppression of substrate noise entering from the die periphery,
- confinement of parasitic surface currents.

In the SG13G2 PDK, the seal ring is provided as a **P_Cell** (parameterized cell), built from stacked metal layers and dense via walls. Its geometry is fully adjustable—width, margin, and spacing can be tuned—allowing the designer to adapt it to the die dimensions while maintaining foundry compliance.
For this work, we instantiated and customized the provided seal ring PCell and placed it around the complete pad frame, ensuring full mechanical and electrical enclosure of the analog OTA core.

12.2. Bond Pads and External Interfacing

12.2.1. 2.1 Purpose and Structure

Bond pads form the electrical interface between the on-chip metal layers and the external package or measurement environment.
Each pad is a large, top-metal landing area with a passivation opening, designed to withstand mechanical bonding forces while maintaining low series resistance.
To minimize parasitic disturbance, each functional bond pad was further **surrounded by a dummy pad frame**—a ring of electrically inactive pads. These dummy pads act as protective buffers: if parasitic displacement currents or surface leakage occur near the die edge, they discharge into the dummy pads rather than coupling into the functional I/O pads. This arrangement reduces the susceptibility of the pad ring to noise and preserves the integrity of the sensitive analog connections.[6]

The placement of pads was also optimized to minimize parasitic capacitive coupling to the underlying analog core. Pads were spaced to avoid long parallel overlaps with internal routing, and their distribution was balanced to avoid asymmetries in substrate stress and top-metal loading.

12.3. Bonding Techniques: Wire Bonding vs. Flip-Chip

Two principal bonding methods exist in industry, each with distinct process and performance characteristics.

12.3.1. Wire Bonding

Wire bonding uses thin gold or aluminum wires, thermosonically welded between the pad and the package lead.
It is widely used for low- to medium-pin-count analog ICs because:

- it requires minimal packaging infrastructure,
- pads can be placed on the periphery,
- the process is accessible and cost-effective,
- bare dies can be bonded directly without redistribution layers.

For this project—intended for bare-die evaluation—**wire bonding** was the natural choice and fully supported by the SG13G2 pad library.

12.3.2. Flip-Chip Bonding

Flip-chip uses solder microbumps on the die surface, flipping the die face-down onto the substrate.
It offers extremely low-inductance interconnects but demands:

- redistribution layers,
- specialized bump-formation equipment,
- underfill materials,
- and advanced assembly infrastructure.

These requirements go beyond the scope of a prototype analog OTA; therefore, flip-chip pads were not used.

12.4. Pad Geometry: Orthogonal vs. Square Pads

Bond pad geometry influences both mechanical reliability and parasitic loading.
Classical square pads are straightforward but can concentrate mechanical stress at their sharp corners during bonding. To mitigate this, the SG13G2 PDK also offers **orthogonal (octagonal) pads**, which soften corner transitions and distribute force more evenly across the pad boundary.
They inherently reduce the tendency for:

- metal delamination,
- passivation cracking,
- and edge-induced parasitic capacitive coupling.

In this design, **orthogonal octagonal pads** were selected for all I/O connections, ensuring robust bonding and improved parasitic behavior relative to simple square geometries.

12.5. ESD Protection

Electrostatic discharge (ESD) can destroy sensitive analog devices long before a chip is ever powered in the lab.
To prevent this, the SG13G2 PDK provides a series of **ESD protection PCells**, including:

- diode clamps,
- grounded-gate NMOS structures

PCells are invaluable because they embed **foundry-validated device geometries**, with correct well biasing, spacing rules, finger structures, and metal enclosures. Designing ESD structures manually would risk violating subtle DRC rules—particularly those related to well spacing, diffusion density, and metal overlap.
By using the PDK-provided PCells, the pad frame automatically complies with:

- ESD breakdown voltage limits,
- safe discharge paths during bonding and handling,
- and all SG13G2-specific reliability constraints.

Although not all ESD structures were required for our simple OTA test chip, their availability ensured that the pad frame could be expanded into a full mixed-signal environment if needed.

12.6. Density Fill Structures (Metal, Active, and Poly Fillers)

The final step before tape-out is the insertion of **density fill structures**, often simply referred to as *fillers*. Modern fabrication processes require that each metal, diffusion, and polysilicon layer maintain a specified **pattern density** within every local window of the layout. These density rules prevent issues such as erosion, and non-uniform planarization during CMP (Chemical–Mechanical Polishing), which can adversely affect metal thickness, resistance, and interlayer dielectric integrity.

Fillers are non-functional geometries—typically small, repeatable patterns of **metal**, **active diffusion**, and **gate poly**—that ensure the overall layout meets the foundry's density requirements without altering circuit behavior.
They are deliberately placed far from sensitive analog nodes or designed with grounded landing points so that any residual capacitive effect remains negligible. In analog circuits, filler insertion must be performed carefully to avoid parasitic loading of high-impedance nodes or unintentional coupling to bias networks.

The IHP-SG13G2 PDK provides automated filler-generation utilities that greatly simplify this process.
Filling can be performed directly within **KLayout** using built-in irreversible function, or, more robustly, through a **Python script supplied by the PDK**, which analyzes layer densities and inserts the appropriate fill shapes according to process-specific rules.
The script outputs a final **GDS2 file** in which all required fills—metal, active, and poly—have been inserted and you have to verify against the SG13G2 density constraints.

This finalized and verified, density-compliant GDS2 file represents the **manufacturing-ready layout**, suitable for submission to the foundry in compliance with all CMP and density specifications.

Part IV.

Post-Layout Simulation

13. Post layout simulation.

We have established testbenches for DC, AC, and transient analysis using the symbol of the designed inverter as a subcircuit. The next step is to perform post-layout simulation by replacing the schematic netlist with the extracted layout netlist (PEX), in order to verify whether the performance remains consistent after layout parasitics are included.

Firstly we need to open the symbol file in Xschem, edit the properties of the symbol See Figure 13.1. Change from `subcircuit` to `primitive`.

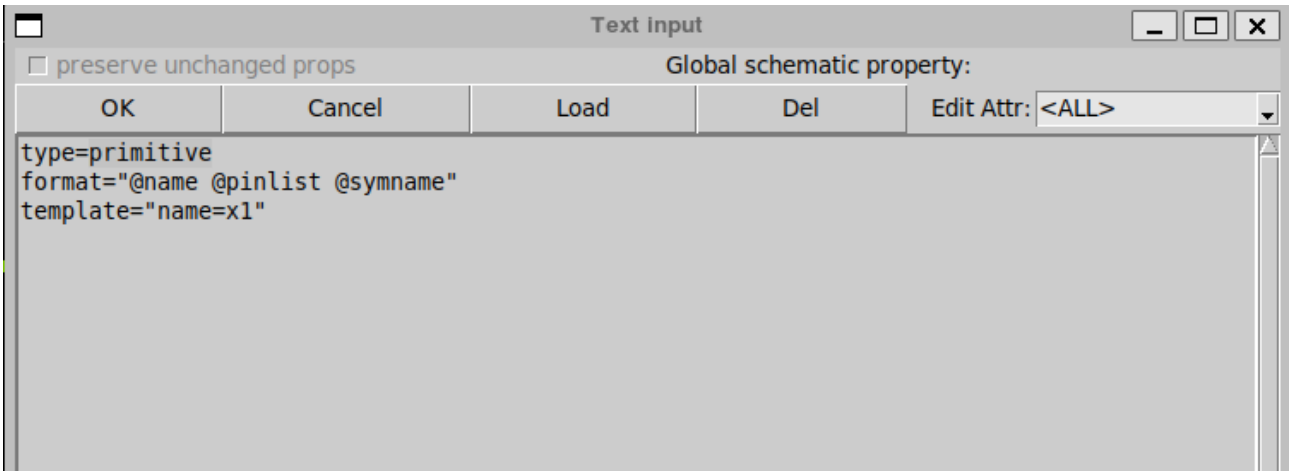


Figure 13.1.: Primitive

Once changed to primitive, now we can include the `.spice` file or `.cir` file from layout to the testbench. See Figure 13.2

Once the file is declared, we can run the testbench.

> **!** Correct pin order
>
> It is critically important to match the order of pins and their names. The pin order and names in the `.cir` files need to be exact same as included in the testbench. See the figures mentioned below

13. Post layout simulation.

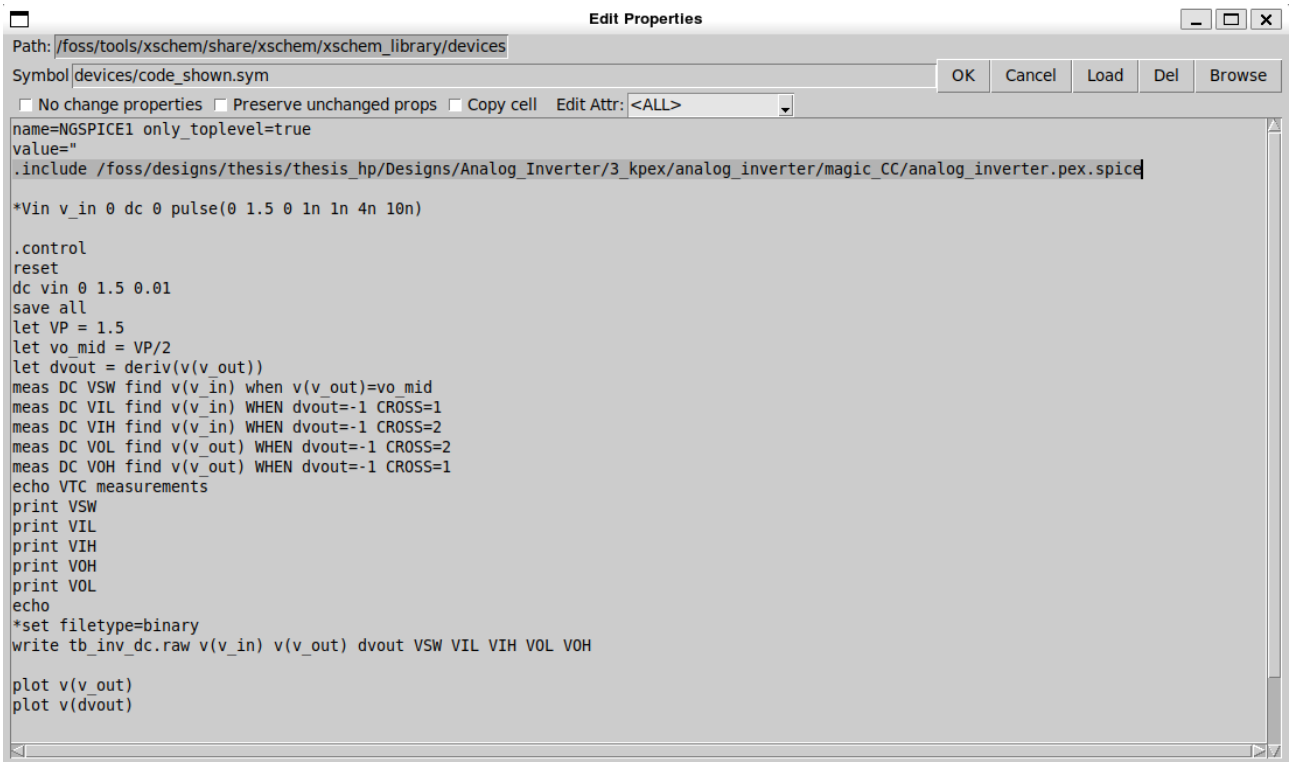
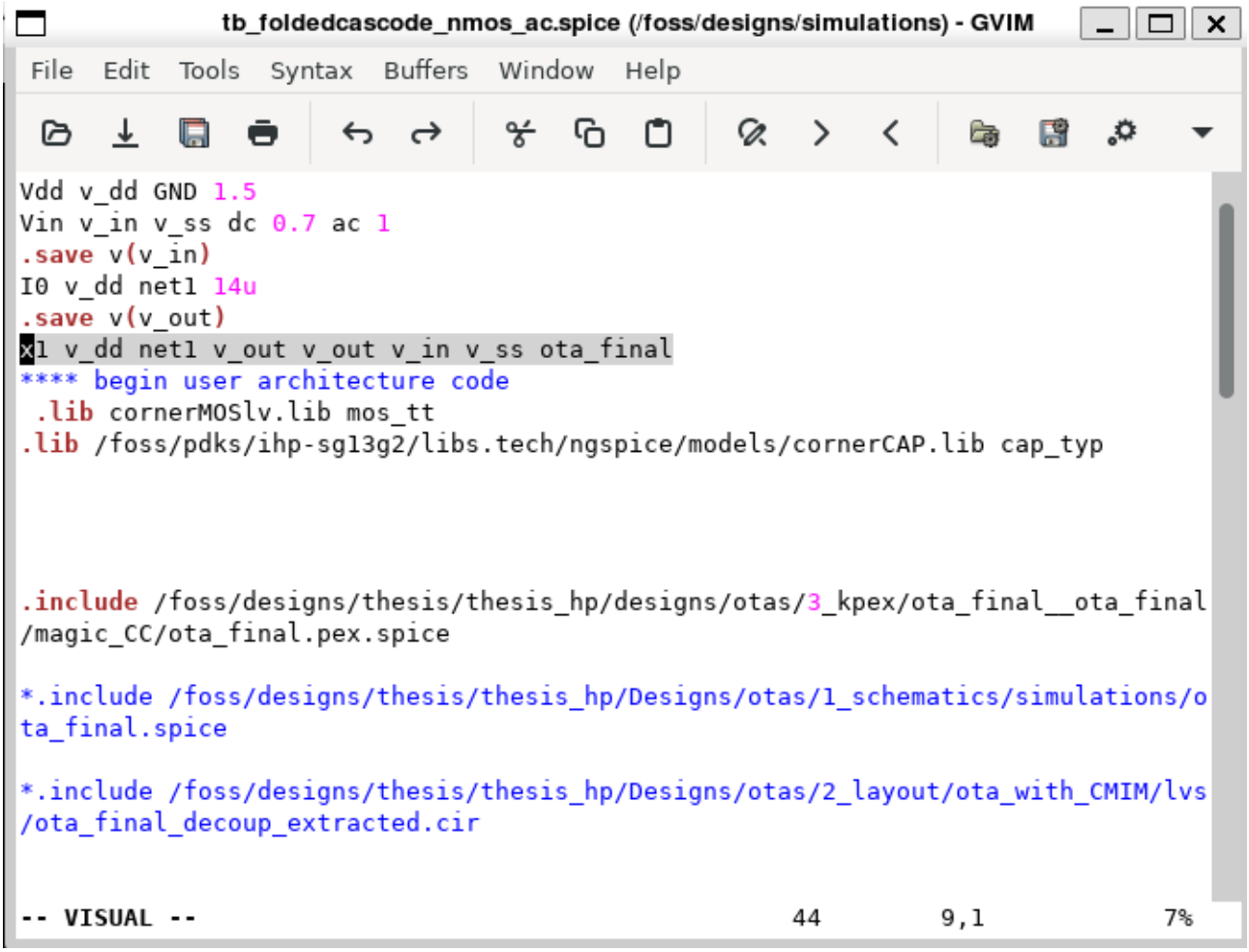


Figure 13.2.: Include


tb_foldedcascode_nmos_ac.spice (/foss/designs/simulations) - GVIM
File Edit Tools Syntax Buffers Window Help
Vdd v_dd GND 1.5
Vin v_in v_ss dc 0.7 ac 1
.save v(v_in)
I0 v_dd net1 14u
.save v(v_out)
x1 v_dd net1 v_out v_out v_in v_ss ota_final
**** begin user architecture code
.lib cornerMOSlv.lib mos_tt
.lib /foss/pdks/ihp-sg13g2/libs.tech/ngspice/models/cornerCAP.lib cap_typ
.include /foss/designs/thesis/thesis_hp/designs/otas/3_kpex/ota_final__ota_final
/magic_CC/ota_final.pex.spice
*.include /foss/designs/thesis/thesis_hp/Designs/otas/1_schematics/simulations/o
ta_final.spice
*.include /foss/designs/thesis/thesis_hp/Designs/otas/2_layout/ota_with_CMIM/lvs
/ota_final_decoup_extracted.cir
-- VISUAL -- 44 9,1 7%

13. Post layout simulation.

```
* Extracted by KLayout with SG13G2 LVS runset on : 14/07/2025 15:15

.SUBCKT ota_final AVDD IBIAS VOUT MINUS PLUS AVSS
M$1 AVSS AVSS \$75 AVSS sg13_lv_nmos L=1u W=6u AS=1.59p AD=1.59p PS=10.06u
+ PD=10.06u
M$2 \$75 MINUS \$62 AVSS sg13_lv_nmos L=1u W=6u AS=1.14p AD=1.14p PS=6.76u
+ PD=6.76u
M$3 \$62 PLUS \$71 AVSS sg13_lv_nmos L=1u W=6u AS=1.14p AD=1.14p PS=6.76u
+ PD=6.76u
M$4 \$71 AVSS AVSS AVSS sg13_lv_nmos L=1u W=6u AS=1.59p AD=1.59p PS=10.06u
+ PD=10.06u
M$5 IBIAS \$55 \$56 AVSS sg13_lv_nmos L=0.13u W=1u AS=0.34p AD=0.34p PS=2.68u
+ PD=2.68u
M$6 AVSS \$56 IBIAS AVSS sg13_lv_nmos L=5u W=8.5u AS=1.93375p AD=1.93375p
+ PS=12.445u PD=12.445u
M$10 AVDD VOUT \$22 AVSS sg13_lv_nmos L=0.5u W=15u AS=3.4125p AD=3.4125p
+ PS=20.57u PD=20.57u
M$14 AVSS \$56 \$63 AVSS sg13_lv_nmos L=5u W=8.5u AS=1.93375p AD=1.93375p
+ PS=12.445u PD=12.445u
M$18 AVSS \$3 \$2 AVSS sg13_lv_nmos L=0.13u W=1u AS=0.34p AD=0.34p PS=2.68u
+ PD=2.68u
M$23 AVSS \$56 \$62 AVSS sg13_lv_nmos L=5u W=8.5u AS=1.93375p AD=1.93375p
+ PS=12.445u PD=12.445u
M$27 \$63 \$73 \$65 AVDD sg13_lv_pmos L=2u W=14u AS=3.185p AD=3.185p PS=19.32u
+ PD=19.32u
M$31 AVSS \$73 \$66 AVDD sg13_lv_pmos L=0.5u W=14u AS=3.185p AD=3.185p
```

Post layout and prelayout graphs to be added here as results

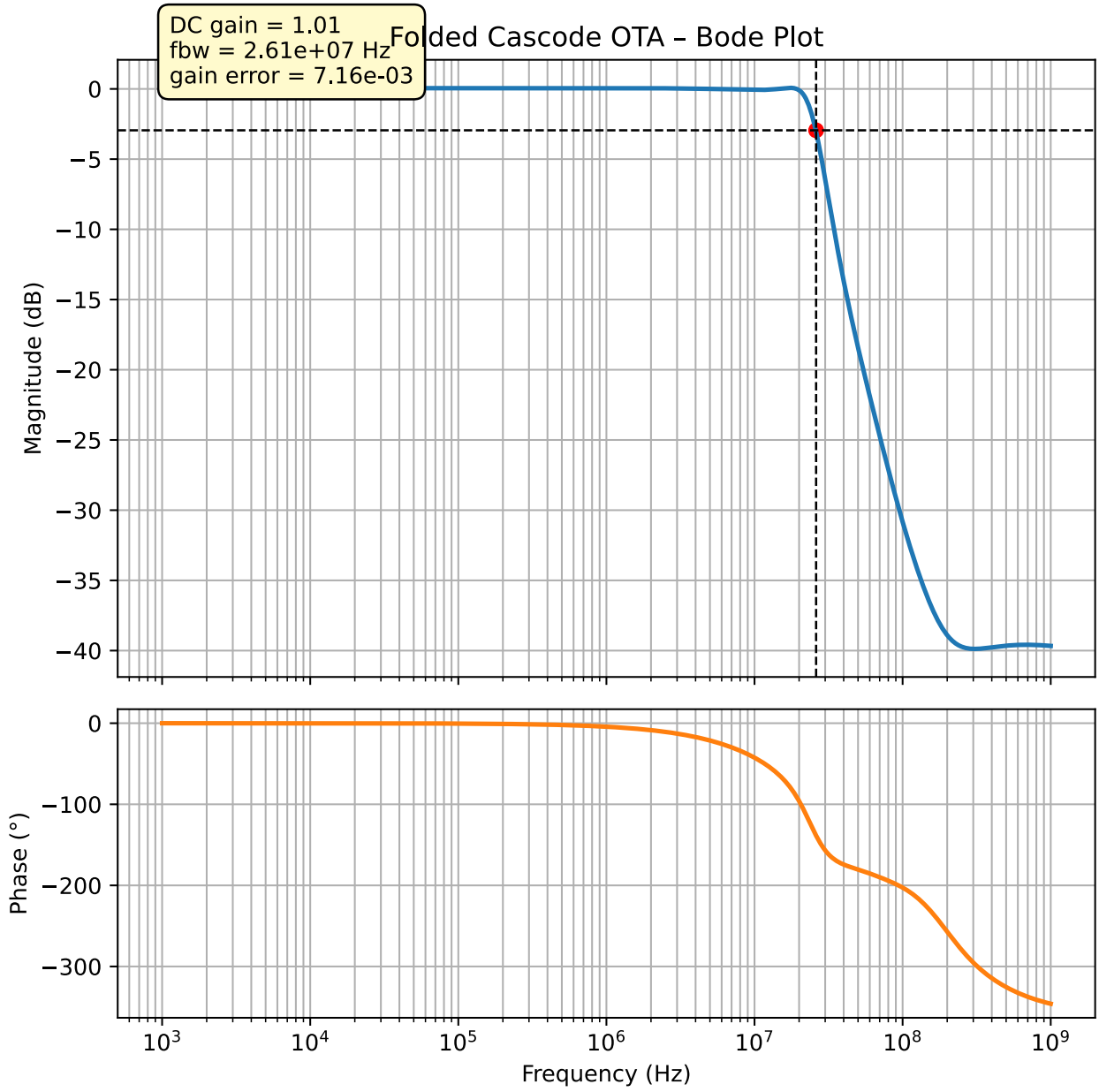


Figure 13.3.: Postlayout Results

14. PEX

Parasitic Extraction (PEX) is a critical step in analog IC design that occurs **after layout** and **before final verification/tapeout**. It models the unintended resistive and capacitive effects introduced by routing, device placement, and proximity in silicon.

In analog circuits — especially precision blocks like **folded cascode OTAs** — these parasitics significantly affect performance, making PEX-based simulations essential.

14.1. Why Do We Need PEX?

Reason	Description
Performance Degradation	Parasitic capacitance reduces bandwidth and phase margin.
Offset and Mismatch Effects	Parasitic coupling causes imbalance in differential paths.
Stability Impact	Additional phase lag from parasitics can destabilize feedback loops.
Post-Layout Verification	Confirms that layout did not violate design intent (gain, UGB, PSRR, etc.).
Tapeout Confidence	Ensures high correlation between simulation and silicon behavior.

For a folded cascode OTA, RC + CC extraction is essential. It ensures layout-induced degradation is captured early in simulation, enabling confident tapeout. Neglecting coupling capacitance or interconnect resistance can lead to significant performance loss or instability in silicon.

14.2. PEX using Klayout-PEX tool

IIC-OSIC-Tools comes with pre-installed tool called as **K-pex**. The current status of KLayout-PEX says as following:

Available KLayout PEX Engines:

Engine	PEX Type	Status	Description
`KPEX/MAGIC`	CC, RC	Usable	Wrapper engine, using installed `magic` tool
`KPEX/FasterCap`	CC	Usable, *pending QA*	Field solver engine using `FasterCap`
`KPEX/FastHenry2`	R, L	Planned	Field solver engine using `FastHenry2`

Engine	PEX Type	Status	Description
KPEX/2.5D	CC	Under construction	Prototype engine implementing MAGIC concepts/formulas with KLayout means
KPEX/2.5D	R, RC	Planned	Prototype engine implementing MAGIC concepts/formulas with KLayout means

14.3. Running the KPEX/MAGIC Engine

The magic section of `kpex --help` describes the arguments and their defaults. Important arguments:

- `--magicrc`: specify location of the `magicrc` file
- `--gds`: path to the GDS input layout
- `--magic`: enable magic engine

14.3.1. Example Command

```
kpex --pdk ihp_sg13g2 --magic --gds GDS_PATH --out_dir OUTPUT_DIR_PATH
```

more to kpex can be found under the command `kpex --help`

```
`kpex --help` command gives us the complete list of useful commands in order to understand the
```

Once PEX is succesfully done, the extracted netlist SPICE file will be generated in the provided output path, which consists of extracted parasitics which can be further used to do post layout simulations and verify whether the design layout satisfies the design specifications or not.

14.4. Observations and Practical Results

During this work, the 2025.05 Docker release of IIC-OSIC-Tools was used, since the design was submitted on 21 August 2025, and subsequent debugging was performed in that version. When running the FasterCap engine, the process executed successfully; however, the generated `.pex.spice` file was empty, indicating that no valid parasitic data were written. Conversely, when using the MAGIC engine, a `.pex.spice` file was generated but contained improperly scaled parameters, leading to simulation inconsistencies.

Upon investigation of the official IIC-OSIC-Tools release notes, it was confirmed that in the `2025.07` update, the developers temporarily removed klayout-pex due to incompatibility with several dependencies.

This subsequent removal confirms that the encountered issues in the 2025.05 environment were indeed genuine tool-level compatibility problems, not user-level errors. Therefore, for this design, PEX-based post-layout

simulations were omitted, and circuit validation was based on pre-layout verification complemented by manual parasitic estimation from layout dimensions.

Bibliography

[1] B. Wicht, "Analog building blocks of dc-dc converters: Examining fundamental concepts," *IEEE Solid-State Circuits Magazine*, vol. 12, no. 3, pp. 42–47, 2020, doi: 10.1109/mssc.2020.3002141.

[2] P. G. A. Jespers and B. Murmann, *Systematic design of analog CMOS circuits: Using pre-computed lookup tables*. Cambridge University Press, 2017.

[3] I. PDK, "SG13G2 IHP open source layout rules," Leibniz Institute for High Performance Microelectronics, Design Reference, Dec. 2024.

[4] R. J. Baker, *CMOS: Circuit design, layout, and simulation*, 3rd ed. Wiley-IEEE Press, 2010, p. 1208. doi: 10.1002/9780470891179.

[5] J. Lienig and J. Scheible, *Fundamentals of layout design for electronic circuits*. Springer International Publishing, 2020. doi: 10.1007/978-3-030-39284-0.

[6] B. Razavi, *Design of analog CMOS integrated circuits*. McGraw-Hill, 2011.

[7] B. Razavi, *Design of analog CMOS integrated circuits*. McGraw-Hill, 2001.

A. CACE Summary for foldedcascode_nmos

netlist source: schematic

Parameter	Tool	Result	Min Limit	Min Value	Typ Target	Typ Value	Max Limit	Max Value	Status
Output voltage ratio	ngspice	gain	0.98 V/V	0.996 V/V	any	0.999 V/V	1.1 V/V	1.000 V/V	Pass
Bandwidth	ngspice	bw	1e6 Hz	5118320.000 Hz	any	7827360.000 Hz	any	13271000.000 Hz	Pass
Output voltage ratio (MC)	ngspice	gain_mc	any	0.671 V/V	any	0.996 V/V	any	1.502 V/V	Pass
Bandwidth (MC)	ngspice	bw_mc	1e6 Hz	1024950.000 Hz	any	7454465.000 Hz	any	91913200.000 Hz	Pass
Output noise	ngspice	noise	any	0.069 mV	any	0.101 mV	0.2 mV	0.134 mV	Pass
Settling time	ngspice	tsettle	any	0.259 us	any	0.287 us	1.5 us	0.320 us	Pass

A. CACE Summary for foldedcascode_nmos

A.1. Plots

A.2. gain_vs_temp

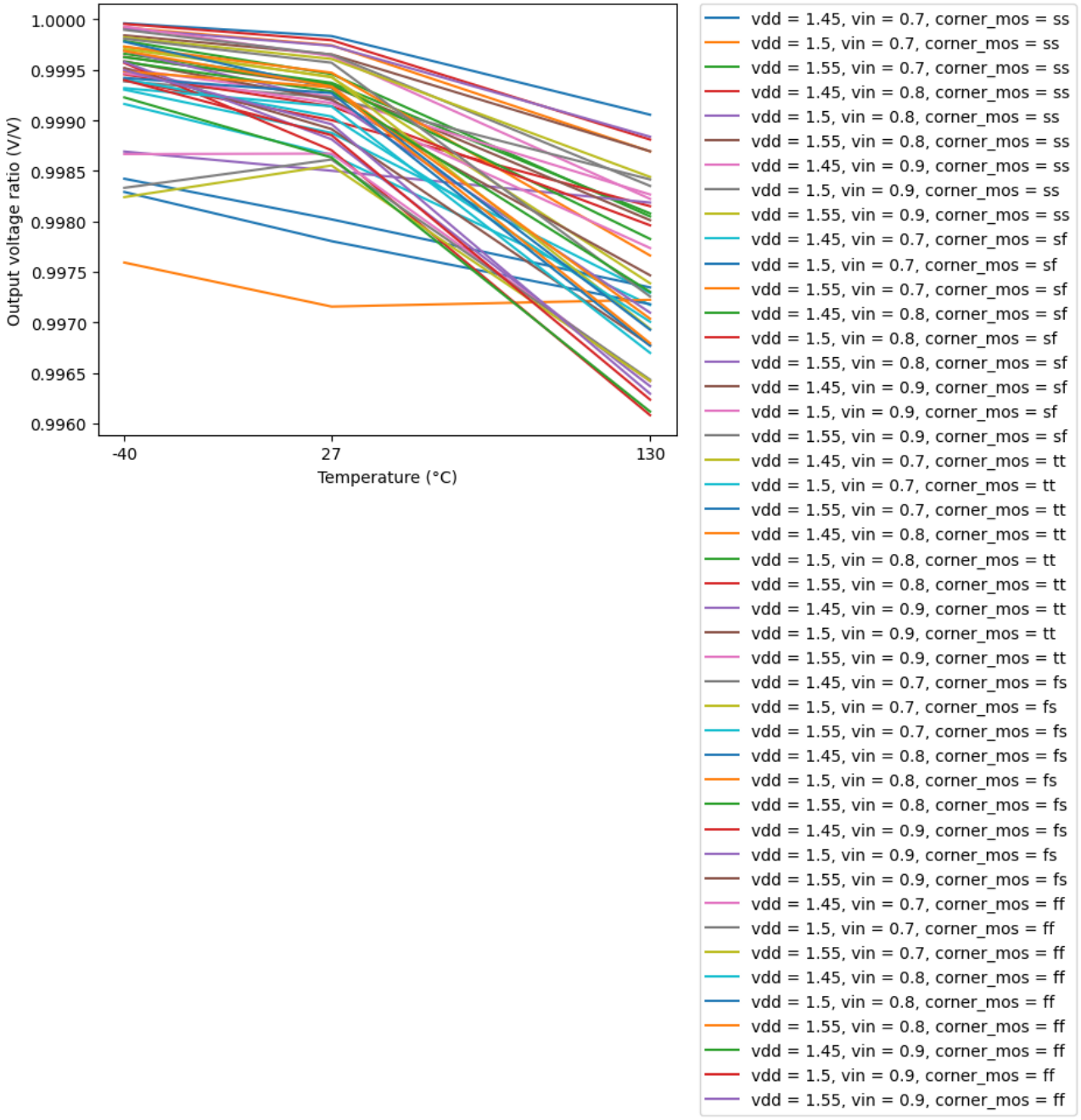


Figure A.1.: gain_vs_temp

A.3. gain_vs_vin

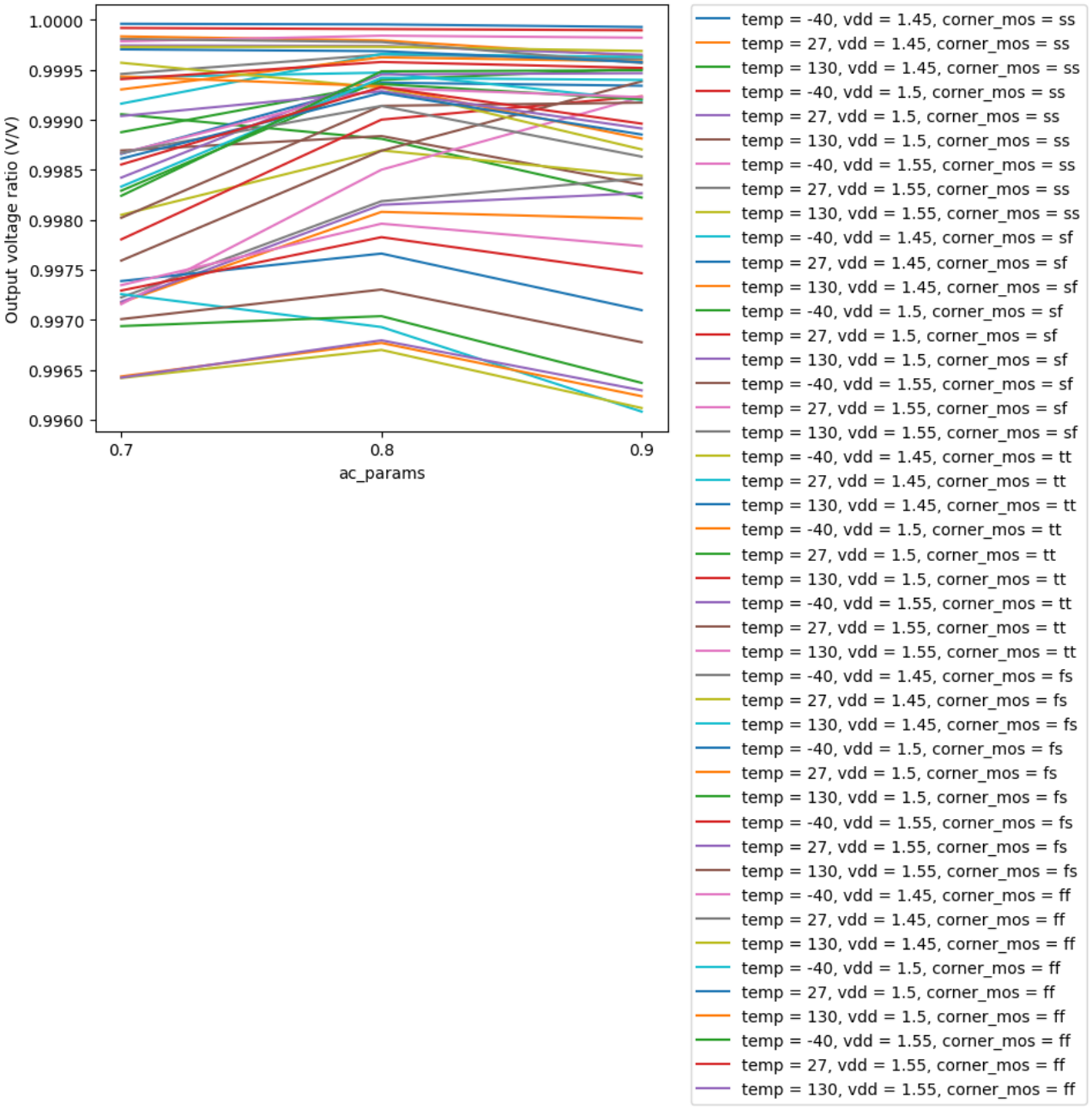


Figure A.2.: gain_vs_vin

A.4. gain_vs_vdd

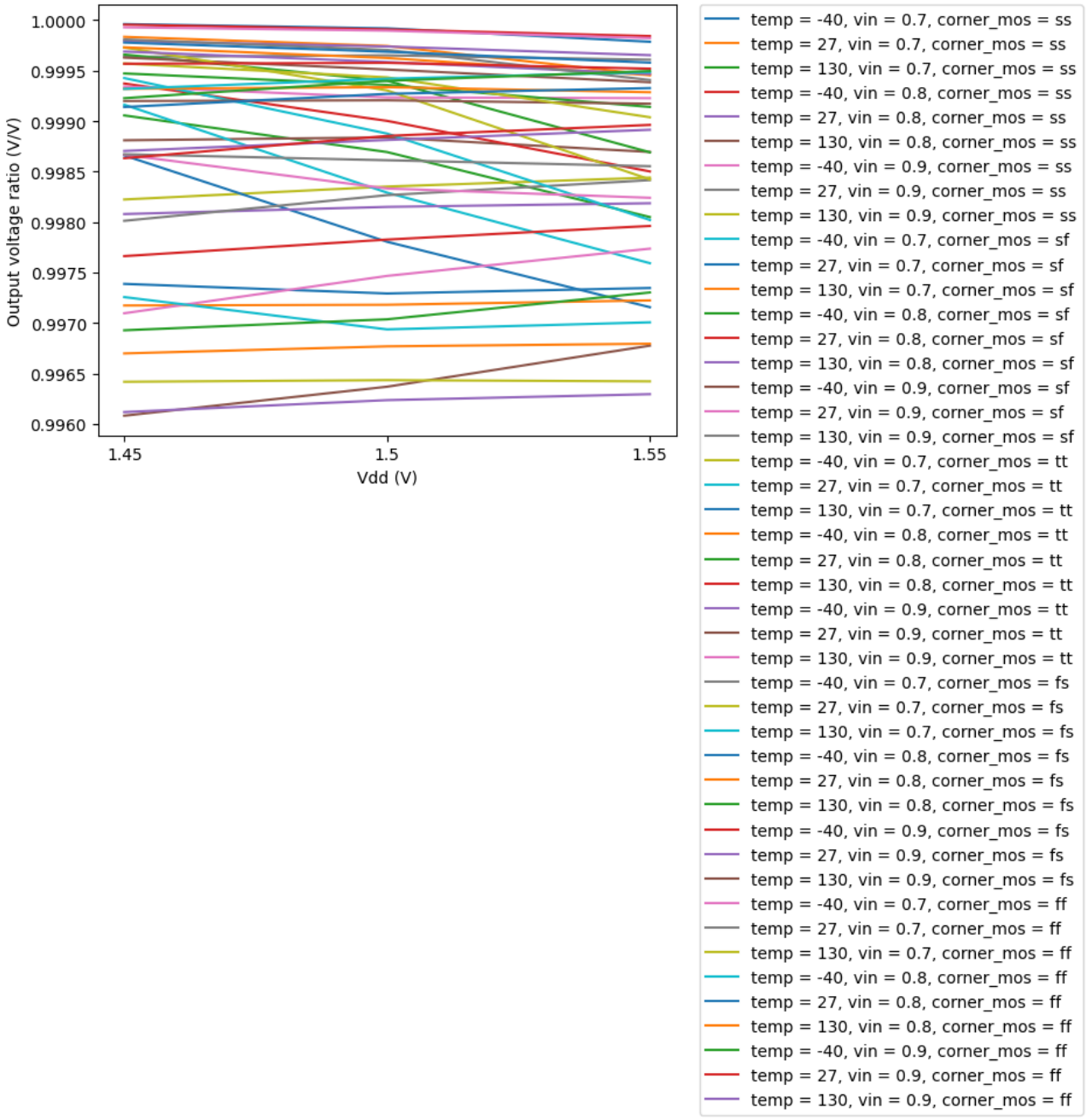


Figure A.3.: gain_vs_vdd

A.5. gain_vs_corner

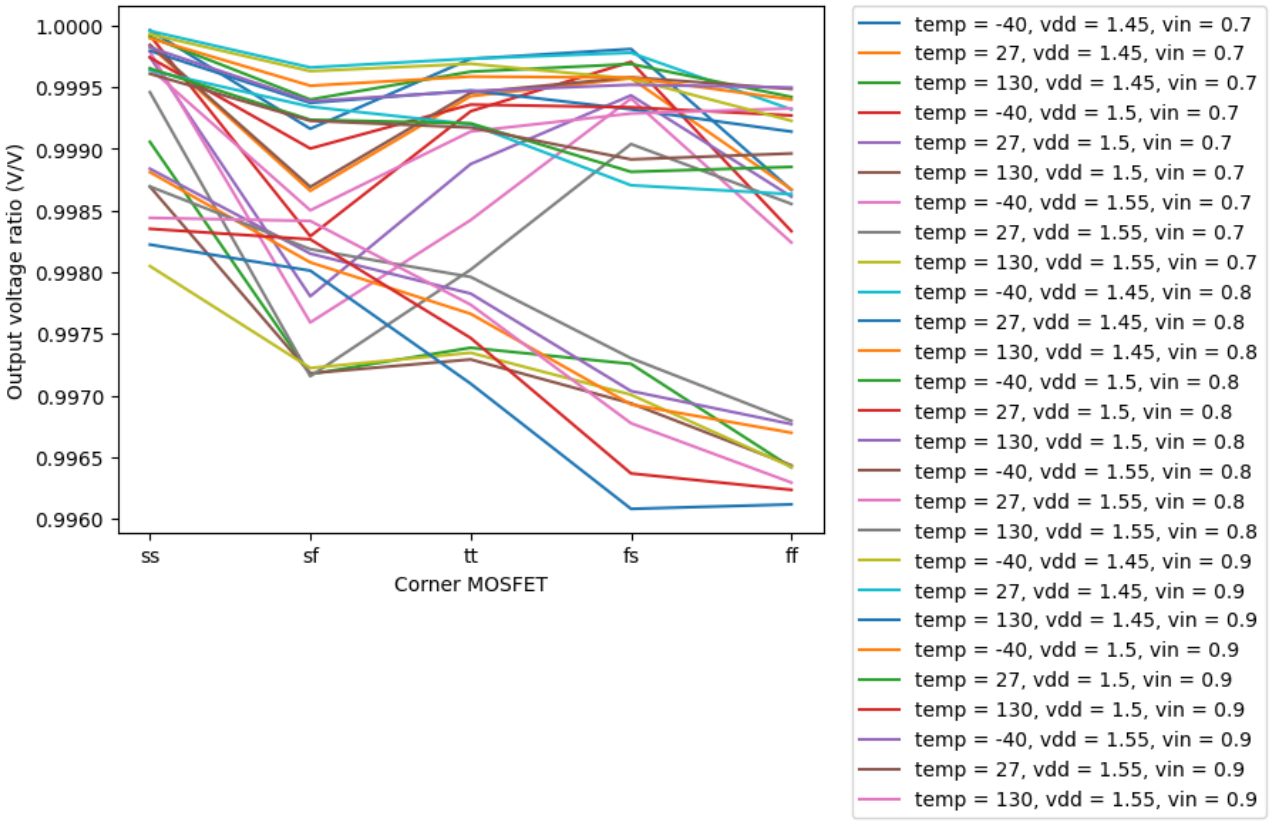


Figure A.4.: gain_vs_corner

A.6. bw_vs_temp

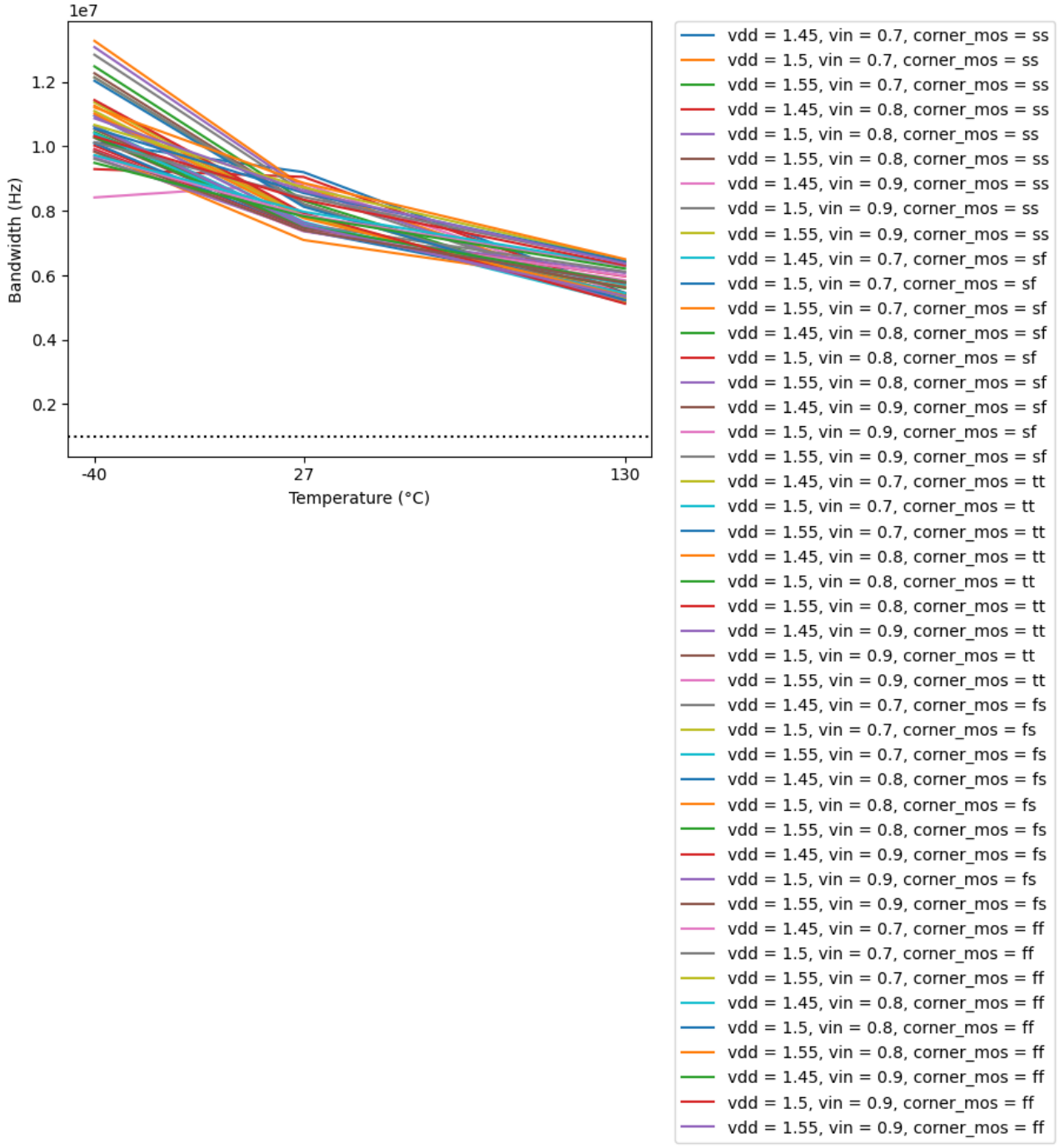


Figure A.5.: bw_vs_temp

A.7. bw_vs_vin

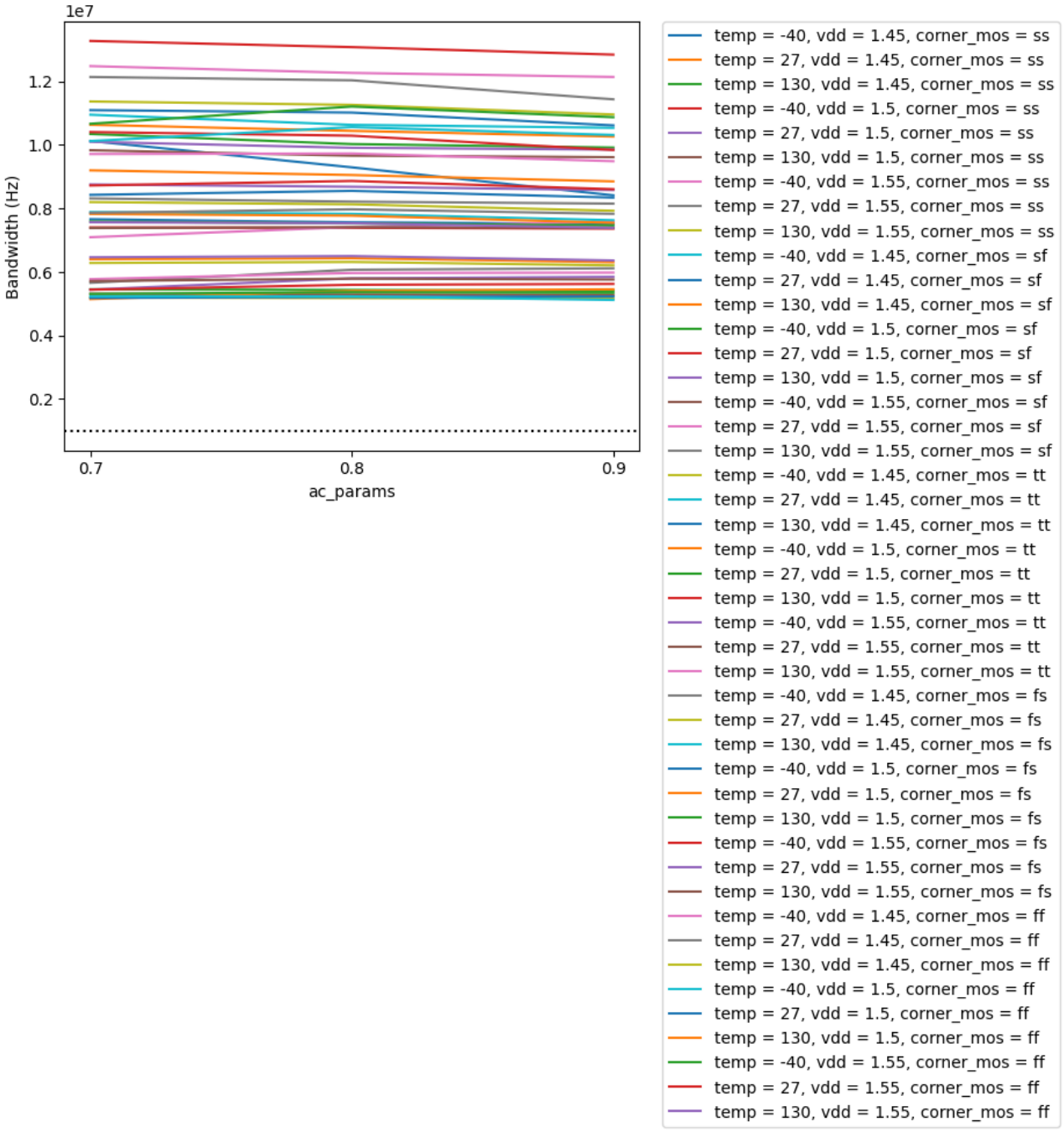


Figure A.6.: bw_vs_vin

A.8. bw_vs_vdd

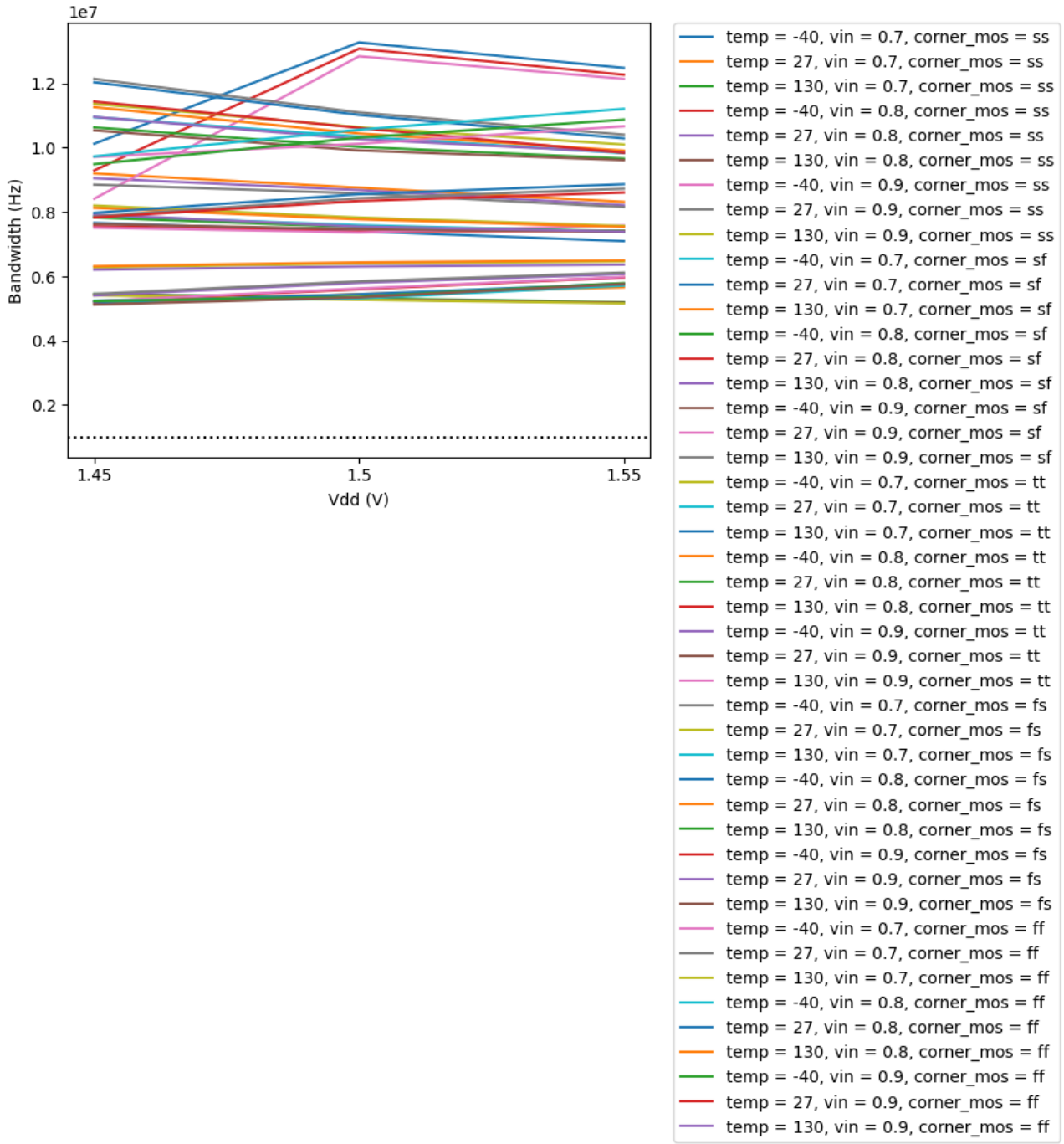


Figure A.7.: bw_vs_vdd

A.9. bw_vs_corner

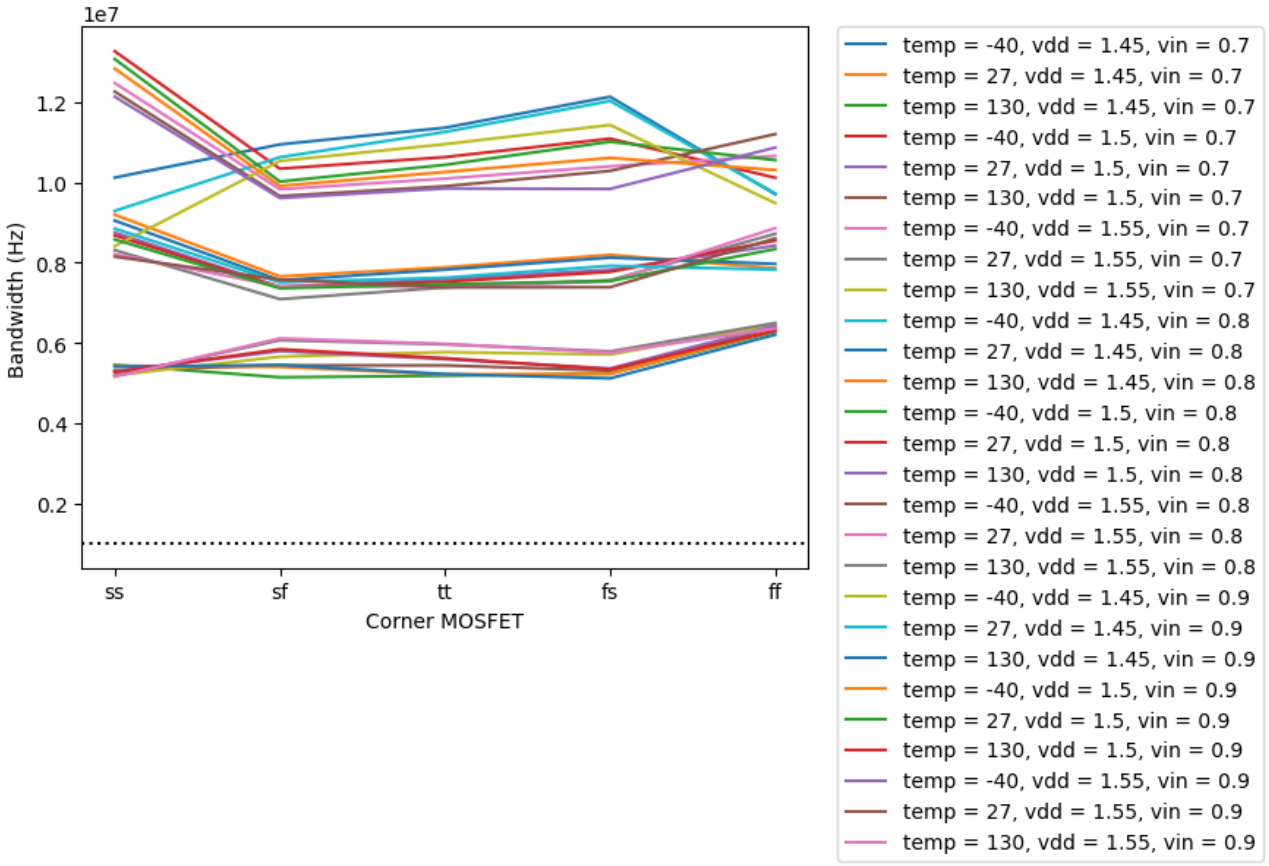


Figure A.8.: bw_vs_corner

A.10. gain_mc

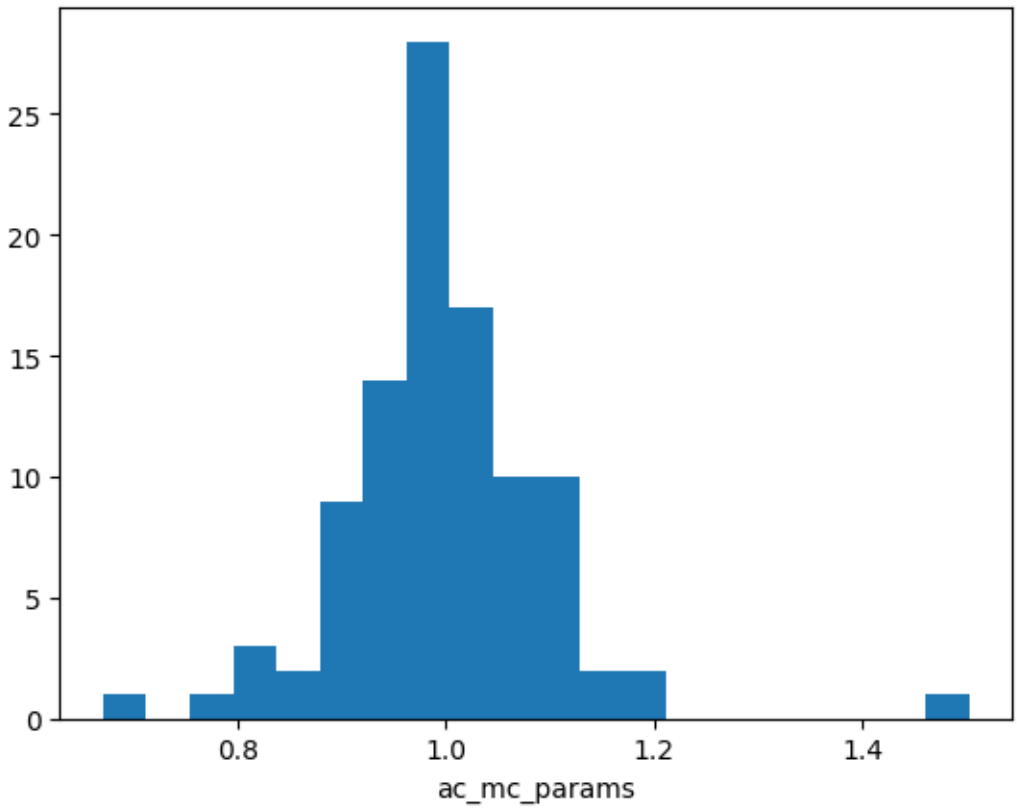


Figure A.9.: gain_mc

A.11. bw_mc

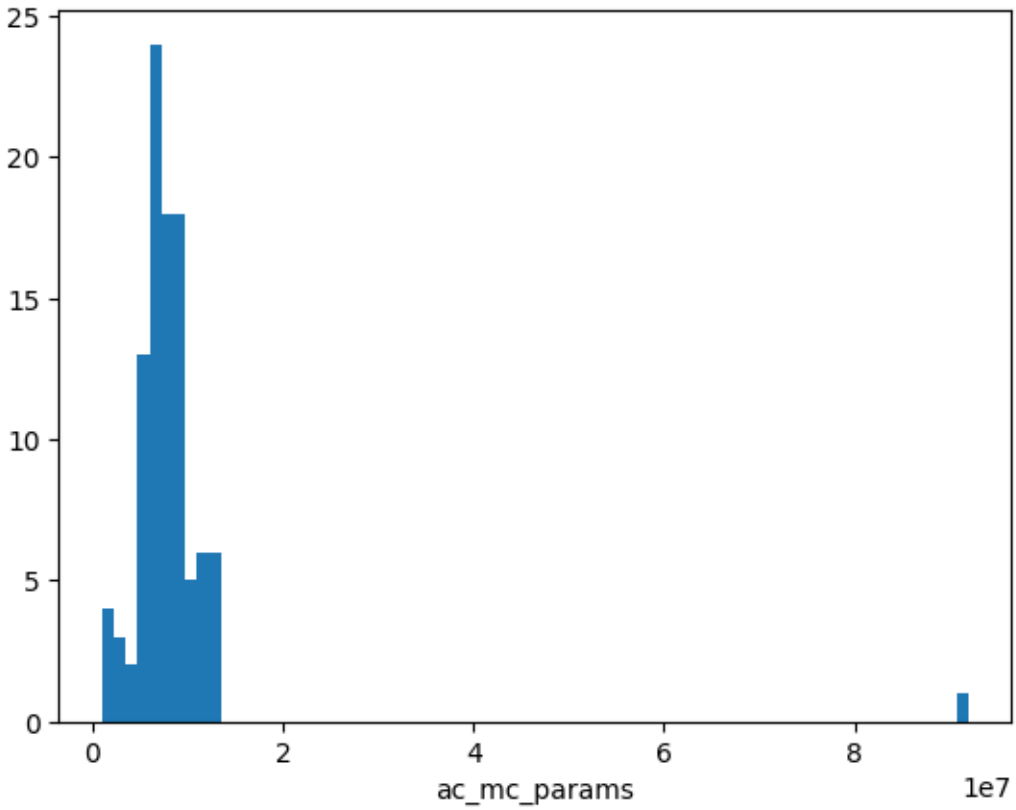


Figure A.10.: bw_mc

A.12. noise_vs_temp

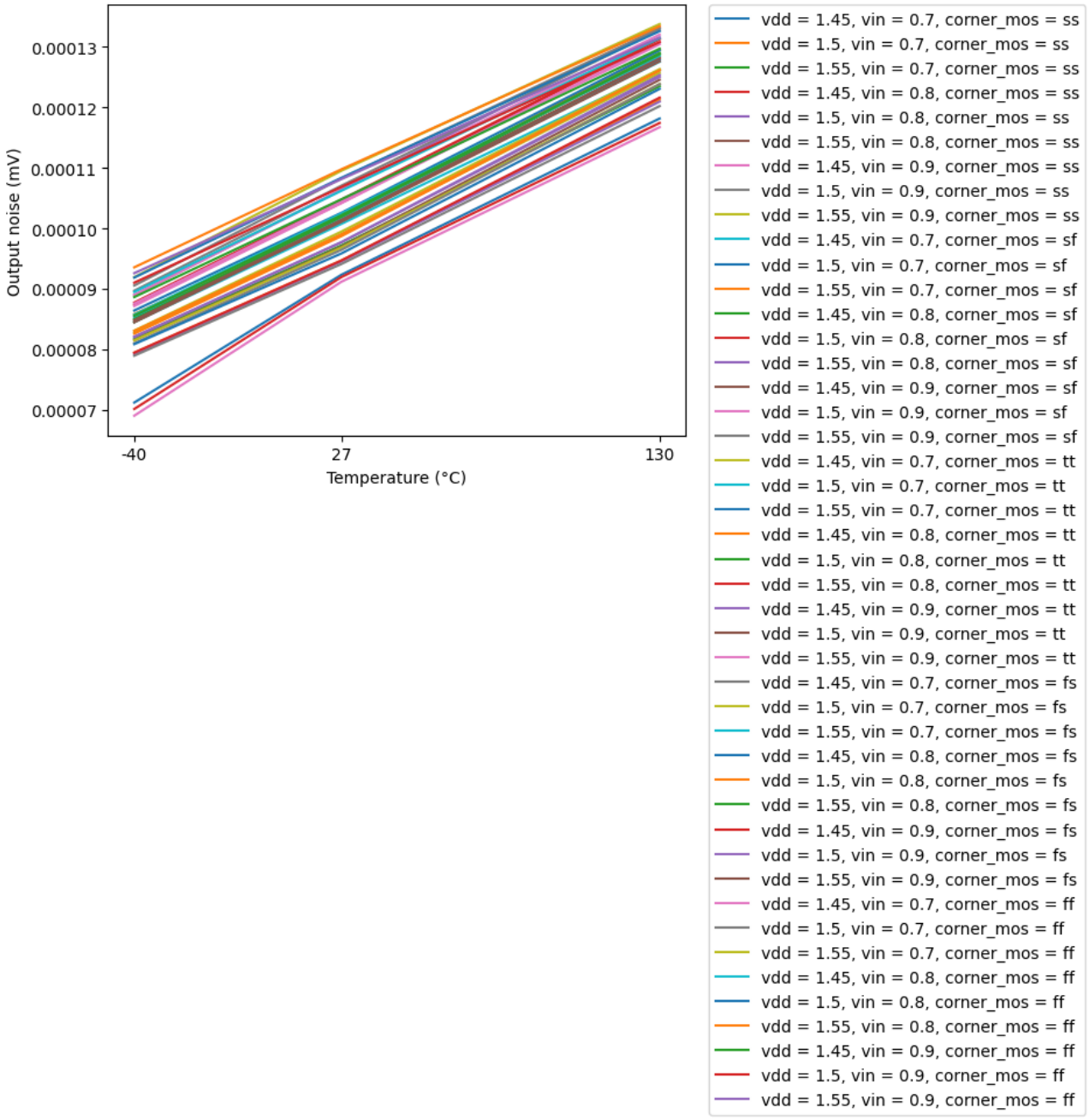


Figure A.11.: noise_vs_temp

A.13. noise_vs_vin

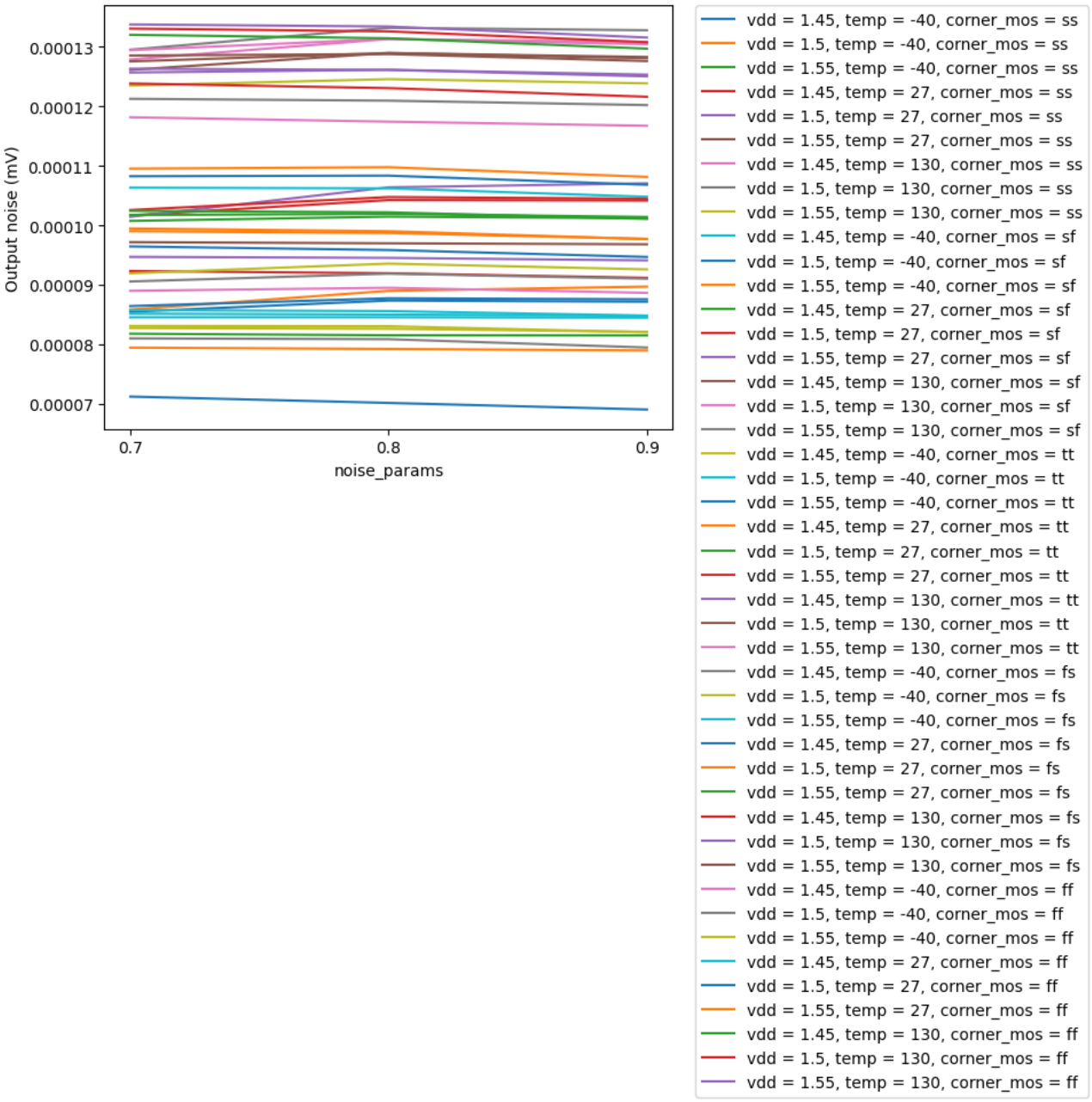


Figure A.12.: noise_vs_vin

A.14. noise_vs_vdd

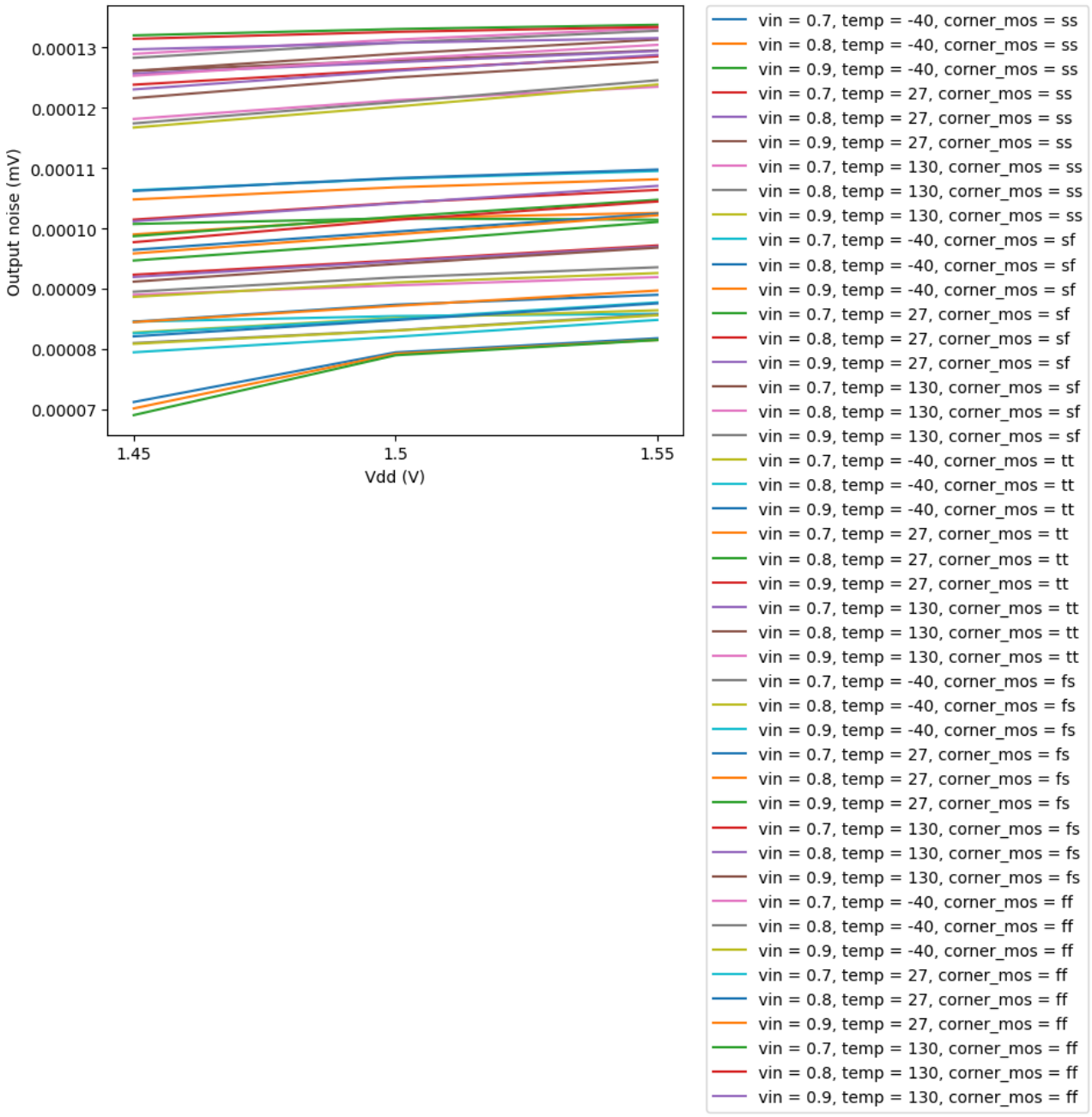


Figure A.13.: noise_vs_vdd

A.15. noise_vs_corner

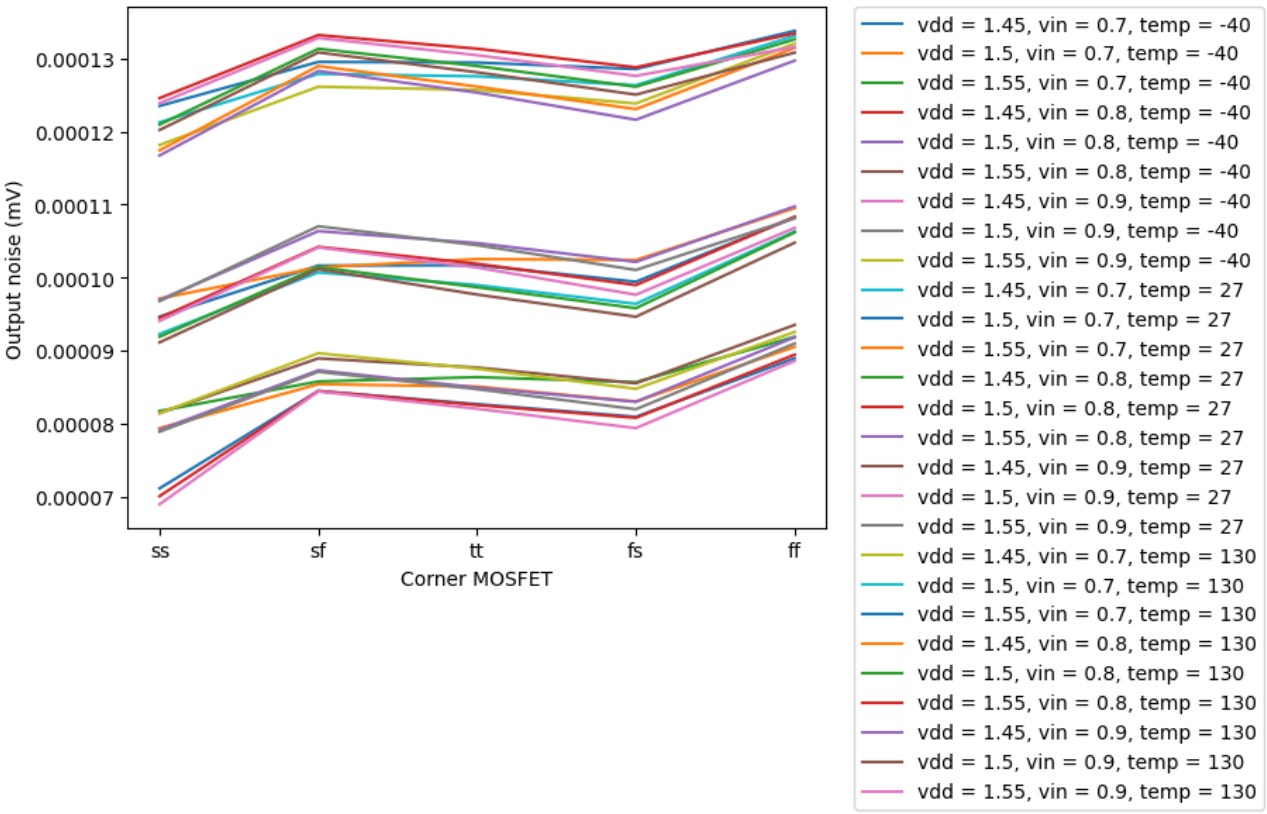


Figure A.14.: noise_vs_corner

A.16. settling_vs_temp

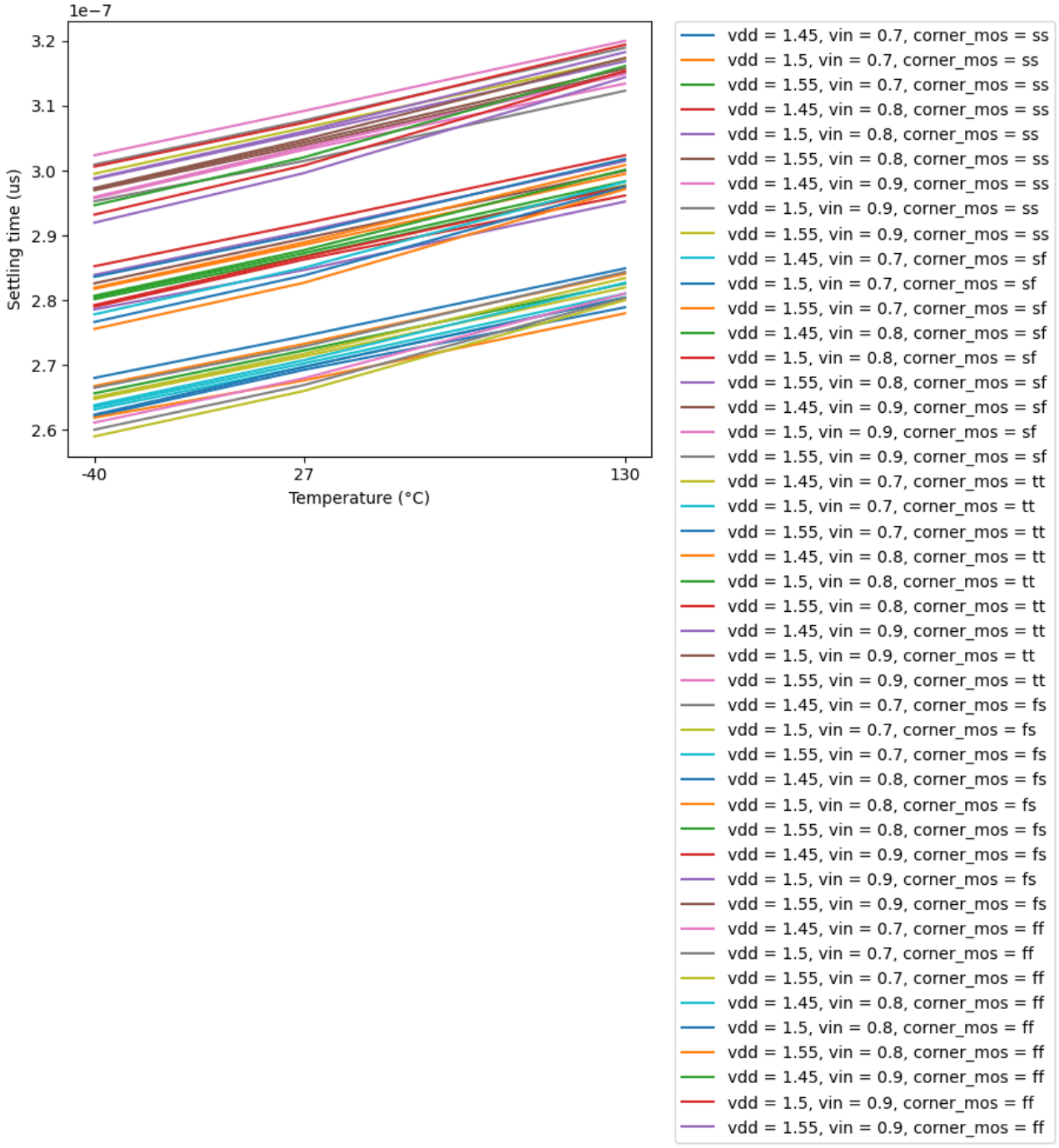


Figure A.15.: settling_vs_temp

A.17. settling_vs_vin

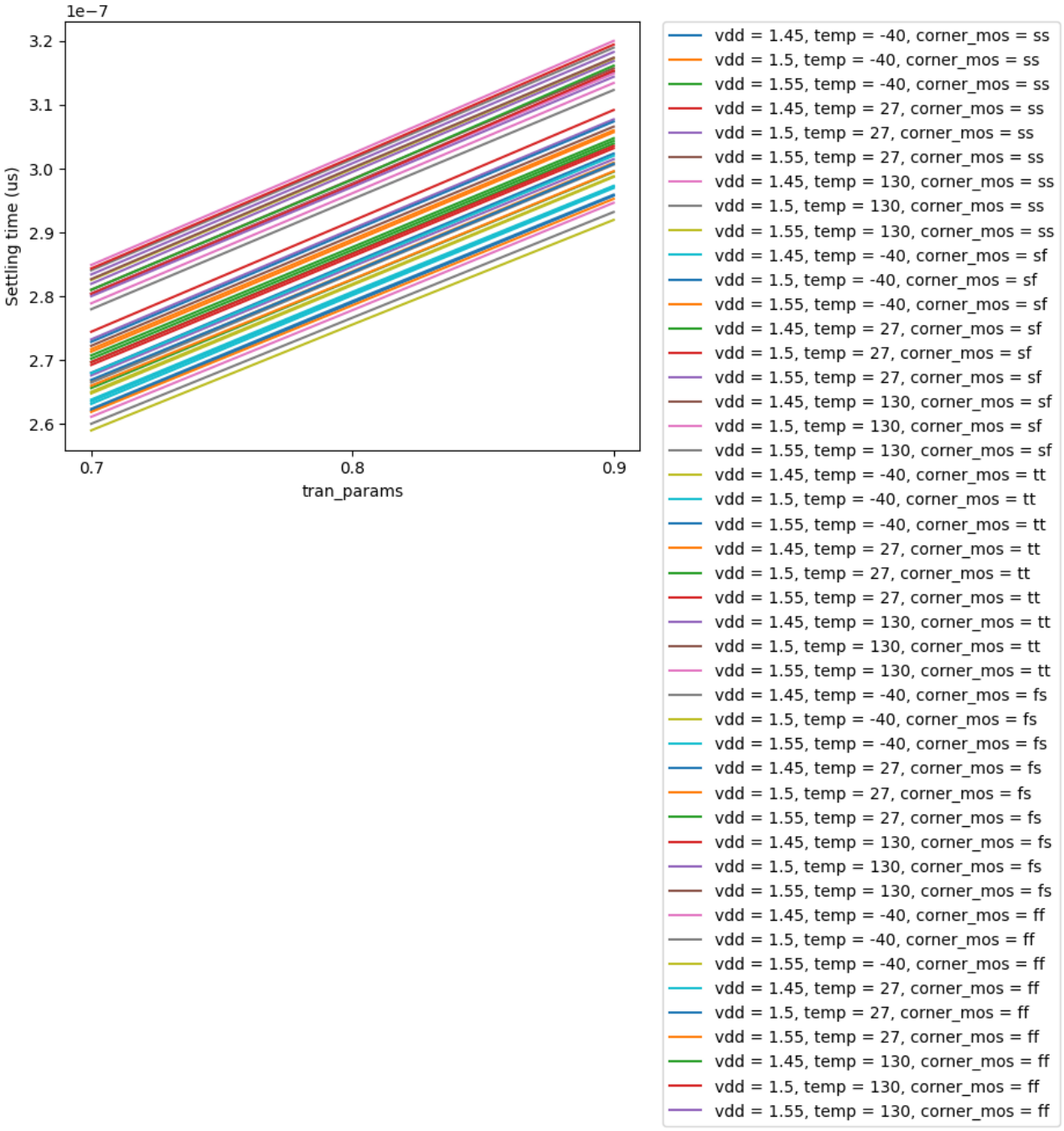


Figure A.16.: settling_vs_vin

A.18. settling_vs_vdd

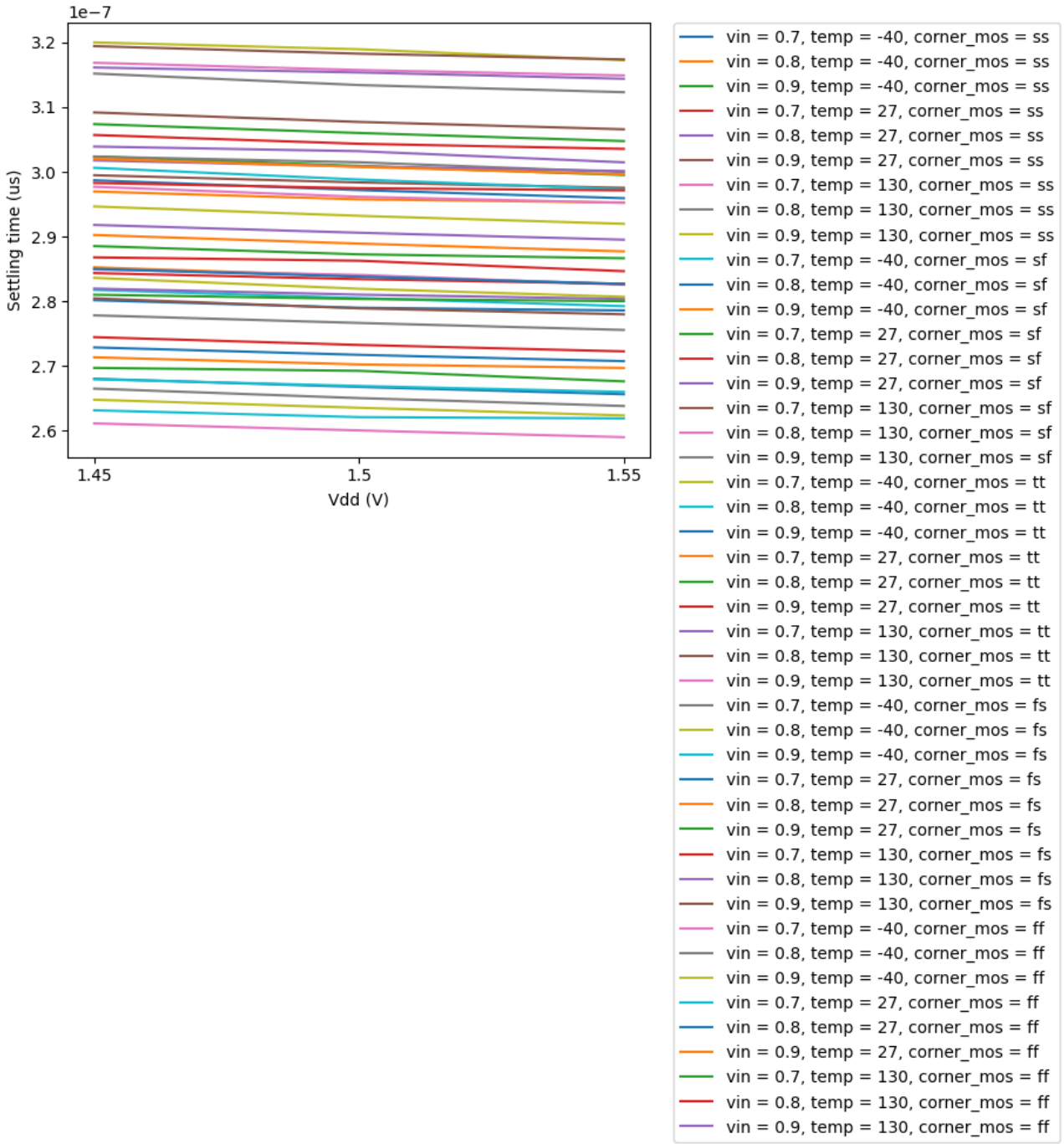


Figure A.17.: settling_vs_vdd

A.19. settling_vs_corner

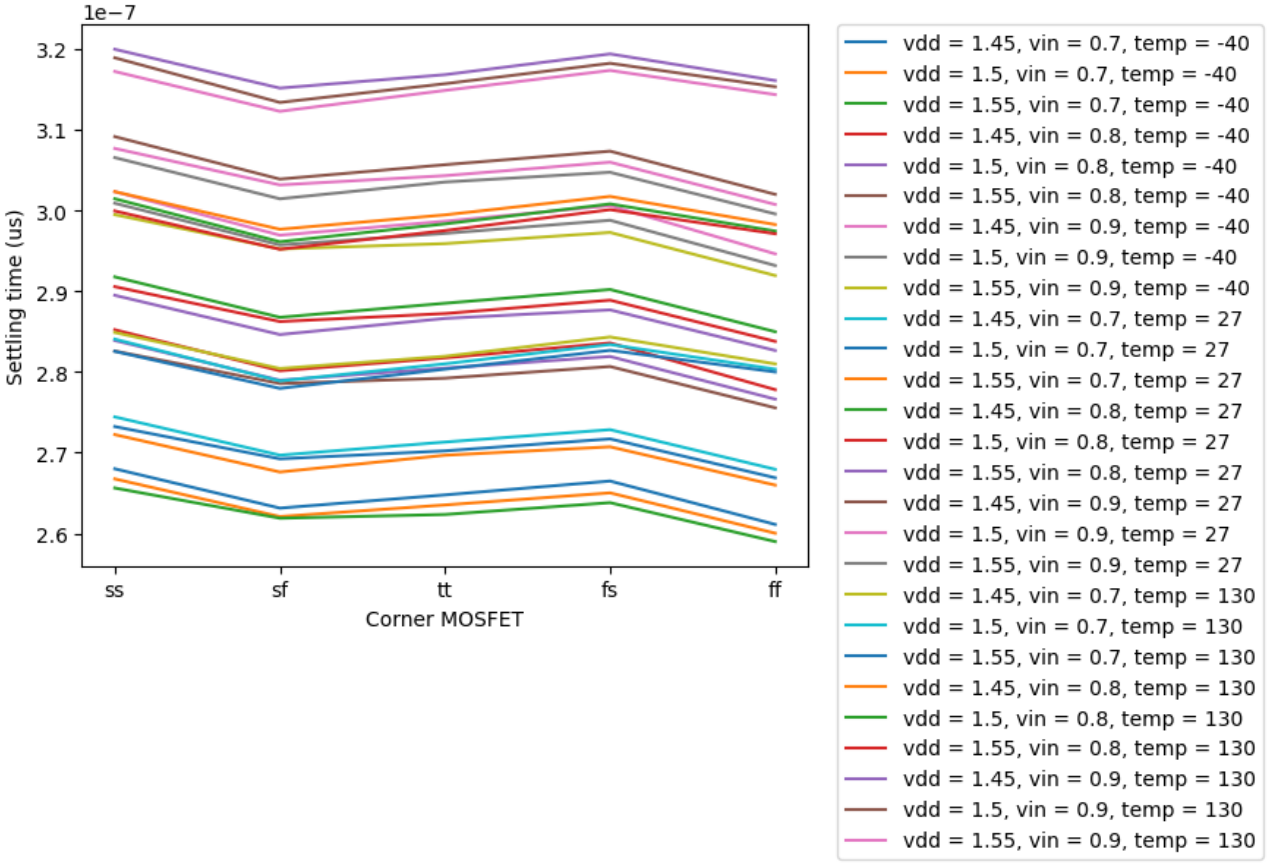


Figure A.18.: settling_vs_corner

B. DRC Output

C. USB-Stick

```
thesis_hp/
   .gitattributes
   .gitignore
   _book/
   _quarto/
   _quarto.yml
   _bibliography
   _manuscript
   appendix.qmd          # The file where this structure will be placed
   index.qmd             # Main project page / Introduction
   README.md
   references.qmd
   KNOWN_ISSUES.md
   cace/                 # (Folder likely related to CACE/CAD tools)
   content/
   Designs/              # Top-level Design Folder
      foldedcascode_nmos_withdummies.gds   # (GDS file for layout fabrication)
      foldedcascode_nmos_withdummies.spice # (SPICE netlist for simulation)
      1_schematics/     # (Design subfolder: Circuit schematics)
      2_layout/         # (Design subfolder: Physical layout files)
      3_kpex/           # (Design subfolder: Extracted parasitic netlists)
   Expose_HP/            # (Likely High-Power related files/code)
   figures/              # Location for generated images and plots
   layout_scripts/       # Scripts specific to physical layout generation/checking
   Literature/           # References or literature review material
   python_codes/         # General scripts for data processing or analysis
   sizing/               # Files related to component sizing/calculations
   simulations/          # Folder for simulation setup and raw outputs
   postlayout_simulations/   # Post-Layout Simulation results/setup
```