Setting up the open-source tools ecosystem

- Install WSL2
  https://learn.microsoft.com/en-us/windows/wsl/
- Install and start MobaXterm Home Edition
  https://mobaxterm.mobatek.net/download.html
- Install and configure docker
  https://docs.docker.com/desktop/setup/install/windows-install/
- Download IIC-OSIC-TOOLS
  https://github.com/iic-jku/IIC-OSIC-TOOLS

```
cd ~
git clone --depth=1 https://github.com/iic-jku/iic-osic-tools.git
```

- Install VS code and the WSL extension
  https://code.visualstudio.com/download

Setting up the open-source tools ecosystem

- Start Docker Desktop 

- Customize the **DESIGNS** environment variable and start the container using a local X-Server.
  All user data is persistently placed in the directory pointed to by the environment variable **DESIGNS** (the default is **$HOME/eda/designs** for Linux/macOS and **%USERPROFILE%\eda\designs** for Windows).

  Browse to the **iic_osic_tools** directory
  cd ~/iic_osic_tools
  and start the container using the following CLI commands:

```
export DESIGNS="$HOME/ghome/eda/designs"
export DOCKER_TAG="latest"
./start_x.sh
```

Setting up the open-source tools ecosystem

- Once the container's terminal pops-up, setup the desired PDK as follows:
  `sak-pdk`
  `sak-pdk gf180mcuD`

- When starting the docker container, if a file **`.designinit`** is put in the **`$DESIGNS`** directory, the file is sourced last. In this way, users can adapt settings to their needs.

.designinit

```
echo iic version = $IIC_OSIC_TOOLS_VERSION
export PDK=gf180mcuD
export PDKPATH=$PDK_ROOT/$PDK
export STD_CELL_LIBRARY=gf180mcu_fd_sc_mcu7t5v0
export KLAYOUT_PATH=$PDKPATH/libs.tech/klayout:$PDKPATH/libs.tech/klayout/tech
echo PDK = $PDK
echo PDKPATH = $PDKPATH
echo STD_CELL_LIBRARY = $STD_CELL_LIBRARY
echo KLAYOUT_PATH = $KLAYOUT_PATH
export SPICE_USERINIT_DIR=.
echo SPICE_USERINIT_DIR = $SPICE_USERINIT_DIR
```

NOTE: any error in **`.designinit`** will inhibit the terminal to pop up! I wasted a day on it!

Setting up the open-source tools ecosystem: step by step tutorial

- https://github.com/claudiotalarico/vlsi-2025/tree/main

- https://kwantaekim.github.io/2024/05/25/OSE-Docker/

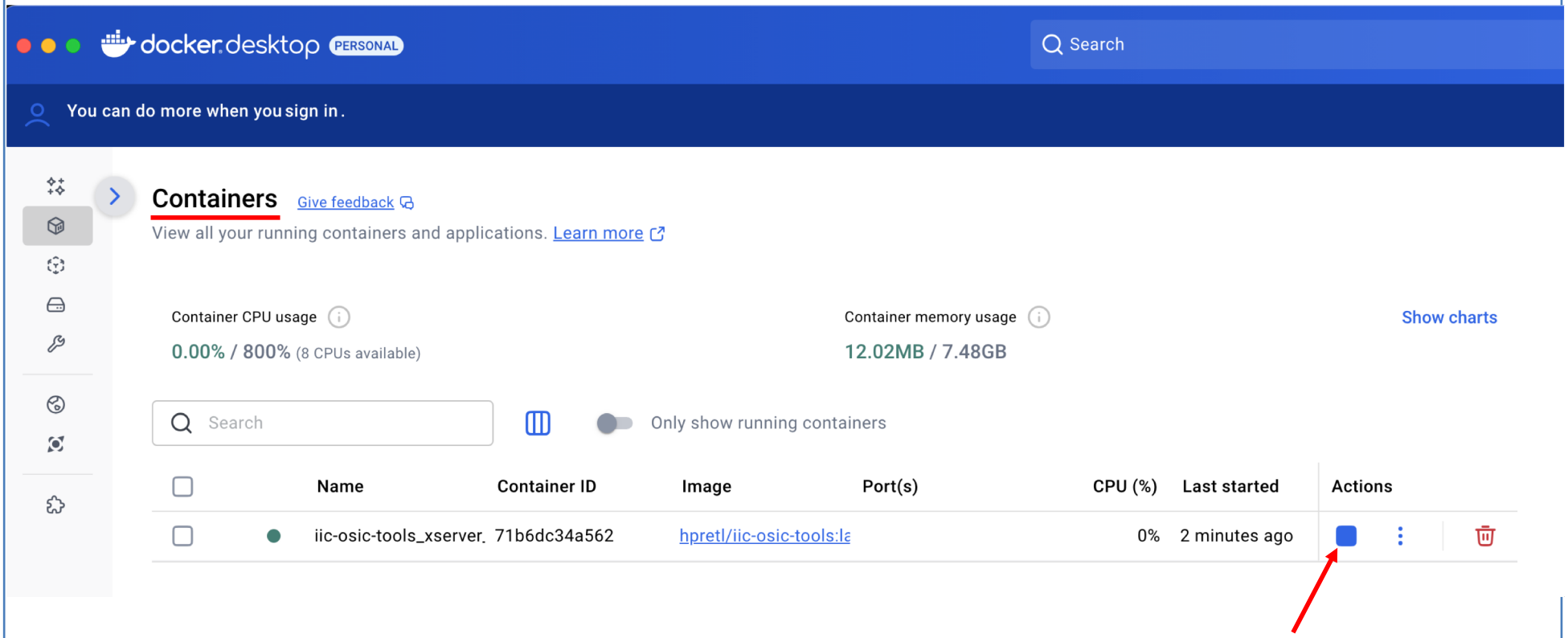
The Terminal

Terminal -

File Edit View Terminal Tabs Help

```
iic version = 2025.11
PDK = gf180mcuD
PDKPATH = /foss/pdks/gf180mcuD
STD_CELL_LIBRARY = gf180mcu_fd_sc_mcu7t5v0
KLAYOUT_PATH = /foss/pdks/gf180mcuD/libs.tech/klayout:/foss/pdks/gf180mcuD/libs.tech/klayout/tech
SPICE_USERINIT_DIR = .
/foss/designs >
```

Starting and Stopping the Container



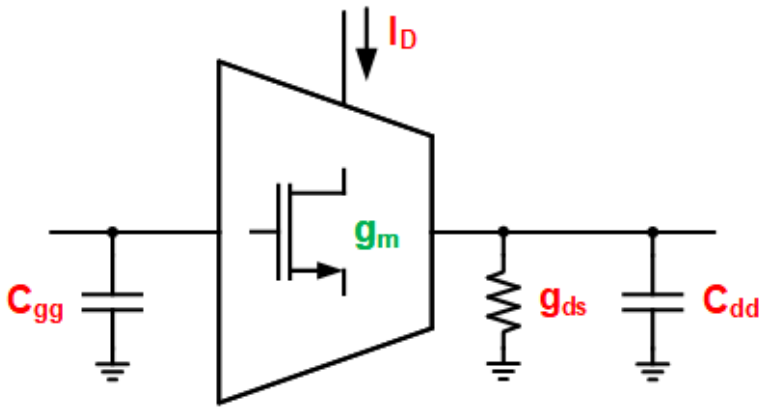
Finally, back to question of design …

- How exactly should we use the LUTs for systematic design ?
- Many options exists (you can invent your own)
- Suggested approach
  - Think of a transistor in terms of (approximately) width-independent figures of merit that are linked to design specification (rather than some physical parameter that does not directly relate to the circuit specs.)
    - Because we have no idea what the width is when we start to design ("normalized design")
  - Think about the design tradeoffs in terms of the MOSFET's inversion level
    - Using the transconductance efficiency (g_m/I_D) as a proxy

The cost of transconductance

- An ideal transistor minimizes the undesired quantities (red), relative to the desired quantity (green)

$$C_{gg} = C_{gs} + C_{gd} + C_{gb}$$



$$C_{dd} = C_{db} + C_{gd}$$

Figures of Merit for Design

desired quantity over undesired quantities

Square-law:

$$V_{OV} = V_{GS} - V_t$$

- Transconductance efficiency
  - Want large g_m, for as little current as possible

$$g_m = \mu C_{ox} \frac{W}{L} V_{OV} = \frac{2I_D}{V_{OV}}$$

$$\boxed{\frac{g_m}{I_D}} = \frac{2}{V_{OV}}$$ Want small V_{OV}

- Angular transit frequency
  - Want large g_m, small C_{gg}

$$C_{gs} = \frac{2}{3} C_{ox} WL$$

$$\boxed{\frac{g_m}{C_{gg}}} \approx \frac{3}{2} \frac{\mu V_{OV}}{L^2}$$ Want large V_{OV}

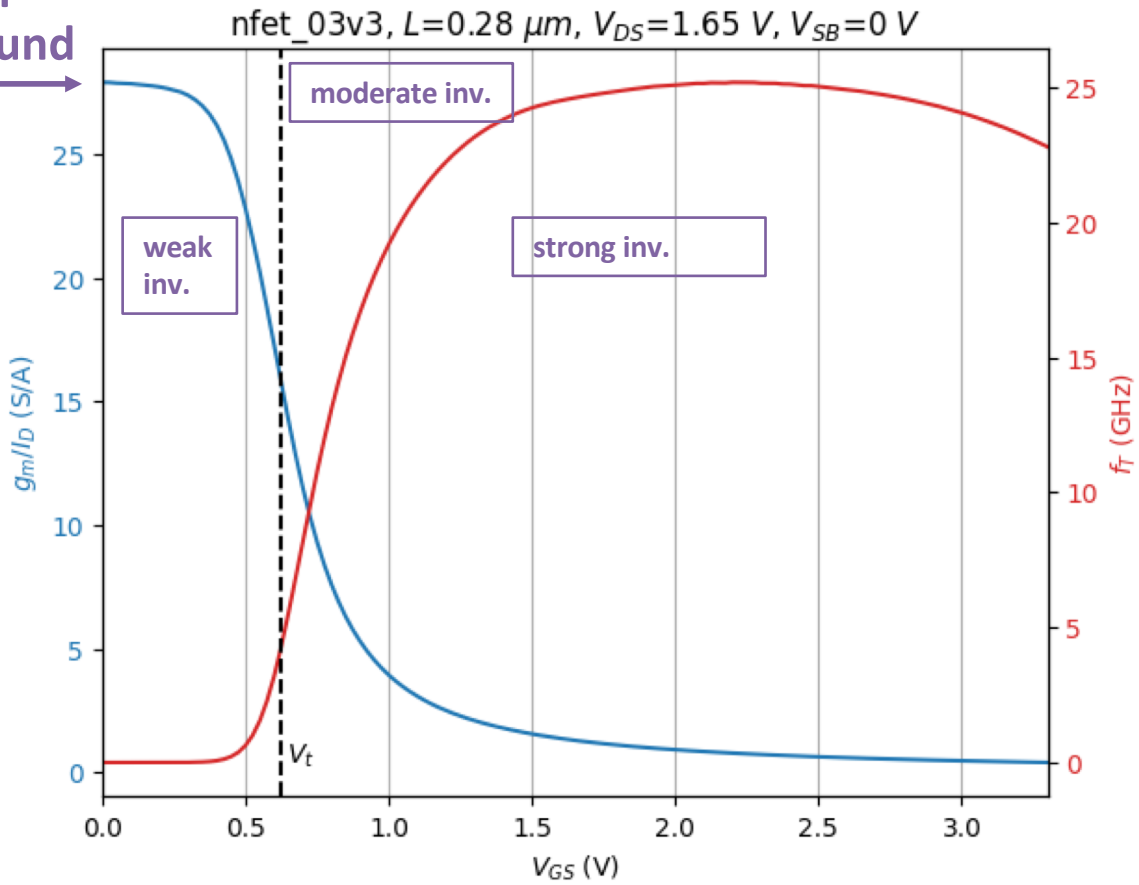
- Intrinsic gain
  - Want large g_m, small g_{ds}

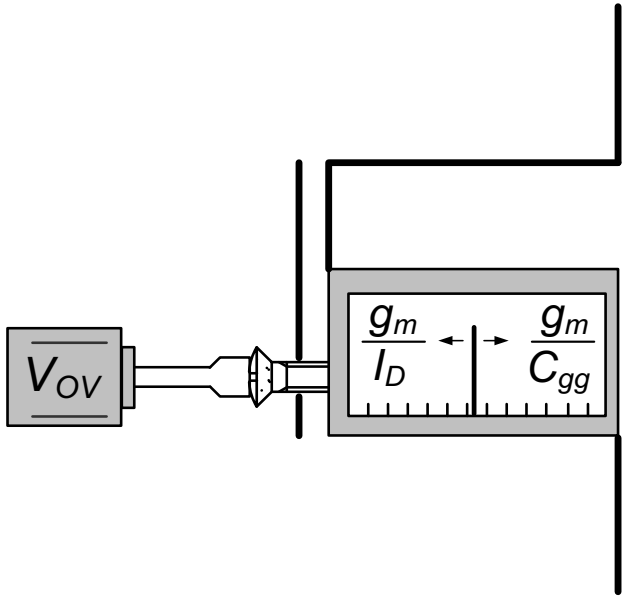
$$g_{ds} \approx \lambda I_D$$

$$\boxed{\frac{g_m}{g_{ds}}} \approx \frac{2}{\lambda V_{OV}}$$ Want small V_{OV}

Fundamental Design Tradeoff: g_m/I_D vs. f_T (efficiency vs. speed)

Upper Bound

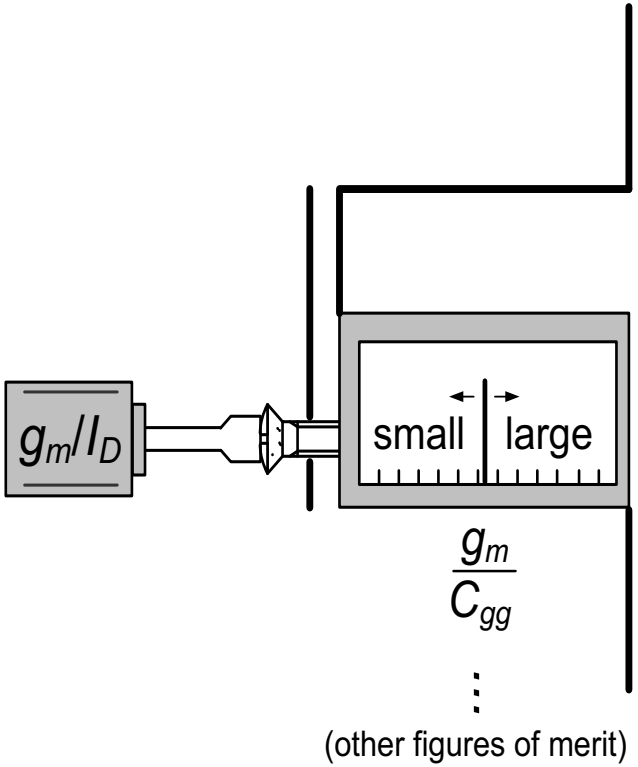


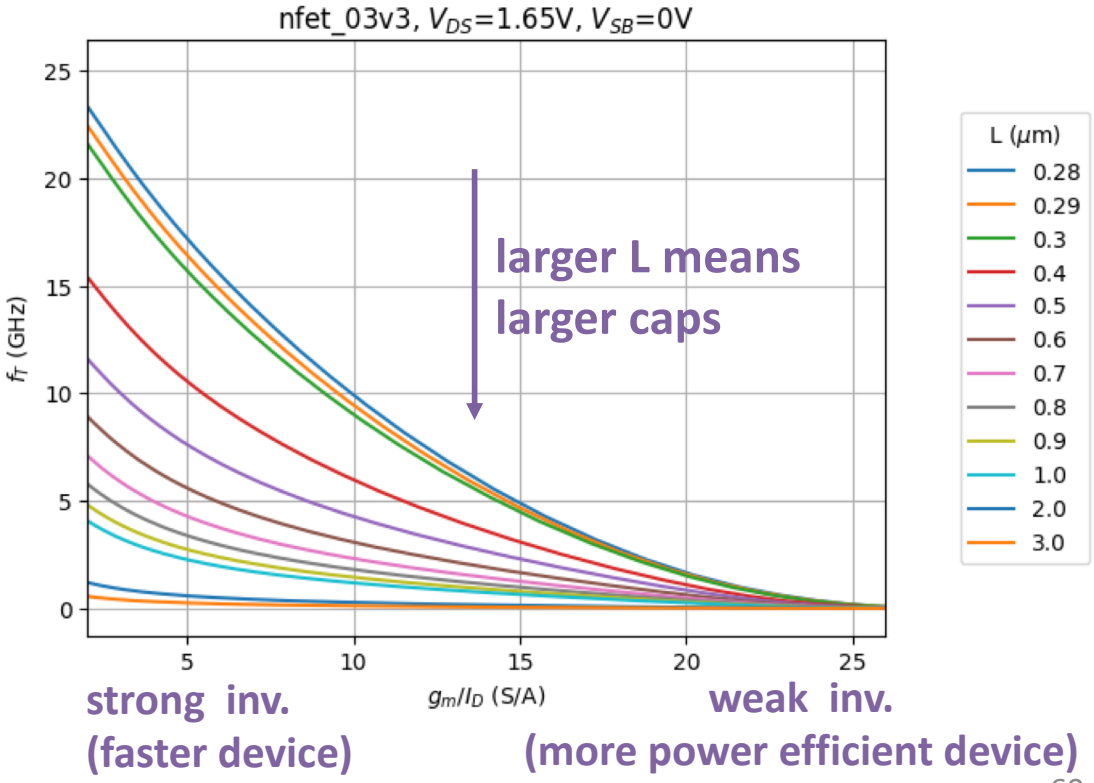


The g_m/I_D curves are similar for all techs, because the "asymptote" is related to $\frac{q}{KT} \cdot \frac{1}{n}$

Eliminating V_{OV}

- The inversion level is also fully defined by picking g_m/I_D
- There is no need to know $V_{OV} = V_{GS} - V_t$
  - Outside the square-law framework, Vov is relatively useless: the culprit is V_t

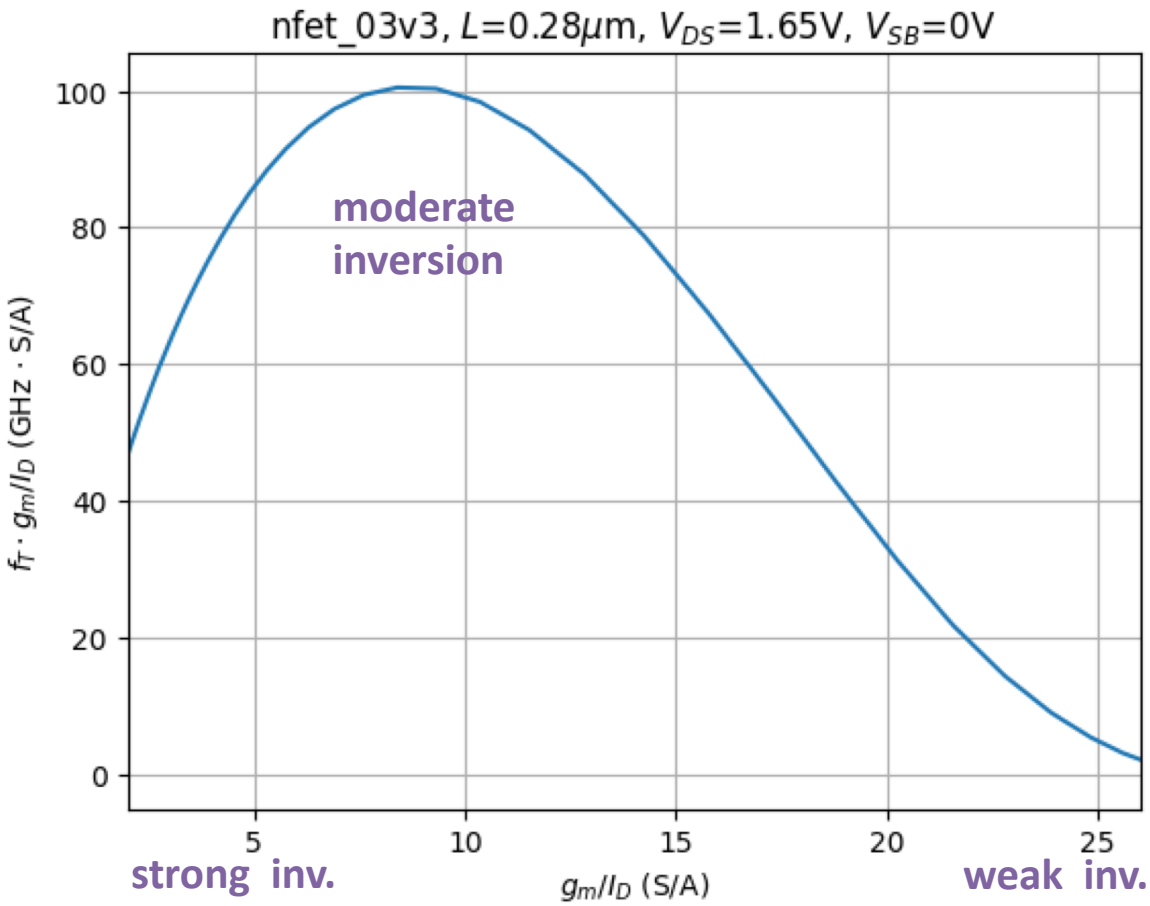




strong inv. (faster device)

weak inv. (more power efficient device)

Product of g_m/I_D and f_T



- Interestingly, the product of g_m/I_D and f_T peaks in moderate inversion
- Operating the transistor in moderate inversion is optimal when we value speed and power efficiency equally (not always the case)

What about V_{Dsat} ?

- V_{Dsat} can be estimated using g_m/I_D (see 2.3.2 of Jesper and Murman)
- $V^* = 2/(g_m/I_D)$ is a good estimate for V_{Dsat}

Square Law

$$I_D = \frac{1}{2}\mu C_{ox}\frac{W}{L}(V_{GS}-V_t)^2$$

$$g_m = \mu C_{ox}\frac{W}{L}(V_{GS}-V_t)$$

$$\frac{2}{g_m/I_D} = V_{GS}-V_t = V_{Dsat}$$

∴ Consistent with the classical square-law relationship

Weak Inversion

Need ~$3U_T$ for saturation (I_D "independent" of V_{DS})

$$I_D = I_{D0}e^{\frac{V_{GS}-V_t}{nU_T}}\left(1-e^{-\frac{V_{DS}}{U_T}}\right)$$

$e^{-3} \approx 0.05 \ll 1$

$$g_m = \frac{I_{D0}}{nU_T}e^{\frac{V_{GS}-V_t}{nU_T}}\left(1-e^{-\frac{V_{DS}}{U_T}}\right)$$

$$\frac{2}{g_m/I_D} = 2nU_T \approx 3U_T \qquad U_T = \frac{kT}{q}$$

∴ Matches well with the required minimum V_{DS}

Estimation of V_{Dsat}

Source: Jespers and Murmann, p. 46

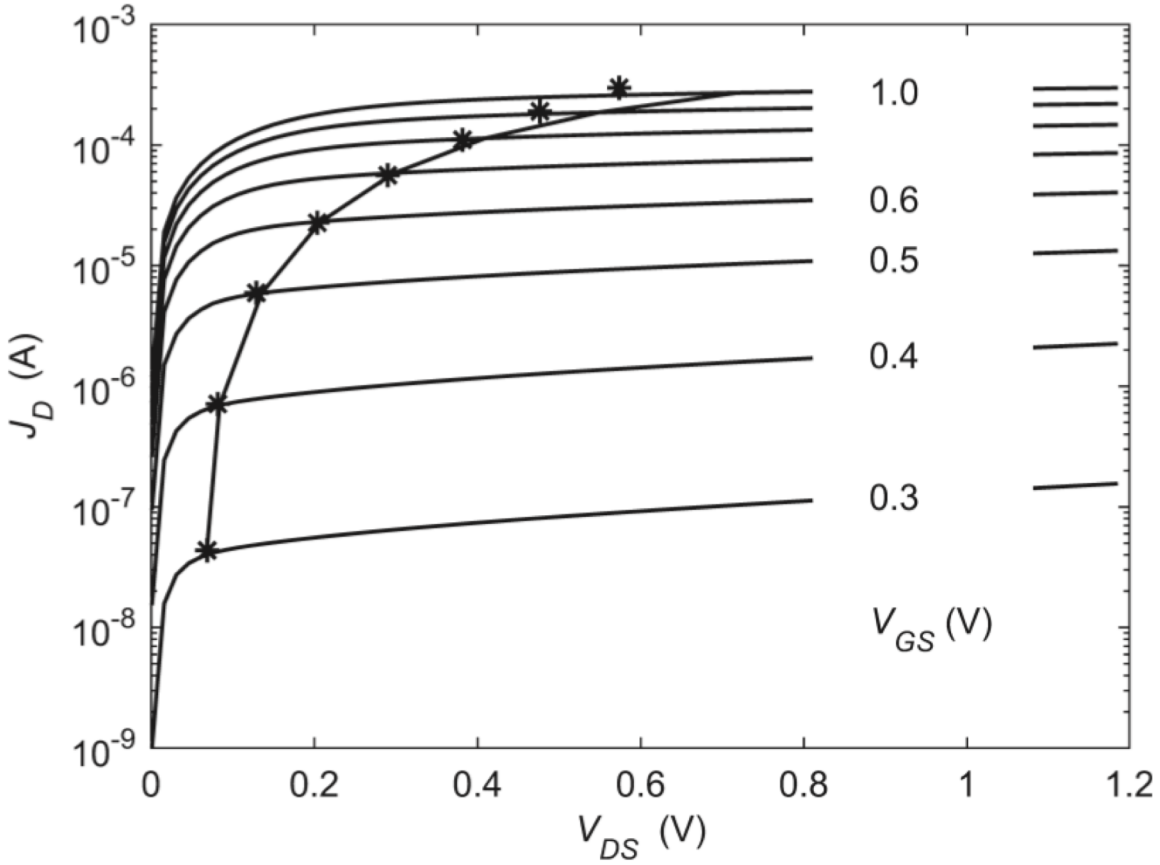


Figure 2.16 Evolution of V_{Dsat} from weak to strong inversion (L = 100 nm).

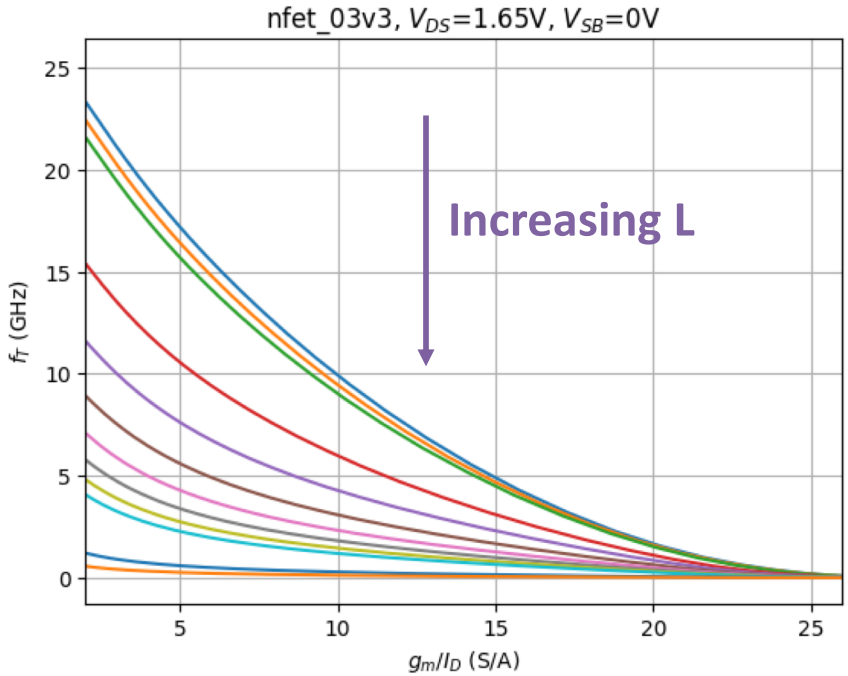
The asterisks are the estimation ($V^* = 2/(g_m/I_D)$), whereas the solid line comes from the EKV model

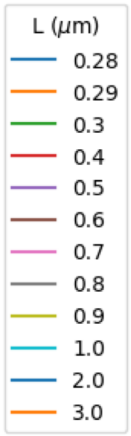
g_m/I_D-Centric Design

- Tabulate/Plot the following parameters over a reasonable range of g_m/I_D and channel lengths
  - Transit frequency (f_T)
  - Intrinsic gain (g_m/g_{ds})
- Tabulate/Plot relative estimates of extrinsic capacitances
  - C_{gd}/C_{gg} and C_{dd}/C_{gg}
- Note that all of these parameters (to first order) are independent of device width

- To compute the device widths, we need one more table/plot that links g_m/I_D and current density $J_D = I_D/W$

Basic Design FOMs in a Nutshell

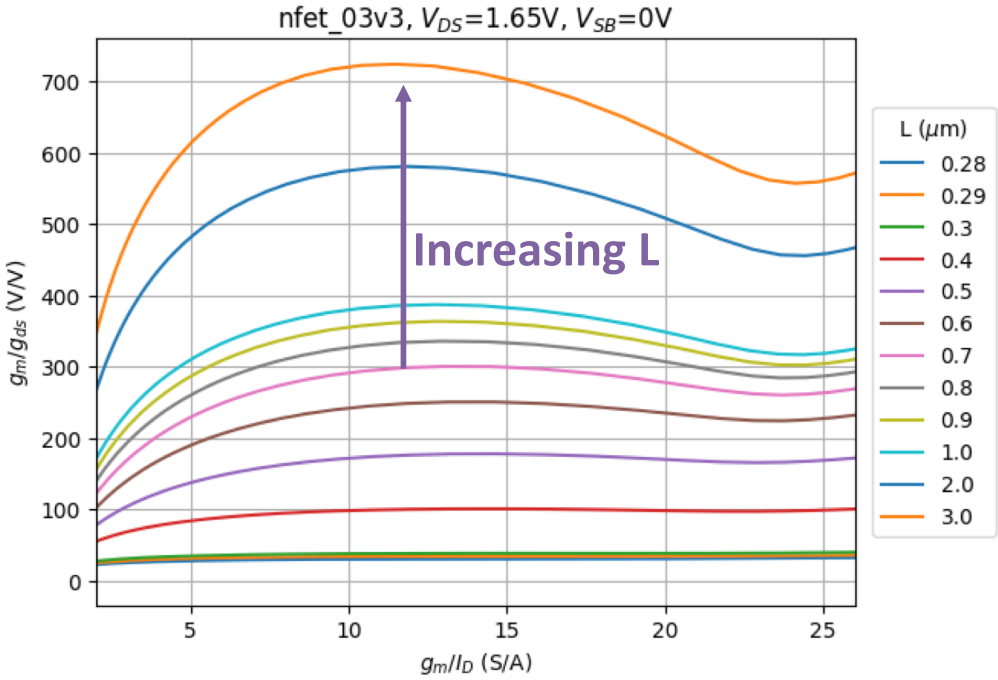




Transit frequency

$$\left(\frac{1}{2\pi}\cdot\frac{g_m}{C_{gg}}\right)$$

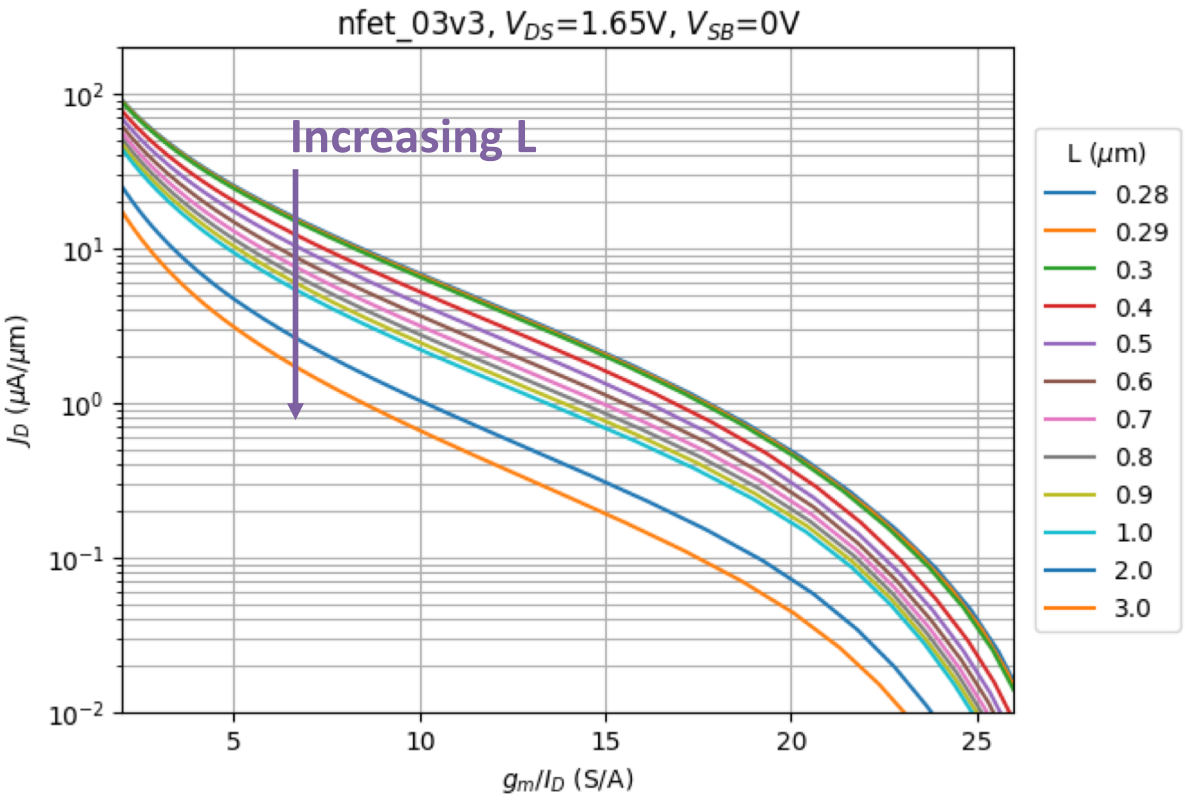
Intrinsic Gain $\left(\frac{g_m}{g_{ds}}\right)$



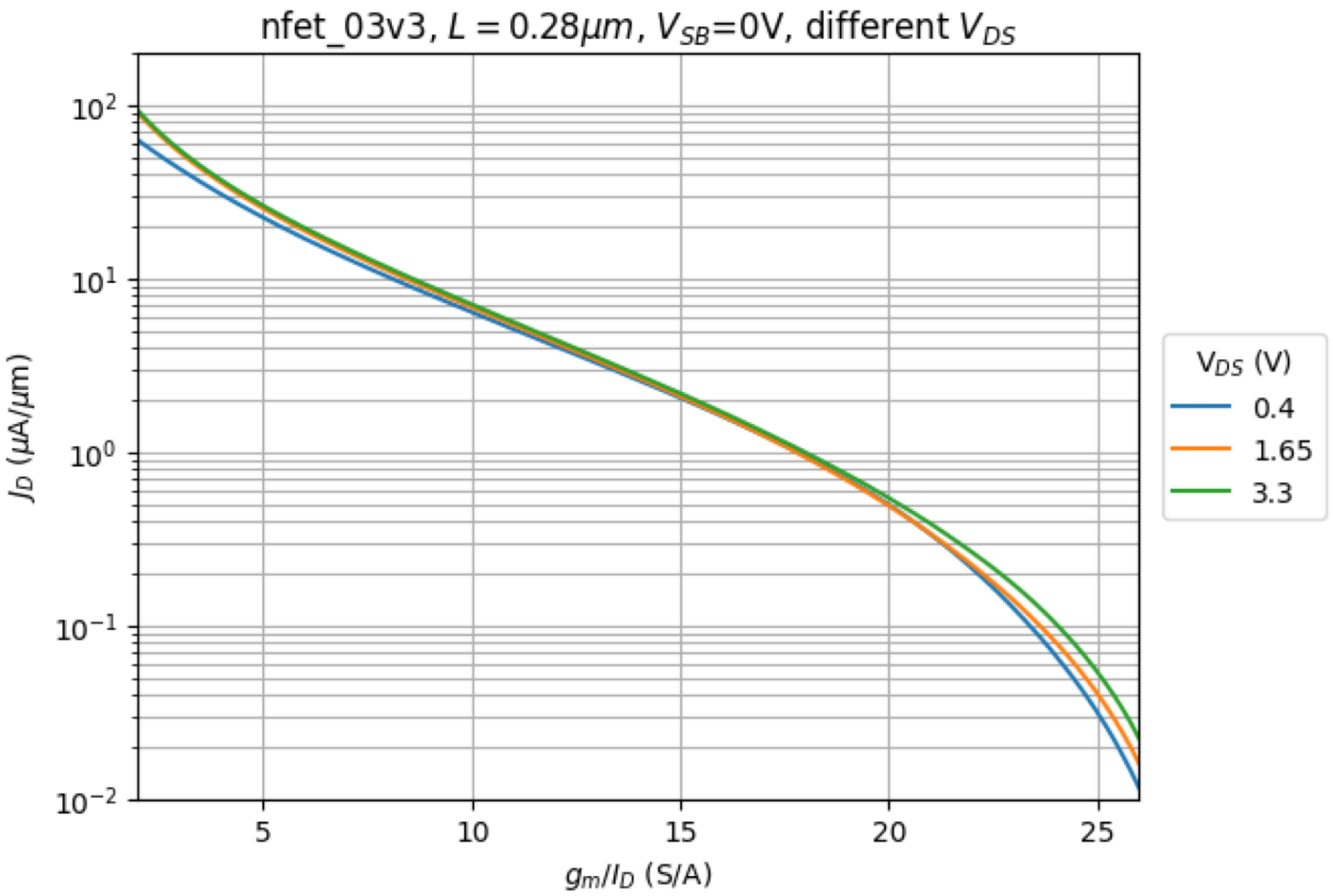
One benefit of making the LUTs is also to spot issues in the modeling of the technology

Basic Design FOMs in a Nutshell

Current Density Chart: $J_D = \left(\frac{I_D}{W}\right)$

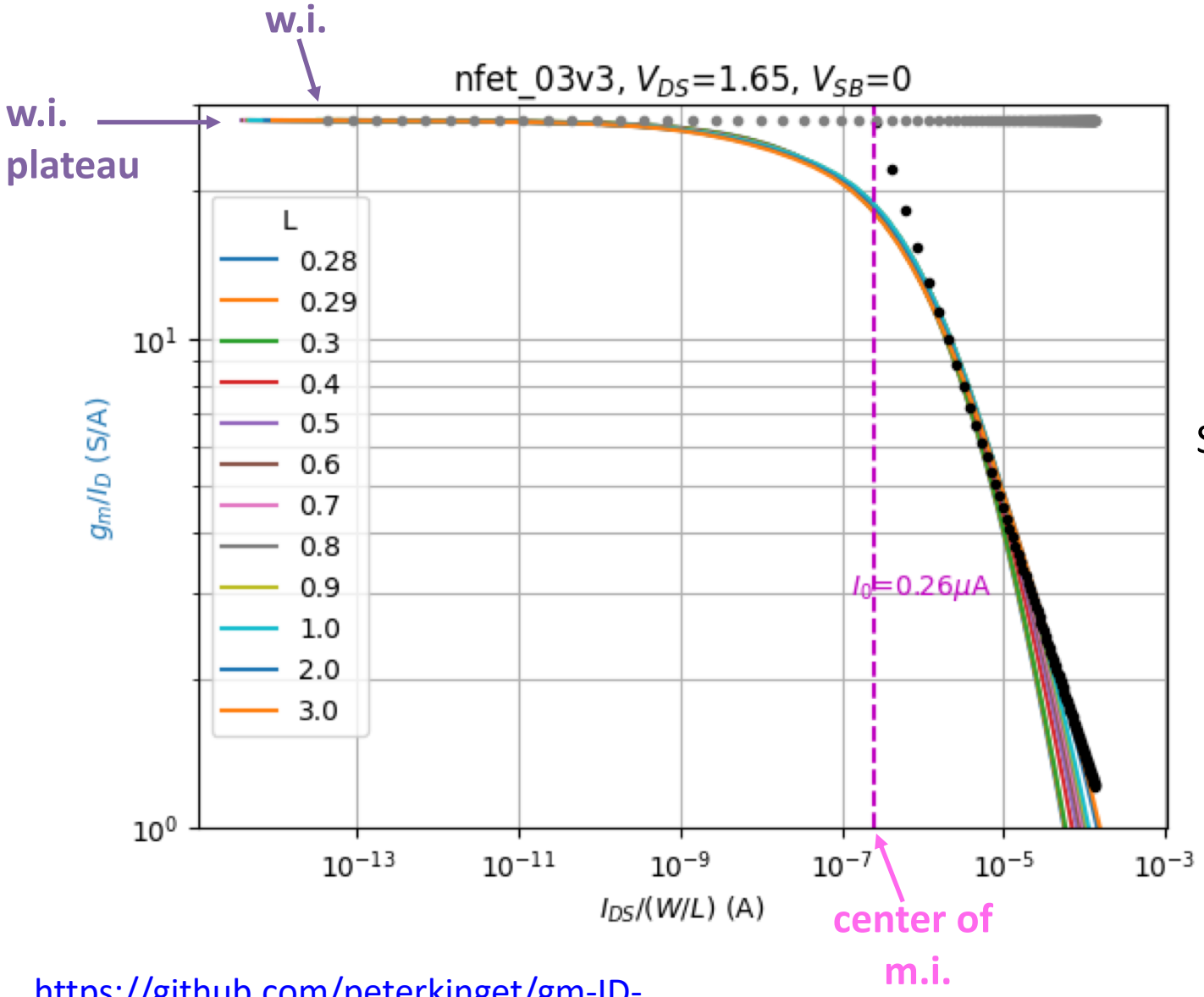


V_{DS} dependence of current density



- V_{DS} dependence is relatively weak
- Typically OK to work with data generated for $V_{DD}/2$

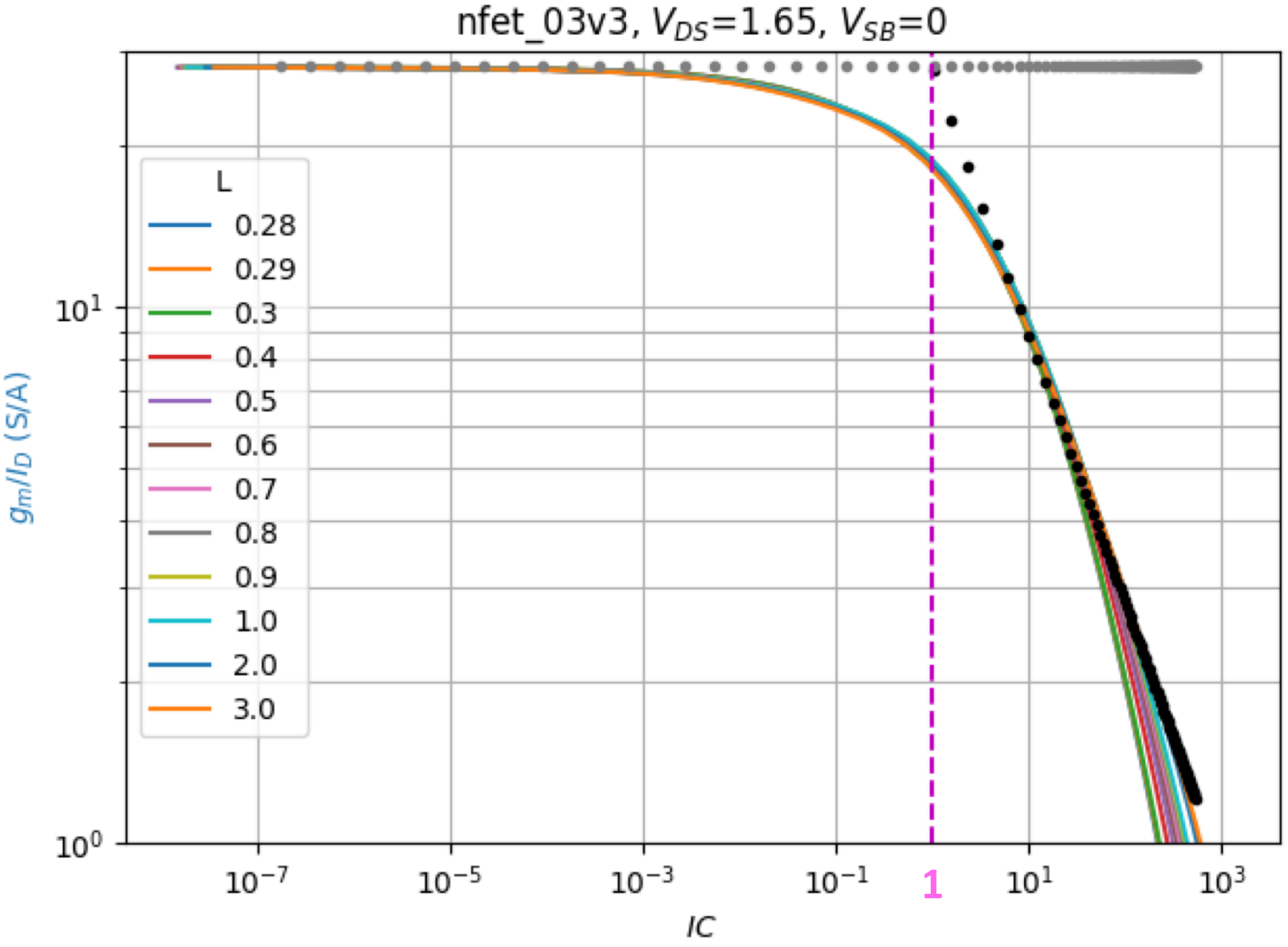
Another way to explore the design tradeoffs



See Peter Kinget's Repo

https://github.com/peterkinget/gm-ID-gf180mcuD/tree/main/starter_files_open_source_tools/gf180mcuD

Another way to explore the design tradeoffs



W.I.: IC ≤ 0.1
M.I.: 0.1 < IC ≤ 10
S.I.: IC > 10

PK_techsweep_plots_from_mat_nfet.ipynb

PK_techsweep_plots_from_mat_pfet.ipynb

EKV equations and the inversion coefficient

- Kevin Fronczak, Inversion Coefficient Based Circuit Design https://kevinfronczak.com/blog/inversion-coefficient-based-circuit-design
- C. Enz, F. Chicco and A. Pezzotta, "Nanoscale MOSFET Modeling: Part 1: The Simplified EKV Model for the Design of Low-Power Analog Circuits," in IEEE Solid-State Circuits Magazine, Fall 2017. https://ieeexplore.ieee.org/document/8016485
- C. Enz, F. Chicco and A. Pezzotta, "Nanoscale MOSFET Modeling: Part 2: Using the Inversion Coefficient as the Primary Design Parameter," in IEEE Solid-State Circuits Magazine, Fall 2017. https://ieeexplore.ieee.org/document/8110872
- C. Enz, M. -A. Chalkiadaki and A. Mangla, "Low-power analog/RF circuit design based on the inversion coefficient," in *IEEE ESSCIRC Conference 2015. https://ieeexplore.ieee.org/document/7313863*
- W. Sansen, "Analog design procedures for channel lengths down to 20 nm," in *2013 IEEE ICECS Conference*, 2013. https://ieeexplore.ieee.org/document/6815423

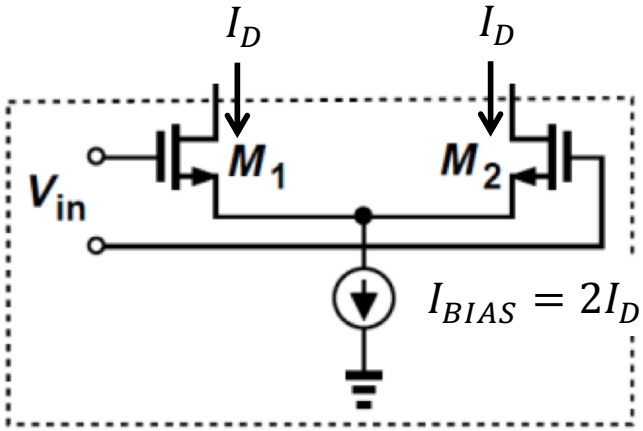
Generic Design Flow

1) Determine g_m (from design objectives: e.g. gain)
2) Pick L
   - Short channel → high f_T (high speed)
   - Long channel → high intrinsic gain (lower noise, better matching)
3) Pick g_m/I_D (or f_T)
   - Large g_m/I_D → low current, large signal swing (low $V_{DS,sat}$)
   - Small g_m/I_D → high f_T (high speed)
4) Compute I_D (from g_m and g_m/I_D)
5) Compute W (from $J_D = I_D/W$)

Many other possibilities exist (depending on *circuit specifics, design constrains, and objectives*)

Basic Sizing Example

- Size the differential pair to achieve gm = 10 mS
- Use minimal channel length (L=0.28 μm)
- Consider g_m/I_D = [5, 10, 20] S/A
- Compute I_D and W
- Estimate C_{gg} and f_T



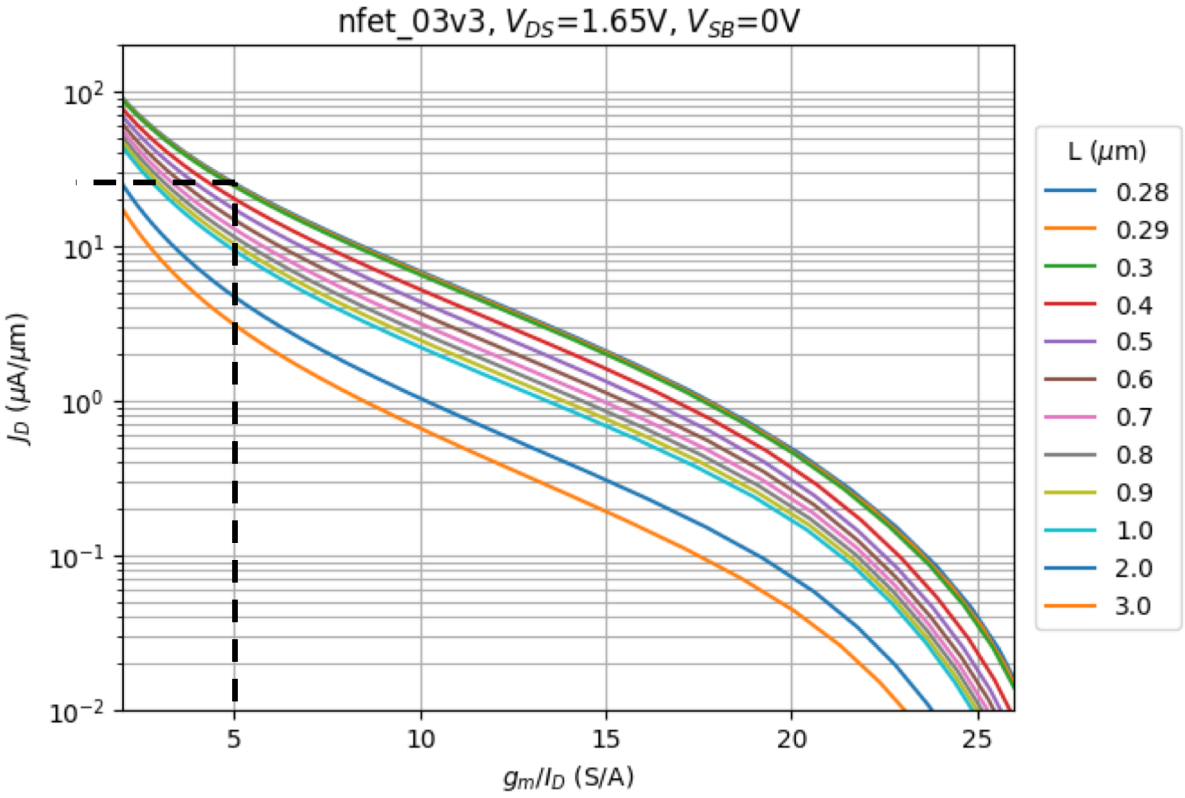
Assumption: we have a clear idea of the value of g_m we need (not always the case)

g_m/I_D = 5 S/A → strong inversion (fast design)
g_m/I_D = 10 S/A → moderate inversion
g_m/I_D = 20 S/A → weak inversion (low current)

Solution for g_m/I_D = 5 S/A: Graphical illustration

$$I_D = \frac{g_m}{\frac{g_m}{I_D}} = \frac{10mS}{5\frac{S}{A}} = 2\ mA$$

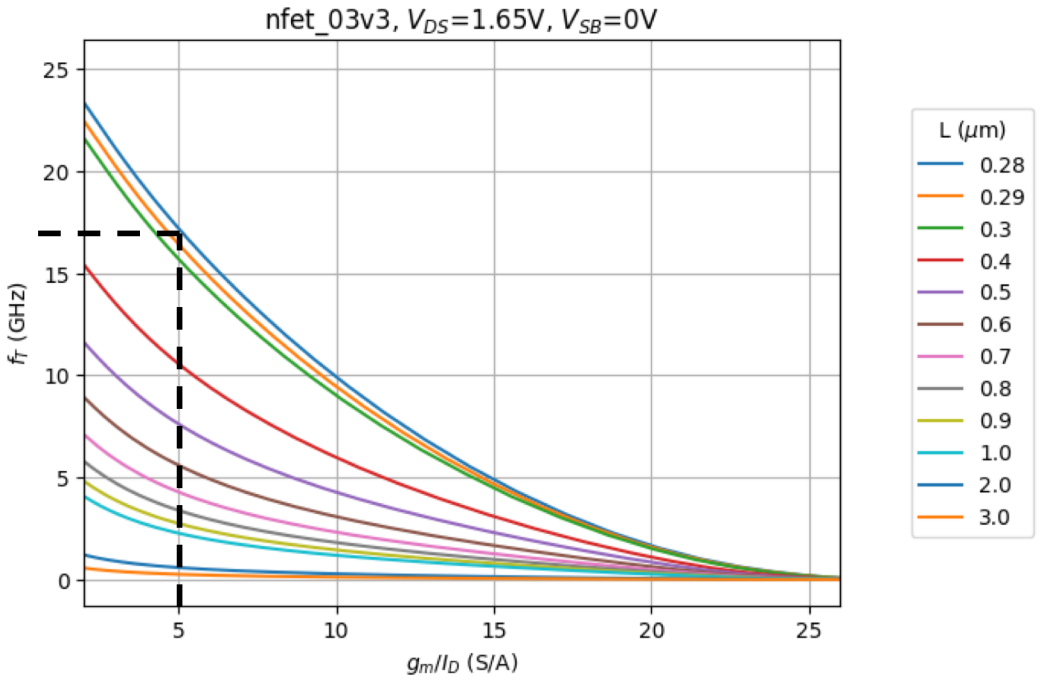
$$W = \frac{I_D}{\frac{I_D}{W}} = \frac{2\ mA}{25.4\frac{\mu A}{\mu m}} = 78.8\ \mu m$$



Solution for g_m/I_D = 5 S/A: Graphical illustration

$$f_T = \frac{1}{2\pi} \cdot \frac{g_m}{C_{gg}} \cong 17.2\ GHz$$

$$C_{gg} = \frac{g_m}{2\pi f_T} = \frac{10mS}{1.08 \cdot 10^{11}\ rad/s} \cong 92.6\ fF$$



A more practical solution: use a script

```
/foss/designs/gf180-2025/basic > jupyter notebook basic.ipynb
```

Design Choices

```
L1 = 0.28 # um
gm_id = np.array([5, 10, 20])
```

[4] ✓ 0.0s

← minimum length for the process

Sizing and benchmarking

```
id = gm/gm_id
jd = n.lookup('ID_W', GM_ID=gm_id, L=L1)
w = id/jd
cgg_w = n.lookup('CGG_W', GM_ID=gm_id, L=L1)
cgg = cgg_w*w
ft = gm/cgg/2/np.pi
```

[5] ✓ 0.0s

```
df = pd.DataFrame([gm_id, id, jd, w, cgg, ft],['gm_id','id','jd','w','cgg','ft'],columns=['option1','option2','option3'])
```

[6] ✓ 0.0s

```
df
```

[7] ✓ 0.0s

…

	option1	option2	option3
gm_id	5.000000e+00	1.000000e+01	2.000000e+01
id	2.000000e-03	1.000000e-03	5.000000e-04
jd	2.539270e-05	6.847242e-06	4.903808e-07
w	7.876281e+01	1.460442e+02	1.019616e+03
cgg	9.257212e-14	1.603948e-13	9.581458e-13
ft	1.719253e+10	9.922699e+09	1.661072e+09

option 1 (strong inv): large current, small device, high f_T
option 2 (moderate inv): often a reasonable compromise!
option 3 (weak inv.): small current, huge device, low f_T

← the units of W are um

SPICE Validation

Spice Validation (option 2)

```
%%writefile ./.spiceinit
set ngbehavior=hsa
set ng_nomodcheck
set color0=white
set color1=black
set xbrushwidth=2
set altshow
```

[8] ✓ 0.0s

```
%%writefile ./sizing_diffpair.ngspice
** differential pair sizing example

.lib /foss/pdks/gf180mcuD/libs.tech/ngspice/sm141064.ngspice typical
.inc /foss/pdks/gf180mcuD/libs.tech/ngspice/design.ngspice
.param lx=0.28e-6 wx=146e-6 nfx=30 idx=1e-3

XM1 d g s 0 nfet_03v3 L={lx} W={wx} nf={nfx}
+ ad='int((nf+2)/2) * W/nf * 0.18u' as='int((nf+2)/2) * W/nf * 0.18u'
+ pd='2*int((nf+2)/2) * (W/nf + 0.18u)' ps='2*int((nf+2)/2) * W/nf * 0.18u'
+ nrd='0.18u / W' nrs='0.18u / W' sa=0 sb=0 sd=0 m=1
XM2 d g s 0 nfet_03v3 L={lx} W={wx} nf={nfx}
+ ad='int((nf+2)/2) * W/nf * 0.18u' as='int((nf+2)/2) * W/nf * 0.18u'
+ pd='2*int((nf+2)/2) * (W/nf + 0.18u)' ps='2*int((nf+2)/2) * W/nf * 0.18u'
+ nrd='0.18u / W' nrs='0.18u / W' sa=0 sb=0 sd=0 m=1

vg g 0 1
vd d 0 2
ibias s 0 {2*idx}

.control
op
*show
let gm = @m.xm1.m0[gm]
let id = @m.xm1.m0[id]
let gm_id = @m.xm1.m0[gm]/@m.xm1.m0[id]
let cgg = @m.xm1.m0[cgg] + @m.xm1.m0[cgso] + @m.xm1.m0[cgdo]
print gm
print id
print cgg
.endc
```

← Note fingered transistor Implementation (nf=30)

[9] ✓ 0.0s

```
!/foss/tools/bin/ngspice -b ./sizing_diffpair.ngspice
[10]  ✓  0.2s
```

```
gm = 9.983304e-03
id = 1.000001e-03
cgg = 1.596767e-13
```

close to design target of 10 mS

close to estimate of 160 fF

- Error sources
  - Width of device fingers is not equal to width used for lookup table
  - Sizing did not consider exact values of V_{DS}, V_{SB} (can include)
  - Lookup interpolation error

Typically add up to <10% error

To get more accurate values, you can iterate. But in general you should not. Think about PVT corners, there is no point in getting obsessive !

How can we extend this approach to more complex circuits ?

- Consider a slightly larger circuit (5T-OTA), that brings out most of the conceptual challenges that we need to cover
  - Larger designs will follow the same general flow

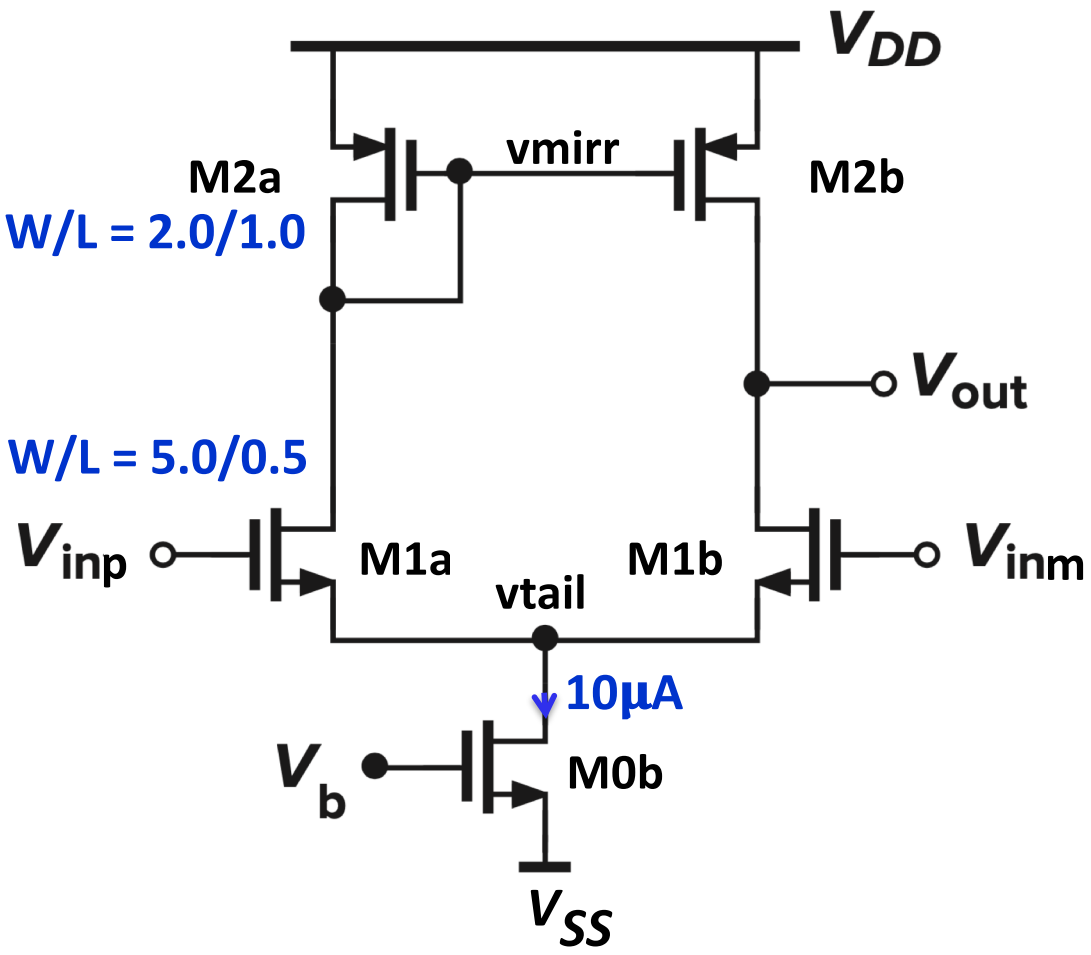
- Overarching goal

  design flow

  - Maintain a systematic and programmatic approach
  - As opposed to random SPICE tweaking

Example: 5T-OTA

Conventional Schematic

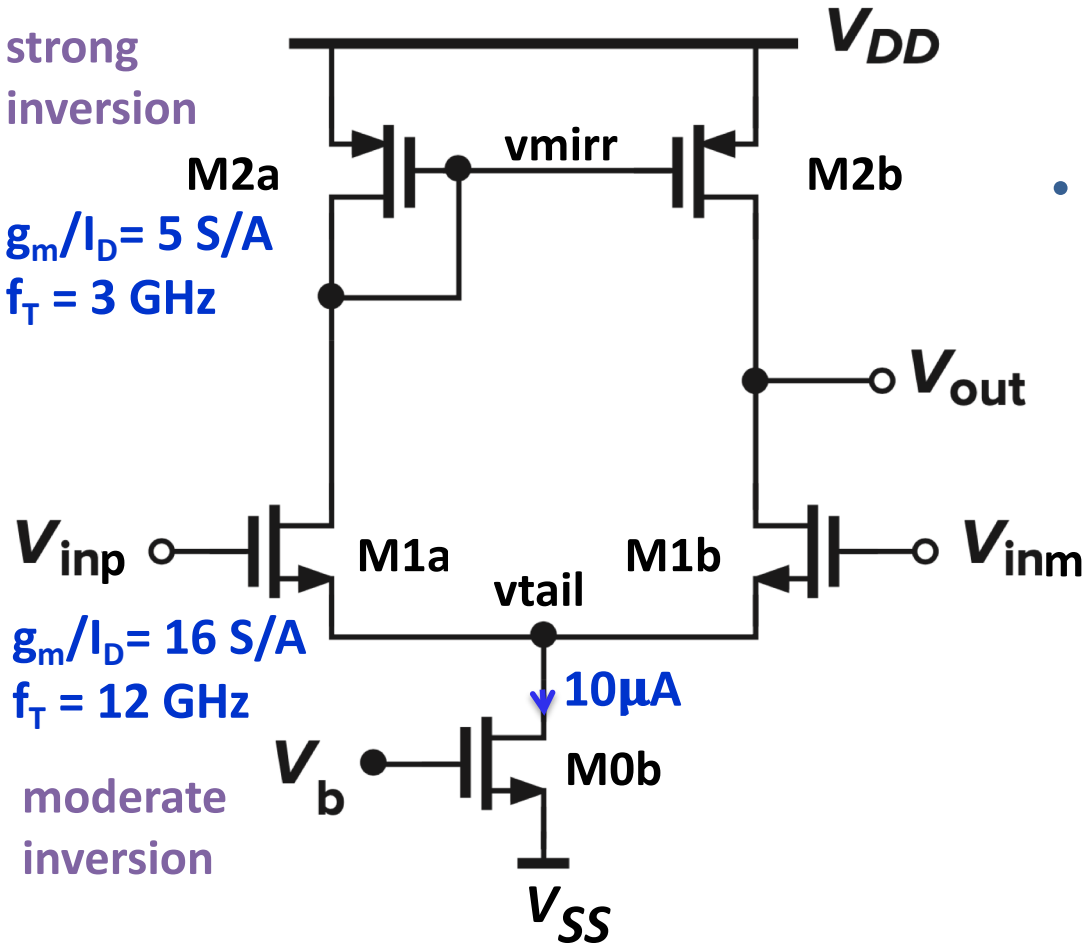


- Good for layout stage, not much else
- Can't deduce useful design insights/intuition from this schematic
- Why care about W/L?
- During design, we worry about gain, bandwidth, noise, etc.
  - W/L tells us nothing about this unless we fire up a simulator

Not insightful information for Design (useful for layout)

Example: 5T-OTA

A Better "proxy" for Design (valid in all technologies)

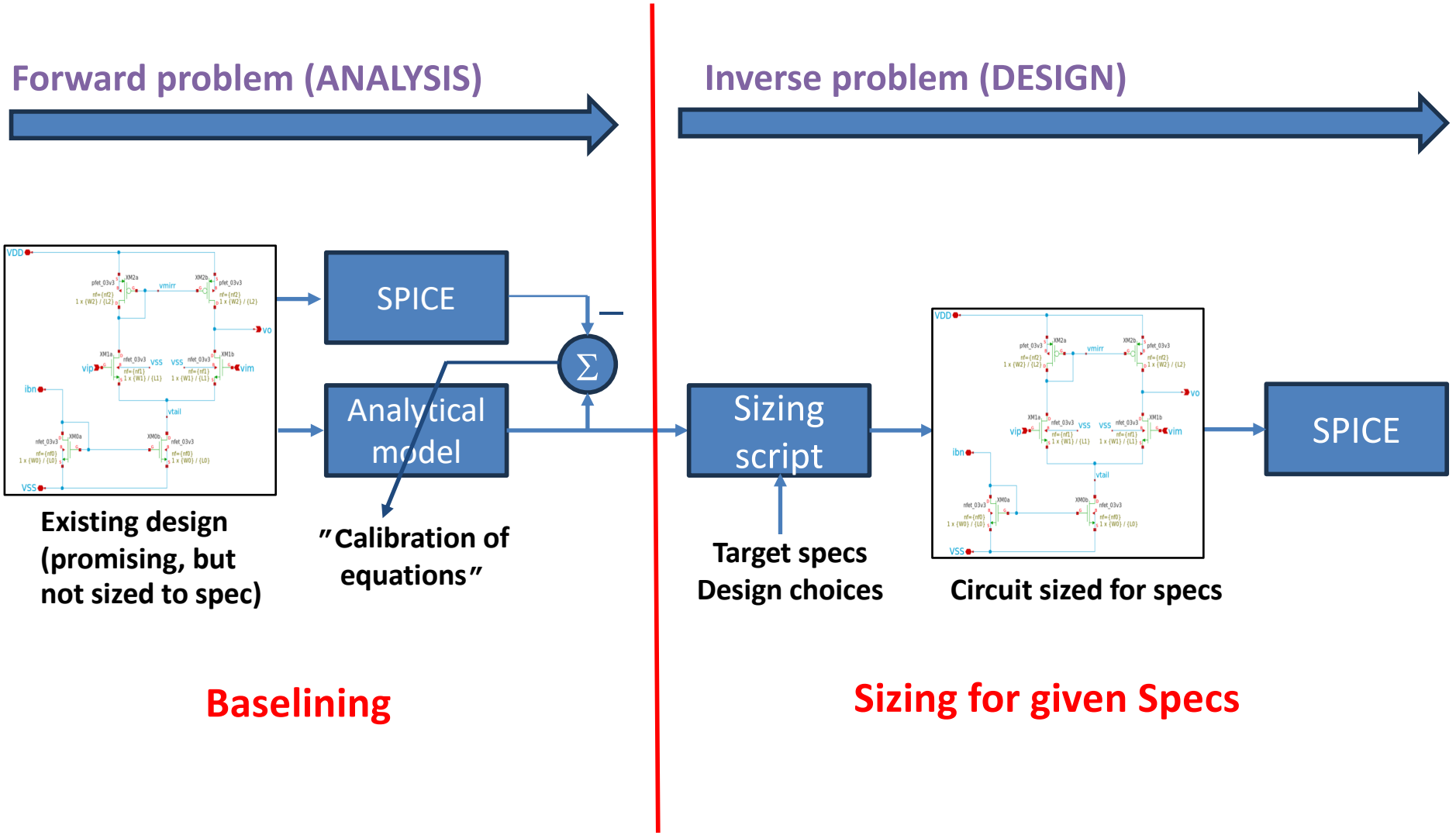


- Can directly "see" important info
  - Input pair in moderate inversion
  - Linear input range about V* = 2/16 V
  - Mirror in strong inversion
  - Pole-zero doublet at 1.5/3 GHz
  - Noise overhead from mirror is
  - about 25 %

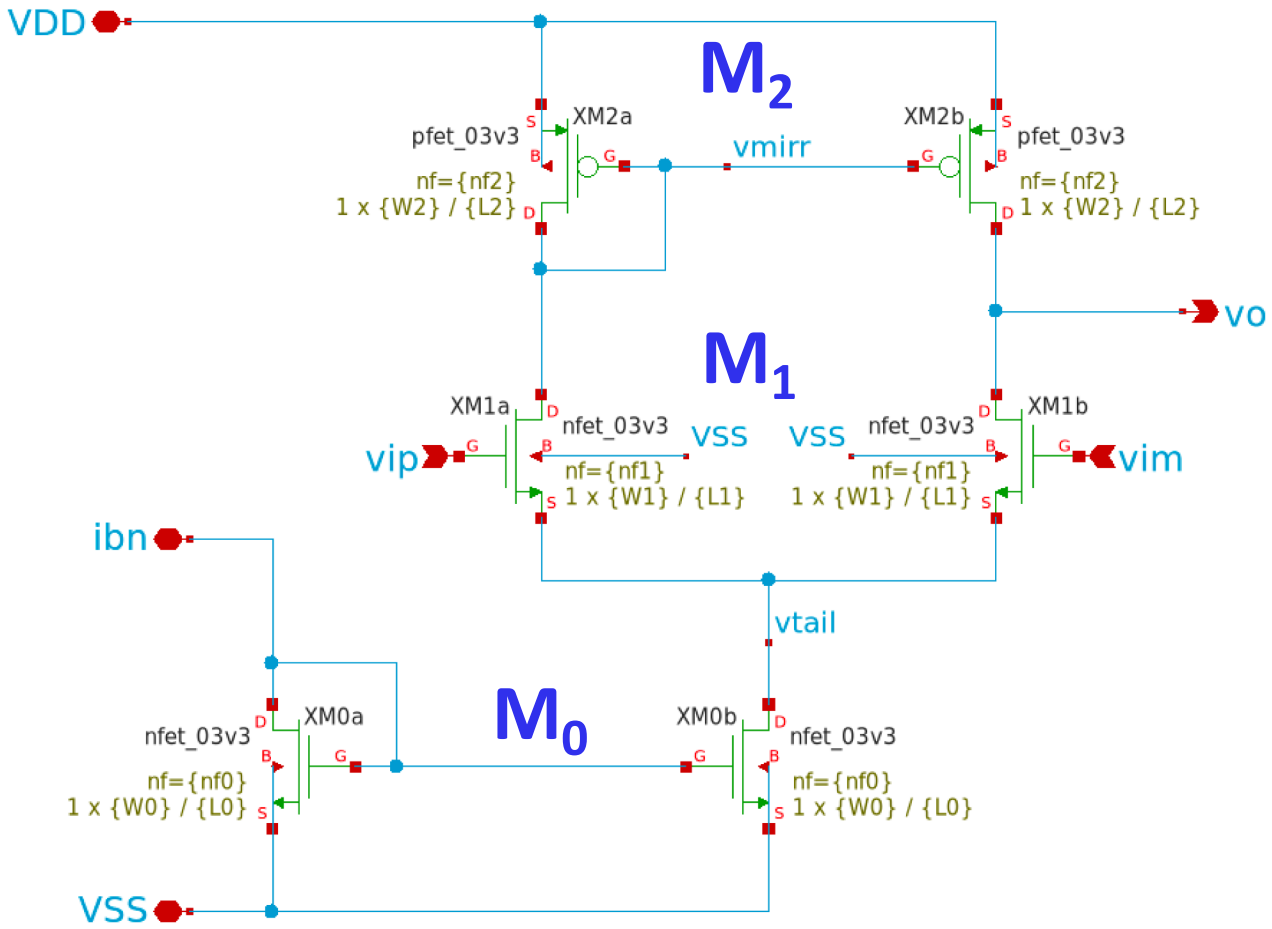
…

We can easily back-annotate the schematic through DC operating point

Design Flow Overview



Baseline Design (ota-5t_gf.sch)

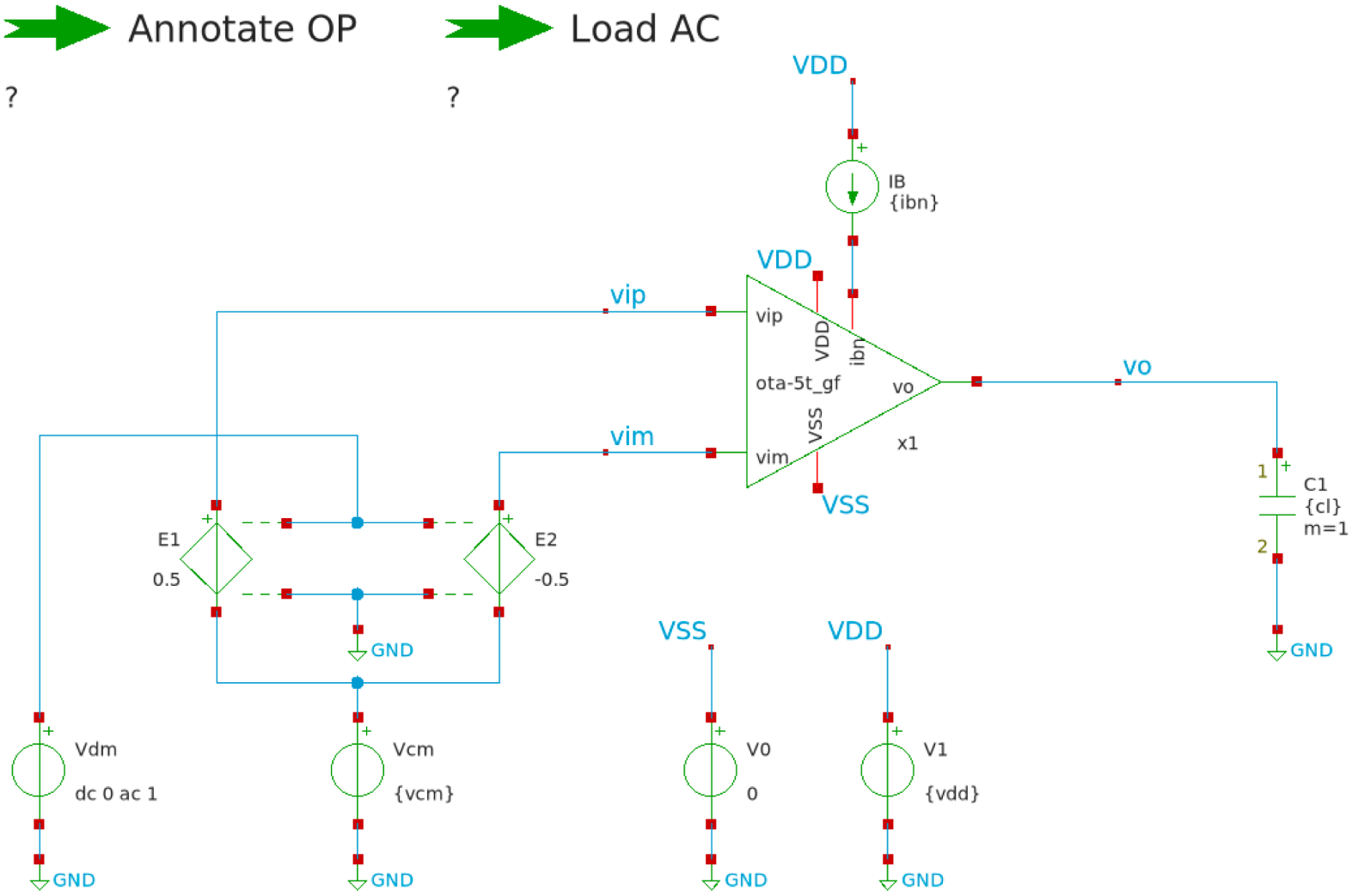


M_0 → bias
M_1 → workhorse
M_2 → mirror

Get started "recycling" some existing design

```
.param ib = 10.0e-06
.param cl = 1.0e-12
.param w0 = 2.0e-06
.param l0 = 1.0e-06
.param nf0 = 1.00
.param w1 = 5.0e-06
.param l1 = 0.5e-06
.param nf1 = 1.00
.param w2 = 2.0e-06
.param l2 = 1.0e-06
.param nf2 = 1.00
```

Testbench for Baseline Design (baseline-ota-5t_gf_tb.sch)



COMMANDS

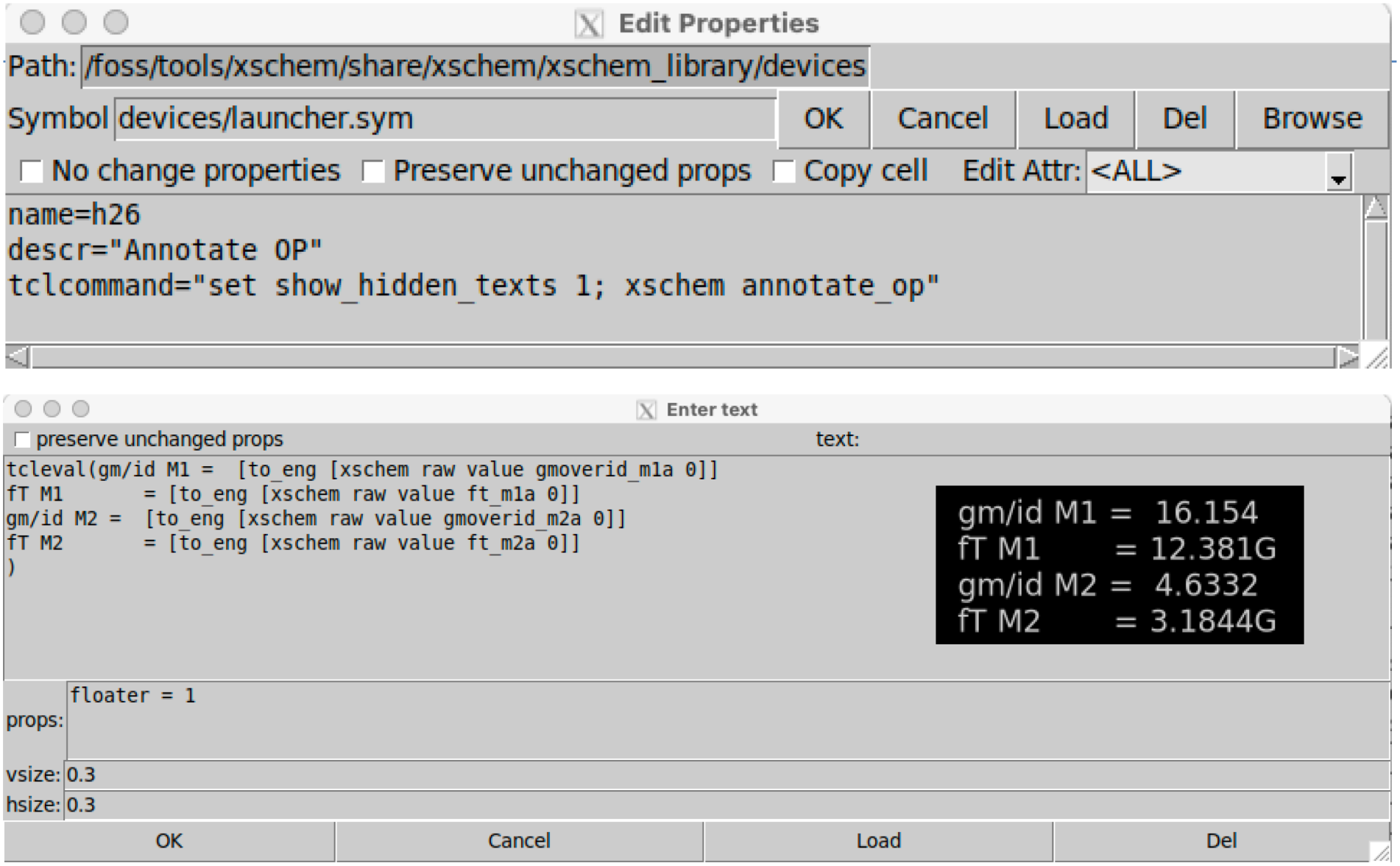
```
.lib /foss/pdks/gf180mcuD/libs.tech/ngspice/sm141064.ngspice typical
.inc /foss/pdks/gf180mcuD/libs.tech/ngspice/design.ngspice
.param vdd=3 vcm=1.5 cl=1p
.param ibn=10e-6
.param W0=2e-06 W1=5e-06 W2=2e-06
.param L0=1e-06 L1=0.5e-06 L2=1e-06
.param nf0=1.00 nf1=1.00 nf2=1.00
.option savecurrents

.control
    save all
    op
    show
    let gmoverid_m1a = {@m.x1.xm1a.m0[gm]/@m.x1.xm1a.m0[id]}
    let gmoverid_m1b = {@m.x1.xm1b.m0[gm]/@m.x1.xm1b.m0[id]}
    let gmoverid_m2a = {@m.x1.xm2a.m0[gm]/@m.x1.xm2a.m0[id]}
    let gmoverid_m2b = {@m.x1.xm2b.m0[gm]/@m.x1.xm2b.m0[id]}
    let ft_m1a = {@m.x1.xm1a.m0[gm]/(@m.x1.xm1a.m0[cgg]+@m.x1.xm1a.m0[cgso]+@m.x1.xm1a.m0[cgdo])}
    let ft_m1b = {@m.x1.xm1b.m0[gm]/(@m.x1.xm1b.m0[cgg]+@m.x1.xm1b.m0[cgso]+@m.x1.xm1b.m0[cgdo])}
    let ft_m2a = {@m.x1.xm2a.m0[gm]/(@m.x1.xm2a.m0[cgg]+@m.x1.xm2a.m0[cgso]+@m.x1.xm2a.m0[cgdo])}
    let ft_m2b = {@m.x1.xm2b.m0[gm]/(@m.x1.xm2b.m0[cgg]+@m.x1.xm2b.m0[cgso]+@m.x1.xm2b.m0[cgdo])}
    echo gmoverid_m1a = $&gmoverid_m1a
    echo gmoverid_m1b = $&gmoverid_m1b
    echo ft_m1a = $&ft_m1a
    echo ft_m1b = $&ft_m1b
    echo gmoverid_m2a = $&gmoverid_m2a
    echo gmoverid_m2b = $&gmoverid_m2b
    echo ft_m2a = $&ft_m2a
    echo ft_m2b = $&ft_m2b
    write baseline-ota-5t_gf_tb.raw
    set appendwrite

    ac dec 20 1 10e10
    let vo_mag = abs(v(vo))
    let vo_phase_margin = phase(v(vo)) * 180/pi + 180
    meas ac A0 max vo_mag
    let ref = A0/sqrt(2)
    meas ac BW when vo_mag=ref fall=1
    let GBW = A0*BW
    meas ac UGF when vo_mag=1 fall=1
    meas ac PM find vo_phase_margin when vo_mag=1
    echo A0 = $&A0
    echo GBW = $&GBW
    echo UGF = $&UGF
    echo PM = $&PM
    remzerovec
    write baseline-ota-5t_gf_tb.raw
.endc
```

```
gm/id M1 =  16.154
fT M1    = 12.381G
gm/id M2 =  4.6332
fT M2    = 3.1844G
```

```
A0: 139.34
GBW: 12.477M
UGF: 12.471M
PM: 88.398
```

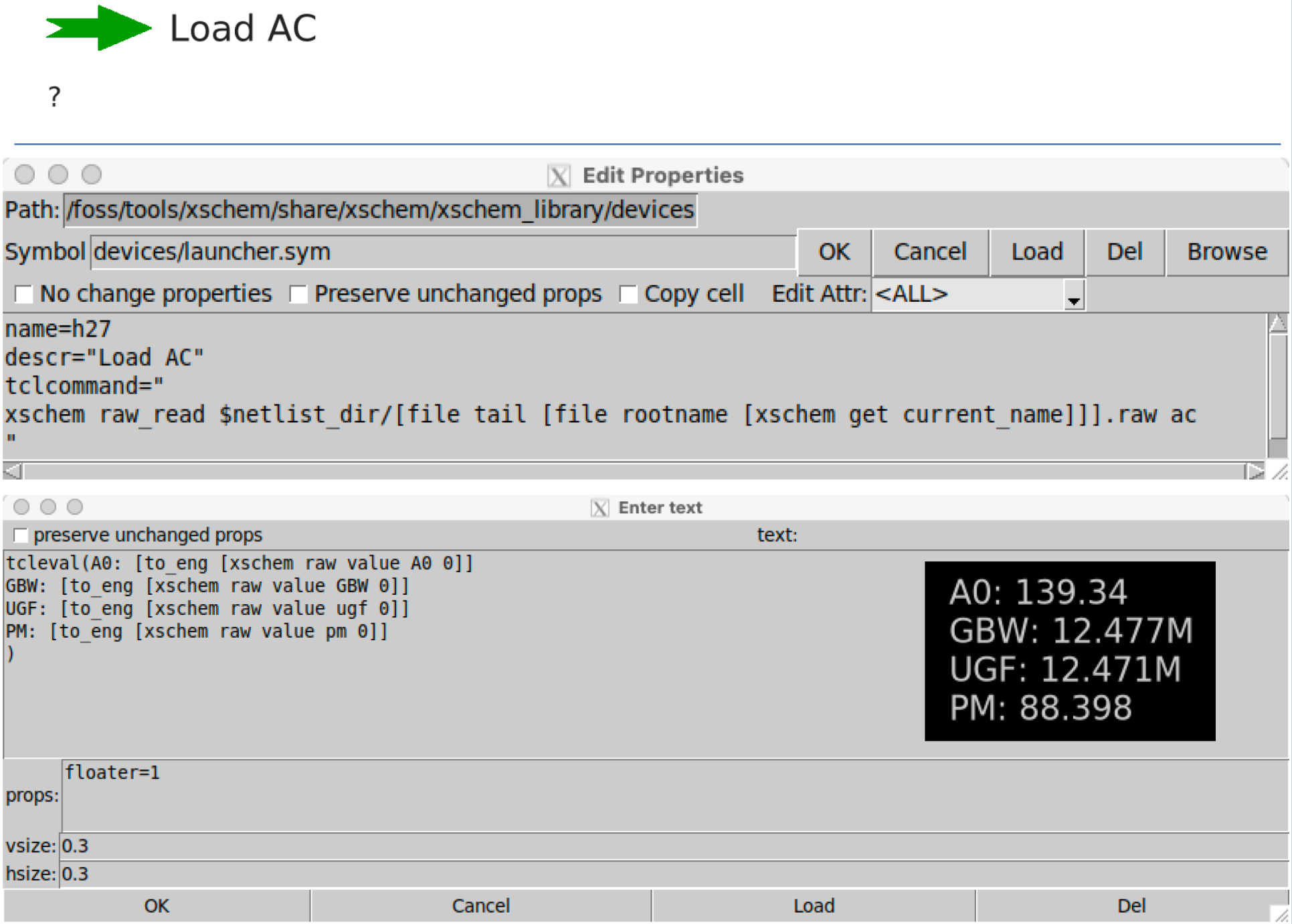
Annotate OP

?

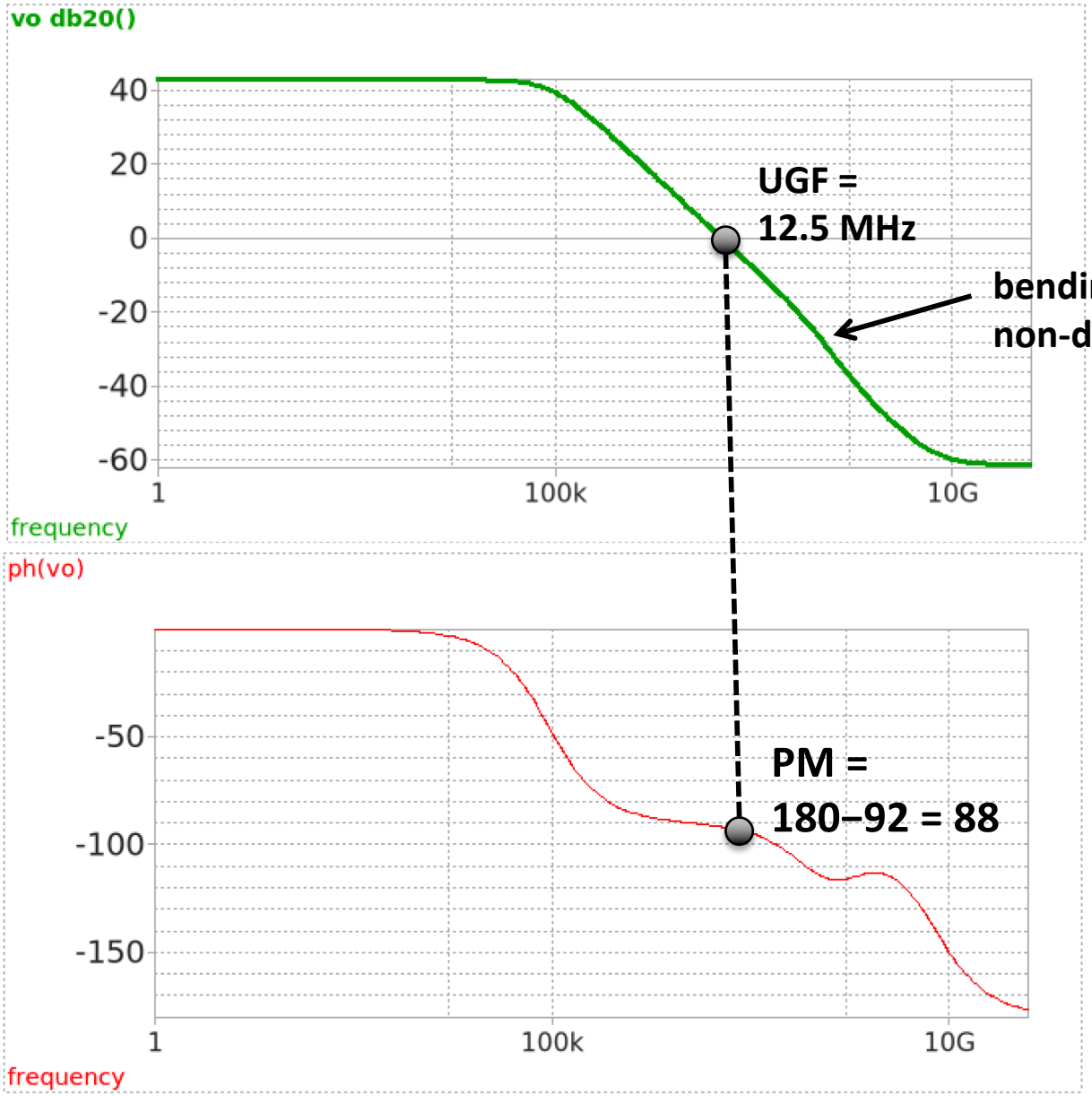

Edit Properties
Path: /foss/tools/xschem/share/xschem/xschem_library/devices
Symbol devices/launcher.sym
OK
Cancel
Load
Del
Browse
No change properties
Preserve unchanged props
Copy cell
Edit Attr: <ALL>
name=h26
descr="Annotate OP"
tclcommand="set show_hidden_texts 1; xschem annotate_op"
Enter text
preserve unchanged props
text:
tcleval(gm/id M1 = [to_eng [xschem raw value gmoverid_m1a 0]]
fT M1 = [to_eng [xschem raw value ft_m1a 0]]
gm/id M2 = [to_eng [xschem raw value gmoverid_m2a 0]]
fT M2 = [to_eng [xschem raw value ft_m2a 0]]
)
gm/id M1 = 16.154
fT M1 = 12.381G
gm/id M2 = 4.6332
fT M2 = 3.1844G
props:
floater = 1
vsize: 0.3
hsize: 0.3
OK
Cancel
Load
Del

Load AC

?


Edit Properties
Path: /foss/tools/xschem/share/xschem/xschem_library/devices
Symbol devices/launcher.sym
OK
Cancel
Load
Del
Browse
No change properties
Preserve unchanged props
Copy cell
Edit Attr: <ALL>
name=h27
descr="Load AC"
tclcommand="
xschem raw_read $netlist_dir/[file tail [file rootname [xschem get current_name]]].raw ac
"
Enter text
preserve unchanged props
text:
tcleval(A0: [to_eng [xschem raw value A0 0]]
GBW: [to_eng [xschem raw value GBW 0]]
UGF: [to_eng [xschem raw value ugf 0]]
PM: [to_eng [xschem raw value pm 0]]
)
A0: 139.34
GBW: 12.477M
UGF: 12.471M
PM: 88.398
props:
floater=1
vsize:
0.3
hsize:
0.3
OK
Cancel
Load
Del

Baseline AC Simulation Results



At this stage of the design flow we may have only an incomplete subset of high level specs.

In practice, we'll never have all specs. We need to learn to make decisions!
Baselining helps figuring out more specs.

In the baseline design the specs are not met, but can we analytically predict the performance ?

Can I write equations that gives numbers matching the baseline simulation results ?

Baseline Notebook

baseline_ota-5t_gf.ipynb

Import Packages

```
import numpy as np
import pandas as pd
from pygmid import Lookup as lk
```

[1] ✓ 1.2s

Technology data

```
n = lk('../nfet_03v3.mat')
p = lk('..//pfet_03v3.mat')
```

[2] ✓ 2.3s

Specifications

```
CL=1e-12; id0 = 10e-6
```

[3] ✓ 0.0s

Design choices

```
gm_id0 = 5
gm_id1 = 16 # power efficient at cost of large gate cap.
gm_id2 = 5  # lower noise from mirror, higher non-dominant pole
l0 = 1
l1 = 0.5
l2 = 1
```

[4] ✓ 0.0s

Sizing and benchmarking

```
# calculate the gm
id1 = id2 = id0/2
gm1 = gm_id1*id1
gm2 = gm_id2*id2

# estimate GBW
GBW = gm1/CL/2/np.pi

# compute DC gain
gm_gds1 = n.lookup('GM_GDS', GM_ID=gm_id1, L=l1)
gds1 = gm1/gm_gds1
gm_gds2 = p.lookup('GM_GDS', GM_ID=gm_id2, L=l2)
gds2 = gm2/gm_gds2
A0 = gm1/(gds1 + gds2)

# size all transistors
jd1 = n.lookup('ID_W', GM_ID=gm_id1, L=l1)
w1 = id1/jd1
jd2 = p.lookup('ID_W', GM_ID=gm_id2, L=l2)
w2 = id1/jd2
w0 = w1       # executive decision

# estimate mirror pole
cgg2 = w2*p.lookup('CGG_W', GM_ID=gm_id2, L=l2)
cdd2 = w2*p.lookup('CDD_W', GM_ID=gm_id2, L=l2)
cdd1 = w2*n.lookup('CDD_W', GM_ID=gm_id1, L=l1)
fp2 = gm2/(2*cgg2+cdd1+cdd2)/(2*np.pi)
CLtot = CL + cdd1 + cdd2

# estimate phase margin (mirror pole, LHP zero, RHP zero)
phip2 = -np.arctan(GBW/fp2)*180/np.pi
fz2 = 2*fp2
phiz2 = +np.arctan(GBW/fz2)*180/np.pi
cgd1 = w1*n.lookup('CGD_W', GM_ID=gm_id1, L=l1)
fz3 = gm1/cgd1/(2*np.pi)
phiz3 = -np.arctan(GBW/fz3)*180/np.pi
PM = 90 + phip2 + phiz2 +phiz3

df = pd.DataFrame( [id1/1e-6, gm1/1e-6, gm2/1e-6, A0, GBW*1e-6, fp2, PM], \
                   ['id1 (uA)', 'gm1 (uS)', 'gm2 (uS)', 'A0 (V/V)', 'GBW (MHz)', 'fp2 (Hz)', 'PM (deg)'], columns=['Value']);

pd.set_option('display.float_format', '{:.2e}'.format); df
```

[5] ✓ 0.0s

	Value
id1 (uA)	5.00e+00
gm1 (uS)	8.00e+01
gm2 (uS)	2.50e+01
A0 (V/V)	1.53e+02
GBW (MHz)	1.27e+01
fp2 (Hz)	2.02e+08
PM (deg)	8.82e+01

SPICE

id1: 4.94 uA
gm1: 79.83 uS
gm2: 22.90 uS
A0: 139.34
GBW: 12.47 MHz

PM: 88.40

```
    # finger the devices
    wfing = 5
    nf0 = 1+np.floor_divide(w0, wfing)
    nf1 = 1+np.floor_divide(w1, wfing)
    nf2 = 1+np.floor_divide(w2, wfing)
    df = pd.DataFrame( [(id0*1e6, id1*1e6, id2*1e6), (w0, w1, w2), (l0, l1, l2), (nf0, nf1, nf2)], \
                       ['ID (uA)', 'W (um)', 'L (um)', 'nf'], columns=['M0', 'M1', 'M2']); df.round(2)
```

[6] ✓ 0.0s

	M0	M1	M2
ID (uA)	10.00	5.00	5.00
W (um)	4.83	4.83	2.25
L (um)	1.00	0.50	1.00
nf	1.00	1.00	1.00

Lessons from baseline effort

```
    CLtot = CL + cdd1 + cdd2
    print(f'CLtot = {CLtot*1e12:0.4f} pF')
    print(f'fp2 = {fp2*1e-6:0.4f} MHz')
    print(f'fz2 = {fz2*1e-6:0.4f} MHz')
    print(f'fz3 = {fz3*1e-6:0.4f} MHz')
    print(f'phip2 = {phip2:0.2f} degree')
    print(f'phiz2 = {phiz2:0.2f} degree')
    print(f'phiz3 = {phiz3:0.2f} degree')
```

[7] ✓ 0.0s

```
CLtot = 1.0036 pF
fp2 = 202.0059 MHz
fz2 = 404.0117 MHz
fz3 = 19050.1233 MHz
phip2 = -3.61 degree
phiz2 = 1.81 degree
phiz3 = -0.04 degree
```

- **RHP zero insignificant for PM (and perhaps also LHP zero, which gives only about half the phase of the non-dominant pole)**
- **Device caps are negligible compared to the load cap.**

SPICE's OP output

```
show m.x1.xm0b.m0 : gm : gds : id : vgs : vbs : vds
show m.x1.xm1b.m0 : gm : gds : id : vgs : vbs : vds
show m.x1.xm2b.m0 : gm : gds : id : vgs : vbs : vds
```

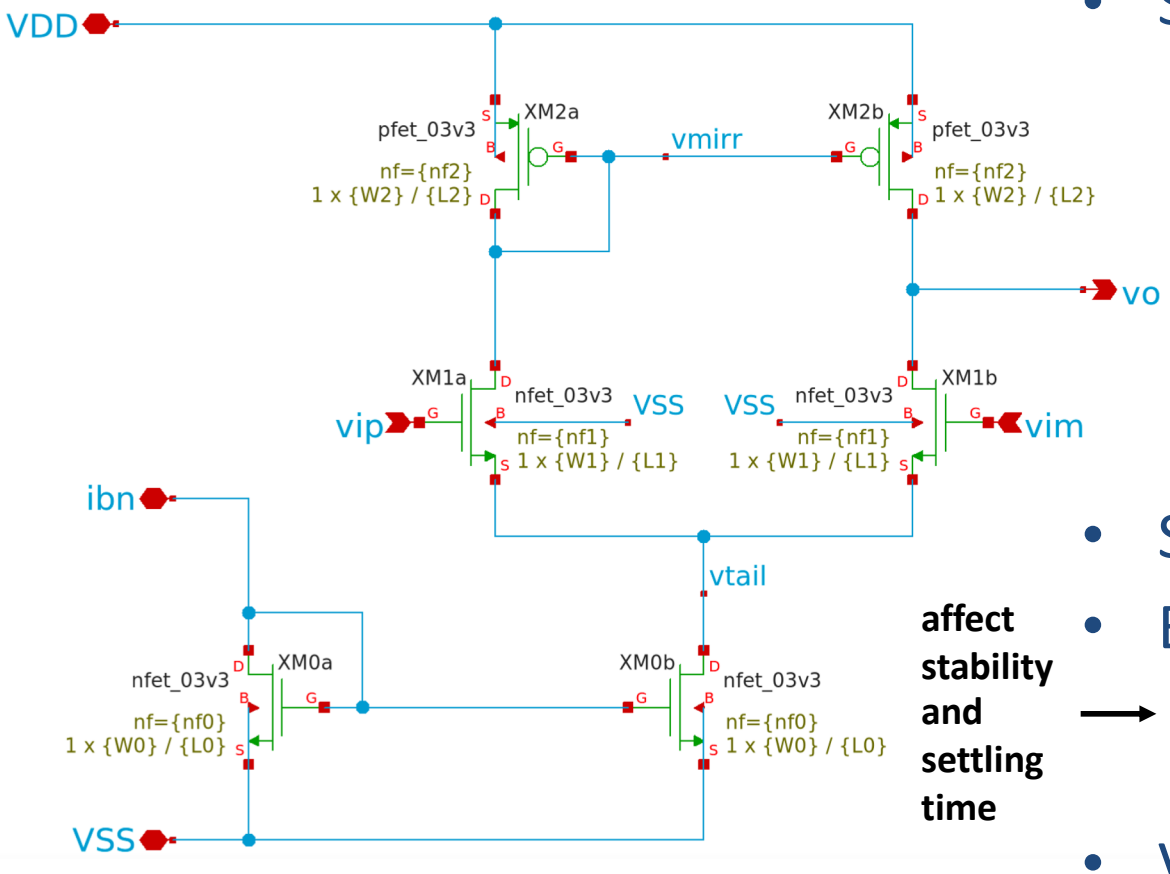
	M0	M1	M2
gm	6.62615e-05	7.98334e-05	2.28975e-05
gds	4.1822e-07	4.71716e-07	9.99486e-08
vdsat	0.249146	0.100143	0.324162
id	9.88417e-06	4.94211e-06	4.94209e-06
vbs	-6.22703e-06	-0.589809	-3.11352e-06
vgs	0.939757	0.910191	1.17694
vds	0.589796	1.23325	1.17694
gm/id	**6.70**	**16.15**	**4.63**
gm/gds	**158.44**	**169.24**	**229.09**

The discrepancies between the LUTs and SPICE comes from the fact that the bias mirror (M0a and Mob) is not exactly 1:1 and the VDS are lower than the default used in the LUTs (1.65V)

At this point we are done with the baseline analysis, and we are ready for design !

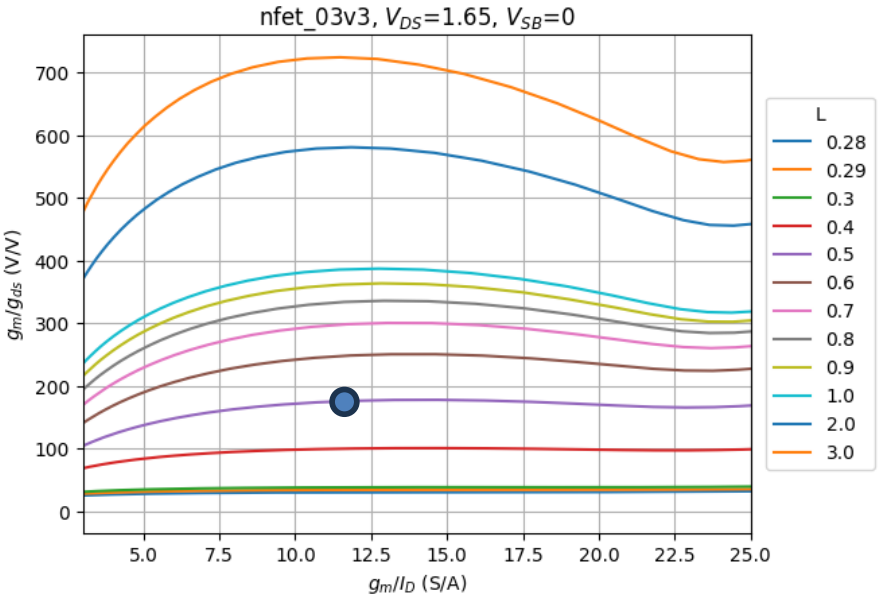
Design: Sizing Example (5T-OTA)

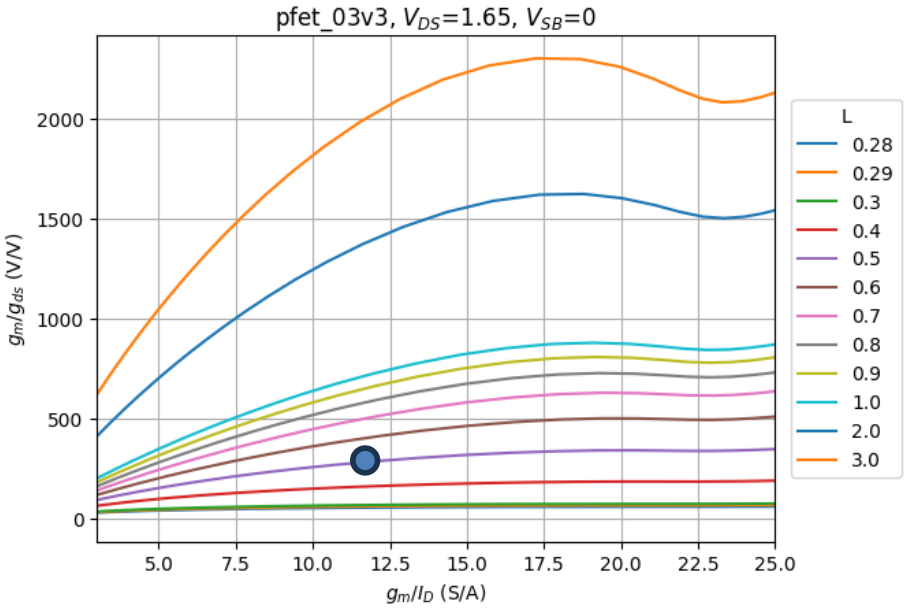
ota-5t_gf.sch



- Specs and design choices
  - GBW = 100 MHz with C_L = 1 pF
  - Pick channel lengths to achieve DC gain > 50
  - Use g_m/I_D = 12 S/A for all transistors ← **close to moderate inversion**
- Size the circuit
- Estimate → **affect stability and settling time**
  - Location of mirror pole
  - Phase margin
- Verify using OP and AC simulations

Pick Channel Lengths for A_{v0} > 50





$$A_{v0} = \frac{g_{m1}}{g_{ds1} + g_{ds2}} = \frac{\frac{g_{m1}}{g_{ds1}}}{1 + \frac{g_{ds2}}{g_{ds1}}} \approx \frac{1}{2}\frac{g_{m1}}{g_{ds1}}$$

$L = 0.5\ \mu m$ should leave some margin

Give yourself some margin. We need g_m/g_{ds} > 100. Don't sweat the choice too much !!

Sizing Notebook

sizing_ota-5t_gf.ipynb

Sizing and benchmarking

```
# calculate gm of differential pair
gm1 = 2*np.pi*GBW*CL

# size all transistors
id1 = gm1/gm_id1; id2=id1; id0=2*id1
jd1 = n.lookup('ID_W', GM_ID=gm_id1, L=l1)
w1 = id1/jd1
jd2 = p.lookup('ID_W', GM_ID=gm_id2, L=l2)
w2 = id1/jd2
w0 = 2*w1

# estimate mirror pole
cgg2 = w2*p.lookup('CGG_W', GM_ID=gm_id2, L=l2)
cdd2 = w2*p.lookup('CDD_W', GM_ID=gm_id2, L=l2)
cdd1 = w2*n.lookup('CDD_W', GM_ID=gm_id1, L=l1)
gm2 = gm1
fp2 = gm2/(2*cgg2+cdd1+cdd2)/(2*np.pi)

# estimate phase margin (mirror pole, LHP zero, RHP zero)
phip2 = -np.arctan(GBW/fp2)*180/np.pi
fz2 = 2*fp2
phiz2 = +np.arctan(GBW/fz2)*180/np.pi
cgd1 = w1*n.lookup('CGD_W', GM_ID=gm_id1, L=l1)
fz3 = gm1/cgd1/(2*np.pi)
phiz3 = -np.arctan(GBW/fz3)*180/np.pi
PM = 90 +phip2 +phiz2 +phiz3

# calculate the gain
gm_gds1 = n.lookup('GM_GDS', GM_ID=gm_id1, L=l1)
gds1 = gm1/gm_gds1
gm_gds2 = p.lookup('GM_GDS', GM_ID=gm_id2, L=l2)
gds2 = gm2/gm_gds2
A0 = gm1/(gds1 + gds2)


df = pd.DataFrame( [id1/1e-6, gm1/1e-3, A0, fp2, PM], \
                   ['id1 (uA)', 'gm1 (mS)', 'A0 (V/V)', 'fp2 (Hz)', 'PM (deg)'], columns=['Value']);
pd.set_option('display.float_format', '{:.4e}'.format); df
```

[5] ✓ 0.0s

	Value
id1 (uA)	5.2360e+01
gm1 (mS)	6.2832e-01
A0 (V/V)	1.0938e+02
fp2 (Hz)	2.8143e+08
PM (deg)	8.0359e+01

```
# finger the devices
wfing = 5
nf0 = 1+np.floor_divide(w0, wfing)
nf1 = 1+np.floor_divide(w1, wfing)
nf2 = 1+np.floor_divide(w2, wfing)
df = pd.DataFrame( [(id0*1e6, id1*1e6, id2*1e6), (w0, w1, w2), (l0, l1, l2), (nf0, nf1, nf2)], \
                   ['ID (uA)', 'w (um)', 'l (um)', 'nf'], columns=['M0', 'M1', 'M2']);
pd.set_option('display.float_format', '{:.2e}'.format); df
```

[6] ✓ 0.0s

	M0	M1	M2
ID (uA)	1.05e+02	5.24e+01	5.24e+01
w (um)	3.85e+01	1.92e+01	6.65e+01
l (um)	5.00e-01	5.00e-01	5.00e-01
nf	8.00e+00	4.00e+00	1.40e+01

Write spice include file

```
with open('sizing_ota-5t_gf.param', 'w') as file:
    file.write(".param ibn = " + "{:.2e}".format(id0) + '\n')
    file.write(".param W0 = " + "{:.2e}".format(w0*1e-6) + '\n')
    file.write(".param W1 = " + "{:.2e}".format(w1*1e-6) + '\n')
    file.write(".param W2 = " + "{:.2e}".format(w2*1e-6) + '\n')
    file.write(".param L0 = " + "{:.2e}".format(l0*1e-6) + '\n')
    file.write(".param L1 = " + "{:.2e}".format(l1*1e-6) + '\n')
    file.write(".param L2 = " + "{:.2e}".format(l2*1e-6) + '\n')
    file.write(".param nf0 = " + "{}".format(nf0) + '\n')
    file.write(".param nf1 = " + "{}".format(nf1) + '\n')
    file.write(".param nf2 = " + "{}".format(nf2) + '\n')
```

[7] ✓ 0.0s

A quick sanity check

```
CLtot = CL+ cdd1 + cdd2
print(f'CLtot = {CLtot*1e12:0.2f} pF' )
```

[8] ✓ 0.0s

```
CLtot = 1.11 pF
```

SPICE's OP Output

```
show m.x1.xm0b.m0 : gm : gds : vdsat : id : vbs : vgs : vds
show m.x1.xm1b.m0 : gm : gds : vdsat : id : vbs : vgs : vds
show m.x1.xm2b.m0 : gm : gds : vdsat : id : vbs : vgs : vds
```

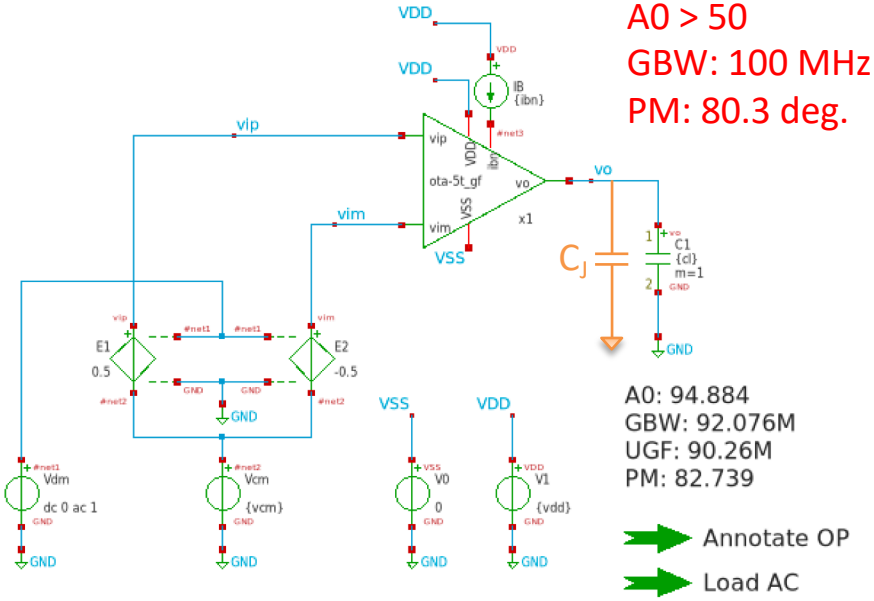
The first thing to do is to look at the Operating Point of the transistors

	M0	M1	M2
gm	0.00122843	0.000617758	0.000613944
gds	1.09258e-05	3.52038e-06	2.97305e-06
vdsat	0.141189	0.142775	0.141523
id	0.00010239	5.1195e-05	5.11951e-05
vbs	-3.35094e-06	-0.535732	-9.70012e-07
vgs	0.803628.	0.964268	0.900784
vds	0.535722	1.56348	0.900783
gm/ID	**11.99**	**12.07**	**11.99**
gm/gds	**112.43**	**175.48**	**206.50**

The value of vdsat issued by SPICE is a bit odd, use V*=2/(gm/ID) instead

Testbench with AC results

The design does not require any tweaking!



Expected
A0 > 50
GBW: 100 MHz
PM: 80.3 deg.

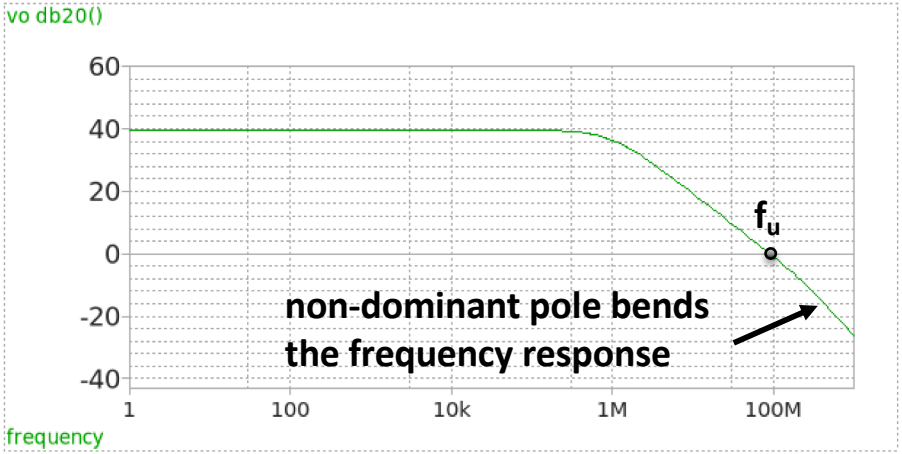
A0: 94.884
GBW: 92.076M
UGF: 90.26M
PM: 82.739

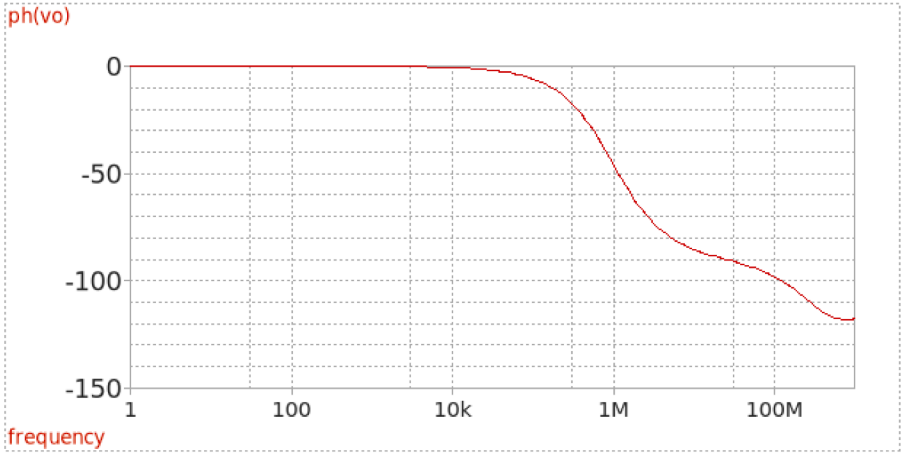
Annotate OP
Load AC

```
COMMANDS

.lib /foss/pdks/gf180mcuD/libs.tech/ngspice/sm141064.ngspice typical
.inc /foss/pdks/gf180mcuD/libs.tech/ngspice/design.ngspice
.inc ./sizing_ota-5t_gf.param
.param vdd=3 vcm=1.5 cl=1p
.option savecurrents

.control
    save all
    op
    show
    write ota-5t_gf_tb.raw
    set appendwrite

    ac dec 20 1 10e9
    let vo_mag = abs(v(vo))
    let vo_phase_margin = phase(v(vo)) * 180/pi + 180
    meas ac A0 max vo_mag
    let ref = A0/sqrt(2)
    meas ac BW when vo_mag=ref fall=1
    let GBW = A0*BW
    meas ac UGF when vo_mag=1 fall=1
    meas ac PM find vo_phase_margin when vo_mag=1
    echo $&A0 $&GBW $&UGF $&PM
    remzerovec
    write ota-5t_gf_tb.raw
.endc
```

GBW is somewhat smaller due to:

1. We did not include the junction cap at the output
2. impact of non dominant mirror pole fp2





What do we do for PVT ?
Design in nominal and pre-distort specs to have enough margin