MCCI
Microelectronic Circuits Centre Ireland

PyGMID

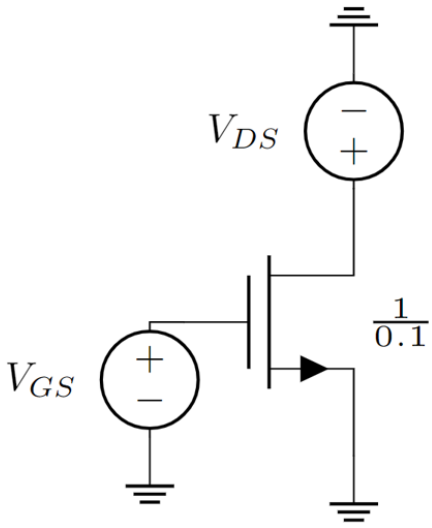
Author : Cian O'Donnell

Date : 26th September 2023

1 Introduction

PyGMID is an open source python3 g_m/I_D toolkit. It emulates and extends upon an existing MATLAB based implementation.

Devices are characterised by logging small signal characteristics for a predefined range of DC bias points (see fig. 1). During design, small signal characteristics can be *quickly* retrieved from look up tables in a scripting interface ($\approx 200\,\text{ms}$) . If the query point is not available within the grid, data is linearly interpolated across four dimensions: $V_{GS}, V_{DS}, L, V_{SB}$. If a query point does not lie on the grid, data is extrapolated.



(a)

$g_m\,(V_{GS}, V_{DS})$ [uS]

V_{GS} [V] \ V_{DS} [V]	0.4	0.5	0.6	0.7
0.4	532.738	582.085	627.285	668.851
0.5	721.088	760.610	793.410	821.989
0.6	791.855	833.667	865.109	891.058

```
gm = NCH.look_up('GM', vds=0.5, vgs=0.6)
print(f"gm is: {gm}")
```

```
>> gm is: 833.667 [uS]
```

(b)

Figure 1: (a) Operating point simulation for device characterisation (b) Sample lookup table

The toolkit allows designers to reformulate amplifier design equations in terms of meaningful ratios like transconductance efficiency g_m/I_D and transit frequency g_m/C_{gg}. A founding assumption is that transistor characteristics are linearly dependent on width meaning devices can be sized easily using drain current density I_D/W. Proper formulation enables visualisation of efficiency, speed and area trade-offs . Paul Jespers offers a comprehensive guide to g_m/I_D design [1].

1.1 Notation

In this manual, a python script looks like this:

```
from pygmid import Lookup as lk
# this is a python script
print("I am a python script")
```

A python console session looks like this:

```
>>> from pygmid import Lookup as lk
>>> # this is a python console session
>>> print("I am a python console session")
I am a python console session
```

A bash session looks like this:

```
$ cd zookeeper
$ echo I am a bash shell
I am a bash shell
```

2 Setup

The package can be used on either the MCCI server/laptop or a personal laptop. The following guide consists of:

1. A short guide specific for those using the software on the MCCI servers using premade scripts
2. A supplementary guide for those looking to set up a more personalised installation on their own computer/MCCI servers

2.1 MCCI: Minimal Setup

PyGMID is currently available on mcci-d0,d6,d7,d8.

1. Navigate to `/mcci/gmid/pygmid`. Copy `start.sh`, `setup.sh`, `requirements.txt`, and `hello_world.py` to a directory of your choice. For example, a `gmid` directory in your project workspace or your linuxhome directory
2. Make both scripts executable:

```
$ chmod +X ./setup.sh
$ chmod +X ./start.sh
```

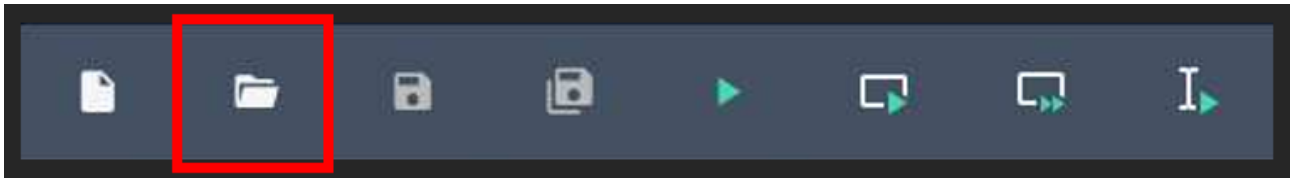
3. Run `setup.sh`. This can take up to 10 min to run.

```
$ ./setup.sh
```

4. Each time you want to launch Spyder and PyGMID, run `start.sh`:

```
$ ./start.sh
```

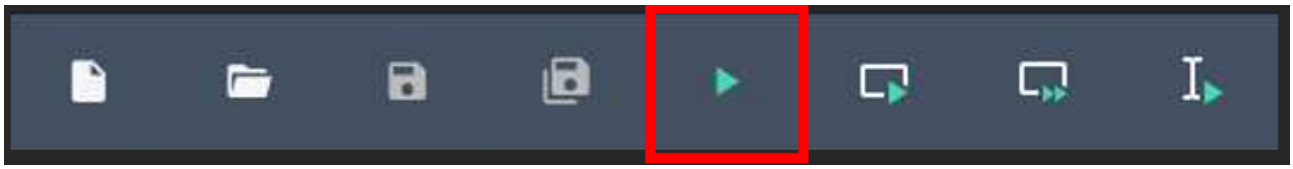
5. Spyder will launch with the `pygmid` plugin. Undock the plugin.
6. At this point, you will need to get copies of some characterised MOS data. On the MCCI server, this data is stored at `/mcci/gmid/`. In this example, the 2 V `90nm GPDK` data from Cadence is used. Copy the data to the same directory as your `hello_world.py` script.
7. Within Spyder, open `hello_world.py`. Change the string variable `fname` to the name of your data file. In the case of `gpdk090`, this filename is `"90n2vrvt.mat"`.

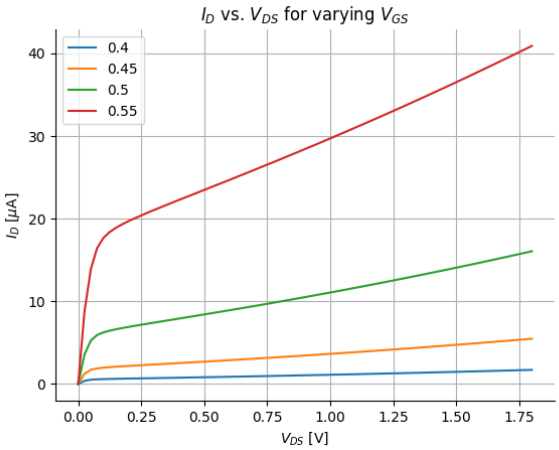


```
from pygmid import Lookup as lk
import numpy as np
from matplotlib import pyplot as plt
import matplotlib as mpl

##############################
# put your data filename here
fname = "MY DATA FILENAME"
##############################
```

8. Run the script within Spyder. You should be presented with the transistor I-V characteristic. Setup is now complete.





2.2 Advanced Setup

2.2.1 Virtual Environment

A virtual environment can be set up using a bash terminal. The following example creates a virtual environment called `pygmid`, sources the relevant technology PDK, and sources the virtual environment activation script. Replace `gpdk_script` with the appropriate technology.

```
$ python3.8 -m venv pygmid
$ source /mcci/gmid/gpdks/cian.odonnell/gpdk_script
$ source ./pygmid/bin/activate.csh
```

Listing 1: Creating `pygmid` virtual environment

After creation of the virtual environment, it is necessary to upgrade pip (the python package manager). This can be done as follows:

```
$ python -m pip install --upgrade pip
```

Listing 2: Upgrading pip

It is recommended to install the following packages when using python3.

Figure 2: Sample `requirements.txt` file

Packages can be placed together in a single file separated by newlines (called `requirements.txt`), or installed individually using `pip`. In the following terminal, all packages in `requirements.txt` are installed at once and an additional package (`numpy`) is installed individually:

```
$ python -m pip install -r requirements.txt
$ python -m pip install numpy
```

Listing 3: Installing packages

After the initial installation, it is not necessary to reinstall packages. The virtual environment can be sourced after opening a new terminal session. Make sure to source your PDK first.

```
$ source /mcci/gmid/gpdks/cian.odonnell/gpdk_script
$ source ./pygmid/bin/activate.csh
```

Listing 4: Sourcing virtual environment and PDK after opening new bash session

2.2.2 PyGMID Setup

If `pygmid` was included in your `requirements.txt` file, this step is not necessary.

`pygmid` can be installed using pip in a bash terminal as follows:

```
$ python -m pip install pygmid
```

Alternatively, the source code can be downloaded from Github. To install the package from source, navigate to the root directory and run pip locally:

```
$ cd ~/Downloads/pygmid
$ python -m pip install .
```

3 Usage

3.1 Lookup Class

The `PyGMID` package consists of a single `Lookup` class. You can easily import the class into a scripting environment:

```
from pygmid import Lookup as lk
```

...or into the python interpreter:

```
>>> from pygmid import Lookup as lk
```

Instantiate a lookup object by loading device data into a class instance. For example, the following script loads data from `90n1rvt.pkl` into an object instance called `NCH`. You can directly access the sweep points using dict style access.

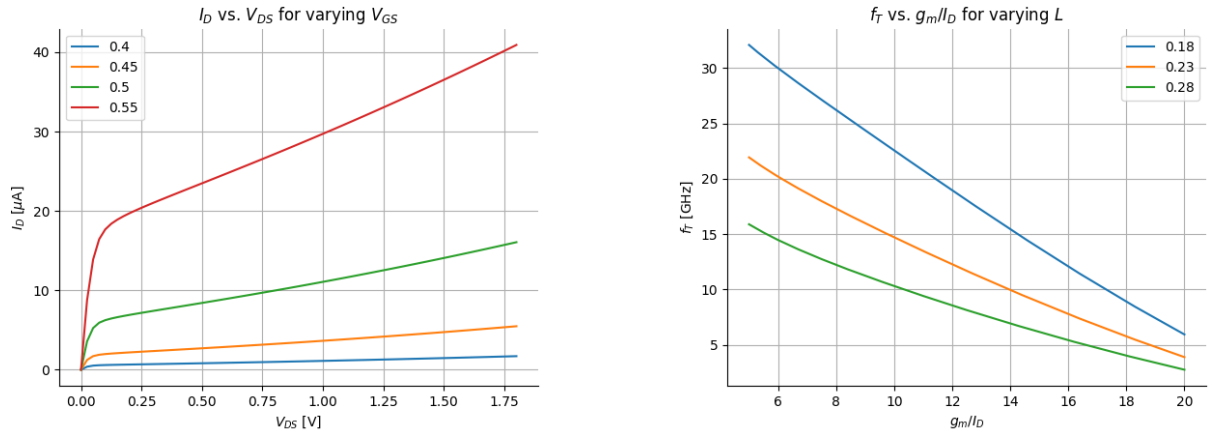
```
NCH = lk('90n1rvt.pkl')  # load MATLAB data into pygmid lookup object

VDSs = NCH['VDS']        # lookup object has pseudo-array access to data
```

Data can be retrieved through N-D interpolation using the `look_up` (or legacy `lookup`) function. For example, you can plot the transistor I-V characteristic (see fig. 3a).

```python
## Plot ID versus VDS
ID = NCH.look_up('ID', vds=VDSs, vgs=VGSs)
plt.figure()
plt.plot(VDSs, 1e6*ID.T)
plt.ylabel(r"$I_D$ [$\mu$A]")
plt.xlabel(r"$V_{DS}$ [V]")
plt.title(r'$I_D$ vs. $V_{DS}$ for varying $V_{GS}$')
plt.legend(VGSs)
plt.show()

# Plot ft against gm_id for different L
step = 0.1
gm_ids = np.arange(5, 20+step, step)
Ls = np.arange(min(NCH['L']),0.3,0.05)
s = time()
ft = NCH.look_up('GM_CGG', GM_ID=gm_ids, L
↪  =np.arange(min(Ls),0.3,0.05))/2/np.pi
e = time()
print(f"Time taken: {(e-s)*1000} [ms]")
plt.figure()
plt.plot(gm_ids, 1e-9*ft.T)
plt.ylabel(r"$f_T$ [GHz]")
plt.xlabel(r"$g_m/I_D$")
plt.title(r'$f_T$ vs. $g_m/I_D$ for varying $L$')
plt.legend(np.around(Ls, decimals=2))
plt.show()
```



(a) **(b)**

Figure 3: (a) Transistor I-V characteristic (b) Transistor transit frequency f_t vs. g_m/I_D for varying L

Other usage examples are given online.

3.2 Design Example

3.3 Lookup GUI

3.4 Spyder

Setting Up Interactive Plotting in Spyder

To enable interactive plotting in Spyder, you need to set the backend of Matplotlib to an interactive backend. Here's how to do it:

1. Open Spyder and navigate to `Tools > Preferences`
2. In the `Preferences` window, go to `IPython console > Graphics > Graphics Backend`
3. Select the `Automatic` option. This will allow Spyder to automatically choose the best backend for your system.

Setting Up a Cursor for Plotting in Spyder

Add the following line of code to your script:

```
cursor = Cursor(ax, useblit=True, color='red', linewidth=2)
```

4 Device Characterisation

`PySweep` is a built-in technology characterisation sweep. `PySweep` can be called directly from the command line with an appropriate configuration file, or interactively through a GUI in the Spyder IDE.

4.1 Command Line Interface

To parametrise a technology, make sure that your virtual environment is correctly set up and you have sourced your setup script.

Run the following command from the terminal:

```
$ python -m pygmid --mode sweep --config <config.cfg>
```

where `config.cfg` is an appropriate configuration file. Config files for most technologies are available at `/mcci/gmid/configs`. Alternatively, section 4.2 details how to generate your own configuration file.

4.2 Generating a Configuration

Listing 5 show a sample config file used to characterise `gpdk090`, a generic PDK from Cadence.

The config (`.cfg` extension) file is a generic preference file that stores settings and configuration info. Each section is declared on an individual line and surrounded by square brackets. In listing 5 there are two sections:

- `MODEL`
- `SWEEP`

A subsection component is declared on an individual line and formatted as follows:

```
<KEY> = <VALUE>
```

The following section explains how to identify and construct the `MODEL` section of a configuration file for a given technology:

file : this key specifies the location of the device (NMOS and PMOS) netlist. Typically, design kit installations are located at `/mcci-sw/cadence.64/design_kits/cadence/` . Locate the appropriate PDK installation and copy the working directory. You can do this in a bash terminal with the pwd command. Make sure to append the netlist file name to the working directory.

```
$ pwd
mcci-sw/cadence.64/design_kits/cadence/90nm/gpdk090_v4.6/models/spectre
```

After this, decide on the appropriate simulation corner. Listing 5 uses the nominal corner (NN). The value should be formatted as follows:

```
file = "<directory>" section=<corner>
```

You can explore alternative simulation corners in ADE->Setup->Model Libraries (see fig. 4) or by examining the model netlist.

info : this field is for the designer's personal use. Appropriate device info should be placed here in a single string

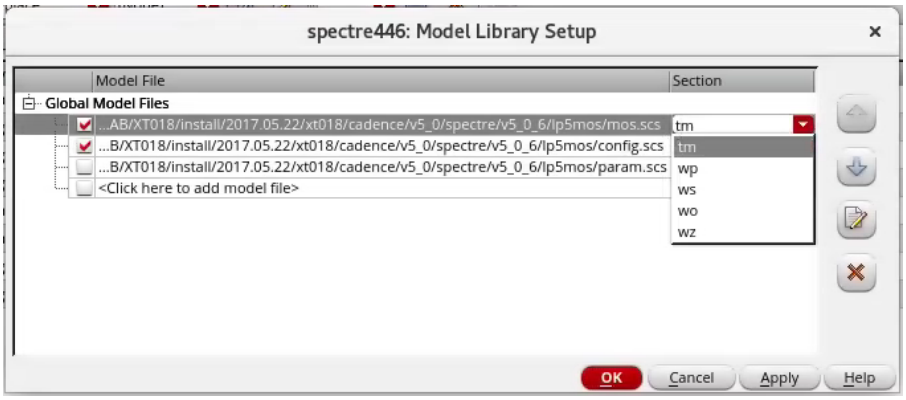


Figure 4: Available simulation corners

corner : this field is for the designer's personal use. The appropriate corner should be placed here in a single string

temp : simulation temperature. This field should be formatted as follows:

```
temp = <temperature>
```

where `temperature` is an integer in ° K

modeln : This information is available in the technology netlist, or a sub-netlist referenced by the technology netlist. For example, in the `gpdk090` netlist, this device is called `gpdk090_nmos1v` (see fig. 5).

```
section s1v_mos

*inline subckt gpdk090_nmoscap1v (D G S B)
*parameters l=0.1u w=10u simM=1 nrd=s1v_hdif_ne/w nrs=s1v_hdif_ne/w  as=1p ad=1p ps=1u
pd=1u
*gpdk090_nmoscap1v (D G S B) gpdk090_nmos1v  w=w l=l nrd=nrd nrs=nrs
*ends gpdk090_nmoscap1v

inline subckt gpdk090_nmos1v (D G S B)
parameters l=0.1u w=10u simM=1 nrd=s1v_hdif_ne/w nrs=s1v_hdif_ne/w  as=1p ad=1p ps=1u
pd=1u
+ garea=w*l  maforward=2.5e3*garea*gleak_scale   mareverse=.03*maforward
//mismatch
+ varvt =  .004   // 1 sigma Vt mismatch variation in unit of v-um
+ geo_fac =  0.7071 / sqrt(l*w*simM*1e12)
+ mm_delvt = varvt * geo_fac * pvt_mc
+ mm_mu0 = 1-(.005 * geo_fac * pu0_mc )
+ mm_dl= 2e-03  * geo_fac * l * pltw_mc
+ mm_dw = - 24e-03 * geo_fac * w *  pltw_mc
+ mm_dtox = 2e-03 * geo_fac * s1v_tox_ne *  pltw_mc
```

Figure 5: Modeln

modelp : see above

savefiln : At the end of the characterisation, N-type data will be stored in a python binary named `<savefiln>.pkl` .

savefilp : At the end of the characterisation, P-type data will be stored in a python binary named `<savefilp>.pkl` .

paramfile : This field should always be left as: `paramfile = params.scs`

mp : This field contains the parametric expressions for the P-type device. To get this data, open a new schematic in cadence and add a parametric PMOS transistor with `W=Wtot` and `l=L` (see fig. 6).

In an ADE session, click `Simulation->Netlist->Create` . In the parametrised netlist, you should see an instance identifier for your device. Copy all of the text after the model name. This is illustrated in fig. 7. Delete all backslash characters and delimit the text field with quotation marks. For example, the text in fig. 7 should look as follows:

```
"w=Wtot l=L as=4.8e-07*(Wtot) ad=4.8e-07*(Wtot) ps=1*2*(4.8e-07+(Wtot))
pd=2*(4.8e-07+(Wtot)) nrs=2.7e-07/(Wtot) nrd=2.7e-07/(Wtot) m=(1)*(1) par1=((1)*(1))"
```

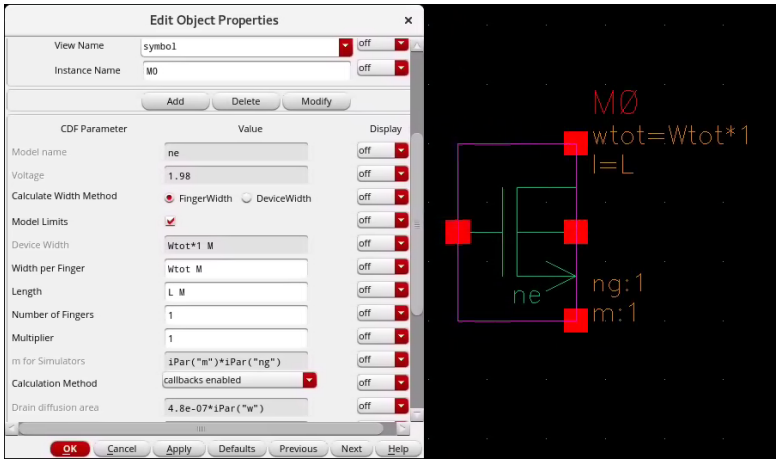


Figure 6: Parametric transistor

```
// Point Netlist Generated on: Aug 22 09:51:27 2023
// Generated for: spectre
// Design Netlist Generated on: Aug 22 09:51:27 2023
// Design library name: logturtle_cian
// Design cell name: test
// Design view name: schematic
simulator lang=spectre
global 0
parameters Wtot L

// Library name: logturtle_cian
// Cell name: test
// View name: schematic
M0 (net1 net3 net4 net2) ne w=Wtot l=L as=4.8e-07*(Wtot) ad=4.8e-07*(Wtot) \
        ps=1*2*(4.8e-07+(Wtot)) pd=2*(4.8e-07+(Wtot)) nrs=2.7e-07/(Wtot) \
        nrd=2.7e-07/(Wtot) m=(1)*(1) par1=((1)*(1))
simulatorOptions options psfversion="1.4.0" reltol=1e-3 vabstol=1e-6 \
    iabstol=1e-12 temp=27 tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 \
    maxnotes=5 maxwarns=5 digits=5 cols=80 pivrel=1e-3 \
    sensfile="../psf/sens.output" checklimitdest=psf
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub
```

Figure 7: Parametric netlist

mn : repeat above for N-type device

The following section explains how to identify and construct the `SWEEP` section of a configuration file for a given technology:

VGS : V_{GS} values to sweep. This key/value pair should be formatted as follows:

```
VGS = (<start>,<step>,<stop>)
```

VDS : V_{DS} values to sweep. This key/value pair should be formatted as follows:

```
VDS = (<start>,<step>,<stop>)
```

VSB : V_{SB} values to sweep. This key/value pair should be formatted as follows:

```
VSB = (<start>,<step>,<stop>)
```

LENGTH : L values to sweep. This value is a list of : `(<start>,<step>,<stop>)` arrays. Each successive array should start with a value higher than the stop value of the previous array. This allows for non-uniform spacing.

```
L = [(<start>,<step>,<stop>), ..., (<start>,<step>,<stop>)]
```

WIDTH : This should always be set to 1.

config.cfg

```
[MODEL]
file = "/mcci-sw/cadence.64/design_kits/cadence/90nm/gpdk090_v4.6/models/spectre/gpdk090.scs" section=NN
info = 90nm CMOS, cadence gpdk, BSIMv4
corner = NOM
temp = 300
modeln = gpdk090_nmos1v
modelp = gpdk090_pmos1v
savefilen = 90n1rvt
savefilep = 90p1rvt
paramfile = params.scs
# Make sure to indent the strings in this array or you'll get a DuplicateOptionError
mp = [

    # p-cell data

]
mn = [
    # p-cell data
]

[SWEEP]
#       (start   ,step,stop)
VGS =   (0       ,100e-3,1.8)
VDS = (0,200e-3,1.8)
VSB = (0,0.5,1)
LENGTH = [(0.1,0.05,0.2)]
WIDTH = 1
NFING = 1
```

Listing 5: A sample configuration file for gpdk090

References

[1] Paul Jespers. *The gm/ID Methodology, a sizing tool for low-voltage analog CMOS Circuits: The semi-empirical and compact model approaches.* Springer Science & Business Media, 2009.