Neuromorphic engineering I

Lab 1: Automated Data Acquisition and Analysis

Group number: 18

Team member 1: Alessandro Pasini

Team member 2: Quillan Favey

Date: 09.28.2022

You need to bring your own USB cable to connect the board with your own PC.

The objectives of this lab are as follows:

- To become acquainted with the experimental board setup.
- To become acquainted with Python for data acquisition and manipulation.
- To measure and characterize a NFET.

The aim of this first lab is to familiarize everyone with the lab equipment and software.

1. Getting started

1.1 Class chip documentation

You can find the documentation for the classchip here:
https://drive.google.com/drive/u/0/folders/1VBPKVfS9zwu_I2ExR1D0jU2eCSgleoQG

1.2 Python

If you are new to Python, you can find a detailed tutorial at the link:
https://docs.python.org/3/tutorial/ .

In particular the concepts in *3. An Informal Introduction to Python* and in *4. More Control FLow Tools* will be useful for the exercises you will have to solve.

1.3 Report

You hand in the report (prelab + lab report + postlab) in both `.ipynb` and `.pdf` format as a

group. Make sure that the markdowns are ran and the generated figures are shown.

The deadline is the beginning of the next lab.

There is no prelab for this week, but from the next week on you should finish the prelab before the lab starts.

1.4. Virtual machine

If you are a Linux user you can skip this step.

If you are a Windows or Mac user, we have set-up an Ubuntu-based virtual machine with the necessary libraries to communicate with the board. To import the existent virtual machine, Windows users should install VirtualBox. Mac users can either install VirtualBox or Parallels. If you want to use a pre-configured VirtualBox virtual machine with all the tools pre-installed then follow the instructions in the below.

To install VirtualBox, please follow these steps:

For Windows users

Step 1. Dowload VirtualBox

You can download VirtualBox from the link: https://www.virtualbox.org/wiki/Downloads , VirtualBox 6.1.26 platform packages -> Windows hosts

Execute the .exe file and follow the steps. In the window "Would you like to install this device software" select Install.

Once the installation is complete, press Finish and launch the VirtualBox.

Step 2. Download the Ubuntu 20.04 ISO file

You can download the necessary files for the NE-I virtual machine from the link: https://ubuntu.com/. We will use Ubuntu '20.04 LTS' so to download you have to go to previous versions. The ISO file can be found here: https://releases.ubuntu.com/20.04.5/?_ga=2.183376806.1236968508.1664199734-1751779002.1664199734

Step 3. Import the file in VirtualBox and install Ubuntu

Once you launch VirtualBox, the window Oracle VM VirtualBox Manager should appear on your screen. Then follow this link https://itsfoss.com/install-linux-in-virtualbox/ from 'Step 3: Install Linux using VirtualBox'.

For Mac users

Step 1. Dowload VirtualBox

You can download VirtualBox from the link: https://www.virtualbox.org/wiki/Downloads , VirtualBox 6.1.26 platform packages -> Windows hosts

Execute the .dmg file and follow the steps.

Once the installation is complete, press Finish and launch the VirtualBox.

Step 2. Download the Ubuntu 20.04 ISO file

You can download the necessary files for the NE-I virtual machine from the link: https://ubuntu.com/. We will use Ubuntu '20.04 LTS' so to download you have to go to previous versions. The ISO file can be found here: https://releases.ubuntu.com/20.04.5/?_ga=2.183376806.1236968508.1664199734-1751779002.1664199734

Step 3. Import the file in VirtualBox and install Ubuntu

Once you launch VirtualBox, the window Oracle VM VirtualBox Manager should appear on your screen. Then follow this link https://medium.com/tech-lounge/how-to-install-ubuntu-on-mac-using-virtualbox-3a26515aa869

2. Python exercises

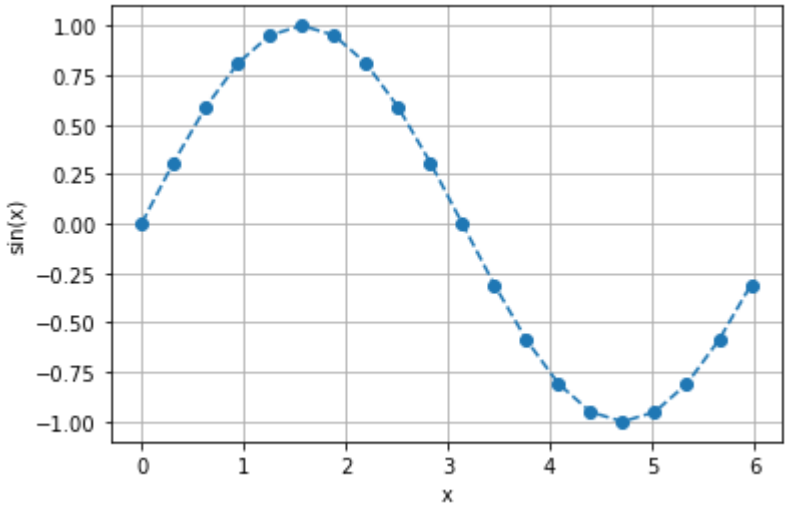
2.1 Jupyter Lab

You can install Jupyter Lab or Jupyter Notebook according to this link:https://jupyter.org/install

2.2 Making plots

- Plot a Sine curve from 0 to 2π with 20 points. (Hint: Do not forget to properly label the axis and add figure legends when necessary.)

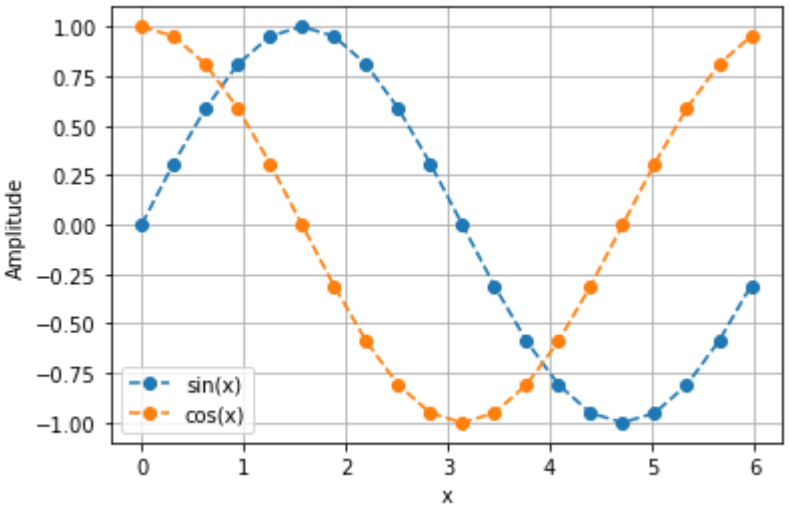
In [ ]:
```
#import sys
#!{sys.executable} -m pip install numpy
#!{sys.executable} -m pip install matplotlib
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0,np.pi*2,np.pi*2/20)
y = np.sin(x)
plt.plot(x, y, '--o')
plt.xlabel("x")
plt.ylabel('sin(x)')
#plt.legend('sin(x)')
plt.grid()
plt.show()
```



- Add a plot of a Cosine (Use the same range to the Sine plot, but set a different point marker and color.)

In [ ]:
```
#%matplotlib inline
x = np.arange(0,np.pi*2,np.pi*2/20)
y = np.sin(x)
x2 = np.arange(0,2*np.pi,2*np.pi/20)
y2 = np.cos(x2)

plt.plot(x, y, '--o')
plt.plot(x2,y2,'--o')
plt.xlabel("x")
plt.ylabel('Amplitude')
plt.legend(['sin(x)','cos(x)'])
plt.grid()
plt.show()
```

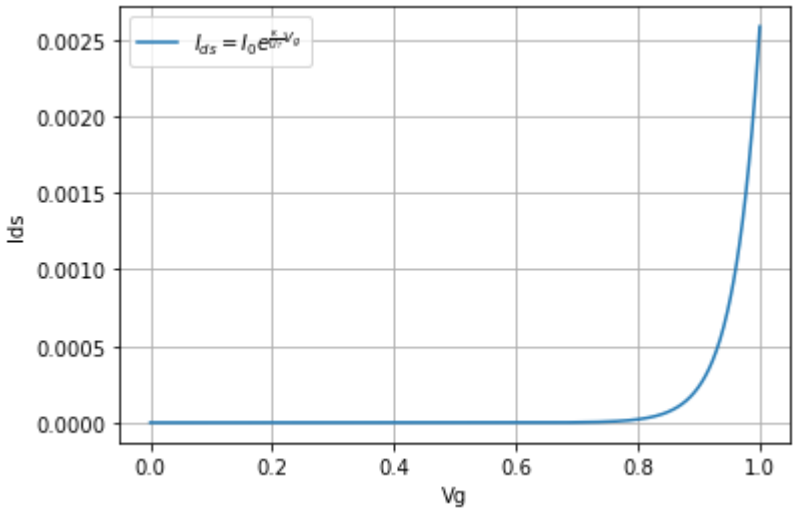


- Make a plot of the equation $I_{ds} = I_0 e^{\frac{\kappa}{U_T} V_g}$ using the following parameters. Generate two plots, one with linear scaling and one with log scaling on the y axis. Put the right labels on both x axis and y axis. (Hint: use `plt.semilogy`)

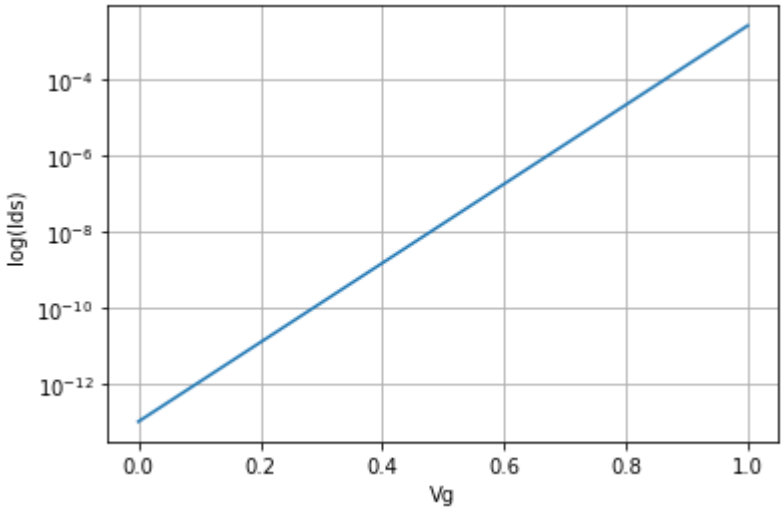
In [ ]:
```
Vg = np.arange(0, 1, 0.001)
I0 = 1e-13
k = 0.6
UT = 25e-3
```

In [ ]:
```
Ids = I0*np.exp(k/UT*Vg)
```

In [ ]:
```
#linearly scaled plot
plt.plot(Vg, Ids,label=r'$ I_{ds} = I_0 e^{\frac{\kappa}{U_T} V_g}$')
plt.xlabel("Vg")
plt.ylabel('Ids')
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
#log-scaled plot
plt.plot(Vg, Ids,label=r'$ I_{ds} = I_0 e^{\frac{\kappa}{U_T} V_g}$')
plt.xlabel("Vg")
plt.ylabel('log(Ids)')
plt.grid()
plt.semilogy()
plt.show()
```



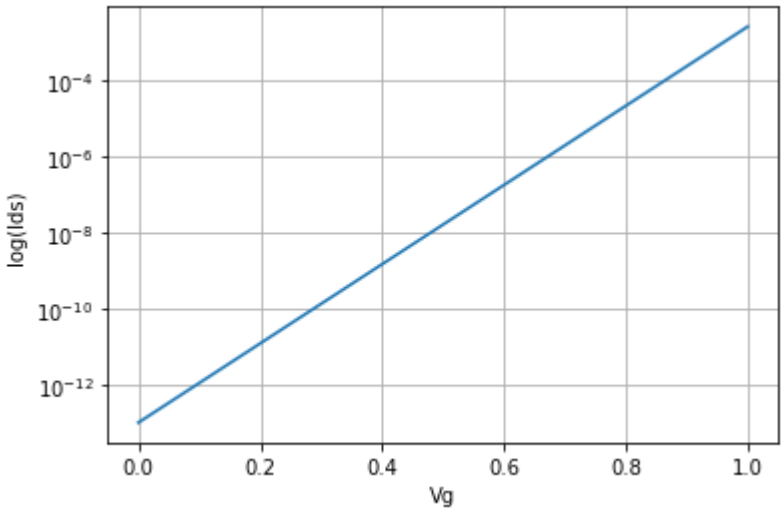
2.3 Saving and loading data

You may want to work on the data for your report after the lab. Use `np.savetxt('data.csv', data, delimiter=',')` to save it to the working directory.

In [ ]:
```
# define data
data = [Vg,Ids]
# save to csv file
np.savetxt('data.csv',data, delimiter = ',')
```

Check if the data saved is correct by loading it again using `np.loadtxt('data.csv', delimiter=',')` and plot.

In [ ]:
```
 # load from csv file
x, y = np.loadtxt('data.csv',delimiter=',')
# plot
plt.plot(x, y,label=r'$ I_{ds} = I_0 e^{\frac{\kappa}{U_T} V_g}$')
plt.xlabel("Vg")
plt.ylabel('log(Ids)')
plt.grid()
plt.semilogy()
plt.show()
```

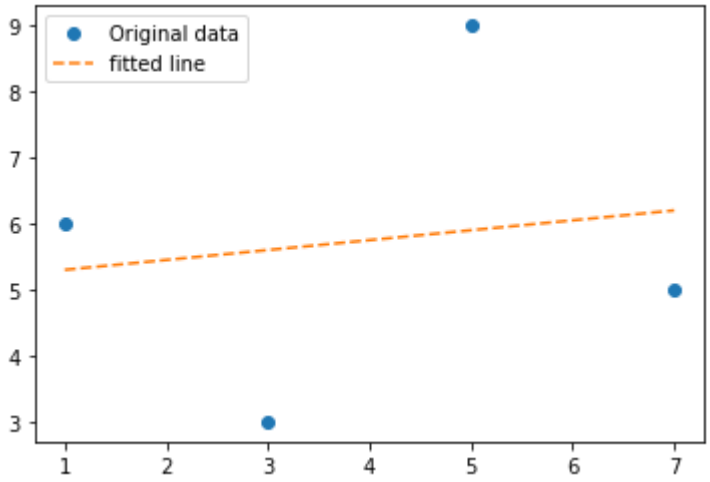


2.4 Fitting data with a line

Sometimes you may need to find the relationship between data using regression. Try to extract the slope and intercept of the following given data using linear regression with `np.polyfit`

In [ ]:
```
x = np.array([1, 3, 5, 7])
y = np.array([ 6, 3, 9, 5 ])
m, b = np.polyfit(x,y,deg=1)
# compare the orignial data points and the fitted line
fitted_line = [m * i + b for i in x]
plt.plot(x,y,'o')
plt.plot(x,fitted_line,'--')
plt.legend(['Original data','fitted line'])
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



---

3 Experiments

From now on you will be using the real board!

3.0 How to install pyplane

In order to communicate with the chip through Python you need to install *pyplane* , a library that provides an easy interface to control the chip from jupyter notebook. The interface requires to use Ubuntu 20.04. You can follow different methods based on your machine's operating system.

3.0.1 Installing *pyplane* using pip

Open a command window and run `pip install pyplane`

3.0.2 Set up the USB connection of the teensy board in the Virtual machine

Windows users

1. check teensy vendor and product id

lsusb

you should see something like "XXXX:YYYY" for teensy

ID 16c0:0483 Van Ooijen Technische Informatica Teensyduino Serial

1. add usb rules

cd /etc/udev/rules.d

sudo gedit 10-my-usb.rules

Add the information in this 10-my-usb.rules file and save it: ATTR{idVendor}=="XXXX", ATTR{idProduct}=="YYYY", MODE="0666", GROUP="dialout"

1. add user to group and change mode

sudo usermod -a -G dialout $USER

sudo chmod a+rw /dev/ttyACM0

1. restart the virtual machine

Mac Users

Step 1 Settings --> Ports --> USB Add Teensyduino USB Serial

Step 2 Device --> Connect

Step 3 Change mode: sudo chmod a+rw /dev/ttyACM0

Step 4 Restart kernel in Jupyter noytebook

Step 5 Run the code again

3.1 Load the firmware (you can skip this step)

You don't need to load the firmware,we have done it for you.

To load the firmware, please follow these steps:

Step 1. Dowload and install the Teensy Loader

wget https://www.pjrc.com/teensy/teensy_linux64.tar.gz

wget https://www.pjrc.com/teensy/00-teensy.rules

sudo cp 00-teensy.rules /etc/udev/rules.d/

tar -xvzf teensy_linux64.tar.gz

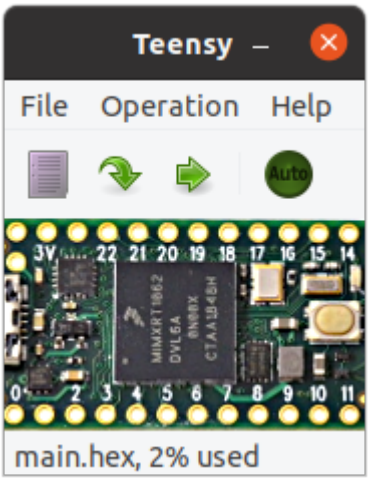
./teensy &

You will see

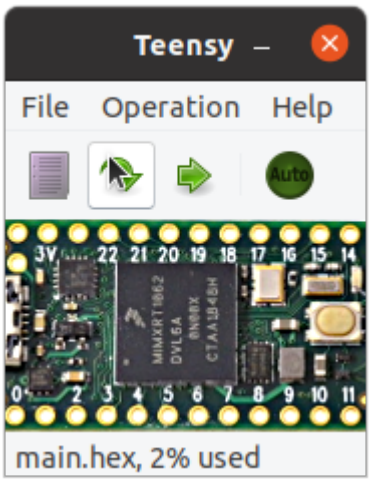
Step 2. Load main.hex

Connect the board to your computer by USB

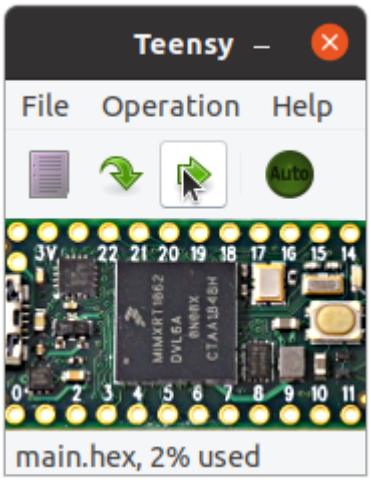
Open maim.hex file



Press button on Teensy to manually enter Program Mode



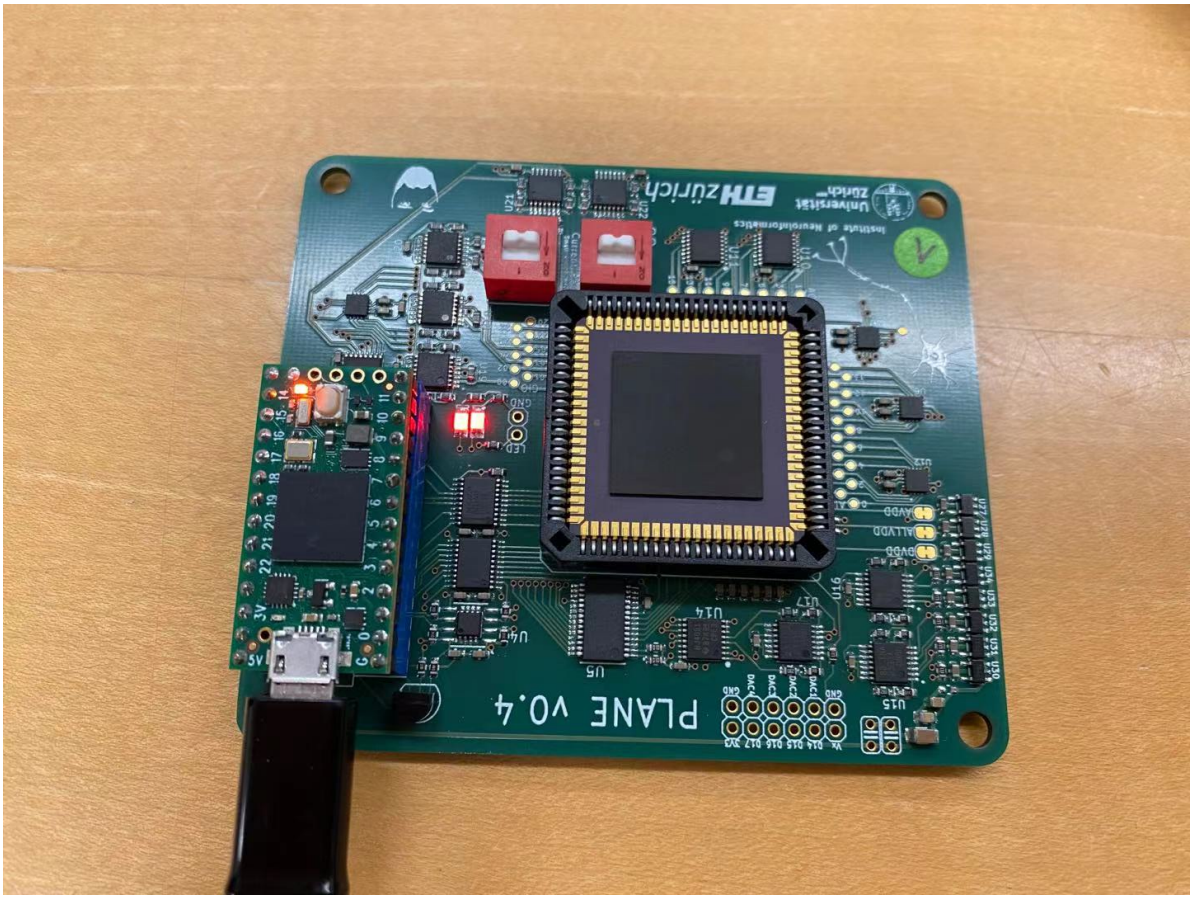
Program



Reboot

3.2 Set up the communication between jupyter notebook and PCB

Verify that the LEDs of your board are on as in the following picture



In [ ]:
```
# import the necessary library to communicate with the hardware
import sys
import pyplane
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
# create a Plane object and open the communication
if 'p' not in locals():
    p = pyplane.Plane()
    try:
        p.open('/dev/ttyACM0')
    except RuntimeError as e:
        print(e)
```

Make sure all these steps are executed correctly (the [*] in front of the line turns into a number)

In [ ]:
```
# Send a reset signal to the board
reset_type = pyplane.ResetType(0)
p.reset(pyplane.ResetType.Soft)
```

Out[ ]:
```
<TeensyStatus.Success: 0>
```

Was the LED flashing? How?

Yes, it blinked once.

In [ ]:
```
#NOTE: You must send this request events every time you do a reset operetion, otherwis
p.request_events(1)
```

You could check firmware version, which should be 1.8.3.

In [ ]:
```
p.get_firmware_version()
```

Out[ ]:
```
(1, 8, 6)
```

See all the possible functions

In [ ]:
```
dir(pyplane)
```

Out[ ]:
```
['AdcChannel',
 'BitDepth',
 'Coach',
 'CoachInputEvent',
 'CoachOutputEvent',
 'CurrentRange',
 'DacChannel',
 'Plane',
 'ResetType',
 'TeensyStatus',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 'get_version',
 'pyplane']
```

3.3 Basic function operation

You can set and read voltage using function p.set_voltage and p.read_voltage.

Very importantly, all voltage you set on this board must be between 0 and 1.8 V!

Now set a voltage at AIN0

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN0, 0)
```

Out[ ]:
```
0.0
```

Because of the quantization error of the DAC, you may want to see the actual value you have set using p.get_set_voltage

In [ ]:
```
p.get_set_voltage(pyplane.DacChannel.AIN0)
```

Out[ ]:
```
0.0
```

Read the voltage at adc channel AOUT0

In [ ]:
```
p.read_voltage(pyplane.AdcChannel.AOUT0)
```

Out[ ]:
```
0.4833984375
```

Read the voltage at adc channel GO22

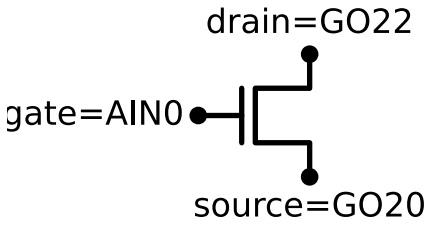
In [ ]:
```
p.read_current(pyplane.AdcChannel.GO22)
```

Out[ ]:
```
2.685546860448085e-07
```

3.4 AER

In [ ]:
```
# uses schemdraw, you may have to install it in order to run it on your PC
import schemdraw
import schemdraw.elements as elm
d = schemdraw.Drawing()
Q = d.add(elm.NFet, reverse=True)
d.add(elm.Dot, xy=Q.gate, lftlabel='gate=AIN0')
d.add(elm.Dot, xy=Q.drain, toplabel='drain=GO22')
d.add(elm.Dot, xy=Q.source, botlabel='source=GO20')
d.draw()
```

Out[ ]:



Read the current of the drain of the transistor.

In [ ]:
```
I_d = p.read_current(pyplane.AdcChannel.GO22)
print("The measured drain current is {} A".format(I_d))
```

```
The measured drain current is 2.685546860448085e-07 A
```

Do you think this current is reasonable? WHy?

First, we tested setting AIN0 voltage at 0.62 V. For AOUT0 we obtained a current of 4.6386719532165444e-07 A. Then, as you can see above, we obtained a current of 2.685546860448085e-07 A by setting the voltage of AIN0 at 0.0 V.

These current values are reasonable. In fact, no voltage was set for the source and the drain leading to Vds = 0. Therefore, we can consider this current as noise/leakage. Having tested the drain current with two different gate voltage values, the order of magnitude is the same (e-07).

Now try to set voltage of this transistor by AER (Address Event Representation).

Find the documentation "chip_architecture.pdf" for the classchip introduced in 1.1. See how to select signal communication on pages 11 and 21.

Because you need to read the current and write the voltage of the NFET. You have to set demultiplexer by sending the configuration event:

In [ ]:
```
events = [pyplane.Coach.generate_aerc_event( \
    pyplane.Coach.CurrentOutputSelect.SelectLine5, \
    pyplane.Coach.VoltageOutputSelect.NoneSelected, \
    pyplane.Coach.VoltageInputSelect.SelectLine2, \
    pyplane.Coach.SynapseSelect.NoneSelected, 0)]

p.send_coach_events(events)
```

Make sure the chip receives the event by a blink of LED1, if it's not the case, the chip is dead.

Now set source GO20 voltage using the function introduced above

In [ ]:
```
p.set_voltage(pyplane.DacChannel.GO20,1)
p.get_set_voltage(pyplane.DacChannel.GO20)
```

Out[ ]:
```
0.9994136095046997
```

Set drain GO22 voltage

In [ ]:
```
p.set_voltage(pyplane.DacChannel.GO22,0)
p.get_set_voltage(pyplane.DacChannel.GO22)
```

Out[ ]:
```
0.0
```

Set trial gate AIN0 voltage (you can try different voltage between 0~1.8V to see different output current)

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN0,1)
p.get_set_voltage(pyplane.DacChannel.AIN0)
```

Out[ ]:
```
0.9994136095046997
```

Read drain GO22 current

In [ ]:
```
I_d = p.read_current(pyplane.AdcChannel.GO22)
print("The measured drain current is {} A".format(I_d))
```

```
The measured drain current is 1.708984314063855e-07 A
```

Compare this current with the drain current measured above.

- *first drain current measurement (to compare):*

Test	Ids [A]	V_{GO20} $(source)$ [V]	V_{GO22} $(drain)$ [V]	V_{AIN0} $(gate)$ [V]
Test A	2.685546860448085e-07	not set	not set	0

- *second drain current measurement:*

Test	Ids [A]	V_{GO20} $(source)$ [V]	V_{GO22} $(drain)$ [V]	V_{AIN0} $(gate)$ [V]
Test B	1.82746884628187e-05	0	1	1
Test C	1.708984314063855e-07	1	0	1

There is a 100 fold increase in G022 drain current measurement for test B (compared to A and C). (which makes sense, in test A there is no Vds and in test C Vds is inverted)

Set trial gate AIN0 voltage 0 and see if the drain current is also zero. If not, why?

Drain current is ~3.5e-06 A when gate AIN0 voltage is set to zero. Probably due to leakage or noise.

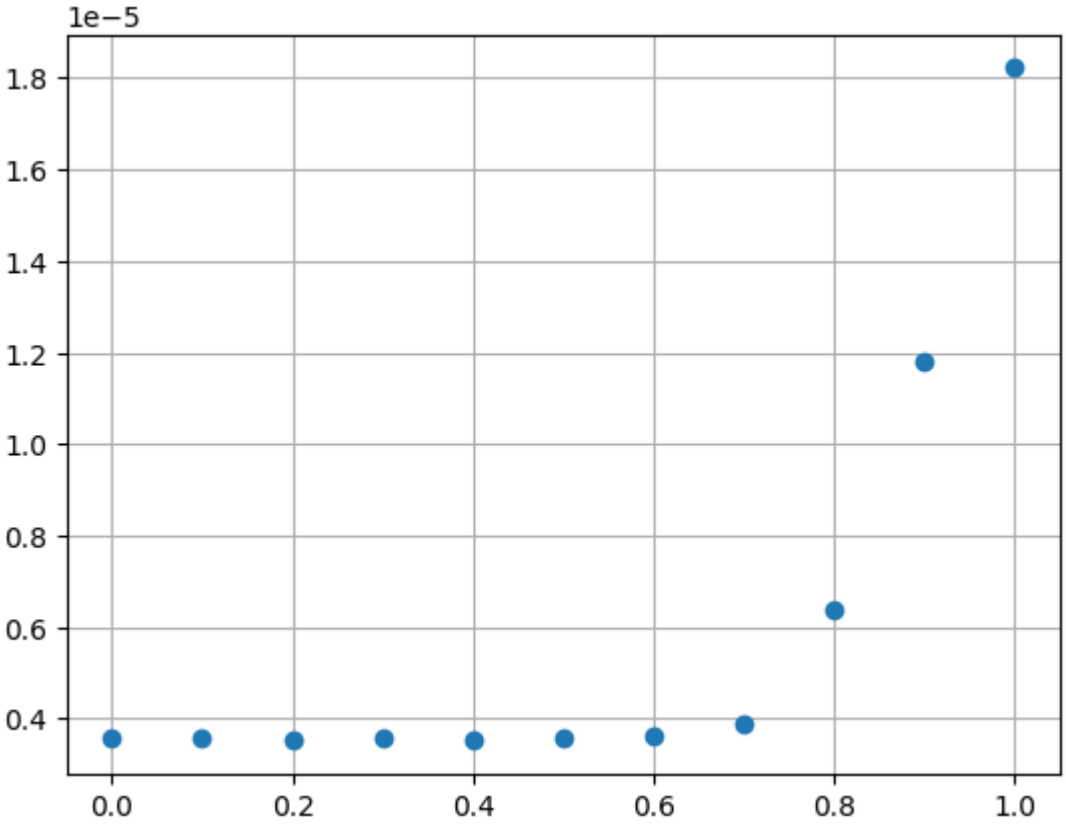
Now you can try some challenging experiments! Sweep gate voltage between 0~1V and see how the output current change

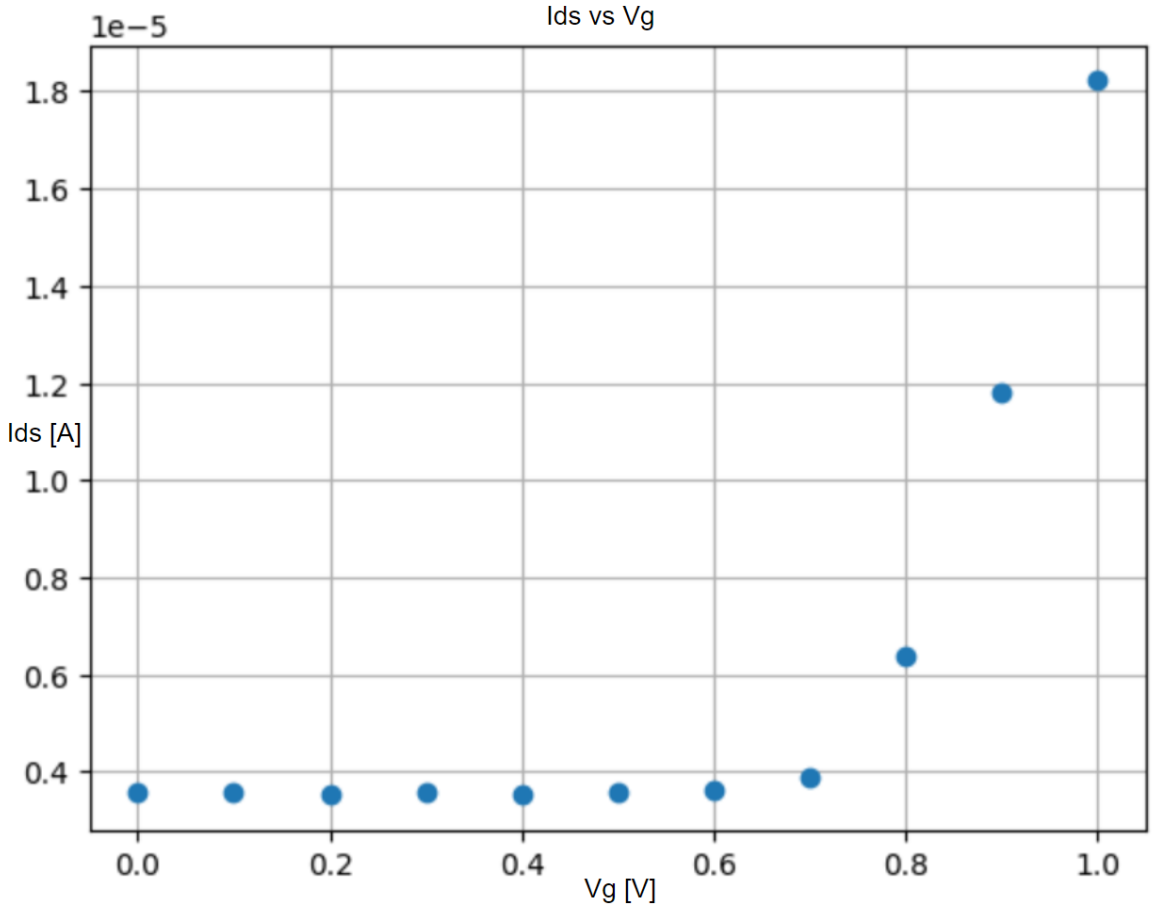
In [ ]:
```
import time
#Sweep gate voltage
drain_current = []
leakage = I_d
for n in range(0,11,1):
    if n*0.1 > 1.1:
        print('too high voltage')
        break

    p.set_voltage(pyplane.DacChannel.AIN0,n*0.1)
    time.sleep(0.5)
    drain_current.append(p.read_current(pyplane.AdcChannel.GO22)-leakage)

x = np.arange(0,11,1)*0.1

plt.plot(x,drain_current,'o')
plt.grid()
plt.show()
```





The following chunks were already implemented in the previous one

In [ ]:
```
#Initialize current variables
p.set_voltage(pyplane.DacChannel.GO22,1)  #set drain
p.set_voltage(pyplane.DacChannel.GO20,0)    #set source
```

Out[ ]:
```
0.0
```

In [ ]:
```
#Set Vg = 0 and wait 0.5 second for it to settle
time.sleep(0.5) # delay 0.5 second
```

In [ ]:
```
#Read leakage current Ids=Ids0
p.set_voltage(pyplane.DacChannel.GO20,1)
p.set_voltage(pyplane.DacChannel.GO22,1)
p.set_voltage(pyplane.DacChannel.AIN0,1)
I_d = p.read_current(pyplane.AdcChannel.GO22) #view in step 1
print(I_d)
```

```
3.588867230064352e-06
```

In [ ]:
```
#Read Ids at Vg sweep and wait for it to settle
for n in range(N_samples):
    p.set_voltage(pyplane.DacChannel.AIN0,Vg[n])
    time.sleep(0.05) # delay 0.1 second
    #Subtract leakage and shunt resistance from read current value
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [44], line 2
      1 #Read Ids at Vg sweep and wait for it to settle
----> 2 for n in range(N_samples):
      3     p.set_voltage(pyplane.DacChannel.AIN0,Vg[n])
      4     time.sleep(0.05)

NameError: name 'N_samples' is not defined
```

4. Clean up

Well done! That's all for today.

Remember you have to clean up in the end just as in a real lab!

- Close you device and release memory by doing

In [ ]:
```
del p
```

- Save your changes
- Download the files you need for the report to your own PC

5. Postlab Questions

1. Why is there no pin for the bulk of NFET? What is it's voltage then?

The bulk pin of the NFET is connected to Vss (the lowest voltage (or the ground)). It's voltage is 0V (without the noise).

1. How precise are the measurements of voltage and current using DAC?

According to the documentation the DAC is an 8-bit DAC and can only set voltages in steps of 13mv. The ADC is a 12-bit ADC, but only using effectively 10 of these bits so it's precision would be $\frac{10\mu A}{2^{15}} = 3.0517 \times 10^{-10} A$ *where* $10\mu A$ *is the maximum expected current and* 15 *is the number of bits in the register. This level of precision is useful when working in the subthreshold domain of MOS transistors as the relationship between* V_{gs} *and* I_d *is exponential and any "error" due to noise may be significant*

1. Do you think building a "computer" whose inputs and outputs are analog voltage/current signals is a good idea? Why or why not?

Yes, even though traditional computers operate with digital inputs and outputs, building a computer with analog I/Os would permit us to reduce energy consumption (for example by using MOS transistors in the subthreshold region) as well as mimic neural architectures present in the brain. However such computers would be specialized in doing only one task, but with better performance than a classical computer