Neuromorphic engineering I

# Lab 3: Subthreshold Behavior of Transistors

Group number: 18

Team member 1: Quillan Favey

Date: 04.11.2022

---

In this lab exercise we will be investigating the subthreshold (weak inversion) behavior of isolated *p*-- and *n*--channel MOSFETs. Specifically, we will

- measure the currents through the transistors as a function of their gate and source voltages
- determine how effective these terminals are at changing the current
- compare the characteristics of p and n-fet devices.

# 1. Prelab

Make sure you have studied the lecture material before attempting this prelab. The questions will also make much more sense if you read through the entire lab handout first. *You are required to complete this prelab before you can begin taking data.*

## 1.1 n- and p-fets, in an *n* well Process

A vertical section through the silicon with both n and p-fet transistors is shown in Figure 1. The class chip has a p-type substrate (like almost all chips nowadays) and both p- and n-wells.
The p-wells (not shown in the figure) are shorted to the p-substrate because the doping is of the same type.

Because we are grounding the substrate and we are connecting *n*--well to the power supply, $V_{dd}$ is positive. This positive voltage reverse biases the junction between the *n*--wells (which are tied to $V_{dd}$) and the substrate (which is tied to gnd).

For this process, $V_{dd}$=1.8 V.

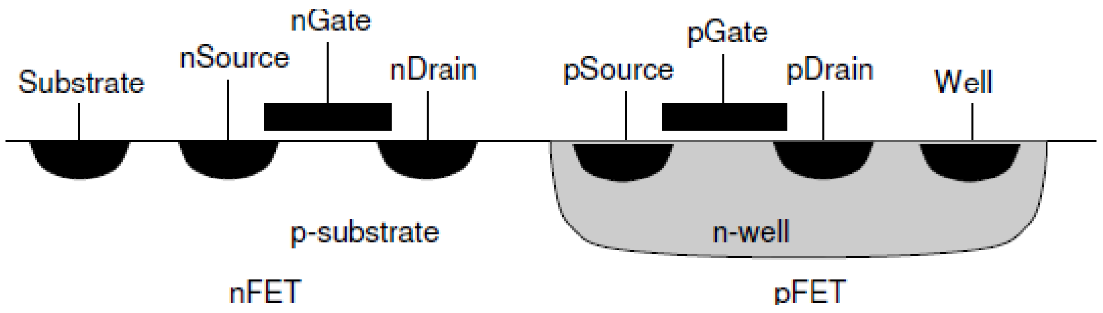
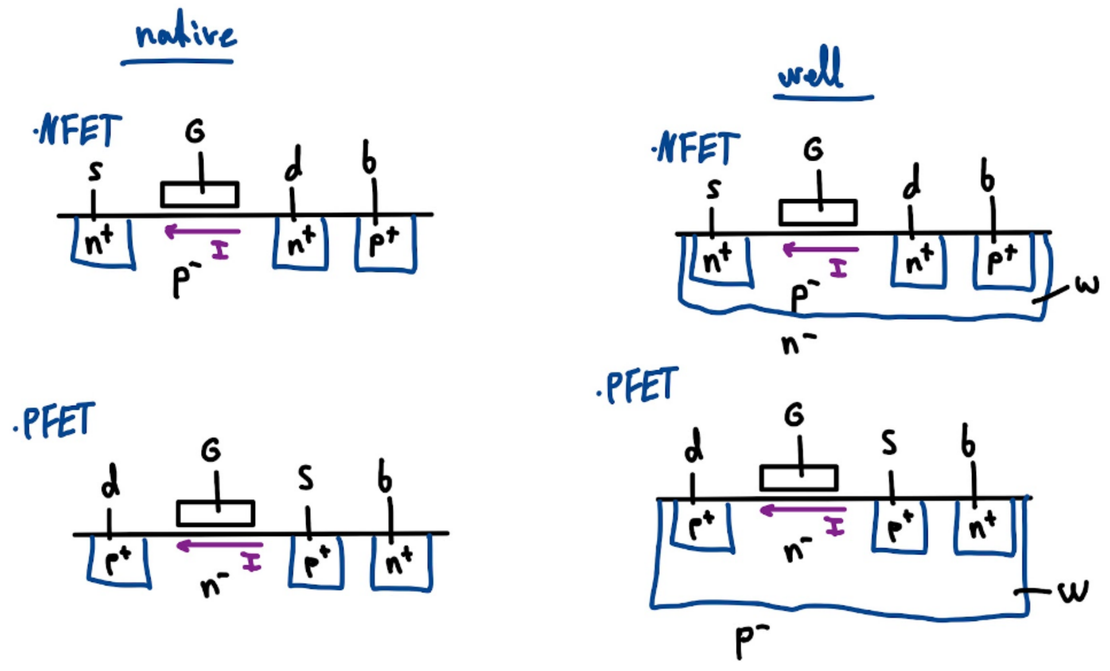


**Figure 1: a cross section through p-substrate chip.**

For the following questions assume an *n*--well process -- unless stated otherwise.

**1.** Draw four-terminal symbols for native and well transistors and label all the terminals; use *d* for drain, *s* for source, *g* for gate, *b* for bulk, and *w* for well. Indicate the direction of current flow that is consistent with your choice of drain and source (you can drag the * .png file into the cell below).



**2.** Write the expressions for the subthreshold (weak inversion) current $I_{ds}$ for both types of transistors.

- For n-fet: $I = I_0 e^{\kappa V_g/U_T}(e^{-V_s/U_T} - e^{-V_d/U_T})$
- For p-fet: $I = I_0 e^{-\kappa V_g/U_T}(e^{V_s/U_T} - e^{V_d/U_T})$

**3.** Write the expressions for the *saturation* current of these transistors, that is, the value of the current when $V_{ds} \gg \frac{4kT}{q}$. For the remaining questions you may assume that the transistor is in saturation.

- For n-fet: $I = I_f = I_0 e^{(\kappa V_g - V_s)/U_T}$
- For p-fet: $I = I_f = I_0 e^{(-\kappa V_g + V_s)/U_T}$

**4.** For both transistors, write an expression for source voltage as a function of gate voltage if the channel current is constant and the transistor is in saturation. In each case, what is $\frac{\mathrm{d}V_s}{\mathrm{d}V_g}$?

- For n-fet: $V_s = \kappa V_g - \ln(\frac{I}{I_0})U_T$
- For p-fet: $V_s = \kappa V_g + \ln(\frac{I}{I_0})U_T$

Taking the derivative for each expression with respect to $V_g$, we get: $\frac{dV_s}{dV_g} = \kappa$

## 1.2 ESD protection of CMOS Chips (need to know)

All MOSFET chips are *extremely* prone to damage by static electricity. The current through the transistors is controlled by an insulated gate.

Not so fun facts: **Even a few tens of volts can blow up the gate. A short walk across the room can build up kilovolts of static potential.**

There are electrostatic discharge (ESD) protection structures on the chip inputs that are designed to leak off the static charge before it can damage the chip, but often this will not be enough.

There are two simple precautions that can definitely keep the chip safe.

**1. When the chip is not powered up in a socket, keep it stuck into a piece of black conductive foam.** This will short all the pins together.

**2. Always ground yourself to chassis (potbox) ground before picking up or touching a chip.** This will discharge the static charge.

## 1.3 Experiments and Lab Reports

For the following experiments, include in your lab reports, graphs of all theoretical and experimental curves. Experimental data should be plotted in a point style so that individual data points are visible. Make sure you take enough data points and label your axes. The theoretical fit should be graphed on the same plot in a line style.

Your written interpretation of the results and any anomalies are essential. Please do not just hand in the plots without any interpretation on your part. Your report does not need to be beautiful, but it should show that you understand what you are measuring.

Remember that the purpose of this lab is to investigate *subthreshold* transistor characteristics. Therefore, all voltage sweeps should span the measurable subthreshold regime while extending just far enough above threshold to show where the threshold is.

Avoid these common mistakes in your report:

- **Not discussing your data sufficiently.** Think about a publication. The readers want to understand your reasoning with you. They want to be able to reproduce your results.
- **Not using cross-hair axes when 0,0 is relevant.** Use grid on to turn on grid, which will draw dotted lines from major ticks. See fontsize to make spacing readable.
- **Forgetting to mention what your plot shows.**
- **Not labeling your figures with a caption,** e.g., "Fig. 1: Transistor drain current vs. gate voltage, Experiment 1."
- **Insufficiently annotating your data.** It's OK to draw on your plots to indicate the slope of the curve, or the x / y intercepts.
- **Using identical markers for all plots.** Your curves must be distinguishable when printed in black and white. Use e.g. plot(v,i,'o-',v,i2,'s-'), which labels one curve with circle markers and the other with square markers.
- **Forgetting units on measurements,** e.g. "our conductance is 1.000653e-10". What are the units?
- **Giving your measurements too many digits of precision;** see previous error. Do your instruments really give you 7 digits of precision?

# 2 Set up the experiment

## 2.1 Connect the device

In [ ]:
```
# import the necessary library to communicate with the hardware
import pyplane
import time
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') # Open the USB device ttyACM0 (the board).
    except RuntimeError as e:
        print(e)

# Note that if you plug out and plug in the USB device in a short time interval, the c
# then you may get error messages with open(...ttyACM0). So please avoid frenquently p
```

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

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

In [ ]:
```
# Send a reset signal to the board, check if the LED blinks
p.reset(pyplane.ResetType.Soft)

time.sleep(1)
# NOTE: You must send this request events every time you do a reset operetion, otherwi
# Because the class chip need to do handshake to get the communication correct.
p.request_events(1)
```

In [ ]:
```
# Try to read something, make sure the chip responses
p.read_current(pyplane.AdcChannel.GO0_N)
```

Out[ ]:
```
1.4501952705359145e-07
```

In [ ]:
```
# If any of the above steps fail, delete the object, and restart the kernel

# del p
```

## 2.2 Recommendations to this lab

- You do not need to follow the order, it is actually better to do all the measurement of one device together, e.g. 3.1 -> 4.1 -> 3.2 -> 4.2
- **Please save the data as frequent as possible and use the loaded data for processing**

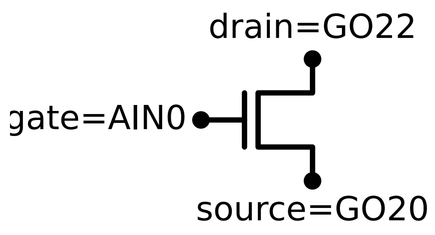
# 3 Current as a Function of Gate Voltage

## 3.1 N-FET

For the N-FET device on the CoACH chip, measure current $I_{ds}$ as a function of gate voltage $V_g$ for fixed source, bulk (substrate or well), and drain voltages.

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[ ]:



Hint: To cancel out the leakage current and shunt resistance (recall lab1), you may want to do a subtraction

$$I_{ds} = I_{GO20} - I_{GO20}|_{V_g=0}$$

- You have to set the input voltage demultiplexer by sending a 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 and you must replug it.**

- What will be the fixed value for source, bulk (substrate or well), and drain voltages?

In [ ]:
```
# set source voltage
vs_n = 0
p.set_voltage(pyplane.DacChannel.GO20,vs_n)
print("The source voltage is set to {} V".format(vs_n)) #better use p.get_set_voltage
```

```
The source voltage is set to 0 V
```

In [ ]:
```
# set drain voltage
vd_n = 1.8
p.set_voltage(pyplane.DacChannel.GO22, vd_n)
print("The drain voltage is set to {} V".format(vd_n))
```

```
The drain voltage is set to 1.8 V
```

In [ ]:
```
#get leakage current
vg_leakage = 0
p.set_voltage(pyplane.DacChannel.AIN0, vg_leakage)
print("The trial gate voltage is set to {} V".format(vg_leakage))
leakage = p.read_current(pyplane.AdcChannel.GO20_N)
print("Leakage current is {} A".format(leakage))
```

```
The trial gate voltage is set to 0 V
Leakage current is 2.4414064103694955e-09 A
```

In [ ]:
```
# set trial gate voltage
vg_n = 1.8
p.set_voltage(pyplane.DacChannel.AIN0, vg_n)
print("The trial gate voltage is set to {} V".format(vg_n))
```

```
The trial gate voltage is set to 1.8 V
```

In [ ]:
```
# read Ids, from *Source* --> NOTE THAT THE ADC CHANNEL PIN CHANGES THE NAME FOR THE S
ids_n = p.read_current(pyplane.AdcChannel.GO20_N)
print("Ids is {} A".format(ids_n))
```

```
Ids is 9.98779228211788e-07 A
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
import numpy as np
Ids_sweep = []
V_sweep = np.arange(0,1.8,0.01)
for voltage in V_sweep:
    #set voltage
    p.set_voltage(pyplane.DacChannel.AIN0, voltage)
    #print("The gate voltage is set to {} V".format(voltage))
    time.sleep(0.05)
    Ids_sweep.append(p.read_current(pyplane.AdcChannel.GO20_N)-leakage)
```

In [ ]:
```
# plot in linear scale
plt.rcParams.update({'font.size': 14})
plt.plot(V_sweep,Ids_sweep,"o")
plt.xlabel('''Vgs [V]

        Fig.1.1: Ids as a function of Vgs in an N-FET transistor            ''')
plt.ylabel('Ids [A]')
plt.title("Ids vs Vgs")
#plt.legend()
plt.grid()
plt.show()
```

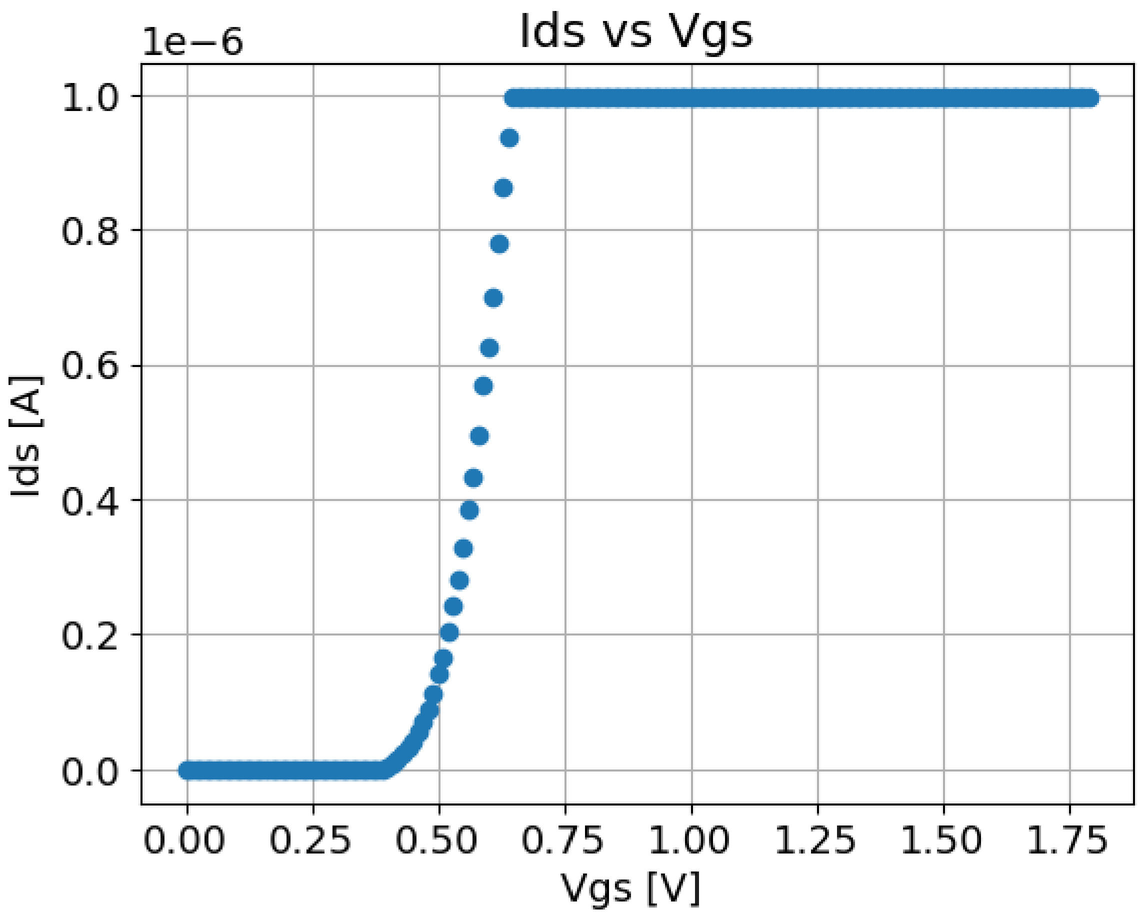


Fig.1.1: Ids as a function of Vgs in an N-FET transistor

```
In [ ]: # if it looks nice in the plot, save it!

        #plt.savefig("TP_2_plot_3_1")
        data = [Ids_sweep,V_sweep]
        np.savetxt('TP_2_data_3_1.csv',data, delimiter = ',')
```

```
In [ ]: # Load data you saved and plot, to check if the data is saved correctly
        y,x = np.loadtxt('TP_2_data_3_1.csv',delimiter=',')
```

```
In [ ]: # plot in logarithmic scale
        plt.rcParams.update({'font.size': 14})
        plt.semilogy(x[10:80],y[10:80],"o-",label="$I_{ds}$")
        plt.xlabel('''$V_{gs}$ [V]

                Fig.1.2: $I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturati
        plt.ylabel('$I_{ds}$ [A]')
        plt.title("log plot of $I_{ds}$ vs $V_{gs}$ for nFET")
        plt.grid()
        plt.legend()
        plt.show()
```

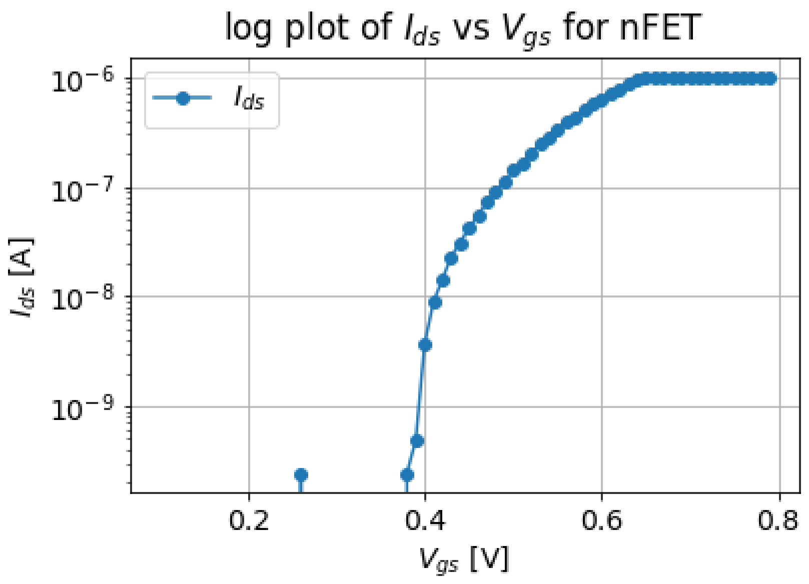


Fig.1.2: $I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation

```
In [ ]: # extract the valid range and plot in logarithmic scale
        valid_y = y[47:60]
        valid_x = x[47:60]
        plt.plot(valid_x,valid_y,"o-")
        plt.xlabel('''$V_{gs}$ [V]

                Fig.1.3: Exponential range of
                $I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation
        plt.ylabel('$I_{ds}$ [A]')
        plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
        plt.semilogy()
        plt.grid()
        plt.show()
```

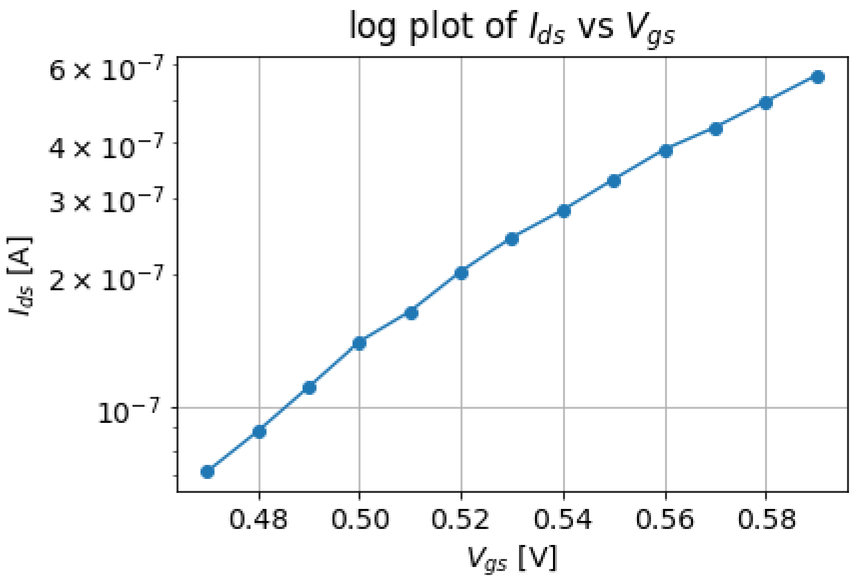


Fig.1.3: Exponential range of
$I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation

```
In [ ]: # fit in the valid range (you may want to add the fitted line in the plot)
        # compare the orignial data points and the fitted line
        from scipy.optimize import curve_fit

        def Ids_nFET_func(Vgs,k,I0):
                Ut = 0.025
                return I0 * np.exp(k*Vgs/Ut)

        popt, pcov = curve_fit(Ids_nFET_func,valid_x,valid_y,p0=[0.48, 1e-7])
        fit_k,fit_I0 = popt
        fitted_y = Ids_nFET_func(valid_x,fit_k,fit_I0)


        plt.plot(valid_x,valid_y,"o-",label='original data')
        plt.plot(valid_x,fitted_y,"x--",label="Fitted $I_{ds}$")
        plt.xlabel('''$V_{gs}$ [V]

                Fig.1.4: Exponential range of
                $I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation
                 fitted using $I=I_{0}e^{(\kappa V_{g}-V_{s})/U_{T}}$
                          ''')
        plt.ylabel('$I_{ds}$ [A]')
        plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
        plt.legend()
        plt.semilogy()
        plt.grid()
        plt.show()
```

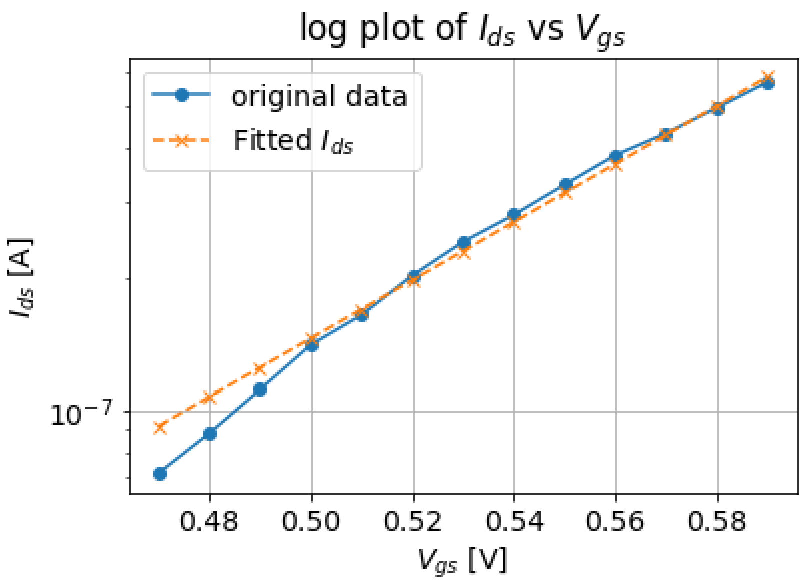


Fig.1.4: Exponential range of
$I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation
fitted using $I = I_0 e^{(\kappa V_g - V_s)/U_T}$

- Extract $I_0$ and $\kappa$

In [ ]:
```
# I_0
print(f"I_0 = {fit_I0}")
```

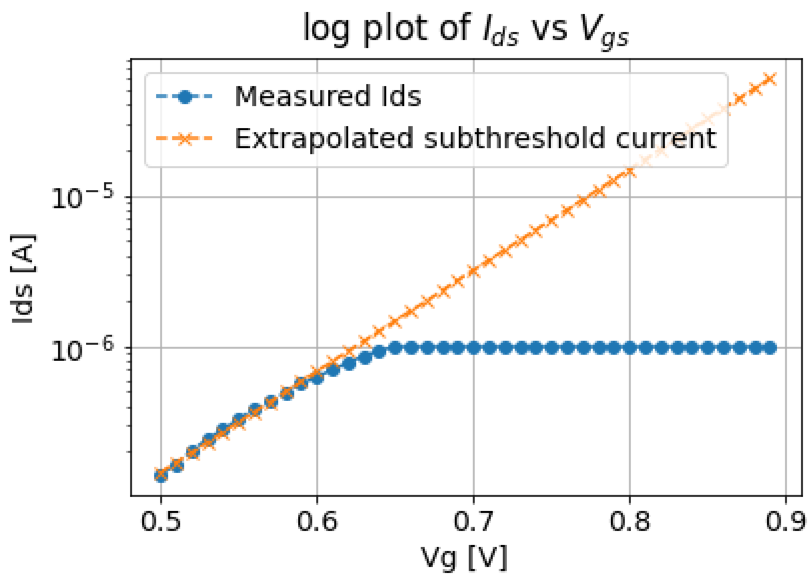
```
I_0 = 6.558069379435462e-11
```

In [ ]:
```
# kappa
print(f"kappa = {fit_k}")
```

```
kappa = 0.3853408682973667
```

- Extract the threshold voltage and the current at threshold, using the definition given in class: $I_{ds}$ is half of the extrapolated subthreshold current.

In [ ]:
```
# compute threshold voltage
from scipy import interpolate

#we start by extrapolating our fitted data (from the equation)
f = interpolate.interp1d(valid_x, np.log(fitted_y),fill_value='extrapolate')
xnew = np.arange(0,1.8,0.01)
ynew = f(xnew)

#we can the plot the simulated data and the measured data
plt.semilogy(x[50:90],y[50:90],"o--",label="Measured Ids")
plt.semilogy(xnew[50:90],np.exp(ynew[50:90]),"x--",label="Extrapolated subthreshold cu
plt.legend()
plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
plt.xlabel("Vg [V]")
plt.ylabel("Ids [A]")
plt.grid()
```

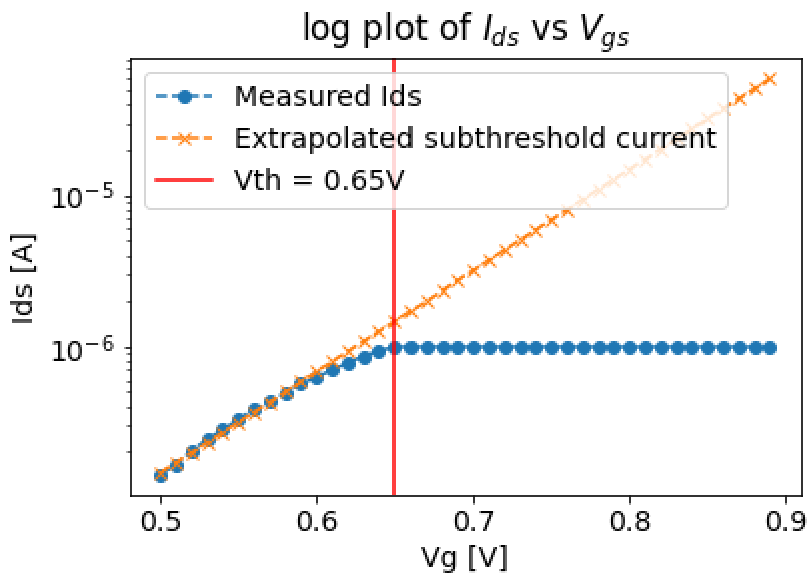


```
In [ ]: # compute Ids at threshold voltage

        #we start by finding the point (Ids) where Ids is half of the extrapolated subthreshol
        threshold_I = None
        for i,ext in zip(y[60:90],np.exp(ynew[60:90])):
            if i <= ext*0.5:
                threshold_I = i

        print("Measured current at threshold: ",threshold_I,"A")
        #next we find the Vds for the threshold Ids
        index = None
        for i in y[60:90]:
            if i <= threshold_I:
               index = np.where(y == i)
        print("Threshold voltage: ", xnew[index[0][0]],"V")
```

```
Measured current at threshold:  9.963378218014185e-07 A
Threshold voltage:  0.65 V
```

```
In [ ]: #we can the plot the simulated data and the measured data
        plt.semilogy(x[50:90],y[50:90],"o--",label="Measured Ids")
        plt.semilogy(xnew[50:90],np.exp(ynew[50:90]),"x--",label="Extrapolated subthreshold cu
        plt.axvline(x=xnew[index[0][0]], ymin=0, ymax=1,label=f'Vth = {xnew[index[0][0]]}V',co
        plt.legend()
        plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
        plt.xlabel("Vg [V]")
        plt.ylabel("Ids [A]")
```

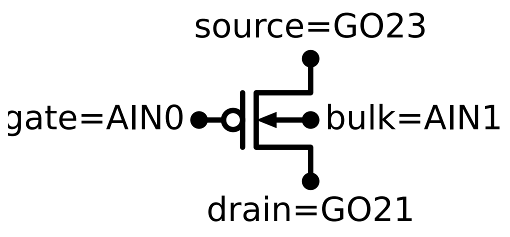
```
Out[ ]: Text(0, 0.5, 'Ids [A]')
```



## 3.2 P-FET

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.PFet, reverse=True, bulk=True)
d.add(elm.Dot, xy=Q.gate, lftlabel='gate=AIN0')
d.add(elm.Dot, xy=Q.bulk, rgtlabel='bulk=AIN1')
d.add(elm.Dot, xy=Q.drain, botlabel='drain=GO21')
d.add(elm.Dot, xy=Q.source, toplabel='source=GO23')
d.draw()
```

Out[ ]:



Hint: To cancel out the leakage current and shunt resistance, you may want to do a subtraction:

$$I_{ds} = I_{GO21} - I_{GO21}|_{V_g=0}$$

- You have to choose the input voltage demultiplexer by sending a configuration event (make sure LED1 blinks):

In [ ]:
```
events = [pyplane.Coach.generate_aerc_event( \
    pyplane.Coach.CurrentOutputSelect.SelectLine5, \
    pyplane.Coach.VoltageOutputSelect.NoneSelected, \
    pyplane.Coach.VoltageInputSelect.SelectLine1, \
    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 and you must replug it.**

- What will be the fixed source, bulk (substrate or well), and drain voltages?

In [ ]:
```
# set bulk voltage (to Vdd for p-fets)
vb_p = 1.8
p.set_voltage(pyplane.DacChannel.AIN1, vb_p)
print("The bulk voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.AI
```

```
The bulk voltage is set to 1.7982406616210938 V
```

In [ ]:
```
# set source voltage
vs_p = 1.8
p.set_voltage(pyplane.DacChannel.GO23, vs_p)
print("The source voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.
```

```
The source voltage is set to 1.7982406616210938 V
```

In [ ]:
```
# set drain voltage
vd_p = 0
p.set_voltage(pyplane.DacChannel.GO21, vd_p)
print("The drain voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.G
```

```
The drain voltage is set to 0.0 V
```

In [ ]:
```
# set trial gate voltage
vg_p = 1.8
p.set_voltage(pyplane.DacChannel.AIN0, vg_p)
print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.AI
```

```
The gate voltage is set to 1.7982406616210938 V
```

In [ ]:
```
# read Ids
ids_p = p.read_current(pyplane.AdcChannel.GO21_N)
print("Ids is {} A".format(ids_p))
```

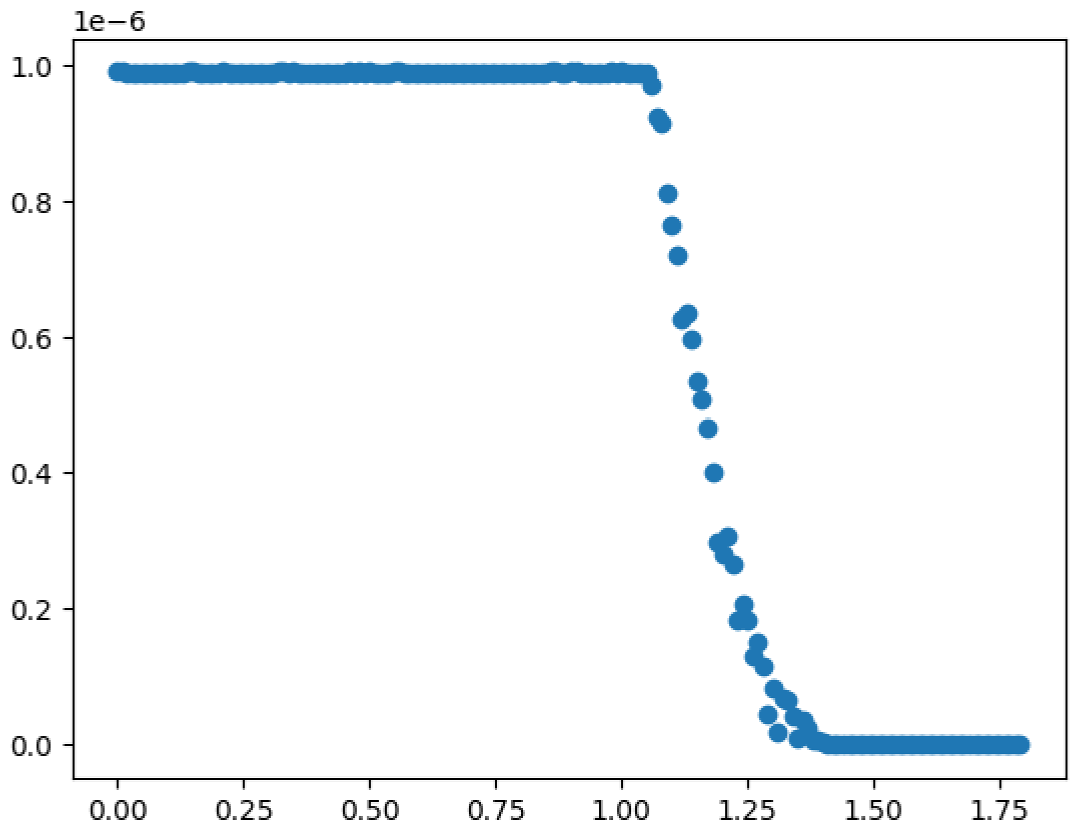
```
Ids is 8.544922103226327e-09 A
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
import numpy as np
Ids_sweep_p = []
V_sweep_p = np.arange(0,1.8,0.01)
for voltage in V_sweep_p:
    #set voltage
    p.set_voltage(pyplane.DacChannel.AIN0, voltage)
    #print("The gate voltage is set to {} V".format(voltage))
    time.sleep(0.05)
    Ids_sweep_p.append(p.read_current(pyplane.AdcChannel.GO21_N)-ids_p)
```

In [ ]:
```
# plot in linear scale
plt.plot(V_sweep_p,Ids_sweep_p,"o")
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7fa1a527b6d0>]
```



In [ ]:
```
# if it looks nice in the plot, save it!
data = [Ids_sweep_p,V_sweep_p]
np.savetxt("pfet31.csv",data,delimiter=",")
```

In [ ]:
```
# Load data you saved and plot, to check if the data is saved correctly
y,x = np.loadtxt('pfet31.csv',delimiter=',')
```

In [ ]:
```
# plot in logarithmic scale
plt.rcParams.update({'font.size': 14})
plt.semilogy(x,y,"o-",label="$I_{ds}$")
plt.xlabel('''$V_{gs}$ [V]

        Fig.2.2: $I_{ds}$ as a function of $V_{gs}$ in a pFet transistor in saturatior
plt.ylabel('$I_{ds}$ [A]')
plt.title("log plot of $I_{ds}$ vs $V_{gs}$ in a pFET")
plt.grid()
plt.legend()
plt.show()
```

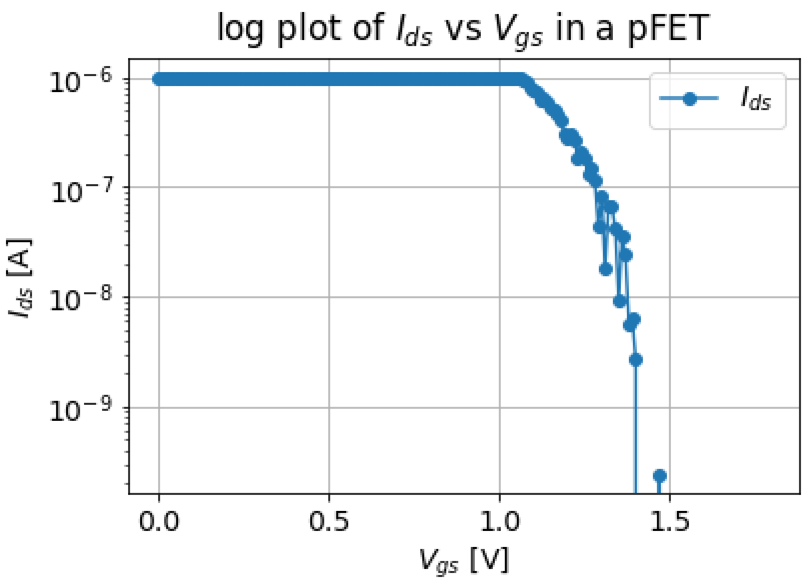


Fig.2.2: $I_{ds}$ as a function of $V_{gs}$ in a pFet transistor in saturation region

In [ ]:
```
# extract the valid range and plot in logarithmic scale
valid_y = y[115:130]
valid_x = x[115:130]
plt.plot(valid_x,valid_y,"o-")
plt.xlabel('''$V_{gs}$ [V]

        Fig.2.3: Exponential range of
        $I_{ds}$ as a function of $V_{gs}$ in an P-FET transistor in saturation
plt.ylabel('$I_{ds}$ [A]')
plt.title("log plot of $I_{ds}$ vs $V_{gs}$ in a p-fet")
plt.semilogy()
plt.grid()
plt.show()
```

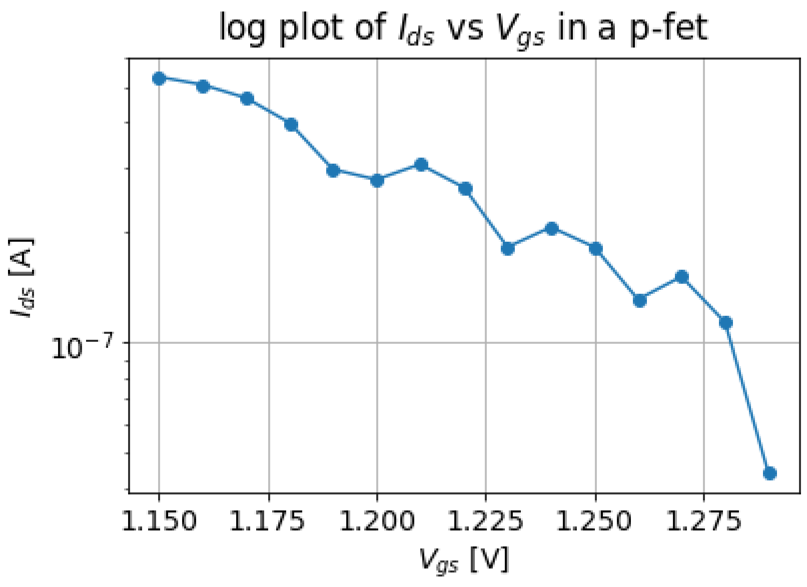


Fig.2.3: Exponential range of
$I_{ds}$ as a function of $V_{gs}$ in an P-FET transistor in saturation

In [ ]:
```
# fit in the valid range (you may want to add the fitted line in the plot)
```

```
def Ids_nFET_func(Vgs,k,I0):
        Ut = 0.025
        return I0 * np.exp((-k*Vgs+1.8)/Ut) #for pfet Vg+Vs
#next we can estimate our parameters and get our fitted y values
popt, pcov = curve_fit(Ids_nFET_func,valid_x,valid_y,p0=[0.78, 1e-10])
fit_k,fit_I0 = popt
fitted_y = Ids_nFET_func(valid_x,fit_k,fit_I0)


plt.plot(valid_x,valid_y,"o-",label='original data')
plt.plot(valid_x,fitted_y,"x--",label="Fitted $I_{ds}$")
plt.xlabel('''$V_{gs}$ [V]

        Fig.2.4: Exponential range of
        $I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation
         fitted using $I=I_{0}e^{(-\kappa V_{g}+V_{s})/U_{T}}$
                  ''')
plt.ylabel('$I_{ds}$ [A]')
plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
plt.legend()
plt.semilogy()
plt.grid()
plt.show()
```

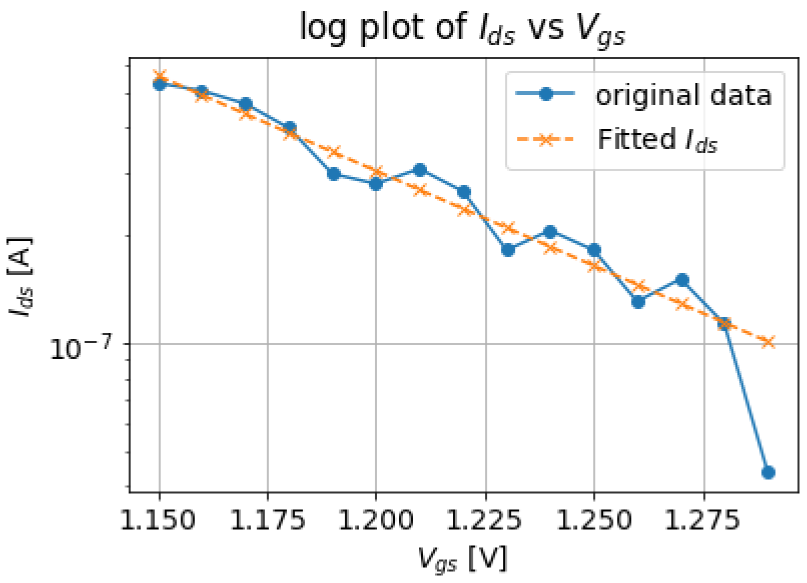


Fig.2.4: Exponential range of
$I_{ds}$ as a function of $V_{gs}$ in an N-FET transistor in saturation
fitted using $I = I_0 e^{(-\kappa V_g + V_s)/U_T}$

Extract $I_0$ and $\kappa$

In [ ]:
```
# I_0
print(f"I_0 = {fit_I0}")
```

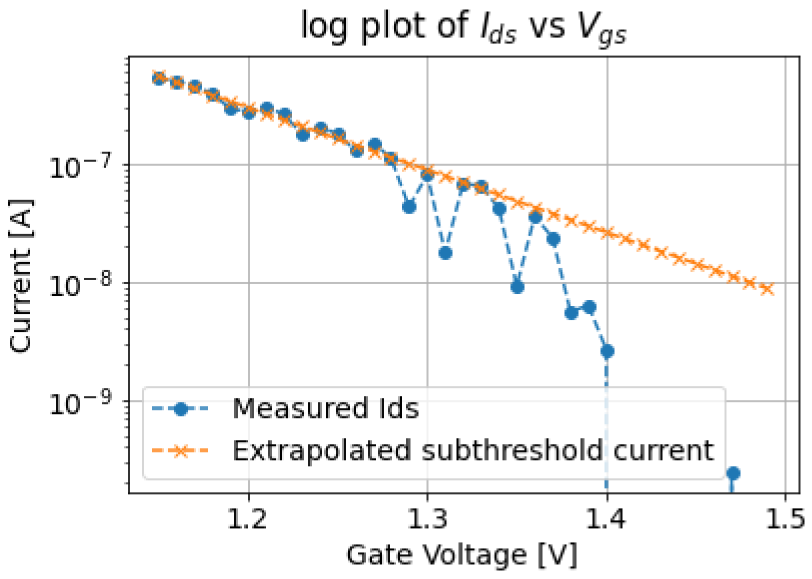
```
I_0 = 3.730100621752887e-32
```

In [ ]:
```
# kappa
print(f"kappa = {fit_k}")
```

```
kappa = 0.3050211127013173
```

Extract the threshold voltage and the current at threshold, using the definition given in class: $I_{ds}$ is half of the extrapolated subthreshold current.

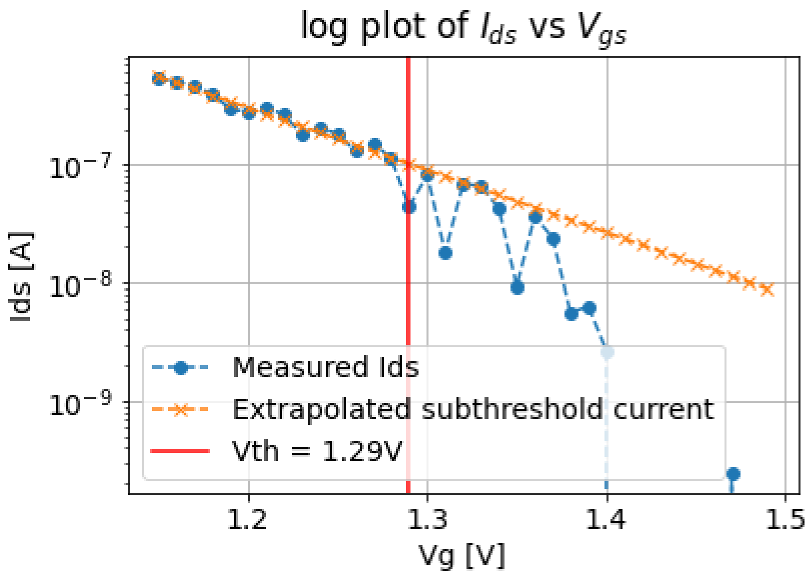
In [ ]:
```
# compute threshold voltage
#we start by extrapolating our fitted data (from the equation)
f = interpolate.interp1d(valid_x, np.log(fitted_y),fill_value='extrapolate')
xnew = np.arange(0,1.8,0.01)
ynew = f(xnew)

#we can the plot the simulated data and the measured data
plt.semilogy(x[115:150],y[115:150],"o--",label="Measured Ids")
plt.semilogy(xnew[115:150],np.exp(ynew[115:150]),"x--",label="Extrapolated subthreshol
plt.legend()
plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
plt.xlabel("Gate Voltage [V]")
plt.ylabel("Current [A]")
plt.grid()

#Threshold voltage in next cell
```



In [ ]:
```
# compute Ids at threshold voltage
#we start by finding the point (Ids) where Ids is half of the extrapolated subthreshol
threshold_I = None
for i,ext in zip(y[115:150],np.exp(ynew[115:150])):
    if i <= ext*0.5:
        threshold_I = i
        break

print("Measured current at threshold: ",threshold_I,"A")
#next we find the Vds for the threshold Ids
index = np.where(y[115:150] <= threshold_I)
Vth =  x[115+index[0][0]]       # we add 115 (the start of where we evaluated our list
print("Threshold voltage: ",Vth,"V")
```

```
Measured current at threshold:  4.394531405438329e-08 A
Threshold voltage:  1.29 V
```

In [ ]:
```
plt.semilogy(x[115:150],y[115:150],"o--",label="Measured Ids")
plt.semilogy(xnew[115:150],np.exp(ynew[115:150]),"x--",label="Extrapolated subthreshol
plt.axvline(x=Vth, ymin=0, ymax=1,label=f'Vth = {Vth}V',color="red")
plt.legend()
plt.title("log plot of $I_{ds}$ vs $V_{gs}$")
plt.xlabel("Vg [V]")
plt.ylabel("Ids [A]")
plt.grid()
```



*There seems to be a lot of "noise" in the measurements. This is probably due to the fact that a too high sample rate and not enough time between measurements were used.*

# 4 Back Gate Effect

In this experiment, we will characterize the relationship between the gate and source voltages for both the N-FET and the P-FET devices when the channel current is held constant. This experiment shows convincingly the relative effectiveness of each terminal and provides a direct measurement of $\kappa$.

Hint: Because we cannot read the voltage of the GO pins, it is not possible to simply keep $I_{ds}$ fixed and read $V_s$. In order to do so, we have to use some searching algorithm (e.g. binary search) to find the corresponding $V_s$.

## 4.1 N-FET

- If you are not coming from 3.1 directly, you have to set the input voltage demultiplexer by sending a configuration event (make sure LED1 blinks):

In [ ]:
```
#came from 3.1
```

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

- set fixed voltages

```
In [ ]: # set drain voltage
        # set drain voltage
        vd_n = 1.8
        p.set_voltage(pyplane.DacChannel.GO22, vd_n)
        print("The drain voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.G
```

```
The drain voltage is set to 1.7982406616210938 V
```

```
In [ ]: # set trial gate
        vg_n = 0
        p.set_voltage(pyplane.DacChannel.AIN0, vg_n)
        print("The trial gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChan
        # set source voltage
        vs_n = 0
        p.set_voltage(pyplane.DacChannel.GO20,vs_n)
        print("The source voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.
```

```
The trial gate voltage is set to 0.0 V
The source voltage is set to 0.0 V
```

```
In [ ]: # read trial Ids


        # read Ids, from *Source* --> NOTE THAT THE ADC CHANNEL PIN CHANGES THE NAME FOR THE S
        ids_n = p.read_current(pyplane.AdcChannel.GO20_N)
        print("trial Ids is {} A".format(ids_n))
```

```
trial Ids is 2.4414064103694955e-09 A
```

- Data aquisition

```
In [ ]: # define constants
        Vdd = 1.8
        max_iter = 10
        N_samples = 30
```

What Ids target should you set? And what is the corresponding Vg? Hint: Refer to 3.1

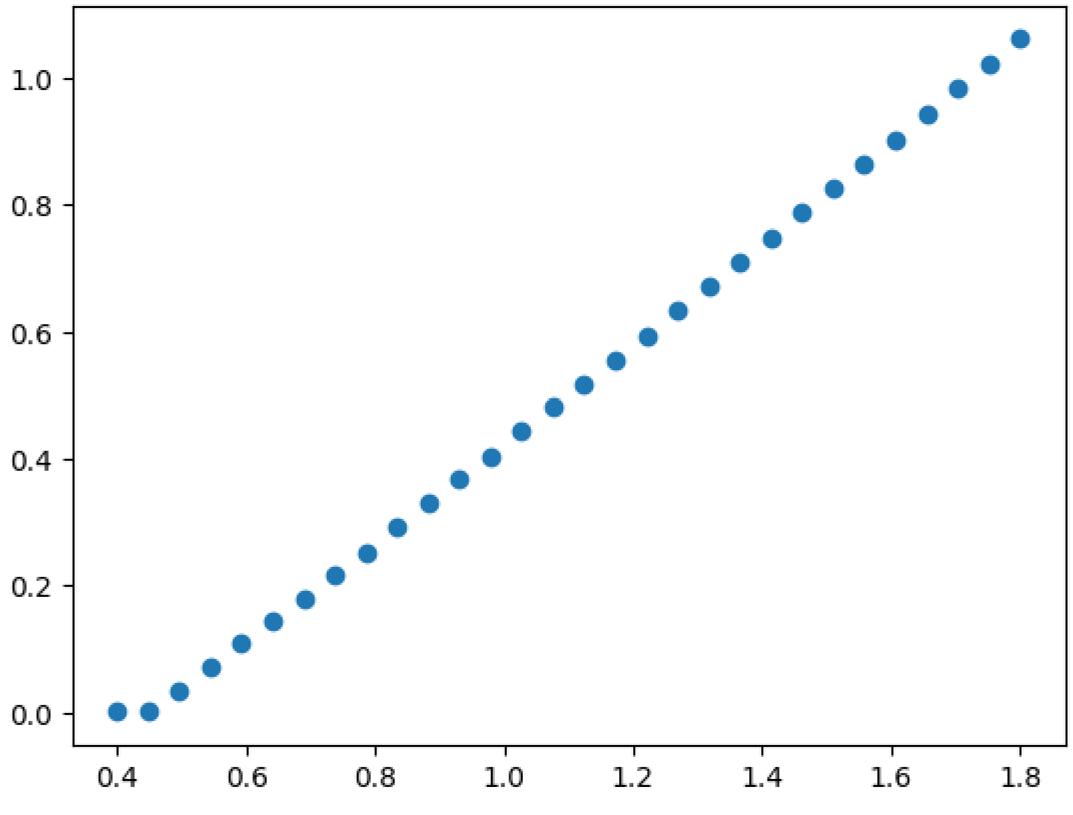
```
In [ ]: Ids_target = 5e-8
```

```
In [ ]: # initialize variables

        import numpy as np
        import time

        Vg = np.linspace(0.4,Vdd,N_samples)
        Vs = np.ones(N_samples) * Vdd/2
        #print(Vg)
```

```
In [ ]: # sweep Vg
        for n in range(N_samples):

            Vstep = Vdd/4

            # set Vg
```

```
    p.set_voltage(pyplane.DacChannel.AIN0,Vg[n])

    # search for Vs that gives Ids_target
    for j in range(max_iter):

        # set Vs
        p.set_voltage(pyplane.DacChannel.GO20,Vs[n])

        # wait to settle
        time.sleep(0.1)

        # read Ids and compute its difference with the target
        dI = p.read_current(pyplane.AdcChannel.GO20_N) - Ids_target

        # check for convergence
        if np.abs(dI) < Ids_target * 0.05:
            break

        # update Vs and step
        Vs[n] = Vs[n] + Vstep * np.sign(dI)
        Vstep = Vstep/2
```

In [ ]:
```
# plot
plt.plot(Vg,Vs,"o")
data = np.array([Vg,Vs])
np.savetxt("part4nfet.csv",data,delimiter=",")
```



In [ ]:
```
# if it looks nice in the plot, save it!
Vg, Vs = np.loadtxt("part4nfet.csv", delimiter=",")

plt.plot(Vg,Vs,"o-",label="Vs")
plt.legend()
plt.title("Vs vs Vg ")
plt.xlabel('''Vg [V]
Fig.3.1 Vs vs Vg for $I_{ds}$ = 5e-08 in an Nfet''')
plt.ylabel("Vs [V]")
plt.grid()
```

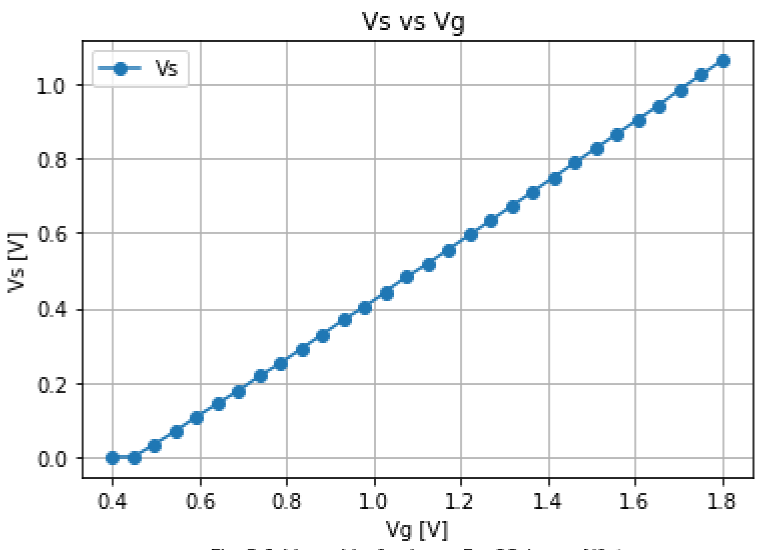


Fig.3.1 Vs vs Vg for $I_{ds}$ = 5e-08 in an Nfet

- How do you compute $\kappa$? Does it stay constant for different $V_g$?

*We can compute $\kappa$ by computing the slope of Vs plotted against Vg at constant Ids*

In [ ]:
```
# calculate kappa

kappa = [(Vs[i]-Vs[i-7])/(Vg[i]-Vg[i-7]) for i in range(7,30,4)]
print(r'Values of kappa at different points of the Vs vs Vg curve: ',kappa)
```

```
Values of kappa at different points of the Vs vs Vg curve:  [0.6424087213010204, 0.76
98501275510206, 0.780253507653061, 0.790656887755102, 0.8010602678571433, 0.811463647
9591832]
```

*We can see that $\kappa$ remains relatively consistent throughout the measurements*

## 4.2 P-FET

- If you are not coming from 3.2 directly, you have to set the input voltage demultiplexer by sending a configuration event (make sure LED1 blinks):

In [ ]:
```

```

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

- set fixed voltages

In [ ]:
```
# set bulk voltage
```

```
p.set_voltage(pyplane.DacChannel.AIN1,1.8)
```

Out[ ]:
```
1.7982406616210938
```

In [ ]:
```
# set drain voltage
# set drain voltage
vd_n = 0
p.set_voltage(pyplane.DacChannel.GO22, vd_n)
```

Out[ ]:
```
0.0
```

In [ ]:
```
# set trial gate and source voltages
#source
p.set_voltage(pyplane.DacChannel.GO23,1.8)
#gate
p.set_voltage(pyplane.DacChannel.AIN0,1.8)
```

Out[ ]:
```
1.7982406616210938
```

In [ ]:
```
# read trial Ids
Ids_trial = p.read_current(pyplane.AdcChannel.GO21_N)
print(Ids_trial)
```

```
7.0800778573243406e-09
```

- Data aquisition

In [ ]:
```
# define constants
Vdd = 1.8
max_iter = 10
N_samples = 30
```

What Ids target should you set? And what is the corresponding Vg? Hint: Refer to 3.2

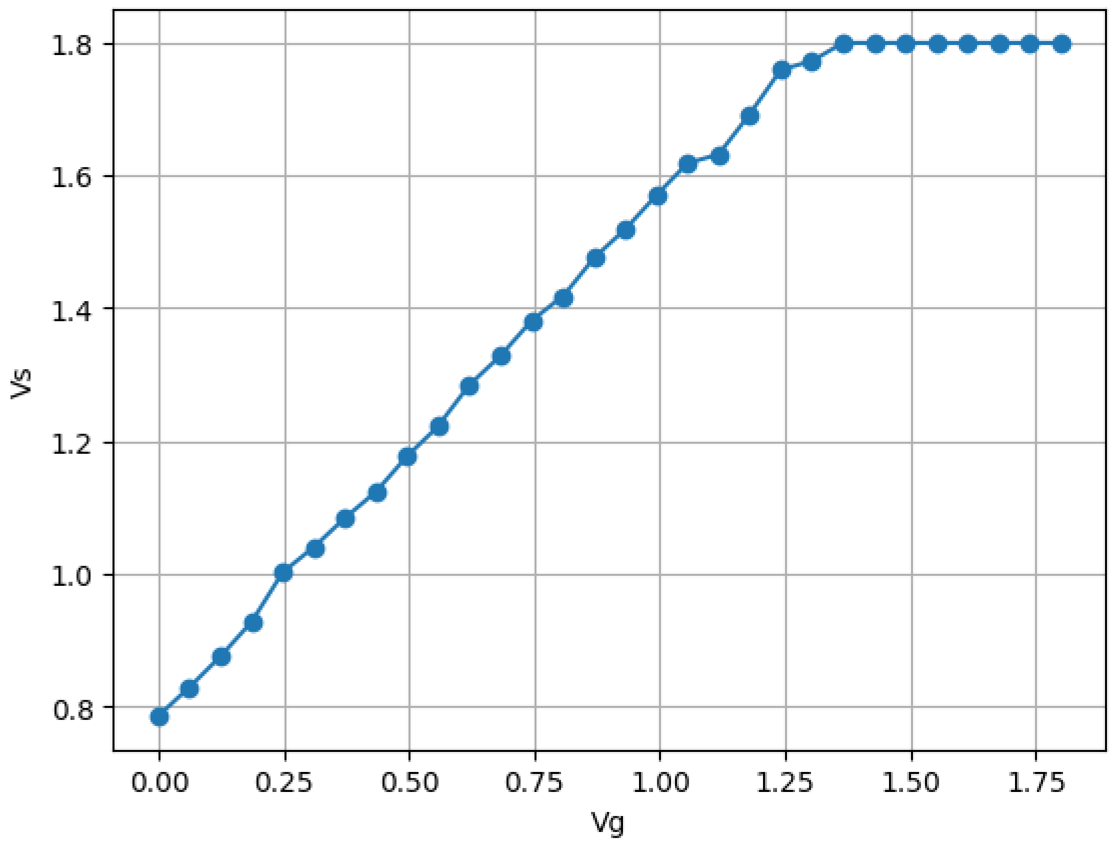
In [ ]:
```
Ids_target = 6e-08
#Vg_target =
```

In [ ]:
```
# initialize variables

import numpy as np
import time

Vg = np.linspace(0,Vdd,N_samples)
Vs = np.ones(N_samples) * Vdd/2
```

In [ ]:
```
# sweep Vg
for n in range(N_samples):

    Vstep = Vdd/4

    # set Vg
    p.set_voltage(pyplane.DacChannel.AIN0,Vg[n])

    # search for Vs that gives Ids_target
    for j in range(max_iter):
        # set Vs
        p.set_voltage(pyplane.DacChannel.GO23,Vs[n])

        # wait to settle
        time.sleep(0.1)

        # read Ids and compute its difference with the target
        dI = p.read_current(pyplane.AdcChannel.GO21_N) - Ids_target

        # check for convergence
        if np.abs(dI) < Ids_target * 0.05:
            break

        # update Vs and step
        Vs[n] = Vs[n] - Vstep * np.sign(dI)
        Vstep = Vstep/2
```

In [ ]:

```
# plot
plt.plot(Vg,Vs,"-o")
plt.grid()
plt.xlabel("Vg")
plt.ylabel("Vs")
data = [Vg,Vs]
np.savetxt("pfet-4.csv",data,delimiter=",")
```



In [ ]:

```
# if it looks nice in the plot, save it!
Vg, Vs = np.loadtxt("pfet-4.csv", delimiter=",")

plt.plot(Vg,Vs,"o-",label="Vs")
plt.legend()
plt.title("Vs vs Vg pFet")
plt.xlabel('''Vg [V]
Fig.3.2 Vs vs Vg for $I_{ds}$ = 6e-08 in an pfet''')
plt.ylabel("Vs [V]")
plt.grid()
```

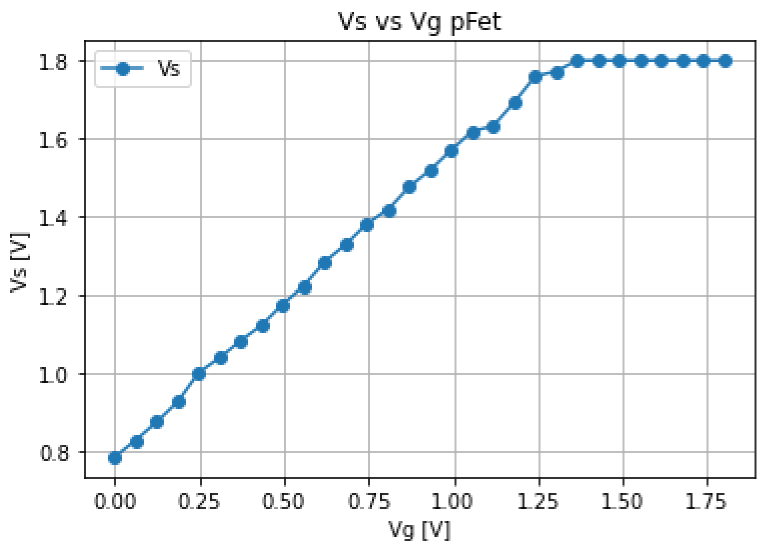


Fig.3.2 Vs vs Vg for $I_{ds}$ = 6e-08 in an pfet

- How do you compute $\kappa$? Does it stay constant for different $V_g$?

In [ ]:
```
# calculate kappa
kappa = [(Vs[i]-Vs[i-7])/(Vg[i]-Vg[i-7]) for i in range(7,20,4)]
print(r'Values of kappa at different points of the Vs vs Vg curve: ',kappa)
```

```
Values of kappa at different points of the Vs vs Vg curve:  [0.7767857142857141, 0.74
84654017857147, 0.7848772321428568, 0.7160993303571417]
```

*We can see that $\kappa$ varies in the 10e-2 order of magnitude*

## 5. Clean up

- Close you device and release memory by doing

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

- Save your changes
- Close jupyter notebook properly ( `File -> Close and Halt` )
- Save the files you need for the report to your own PC

## 6. Postlab

*The answers to the postlab should be included below.*

For the following questions assume an *n*-well process with the transistor in saturation.

Many differences in the properties of native and well transistors arise from the fact that the well is usually more heavily doped than the substrate. That was certainly true in the past when people were using a single well process. Today, most processes are double-well or *twin-tub* both n- and p-wells are implanted in a lightly doped epitaxially-grown p-substrate. *Epitaxially* means that the layer is grown atom by atom by chemical vapor deposition, resulting in a very regular and pure crystal structure.

The classchips you are using were fabricated in a 0.18 $\mu m$ process. For an *n*-well device, the *n*-type material of the well is the bulk, while the active areas (source and drain) are *p*-type. The gate voltage must force all the electrons in the *n*-well away from the surface. The resulting depletion region provides a channel for holes through enemy territory (n-well) separating the p-type source and drain. If the bulk is heavily doped, the gate must work harder to repel electrons.

Another way of saying this is that the capacitance of this depletion layer and the gate oxide capacitance form a *capacitive divider* that determines how much of the gate voltage appears at the surface channel. If the depletion layer is thin, the depletion capacitance will be large and hence the divider ratio will be unfavorable.

**(1)** How does the thickness of the depletion region depend on the doping and on the channel (surface) potential? Assume that the doping density is uniform.

From *Analog vlsi circuits and principles*, we have the following equation (2.6.15):

- $d = \sqrt{\frac{2\epsilon_s}{q}\frac{N_A+N_D}{N_AN_D}(\psi_{bi} - V)}$, where $d$ is the depletion region width.

We can see that the depletion region thickness is inversely proportional to the doping and proportional to the surface potential

**(2)** Explain why $\kappa$ varies with the source voltage at constant current (as in the source follower).

At constant current we have $\kappa = \frac{dVs}{dVg}$. As $V_g$ increases,the depletion width as well (and therefore $C_d$ decreases). So $\kappa$ increases with $V_g$. Because, $\kappa = \frac{C_{ox}}{C_{ox}+C_d}$

**(3)** Is there a difference in $\kappa$ between the n- and p-type devices? Explain the reason.

Yes, because in p-fets the n-well is more lightly doped than the p-substrate. This makes a difference in the depletion layer width and therefore in $C_d$ as well. (ultimately p-fets will have a larger $\kappa$ value for the same $V_g$ compared to n-fets)

**(4)** From your results in the experiments, state under what conditions the assumption that $\kappa$ is constant is reasonable.

Because $\kappa$ varies slowly with $V_g$, and we can assume it is constant in an approximate analysis of a subthreshold circuit