Neuromorphic engineering I

Lab 2: Transistor superthreshold saturation current and drain characteristics

Group number: 18

Team member 1: Quillan Favey

Date: 10.11.2011

The objective of this lab is to understand *super-threshold* (also called *above-threshold* or *strong inversion*) transistor operation and to understand transistor drain conductance characteristics, particularly *channel length modulation*.

The specific experimental **objectives of this lab** are as follows:

1. To characterize drain current of a transistor as a function of gate voltage in superthreshold operation in the ohmic (triode) and saturation regions.
2. To characterize the drain saturation properties in super-threshold.
3. To characterize drain conductance (the Early effect) and how it scales with transistor length (may not be possible this year) and saturation drain current.

An intuitive and quantitative understanding of all these effects, along with the subthreshold behavior (next week), is useful for the design of effective circuits, especially analog design of high performance amplifiers.

1 Terminology

- above-threshold = super-threshold = strong inversion
- sub-threshold = below-threshold = weak inversion
- triode region = ohmic region = linear drain conductance behavior with small drain-source voltage
- saturation = large V_{ds}
- overdrive = $V_{\mathrm{g}} - V_{\mathrm{T}}$
- $U_{\mathrm{T}} = kT/q$ = thermal voltage = 25mV at room temperature
- V_{T} = threshold voltage = 0.4V to 0.8V depending on process

2 Useful Quantities

The following is a list of the physical parameters and constants we will be referring to in this lab, along with their values when appropriate. The units that are most natural for these quantities are also included; these units are not self--consistent, so make sure you convert the units when appropriate.

ϵ_0 : Permittivity of vacuum = $8.86 \times 10^{-12} \mathrm{F/m}$

ϵ_{Si} : Relative permittivity of Si = $11.7\epsilon_0$

ϵ_{ox} : Relative permittivity of $\mathrm{SiO_2}$ = $3.9\epsilon_0$

μ_n : electron surface mobility, $\mathrm{cm^2/V/s}$

μ_p : hole surface mobility, $\mathrm{cm^2/V/s}$

C_{ox} : gate capacitance across the oxide per unit area, $\mathrm{fF}/\mu\mathrm{m}^2$

C_{dep} : capacitance of depletion region per unit area, $\mathrm{fF}/\mu\mathrm{m}^2$

t_{ox} : gate oxide thickness $\approx$ 3.8 nm for the class chip in 180 nm techology.

V_T : threshold voltage, V (V_{T0} is V_T when V_s = 0).

W : electrical width of transistor channel, $= 4$ μm for both devices in this lab

L : electrical length of transistor channel, $= 4$ μm for both devices in this lab

$\beta \equiv \mu C_{ox} W/L, \ \mu\mathrm{A/V^2}$

V_E : Early voltage, characterizes drain conductance.

3 Prelab

Write the expressions/eqations in LaTeX, like Vod = $V_\mathrm{g} - V_\mathrm{T}$, or upload the pictures of handwritten expressions.

- For nFET, write the most general expression for I_{ds} above threshold in terms of $V_g, \ V_s, \ V_d$ (all voltages are referenced to the bulk), and the parameters and constants given above. Leave out the drain conductance Early effect in this equation. Assume $\kappa = 1$ and that $V_{Tn} > 0$.

- Triode region: $I_{ds} = \beta(V_g - V_s - V_{Tn})(V_d - V_s)$
- Saturation region: $I_{ds} = \frac{\beta}{2}(V_g - V_s - V_{Tn})^2$

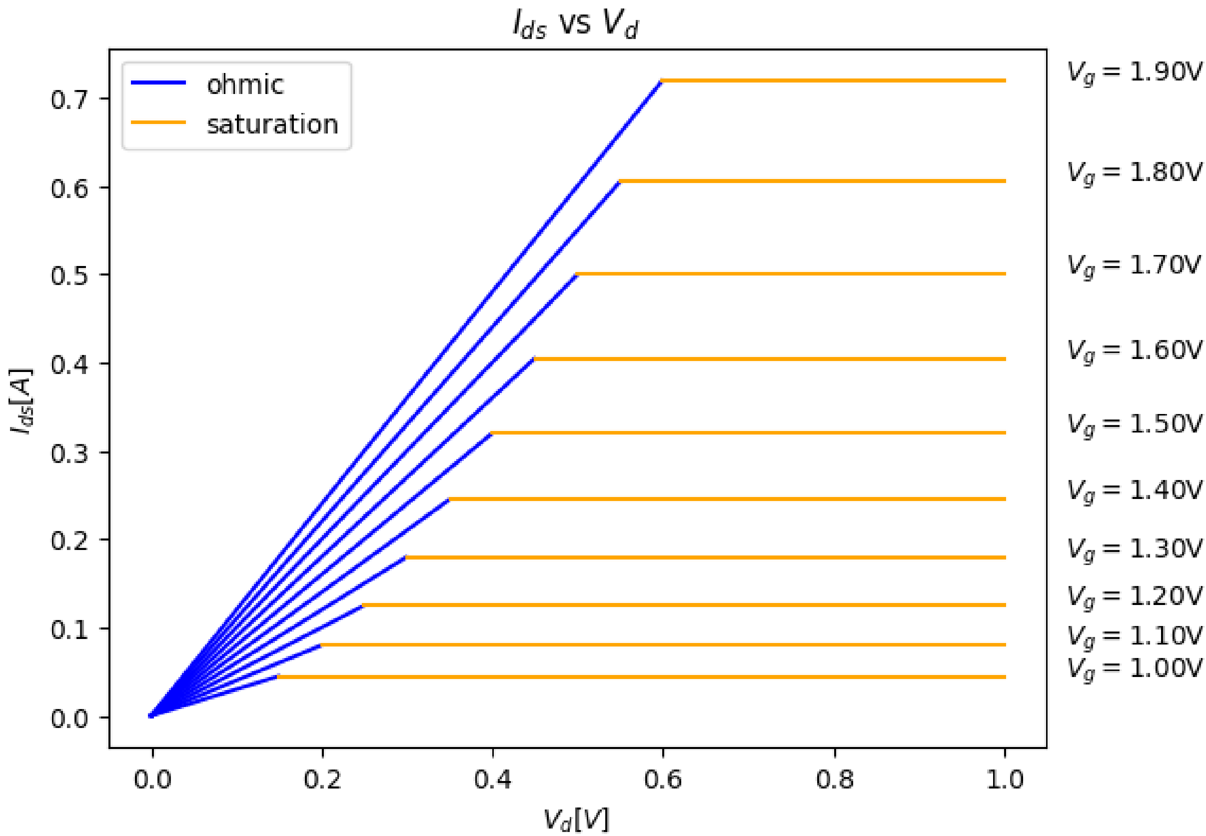
- For pFET, write the most general expression for I_{ds} above threshold in terms of $V_g, \ V_s, \ V_d$ (all voltages are referenced to the bulk), and the parameters and constants given above.

Leave out the drain conductance Early effect in this equation. Assume $\kappa = 1$ and that $V_{Tp} < 0$.

- Triode region: $I_{ds} = \beta(V_g - V_s - V_{Tp})(V_d - V_s)$
- Saturation region: $I_{ds} = \frac{\beta}{2}(V_g - V_s - V_{Tp})^2$

- For nFET, sketch graphs of I_{ds} vs the V_d for several gate voltages V_g above threshold, with $V_s = 0$. Indicate the ohmic and saturation regions and the behavior of the saturation voltage V_{dsat} as the gate overdrive voltage increases.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
v_t = 0.7
v_s = 0
beta = 1
for v_g in np.arange(1, 2, 0.1):
    v_d1 = np.arange(0, 1, 0.001)
    v_d2 = np.arange(0, 1, 0.001)
    i_ds1 = beta*(v_g - v_s - v_t)*(v_d1 - v_s)
    i_ds2 = np.repeat(beta*0.5*(v_g - v_s - v_t)**2, 1/0.001)
    index = next(filter(lambda i_ds: abs(i_ds[1][0]-i_ds[1][1]) < 0.0001, enumerate(zi
    plt.plot(v_d1[:index], i_ds1[:index],color ='blue')
    plt.plot(v_d2[index:], i_ds2[index:],color = 'orange')
    plt.text(1.07, i_ds2[-1], f"$V_g = {v_g:.2f}$V")
plt.title("$I_{ds}$ vs $V_{d}$ ")
plt.xlabel('''$V_d[V]$
Plot of $I_{ds}$ vs $V_{d}$
with $V_{t} = 0.8V, V_{s} = 0V $ and $\\beta = 1$''')
plt.ylabel("$I_{ds}[A]$")
plt.legend(["ohmic", "saturation"])
plt.show()
```

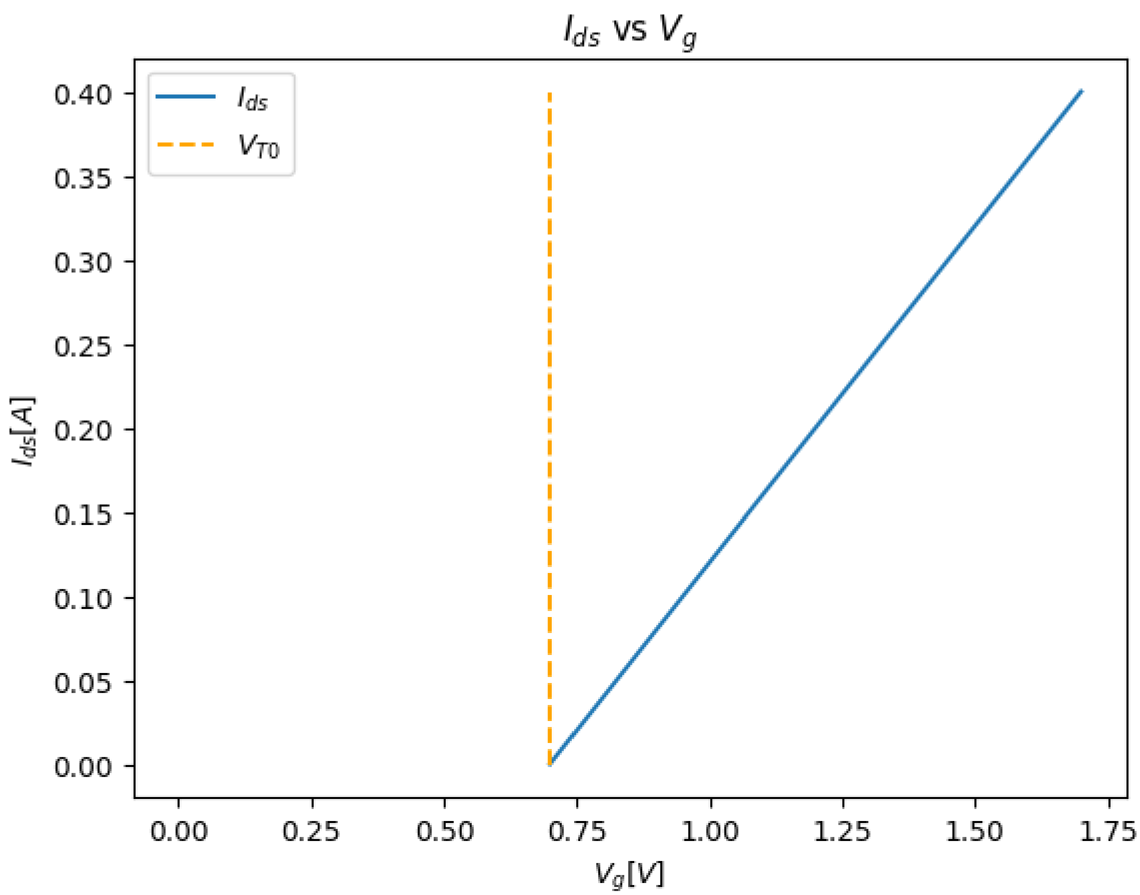


Plot of I_{ds} vs V_d
with $V_t = 0.8V$, $V_s = 0V$ and $\beta = 1$

- For nFET, derive an expression for the current I_{ds} in the ohmic region in terms of V_g and $V_{ds} \equiv V_d - V_s$,. You may assume that $V_s = 0$. Sketch a graph of I_{ds} vs V_g, showing V_{T0} and an expression for the slope.

$$I_{ds} = \beta(V_g - V_T)V_d$$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
v_t = 0.7
v_s = 0
v_d = 0.4
beta = 1
v_g = np.arange(0.0, 1.8, 0.1)
i_ds = beta*(v_g - v_s - v_t)*(v_d - v_s)
i_ds = np.array([max(i, 0) for i in i_ds])  #floor neagtive values
plt.plot(v_g[:8], i_ds[:8], color="white")
plt.plot(v_g[7:], i_ds[7:], label="$I_{ds}$")
plt.xlabel('''$V_g [V]$
Plot of $I_{ds}$ vs $V_{g}$
for $V_{t}$ = 0.7V, $V_{d}$ = 0.4V, $V_{s}$ = 0V, $\\beta$ = 1,
 Slope = $V_{d}  \\beta = 0.4$''') #slope is from y=mx+h , m=beta*Vd
plt.ylabel("$I_{ds} [A]$")
plt.title("$I_{ds}$ vs $V_{g}$")
plt.vlines(v_t, 0, v_d, linestyles="dashed", label="$V_{T0}$",color = 'orange')
plt.legend()
plt.show()
```



Plot of I_{ds} vs V_g
for V_t = 0.7V, V_d = 0.4V, V_s = 0V, β = 1,
Slope = $V_d\beta = 0.4$

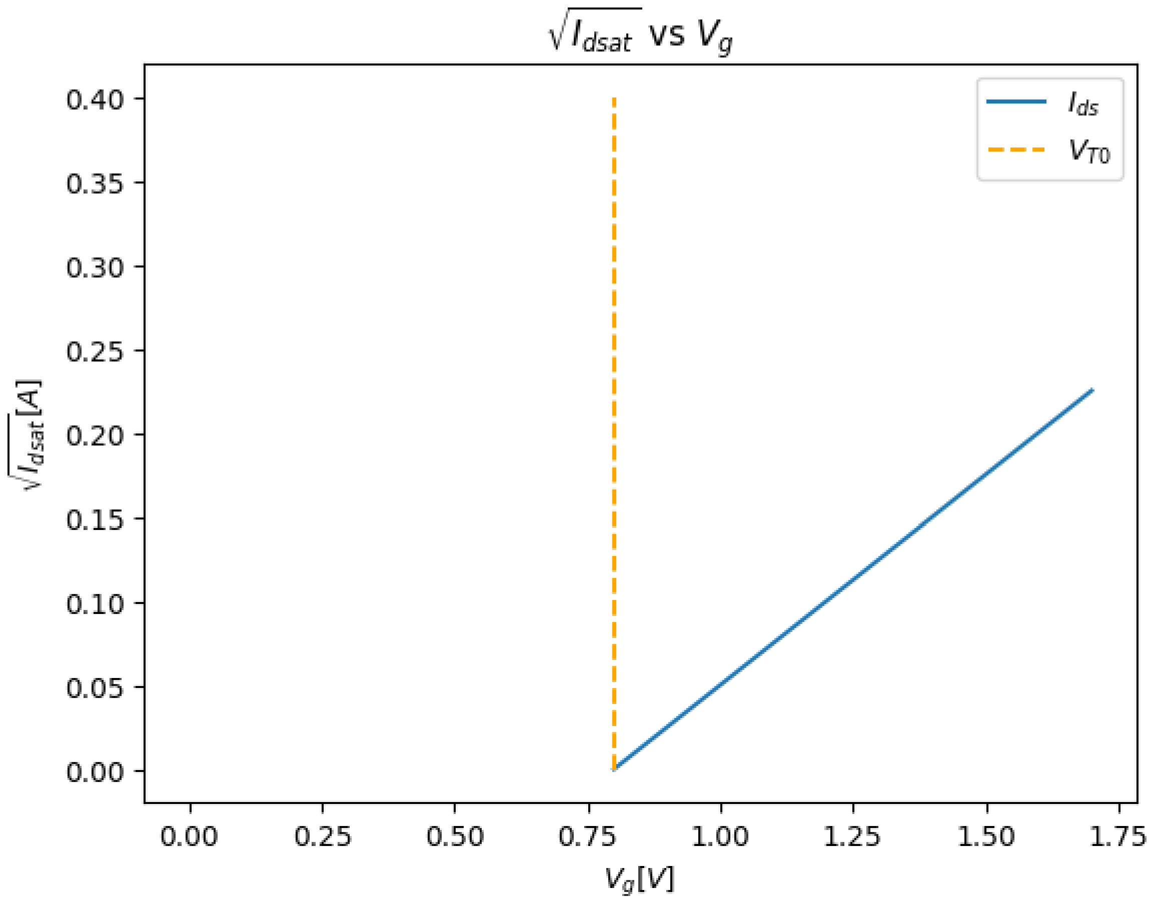
- For nFET, state the drain voltage condition for above-threshold saturation and derive an expression for the saturation current I_{dsat} in terms of V_g. Sketch a graph of $\sqrt{I_{dsat}}$ vs V_g with $V_s = 0$, showing V_{T0} and an expression for the slope. Do not consider the Early effect here.

The drain voltage condition is to be at or below threshold ($V{d}<=V{gs}-V{T}=V{ov}$)

$I_{dsat} = \frac{\beta}{2}(V_g - V_{T0})^2$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
v_t = 0.8
v_s = 0
beta = 1
v_g = np.arange(0.0, 1.8, 0.1)
i_ds = np.sqrt(beta)/4*(v_g - v_s - v_t)
i_ds = np.array([max(i, 0) for i in i_ds])  #floor neagtive values
plt.plot(v_g[:9], i_ds[:9],color="white")
plt.plot(v_g[8:], i_ds[8:], label="$I_{ds}$")
plt.title("$\\sqrt{I_{dsat}}$ vs $V_{g}$")
plt.xlabel('''$V_g [V]$
Plot of $\\sqrt{I_{dsat}}$ vs $V_{g}$
```

```
for $V_{t}$ = 0.8V, $V_{s}$ = 0V, $\\beta$ = 1,
 Slope = $\\sqrt{\\beta}/4 = 1/4$''') #slope is from y=mx+h , m=beta*Vd
plt.ylabel("$\\sqrt{I_{dsat}} [A]$")
plt.vlines(v_t, 0, v_d, linestyles="dashed", label="$V_{T0}$",color = 'orange')
plt.legend()
plt.show()
```



Plot of $\sqrt{I_{dsat}}$ vs V_g
for V_t = 0.8V, V_s = 0V, β = 1,
Slope = $\sqrt{\beta}/4 = 1/4$

- Calculate C_{ox} for the classchip from the values given above. What is C_{ox} per square micron in fF?

$$C_{ox} = \epsilon_{ox}/t_{ox} = \frac{3.9\times 8.86\times 10^{-12}\mathrm{F/m}}{3.8\times 10^{-9}m} = 9.0931 \times 10^{-3}\frac{F}{m^2} = 9093 fF/\mu m^2$$

- Write the expression for the drain current in saturation including the Early effect, using I_{dsat} to represent the saturation current in the absence of the Early effect. Use V_E to represent the Early voltage.

$$I_{ds} = I_{dsat}(1 + \frac{V_{ds}}{V_E})$$

4 Setup

-

4.1 Connect the device

In [ ]:
```
# import the necessary library to communicate with the hardware

import pyplane
import time
```

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()   #firmware version should be 1.8.3
```

Out[ ]:
```
(1, 8, 4)
```

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.GO20_N)
```

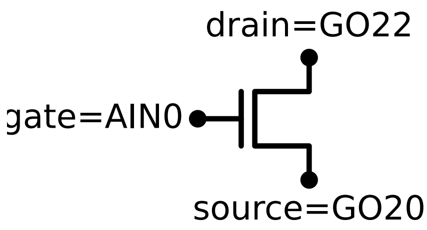
Out[ ]:
```
8.78906234902388e-07
```

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

# del p
```

4.2 Configurations for N-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.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[ ]:



To cancel out the leakage current and shunt resistance, you may need to do a subtraction in Section 5.1.

$$I_{ds} = I_{GO20} - I_{GO20}|_{V_{gs}=0}$$

Note: It's better to measure source because its leakage is constant in this lab

- You have to set the input voltage demultiplexer by sending a configuration event:

In [ ]:
```
# Configure NFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the NFET
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)
```

- Check the configuration is correct. If the measured result is not as expected, try sending the configration event again.

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

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

In [ ]:
```
# set drain voltage
vd = 1.8
p.set_voltage(pyplane.DacChannel.GO22,vd)
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 gate voltage
vg = 1
p.set_voltage(pyplane.DacChannel.AIN0, vg)
print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChannel.AI
```

```
The gate voltage is set to 0.9994136095046997 V
```

In [ ]:
```
# read I_{ds}
I_s = p.read_current(pyplane.AdcChannel.GO20_N)      #source: note the pin name is dif
```

```
print("The measured source current is {} A".format(I_s))

time.sleep(0.1)  # wait for it to settle

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

```
The measured source current is 2.1142577679711394e-05 A
The measured drain current is 1.8505859770812094e-05 A
```

- Question: Check if the measured currents change with different gate voltages?

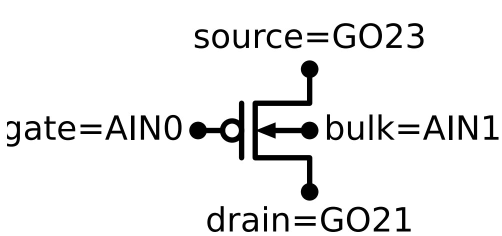
They do (higher VG = higher current through the transistor)

4.3 Configurations for 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 need to do a subtraction:

$$I_{ds} = I_{GO23} - I_{GO23}|_{V_{gs}=0}$$

Note: Measure drain of PFET in this lab. Also think about the difference of V_{gs} between PMOS and NMOS?

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

In [ ]:

```
# Configure PFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the PFET
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)
```

- Check the configuration is correct. If the measured result is not as expected, try sending the event again.

In [ ]:
```
# set trial voltages
Vsp = 1.8
Vdp = 0
Vgp = 1.0
# set bulk voltage
p.set_voltage(pyplane.DacChannel.AIN1, Vsp)
Vb_p = p.get_set_voltage(pyplane.DacChannel.AIN1)
print("The bulk voltage is set to {} V".format(Vb_p))
time.sleep(0.1)  # wait 0.1s for it to settle

# set source voltage
p.set_voltage(pyplane.DacChannel.GO23, Vsp)
Vs_p = p.get_set_voltage(pyplane.DacChannel.GO23)
print("The source voltage is set to {} V".format(Vs_p))
time.sleep(0.1)  # wait 0.1s for it to settle

# set drain voltage
p.set_voltage(pyplane.DacChannel.GO21, Vdp)
Vd_p = p.get_set_voltage(pyplane.DacChannel.GO21)
print("The drain voltage is set to {} V".format(Vd_p))
time.sleep(0.1)  # wait for it to settle

# set gate voltage
p.set_voltage(pyplane.DacChannel.AIN0, Vgp)
Vg_p = p.get_set_voltage(pyplane.DacChannel.AIN0)
print("The gate voltage is set to {} V".format(Vg_p))
```

```
The bulk voltage is set to 1.7982406616210938 V
The source voltage is set to 1.7982406616210938 V
The drain voltage is set to 0.0 V
The gate voltage is set to 0.9994136095046997 V
```

In [ ]:
```
# read I_{ds}
Is_p = p.read_current(pyplane.AdcChannel.GO21_N)
print("The measured source current of PMOS is {} A".format(Is_p))

time.sleep(0.1)  # wait for it to settle

Id_p = p.read_current(pyplane.AdcChannel.GO23)
print("The measured drain current of PMOS is {} A".format(Id_p))
```

```
The measured source current of PMOS is 1.6601562720097718e-06 A
The measured drain current of PMOS is 6.005859177093953e-06 A
```

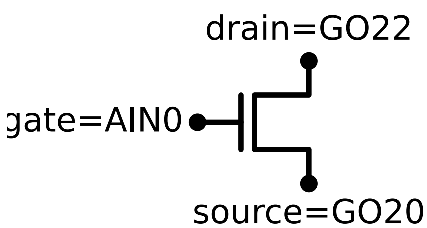
5 Ohmic region

In this experiment you will characterize the *linear* dependence of the current on the gate voltage in the strong-inversion ohmic region.

5.1 N-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.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[ ]:



(a) Configure the chip following Section 4.2 if you haven't

(b) Measure I_{ds} as a function of V_g in ohmic region

In [ ]:
```
# Configure NFET, set the input voltage demultiplexer by AER event.
# Configure NFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the NFET
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)
```

- What will be the fixed value for source and drain voltages?

Answer:

In [ ]:
```
# set source voltage
Vs = 0
p.set_voltage(pyplane.DacChannel.GO20,Vs)
```

Out[ ]:
```
0.0
```

We don't want a too high Vds and end up in the saturation region

In [ ]:
```
# set drain voltage
Vd = 0.2
p.set_voltage(pyplane.DacChannel.GO22,Vd)
```

Out[ ]:
```
0.19882699847221375
```

- For very close voltages, you may want to call `get_set_voltage` to check the actual output of the DAC.

In [ ]:
```
# get set voltage
Vs_n = p.get_set_voltage(pyplane.DacChannel.GO20)
print("The source voltage is set to {} V".format(Vs_n))

time.sleep(0.1)  # wait for it to settle

# get set voltage
Vd_n = p.get_set_voltage(pyplane.DacChannel.GO22)
print("The drain voltage is set to {} V".format(Vd_n))
```

```
The source voltage is set to 0.0 V
The drain voltage is set to 0.19882699847221375 V
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
import time
import numpy as np

# Get the leakage current, Read Ids=Ids0 at Vg = 0
p.set_voltage(pyplane.DacChannel.AIN0,0)
time.sleep(0.5) # wait 0.5 second for it to settle
Is0_n = p.read_current(pyplane.AdcChannel.GO20_N) #REMEMBER: reading from source is pi
print("Offset Is0_n: {} A".format(Is0_n))

IDS = []
read_Ids = 0
for gate in np.arange(0,1.8,0.1) :
    # set gate voltage
    p.set_voltage(pyplane.DacChannel.AIN0,gate)

    print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChanne

    time.sleep(0.05)  # wait for it to settle
    # read I_{ds}
    read_Ids = p.read_current(pyplane.AdcChannel.GO20_N)


    print("The measured source current is {} A".format(read_Ids))  ## print the raw da

    # substract leakage current
    IDS.append(read_Ids-Is0_n)
```

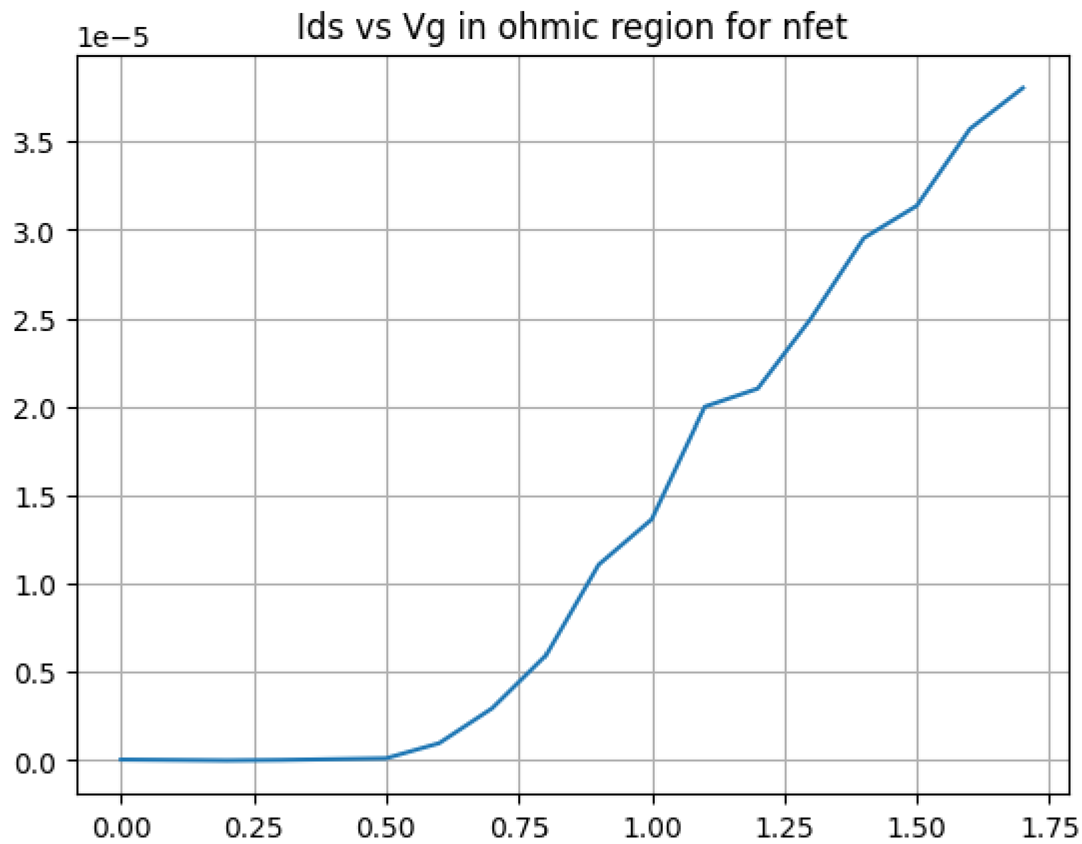
```
Offset Is0_n: 4.882812731921149e-07 A
The gate voltage is set to 0.0 V
The measured source current is 5.37109372089617e-07 A
The gate voltage is set to 0.0985337346792221 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 0.19882699847221375 V
The measured source current is 4.882812731921149e-07 A
The gate voltage is set to 0.2991202771663666 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 0.399413526058197 V
The measured source current is 5.615234499600774e-07 A
The gate voltage is set to 0.49970680475234985 V
The measured source current is 6.103515488575795e-07 A
The gate voltage is set to 0.5982405543327332 V
The measured source current is 1.4648437627329258e-06 A
The gate voltage is set to 0.698533833026886 V
The measured source current is 3.442382876528427e-06 A
The gate voltage is set to 0.798827052116394 V
The measured source current is 6.39648442302132e-06 A
The gate voltage is set to 0.8991203308105469 V
The measured source current is 1.1572265975701157e-05 A
The gate voltage is set to 0.9994136095046997 V
The measured source current is 1.4135742276266683e-05 A
The gate voltage is set to 1.0997068881988525 V
The measured source current is 2.0483397747739218e-05 A
The gate voltage is set to 1.1982406377792358 V
The measured source current is 2.150878935935907e-05 A
The gate voltage is set to 1.2985339164733887 V
The measured source current is 2.5463867132202722e-05 A
The gate voltage is set to 1.3988271951675415 V
The measured source current is 3.0029297704459168e-05 A
The gate voltage is set to 1.4991203546524048 V
The measured source current is 3.1860352464718744e-05 A
The gate voltage is set to 1.5994136333465576 V
The measured source current is 3.6206056392984465e-05 A
The gate voltage is set to 1.6997069120407104 V
The measured source current is 3.852539157378487e-05 A
```

In [ ]:

```python
# plot
Vg = np.arange(0,1.8,0.1)
plt.plot(Vg,IDS)
plt.grid()
plt.title("Ids vs Vg in ohmic region for nfet")
```

Out[ ]:

```
Text(0.5, 1.0, 'Ids vs Vg in ohmic region for nfet')
```

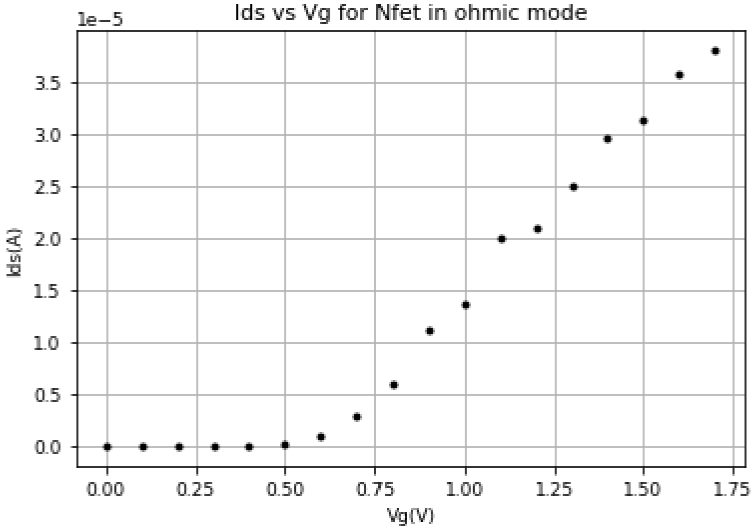


In [ ]:

```
# if the data looks nice, save it!
data = [Vg,IDS]
np.savetxt("Lab2_data_nFETVgIds.csv",data,delimiter=",")
```

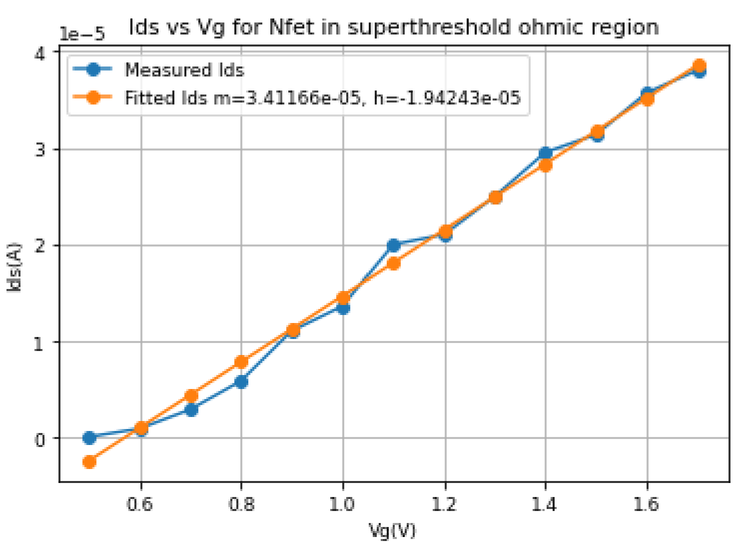
In [ ]:

```
# Load data you saved and plot, to check if the data is saved correctly or not
import numpy as np
import matplotlib.pyplot as plt

Vgn_save, Isn_save = np.loadtxt('Lab2_data_nFETVgIds.csv',delimiter=",")
plt.rcParams.update({'font.size': 9})
plt.plot(Vgn_save, Isn_save, '.k')
plt.xlabel('Vg(V)')
plt.ylabel('Ids(A)')
plt.title("Ids vs Vg for Nfet in ohmic mode")
plt.grid()
plt.show()
```



In [ ]:
```
# extract the valid range
Ids_valid = Isn_save[5:]
Vg_valid = Vgn_save[5:]
#and store data for part 5.3
nfet_Ids = Isn_save
nfet_Vg = Vgn_save
```

In [ ]:
```
# fit in the valid range (you may want to go back and add the fitted line in the plot)
#
#We can use polyfit to fit a regression line to our data
coffs = np.polyfit(Vg_valid,Ids_valid,1)
Ids_fit = []
for i in Vg_valid:
    Ids_fit.append(coffs[0]*i + coffs[1])

plt.plot(Vg_valid,Ids_valid,'o-',label="Measured Ids")
plt.plot(Vg_valid,Ids_fit,'o-',label=f"Fitted Ids m={np.round(coffs[0],10)}, h={np.rou
plt.legend()
plt.xlabel('Vg(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vg for Nfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



(c) Determine V_{T0} and β for both devices by fitting your data to the expression derived in the prelab

In [ ]:
```python
# V_T0
from scipy.optimize import curve_fit
#define function
def Ids_nFET_func(Vg,beta,Vt):
        Vs = 0.0
        Vd = 0.2
        return beta*(Vg-Vs-Vt)*(Vd-Vs)
#fit curve
popt, pcov = curve_fit(Ids_nFET_func,Vg_valid,Ids_valid,p0=[1.1, 2e-5])
#get parameters
fit_beta,fit_vt = popt
#compute theoret. data with fitted parameters
fitted_y = Ids_nFET_func(Vg_valid,fit_beta,fit_vt)
#print result
v_t0 = fit_vt
print('Fitted Vt0: ',v_t0)
```

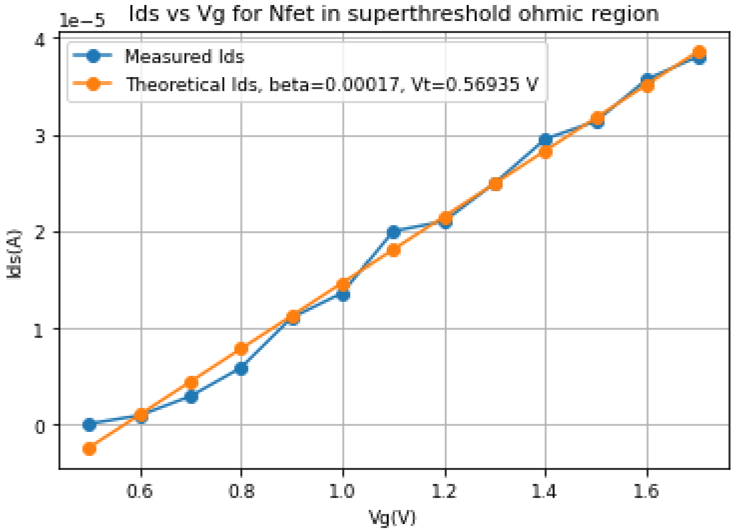
```
Fitted Vt0:  0.5693508502241923
```

In [ ]:
```python
# beta => m/Vd

beta = fit_beta #no need rto divide by Vd as we already computed beta

print("Fitted beta: ",fit_beta)
print("beta from slope:",coffs[0]/0.2)
```

```
Fitted beta:  0.00017058320621908408
beta from slope: 0.00017058320628955814
```

We can put everything in one plot

In [ ]:
```python
Ids_theo = [fit_beta*(i-fit_vt)*0.2 for i in Vg_valid]

plt.plot(Vg_valid,Ids_valid,'o-',label="Measured Ids")
plt.plot(Vg_valid,Ids_theo,'o-',label=f"Theoretical Ids, beta={np.round(fit_beta,5)},
plt.legend()
plt.xlabel('Vg(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vg for Nfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



5.2 P-FET

(a) Configure the chip following Section 4.3 if you haven't

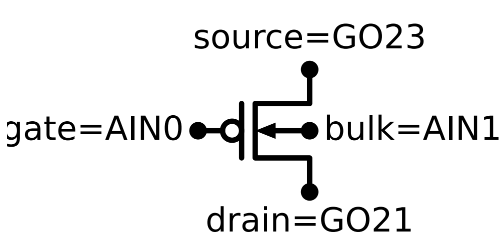
(b) Measure I_{ds} as a function of V_g in ohmic region

- What will be the fixed value for bulk, source 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.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[ ]:



In [ ]:

```
# Configure PFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the PFET
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)
```

In [ ]:
```
# set bulk voltage
p.set_voltage(pyplane.DacChannel.AIN1,1.8)

time.sleep(0.05)  # wait for it to settle

# set source voltage
p.set_voltage(pyplane.DacChannel.GO23,1.8)

# set drain voltage
p.set_voltage(pyplane.DacChannel.GO21,0.8)
# Print I_ds for checking
print(p.read_current(pyplane.AdcChannel.GO21_N))
```

```
3.0761718790017767e-06
```

- For very close voltages, you may want to call `get_set_voltage` to check the actual output of the DAC.

In [ ]:
```
# get set voltage
print("drain",p.get_set_voltage(pyplane.DacChannel.GO21))#drain
print("source",p.get_set_voltage(pyplane.DacChannel.GO23))#source
```

```
drain 0.798827052116394
source 1.7982406616210938
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
# sweep gate voltage
import time
import numpy as np

# Get the leakage current, Read Ids=Ids0 at Vg = 0
p.set_voltage(pyplane.DacChannel.AIN0,1.8)
time.sleep(0.5) # wait 0.5 second for it to settle
Is0_n = p.read_current(pyplane.AdcChannel.GO21_N) #REMEMBER: reading from source is pi
print("Offset Is0_n: {} A".format(Is0_n))

IDS = []
read_Ids = 0
for gate in np.arange(0,1.8,0.1) :
    # set gate voltage
    p.set_voltage(pyplane.DacChannel.AIN0,gate)

    print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChanne

    time.sleep(0.05)  # wait for it to settle
    # read I_{ds}
    read_Ids = p.read_current(pyplane.AdcChannel.GO21_N)


    print("The measured source current is {} A".format(read_Ids))  ## print the raw da

    # substract leakage current
    IDS.append(read_Ids-Is0_n)
```

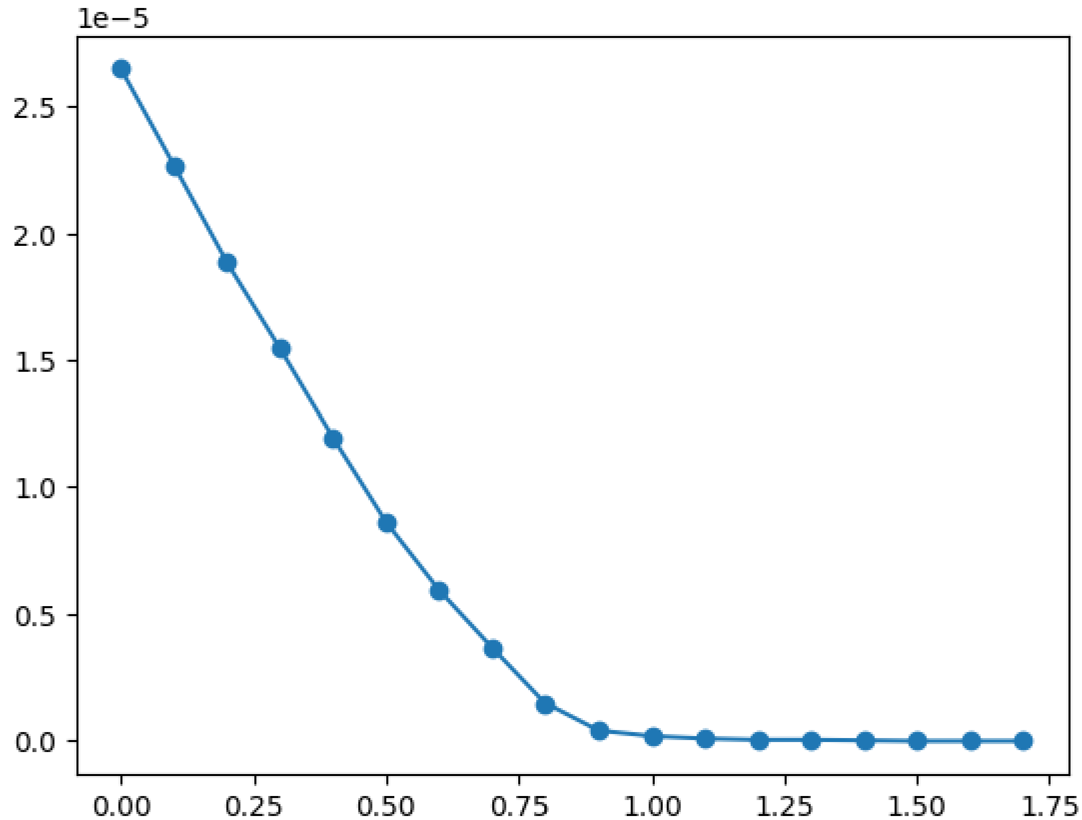
```
Offset Is0_n: 3.1005859000288183e-06 A
The gate voltage is set to 0.0 V
The measured source current is 2.9589844416477717e-05 A
The gate voltage is set to 0.0985337346792221 V
The measured source current is 2.5781249860301614e-05 A
The gate voltage is set to 0.19882699847221375 V
The measured source current is 2.194824264734052e-05 A
The gate voltage is set to 0.2991202771663666 V
The measured source current is 1.8554686903371476e-05 A
The gate voltage is set to 0.399413526058197 V
The measured source current is 1.5039062418509275e-05 A
The gate voltage is set to 0.49970680475234985 V
The measured source current is 1.1718750101863407e-05 A
The gate voltage is set to 0.5982405543327332 V
The measured source current is 9.033203241415322e-06 A
The gate voltage is set to 0.698533833026886 V
The measured source current is 6.762695193174295e-06 A
The gate voltage is set to 0.798827052116394 V
The measured source current is 4.565429662761744e-06 A
The gate voltage is set to 0.8991203308105469 V
The measured source current is 3.49121091858251e-06 A
The gate voltage is set to 0.9994136095046997 V
The measured source current is 3.271484274591785e-06 A
The gate voltage is set to 1.0997068881988525 V
The measured source current is 3.1738281904836185e-06 A
The gate voltage is set to 1.1982406377792358 V
The measured source current is 3.12499992105586e-06 A
The gate voltage is set to 1.2985339164733887 V
The measured source current is 3.12499992105586e-06 A
The gate voltage is set to 1.3988271951675415 V
The measured source current is 3.1005859000288183e-06 A
The gate voltage is set to 1.4991203546524048 V
The measured source current is 3.0761718790017767e-06 A
The gate voltage is set to 1.5994136333465576 V
The measured source current is 3.0761718790017767e-06 A
The gate voltage is set to 1.6997069120407104 V
The measured source current is 3.0761718790017767e-06 A
```

In [ ]:

```python
import matplotlib.pyplot as plt
# plot
Vg = np.arange(0,1.8,0.1)
plt.plot(Vg,IDS,"o-")
```

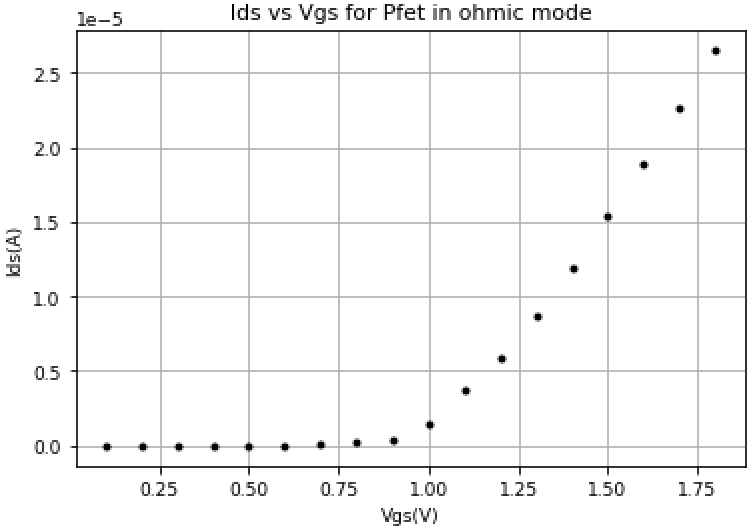
Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f1fd6ddbb80>]
```



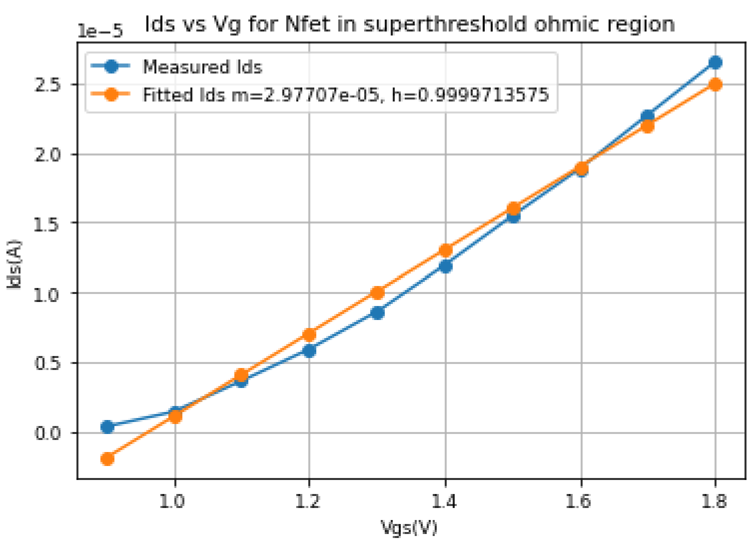
In [ ]:
```python
# if the data looks nice, save it!
data = [Vg,IDS]
np.savetxt("data_pfet_52.csv",data,delimiter = ",")
```

In [ ]:
```python
# Load data you saved and plot, to check if the data is saved correctly or not
Vgn_save , Isn_save = np.loadtxt("data_pfet_52.csv",delimiter = ",")
Vgn_save = 1.8-Vgn_save
Vgn_save.sort()
Isn_save.sort()
plt.rcParams.update({'font.size': 9})
plt.plot(Vgn_save, Isn_save, '.k')
plt.xlabel('Vgs(V)')
plt.ylabel('Ids(A)')
plt.title("Ids vs Vgs for Pfet in ohmic mode")
plt.grid()
plt.show()
```



In [ ]:
```
# extract the valid range
Ids_valid = Isn_save[8:]   #first values ar negative
Vg_valid = Vgn_save[8:]
#save data for part 5.3
pfet_IDS = Isn_save
pfet_Vg = Vgn_save
```

In [ ]:
```
# fit in the valid range (you may want to go back and add the fitted line in the plot)
#
#We can use polyfit to fit a regression line to our data
coffs = np.polyfit(Vg_valid,np.exp(Ids_valid),1)
Ids_fit = []
for i in Vg_valid:
    Ids_fit.append(coffs[0]*i + coffs[1])

plt.plot(Vg_valid,Ids_valid,'o-',label="Measured Ids")
plt.plot(Vg_valid,np.log(Ids_fit),'o-',label=f"Fitted Ids m={np.round(coffs[0],10)}, h
plt.legend()
plt.xlabel('Vgs(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vg for Nfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



(c) Determine V_{T0} and β for both devices by fitting your data to the expression derived in the prelab

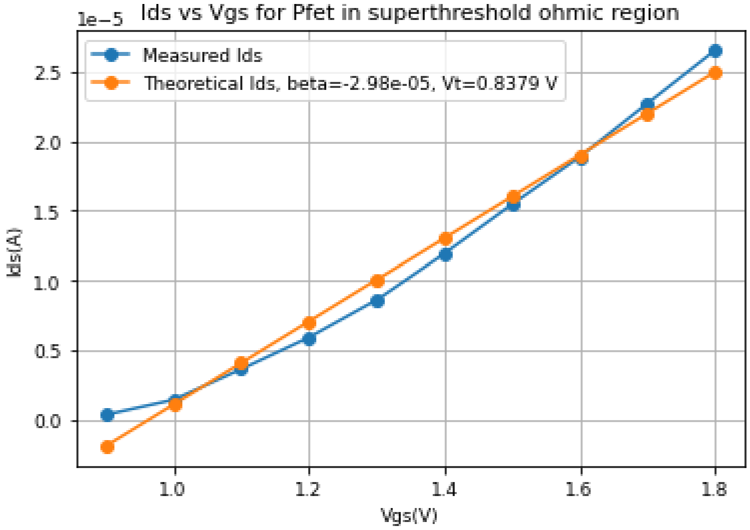
In [ ]:
```
# V_T0
from scipy.optimize import curve_fit
#define function
def Ids_nFET_func(Vgs,beta,Vt):
        Vs = 1.8
        Vd = 0.8
        return beta*(Vgs-Vt)*(Vd-Vs)
#fit curve
popt, pcov = curve_fit(Ids_nFET_func,Vg_valid,Ids_valid,p0=[0.6, 2e-8])
#get parameters
fit_beta,fit_vt = popt
#compute theoret. data with fitted parameters
fitted_y = Ids_nFET_func(Vg_valid,fit_beta,fit_vt)
#print result
v_t0 = fit_vt
print('Fitted Vt0: ',1.8-v_t0)
```

```
Fitted Vt0:  0.8378976123774671
```

In [ ]:
```
# beta => m/Vd
#from slope
print("beta from slope of regression fit: ",coffs[0]/0.8)   #probably more plausible b
#from scipy
beta = fit_beta #no need to divide by Vd as we already computed beta

print("Fitted beta (scipy): ",beta)
```

```
beta from slope of regression fit:  3.721341571456795e-05
Fitted beta (scipy):  -2.977036018962277e-05
```

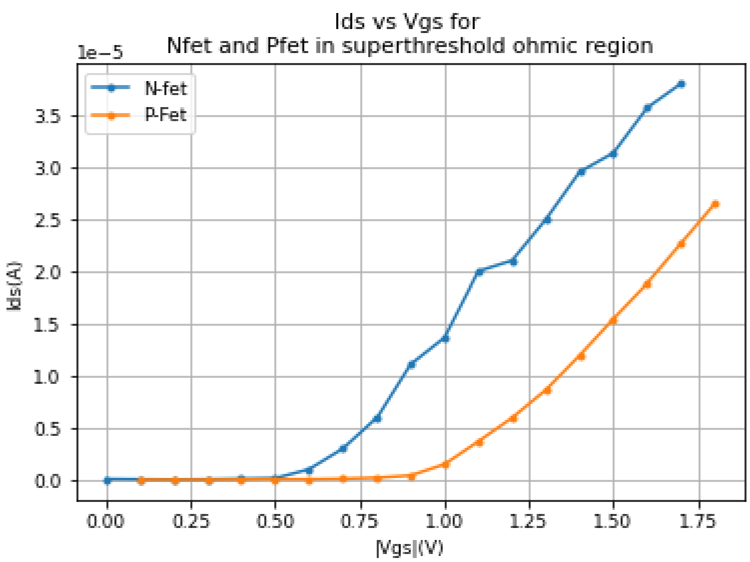
In [ ]:
```
Ids_theo = [fit_beta*(i-fit_vt)*(0.8-1.8) for i in Vg_valid]    #update formual for p-

plt.plot(Vg_valid,Ids_valid,'o-',label="Measured Ids")
plt.plot(Vg_valid,Ids_theo,'o-',label=f"Theoretical Ids, beta={np.round(fit_beta,7)},
plt.legend()
plt.xlabel('Vgs(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vgs for Pfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



5.3 Comparisons

- Include a single plot showing the curves for both devices.

In [ ]:
```python
# plot both Ids vs |Vgs|

plt.plot(nfet_Vg,nfet_Ids ,'.-',label="N-fet")
plt.plot(np.abs(pfet_Vg),pfet_IDS,'.-',label=f"P-Fet")
plt.legend()
plt.xlabel('|Vgs|(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vgs for
Nfet and Pfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



- What is the ratio between β for the 2 devices? Does it make sense?

In [ ]:
```python
betap = 3.721e-05
```

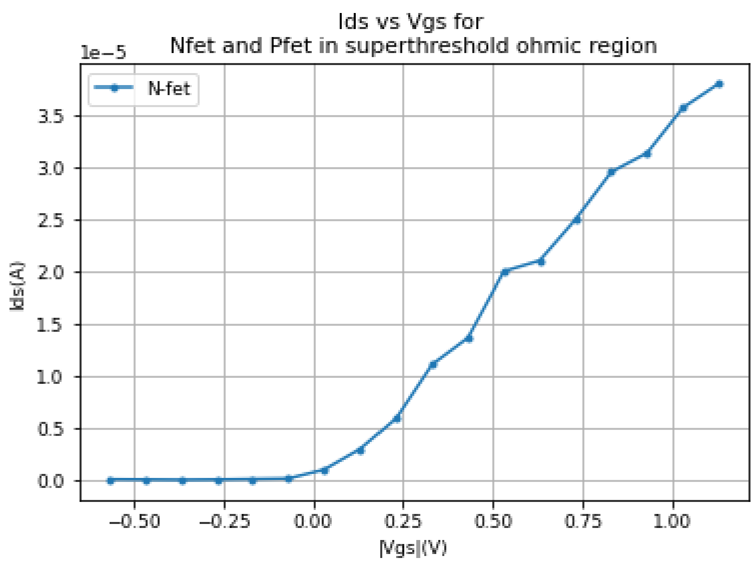
```
betan = 0.00017058320621908408
print(f'ratio between betas: ',betap/betan)
```

```
ratio between betas:  0.21813401696888207
```

The ratio should be around 1 but this not the case probably because the Vds that were set for pfet and nfet are not equivalent (0.8 for pfet and 0.2 for nfet)

- Is the relationship between I_{ds} and $V_{gs} - V_T$ really linear? What is likely the cause of any discrepancy?

In [ ]:
```
plt.plot(nfet_Vg-0.5693508,nfet_Ids ,'.-',label="N-fet")
plt.legend()
plt.xlabel('|Vgs|(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vgs for
Nfet and Pfet in superthreshold ohmic region''')
plt.grid()
plt.show()
```



The relationship seems to be linear (omitting the negative values). According to the lectures, when the channel is short and the field is high along the channel, carriers will saturate at thermal velocity. So Ids is a function of V_{ov}^2 to a certain point and then increases linearly with V_{ov}

5.4 Effective surface mobility (optional)

Hint: Use the V_{T0} you obtained in the last experiments but assume β changes with V_{gs} (thus μ_n and μ_p changes). **No need to measure again.**

In [ ]:
```
# plot mu vs Vgs for both devices in the same figure
```

- Why does the mobility peak and then decay instead of remaining constant?

- What is the ratio between the peak mobilities for electrons and holes?

- How different are these values from the bulk mobilities for electrons (1350 $\mathrm{cm}^2/\mathrm{V}/\mathrm{s}$) and holes (480 $\mathrm{cm}^2/\mathrm{V}/\mathrm{s}$)?

6 Drain Current in the saturation region

In this experiment you will characterize the *quadratic* dependence of the current on the gate voltage in the saturation region.

6.1 N-FET

(a) Configure the chip following Section 4.2 if you haven't

(b) Measure I_{ds} as a function of V_g in saturation region

- What will be the fixed value for source and drain voltages?

In [ ]:
```
## configure NMOS by AER event
# Configure NFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the NFET
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)
```

In [ ]:
```
# set source voltage
Vs = 0
p.set_voltage(pyplane.DacChannel.GO20,Vs)
```

Out[ ]:
```
0.0
```

In [ ]:
```
# set drain voltage     #######1.8
Vd = 1.8
p.set_voltage(pyplane.DacChannel.GO22,Vd)
```

Out[ ]:
```
1.7982406616210938
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
# sweep gate voltage
import time
import numpy as np

# Get the leakage current, Read Ids=Ids0 at Vg = 0
p.set_voltage(pyplane.DacChannel.AIN0,0)
time.sleep(0.5) # wait 0.5 second for it to settle
Is0_n = p.read_current(pyplane.AdcChannel.GO20_N) #REMEMBER: reading from source is pi
print("Offset Is0_n: {} A".format(Is0_n))

IDS = []
read_Ids = 0
for gate in np.arange(0.0,1.8,0.1) :
    # set gate voltage
    p.set_voltage(pyplane.DacChannel.AIN0,gate)

    print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChanne

    time.sleep(0.05)  # wait for it to settle
    # read I_{ds}
    read_Ids = p.read_current(pyplane.AdcChannel.GO20_N)


    print("The measured source current is {} A".format(read_Ids))  ## print the raw dc

    # substract leakage current
    IDS.append(read_Ids-Is0_n)
```

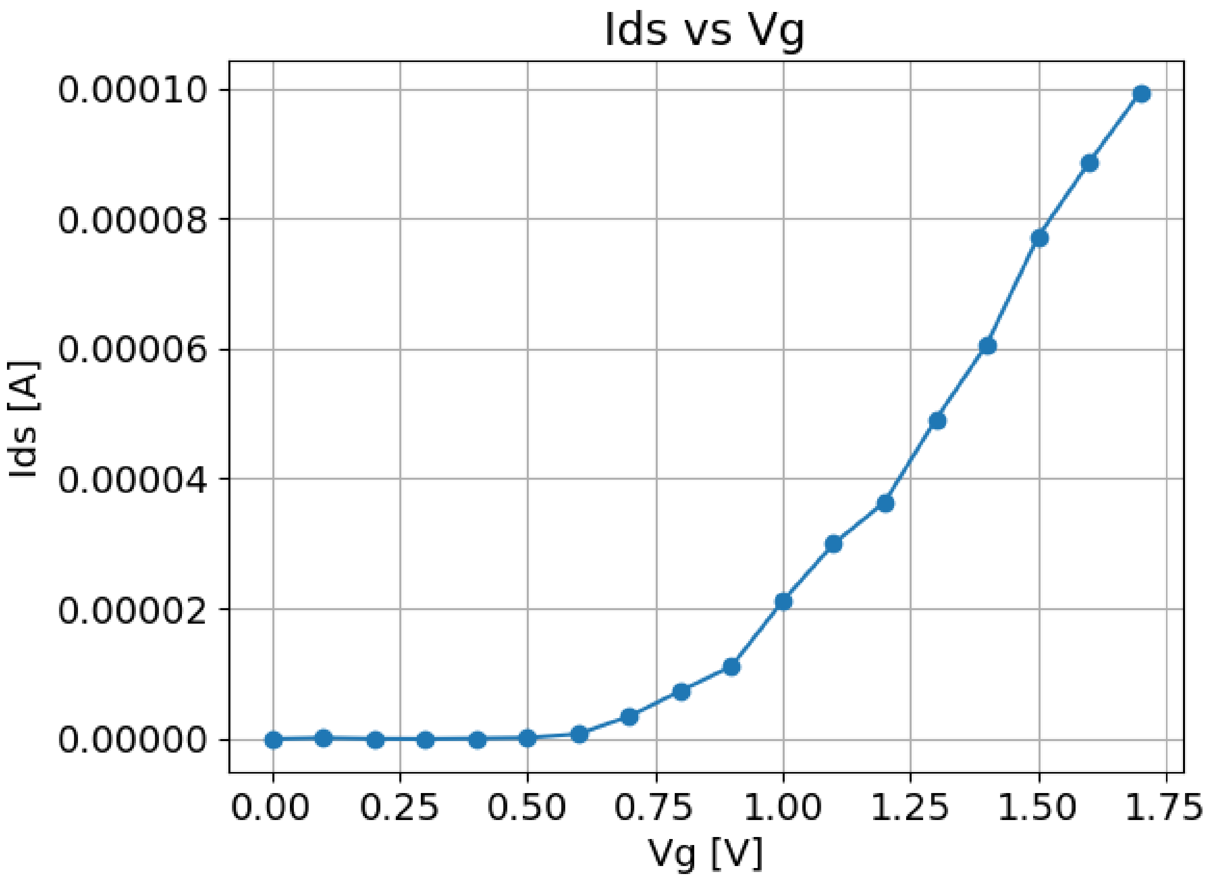
```
Offset Is0_n: 6.103515488575795e-07 A
The gate voltage is set to 0.0 V
The measured source current is 4.882812731921149e-07 A
The gate voltage is set to 0.0985337346792221 V
The measured source current is 6.347656267280399e-07 A
The gate voltage is set to 0.19882699847221375 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 0.2991202771663666 V
The measured source current is 4.882812731921149e-07 A
The gate voltage is set to 0.399413526058197 V
The measured source current is 5.615234499600774e-07 A
The gate voltage is set to 0.49970680475234985 V
The measured source current is 6.591797045985004e-07 A
The gate voltage is set to 0.5982405543327332 V
The measured source current is 1.2695312534560799e-06 A
The gate voltage is set to 0.698533833026886 V
The measured source current is 3.979492248618044e-06 A
The gate voltage is set to 0.798827052116394 V
The measured source current is 7.885741979407612e-06 A
The gate voltage is set to 0.8991203308105469 V
The measured source current is 1.1669922059809323e-05 A
The gate voltage is set to 0.9994136095046997 V
The measured source current is 2.167968705180101e-05 A
The gate voltage is set to 1.0997068881988525 V
The measured source current is 3.0517578125e-05 A
The gate voltage is set to 1.1982406377792358 V
The measured source current is 3.693847611430101e-05 A
The gate voltage is set to 1.2985339164733887 V
The measured source current is 4.9633788876235485e-05 A
The gate voltage is set to 1.3988271951675415 V
The measured source current is 6.120605394244194e-05 A
The gate voltage is set to 1.4991203546524048 V
The measured source current is 7.770996307954192e-05 A
The gate voltage is set to 1.5994136333465576 V
The measured source current is 8.916015940485522e-05 A
The gate voltage is set to 1.6997069120407104 V
The measured source current is 9.990234684664756e-05 A
```

In [ ]:

```
# plot
Vg = np.arange(0.0,1.8,0.1)
plt.plot(Vg,IDS,"o-")
plt.grid()
plt.title("Ids vs Vg")
plt.xlabel("Vg [V]")
plt.ylabel("Ids [A]")
```

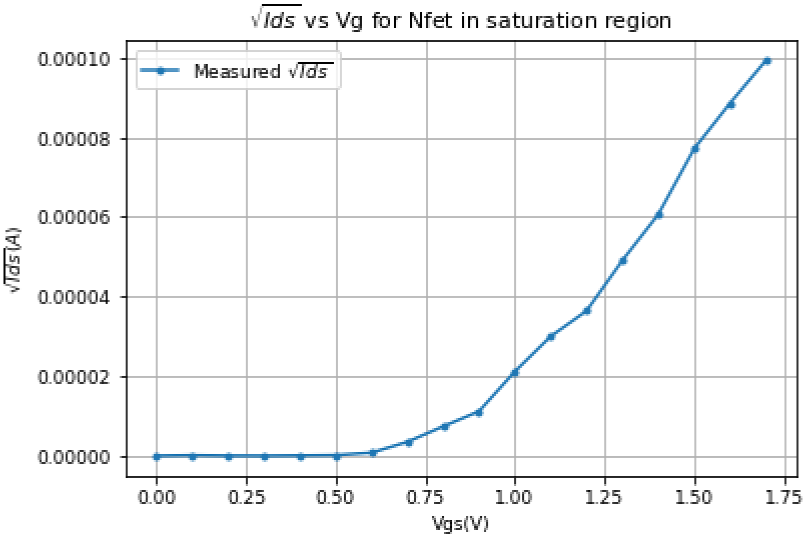
Out[ ]:

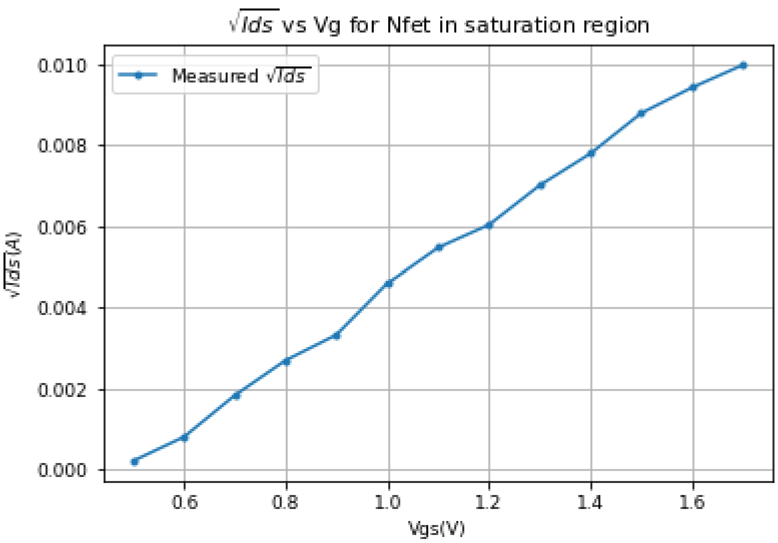
```
Text(0, 0.5, 'Ids [A]')
```



In [ ]:
```
# if the data looks nice, save it!
data = [Vg,IDS]
np.savetxt("data_nfet_6.1.csv",data,delimiter=",")
```

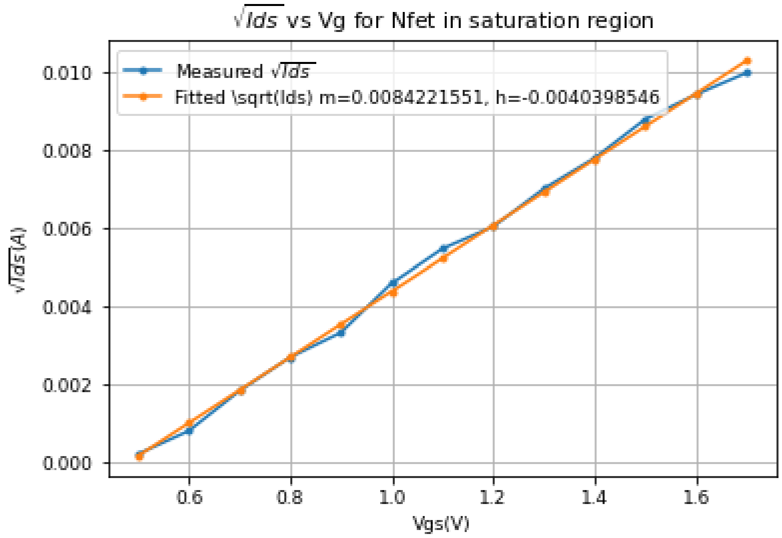
In [ ]:
```
# extract the valid range and plot sqrt(Ids) vs Vgs
Vg_save,Ids_save = np.loadtxt("data_nfet_6.1.csv",delimiter = ",")


plt.rcParams.update({'font.size': 9})
plt.plot(Vg_save, Ids_save, '.-',label=r"Measured $\sqrt{Ids}$")
plt.xlabel('Vgs(V)')
plt.ylabel(r'$\sqrt{Ids}(A)$')
plt.title(r"$\sqrt{Ids}$ vs Vg for Nfet in saturation region")
plt.legend()
plt.grid()
plt.show()

Ids_valid = Ids_save[5:]
Vg_valid = Vg_save[5:]

plt.rcParams.update({'font.size': 9})
plt.plot(Vg_valid, np.sqrt(Ids_valid), '.-',label=r"Measured $\sqrt{Ids}$")
plt.xlabel('Vgs(V)')
plt.ylabel(r'$\sqrt{Ids}(A)$')
plt.title(r"$\sqrt{Ids}$ vs Vg for Nfet in saturation region")
plt.legend()
plt.grid()
plt.show()
```





In [ ]:

```
# fit in the valid range (you may want to go back and add the fitted line in the plot)

#We can use polyfit to fit a regression line to our data
coffs = np.polyfit(Vg_valid,np.sqrt(Ids_valid),1)
Ids_fit = []
for i in Vg_valid:
    Ids_fit.append(coffs[0]*i + coffs[1])

plt.rcParams.update({'font.size': 9})
plt.plot(Vg_valid, np.sqrt(Ids_valid), '.-',label=r"Measured $\sqrt{Ids}$")
plt.plot(Vg_valid, Ids_fit, '.-',label=f"Fitted \sqrt(Ids) m={np.round(coffs[0],10)},
plt.xlabel('Vgs(V)')
plt.ylabel(r'$\sqrt{Ids}(A)$')
plt.title(r"$\sqrt{Ids}$ vs Vg for Nfet in saturation region")
plt.legend()
plt.grid()
plt.show()
```



(c) Determine V_{T0} and β for both devices by fitting your data to the expression derived in the prelab

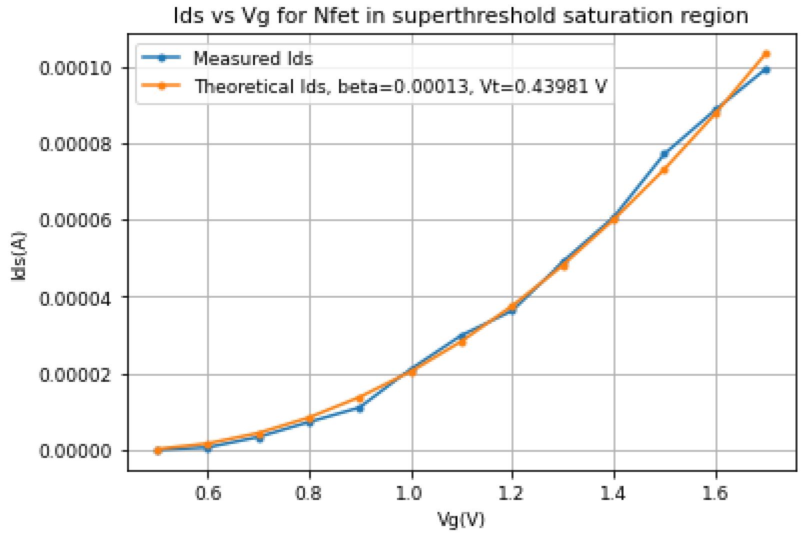
In [ ]:
```
# V_T0
from scipy.optimize import curve_fit
#define function
def Ids_nFET_func(Vg,beta,Vt):
        Vs = 0.0
        Vd = 1.8
        return 0.5*beta*(Vg-Vs-Vt)**2
#fit curve
popt, pcov = curve_fit(Ids_nFET_func,Vg_valid,Ids_valid,p0=[1, 2e-5])
#get parameters
fit_beta,fit_vt = popt
#compute theoret. data with fitted parameters
fitted_y = Ids_nFET_func(Vg_valid,fit_beta,fit_vt)
#print result
v_t0 = fit_vt
print('Fitted Vt0: ',v_t0)
```

```
Fitted Vt0:  0.4398106769084099
```

In [ ]:
```
# beta
betan = fit_beta
print('beta = ',betan)
```

```
beta =  0.00013010032550624882
```

In [ ]:
```
Ids_theo = [0.5*fit_beta*(i-0-fit_vt)**2 for i in Vg_valid]

plt.plot(Vg_valid,Ids_valid,'.-',label="Measured Ids")
plt.plot(Vg_valid,Ids_theo,'.-',label=f"Theoretical Ids, beta={np.round(fit_beta,5)},
plt.legend()
plt.xlabel('Vg(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vg for Nfet in superthreshold saturation region''')
plt.grid()
plt.show()
```



6.2 P-FET

(a) Configure the chip following Section 4.3 if you haven't

(b) Measure I_{ds} as a function of V_g in ohmic region

- What will be the fixed value for bulk, source and drain voltages?

In [ ]:
```
## configure PMOS by AER event
# Configure PFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the PFET
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)
```

In [ ]:
```
#set bulk voltage
p.set_voltage(pyplane.DacChannel.AIN1,1.8)

time.sleep(0.05)  # wait for it to settle

# set source voltage
p.set_voltage(pyplane.DacChannel.GO23,1.8)

# set drain voltage
p.set_voltage(pyplane.DacChannel.GO21,0)
# Print I_ds for checking
print(p.read_current(pyplane.AdcChannel.GO21_N))
```

```
5.126952942191565e-07
```

- Data aquisition

In [ ]:
```
# sweep gate voltage
```

```
# sweep gate voltage
import time
import numpy as np

# Get the leakage current, Read Ids=Ids0 at Vg = 0
p.set_voltage(pyplane.DacChannel.AIN0,1.8)
time.sleep(0.5) # wait 0.5 second for it to settle
Is0_n = p.read_current(pyplane.AdcChannel.GO21_N) #REMEMBER: reading from source is pi
print("Offset Is0_n: {} A".format(Is0_n))

IDS = []
read_Ids = 0
for gate in np.arange(0,1.8,0.1) :
    # set gate voltage
    p.set_voltage(pyplane.DacChannel.AIN0,gate)

    print("The gate voltage is set to {} V".format(p.get_set_voltage(pyplane.DacChanne

    time.sleep(0.05)  # wait for it to settle
    # read I_{ds}
    read_Ids = p.read_current(pyplane.AdcChannel.GO21_N)


    print("The measured source current is {} A".format(read_Ids))  ## print the raw da

    # substract leakage current
    IDS.append(read_Ids-Is0_n)
```

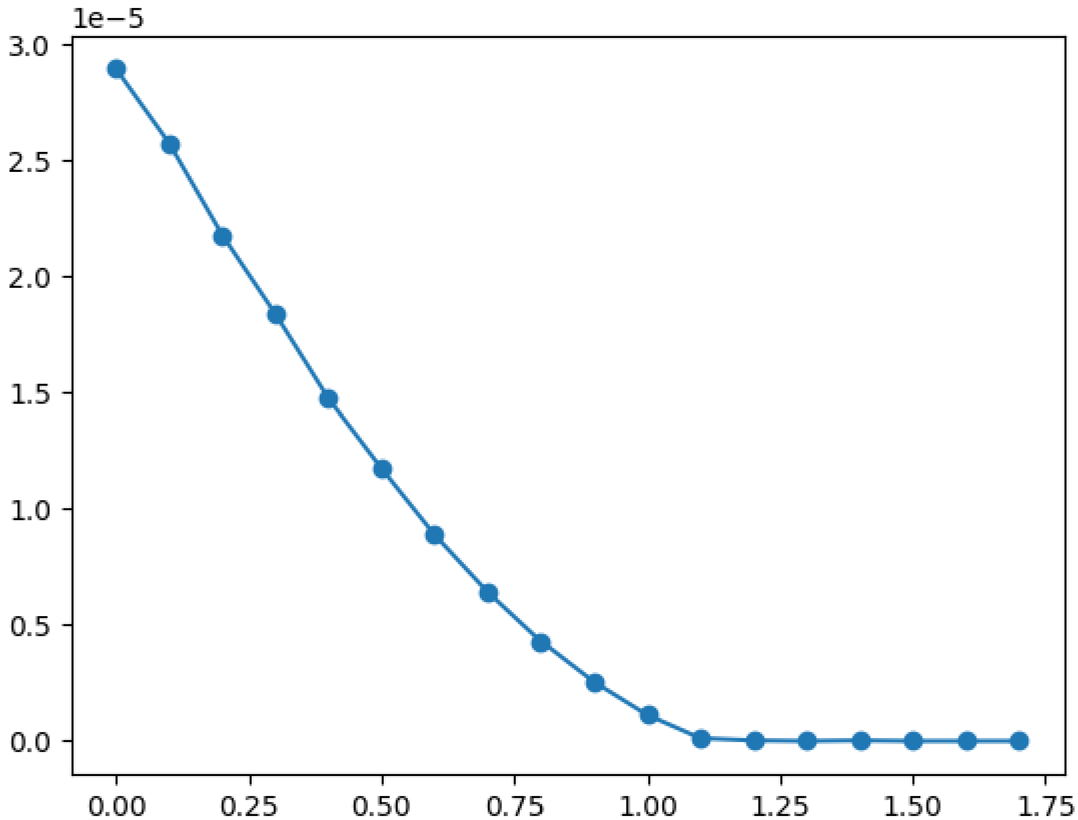
```
Offset Is0_n: 5.126952942191565e-07 A
The gate voltage is set to 0.0 V
The measured source current is 3.0297851481009275e-05 A
The gate voltage is set to 0.0985337346792221 V
The measured source current is 2.5854491468635388e-05 A
The gate voltage is set to 0.19882699847221375 V
The measured source current is 2.1826172087457962e-05 A
The gate voltage is set to 0.2991202771663666 V
The measured source current is 1.860351585492026e-05 A
The gate voltage is set to 0.399413526058197 V
The measured source current is 1.511230493633775e-05 A
The gate voltage is set to 0.49970680475234985 V
The measured source current is 1.2280273040232714e-05 A
The gate voltage is set to 0.5982405543327332 V
The measured source current is 9.545898137730546e-06 A
The gate voltage is set to 0.698533833026886 V
The measured source current is 6.787109214201337e-06 A
The gate voltage is set to 0.798827052116394 V
The measured source current is 4.809570327779511e-06 A
The gate voltage is set to 0.8991203308105469 V
The measured source current is 3.0761718790017767e-06 A
The gate voltage is set to 0.9994136095046997 V
The measured source current is 1.6113281162688509e-06 A
The gate voltage is set to 1.0997068881988525 V
The measured source current is 6.103515488575795e-07 A
The gate voltage is set to 1.1982406377792358 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 1.2985339164733887 V
The measured source current is 5.37109372089617e-07 A
The gate voltage is set to 1.3988271951675415 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 1.4991203546524048 V
The measured source current is 4.882812731921149e-07 A
The gate voltage is set to 1.5994136333465576 V
The measured source current is 5.126952942191565e-07 A
The gate voltage is set to 1.6997069120407104 V
The measured source current is 5.37109372089617e-07 A
```

In [ ]:

```
# plot
Vg = np.arange(0,1.8,0.1)
plt.plot(Vg,IDS,"o-")
```

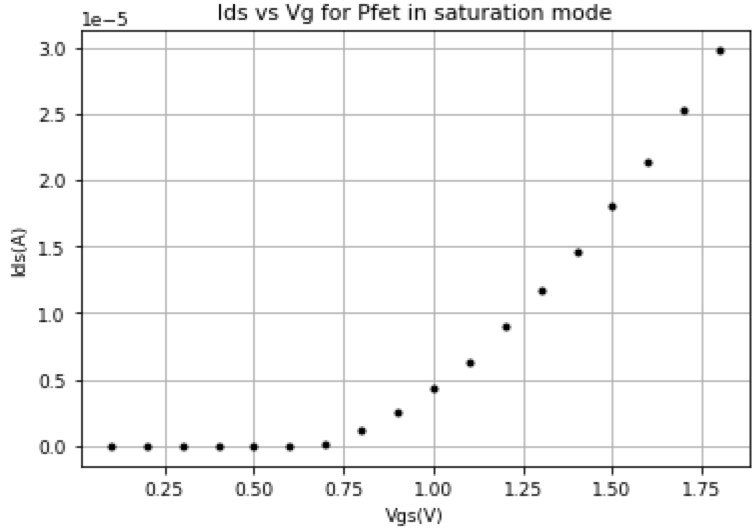
Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f1fd71e55b0>]
```

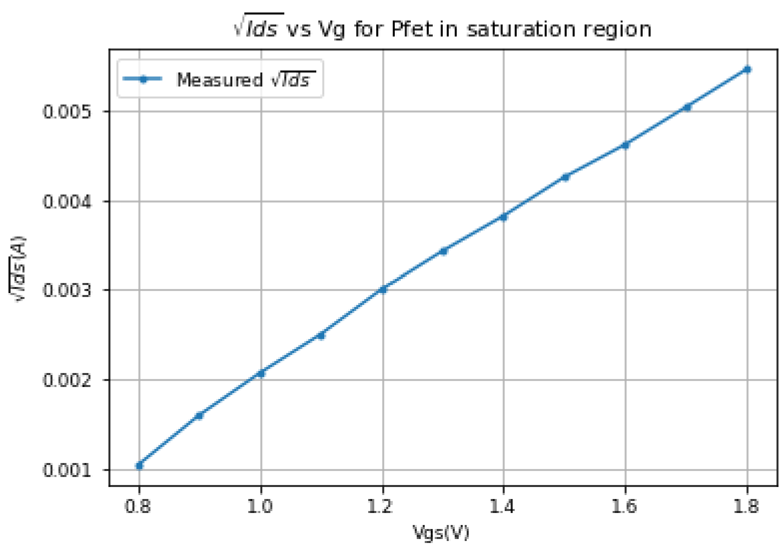


In [ ]:
```
# if the data looks nice, save it!
data = [Vg,IDS]
np.savetxt("data_pfet_62.csv",data,delimiter = ",")
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
#load data and plot it
Vg_save , Isn_save = np.loadtxt("data_pfet_62.csv",delimiter = ",")
Vgs_save = 1.8-Vg_save

Vgs_save.sort()
Isn_save.sort()

plt.rcParams.update({'font.size': 9})
plt.plot(Vgs_save, Isn_save, '.k')
plt.xlabel('Vgs(V)')
plt.ylabel('Ids(A)')
plt.title("Ids vs Vg for Pfet in saturation mode")
plt.grid()
plt.show()
```

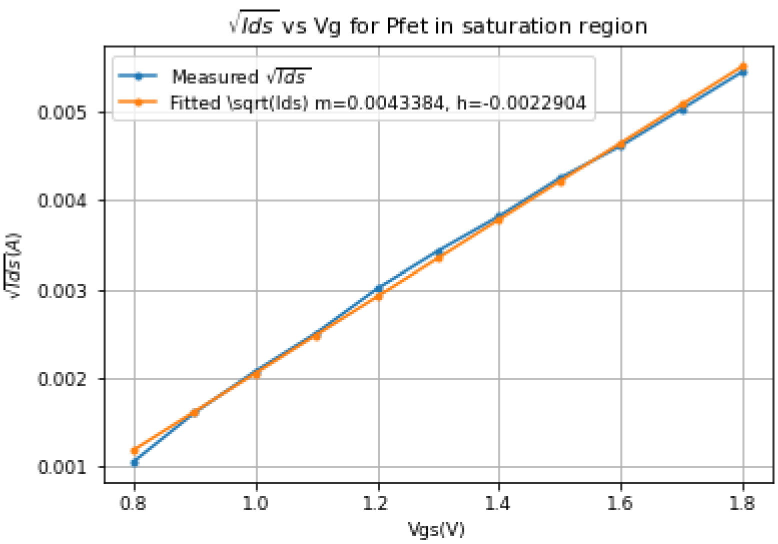


In [ ]:
```
# extract the valid range and plot sqrt(Ids) vs Vgs

Ids_valid = Isn_save[7:]
Vgs_valid = Vgs_save[7:]

plt.rcParams.update({'font.size': 9})
plt.plot(Vgs_valid, np.sqrt(Ids_valid), '.-',label=r"Measured $\sqrt{Ids}$")
plt.xlabel('Vgs(V)')
plt.ylabel(r'$\sqrt{Ids}(A)$')
plt.title(r"$\sqrt{Ids}$ vs Vg for Pfet in saturation region")
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
# fit in the valid range (you may want to go back and add the fitted line in the plot)
#We can use polyfit to fit a regression line to our data
coffs = np.polyfit(Vgs_valid,np.sqrt(Ids_valid),1)
Ids_fit = []
for i in Vgs_valid:
    Ids_fit.append(coffs[0]*i + coffs[1])

plt.rcParams.update({'font.size': 9})
plt.plot(Vgs_valid, np.sqrt(Ids_valid), '.-',label=r"Measured $\sqrt{Ids}$")
```

```
plt.plot(Vgs_valid, Ids_fit, '.-',label=f"Fitted \sqrt(Ids) m={np.round(coffs[0],7)},
plt.xlabel('Vgs(V)')
plt.ylabel(r'$\sqrt{Ids}(A)$')
plt.title(r"$\sqrt{Ids}$ vs Vg for Pfet in saturation region")
plt.legend()
plt.grid()
plt.show()
```



(c) Determine V_{T0} and β for both devices by fitting your data to the expression derived in the prelab

In [ ]:

```
# V_T0
from scipy.optimize import curve_fit
#define function (formula is the same as for nfet)
def Ids_nFET_func(Vgs,beta,Vt):
        Vs = 1.8
        Vd = 0.0
        return 0.5*beta*(Vgs-Vt)**2
#fit curve
popt, pcov = curve_fit(Ids_nFET_func,Vgs_valid,Ids_valid,p0=[1, 0.5e-5])
#get parameters
fit_beta,fit_vt = popt
#compute theoret. data with fitted parameters
fitted_y = Ids_nFET_func(Vgs_valid,fit_beta,fit_vt)
#print result
print('Fitted Vt0: ',1.8-fit_vt)
```

```
Fitted Vt0:  1.303970054219272
```

In [ ]:

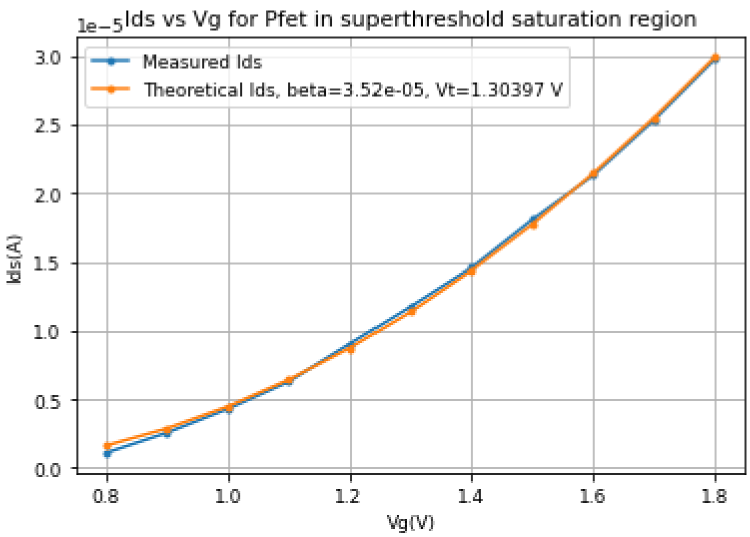
```
# beta from scipy
betan = fit_beta
print('beta = ',betan)
#FROM THE SLOPE
print('beta2:',2*coffs[0]**2)
```

```
beta =  3.521984209152126e-05
beta2: 3.7643775834363996e-05
```

In [ ]:

```
Ids_theo = [0.5*fit_beta*(i-fit_vt)**2 for i in Vgs_valid]

plt.plot(Vgs_valid,Ids_valid,'.-',label="Measured Ids")
```

```
plt.plot(Vgs_valid,Ids_theo,'.-',label=f"Theoretical Ids, beta={np.round(fit_beta,7)},
plt.legend()
plt.xlabel('Vg(V)')
plt.ylabel('Ids(A)')
plt.title('''Ids vs Vg for Pfet in superthreshold saturation region''')
plt.grid()
plt.show()
```



6.3 Comparisons

- Are the measurements of V_{T0} and β from the saturation measurement consistent with the values obtained in the ohmic region?

Vt0 seems to vary slightly more than the values obtained in the ohmic region for pfets.

Parameter	Ohmic	Saturation
β	3.721e-05	3.764e-05
$Vt0$	0.837 V	1.303 V

And for nfet the results seem to be pretty consistent

Parameter	Ohmic	Saturation
β	0.00017	0.00013
$Vt0$	0.569 V	0.439 V

- Which is a better approximation, the linear one or the quadratic?

The quadratic curves seem to fit better the measurements, although any error/variation should increase quadratically

7 Early effect

This experiment studies how Early voltage scales with transistor current; in particular, how valid are the simple assumptions about channel length modulation?

You only need to do N-FET

(a) Measure I_{ds} vs V_{ds} for different V_{gs}

In [ ]:
```
### AER to configure NMOS
# Configure NFET, set the input voltage demultiplexer by AER event.
# Note selectlines we should choose for the NFET
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)
```

In [ ]:
```
# set source voltage
Vs = 0
p.set_voltage(pyplane.DacChannel.GO20,Vs)

Vd = 1.8
p.set_voltage(pyplane.DacChannel.GO22, Vd)
```

Out[ ]:
```
1.7982406616210938
```

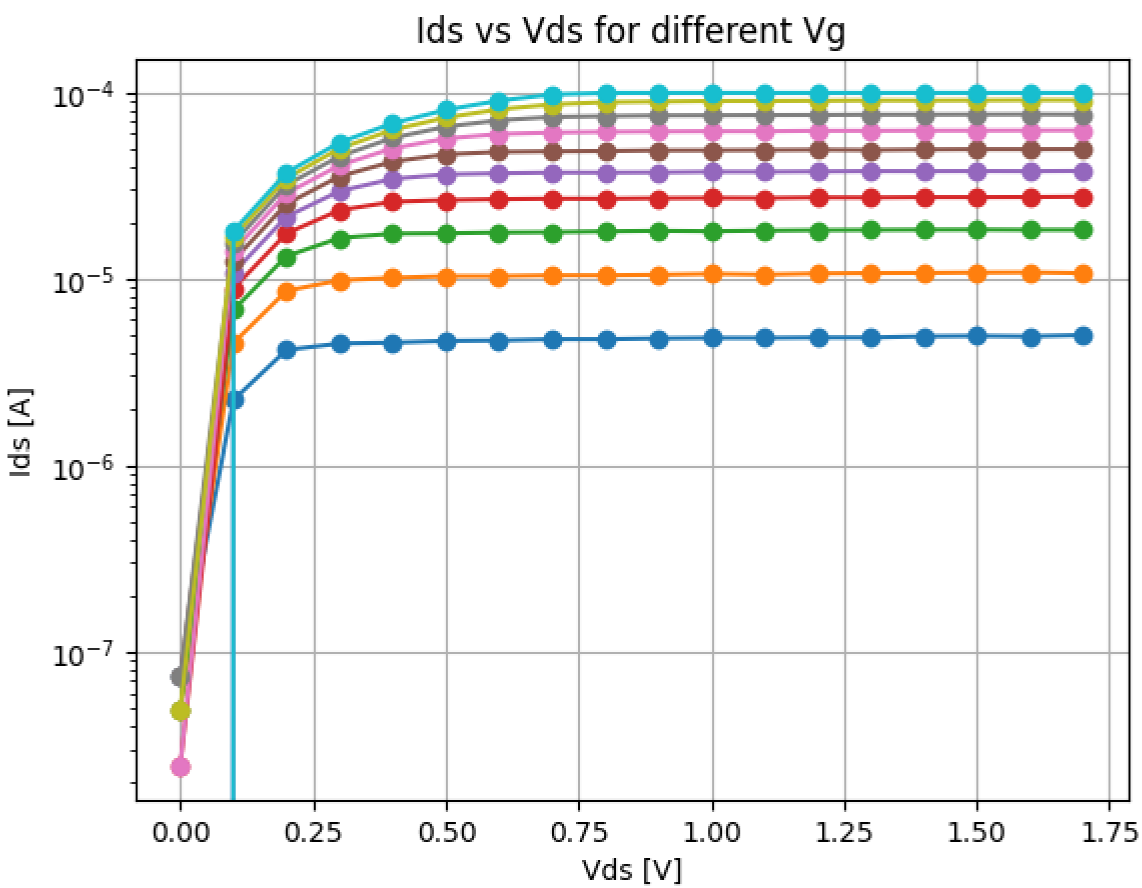
In [ ]:
```
import numpy as np

# Measurement. You may need two 'for' loops (one nested loop) to sweep Vgs and Vds
Vg_sweep = np.arange(0.8,1.8,0.1)
Vd_sweep = np.arange(0,1.8,0.1)

#initialize dataframe for storing IDS
IDS = {}
for i in Vg_sweep:
    IDS[i] = []

# Get the leakage current, Read Ids=Ids0 at Vg = 0
p.set_voltage(pyplane.DacChannel.AIN0,0)
time.sleep(0.5) # wait 0.5 second for it to settle
Is0_n = p.read_current(pyplane.AdcChannel.GO20_N) #REMEMBER: reading from source is pi
print("Offset Is0_n: {} A".format(Is0_n))

for Vg_set in Vg_sweep:
    p.set_voltage(pyplane.DacChannel.AIN0,Vg_set)
    for Vd_set in Vd_sweep:
        p.set_voltage(pyplane.DacChannel.GO22,Vd_set)
        time.sleep(0.05)
        read_IDS = p.read_current(pyplane.AdcChannel.GO20_N)
        IDS[Vg_set].append(read_IDS-Is0_n)
```

```
Offset Is0_n: 8.300781360048859e-07 A
```

- Include a single plot showing all data on a semilogy plot.

In [ ]:
```
import matplotlib.pyplot as plt
# plot
for i in IDS:
    plt.plot(Vd_sweep,IDS[i],"o-")
    plt.semilogy()
plt.grid()
plt.title("Ids vs Vds for different Vg")
plt.xlabel("Vds [V]")
plt.ylabel("Ids [A]")
```

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



In [ ]:
```
# if the data looks nice, save it!
for i in Vg_sweep:
    data = [Vd_sweep,IDS[i]]
    np.savetxt(f"data_early_effect_VG{i}.csv",data,delimiter = ",")
```

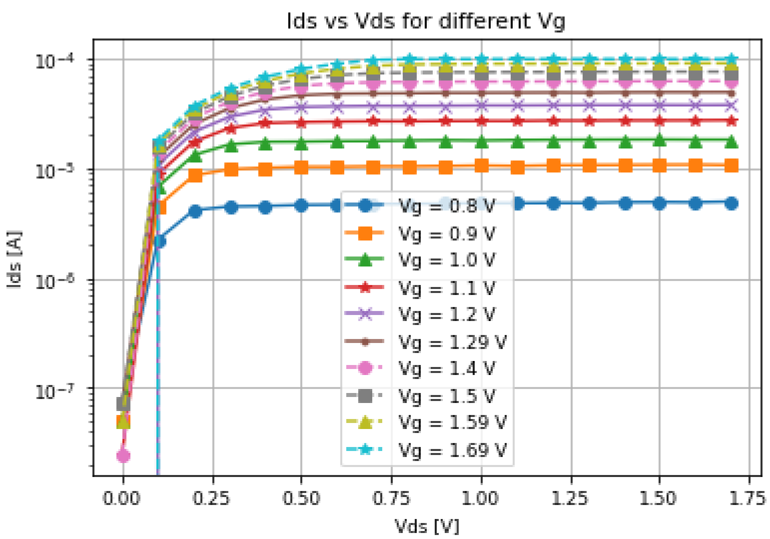
- Can you see how the saturation voltage increases with the gate overdrive $V_G - V_T$ in strong inversion?

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

Vg_sweep = [0.8,0.9,1.0,1.1,1.2,1.29,1.4,1.5,1.59,1.69]
Vd_sweep = np.arange(0,1.8,0.1)
```

```
#re initialize dictionnary for storing IDS
bin = []
IDS = {}
for i in Vg_sweep:
    IDS[i] = []

for Vg_set in Vg_sweep:
    bin, IDS[Vg_set] = np.loadtxt(f"data_early_effect_VG{Vg_set}.csv",delimiter=",")

#plot
styles = ["o-", "s-", "^-", "*-", "x-",".-","o--", "s--", "^--", "*--", "x--",".--"]
for i,j,style in zip(IDS,Vg_sweep,styles):
    plt.plot(Vd_sweep,IDS[i],style,label=f"Vg = {j} V")
    plt.semilogy()
plt.grid()
plt.legend()
plt.title("Ids vs Vds for different Vg")
plt.xlabel("Vds [V]")
plt.ylabel("Ids [A]")
```

Out[ ]:

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



We can see that for a same Vt, the saturation current increases with Vg (and the saturation voltage as well)

In [ ]:

(b) Compute the Early voltage

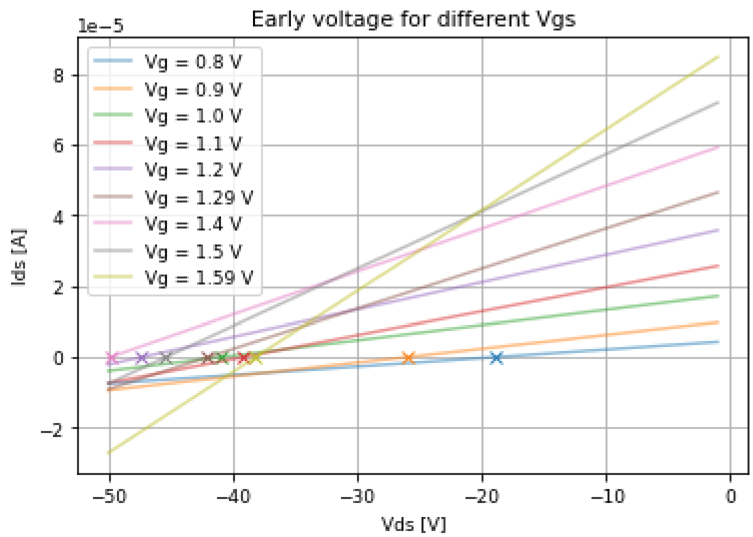
- Fit a line to the "flat" part of each curve. Select a range of drain voltages to fit the line and use the same range for each curve, because the Early effect is actually curved in reality, and what you are actually seeing is the start of Drain Induced Barrier Lowering (DIBL) or impact ionization.

In [ ]:

```
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2',
early_voltages = []
for i,j,c in zip(IDS,Vg_sweep,colors):
    #fit line
```

```
    coffs = np.polyfit(Vd_sweep[8:],IDS[i][8:],1)
    line = []
    #calculate early voltage from estimated slope and intercept
    m,h = coffs
    V_early = -h/m
    early_voltages.append(V_early)
    #generate line
    for i in np.arange(-50,0):
        line.append(coffs[0]*i + coffs[1])
    #plot
    #plt.plot(Vd_sweep[8:],IDS[i][8:],".",label=f"Vg = {j} V")
    if V_early > -500:      #remove excessively high value
        plt.plot(V_early,0,"x",color = c)
        plt.plot(np.arange(-50,0),line,"-",color = c,label=f"Vg = {j} V",alpha=0.5)


plt.grid()
plt.legend()
plt.title('Early voltage for different Vgs')
plt.xlabel('''Vds [V]

The lines represent a regression fit to
the flat saturation part of Ids vs Vgs curve,
the crosses represent the intersection of the
extrapolated line with the x-axis (the early voltage) ''')
plt.ylabel("Ids [A]")
```

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



The lines represent a regression fit to
the flat saturation part of Ids vs Vgs curve,
the crosses represent the intersection of the
extrapolated line with the x-axis (the early voltage)

- Plot the Early voltage vs drain current on a semilogx scale.

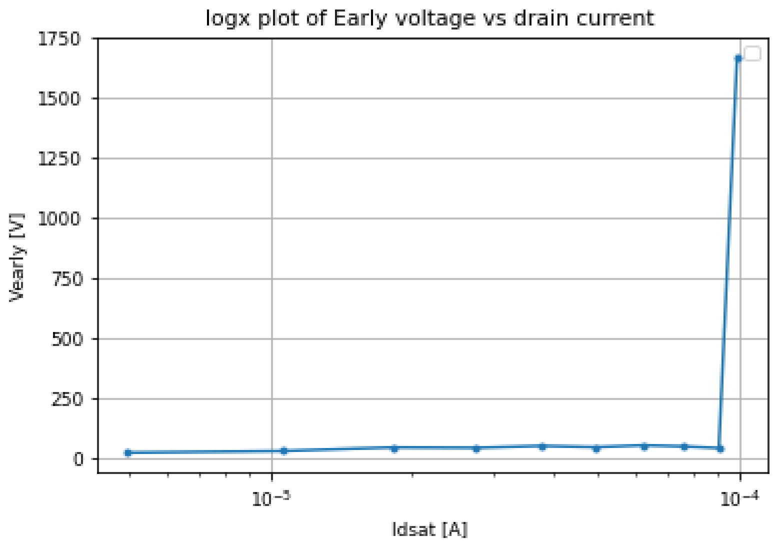
In [ ]:
```
sat_currents = []
for i ,v in zip(IDS,early_voltages):
        sat_currents.append(IDS[i][-1])

plt.semilogx(sat_currents,np.abs(early_voltages),".-")
plt.grid()
plt.legend()
```

```
plt.title('logx plot of Early voltage vs drain current ')
plt.xlabel('''Idsat [A] ''')
plt.ylabel("Vearly [V]")
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

Out[ ]:

```
Text(0, 0.5, 'Vearly [V]')
```



Removing the excessively high value at the end we get

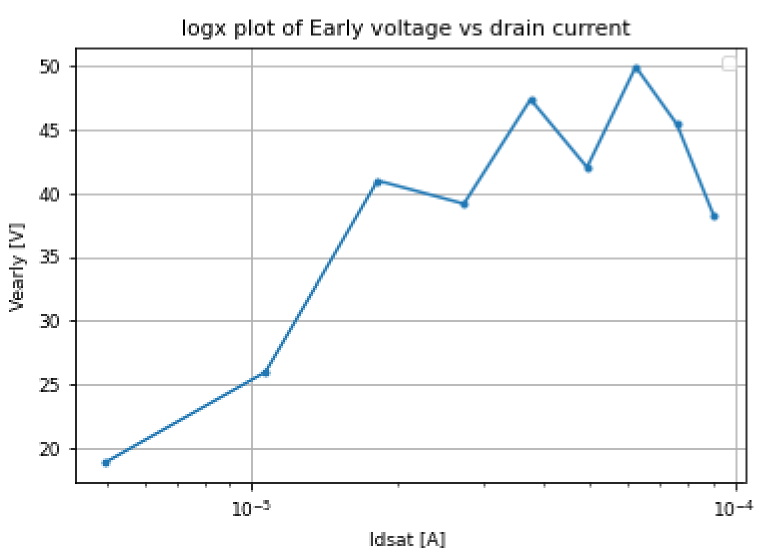
In [ ]:

```
plt.semilogx(sat_currents[:-1],np.abs(early_voltages[:-1]),".-")
plt.grid()
plt.legend()
plt.title('logx plot of Early voltage vs drain current ')
plt.xlabel('''Idsat [A] ''')
plt.ylabel("Vearly [V]")
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

Out[ ]:

```
Text(0, 0.5, 'Vearly [V]')
```



- Comment on your results: How constant is the Early voltage with drain current? Speculate on the reasons for your observations.

The early voltage fluctuates but remains in the same order of magnitude. It is inversely proportional to the slope of the saturation region of the measured current (Ids). We can see that the last two points seem to decrease. Maybe this being due to the fact that the electrons are in free flow.

8 Congratulations

If you did everything in this lab, you have done a lot! This is probably the most difficult but also one of the most important labs, because practical and intuitive knowledge of transistor characteristics is crucial in understanding and synthesizing new circuits.

9 What we expect

How transistors work above threshold.

What is the linear or triode region and what is the saturation region?

How does the linear region depend on gate and threshold voltage?

What is the *overdrive*?

What is the specific current?

How the Early effect comes about?

Typical values for Early voltage.

How to sketch graphs of transistor current vs gate voltage and drain-source voltage.

How above-threshold transistors go into saturation and why the saturation voltage is equal to the gate overdrive. Can you write the above-threshold current equations?

How does above-threshold current depend on W/L, C_{ox}, and mobility μ?

How do transconductance and drain resistance combine to generate voltage gain? And what is the intrinsic voltage gain of a transistor?

What effect does velocity saturation have on transistor operation, specifically, how does it change the relation between saturation current and gate voltage? What is DIBL (drain induced barrier lowering) and II (impact ionization)?

What is the dominant source of mismatch?

How does transistor mismatch scale with transistor size?

What are typical values of transistor threshold voltage mismatch?