Lab 3: Subthreshold Behavior of Transistors

Group number: 4.5

Team member 1: Jan Hohenheim

Team member 2: Maxim Gärtner

Date:

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.

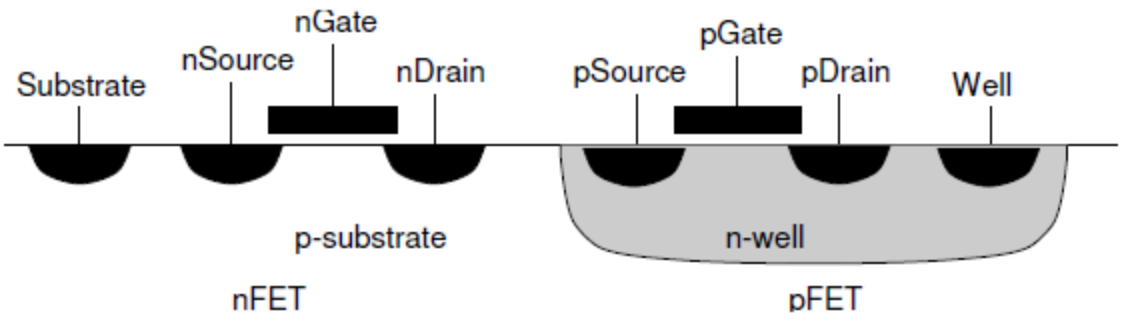
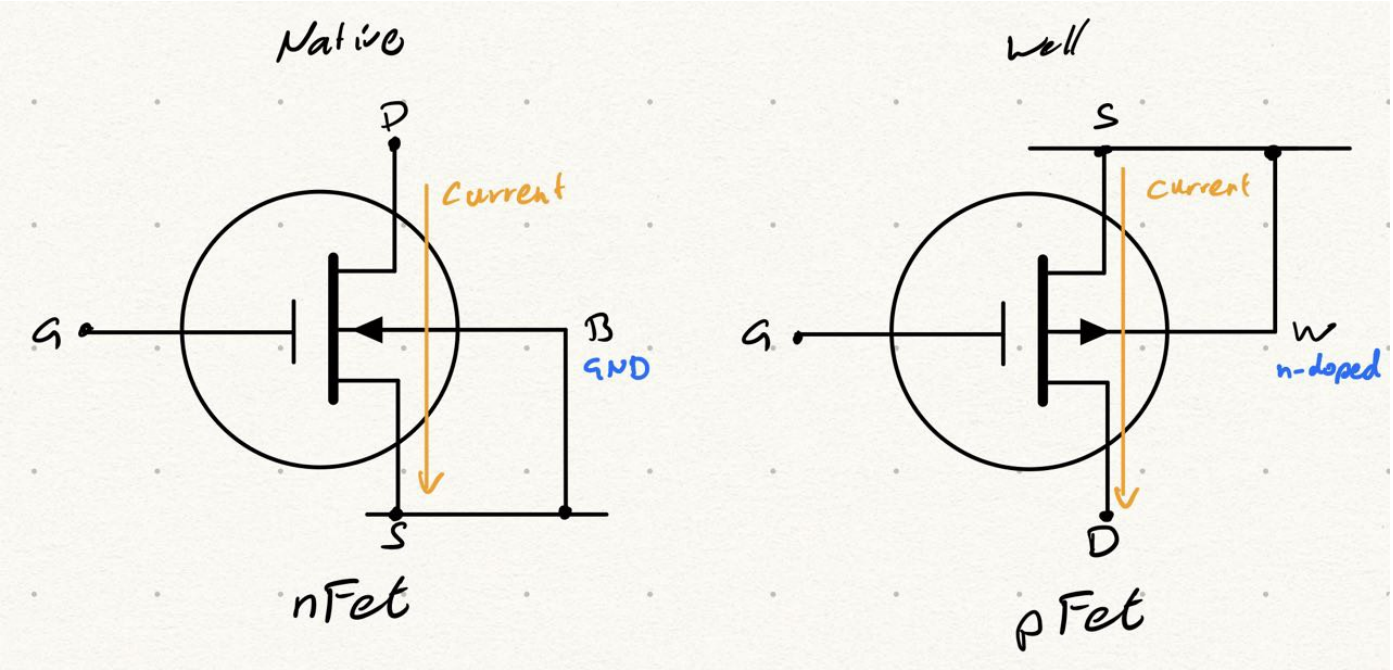


Figure 1: a cross section through p-substrate chip.

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 (in LaTeX) the expressions for the subthreshold (weak inversion) current I_{ds} for both types of transistors.

> nFET: $I = I_0 e^{\kappa \frac{V_g}{U_T}} (e^{-\frac{V_s}{U_T}} - e^{-\frac{V_{ds}}{U_T}})$
> pFET: $I = I_0 e^{-\kappa \frac{V_g}{U_T}} (e^{\frac{V_s}{U_T}} - e^{-\frac{V_{ds}}{U_T}})$

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.

> nFET: $I = I_0 e^{\frac{\kappa V_g - V_s}{U_T}}$
> pFET: $I = I_0 e^{\frac{-\kappa V_g + V_s}{U_T}}$

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 $\frac{\mathrm{d}V_s}{\mathrm{d}V_g}$?

> nFET:
>
> - $V_s = \kappa V_g + U_T \ln \frac{I_0}{I}$
> - $\frac{dV_s}{dV_g} = \kappa$
>
> pFET:
>
> - $V_s = \kappa V_g + U_T \ln \frac{I}{I_0}$
> - $\frac{dV_s}{dV_g} = \kappa$

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; is the reader supposed to guess?
- **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 [1]:
```
# import the necessary library to communicate with the hardware
import pyplane
import time
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# 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 opera
# then you may get error messages with open(...ttyACM0). So please avoid frenquently plugg
```

In [3]:
```
p.get_firmware_version()
```

Out[3]:
```
(1, 8, 3)
```

In [4]:
```
# 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, otherwise t
# Because the class chip need to do handshake to get the communication correct.
p.request_events(1)
```

In [5]:
```
# Try to read something, make sure the chip responses
p.read_current(pyplane.AdcChannel.GO0_N)
```

Out[5]:
```
1.8530273848682555e-07
```

In [6]:
```
# 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**

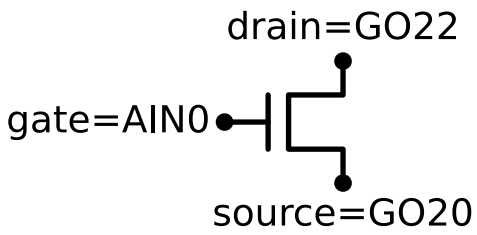
3 Current as a Function of Gate Voltage

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 [7]:
```
# 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[7]:



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 [20]:
```
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 [33]:
```
# set source voltage
vs_n = 0.0
p.set_voltage(pyplane.DacChannel.GO20, vs_n)
print(f"The source voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO20)} V")
```

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

In [34]:
```
# set drain voltage
vd_n = 1.8
p.set_voltage(pyplane.DacChannel.GO22, vd_n)
print(f"The drain voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO22)} V")
```

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

In [35]:
```
# set trial gate voltage
vg_n = 0.0
p.set_voltage(pyplane.DacChannel.AIN0, vg_n)

print(f"The trial gate voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN0)} V")
```

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

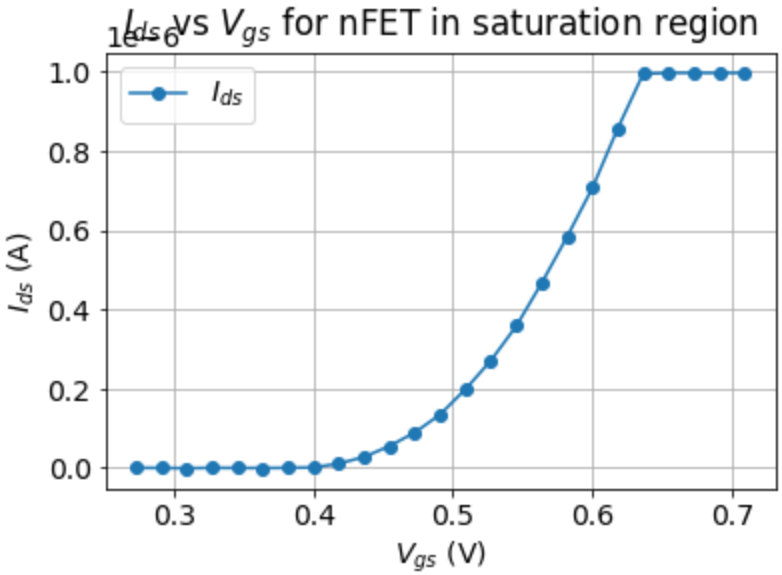
In [36]:
```
# read Ids, from *Source*
Ids_0 = p.read_current(pyplane.AdcChannel.GO20_N)
print(f"The Ids_0 is {Ids_0} A")
```

```
The Ids_0 is 2.4414064103694955e-09 A
```

- Data aquisition

In [37]:
```
# sweep gate voltage
Vgs = np.linspace(0, 1.8, num=100)
Ids = []
for vg_n in Vgs:
    p.set_voltage(pyplane.DacChannel.AIN0, vg_n)
    time.sleep(0.05)
    Id = p.read_current(pyplane.AdcChannel.GO20_N) - Ids_0
    Ids.append(Id)
```

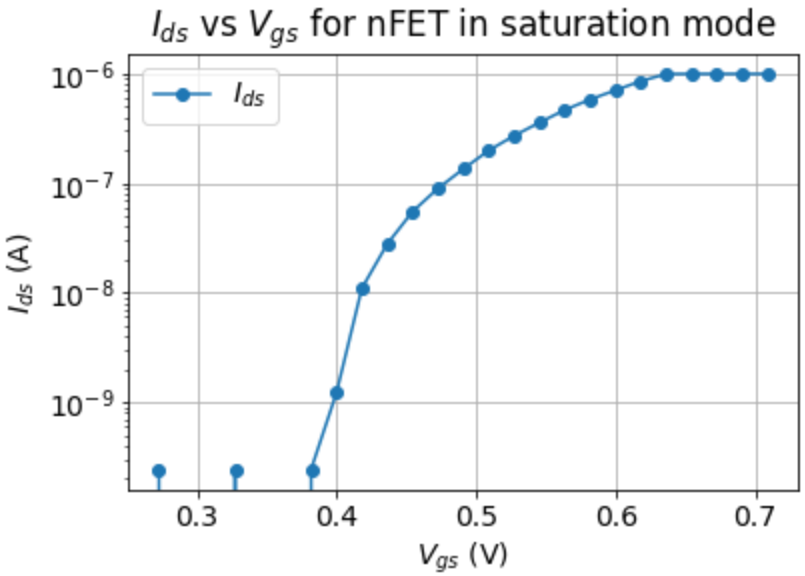
In [75]:
```
# plot in linear scale
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 14})
plt.plot(Vgs[15:40], Ids[15:40], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for nFET in saturation region")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
```



In [40]:
```
# if it looks nice in the plot, save it!

np.savetxt("3_1.csv", [Vgs, Ids], delimiter=",")
```

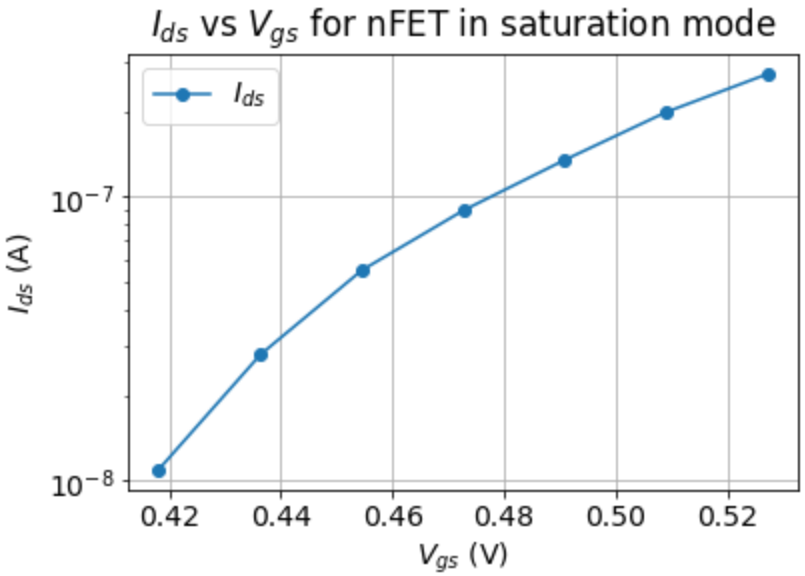
In [15]:
```
# Load data you saved and plot, to check if the data is saved correctly
[Vgs, Ids] = np.loadtxt("3_1.csv", delimiter=",")
```

In [77]:
```
# plot in logarithmic scale
plt.semilogy(Vgs[15:40], Ids[15:40], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for nFET in saturation mode")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
print(f"Arbitrary Vg close to threshold: {Vgs[27]}")
print(f"Arbitrary Ids close to threshold: {Ids[27]}")
```




```
Arbitrary Vg close to threshold: 0.4909090909090909
Arbitrary Ids close to threshold: 1.3500976048241853e-07
```

In [83]:
```
# extract the valid range and plot in logarithmic scale
plt.semilogy(Vgs[23:30], Ids[23:30], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for nFET in saturation mode")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
print(f"Arbitrary Vg close to threshold: {Vgs[27]}")
print(f"Arbitrary Ids close to threshold: {Ids[27]}")
```

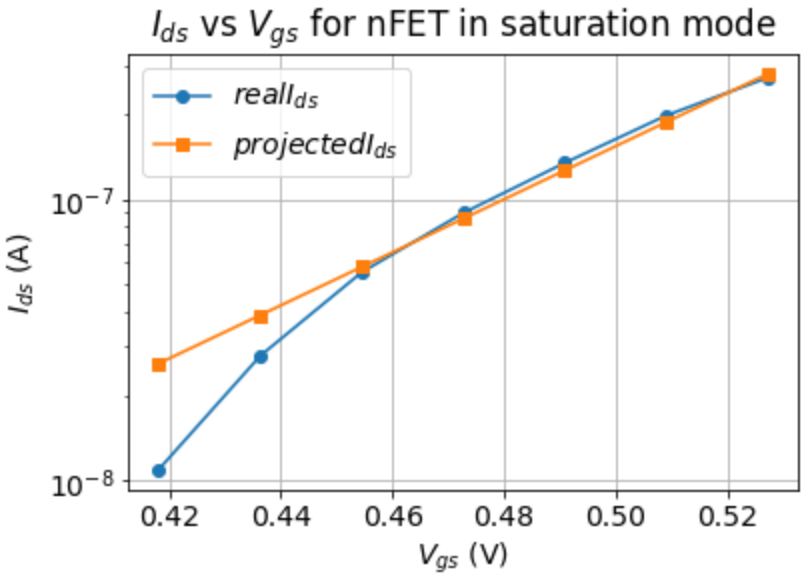



```
Arbitrary Vg close to threshold: 0.4909090909090909
Arbitrary Ids close to threshold: 1.3500976048241853e-07
```

> nFET: $I = I_0 e^{\frac{\kappa V_g - V_s}{U_T}}$
>
> pFET: $I = I_0 e^{\frac{-\kappa V_g + V_s}{U_T}}$

In [89]:

```
# fit in the valid range (you may want to add the fitted line in the plot)
from scipy.optimize import curve_fit

valid_Vgs = Vgs[23:30]
valid_Ids = Ids[23:30]
def get_Ids(Vgs, kappa, I0):
    Ut = 0.025
    return I0 * np.exp(kappa * Vgs/Ut)
popt, pcov = curve_fit(get_Ids, valid_Vgs, valid_Ids, p0=[0.79, 1e-10])
fit_kappa, fit_I0 = popt
y = get_Ids(valid_Vgs, fit_kappa, fit_I0)
plt.semilogy(valid_Vgs, valid_Ids, "o-", label="$real I_{ds}$")
plt.semilogy(valid_Vgs, y, "s-", label="$projected I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for nFET in saturation mode")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
```



- Extract I_0 and κ

In [63]:
```
# I_0

print(f"Fitted I_0: {fit_I0}")
```

```
Fitted I_0: 1.9484859249494753e-12
```

In [64]:
```
# kappa

print(f"Fitted kappa: {fit_kappa}")
```

```
Fitted kappa: 0.5634226704801054
```

- 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 [96]:
```
# compute threshold voltage
from scipy import interpolate
Th = None
extrapolated_Ids = None
actual_threshold_Ids = None
for Vg, Ids_ in zip(Vgs[26:], Ids[26:]):
    extrapolated_Ids = get_Ids(Vg, fit_kappa, fit_I0)
    if Ids_ < 0.5 * extrapolated_Ids:
        Th = Vg
        actual_threshold_Ids = Ids_
        break
print(f"Threshold voltage: {Th} V")
```

```
Threshold voltage: 0.6181818181818182 V
```

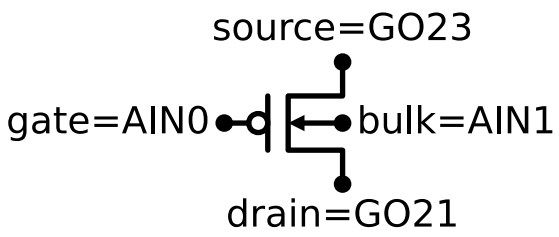
In [98]:
```
# compute Ids at threshold voltage
print(f"Extrapolated Ids at threshold: {extrapolated_Ids} A")
print(f"Actual Ids at threshold: {actual_threshold_Ids} A")
```

```
Extrapolated Ids at threshold: 2.0168452354139867e-06 A
Actual Ids at threshold: 8.566894578176942e-07 A
```

3.2 P-FET

In [35]:
```
# 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[35]:



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 [9]:
```
# Configure PMOS, set the demultiplexer
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 [19]:
```
# set bulk voltage
Vdd = 1.8
V_bulk = Vdd
p.set_voltage(pyplane.DacChannel.AIN1, V_bulk)
print(f"The bulk voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN1)} V")
time.sleep(0.1)
```

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

In [20]:
```
# set source voltage
V_sp = Vdd
p.set_voltage(pyplane.DacChannel.GO23, V_sp)
print(f"The source voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO23)} V")
```

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

In [21]:
```
# set drain voltage
V_dp = 0.0
p.set_voltage(pyplane.DacChannel.GO21, V_dp)
print(f"The drain voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO21)} V")
```

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

In [39]:
```
# set trial gate voltage
V_g0 = 1.8
p.set_voltage(pyplane.DacChannel.AIN0, V_g0)
print(f"The trial gate voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN0)} V")
```

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

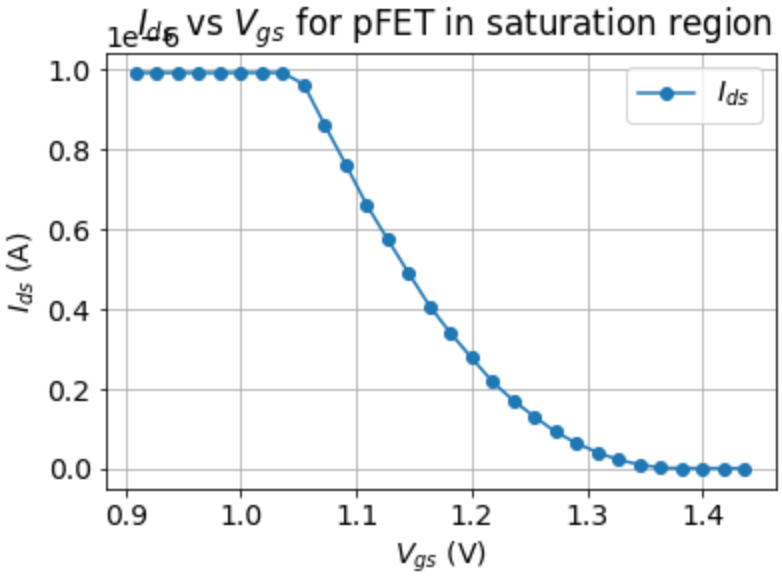
In [40]:
```
# read Ids
Ids_0 = p.read_current(pyplane.AdcChannel.GO21_N)
print(f"The Ids_0 is {Ids_0} A")
```

```
The Ids_0 is 6.347656178462557e-09 A
```

- Data aquisition

In [41]:
```
# sweep gate voltage
Vgs = np.linspace(0, Vdd, num=100)
Ids = []
for V_g0 in Vgs:
    p.set_voltage(pyplane.DacChannel.AIN0, V_g0)
    time.sleep(0.05)
    Id = p.read_current(pyplane.AdcChannel.GO21_N) - Ids_0
    Ids.append(Id)
```

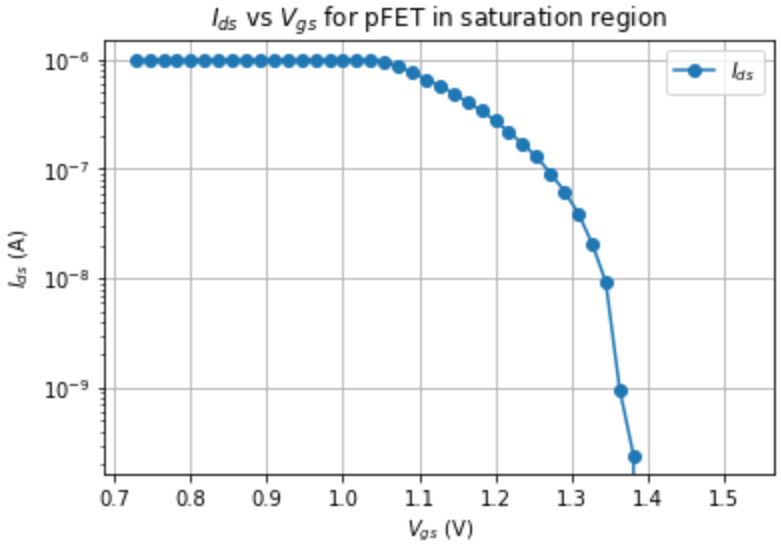
In [44]:
```
# plot in linear scale
plt.plot(Vgs[50:80], Ids[50:80], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for pFET in saturation region")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
```



In [46]:
```
# if it looks nice in the plot, save it!
np.savetxt("3_2.csv", [Vgs, Ids], delimiter=",")
```

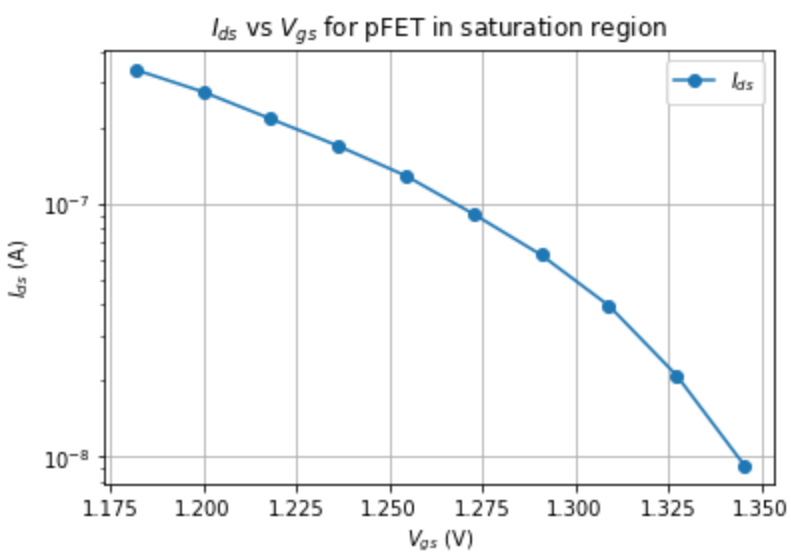
In [2]:
```
# Load data you saved and plot, to check if the data is saved correctly
[Vgs, Ids] = np.loadtxt("3_2.csv", delimiter=",")
```

In [5]:
```
# plot in logarithmic scale
plt.semilogy(Vgs[40:85], Ids[40:85], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for pFET in saturation region")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()

print(f"Arbitrary Vg close to Vds - threshold: {Vgs[71]}")
print(f"Arbitrary Ids close to Vds - threshold: {Ids[71]}")
```




```
Arbitrary Vg close to Vds - threshold: 1.2909090909090908
Arbitrary Ids close to Vds - threshold: 6.274414099394221e-08
```

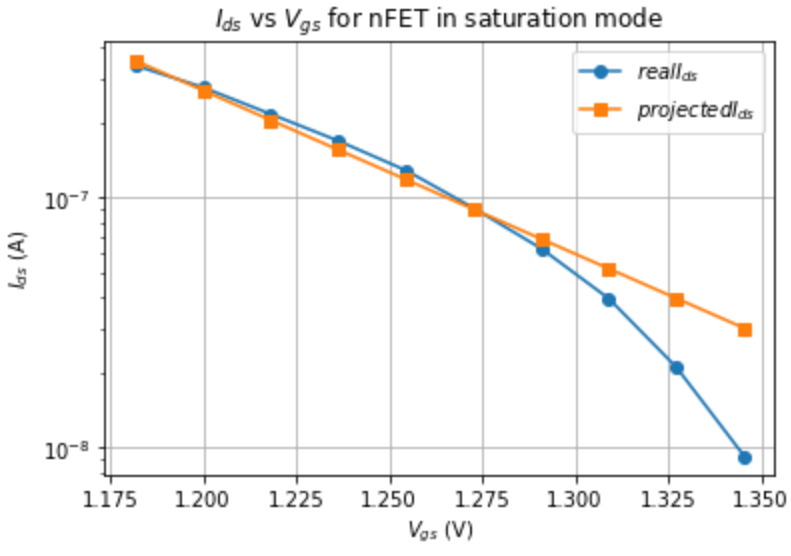
In [14]:
```
# extract the valid range and plot in logarithmic scale
plt.semilogy(Vgs[65:75], Ids[65:75], "o-", label="$I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for pFET in saturation region")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
```



> nFET: $I = I_0 e^{\frac{\kappa V_g - V_s}{U_T}}$
>
> pFET: $I = I_0 e^{\frac{-\kappa V_g + V_s}{U_T}}$

In [15]:

```
# fit in the valid range (you may want to add the fitted line in the plot)
from scipy.optimize import curve_fit

valid_Vgs = Vgs[65:75]
valid_Ids = Ids[65:75]
V_sp = 1.8
def get_Ids(Vgs, kappa, I0):
    Ut = 0.025
    return I0 * np.exp((-kappa * Vgs + V_sp)/Ut)
popt, pcov = curve_fit(get_Ids, valid_Vgs, valid_Ids, p0=[0.79, 1e-10])
fit_kappa, fit_I0 = popt
y = get_Ids(valid_Vgs, fit_kappa, fit_I0)
plt.semilogy(valid_Vgs, valid_Ids, "o-", label="$real I_{ds}$")
plt.semilogy(valid_Vgs, y, "s-", label="$projected I_{ds}$")
plt.title("$I_{ds}$ vs $V_{gs}$ for nFET in saturation mode")
plt.xlabel("$V_{gs}$ (V)")
plt.ylabel("$I_{ds}$ (A)")
plt.grid()
plt.legend()
plt.show()
```



Extract I_0 and κ

In [16]:
```
# I_0
print(f"I_0: {fit_I0} A")
```

```
I_0: 1.0146067145513782e-30 A
```

In [17]:
```
# kappa
print(f"kappa: {fit_kappa}")
```

```
kappa: 0.37634456732088933
```

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 [20]:
```
# compute threshold voltage
from scipy import interpolate
Th = None
extrapolated_Ids = None
actual_threshold_Ids = None
for Vg, Ids_ in zip(Vgs[65:], Ids[65:]):
    extrapolated_Ids = get_Ids(Vg, fit_kappa, fit_I0)
    if Ids_ < 0.5 * extrapolated_Ids:
        Th = Vg
        actual_threshold_Ids = Ids_
        break
Th = V_sp - Th
print(f"Threshold voltage: {Th} V")
    # compute Ids at threshold voltage
print(f"Extrapolated Ids at threshold: {extrapolated_Ids} A")
print(f"Actual Ids at threshold: {actual_threshold_Ids} A")
```

```
Threshold voltage: 0.4545454545454546 V
```

In [21]:
```
# compute Ids at threshold voltage
print(f"Extrapolated Ids at threshold: {extrapolated_Ids} A")
print(f"Actual Ids at threshold: {actual_threshold_Ids} A")
```

```
Extrapolated Ids at threshold: 3.01454427995773e-08 A
Actual Ids at threshold: 9.27734378208811e-09 A
```

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 κ.

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 [ ]:
```
```

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 [28]:
```
# set drain voltage
v_dn = 1.8
p.set_voltage(pyplane.DacChannel.GO22, v_dn)
print(f"The drain voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO22)} V")
```

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

In [29]:
```
# set trial gate and source voltages
vs_n = 0.0
p.set_voltage(pyplane.DacChannel.GO20, vs_n)
print(f"The source voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO20)} V")

vg_n = 0.0
p.set_voltage(pyplane.DacChannel.AIN0, vg_n)
print(f"The trial gate voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN0)} V")
```

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

In [30]:
```
# read trial Ids
Ids_0 = p.read_current(pyplane.AdcChannel.GO20_N)
print(f"The Ids_0 is {Ids_0} A")
```

```
The Ids_0 is 2.9296876036255526e-09 A
```

- Data aquisition

In [4]:
```
# 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 [5]:
```
Ids_target = 1.35009e-07
Vg_target = 0.49090
```

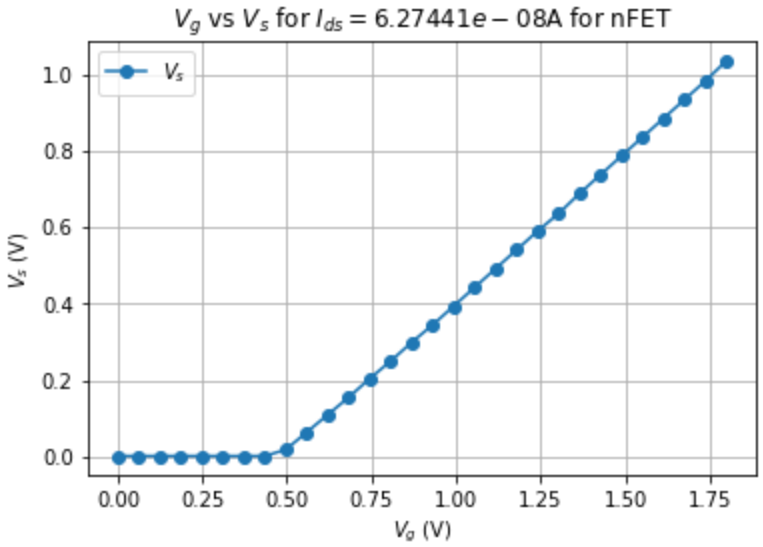
In [33]:
```
# initialize variables

import numpy as np
import time

Vg = np.linspace(0.0, Vdd, num=N_samples)
Vs = np.ones(N_samples) * Vdd/2
```

In [34]:
```
# 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 [31]:

```
# plot
import matplotlib.pyplot as plt

plt.plot(Vg, Vs, "o-", label="$V_{s}$")
plt.title(f"$V_{{g}}$ vs $V_{{s}}$ for $I_{{ds}} = {Ids_target} \\rm{{A}}$ for nFET")
plt.xlabel("$V_{g}$ (V)")
plt.ylabel("$V_{s}$ (V)")
plt.grid()
plt.legend()
plt.show()
```



In [37]:

```
# if it looks nice in the plot, save it!
np.savetxt("4_1.csv", [Vg, Vs], delimiter=",")
```

In [30]:

```
[Vg, Vs] = np.loadtxt("4_1.csv", delimiter=",")
```

- How do you compute κ? Does it stay constant for different V_g?

In [14]:

```
# calculate kappa
delta_index = 4
```

```
min_index = 10
from_indices = np.arange(min_index, N_samples - delta_index , delta_index)
to_indices = np.arange(min_index + delta_index, N_samples, delta_index)
kappas = [\
    (Vs[to] - Vs[from_]) / (Vg[to] - Vg[from_])
    for from_, to
    in zip(from_indices, to_indices)]
print("Kappas at different measurement points:", kappas)
```

```
Kappas at different measurement points: [0.7646484375000001, 0.7788085937499997, 0.7929687
500000006, 0.7929687500000002]
```

> Kappa varies in the magnitude of order of 10^{-2}, but stays relatively consistent for the last 2 kappas.

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 [10]:

```
# set bulk voltage
Vdd = 1.8
V_bulk = Vdd
p.set_voltage(pyplane.DacChannel.AIN1, V_bulk)
time.sleep(0.1)
print(f"The bulk voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN1)} V")
```

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

In [11]:

```
# set drain voltage
V_dp = 0.0
p.set_voltage(pyplane.DacChannel.GO21, V_dp)
print(f"The drain voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO21)} V")
```

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

In [12]:

```
# set trial gate and source voltages
V_sp = 1.8
p.set_voltage(pyplane.DacChannel.GO23, V_sp)
print(f"The source voltage is set to {p.get_set_voltage(pyplane.DacChannel.GO23)} V")

V_g0 = Vdd
p.set_voltage(pyplane.DacChannel.AIN0, V_g0)
print(f"The trial gate voltage is set to {p.get_set_voltage(pyplane.DacChannel.AIN0)} V")
```

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

In [13]:

```
# read trial Ids
Ids_0 = p.read_current(pyplane.AdcChannel.GO21_N)
```

```
print(f"The Ids_0 is {Ids_0} A")
```

```
The Ids_0 is 6.591796886112888e-09 A
```

- Data aquisition

In [24]:
```
# 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

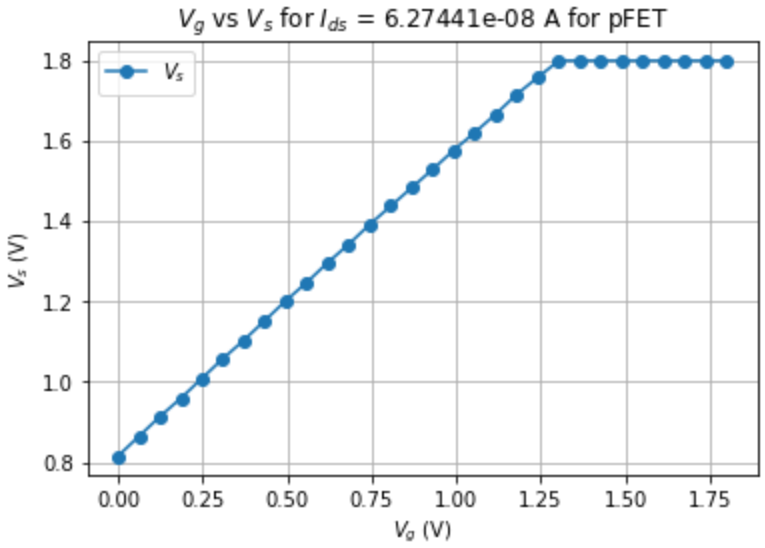
In [25]:
```
Ids_target = 6.27441e-08
Vg_target = 1.29090
```

In [16]:
```
# initialize variables

import numpy as np
import time

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

In [17]:
```
# 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 [29]:
```
# plot
import matplotlib.pyplot as plt
plt.plot(Vg, Vs, "o-", label="$V_{s}$")
plt.title(f"$V_{{g}}$ vs $V_{{s}}$ for $I_{{ds}}$ = {Ids_target} A for pFET")
plt.xlabel("$V_{g}$ (V)")
plt.ylabel("$V_{s}$ (V)")
plt.grid()
```

```
plt.legend()
plt.show()
```



In [19]:
```
# if it looks nice in the plot, save it!
np.savetxt("4_2.csv", [Vg, Vs], delimiter=",")
```

In [28]:
```
[Vg, Vs] = np.loadtxt("4_2.csv", delimiter=",")
```

- How do you compute κ? Does it stay constant for different V_g?

In [27]:
```
# calculate kappa
delta_index = 4
max_index = 18
from_indices = np.arange(0, max_index - delta_index, delta_index)
to_indices = np.arange(delta_index, max_index, delta_index)
kappas = [\
    (Vs[to] - Vs[from_]) / (Vg[to] - Vg[from_])
    for from_, to
    in zip(from_indices, to_indices)]
print("Kappas at different measurement points:", kappas)
```

```
Kappas at different measurement points: [0.7788085937499987, 0.7788085937500009, 0.7646484
375000014, 0.7434082031249999]
```

> Kappa varies in the magnitude of order of 10^{-2}, but stays relatively consistent for the first 2 kappas.

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 $\mu\mathrm{m}$ 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.

> It depends strongly on them. The thickness is antiproportional to the strength of the doping, because with higher doping less diffusion happens and so the required depletion region thickness to balance this is lower. The thickness is proportional to the channel potential, because we are in reverse bias, so a higher potential means more drift. Until, of course, we are in strong inversion, where the thickness will be pinned.

(2) Explain why κ varies with the source voltage at constant current (as in the source follower).

> Since for nFET with constant current:
>
> - $I_0 e^{\frac{\kappa V_g - V_s}{U_T}}$
> - $\Rightarrow V_s = \kappa V_g + U_T \ln \frac{I_0}{I}$
> - $\Rightarrow \kappa = \frac{dV_s}{dV_g}$
>
> But this ratio is not constant, as is clear from the experiments. The closer we are to strong inversion, the less effect can V_g have on V_s.

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

> The κ is pretty similar. I can imagine that the pFET has a slightly lower effective C_{ox} than the nFET, since it has to transport bigger particles. This would match the observation that the κ of pFET is slightly lower, giving more weight to C_{dep} in $\kappa = \frac{C_{ox}}{C_{ox}+C_{dep}}$.

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

> κ is more constant the further away the surface potential is from being pinned. So it is more constant at weak inversion.