Mandatory laboratory exercise 2

Characterizing Field Effect Transistors
INF3410
02.10.2016

Håvard Siljedal

1 Equipment

1.1 Hardware

1. Dual Power Supply (PSU): TTI EX752M
2. Power Supply (PSU): TTI EX355P
3. Bench Multimeter (DMM): Agilent 34401A
4. Hand Held Multimeter (DMM): Fluke 117
5. MC14007UB (DUT)

1.2 Software

1. Simulator: LTspice XVII
2. Programming Language: Python 2.7

2 Prelab

Prior to doing measurements and simulations, the mosfet characteristics will be calculated and plotted using the full EKV model including the inversion interpolation. The model and calculation step is listed below.

$$I_D = I_F - I_R$$

$$I_F = I_S \log\left(1 + e^{\frac{V_P}{2U_T}}\right)^2 (1 + \lambda V_{DS})$$

$$I_R = I_S \log\left(1 + e^{\frac{V_P - V_{DS}}{2U_T}}\right)^2 (1 + \lambda V_{DS})$$

$$I_S = 2nk_n \frac{W}{L} U_T^2$$

$$V_P = \frac{V_{GS} - V_{TO}}{n}$$

A Python implementation of the EKV model used to generate the plots is listed in appendix 6.1.

2.1 Task 1

The following parameters are given to complete the EKV model used in the calculations:

$k_n = 190uV/A$ (trans conductance)

$\lambda = 0.16$ (channel length modulation constant)

$W = 1um$ (transistor width)

$L = 1um$ (transistor length)

2.1.1 Id vs Vgs active region

Figure 1 and Figure 2 shows the Id/Vgs characteristics for the EKV in saturation. The latter is in log scale, and there the cut off region is highlighted (red) and correspond to Vgs > Vto.

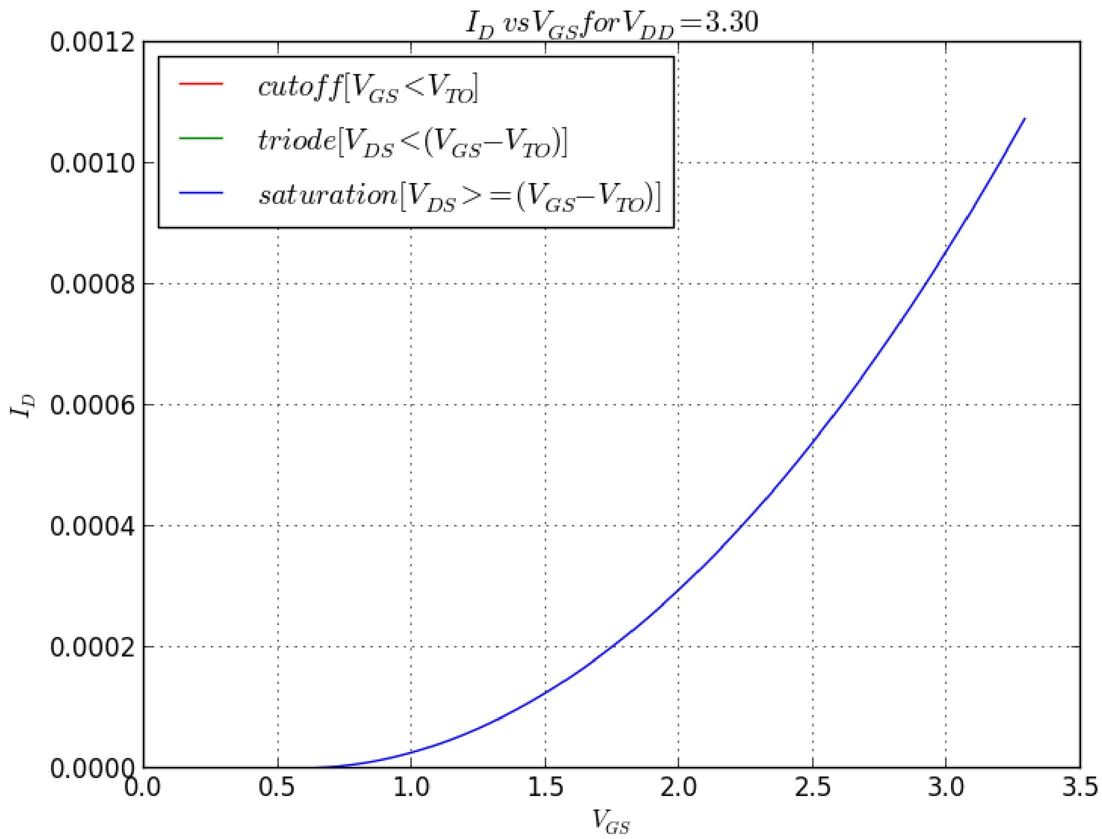


Figure 1 - Id vs Vgs active region

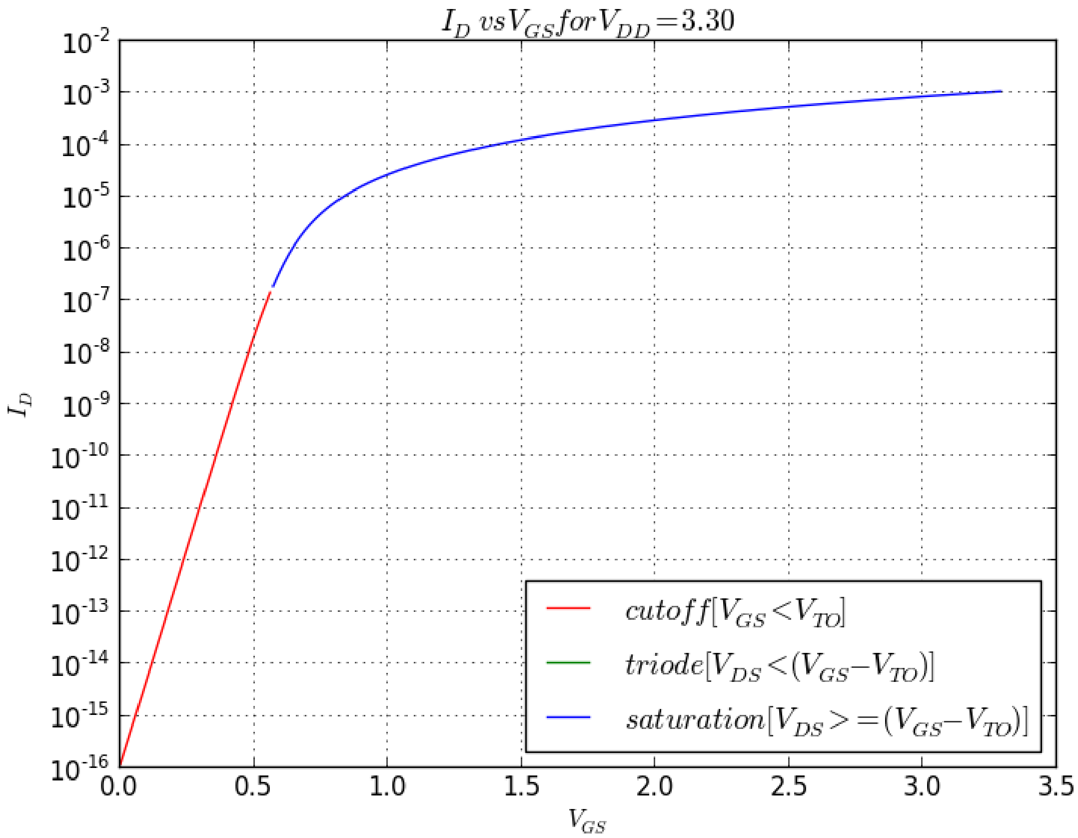


Figure 2 - Id vs Vgs active region (log scale)

2.1.2 Id vs Vgs triode region

Figure 3 and Figure 4 shows the characterisation for the transistor in triode region. Some sections of the plot is actually not in the triode region and these are highlighted (red and blue)

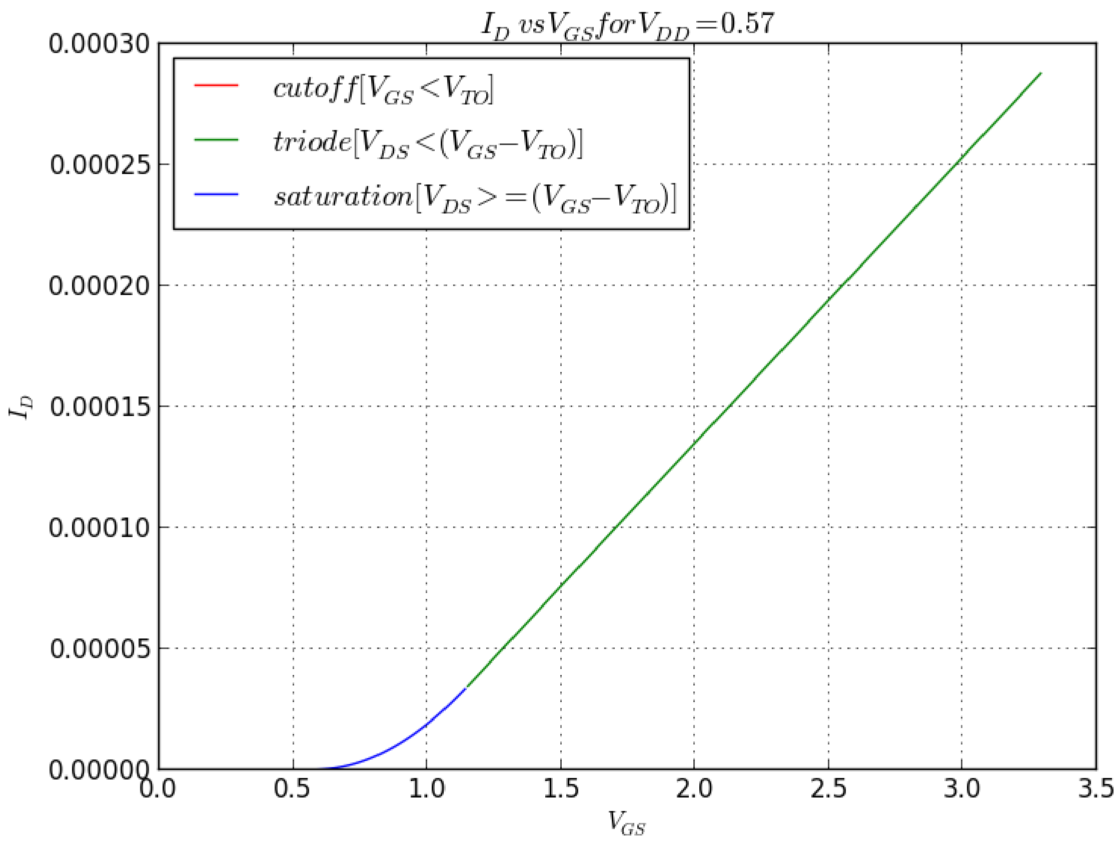


Figure 3 - Id vs Vgs triode region

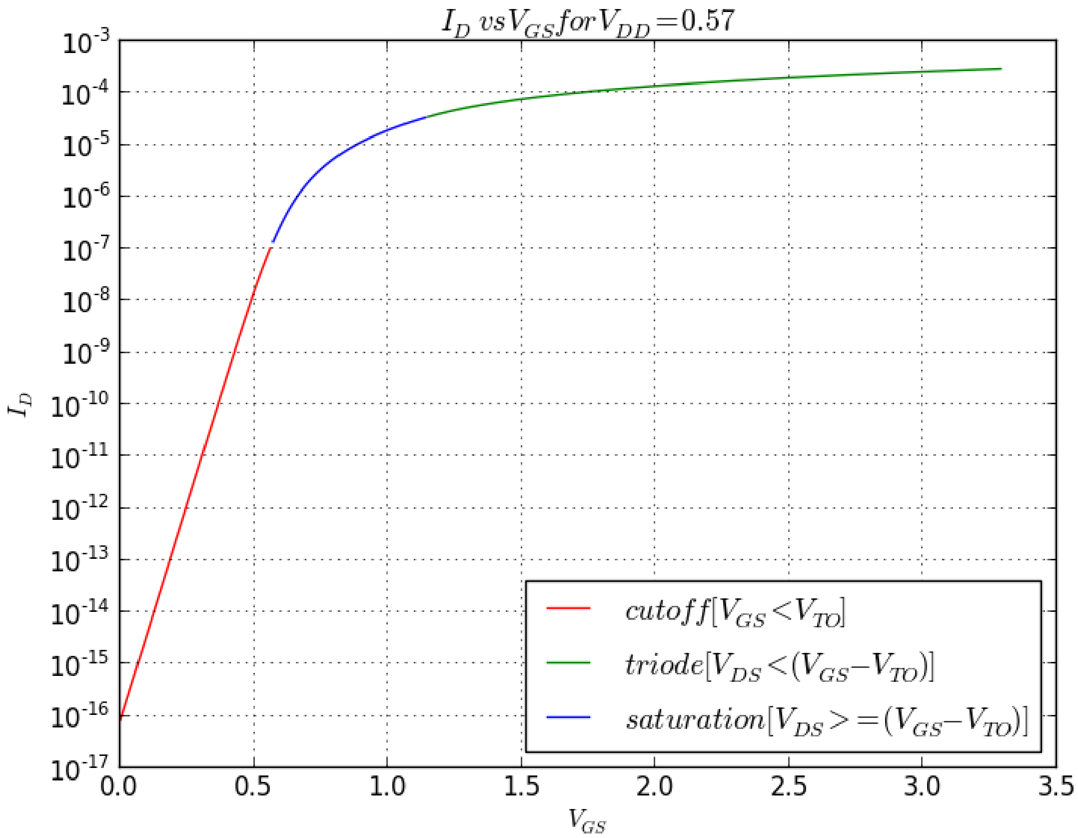


Figure 4 - Id vs Vgs triode region (log scale)

2.1.3 Id vs Vds strong inversion

In this and the next section, the Id/Vds characterisation is investigated. Figure 5 shows this for different values of Vgs. The triode region (Vds<Vgs-Vto) is grey in this plot, and the transitions from triode to saturation (Vov) are marked for each value of Vgs. These points form the exponential shape that is typical for a MOSFET at Vds=Vgs-Vto.

Not the strong influence of the channel length modulation constant. This is visible in the saturation region on the drain current as a linear dependence of Vds.

The curve for Vgs=0.57V (=Vto) does not show up in the figure since this the cut off gate voltage. The Id/Vds relationship for gate voltages lower than the threshold voltage is in weak inversion and must be plotted with a logarithmic scale to be visible. This is done in the next section.

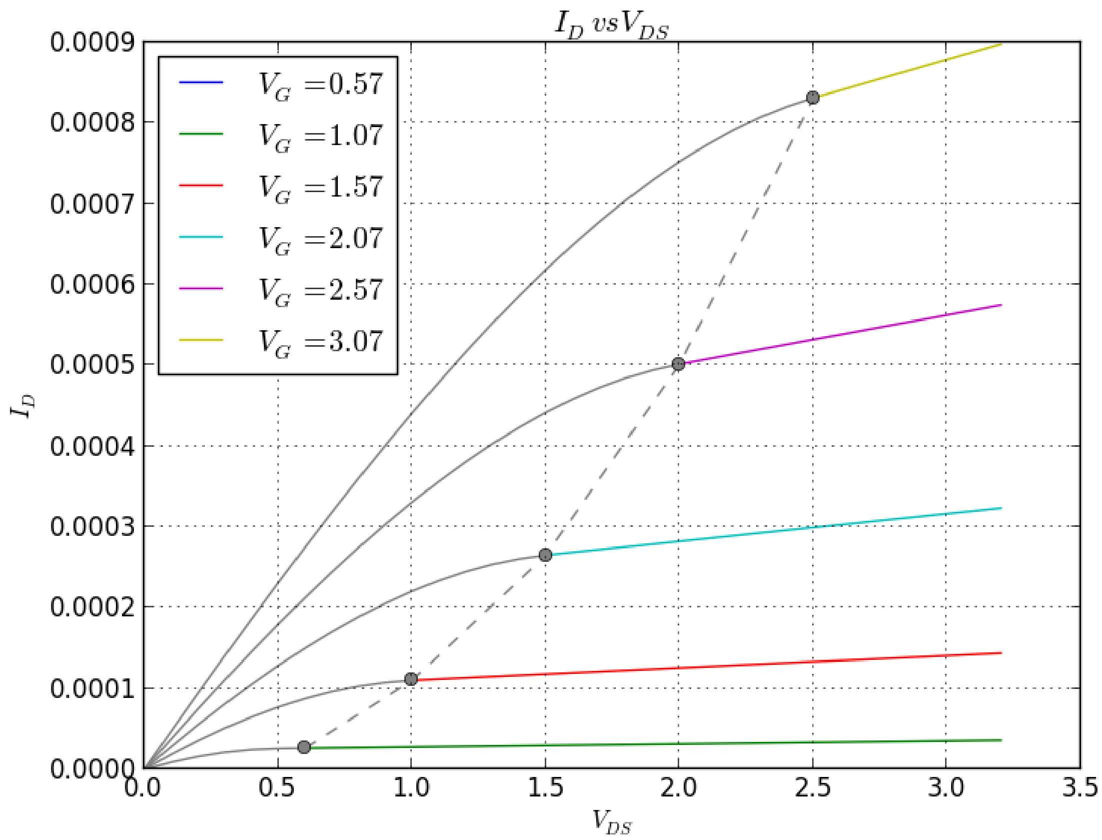


Figure 5 - Id vs Vds in strong inversion

2.1.4 Id vs Vds weak inversion

Doing the same thing in weak inversion (Vgs < Vto) shows the expected transition points at 4*Ut=104mV. These are approximately constant for different Vgs. Ut is the thermic voltage constant of 26mV at room temperature. All curves are for gate voltages lower than the threshold voltage and is thus in weak inversion.

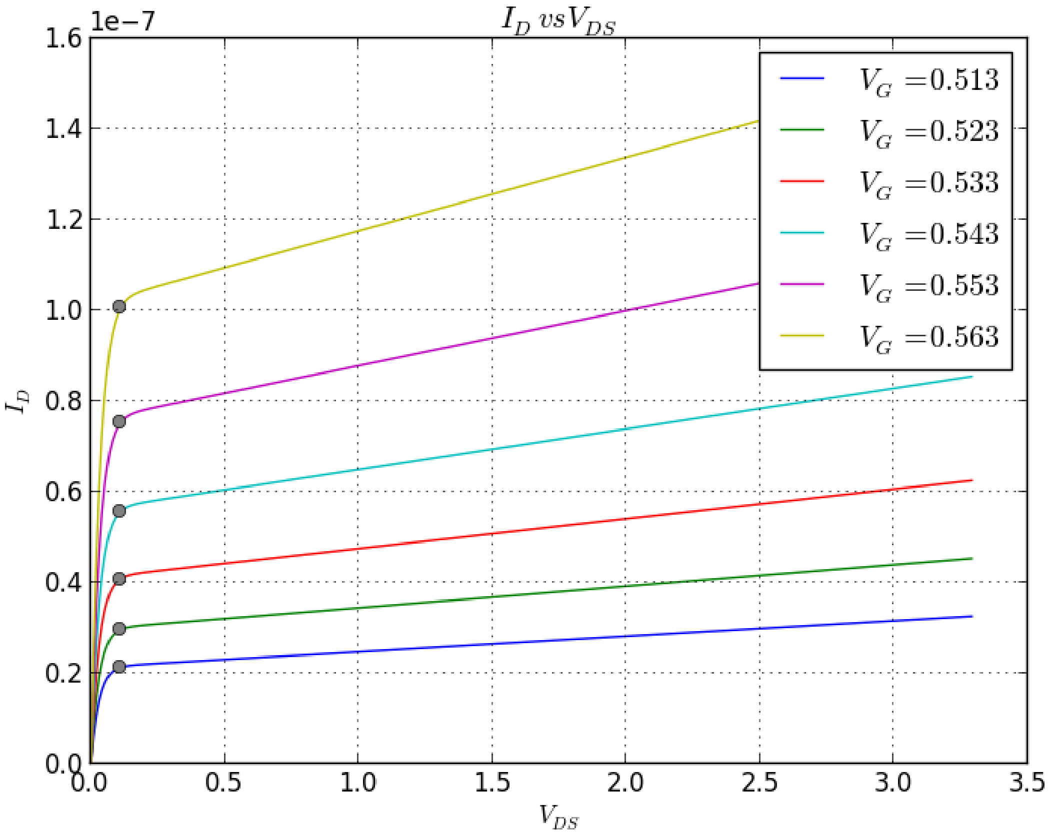


Figure 6 - Id vs Vds in weak inversion

3 Simulation

To verify the calculations, simulations will be performed on a more advanced model than the EKV used in the calculations. All simulations are performed using a 0.35um model from:

http://analogicdesign.com/students/netlists-models/.

The details of this model are listed in appendix 6.2.

As simulator LTspice is used. This is a free simulator with advanced modelling capabilities which covers most of the common MOSFET models (implementation levels):

1 Shichman-Hodges
2 MOS2
3 MOS3
4 BSIM
5 BSIM2
6 MOS6
8 BSIM3v3.3.0
9 BSIMSOI3.2
12 EKV 2.6
14 BSIM4.6.1

Figure 7 shows the schematics used for simulations.

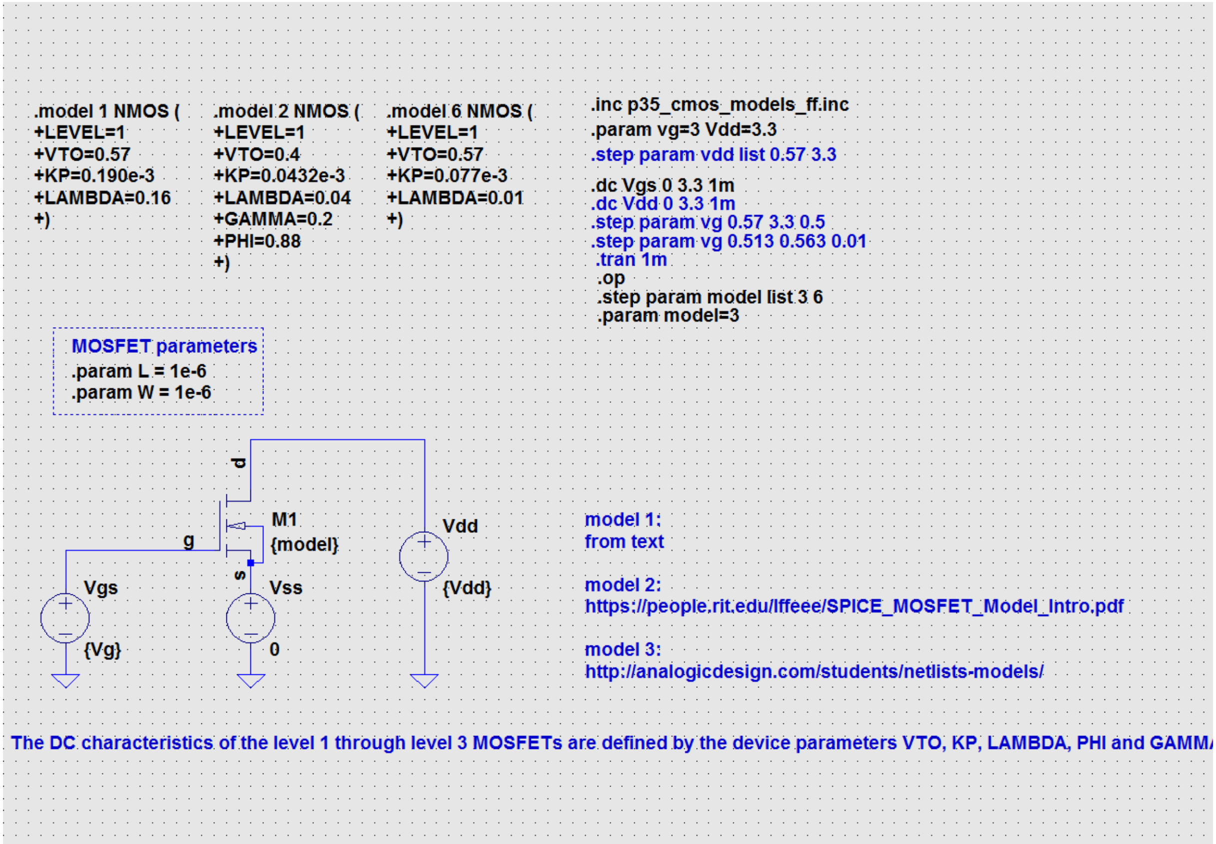


Figure 7 - Schematics of simulation

3.1 Task 2

The measurements performed in Task 1 are now repeated in the LTspice simulator with the MOSFET model mentioned.

3.1.1 Id vs Vgs active region (simulated)

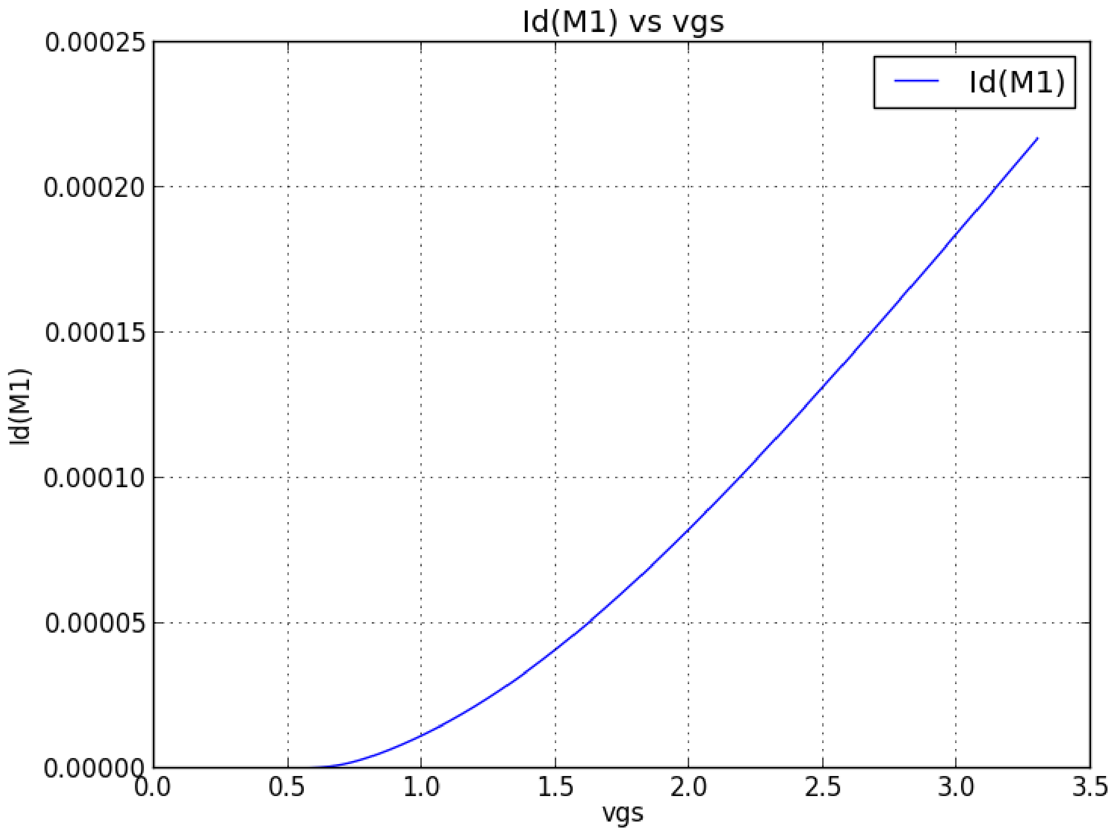


Figure 8 - Id vs Vgs active region, Vds=3.3V

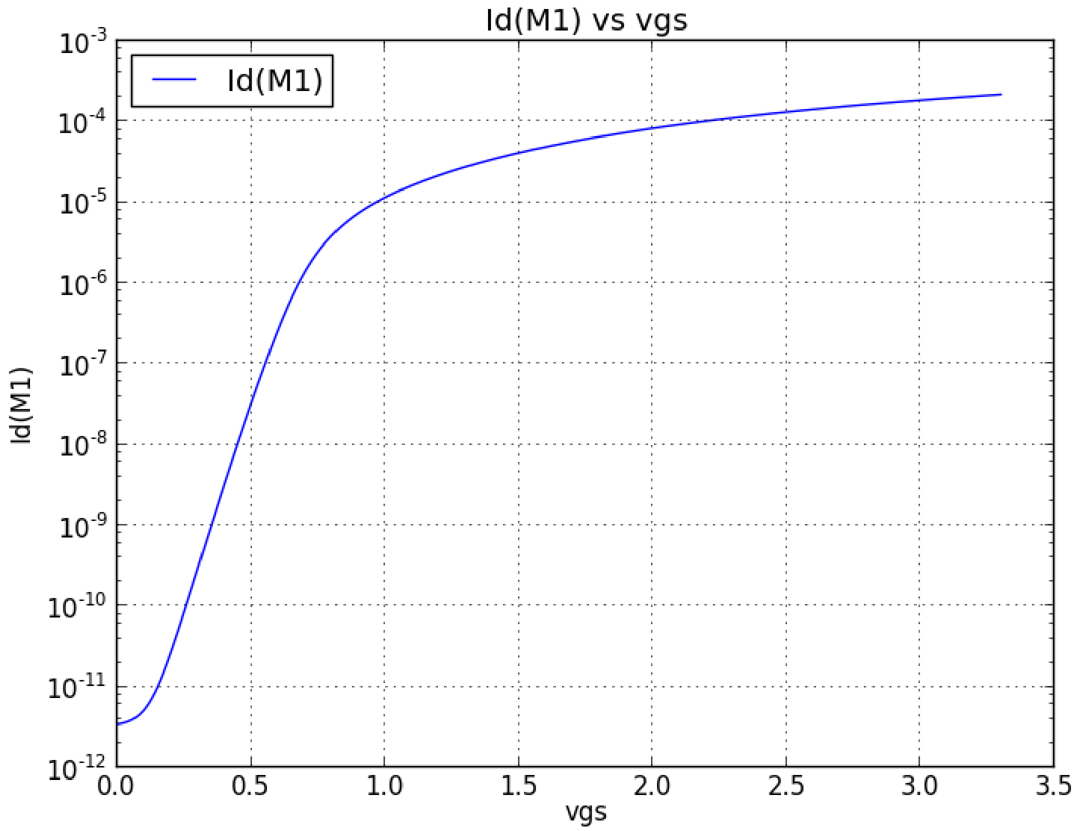


Figure 9 - Id vs Vgs active region (simulated) (log scale), Vds=3.3V

3.1.2 Id vs Vgs triode region (simulated)

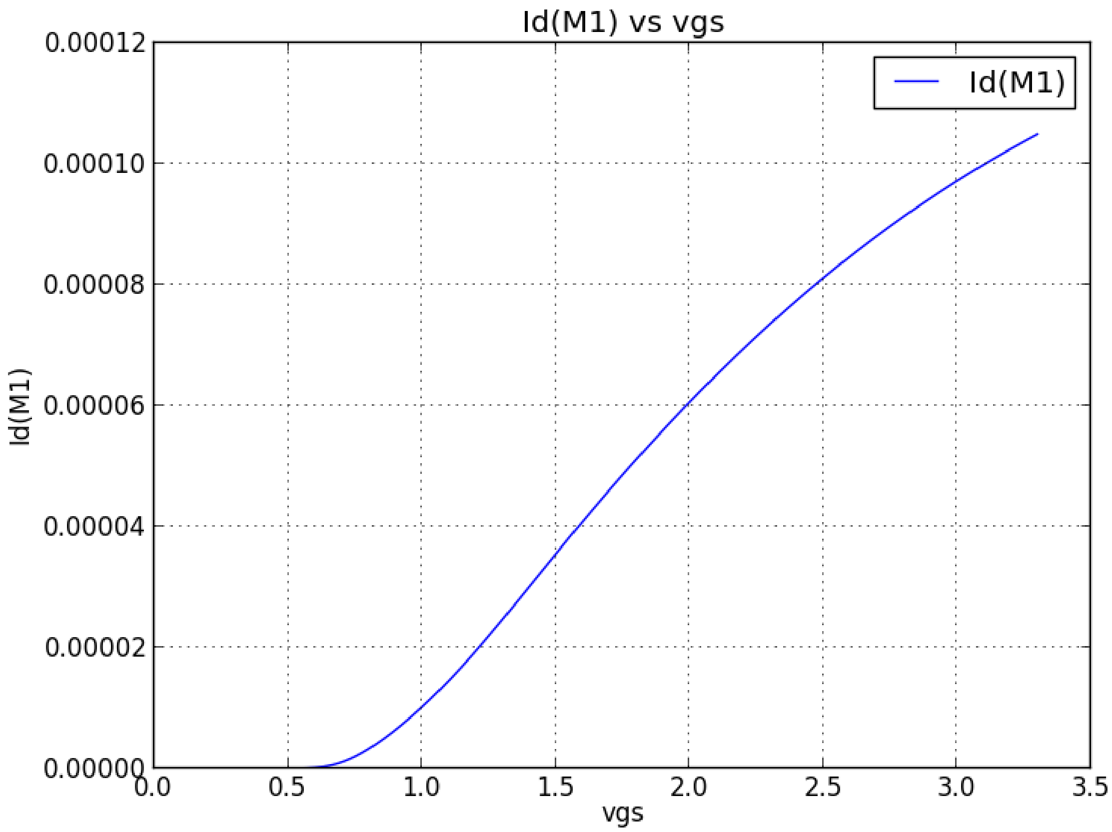


Figure 10 - Id vs Vgs triode region (simulated), Vds=0.57V

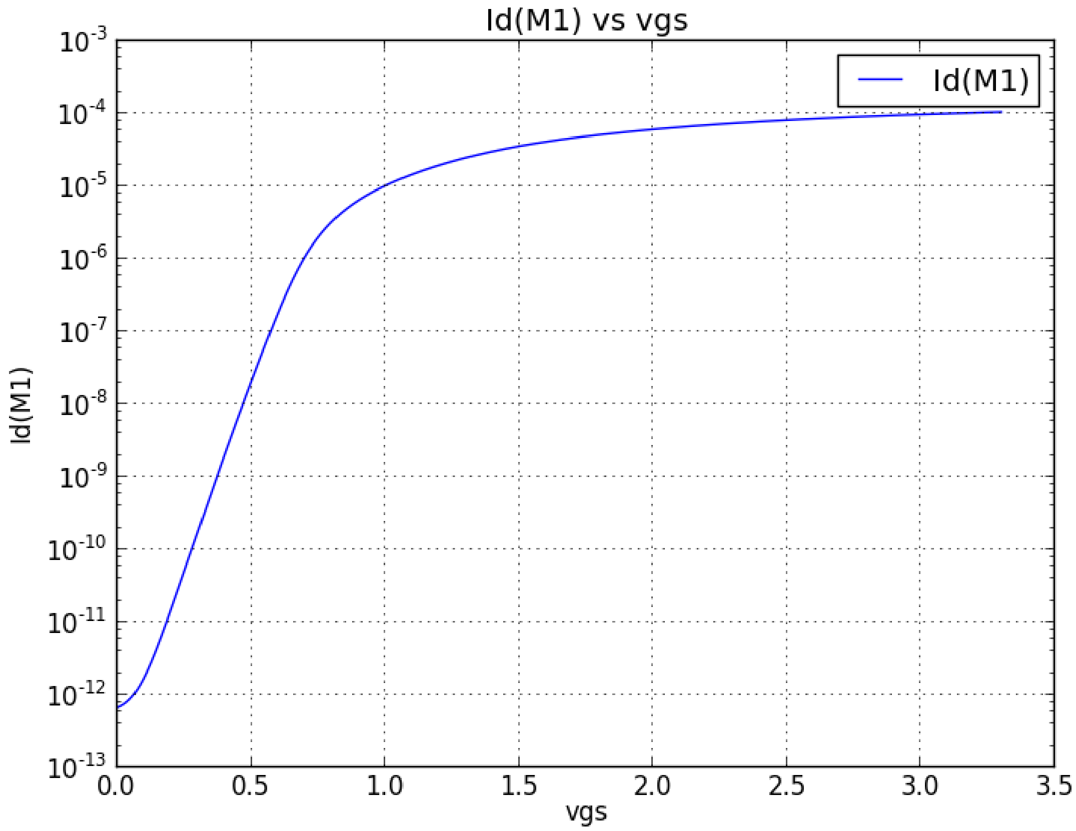


Figure 11 - Id vs Vgs triode region (log scale) (simulated), Vds=0.57V

3.1.3 Id vs Vds strong inversion (simulated)

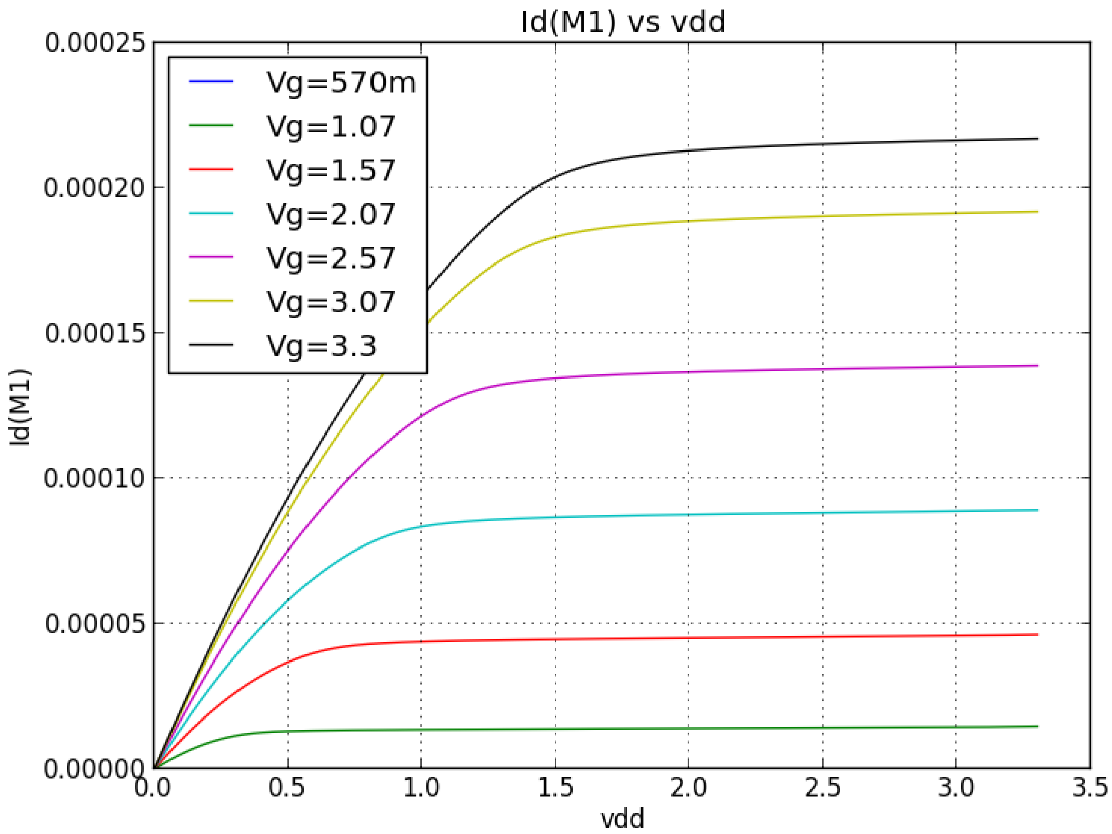


Figure 12 - Id vs Vds in strong inversion (simulated)

3.1.4 Id vs Vds weak inversion (simulated)

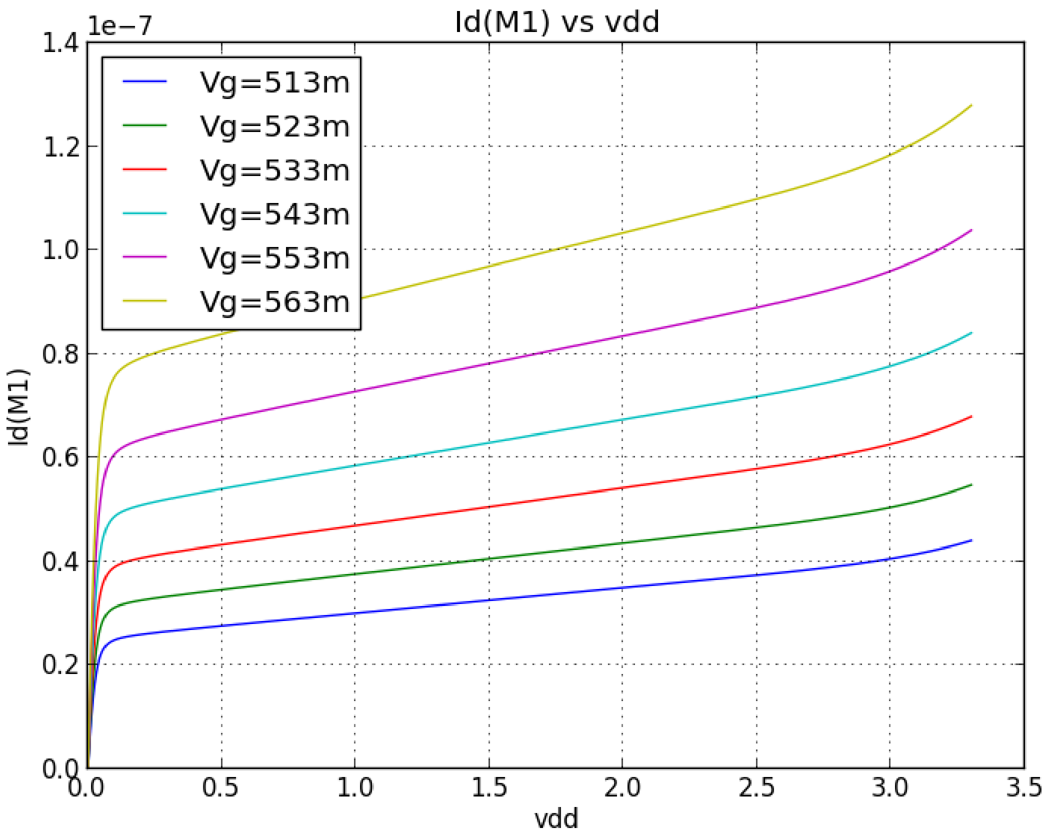


Figure 13 - Id vs Vds in weak inversion (simulated)

3.2 Task 3

To visualize the difference between the simulations and the calculations, the plots for the Id/Vgs characteristics are shown in the same plot.

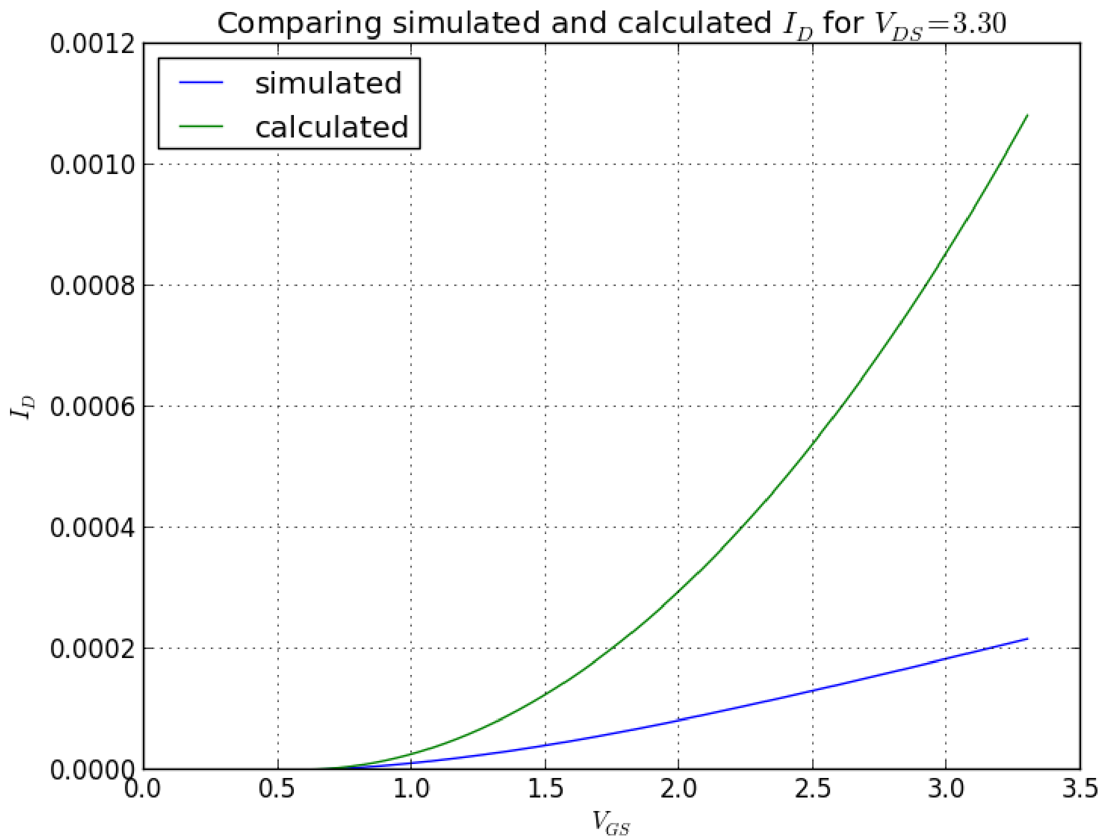


Figure 14 - Comparison of simulated and calculated Id/Vgs characteristics

Figure 14 shows that the calculation and simulation differs quite a lot. This is mainly due to the parameters given in the exercise text does not match the model used for the 0.35um process very well. Also, the simple EKV model will never match this high level model exactly.

The threshold voltage does not seem to be any different, so assuming 570mV is close enough. Regarding k_n, the value could most easily be retrieved by looking at the triode region, where Id is given by:

$$I_D = k_n \frac{W}{L}((V_{GS} - V_T)V_{DS} - \frac{1}{2}{V_{DS}}^2)$$

For small values of Vds and W/L=1, this reduces to:

$$I_D = k_n((V_{GS} - V_T)V_{DS})$$

Solving for k_n:

$$k_n = \frac{I_D}{(V_{GS} - V_{TO})V_{TO}}$$

Figure 15 shows k_n as a function of Vgs.

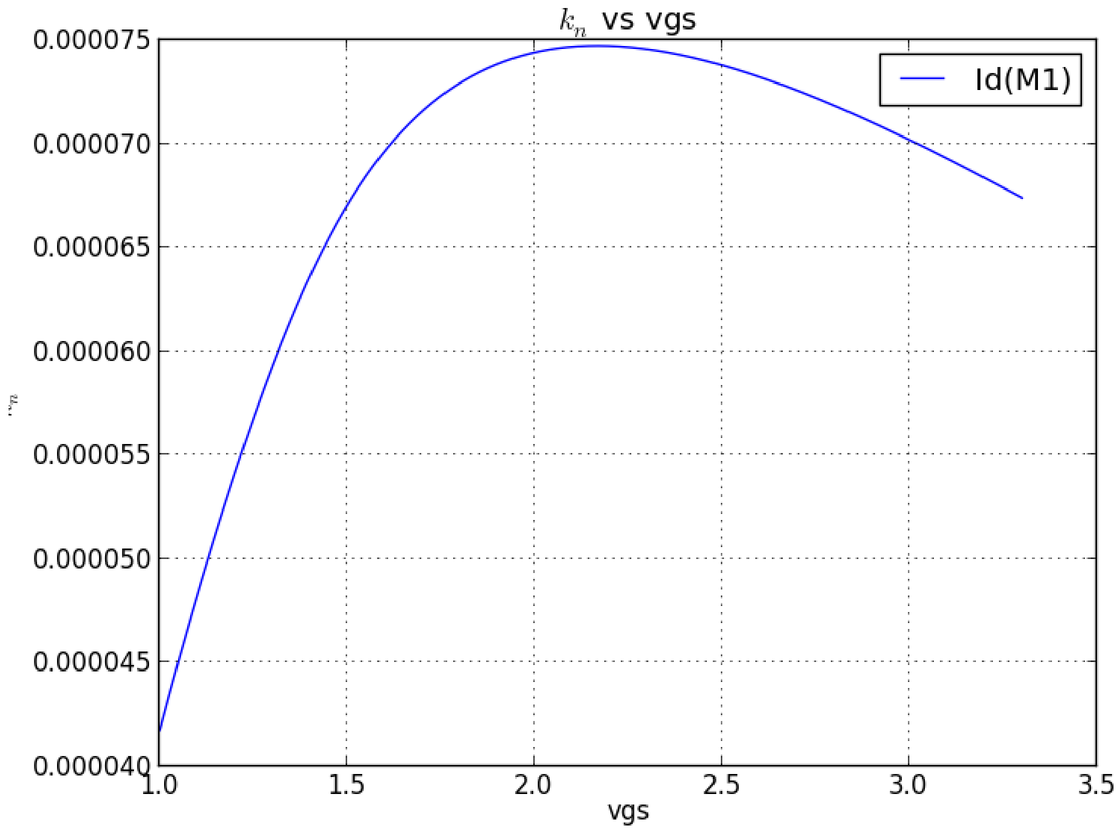


Figure 15 - k_n as a function of Vgs in triode region

An approximate value for k_n could then be 65u, depending of where the model should fit best.

$$k_n = 65u$$

To find lambda saturation region should be investigated. Using the simulation output in Figure 12, and LTspice to measure the values of Id at two different Vds, an approximate value of lambda could be found:

$$I_{D1} = i_d * (1 + \lambda V_{DS1})$$
$$i_d = \frac{I_{D1}}{1 + \lambda V_{DS1}}$$

$$I_{D2} = i_{d(1+\lambda V_{DS2})}$$
$$i_d = \frac{I_{D2}}{1 + \lambda V_{DS2}}$$

$$\frac{I_{D1}}{1 + \lambda V_{DS1}} = \frac{I_{D2}}{1 + \lambda V_{DS2}}$$

$$I_{D1}(1 + \lambda V_{DS2}) = I_{D2}(1 + \lambda V_{DS1})$$
$$I_{D1} + I_{D1}V_{DS2}\lambda = I_{D2} + I_{D2}V_{DS1}\lambda$$
$$I_{D1} - I_{D2} = \lambda(I_{D2}V_{DS1} - I_{D1}V_{DS2})$$

$$\lambda = \frac{I_{D1} - I_{D2}}{I_{D2}V_{DS1} - I_{D1}V_{DS2}}$$

The simulation gives the following values to the two Id:

$$V_{DS1} = 2$$
$$V_{DS2} = 3$$
$$I_{D1} = 181.29u$$
$$I_{D2} = 183.88u$$

Giving a lambda of:

$$\lambda = 0.015$$

A new plot with these values is shown in Figure 16. The match is not exact, but better than before.

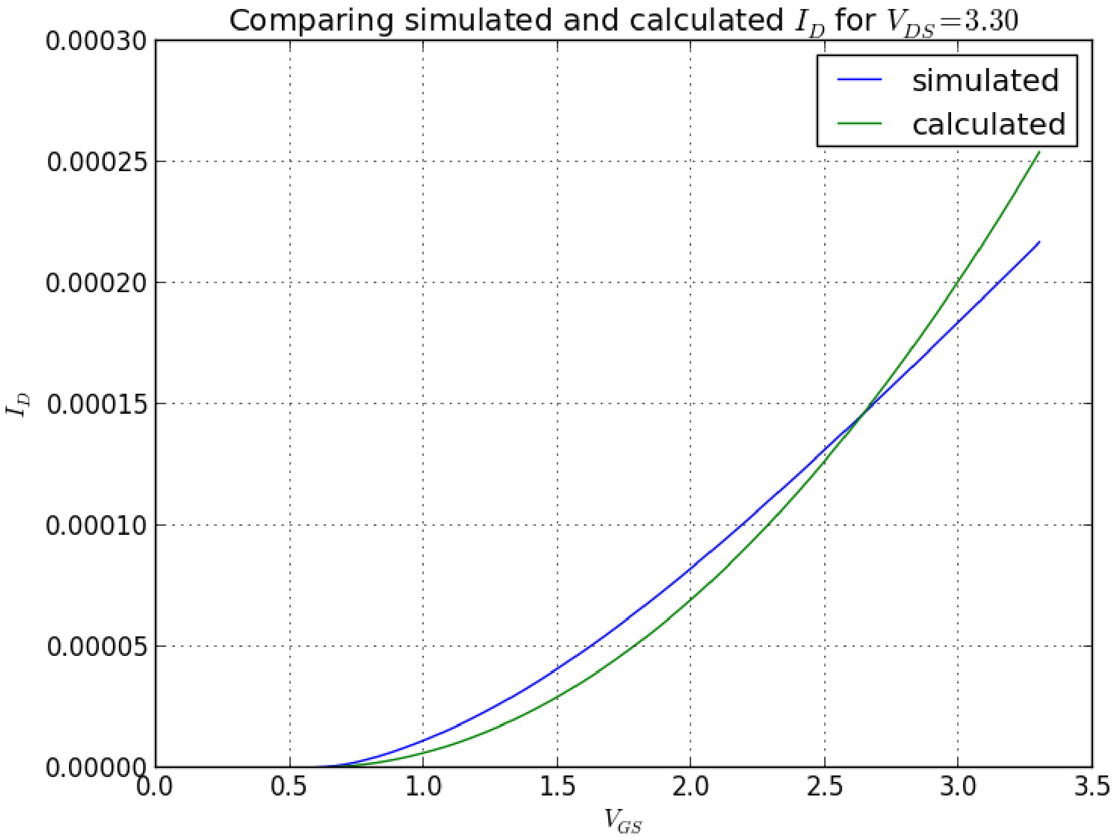


Figure 16 - Comparison of simulated and calculated Id/Vgs characteristics with improved parameters

The new parameters are:

$\lambda = 0.015$
$k_n = 65u$
$V_{TO} = 0.57$ (not changed)

4 Measurement

The measurements should also be executed on a real device (MC14007). Figure 17 shows the instruments used.

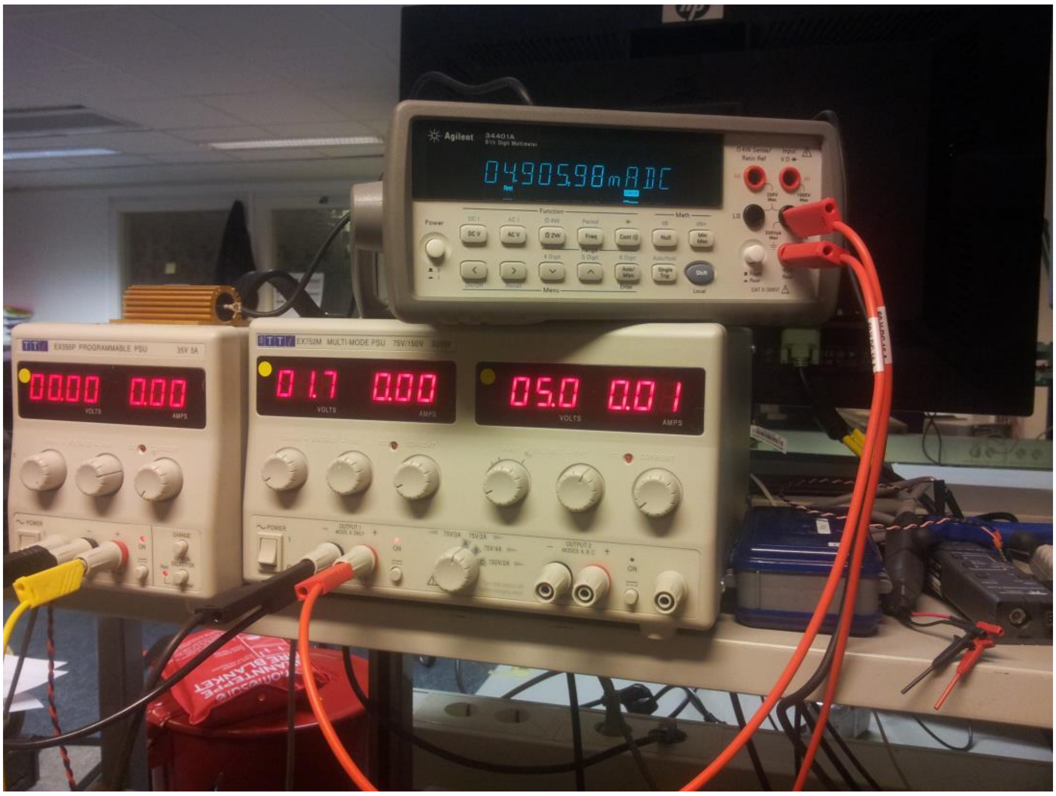
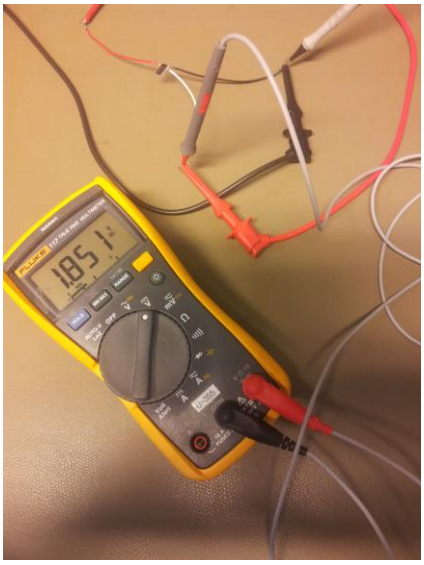
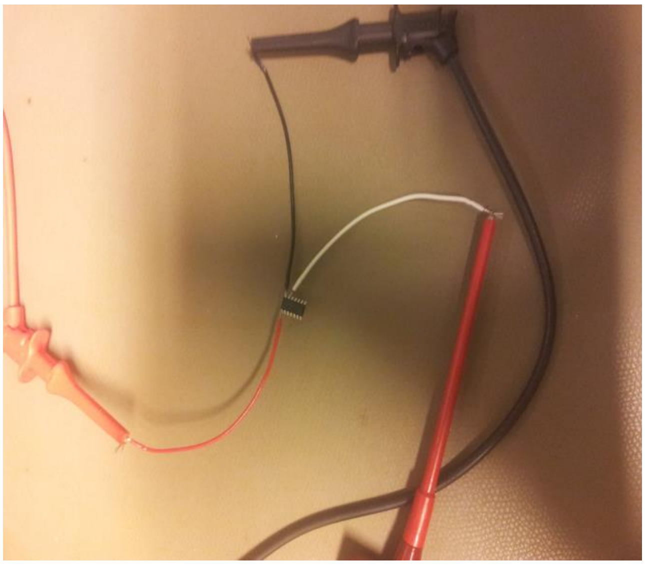


Figure 17 - Instruments used for measurements





4.1 Task 4

4.1.1 Id vs Vgs active region (measured)

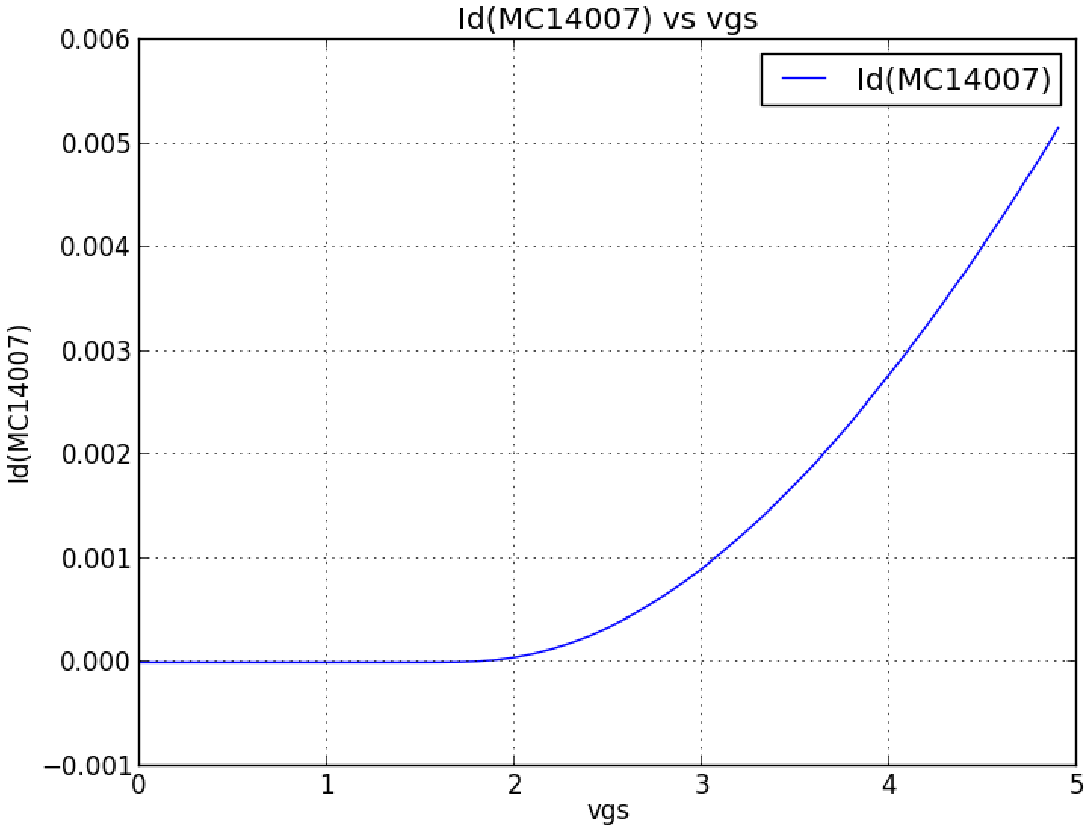


Figure 18 - Id vs Vgs active region, Vds=5.0V

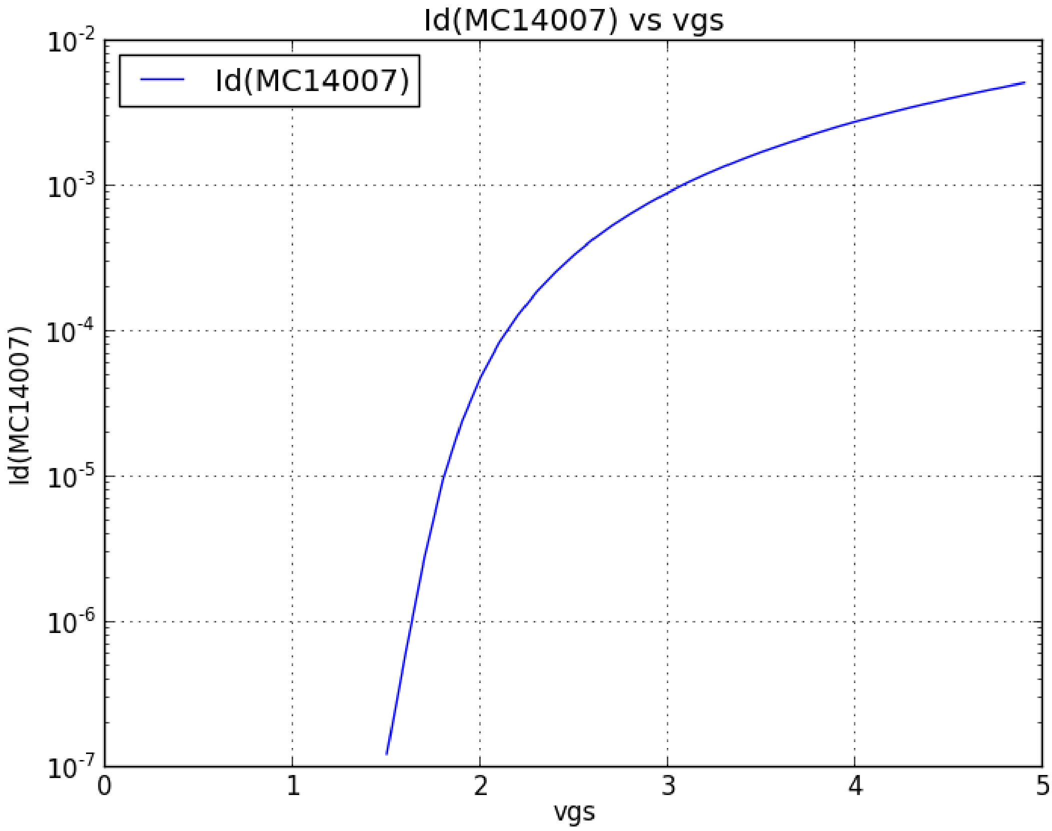


Figure 19 - Id vs Vgs active region, Vds=5.0V (logy)

4.1.2 Id vs Vgs triode region (measured)

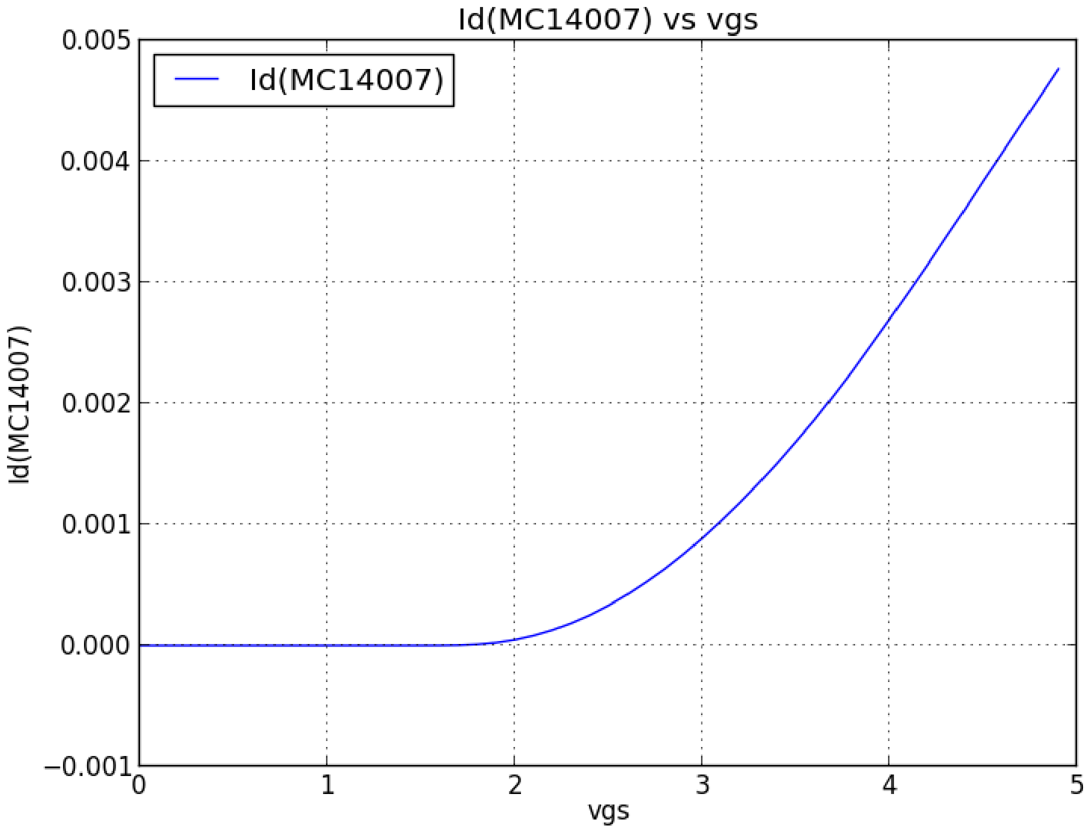


Figure 20 - Id vs Vgs triode region (simulated), Vds=1.5V

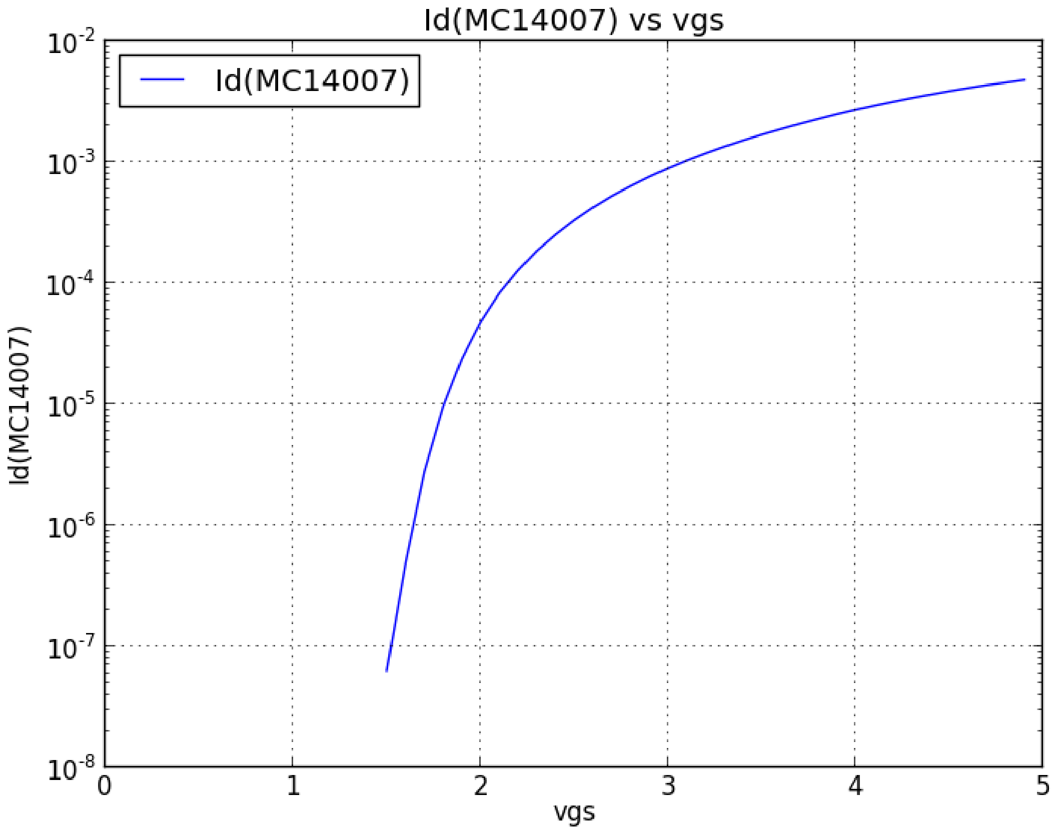


Figure 21 - Id vs Vgs triode region (simulated), Vds=1.5V (logy)

4.1.3 Id vs Vds strong inversion (measured)

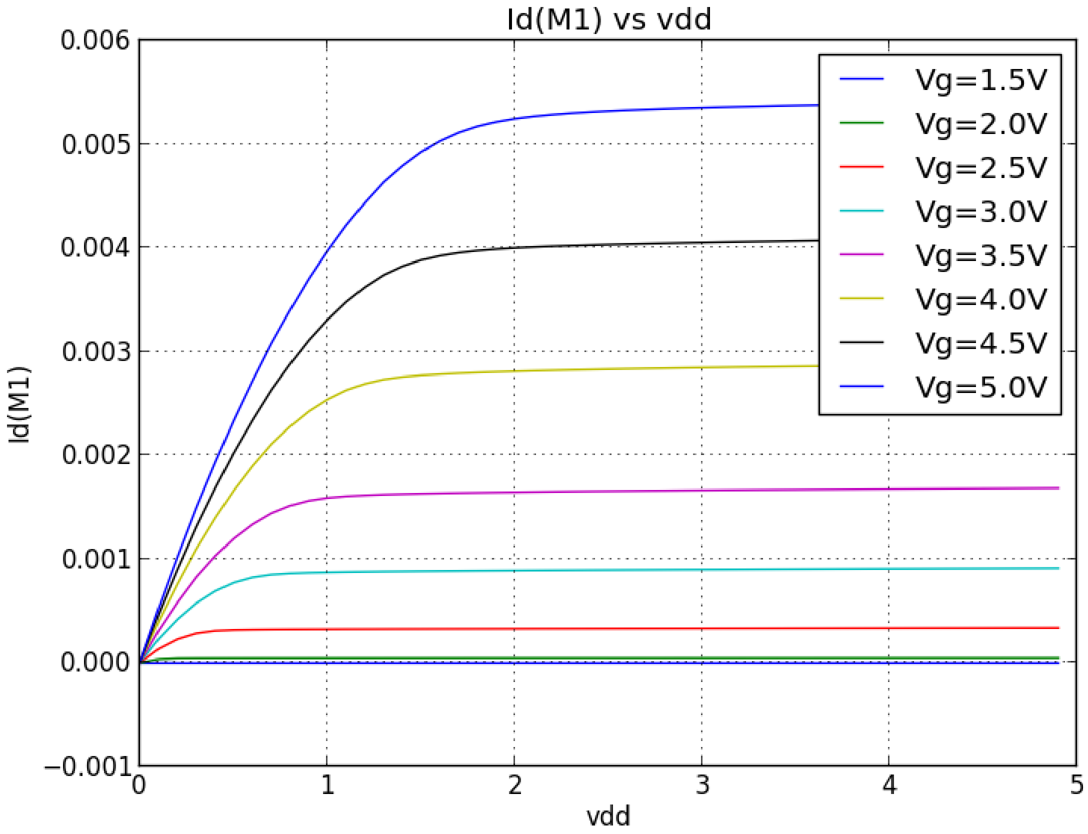


Figure 22 - Id vs Vds in strong inversion (measured)

4.1.4 Id vs Vds weak inversion (measured)

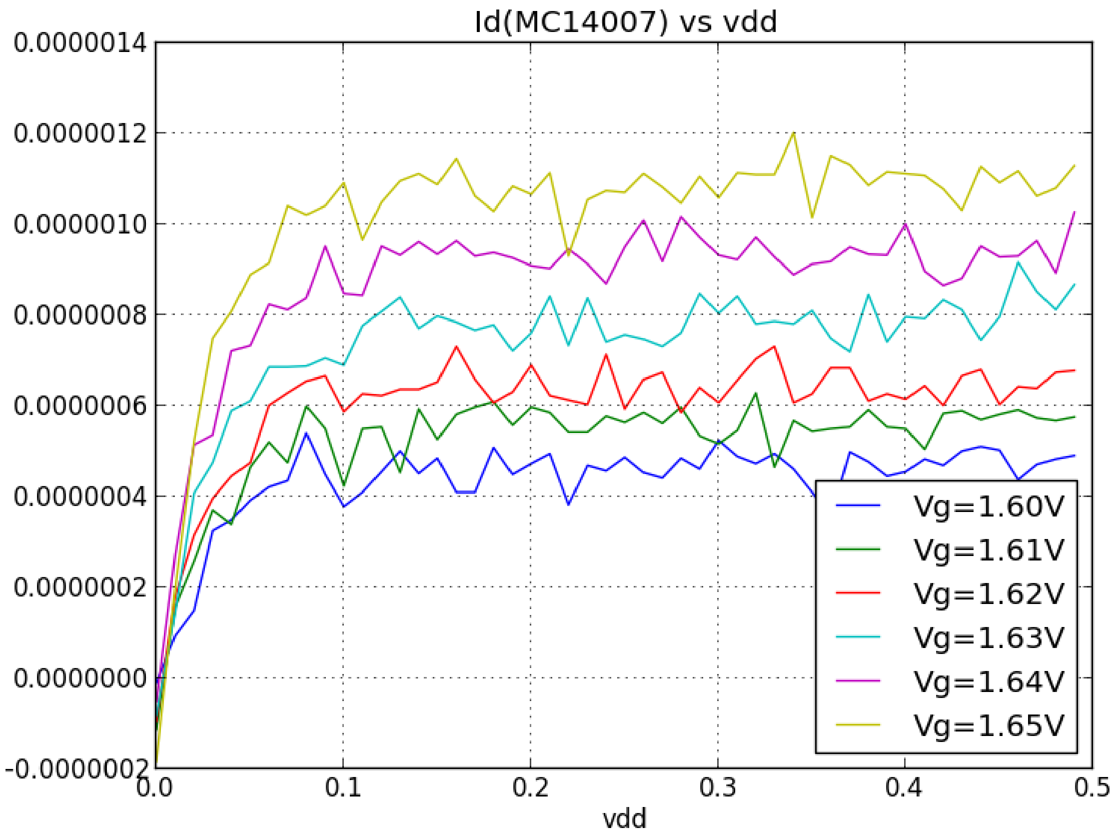


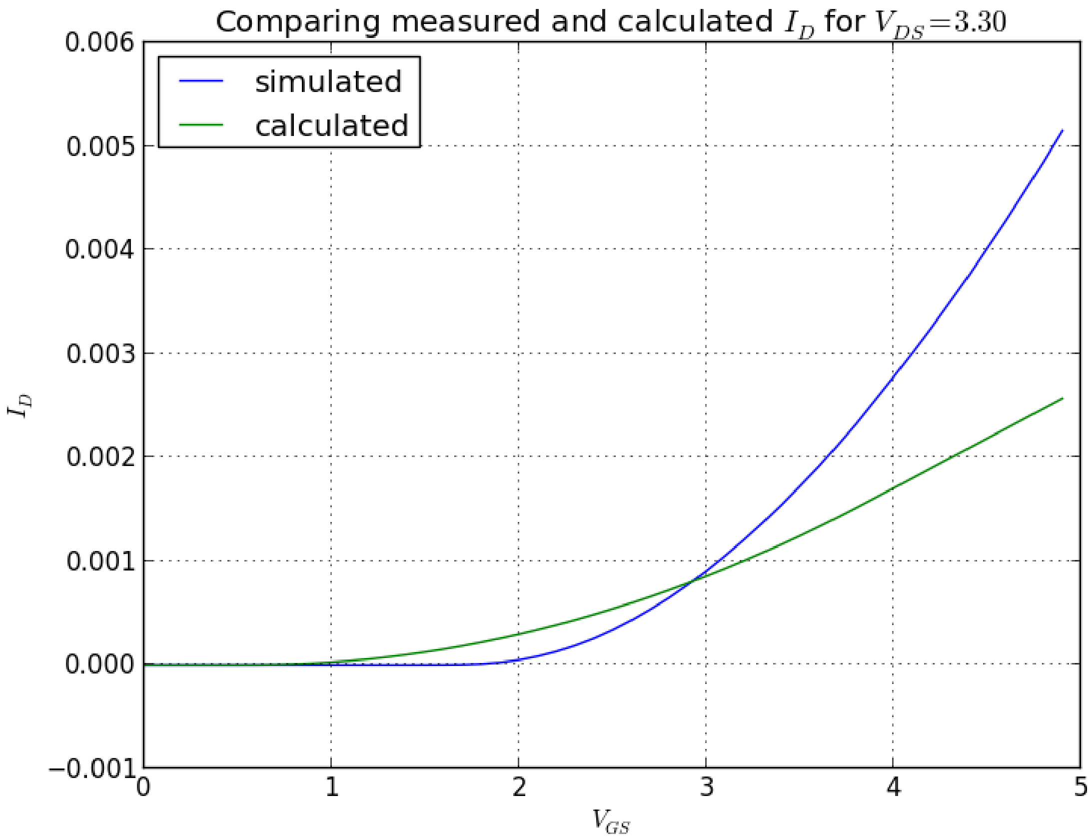
Figure 23 - Id vs Vds in weak inversion (measured)

Despite noisy measurement, Figure 23 shows same behaviour as calculations/simulations.

4.2 Task 5

The same approach as in Task 3 could be used to calculate parameters for the EKV model.

Plotting against the unmodified EKV:



4.2.1 Vth

Using Figure 18 Vth is found to be 1.5V

4.2.2 Lambda

Using curve for Vgs=5V and formula from Task 3.

$V_{DS1} = 3$

$V_{DS2} = 4.9$

$I_{D1} = 5.35m$

$I_{D2} = 5.41m$

$\lambda = 0.006$

4.2.3 K_n

Same approach as in Task 3:

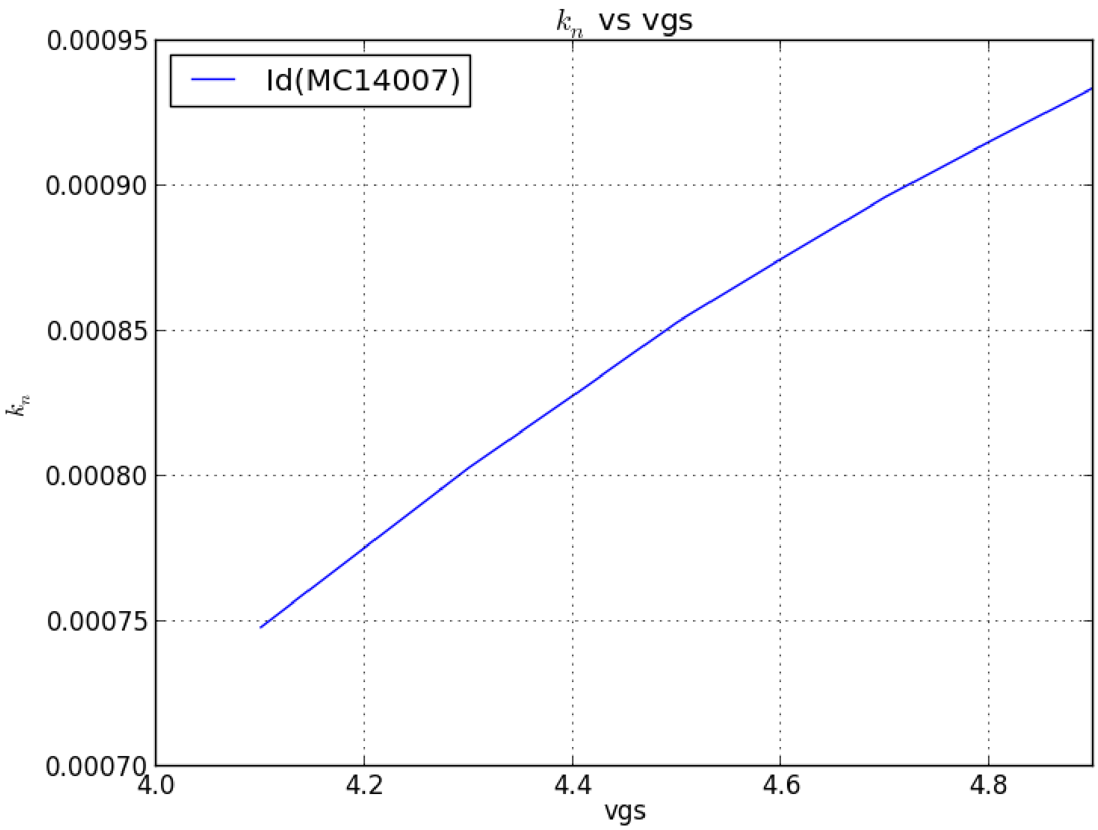


Figure 24 - k_n as a function of Vgs in triode region on MC14007

4.2.4 Compare model with real device

Modified parameters:

$\lambda = 0.0006$
$k_n = 850u$
$V_{TO} = 1.5$

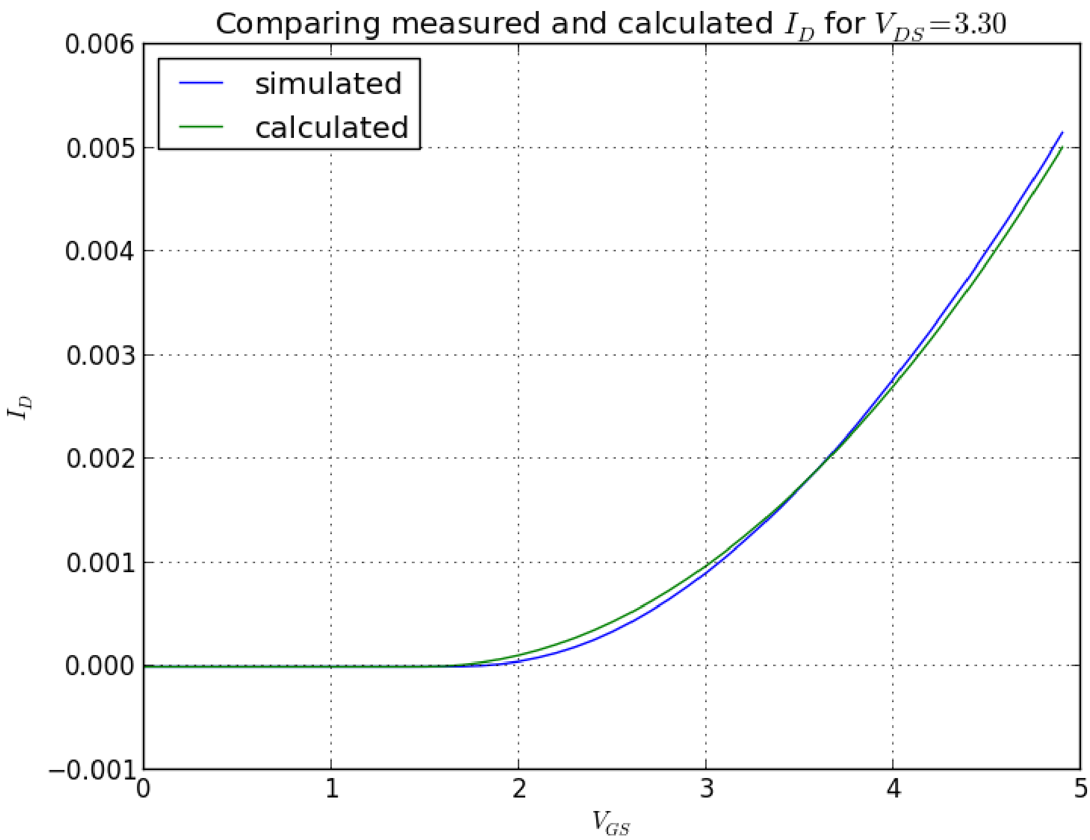


Figure 25 - Comparison of measured and calculated Id/Vgs characteristics with improved parameters

The new model is much closer to the real device

5 Common Source Amplifier

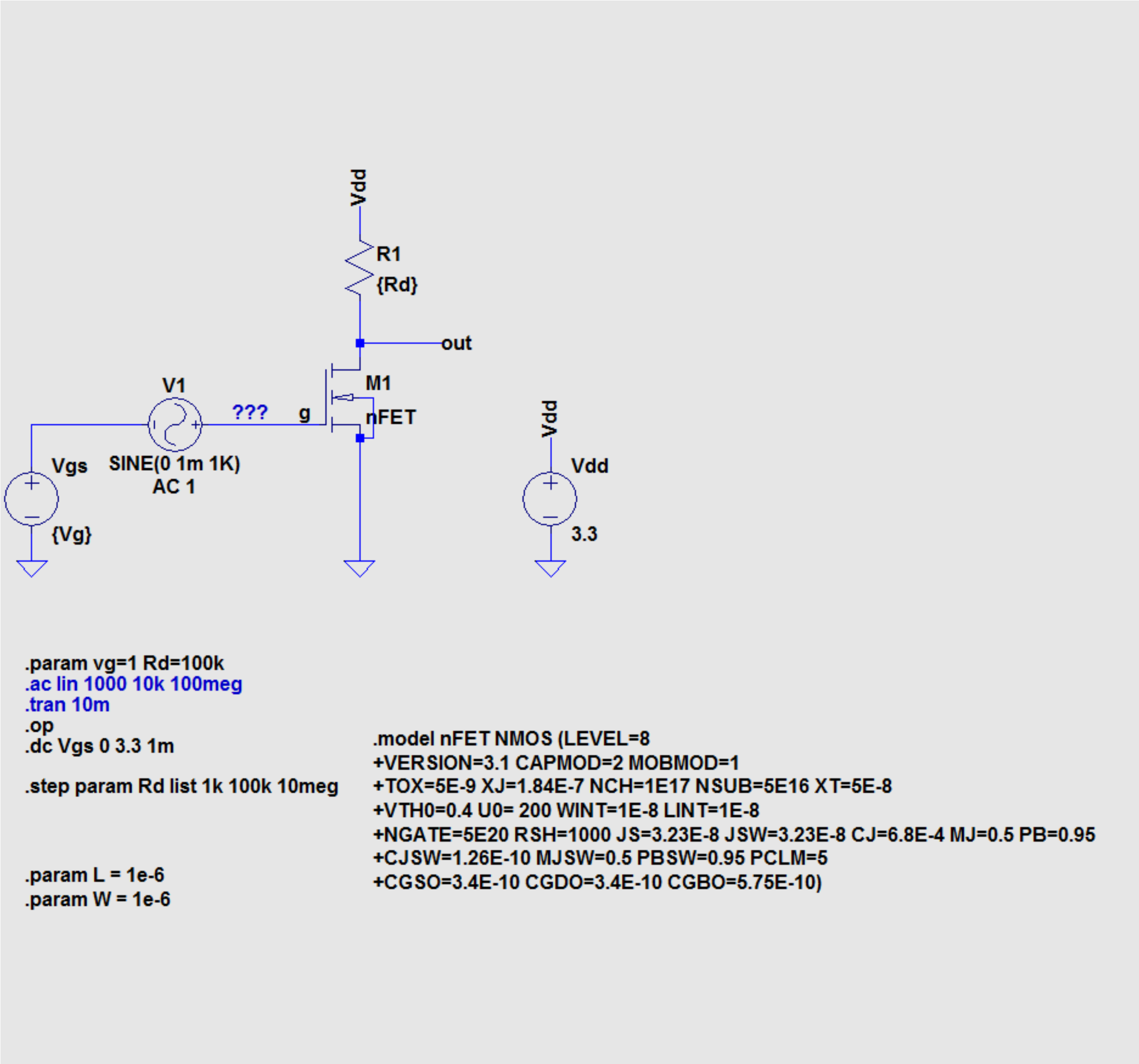
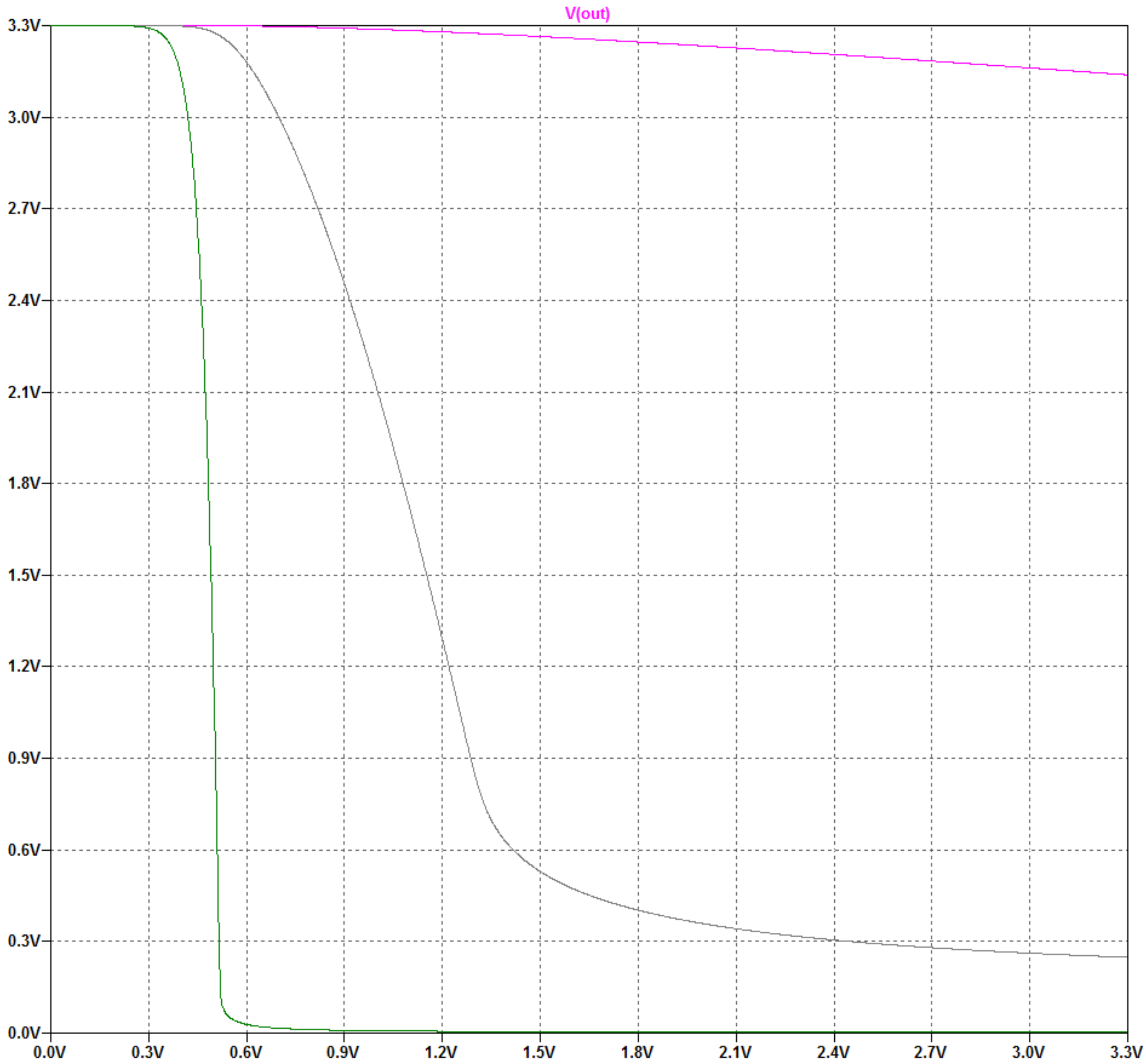


Figure 26 - Common source amplifiser



The Vgs resulting in an output DC i.e. V(out) around Vdd/2 gives best headroom for output voltage swing.

The gain of the amplifier is:

–gm*Rd

And

gm=Id/Vgs

6 Appendix

6.1 Python code

```
# process parameters
u_n = 0.0580        # electron mobility [m^2/Vs]
t_ox = 0.011e-6     # oxide thickess [m]
e_0 = 8.854e-12     # permattivity of free air [F/m]
e_ox = 3.97 * e_0   # oxide permattivity [F/m]
C_ox = e_ox/t_ox    # gate oxide capacitance per unit area [F/m2]
k_n = u_n*C_ox       # transconductance [A/V^2]
k_n = 190e-6    # from text
#k_n = 43.2e-6     # from LTspice
print("Cox: %e" % C_ox)
print("Kn: %e" % k_n)
#return

# transistor parameters
W = 1e-6
L = 1e-6
lmbda = 0.16 # from text
#lmbda = 0.04 # from LTspice
V_TO = 0.57

# Sedra Smith page 369
#lmbda = 0.0
#V_TO = 1
#k_n = 0.5e-3

print("Kn(W/L): %e" % (k_n*W/L))


U_T = 26e-3
n = 1
V_DS = var("V_DS")
V_GS = var("V_GS")
#   I_D = var("I_D")

#   gm = I_D/(n*U_T)

# weak inversion (triode region)
#  k_n*W/L*((V_GS-V_TO)*V_DS-1/2*V_DS**2) # from book
#   I_D = 2*n*k_n*(W/L)*U_T**2*exp((V_GS-V_TO)/(n*U_T))

# strong inversion (saturation region)
#  I_D = 1/2 * k_n * W/L * (V_GS-V_TO)**2*(1+lmbda*V_DS)

Vdd = 3.3
colors = ['r', 'b', 'g', 'y', 'c', 'k', 'm']

def i_d(v_gs, v_ds, k_n=k_n, lmbda=lmbda, v_to=V_TO):
    V_P = (v_gs-v_to)/n
    I_S = 2*n*k_n*W/L*U_T**2
    I_F = I_S*log(1+exp((V_P)/(2*U_T)))**2*(1+lmbda*v_ds)
    I_R = I_S*log(1+exp((V_P-v_ds)/(2*U_T)))**2*(1+lmbda*v_ds)
    I_D = I_F-I_R
    return I_D

def id_vgs(name, v_ds, v_gs, logy=False):
    id_cutoff = []
    id_triode = []
    id_saturation = []
    triode=True
    for V_GS in v_gs:
        I_D = i_d(V_GS, v_ds)
        if V_GS < V_TO:
            region = "cutoff"
        elif v_ds >= (V_GS-V_TO):
            region = "saturation"
        else:
            region = "triode"
```

```
        if region == "cutoff":
            id_cutoff.append((V_GS, I_D))
        elif region == "triode":
            id_triode.append((V_GS, I_D))
        elif region == "saturation":
            id_saturation.append((V_GS, I_D))

    plt.plot([x[0] for x in id_cutoff], [x[1] for x in id_cutoff], label='$cutoff [V_{GS} < V_{TO}]$', color='red')
    plt.plot([x[0] for x in id_triode], [x[1] for x in id_triode], label='$triode [V_{DS} < (V_{GS}-V_{TO})]$', color='green')
    plt.plot([x[0] for x in id_saturation], [x[1] for x in id_saturation], label='$saturation [V_{DS} >= (V_{GS}-V_{TO})]$', color='blue')
    plt.ylabel('$I_{D}$')
    plt.xlabel('$V_{GS}$')
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("$I_D vs V_{GS} for V_{DD}=%.2f$" % v_ds)
    plt.savefig("%s.png" % name)
    plt.show()
    plt.close()

def id_vds(name, v_ds, v_gs, logy=False):
    id_trans = []
    id_knee = []
    for V_GS in v_gs:
        id_triode = []
        id_saturation = []
        triode=True
        knee_found = False
        for V_DS in v_ds:
            I_D = i_d(V_GS, V_DS)
            if V_GS < V_TO:
                if V_DS > 4*U_T:
                    if not knee_found:
                        id_knee.append((V_DS, I_D))
                        knee_found = True
            if V_DS < (V_GS-V_TO):
                triode = True
            else:
                if triode:
                    if V_DS > 0:
                        id_triode.append((V_DS, I_D))
                        id_trans.append((V_DS, I_D))
                    triode=False
            if triode:
                id_triode.append((V_DS, I_D))
            else:
                id_saturation.append((V_DS, I_D))

        plt.plot([x[0] for x in id_triode], [x[1] for x in id_triode], '-', color='grey')
        plt.plot([x[0] for x in id_saturation], [x[1] for x in id_saturation], label='$V_G = {i}$'.format(i=V_GS))
    plt.plot([x[0] for x in id_trans], [x[1] for x in id_trans], '--', marker='o', color='grey')
    plt.plot([x[0] for x in id_knee], [x[1] for x in id_knee], ' ', marker='o', color='grey')

    plt.ylabel('$I_{D}$')
    plt.xlabel('$V_{DS}$')
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("$I_D vs V_{DS}$")
    plt.savefig("%s.png" % name)
    plt.show()
    plt.close()

def plot(name, x, y, logy=False):
    plt.plot(x,y)
    plt.ylabel('$I_{D}$')
    plt.xlabel('$V_{DS}$')
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("$I_D vs V_{DS}$")
    plt.savefig("%s.png" % name)
```

```
    plt.show()
    plt.close()

def plot_dataset(xlabel, ylabel, filename=None, x=None, y=None, logy=False):
    if filename:
        dataset = CDataSet(variables=[xlabel, ylabel],filename=filename)
        (x,y) = dataset.get_xy(xlabel,ylabel)
    elif x and y is not None:
        pass
    plt.plot(x,y, label=ylabel)
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("%s vs %s" % (ylabel,xlabel))
    plt.savefig("%s%s.png" % (os.path.splitext(os.path.basename(filename))[0], "-logy" if logy else ""))
    plt.show()
    plt.close()

def plot_dataset_mod(xlabel, ylabel, filename=None, x=None, y=None, logy=False):
    if filename:
        dataset = CDataSet(variables=[xlabel, ylabel],filename=filename)
        (x,y) = dataset.get_xy(xlabel,ylabel)
    elif x and y is not None:
        pass
    y_mod = []
    x_mod = []
    for i in range(len(y)):
        if x[i] > 1:
            y_mod.append(y[i]/((x[i]-0.57)*0.57))
            x_mod.append(x[i])
    plt.plot(x_mod,y_mod, label=ylabel)
    plt.ylabel("$k_n$")
    plt.xlabel(xlabel)
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("$k_n$ vs %s" % (xlabel))
    plt.savefig("%s%s_mod.png" % (os.path.splitext(os.path.basename(filename))[0], "-logy" if logy else ""))
    plt.show()
    plt.close()

def plot_stepped_dataset(xlabel, ylabel, filename, logy=False):
    dataset = CDataSet()
    dataset.from_ltspice_stepped_file(filename)
    print dataset.variables
    x = dataset.get(xlabel)
    for variable in dataset.variables:
        if variable != xlabel:
            y = dataset.get(variable)
            plt.plot(x,y, label=variable)

    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("%s vs %s" % (ylabel,xlabel))
    plt.savefig("%s%s.png" % (os.path.splitext(os.path.basename(filename))[0], "-logy" if logy else ""))
    plt.show()
    plt.close()

def plot_compare(xlabel, ylabel, filename, outfilename, k_n=k_n, lmbda=lmbda, v_to=V_TO, logy=False):
    v_ds = 3.3
    dataset = CDataSet(variables=[xlabel, ylabel],filename=filename)
    (x,y1) = dataset.get_xy(xlabel,ylabel)
    y2=[]
    for v_gs in x:
        y2.append(i_d(v_gs, v_ds, k_n=k_n, lmbda=lmbda, v_to=V_TO))

    plt.plot(x,y1, label='simulated')
    plt.plot(x,y2, label='calculated')
```

```
    plt.ylabel('$I_{D}$')
    plt.xlabel('$V_{GS}$')
    if logy:
        plt.semilogy()
    plt.legend(loc='best')
    plt.grid()
    plt.title("Comparing simulated and calculated $I_{D}$ for $V_{DS}=%.2f$" % v_ds)
    plt.savefig("%s.png" %outfilename)
    plt.show()
    plt.close()


def main():
    if 0:
        v_ds = Vdd
        v_gs = [v/1000 for v in range(0, int(Vdd*1000), 10)]
        id_vgs("task1-1a", v_ds, v_gs, logy=False)
        id_vgs("task1-1b", v_ds, v_gs, logy=True)
    if 0:
        v_ds = V_TO
        id_vgs("task1-2a", v_ds, v_gs, logy=False)
        id_vgs("task1-2b", v_ds, v_gs, logy=True)

    if 0:
        v_ds = [v/1000 for v in range(0, int(Vdd*1000), 100)]
        v_gs = [v/1000 for v in range(int(V_TO*1000), int(Vdd*1000), 500)]
        id_vds("task1-3", v_ds, v_gs, logy=False)
    if 0:
        v_ds = [v/1000 for v in range(0, int(Vdd*1000), 10)]
        v_gs = [v/1000 for v in range(int(V_TO*0.9*1000), int(V_TO*1000), 10)]
        id_vds("task1-4", v_ds, v_gs, logy=False)

    if 0:
        plot_dataset(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-1.txt")
        plot_dataset(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-1.txt", logy=True)

    if 0:
        plot_dataset(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-2.txt")
        plot_dataset(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-2.txt", logy=True)
    if 0:
        plot_stepped_dataset(xlabel="vdd", ylabel="Id(M1)", filename="mandatory2-2-3.txt", logy=False)
        plot_stepped_dataset(xlabel="vdd", ylabel="Id(M1)", filename="mandatory2-2-4.txt", logy=False)
    if 0:
        plot_compare(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-1.txt", outfilename="mandatory2-3-1")
    if 0:
        plot_dataset_mod(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-2.txt")
    if 1:
        plot_compare(xlabel="vgs", ylabel="Id(M1)", filename="mandatory2-2-1.txt", outfilename="mandatory2-3-1b", k_n=0.065e-
3, lmbda=0.015, v_to=0.57)

if __name__ == '__main__':
    main()
```

6.2 0.35uM MOSFET model

```
.MODEL 3 NMOS                                           LEVEL    =   49
+VERSION    =    3.1      TNOM     =    27       TOX      =   '7.8E-9/proc_delta'
+XJ      =  1E-07      NCH      =  2.18E+17  VTH0     =  '0.48+vt_shift'
+K1      =  6.07E-01   K2       =  1.24E-03  K3       =  9.68E+01
+K3B     =  -9.84E+00           W0       =  2.02E-05  NLX      =   1.62E-07
+DVT0W   =  0          DVT1W    =  0         DVT2W    =  0
+DVT0    =  2.87E+00   DVT1     =  5.86E-01  DVT2     =  -1.26E-01
+U0      =  '360*proc_delta*proc_delta'  UA  =  -8.48E-10  UB  =  2.27E-18
+UC      =  3.27E-11   VSAT     =  1.87E+05  A0       =  1.22E+00
+AGS     =  2.06E-01   B0       =  9.60E-07  B1       =  4.95E-06
+KETA    =  -1.67E-04  A1       =  0         A2       =  3.49E-01
+RDSW    =  8.18E+02   PRWG     =  2.35E-02  PRWB     =  -8.12E-02
+WR      =  9.98E-01   WINT     =  1.55E-07  LINT     =  4.51E-10
+XL      =  -5.00E-08  XW       =  1.50E-07  DWG      =  -4.27E-09
+DWB     =  4.07E-09   VOFF     =  -4.14E-02 NFACTOR           =   1.61E+00
+CIT     =  0          CDSC     =  2.39E-04  CDSCD    =  0.00E+00
+CDSCB   =  0          ETA0     =  1         ETAB     =  -1.99E-01
+DSUB    =  1          PCLM     =  1.32E+00  PDIBLC1  =  2.42E-04
+PDIBLC2    =  8.27E-03   PDIBLCB  =  -9.99E-04  DROUT  =  9.72E-04
+PSCBE1     =  7.24E+08   PSCBE2   =  9.96E-04   PVAG   =  1.00E-02
+DELTA   =  1.01E-02   RSH      =  3.33E+00  MOBMOD            =   1
+PRT     =  0          UTE      =  -1.5      KT1      =  -1.11E-01
+KT1L    =  0          KT2      =  2.22E-02  UA1      =  4.34E-09
+UB1     =  -7.56E-18  UC1      =  -5.62E-11 AT       =  3.31E+04
+WL      =  0          WLN      =  9.95E-01  WW       =  0
+WWN     =  1.00E+00   WWL      =  0         LL       =  0
+LLN     =  1          LW       =  0         LWN      =  1
+LWL     =  0          CAPMOD   =  2         XPART    =  0.5
+CGDO    =  2.76E-10   CGSO     =  2.76E-10  CGBO     =  1.00E-12
+CJ      =  '9e-4/proc_delta'     PB       =  7.95E-01  MJ  =  3.53E-01
+CJSW    =  '2.8e-10/proc_delta'  PBSW     =  7.98E-01  MJSW  =  1.73E-01
+CJSWG   =  1.81E-10   PBSWG    =  7.96E-01  MJSWG    =  1.74E-01
+CF      =  0          PVTH0    =  -1.80E-02 PRDSW    =  -7.56E+01
+PK2     =  4.48E-05   WKETA    =  -1.33E-03 LKETA    =  -8.91E-03
```