MOS_final_DE

October 27, 2020

1 Elektronische Bauelemente: MOS-Transistor

Dies ist eine interaktive Oberfläche zur Erkundung verschiedener Zusammenhänge des MOS-Transistors auf Basis der Vorlesung

1.1 Struktur

Abb. 1a) zeigt den schematischen Querschnitt eines n-Kanal MOS-Transistors mit den wichtigsten Abmessungen für nach folgende Analyse: * Die (metallurgische) Kanallänge L ist durch den Abstand der pn-Übergänge (BSu.BD) an der Si Oberfläche unter dem Gate (G) definiert. * Die S-/D-Gebiete sind bei einem n-Kanal-Transistor n^+-dotiert und dienen zum elektrischen Anschluss des Kanals. Die Eindringtiefe der Gebiete und der damit verbundene pn-Übergang zum Substrat ist mit x_j bezeichnet. Die laterale Ausdebunung S-/D-Gebiete (Unterdiffusion) unter das Gate, y_j, bestimmt die Kanallänge. Die Überlappung des Gates über die S/D-Zonen ist erforderlich für die Funktionsweise eines MOS-Transistors! * Die Kanalweite W erstreckt sich in z-Richtung.

(a)

(b)

Abb. 1: a) Schematischer Querschnitt eines n-Kanal MOS-Transistors, b) Vorspannung eines n-Kanal MOS-Transistors mit Bulk als Referenzelektrode.

1.2 Arbeitspunkteinstellung

Dazu ist in Abb. 1b) die übliche Spannungen eines n-Kanal Transistors gezeigt, wobei das Substrat (Bulk) aus Symmetriegründen als Referenz gewählt wurde.

- Die Spannung U_GB zwischen Gate und Bulk ist positiv, um Elektronen an die Oberfläche zu ziehen was eine Ladungsträgerinversion bewerkt.
- Die Spannungen U_SB und U_DB über den pn-Übergängen (BSu.BD) sind größer oder gleich Null, sodass die Übergänge gesperrt sind und ein Stromfluss über die entsprechenden Raumladungszonen verhindert wird.
- Die Raumladungszonen des BS- und BD-Übergangs gehen in die Verarmungszone unter dem Gate über, die durch die Spannung U_GB (s. MIS-Struktur) gebildet wird.
- Das Koordinatensystem ist für die Stromberechnung gezeichnet. Gemäß der technischen Konvention wird I_D positiv gezählt.

In dieser Demonstration wird die Source als Referenzpunkt mit $U_\mathrm{S} = 0$ verwendet und die Gate-, Drain- und Bulk-Spannungen werden als U_GS, U_DS und U_SB bezeichnet.

1.3 Schwell- und Sättigungsspannung

Zwei Spannungen definieren verschiedene Arbeitsbereiche des MOS-Transistors

1.3.1 Schwellspannung U_{th}

Gilt $U_{\mathrm{GS}} \geq U_{\mathrm{th}}$ beginnt der MOS-Transistors zu leiten (in diesem vereinfachten Modell). U_{th} hängt von U_{SB} ab, sowie den Struktur- und Materialparametern ab, Es gilt:

$$U_{\mathrm{th}} = U_{\mathrm{th0}} + \gamma \left(\sqrt{\phi_h - U_{\mathrm{SB}}} - \sqrt{\phi_h} \right),$$

mit $U_{\mathrm{th0}} = 0.7$ V bei $U_{\mathrm{SB}} = 0$ V und γ bezeichtnet den Body-koeffizienten:

$$\gamma = \frac{1}{\bar{C_{ox}}} \sqrt{2\epsilon_0 \epsilon_{r,si} q N_{\mathrm{A}}^-}.$$

.

$\bar{C_{ox}}$ ist die flächenbezogene Oxidkapazität, ϵ_0 ist die absolute Dielektrizitätskonstante, $\epsilon_{r,si}$ ist die relative Dielektrizitätskonstante vin Si und N_{A}^- ist die Bulkdotierung des p-Substrat. Die Oxidkapazität $\bar{C_{ox}}$ ist gegeben durch:

$$\bar{C_{ox}} = \frac{\epsilon_0 \epsilon_{r,ox}}{d_{ox}},$$

mit $\epsilon_{r,ox}$ die relative Dielektrizitätskonstante von Gate-Oxids (hier SiO_2) und d_{ox} die Oxiddicke ist. $\phi_h \geq 2\phi_F$ und das Fermi-Potential ϕ_F ist gegeben durch:

$$\phi_F = U_{\mathrm{T}} \ln \left| \frac{N_{\mathrm{A}}^-}{n_{\mathrm{i}}} \right|.$$

Hier ist $U_{\mathrm{T}} = 25,9$ mV die thermische Votlage und $n_{\mathrm{i}} = 1,45 \times 10^{10} \mathrm{cm}^{-3}$ ist die intrinische Ladungsträgerkonzentration (bei 300 K).

1.3.2 Sättigungsspannung

Die Sättigungsspannung $U_{\mathrm{DS,sat}}$ ist definiert als die Drain-Spannung, bei der MOS-Transistoren vom linearen Bereich in den Sättigungsbereich übergehen. Die Definition jeder Region und der Ausdruck für den Drainstrom in jeder Region werden weiter unten diskutiert. Ähnlich wie die Schwellenspannung hängt die Sättigungsspannung von den Struktur- und Materialeigenschaften sowie von der Gate-Source-Spannung und der Bulk-Spannung ab. Sie ist gegeben durch:

$$U_{\mathrm{DS,sat}} = \frac{U_{\mathrm{GS}} - U_{\mathrm{th}}}{1 + a_{th}}.$$

wobei a_{th} die Steigung von $-\bar{Q_d}/\bar{C_{ox}}$ gegenüber dem Kanalpotential V_C ist, und sie ist gegeben durch:

$$a_{th} = \left. \frac{d(-\bar{Q_d}/\bar{C_{ox}})}{dV_c} \right|_{V_{c0}=U_{\mathrm{SB}}} = \frac{\gamma}{2\sqrt{\phi_h + U_{\mathrm{SB}}}}.$$

Probieren Sie die Schieberegler für die Oxiddicke und die Akzeptorkonzentration aus und betrachten Sie die Änderungen der Haltespannung und der Sättigungsspannung

[1]:
```
%matplotlib widget

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math

from cycler import cycler
from matplotlib.lines import Line2D

from ipywidgets import (
    HBox, VBox, FloatText, FloatRangeSlider, IntSlider, FloatSlider,␣
 ↪GridspecLayout, GridBox, Button, ButtonStyle, Dropdown,
    Play, jslink, FloatLogSlider, interactive_output, HTML, HTMLMath, Layout,␣
 ↪Checkbox, Label, Text, AppLayout
)
```

[2]:
```
# global constants
eps0        = 8.854e-12     # [As/(Vm)] - permittivity of free space
q           = 1.602e-19     # [As]      - electronic charge
kb          = 1.38e-23      # [J/K]     - Boltzmann's constant

# global fixed parameters
T           = 300           # [K]       - temperature
UT          = kb*T/q        # [V]       - thermal voltage
epsr_si     = 11.7          # [ ]       - relative permittivity of Si
epsr_ox     = 3.73          # [ ]       - relative permittivity of SiO2
ni          = 1.07e10       # [cm^-3]    - intrinsic carrier concentration in␣
 ↪Si (at 300 K)
Lch         = 1e-6        # [m]       - channel length
mu_n        = 0.1           # [m^2/(Vs)]- mobility of electrons in Si
Uth0        = 0.7           # [V]       - threshold voltage
UA          = 50            # [V]       - early voltage

# Bias point initialization
Ugs_sweep   = np.linspace(0,2.5,101)     # [V]   - gate-source voltage
Uds_sweep   = np.linspace(0,5,101)      # [V]   - drain-source voltage
Usb_sweep   = np.linspace(0,5,101)      # [V]   - source-substrate voltage
```

[3]:
```
### Style definition
# ------------------
style = {'description_width': 'initial'}


### Variables and their slider definitions
```

```
# ----------------------------------------
# Width (W)
W_slider = FloatSlider(
    value=5,
    min=1,
    max=10,
    step=0.5,
    description=r'\(W/\rm{\mu m}\)',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.2f',
    # style=style
)

# Oxide thickness (d_ox)
dox_slider = FloatSlider(
    value=10,
    min=1,
    max=100,
    step=1,
    description=r'\(d_{\rm{ox}}/\rm{nm}\)',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
    # style=style
)

# Acceptor concentration (N_A)
NA_slider = FloatLogSlider(
    value=5e16,
    base=10,
    min=15, # max exponent of base
    max=17, # min exponent of base
    step=1e-1, # exponent step
    description=r'\(N_{\rm{A}}^{-}/\rm{cm^{-3}}\)',
    continuous_update=False,
    # style=style
)

# source-substrate voltage (U_SB) for plotting Uth vs Usb
Usb_slider = FloatSlider(
    value=0,
    min=np.min(Usb_sweep),
```

```
    max=np.max(Usb_sweep),
    step=0.1,
    description='$U_{\mathrm{SB}}$/V',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
#     layout={'width':'40%'}
    # style=style
)
```

[4]:
```
### function definitions
# ----------------------
def calc_Cox(dox):
    """Function to oxide capacitance C_ox
    Input
    ------
    d_ox    : float
        oxide thickness.
    Output
    ---------
    C_ox    : float
        oxide capacitance per unit area.
    """
    d_ox_m = dox*1e-9
    C_ox = eps0*epsr_ox/d_ox_m
    return C_ox

def calc_kn(W,L,mu_n,dox):
    """Function to calculate the drain current prefactor.
    Input
    ------
    W       : float
        gate width.
    L       : float
        channel length.
    mu_n    : float
        electron mobility.
    Output
    ---------
    km      : float
        drain current prefactor (per unit area).
    """
    C_ox = calc_Cox(dox)
    W_m = W*1e-6
    kn = W_m/L*mu_n*C_ox
```

```
    return kn

def calc_gamma(NA,dox):
    """Function to calculate gamma, i.e. body effect coefficient (see 2.5.19 in␣
 ↪[1]).
    Input
    ------
    None
    Output
    ---------
    gamma   : float
        body effect coefficient.
    """
    C_ox = calc_Cox(dox)

    gamma = (1/C_ox)*np.sqrt(2*eps0*epsr_si*q*NA*1e6)
    return gamma

def calc_phib(NA):
    """Function to calculate phi_b, i.e. the upper limit of weak inversion (see␣
 ↪2.5.25 in [1]).
    Input
    ------
    None
    Output
    ---------
    phi_b   : float
        the upper limit of weak inversion.
    """
    phi_f = UT*np.log(NA/ni)

    phi_b = 2*phi_f + 2*UT
    return phi_b

def calc_ath(Usb,NA,dox):
    """Function to calculate a_th, i.e. the slope of extrapolated threshold␣
 ↪voltage Uth vs Usb (see 4.4.33b in [1]).
    Input
    ------
    Usb     : float
        source-substrate voltage.
    Output
    ---------
    a_th    : float
        slope of extrapolated threshold voltage Uth vs Usb.
    """
    gamma   = calc_gamma(NA,dox)
```

```
    phi_b   = calc_phib(NA)

    a_th = 0.5*gamma/np.sqrt(phi_b+Usb)
    return a_th

def calc_Uth(Uth0,Usb,NA,dox):
    """Function to calculate Uth, i.e. gate-source extrapolated threshold␣
 ↪voltage (see 4.4.26b in [1]).
    Input
    ------
    Uth0     : float
        threshold voltage.
    Usb      : float
        source-substrate voltage.
    Output
    ---------
    Uth    : float
        gate-source extrapolated threshold voltage.
    """
    gamma   = calc_gamma(NA,dox)
    phi_b   = calc_phib(NA)

    Uth = Uth0 + gamma*(np.sqrt(phi_b+Usb) - np.sqrt(phi_b))
    return Uth

def calc_Udssat(Ugs,Usb,NA,dox):
    """Function to calculate the saturation voltage.
    Input
    ------
    Ugs      : ndarray
        gate-source voltage.
    Usb      : float
        source-substrate voltage.
    Output
    ---------
    Uds_sat : ndarray
        Saturation votlage
    """

    a_th    = calc_ath(Usb,NA,dox)
    Uth     = calc_Uth(Uth0,Usb,NA,dox)

    Uds_sat = (Ugs-Uth)/(1+a_th)
    Uds_sat = Uds_sat.clip(min=0)
    return Uds_sat
```

```
def update_Uth_Udssat(*args,**kwargs):
    """Function to update Uth, gmb vs Usb plot.
    """
    Usb_op  = Usb_slider.value
    NA = NA_slider.value
    dox = dox_slider.value
    index = np.argmin(np.abs(Usb_sweep-Usb_op))

    # calculate Uth and gmb
    Uth = calc_Uth(Uth0,Usb_sweep,NA,dox)
    Uds_sat = calc_Udssat(Ugs_sweep,Usb_op,NA,dox)

    # update Uth and line
    Uth_line[0][0].set_ydata(Uth)
    Uth_line[1][0].set_xdata(Usb_op)
    Uth_line[1][0].set_ydata(Uth[index])

    # update gmb and line
    Udssat_line[0][0].set_ydata(Uds_sat)
#     Udssat_line[1][0].set_xdata(Usb_op)
#     Udssat_line[1][0].set_ydata(Uds_sat[index])

    # update limit
    ax_Uth.set_ylim(ymax=np.amax(Uth)*1.25)
    Udssat_max = np.amax(Uds_sat)
    if Udssat_max>0:
        ax_Udssat.set_ylim(ymax=Udssat_max*1.25)

    fig_Uth_Udssat.canvas.draw()



plt.ioff()
plt.grid(True)

custom_cycler = (cycler(color=list('rgb')) *
       cycler(linestyle=['-', '--', '-.']))

plt.rc('lines', linewidth=1)
plt.rc('axes', prop_cycle=custom_cycler)
plt.rcParams['axes.grid'] = True
plt.rcParams['grid.alpha'] = 1.5
plt.rcParams['grid.color'] = "#cccccc"

figW = 8
figH = 2.25*1.5
```

```
### Begin Uth vs Usb and gmb vs Usb fig
# ------------------------------------
fig_Uth_Udssat = plt.figure(figsize=(figW, figH),constrained_layout=True)
gs = fig_Uth_Udssat.add_gridspec(1, 2, hspace=0.25, wspace=0.25,␣
 ↪height_ratios=[1],bottom=0.2)
(ax_Uth, ax_Udssat) = gs.subplots(sharex=False, sharey=False)
# fig_ID_gds.suptitle('$I_{\mathrm{D}}(U_{\mathrm{DS}})$'+' und␣
 ↪'+'$g_{\mathrm{ds}}(U_{\mathrm{GS}})$')
fig_Uth_Udssat.canvas.header_visible = False
fig_Uth_Udssat.canvas.layout.min_width = '400px'
fig_Uth_Udssat.canvas.toolbar_visible = True
fig_Uth_Udssat.canvas.capture_scroll = True

Usb_op  = Usb_slider.value
NA = NA_slider.value
dox = dox_slider.value
index = np.argmin(np.abs(Usb_sweep-Usb_op))

# calculate Uth and gmb
Uth = calc_Uth(Uth0,Usb_sweep,NA,dox)
Uds_sat = calc_Udssat(Ugs_sweep,Usb_op,NA,dox)

# plot Uth and gmb
Uth_line=[]
Uth_line.append(ax_Uth.plot(Usb_sweep, Uth))
Uth_line.append(ax_Uth.plot(Usb_slider.
 ↪value,Uth[index],color='b',lw=2,marker='o',ls=':'))

Udssat_line=[]
Udssat_line.append(ax_Udssat.plot(Ugs_sweep, Uds_sat))
# Udssat_line.append(ax_Udssat.plot(Usb_slider.
 ↪value,Uds_sat[index],color='b',lw=2,marker='o',ls=':'))

# set legend and label
ax_Uth.set_xlabel('$U_{\mathrm{SB}}\mathrm{/V}\;→$')
ax_Uth.set_ylabel('$U_{\mathrm{th}}\mathrm{/V}\;→$')

ax_Udssat.set_xlabel('$U_{\mathrm{GS}}\mathrm{/V}\;→$')
ax_Udssat.set_ylabel('$U_{\mathrm{DS,sat}}\mathrm{/V}\;→$')

# set title
ax_Uth.set_title('$U_{\mathrm{th}}(U_{\mathrm{SB}})$')
ax_Udssat.set_title('$U_{\mathrm{DS,sat}}(U_{\mathrm{GS}})$')

# set limits
ax_Uth.set_xlim(0, np.max(Uds_sweep))
ax_Uth.set_ylim(ymin=0)
```

```
ax_Udssat.set_xlim(0, np.max(Ugs_sweep))
ax_Udssat.set_ylim(ymin=0) # ,ymax=np.amax(gmb)*1.25e3

# draw figure
fig_Uth_Udssat.canvas.draw()


# observe sliders
# ---------------
dox_slider.observe(update_Uth_Udssat)
NA_slider.observe(update_Uth_Udssat)
Usb_slider.observe(update_Uth_Udssat)

param_layout = VBox([
                    HBox([Label('Veränderbare parameters für␣
↪'+'$U_{\mathrm{th}}$'+' and '+'$U_{\mathrm{DS,sat}}$')]),
                    HBox([dox_slider]),
                    HBox([NA_slider]),
                    HBox([Usb_slider]),
                ],layout=Layout(justify_content = 'center', align_items =␣
↪'center')) # ,layout=Layout(width='32%')

AppLayout(header=None,
          left_sidebar=param_layout,
          center=None,
          right_sidebar=HBox([fig_Uth_Udssat.canvas]),
          footer=None,
          pane_widths=['325px', '0px', '900px'],
          pane_heights=['1px', 5, '1px']
         )
```

```
AppLayout(children=(VBox(children=(HBox(children=(Label(value='Veränderbare␣
↪parameters für $U_{\\mathrm{th}}$ …
```

1.4 Drainstrom

Der Drainstrom in MOS-Transistoren ist wie folgt definiert:

$$I_\mathrm{D} = \begin{cases} 0, & \text{für } U_\mathrm{GS} < U_\mathrm{th} \\ k'_n \left[(U_{GS} - U_{th}) U_{DS} - (1 + a_{th}) \frac{U_{DS}^2}{2} \right], & \text{für } U_\mathrm{GS} \geq U_\mathrm{th}, U_\mathrm{DS} \leq U_\mathrm{DS,sat} \\ k'_n \left[\frac{1-k_{clm}}{2(1+a_{th})} (U_\mathrm{GS} - U_\mathrm{th})^2 + \frac{k_{clm}}{2} (U_\mathrm{GS} - U_\mathrm{th}) U_\mathrm{DS} \right], & \text{für } U_\mathrm{GS} \geq U_\mathrm{th}, U_\mathrm{DS} > U_\mathrm{DS,sat} \end{cases}$$

I_D hängt von den Strukturparametern wie Kanalbreite W, Kanallänge L und den Materialparametern (Elektronenbeweglichkeit μ_n, flächenbezogene Oxidkapazität C_{ox}) ab. Diese Parameter sind in k'_n enthalten:

$$k'_n = \frac{W}{L}\mu_n C'_{ox}.$$

Diese Gleichungen sind nur genau für Langkanal-MOSFETs ($L \geq 1\ \mu$m). Die Elektronenmobilität μ_n hängt von der Akzeptorkonzentration N_A^- im p-Substrat ab. Der in der Vorlesung diskutierte Kanallängenmodulationseffekt wird durch k_{clm} modelliert und ist gegeben durch:

$$K_{clm} = \frac{U_{\mathrm{DS,sat}}}{U_\mathrm{A} + U_{\mathrm{DS,sat}}},$$

wobei U_A die Early-Spannung ist und im Beispiel 50 V betrachtet beträgt.

1.5 Kleinsignalparameter

1.5.1 1. Ausgangsleitwert

$$g_\mathrm{ds} = \left.\frac{\partial I_\mathrm{D}}{\partial U_\mathrm{DS}}\right|_{U_\mathrm{GS},U_\mathrm{SB}}$$

1.5.2 2. Transfersteilheit

$$g_\mathrm{m} = \left.\frac{\partial I_\mathrm{D}}{\partial U_\mathrm{GS}}\right|_{U_\mathrm{DS},U_\mathrm{SB}}$$

1.5.3 3. Back-Gate-Steilheit

$$g_\mathrm{mb} = \left.\frac{\partial I_\mathrm{D}}{\partial U_\mathrm{BS}}\right|_{U_\mathrm{GS},U_\mathrm{DS}} = \left(\frac{\partial I_\mathrm{D}}{\partial U_\mathrm{th}}\right)\left(-\frac{\partial U_\mathrm{th}}{\partial U_\mathrm{SB}}\right)$$

Ändern Sie nun die unten stehenden Schieberegler, um die Änderung der Ausgangs- und Übertragungseigenschaften, sowie der Kleinsignalparameter zu betrachten. (Der Punkt in der Darstellung $I_\mathrm{D}(U_\mathrm{GS})$ bezeichnet die Schwellenspannung U_th)

[5]:
```
%matplotlib widget

### Variables and their slider definitions
# ----------------------------------------
# Oxide thickness (d_ox)
dox_slider2 = FloatSlider(
    value=10,
    min=1,
    max=100,
    step=1,
    description=r'\(d_{\rm{ox}}/\rm{nm}\)',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
    # style=style
```

```
)

# Acceptor concentration (N_A)
NA_slider2 = FloatLogSlider(
    value=5e16,
    base=10,
    min=15, # max exponent of base
    max=17, # min exponent of base
    step=1e-1, # exponent step
    description=r'\(N_{\rm{A}}^{-}/\rm{cm^{-3}}\)',
    continuous_update=False,
    # style=style
)

# Terminal voltage and its slider definitions
# Gate-source voltage (U_GS) for Output characteristics
Ugs_slider = FloatRangeSlider(
    value=[1, 2.5],
    min=0.5,
    max=4.0,
    step=0.1,
    description='Parameter: $U_{\mathrm{GS}}$/V',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
    layout={'width':'70%'},
    style=style
)
# Drain-source voltage (U_DS) for transfer characteristics
Uds_slider = FloatRangeSlider(
    value=[0.1, 3],
    min=0,
    max=5.0,
    step=0.1,
    description='Parameter: $U_{\mathrm{DS}}$/V',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
    layout={'width':'70%'},
    style=style
)
# source-substrate voltage (U_SB) for plotting Uth vs Usb
Usb_slider2 = FloatSlider(
```

```
    value=0,
    min=np.min(Usb_sweep),
    max=np.max(Usb_sweep),
    step=0.1,
    description='$U_{\mathrm{SB}}$/V',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
#     layout={'width':'40%'}
    # style=style
)

Uds_drop = Dropdown(
    options=np.round(np.linspace(0,5,51),decimals=1),
    value=1.5,
    description='$U_{\mathrm{DS}}$/V: ',
    readout_format='.1f',
    layout={'width':'150px'},
    continuous_update=False,
    disabled=False,
)

Ugs_drop = Dropdown(
    options=np.round(np.linspace(0,5,51),decimals=1),
    value=1.5,
    description='$U_{\mathrm{GS}}$/V: ',
    readout_format='.1f',
    layout={'width':'150px'},
    continuous_update=False,
    disabled=False,
)

### function definitions
# ----------------------
def calc_ID(Ugs,Uds,Usb):
    """Function to calculate the drain current ID for single op point (see 4.4.
 ↪30 in [1]).
    Input
    ------
    Ugs     : float
        gate-source voltage.
    Uds     : float
        drain-source voltage.
    Usb     : float
        source-substrate voltage.
```

```
    Output
    ---------
    ID      : float
        Drain current.
    """

    NA = NA_slider2.value
    dox = dox_slider2.value
    W = W_slider.value

    kn      = calc_kn(W,Lch,mu_n,dox)
    a_th    = calc_ath(Usb,NA,dox)
    Uth     = calc_Uth(Uth0,Usb,NA,dox)

    Uds_sat = (Ugs-Uth)/(1+a_th)
    k_clm   = Uds_sat/(UA+Uds_sat)

    if Ugs < Uth:
        ID = 0
    else:
        if Uds <= Uds_sat:
            ID = kn*((Ugs-Uth)*Uds - 0.5*(1+a_th)*Uds**2)
        else:
            ID = kn*((1-k_clm)/(2*(1+a_th))*(Ugs-Uth)**2 + 0.
↪5*k_clm*(Ugs-Uth)*Uds)

    # print(ID)
    return ID

def calc_ID_gds(Ugs,Uds,Usb):
    """Function to calculate the output characteristics and output conductance
    Input
    ------
    Ugs     : list
        gate-source voltage.
    Uds     : ndarray
        drain-source voltage.
    Usb     : float
        source-substrate voltage.
    Output
    ---------
    ID_out  : ndarray
        Drain current (output characteristics).
    gds     : ndarray
        Output conductance.
    """
```

```
    ID_out  = np.zeros((len(Ugs),len(Uds)))
    gds     = np.zeros((len(Ugs),len(Uds)))

    for nUgs in np.arange(np.size(Ugs)):
        for nUds in np.arange(np.size(Uds)):
            ID_out[nUgs,nUds] = calc_ID(Ugs[nUgs],Uds[nUds],Usb)
        gds[nUgs,:] = np.gradient(ID_out[nUgs,:])/np.gradient(Uds)

    return ID_out,gds

def calc_ID_gm(Ugs,Uds,Usb):
    """Function to calculate the transfer characteristics and transconductance
    Input
    ------
    Ugs     : ndarray
        gate-source voltage.
    Uds     : list
        drain-source voltage.
    Usb     : float
        source-substrate voltage.
    Output
    ---------
    ID_trans    : ndarray
        Drain current (transfer characteristics).
    gm          : ndarray
        transconductance.
    """

    ID_trans    = np.zeros((len(Uds),len(Ugs)))
    gm          = np.zeros((len(Uds),len(Ugs)))

    for nUds in np.arange(np.size(Uds)):
        for nUgs in np.arange(np.size(Ugs)):
            ID_trans[nUds,nUgs] = calc_ID(Ugs[nUgs],Uds[nUds],Usb)
        gm[nUds,:]  = np.gradient(ID_trans[nUds,:])/np.gradient(Ugs)

    return ID_trans,gm

def calc_gmb(Ugs,Uds,Usb):
    """Function to calculate the substrate transconductance
    Input
    ------
    Ugs     : ndarray
        gate-source voltage.
    Uds     : list
        drain-source voltage.
    Usb     : float
```

```
        source-substrate voltage.
    Output
    ---------
    gmb           : ndarray
        substrate transconductance.
    """


    NA = NA_slider2.value
    dox = dox_slider2.value

    Uth       = calc_Uth(Uth0,Usb,NA,dox)

    ID_sub    = np.zeros(len(Usb))
    for nUsb in np.arange(np.size(Usb)):
        ID_sub[nUsb] = calc_ID(Ugs,Uds,Usb[nUsb])
    gmb  = (np.gradient(ID_sub)/np.gradient(Uth))*(-np.gradient(Uth)/np.
↪gradient(Usb))

    return gmb

def update_gmb(*args,**kwargs):
    """Function to update Uth vs Usb plot.
    """
    Ugs_op  = Ugs_drop.value
    Uds_op  = Uds_drop.value
    Usb_op  = Usb_slider2.value

    index = np.argmin(np.abs(Usb_sweep-Usb_op))

    # calculate gmb
    gmb = calc_gmb(Ugs_op,Uds_op,Usb_sweep)

    # update gmb and line
    gmb_line[0][0].set_ydata(gmb*1e3)
    gmb_line[1][0].set_xdata(Usb_op)
    gmb_line[1][0].set_ydata(gmb[index]*1e3)

    # update limit
    gmb_max = np.amax(gmb)
    if gmb_max>0:
        ax_gmb.set_ylim(ymax=gmb_max*1.25e3)

    fig_gmb.canvas.draw()

def update_ID_gds(*args,**kwargs):
    [Ugs_op_min,Ugs_op_max] = Ugs_slider.value
```

```
    nOp = (Ugs_op_max-Ugs_op_min)/0.5 + 1
    Ugs_op = np.linspace(Ugs_op_min,Ugs_op_max,4)
    Usb_op = Usb_slider2.value

    [ID_out,gds]  = calc_ID_gds(Ugs_op,Uds_sweep,Usb_op)

    for i in range(len(Ugs_op)):
        ID_out_lines[i][0].set_ydata(ID_out[i]*1e3)
        ID_out_lines[i][0].set_label("{:.2f}".format(round(Ugs_op[i], 2)))

        gds_lines[i][0].set_ydata(gds[i]*1e3)
        gds_lines[i][0].set_label("{:.2f}".format(round(Ugs_op[i], 2)))

    ax_IDout.legend(loc='upper left',title='$U_{\mathrm{GS}}\mathrm{/V}$')
    ax_gds.legend(loc='upper right',title='$U_{\mathrm{GS}}\mathrm{/V}$')

    ax_IDout.set_ylim(ymin=0,ymax=np.max(ID_out*1.1e3))
    ax_gds.set_ylim(ymin=0,ymax=np.max(gds*1.1e3))

    fig_ID_gds.canvas.draw()

def update_ID_gm(*args,**kwargs):
    """Function to update transfer characteristics and transconductance plots.
    """
    # get new op points from slider
    [Uds_op_min,Uds_op_max] = Uds_slider.value
    Uds_op = [Uds_op_min,Uds_op_max]
    Usb_op = Usb_slider2.value

    [ID_trans,gm]  = calc_ID_gm(Ugs_sweep,Uds_op,Usb_op)

    for i in range(len(Uds_op)):
        ID_trans_lines[i][0].set_ydata(ID_trans[i]*1e3)
        ID_trans_lines[i][0].set_label("{:.1f}".format(Uds_op[i]))

        gm_lines[i][0].set_ydata(gm[i]*1e3)
        gm_lines[i][0].set_label("{:.1f}".format(Uds_op[i]))

    ax_IDtrans.legend(loc='upper left',title='$U_{\mathrm{DS}}\mathrm{/V}$')
    ax_gm.legend(loc='upper left',title='$U_{\mathrm{DS}}\mathrm{/V}$')

    ax_IDtrans.set_ylim(ymin=0,ymax=np.max(ID_trans*1.1e3))
    ax_gm.set_ylim(ymin=0,ymax=np.max(gm*1.1e3))

    fig_ID_gm.canvas.draw()

def update_all(*args,**kwargs):
```

```
    """Function to update transfer characteristics and transconductance plots.
    """
    update_ID_gds()
    update_ID_gm()
    update_gmb()

### Begin gmb vs Usb fig
# ------------------------------------
fig_gmb = plt.figure(figsize=(figW/2, figH),constrained_layout=True)
gs = fig_gmb.add_gridspec(1, 1, hspace=0.25, wspace=0.25,␣
 ↪height_ratios=[1],bottom=0.2)
ax_gmb = gs.subplots(sharex=False, sharey=False)
# fig_ID_gds.suptitle('$I_{\mathrm{D}}(U_{\mathrm{DS}})$'+' und␣
 ↪'+'$g_{\mathrm{ds}}(U_{\mathrm{GS}})$')
fig_gmb.canvas.header_visible = False
fig_gmb.canvas.layout.min_width = '400px'
fig_gmb.canvas.toolbar_visible = True
fig_gmb.canvas.capture_scroll = True

Ugs_op  = Uds_drop.value
Uds_op  = Uds_drop.value
Usb_op  = Usb_slider2.value
index = np.argmin(np.abs(Usb_sweep-Usb_op))

# calculate and gmb
gmb = calc_gmb(Ugs_op,Uds_op,Usb_sweep)

# plot gmb
gmb_line=[]
gmb_line.append(ax_gmb.plot(Usb_sweep, gmb*1e3))
gmb_line.append(ax_gmb.plot(Usb_op, gmb[index]*1e3, color='b', lw=2,␣
 ↪marker='o', ls=':'))

# set legend and label
ax_gmb.set_xlabel('$U_{\mathrm{SB}}\mathrm{/V}\;→$')
ax_gmb.set_ylabel('$g_{\mathrm{mb}}\mathrm{/mS}\;→$')

# set title
ax_gmb.set_title('$g_{\mathrm{mb}}(U_{\mathrm{SB}})$')

# set limits
ax_gmb.set_xlim(0, np.max(Uds_sweep))

fig_gmb.canvas.draw()
# end gmb vs Usb fig

### Begin output characterisitics and output conductance fig
```

```
# ------------------------------------------------------------
fig_ID_gds = plt.figure(figsize=(figW, figH),constrained_layout=True)
gs = fig_ID_gds.add_gridspec(1, 2, hspace=0.25, wspace=0.25,␣
 ↪height_ratios=[1],bottom=0.2)
(ax_IDout, ax_gds) = gs.subplots(sharex=False, sharey=False)
# fig_ID_gds.suptitle('$I_{\mathrm{D}}(U_{\mathrm{DS}})$'+' und␣
 ↪'+'$g_{\mathrm{ds}}(U_{\mathrm{GS}})$')
fig_ID_gds.canvas.header_visible = False
fig_ID_gds.canvas.layout.min_width = '400px'
fig_ID_gds.canvas.toolbar_visible = True
fig_ID_gds.canvas.capture_scroll = True

Ugs_op  = [1,1.5,2,2.5]
Usb_op  = Usb_slider2.value

# calculate and plot ID out and gds
[ID_out,gds]  = calc_ID_gds(Ugs_op,Uds_sweep,Usb_op)
ID_out_lines=[]
gds_lines=[]
for i in range(len(Ugs_op)):
    ID_out_lines.append(ax_IDout.plot(Uds_sweep, ID_out[i]*1e3,␣
 ↪label=str(Ugs_op[i])))
    gds_lines.append(ax_gds.plot(Uds_sweep, gds[i]*1e3, label=str(Ugs_op[i])))

# set legend and label
ax_IDout.legend(loc='upper left',title='$U_{\mathrm{GS}}\mathrm{/V}$')
ax_IDout.set_xlabel('$U_{\mathrm{DS}}\mathrm{/V}\;→$')
ax_IDout.set_ylabel('$I_{\mathrm{D}}\mathrm{/mA}\;→$')

ax_gds.legend(loc='upper right',title='$U_{\mathrm{GS}}\mathrm{/V}$')
ax_gds.set_xlabel('$U_{\mathrm{DS}}\mathrm{/V}\;→$')
ax_gds.set_ylabel('$g_{\mathrm{m}}\mathrm{/mS}\;→$')


# set title
ax_IDout.set_title('$I_{\mathrm{D}}(U_{\mathrm{DS}})$')
ax_gds.set_title('$g_{\mathrm{ds}}(U_{\mathrm{DS}})$')

ax_IDout.set_xlim(0, np.max(Uds_sweep))
ax_gds.set_xlim(0, np.max(Uds_sweep))
ax_IDout.set_ylim(ymin=0)
ax_gds.set_ylim(ymin=0)

fig_ID_gds.canvas.draw()
# end output characterisitics and output conductance fig

### Begin transfer characterisitics and transconductance fig
```

```
# ------------------------------------------------------------
fig_ID_gm = plt.figure(figsize=(figW, figH),constrained_layout=True)
gs = fig_ID_gm.add_gridspec(1, 2, hspace=0.25, wspace=0.25,␣
 ↪height_ratios=[1],bottom=0.2)
(ax_IDtrans, ax_gm) = gs.subplots(sharex=False, sharey=False)
# fig_ID_gm.suptitle('$I_{\mathrm{D}}(U_{\mathrm{GS}})$'+' und␣
 ↪'+'$g_{\mathrm{m}}(U_{\mathrm{GS}})$')
fig_ID_gm.canvas.header_visible = False
fig_ID_gm.canvas.layout.min_width = '400px'
fig_ID_gm.canvas.toolbar_visible = True
fig_ID_gm.canvas.capture_scroll = True

Uds_op = [0.1,3]
Usb_op = Usb_slider2.value

# calculate and plot ID and gm
[ID_trans,gm]  = calc_ID_gm(Ugs_sweep,Uds_op,Usb_op)
ID_trans_lines=[]
gm_lines=[]

for i in range(len(Uds_op)):
    ID_trans_lines.append(ax_IDtrans.plot(Ugs_sweep, ID_trans[i]*1e3,␣
 ↪label=str(Uds_op[i])))
    gm_lines.append(ax_gm.plot(Ugs_sweep, gm[i]*1e3, label=str(Uds_op[i])))

# set legend and label
ax_IDtrans.legend(loc='upper left',title='$U_{\mathrm{DS}}\mathrm{/V}$')
ax_IDtrans.set_xlabel('$U_{\mathrm{GS}}\mathrm{/V}\;→$')
ax_IDtrans.set_ylabel('$I_{\mathrm{D}}\mathrm{/mA}\;→$')

ax_gm.legend(loc='upper left',title='$U_{\mathrm{DS}}\mathrm{/V}$')
ax_gm.set_xlabel('$U_{\mathrm{GS}}\mathrm{/V}\;→$')
ax_gm.set_ylabel('$g_{\mathrm{m}}\mathrm{/mS}\;→$')

# set title
ax_IDtrans.set_title('$I_{\mathrm{D}}(U_{\mathrm{GS}})$')
ax_gm.set_title('$g_{\mathrm{m}}(U_{\mathrm{GS}})$')

# set limits
ax_IDtrans.set_xlim(0, np.max(Ugs_sweep))
ax_IDtrans.set_ylim(ymin=0)
ax_gm.set_xlim(0, np.max(Ugs_sweep))
ax_gm.set_ylim(ymin=0)

# draw figure
fig_ID_gm.canvas.draw()
# end transfer characterisitics and transconductance fig
```

```
### observe sliders
# -----------------
dox_slider2.observe(update_all)
NA_slider2.observe(update_all)
Usb_slider2.observe(update_all)
W_slider.observe(update_all)

Ugs_slider.observe(update_ID_gds)
Uds_slider.observe(update_ID_gm)

Uds_drop.observe(update_gmb)
Ugs_drop.observe(update_gmb)

### layouts
# ---------

center_layout = Layout(
#     flex_flow       = 'column',
    align_items     = 'center',
    width           = '80%',
    justify_content = 'center',
#     left = '150px'
)

param_layout = VBox([
                        HBox([Label('Veränderbare parameters für '+␣
 ↪'$I_{\mathrm{D}}$'+' und Kleinsignal-Parameters:')]),
                        HBox([W_slider]),
                        HBox([dox_slider2]),
                        HBox([NA_slider2]),
                        HBox([Usb_slider2]),
                ],layout=Layout(justify_content = 'center',width='36%',␣
 ↪align_items = 'center')) # ,layout=Layout(width='32%')

gmb_layout = VBox([
                    HBox([fig_gmb.canvas]),
                    HBox([Uds_drop,Ugs_drop],␣
 ↪layout=Layout(align_items='center',width='75%',justify_content='flex-start',))
                ], layout=Layout(align_items = 'flex-end',height='auto',␣
 ↪width='auto'))

output_layout = VBox([
                    HBox([fig_ID_gds.canvas]),
                    HBox([Ugs_slider], layout=center_layout)
                ], layout=Layout(align_items = 'flex-end', justify_content =␣
 ↪'center')) # ,layout=Layout(width='68%')
```

```
trans_layout = VBox([
                    HBox([fig_ID_gm.canvas]),
                    HBox([Uds_slider], layout=center_layout)
                ], layout=Layout(align_items = 'flex-end', justify_content =␣
 ↪'center'))


html_line = HTML(
  value="<svg height=\"400\" width=\"5\"><line x1=\"0\" y1=\"0\" x2=\"0\"␣
 ↪y2=\"400\" style=\"stroke:rgb(0,0,0);stroke-width:3\" /></svg>",
)

AppLayout(header=HBox([param_layout,output_layout]),
          left_sidebar=None,
          center=HBox([trans_layout,html_line,gmb_layout]),
          right_sidebar=None,
          footer=None,
          pane_widths=['0px', '1300px', '0px'],
          pane_heights=[5, 5, '1px']
         )
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home',␣
 ↪'home'), ('Back', 'Back to previous …
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home',␣
 ↪'home'), ('Back', 'Back to previous …
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home',␣
 ↪'home'), ('Back', 'Back to previous …
```

```
AppLayout(children=(HBox(children=(VBox(children=(HBox(children=(Label(value='Veränderbare␣
 ↪parameters für $I_{…
```

[ ]: