Lecture 16: Modeling with Athena and Atlas Part II

Announcements

Schedule Changes

- Since the second midterm was originally scheduled for dead week (Week 10), I have had to move it to Week 9.
- See AR16 of https://catalog.oregonstate.edu/regulations.
- The 2nd Midterm will hence take place in Weniger Hall 287 on Wednesday May 29th at 4:00pm.
- Lectures 19 & 20 will no longer take place.
- I will instead upload some slides with practice problems at the start of Week 9.
- I will provide some more details next week.
- There will be no lectures / exams in Week 10.
- Use this time for preparation of the final report.
Announcements

Laboratory Report 2/2:

- Details on the report are online now.
- I.e. regulations, length, content, grading criteria, etc.
- The report will be due at 4pm on Wednesday June 12th. This is in Finals Week and is 4 weeks from today.
- This will be a Group Report on the laboratory work you have carried out on field-effect transistors.
- You will submit one report per group.
- You will have the opportunity to provide feedback on your colleagues if they do not contribute fairly.
- Report 2 will contribute 35% to the final grade of ECE418 / ECE518.

Extra Information

- Hopefully the walk-through in these slides is enough information to get you started with Athena / Atlas.
- These pieces of software have far greater capability than we use in this class.
- The manuals for all relevant pieces of software are available from the course website:
- These will be useful sources of information.
Last Time

- We covered the basics of loading and running Athena.

Lecture 16

- Silvaco Code.
- Defining the Mesh.
- Defining the Substrate.
- Oxidation.
- Etching.
- Doping.
- Gate Oxide.
- Contacts.
- Setting up Atlas.
- Modeling Current-Voltage Characteristics.
Silvaco Code

Silvaco code.

- Before the break we saw that code is entered into the DeckBuild GUI:

- Athena and Atlas use the same syntax.
- In fact, both the Athena and Atlas parts of code are in the same file.
Syntax

- The code runs sequentially, from top to bottom.
- The line numbers are not show by default, but you can turn them on from the main menu bar: “View”→ “Show Line Numbers”.
- We will run code from a single file, with no functions, classes etc.
- Comments start the line with “#” and will be skipped over by the interpreter.
- There is no explicit line termination (c.f. “;” in c/c++ etc.). The new line character indicates a new line.

DeckBuild Commands

- Luckily you do not have to actually write too much code yourself.
- DeckBuild has a few built-in functions and dialogs which will help you to generate relevant code.
  - We will use these later.
  - All they do is automatically create code.
  - You could do the same thing by typing the code.
**Interpreter**

- Deckbuild uses an *interpreter* rather than a *compiler*.
- This means that rather than converting the entire program to machine code (1’s and 0’s) then executing it, Deckbuild will convert each line to instructions, line-by-line.
- You can run code line by line by hitting the “run to line button” then hit F10 to run the code one line at a time.

- There are various other buttons such as Kill, which will stop the program running.

**Output**

- The bottom part of DeckBuild shows the output.
- This will provide any important information that occurs while the code is running.
- If DeckBuild encounters a Syntax Error, an error box will be shown:
Empty Program

- We will start by running an empty project.
- Write the following into DeckBuild:

  ```
  go athena
  quit
  ```

- The output will look something like the following:

Defining the Mesh
Discretization

- We will to a certain-extent treat Silvaco like a “Black Box”.
- I.e. we specify a set of parameters and do not worry about the exact models used to get the results.
- But we do need to know a couple of things about the calculations however.
- The first is how space and time are discretized.
- A lot of methods (Monte Carlo, Finite Difference) rely on approximating a continuous function as a discreet one.
- E.g.:

\[
\int_a^b f(x) \, dx = \lim_{n \to \infty} \frac{b - a}{n} \sum_{i=1}^n f(x_i) \Delta x
\]

\[
\Delta x = \frac{b - a}{n}
\]

\[
x_i = a + i \Delta x
\]

- So to employ these methods we must divide up continuous space and time into finite sections (e.g. \( \Delta x \) and \( \Delta t \)).
- As long as these sections are small enough, the calculations will be accurate.
- We will first define a 2-dimensional spatial grid (in \( x \) and \( y \)).
- Each position on the grid we define as a node.
- We must make the grid fine enough that computation is accurate, but not too fine as the calculation takes too long.
- For most Athena operations we can approximate the processing time to scale with \((N_p)\alpha\), where \(N_p\) are the total number of nodes and \(\alpha\) is normally between 1.5 and 2.0.
Define the Mesh

- To define the mesh, first place your cursor where you want the code to appear:
- Then click "Commands" → “Mesh Define”
- The dialog should appear something like this:

The first thing to be aware of is that all dimensions are microns (despite appearing unlabeled).

We will start by creating a simple 1μm × 1μm simulation area with grid spacing’s of 0.1μm.

First, click the ViewGrid checkbox so the grid is shown:
Define the Mesh

- In the "X coordinates" group box enter:
  - Location = 0. Spacing = 0.10. Click insert.
  - Location = 1 Spacing = 0.10. Click insert.
- Do the same in the "Y coordinates" group box.
- You should see something like the following:

If you click the "write" button it will add the following code where your cursor was:

```
go athena
line x loc=0 spac=0.10
line x loc=1 spac=0.10
line y loc=0 spac=0.1
line y loc=1 spac=0.1
quit
```

- You could have equivalently written the above directly into DeckBuild.
- If you run this code the software will generate the mesh then quit.
Non-Uniform Mesh

- We can add more detail to certain parts of the mesh than others if we want to.
- E.g. we may want a finer mesh close to where we expect the channel to be.
- Let’s delete the code we just produced, then come back to the Mesh Define Dialog box. Enter the following:

![Non-Uniform Mesh Diagram]

Non-Uniform Mesh

- If you define different spacings at different positions, the software will ensure the spacing is correct at the position you specify, and interpolate the space between the other points.
- The code will come out as expected:
Non-Uniform Mesh

- You can make more very complicated meshes by adding more definitions of spacing:

- The number of points is given below the preview of the mesh.

Our Mesh

- For our example we want a large distance on the surface of the wafer, and fine resolution in the $y$-direction.
- These are outside the bounds allowed by the dialog box, so we just enter the code directly:
- After entering a section of code, it is good practice to run the program and make sure you have not made any errors.
- We can’t plot what we have yet.
Defining the Substrate

Substrate
- The next step in our simulation is to define the wafer.
- In particular we are going to define:
  - The crystal orientation.
  - The dopant type.
  - The resistivity / dopant concentration.
- This is the properties of the wafer as we receive it (i.e. before we do anything to it).
- We can either specify the doping concentration as provided by the manufacturers, or we can define the resistivity (as measured) and the software will work out the concentration.
Substrate

- To define the substrate properties go to “Commands” → “Mesh Initialize”.

- This should bring up the dialog box show to the left.

- There are a lot of options, we will just enter the parameters as shown.

- See p33 of the Athena User Manual for more details.

Substrate

- If you hit “write” it should produce something like the following:

- As usual, it is good practice at this stage to run the code and make sure you have not made any typos.

- We have now provided Athena with enough information to plot a graph.
Plotting Doping Concentration

• Add the following line to output the data to a structure file:

```plaintext
structure outf=DopingConc.str
```

• DeckBuild show now look something like the following:

![DeckBuild interface](image)

ECE 418 / 518 – Semiconductor Processing
Spring 2019 - John Labram

---

Plotting Doping Concentration

• If we go out of DeckBuild and back to the UNIX file system and use the `ls -l` command we should see that we have now created this file (among several other files):

```
find / -name DopingConc.str
```

• Go back into DeckBuild and add the following line of code below where we define the structure:

```plaintext
tonyplot DopingConc.str
```

• This will call TonyPlot to plot the DopingConc structure.
Plotting Doping Concentration

• The code should now look like this:

• If you run the code it will complete then a few seconds later display the TonyPlot splash screen:

• A few seconds later the plot should be displayed.

Plotting Doping Concentration

• The data should look like this:

• We have doping concentration (cm$^{-3}$) on the y-axis and depth (in μm) on the x-axis.

• The wafer is described as uniformly doped.

• Since we only have one dopant type, the boron concentration is here the same as the net doping concentration.

• This plot can then be exported as described in the last lecture.
Oxidation

Thermally Grown Oxides

- We are roughly following the process you followed in the laboratory.
- The next step is to grow an oxide.
- There are two ways you can grown an oxide in Athena:
  - Diffusion.
  - Deposition.
- We will carry it out using diffusion first. We talk about deposition later.
- Place the cursor where we want then click “Commands” → “Process” → “Diffuse”.

Lecture 16 – Wednesday May 15th 2019
Thermally Grown Oxides

- The dialog has a few options. We will just cover a few basics.
- Consult the manual for more details.
- The first tab looks like this:
- Hopefully all these parameters are self-explanatory.
- We will just carry out a simple anneal at 1100°C for 60 minutes.

Thermally Grown Oxides

- We will use a dry O₂ atmosphere at 1atm, with no impurities.
Thermally Grown Oxides

- You can also select specific models here.
- It is best to just leave the values as their defaults unless you have a reason not to.
- The best way to understand the effect of different parameters is to run the simulation and compare results.

With the settings we described, we output this line of code:

```
diffus time=60 minutes temp=1100 dryo2 press=1
```

The TonyPlot output will then look something like this:
Thermally Grown Oxides

- So far we should have the following code:

```python
DeckBuild - "http://ttk/users/labram/AthenaCode/TempFile.in" @ Flip3.engr.oregonstate.edu

# Define the mesh
line x loc=0 spac=50
line x loc=100 spac=3
line x loc=200 spac=3
line x loc=300 spac=5

line y loc=0 spac=0.025
line y loc=2.5 spac=0.025
line y loc=4.75 spac=0.025
line y loc=10 spac=1.875

# Define substrate (resistivity is in Ohm cm)
init silicon boron resistivity=10 orientation=111

# Thermally grow the oxide
diffuse time=60 minutes temp=1000 dryo2 press=1

# Save and plot the data
structure out="DopingConc.str"
topodifflo DopingsConc.str
quit
```

Etching
Simple Etch-All

- There are a few different ways to etch a film off the surface of a wafer. Here we will just use the simplest approach.
- We will just remove all of the specified material (oxide) in the positions we define.
- Make sure your cursor is where you want the code to appear.
- On the main menu click “Commands” → “Process” → “Etch” → “Etch...”

Simple Etch-All

- This brings up the simple Process Etch dialog box.
- We will define use the “Geometrical” etch method, just allowing us to define where we want the etch to occur.
- If you just leave the “Geometrical Type” as “All” then click the “Write” button the following code will be generated:

```plaintext
etch oxide all
```
- I.e. all the oxide is removed from the surface of the wafer.
Simple Etch-All

- If you run the program now, the following plot will be created:

  ![Plot](image)

  - I.e. we have removed all the oxide, so we are back to our doping concentration diagram.
  - Notice that thanks to oxidation, the doping is no longer uniform with depth.

Etch Specified Regions

- We can instead remove just some of the substrate
- In this example we just remove the oxide to the left of $x = 50 \mu m$.
- This would create the following code:
  
  ```
  etch oxide left p1.x=50
  ```
- You can edit the code by hand if the values are outside normal limits.
- We will change the position to $100 \mu m$:
  
  ```
  etch oxide left p1.x=100
  ```
Etch Specified Regions

- If you run the program now, the following plot will be created:

- By default TonyPlot will output the most logical information given what you provide.
- Here we see that we have etched all the SiO2 to the left of $x = 100 \, \mu m$.

Etch Specified Regions

- Either by using the dialog or by editing the code directly, let's do the same to the right side of the wafer.
- The code should now look like this.
- The TonyPlot should look like:
Doping

Dopant Deposition

- Athena is not really designed to model spin-on dopants unfortunately.
- It is more designed for ion-implantation (as used in industry).
- We are just going to deposit some SiO$_2$ with a very high impurity concentration.
- We will then drive in the dopant and remove the SiO$_2$ afterwards.
- It is not a perfect approach, but good enough for what we want to achieve.
Dopant Deposition

- As usual, make sure the cursor is in the correct position first.
- To open the dialog box click “Commands” → “Process” → “Deposit” → “Deposit”.

We will just deposit 0.5 μm of SiO₂ (just labelled “Oxide”).
- We will set the grid layers (divisions) to 10.
- We will set the impurity concentration to $10^{21}$ cm⁻³.
Dopant Deposition

- We will also adjust our code so we can see the output before and after the deposition.

- Now, the program should output two separate TonyPlot windows by the time it completes.

Output

- You should see two windows appear that will look something like the following:

- So by default TonyPlot will just give us a two-dimensional map of which material is located where.
Output

- To see the doping concentration, right click on the graph and click “Display” from the menu.
- This will then bring up the Display menu:

  ![Display menu](image)

  - Click the “Contours” button so it is depressed then click Apply to show doping concentration as a function of position.

  ![Contours button](image)

  - Click “OK” to accept and close the dialog box.

Output

- Our two plots should then look like the following:

  ![Before and After plots](image)

  - So you will notice that all of the dopant resides in the new oxide layer we have deposited on top of our old one.
Drive-In

- To drive in our dopants we anneal the wafer again.
- As before, place the cursor where we want then click "Commands" → "Process" → "Diffuse".
- Here we are going to anneal for 30 minutes, at 1100°C, but this time we will anneal in N₂ (not O₂, H₂O as for SiO₂ growth).

Output

- This time we will look at the dopant distribution before and after drive in. The code should look like:
Output

- This calculation is a bit more intensive, so will take a few minutes to conclude.
- After turning on “Contours” in TonyPlot, as we described before, you should see two windows appear that will look something like the following:

![Before][After]

Etching

- The oxide we had previously on the substrate was used as a diffusion mask.
- Since we have now deposited the dopant and carried out the drive-in diffusion process, we can now remove it.
- You could either use the dialog as before, or simply write the following line of code:

```plaintext
etch oxide all
```

- This just removes the oxide from all positions on the surface of the wafer.
Output

- After removing the oxide the dopant concentration and code should now look something like the following:
Gate Oxide

• We have created the wells for our field-effect transistor.
• The next stage is to grow our gate oxide.
• This is similar to our diffusion-mask oxide, but thinner.
  • We also should always grow the gate oxide in an $O_2$ environment (not $H_2O$ in case that was used earlier).
• We will follow a similar process to described earlier, but we anneal for just 20 minutes.
• We should get this line of code:

```
diffus time=20 minutes temp=1100 dryo2 press=1
```

Output and Code

• At this stage the code and doping density should be as shown here.
Zooming

- You can use the magnifying glass button to zoom in on certain areas of your wafer.

Etching

- The oxide growth model, will grow the oxide everywhere.
- We just need it over the channel, so we need to etch the oxide from over the wells (and everywhere else).
- The easiest way to do this is just to add the following two lines of code before we plot the graph.

```
etch oxide left p1.x=100
etch oxide right p1.x=150
```
- We could just copy and paste this from further up in our code.
Contacts

Electrodes

• We have created the wells for our field-effect transistor.
• To finish off the device we need to apply contacts so we can connect the device to the outside world.
• There is no masking option in Athena, so we are just going to deposit the aluminum everywhere, then etch it off after.
• We will just use the deposit dialog: click "Commands" → "Process" → "Deposit" → "Deposit".
Electrodes

- The electrode thickness is not that important, but we will set it to 200nm:

Etching

- Since we did not really etch off the aluminum in our case, we just need to remove it, and not worry about how we would actually carry out such an etch in reality.
- We need three electrodes: source, drain, gate.
Etching

- We will use the normal etch tool (on the main menu click "Commands" → "Process" → "Etch" → "Etch...").
- However we will select “Any Shape” from the Geometrical type dropdown. We will select “Aluminum” from the Material dropdown.
- This allows us to define a region to remove aluminum from.
- We enter vertices of the region to etch.

Etching

- To work out the vertices of the region we want to etch, we should zoom in on our sample.
- Let’s start by etching the edge of wafer (to the left of the left electrode).
- Let's choose the following points:
  - \( x = 0.0, y = 0.2 \).
  - \( x = 20.0, y = 0.2 \).
  - \( x = 20.0, y = -0.2 \).
Etching

- Enter each of these vertices into the box in the correct order.
  - \( x = 0.0, y = -0.2. \)
  - \( x = 0.0, y = 0.2. \)
  - \( x = 20.0, y = 0.2. \)
  - \( x = 20.0, y = -0.2. \)
- You can select the “Bottom” radio button to make sure that each value you enters appears at the bottom of the list.
- As usual, the dialog has small limits for dimensions, so it can be easier to enter this as code directly.

If you click write in the dialog box, you will produce the following 4 lines of code:

```
etch aluminum start x=0.0 y=-0.2
etch cont x=0.0 y=0.2
etch cont x=20.0 y=0.2
etch done x=20.0 y=-0.2
```

- The result plotted will then look like this.
- Next we have to remove the other regions of the aluminum.
Etching

- The process is very similar, so we will here just provide the code. All the etching code (including what we just defined) is shown to the right.

- Follow a similar procedure to last time if you are unsure.

```python
#Etch to the left of the source
etch aluminum start x=0.00 y=-0.20
etch cont x=0.00 y=0.20
etch cont x=20.00 y=0.20
etch done x=20.00 y=-0.20

#Etch between the source and gate
etch aluminum start x=60.00 y=-0.20
etch cont x=60.00 y=0.20
etch cont x=100.00 y=0.20
etch done x=100.00 y=-0.20

#Etch between the gate and drain
etch aluminum start x=150.00 y=-0.20
etch cont x=150.00 y=0.20
etch cont x=190.00 y=0.20
etch done x=190.00 y=-0.20

#Etch to the right of the drain
etch aluminum start x=230.00 y=-0.20
etch cont x=230.00 y=0.20
etch cont x=250.00 y=0.20
etch done x=250.00 y=-0.20
```

If all goes well the wafer should look like the following:

- This is our MOSFET built!
Defining Electrodes

- Before we can model the current-voltage properties of this MOSFET, we need to first define where the electrodes are.
- It is obvious to us, but not to the modelling software.
- To bring up the dialog box, click “Commands” → “Structure” → “Electrode...”

The dialog is hopefully reasonably self-explanatory.

- The electrodes names that we provide will be used by Atlas to simulate current-voltage behavior.
- Because of the dimension limits, it is easier for us just to enter this information as code:

```
#Define Electrodes
electrode name=gate x=125 y=0.0
electrode name=source x=40 y=0.0
electrode name=drain x=210 y =0.0
```

- We just need to pick a single point in the relevant electrode.
Setting up Atlas

Atlas

- Once we are done with Athena, we can go ahead and start Atlas.
- To do this, we use a similar statement as to what we used with Athena:
  
  ```command
  go atlas
  ```

- Note, we do not need close Athena before we run Atlas.
- Atlas used device physics models to evaluate expected current-voltage characteristics.
- It has far greater capability than we are using for this course.
- The manual is available from the course website.
- As with Athena, we will treat Atlas as a “black box” model.
Materials

- DeckBuild does have some tools relating to Atlas, but in general it is going to be easier for us to just enter the remainder of the code by hand.
- The first thing we need to do is define which material we are using.
- To do this we add the code:

  ```
  material material=silicon
  ```

- Atlas has all the relevant information for this material stored internally:
  - Electron & hole mobilities, SRH lifetimes, band gap, conduction and valence band energies, etc...

Materials

- If you wish to adjust parameters manually, you can do this with commands such as:

  ```
  material taun0=1e-11 taup0=1e-11
  ```

  This sets the Schottky-Read-Hall lifetimes to 10ps each for holes and electrons (this is short).

  Equivalently you can put everything on the same line:

  ```
  material material=silicon eg300=1.12 mun=1100
  ```

  Here we define the material as silicon, its band gap at 300K as 1.12eV (band gaps are temperature-dependent) and the electron mobility to be 1100 cm²/Vs.

  Try changing the mobility to 1cm²/Vs (~rough value for amorphous silicon) and see what happens to IV curves.
Initialization

- We have to setup a couple of things before we can use Atlas to calculate IV properties.
- First we need to define the gate electrode (the source and drain will come later).
- To do this, add the following line of code:

  ```
  contact name=gate
  ```

- This assumes that you named the gate electrode “gate” when we defined it in Athena.

Model

- We get to define the model that Atlas uses to solve the electrostatic equations, and calculate electronic properties.
- We are not going to discuss the details of models here. More information can be found on page 76 of the Atlas Manual.
- We are just going to use the Newton Method.
- It is more versatile than other methods, but can take longer.
- To specify we are using the Newton model, add the following line of code:

  ```
  method newton
  ```
Initialization

- Before we can start calculating IV curves, we have to initialize the solver (i.e. tell it we have now chosen a model, material system etc.).
- To do this just add the following code:

```
solve init
```

- The Atlas code so far should look like this:

```
solve atlas
  f opt: material and model
  material material=silicon eg300=1.12 mu=1100
  contact name=gate
  method: aou tout
  solve init
```

- We can now finally calculate current-voltage characteristics.

Modeling Current-Voltage Characteristics
Current-Voltage Characteristics

- We are now finally able to simulate how we expect source-drain current to change as function of applied voltage.
- We will here calculate output curves to show that the devices do indeed behave like transistors.
- I.e. the results should look like the figure shown to the right.
- Transfer characteristics can be calculated using an analogous approach.
  - E.g. see the example from the last lecture (mos1ex01.in).

Defining the Gate Voltage

- We are now finally able to simulate how we expect source-drain current to change as function of applied voltage.
- Start by defining the gate voltage we will apply.
- Write the following line of code to apply $V_{GS} = 1.0\text{V}$:
  
  ```
solve vgate=1.0
  ```
- The solve statement, tells the software to solve the electrostatics for this applied voltage.
  - You will notice the interpreter will stay on this line for a while as it calculates the solution.
  - Atlas will remember which was the last applied gate voltage for any subsequent calculations.
  - It is important the order of solve vgate statements is correct.
Defining the Output File

- As with our structures generated by Athena, we need to define the file the output will be saved to.
- Here we do it before we calculate the current.
- We save the data to a "log" file that TonyPlot can read.
  - We can export the data later.
- Write the following code to define the output:
  
  ```
  log outf=Vgs1.log master
  ```

- When the code runs, it will generate a file called "Vgs1.log" in the same directory as we were when we started DeckBuild.
- We will instruct TonyPlot to plot this data later.

Calculating the Curve

- To calculate the current-voltage behavior we write the following line of code:
  
  ```
  solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain
  ```

- This is the specific syntax that Atlas uses to evaluate current.
- The text "vdrain=0 vstep=0.2 vfinal=10.0" is similar to a for loop, and tells the model to evaluate the current at voltages between 0V and 10V, in steps of 200mV.
- The text "vdrain=0 vstep=0.2 vfinal=10.0" is similar to a for loop, and tells the model to evaluate the current at voltages between 0V and 10V, in steps of 200mV.
  - In this example there are 51 data points.
  - The more data points specified, the longer the calculation.
Calculating the Curve

- You will notice we also have to define the electrode via the "name=drain" command.
- This should be the same name you defined earlier in your Athena code.
- When the interpreter gets to this line it will stay here for a while (several minutes).
- Once complete, the data should be saves to the file we defined earlier (Vgs1.log in our case).

Displaying the Curve

- We can load the graph manually by loading TonyPlot.
  - Click the below icon on the toolbar:
  - This should bring up an empty TonyPlot window.
  - Click “File” → “Open” from the main menu to bring up the open dialog.
Displaying the Curve

- Locate the file we saved in the code earlier.
- For us it was “Vgs1.log”
- Alternatively, we could add the following code to the end of the program (before `quit`) to open it automatically:

  ```
  tonyplot Vgs1.log
  ```

- The Atlas code so far should look like that shown to the right.

Displaying the Curve

- Initially the plot will look like the image to the right:
- By default TonyPlot will plot \( I_D \) on the \( y \) axis and \( V_G \) on the \( x \) axis. Hence the data makes no sense.
- To change the display, right-click anywhere and select Display from the menu.
Displaying the Curve

- The dialog works in the same way as any normal graphing software.
- We just want to change the x-axis to "Drain Voltage":
  - Everything else can stay the same.
  - Click ok to accept and close the dialog.

The curve will look something like the following:

- At a low gate voltage like this the device is not really on.
Multiple Gate Voltages

- When measuring / simulating an output curve, it is usual to stack multiple current sweeps on-top of each other.
- We can do this by simply repeating similar code to before.
- E.g., we could increase the gate voltage to 2V:

```
solve vgate=2
log outf=Vgs2.log master
solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain
```

- Then the same for 3V:

```
solve vgate=3
log outf=Vgs3.log master
solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain
```

- In total our Atlas code should look something like this:

```
# gut material and model
material material-wilicon ep=200.12 mu=110
contact name=gate
method=em
solve init

#Calculate IV curves
solve vgate=1
log outf=Vgs1.log master
solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain

solve vgate=2
log outf=Vgs2.log master
solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain

solve vgate=3
log outf=Vgs3.log master
solve vdrain=0 vstep=0.2 vfinal=10.0 name=drain

#plot results
plotout -overlay Vgs1.log Vgs2.log Vgs3.log Vgs4.log Vgs5.log
```

- To run the entire program (Athena + Atlas) will take roughly 10-20 minutes.
Multiple Gate Voltages

- The data will again come out making no sense initially.
- After you change the $x$-axis to drain voltage:

![Graph of Multiple Gate Voltages](image)

- The plots look good apart from the extraneous points near the end (this is basically the start of the next sweep repeated).

Exporting Data

- We need to export the data so we can plot it with proper graphing software (or Excel).
- As we described in the last lecture, click “File”→”Export” to bring up the dialog box.

- Make sure to select “Comma-Separated Variables” from the drop-down menu.
Exporting Data

- As we described in the previous lecture, I suggest navigating to a windows directory to save the data.
- TonyPlot will output each line as a separate file, so make sure the base filename is recognizable.
- When ready the dialog should look something like the following.
- I am saving directly to my (Windows) desktop.

Analyzing Data

- If we open the data in Excel it will look like this:

- In notepad it looks like:
Analyzing Data

- If you scroll to the bottom in Excel you will notice there is an extra point where the gate voltage is 1V too high.

- We can delete this row.
- We may as well delete all other columns beside “Drain Voltage” and “Drain Current”
- The spreadsheet will look like this now.

Analyzing Data

- We can repeat the process, and put all the sweeps into a single sheet:

- Just remember to label the different gate voltages at the top.
Channel Width

• You may have noticed that we have not explicitly defined the channel width in Athena or Atlas.
• This is because by default this software is carrying out a 2-dimensional simulation.
• Implicitly, the channel width in the simulation is 1 μm.
• This means that you will need to multiply your output current by whatever width your device has in reality.
  • E.g. if you channel width is 200 μm, you must multiply the source-drain current by 200 to get realistic values.
• Explicit simulations of channel width do exist, but are beyond the scope of this course:

Corrected Data

• Multiplying every current value in the sheet brings the peak currents up to ~mA levels.
• Plotting this data with standard graphing software will give you something like what is shown to the right.
Summary

- In these slides we have covered all the steps required to generate current-voltage curves from your process parameters.

Next Time...

- We will talk about ion implantation.
Next Time...

- In these slides we have covered all the steps required to generate current-voltage curves from your process parameters.