On the evaluation of line scans obtained with DektakXT using Python

This document shall provide some basic knowledge on how to conduct the data evaluation on line scans of references for magnetron sputtering. All code written is to be seen as a starting point for the venture of potentially fully automatizing the whole process one day, rather than a finished product. This manual will first briefly explain the intended workflow. In a second step possible errors will be presented. An in-depth explanation of the algorithm will not be provided.

Basic Idea

To gain access to the distribution of material deposited onto a sample using magnetron sputtering a reference is required. In the present case, the reference was a 4” silicon wafer with a grid of 5 mm x 5mm printed onto it, using lithographic methods. Subsequently, material is deposited onto such a reference. After further processing, only the negative of the initial grid is left on the wafer. This is, squares of 5mm x 5mm of deposited material are separated by trenches, reaching down to the bare silicon wafer. Using a profilometer, the height of each square is measurable and thus a height map can be obtained.
Originally, this was done by measuring every single square, followed by its e scan valuation. Since this is a slow process, a new approach is desirable. Conducting line scans might be a first step.

Grid and Coordinate System

In Fig.1 on the right side a schematic of a 4” wafer is shown. The coordinate system implemented should be kept for all measurements intended to be evaluated with the python scripts provided.

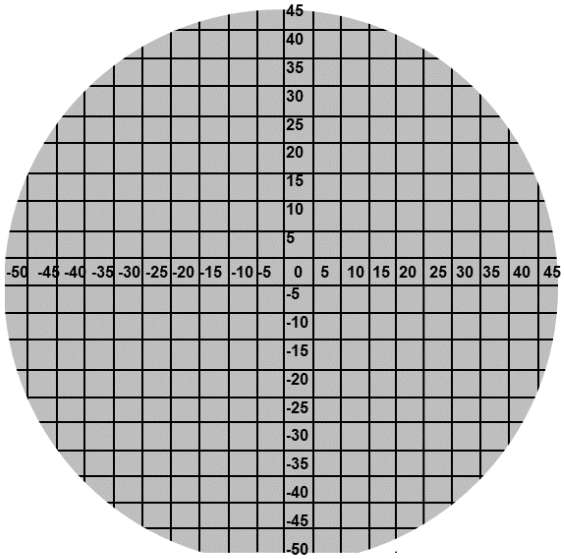

Figure 1: Coordinate system of choice. The long, flat edge of the wafer must be oriented downwards

Conducting Line Scans

Unfortunately, a 4” wafer is too large for the DektakXT to be measured with a single line scan, spanning across the whole wafer. Furthermore, line scans can only be done in the direction of +y. As a result, 20 line scans starting at y = 0 for every x-coordinate will be followed by another 20 line scans starting at the very bottom and ending at y = -5. Thus, the whole wafer is covered.

It is important to note, that all line scans must be exported in the “.csv” format, with a name in the form of:

{Name of reference}_x{Start x coord.}_y{Start y coord}_to_y{End y coord.}.csv

Data Evaluation: Part I

To start the data evaluation, two python scripts will be made use of: One for the evaluation itself, the other for plotting the results. Visual Studio Code (VSC) is a great code editor to work with. (Note: These scripts are python files, not jupyter notebooks).
Open VSC and hit “Ctrl” + “K” + “O” to chose your working directory. This should be the directory where all line scans were saved. The reason for this is, that VSC saves files to its current working directory. Running the scripts from outside this directory is possible, but all files produced by the script will then be saved to this working directory. This might result in troubles finding the resulting files again later.
Once the right working directory is selected, a side bar will open on the left side of the screen. All files contained in this working directory can be found there. Clicking on them opens a preview directly in VSC. First of all, double click on the python script (python scripts are indicated by the “.py” ending) called “read_profilometer.py”. Additionally, check that the correct version of python is chosen. To do so, check the lower left corner of VSC’s window. In the orange banner the version of python is written. Make sure to click on it and select “Python 3.8.3 64-bit (‘base’:conda)”, if it is not already selected.

Summarizing this script: First, all individual segments are identified by the algorithm. In the next step, a linear function is fitted to each segment. This function is then subtracted from the original data and the corrected version is obtained. Based on this, the mean and standard deviation of the height is calculated. Results are then saved as images and .csv files.

Between code lines 10 – 25 user input can be specified. Threshold variables are used for filtering out outliers later on. In essence, these variables are factors determining the interval:

[mean – factor * standard deviation, mean + factor * standard deviation]

Values not contained in this interval will be filtered out, during the calculation of the final mean. The script calculates three kinds of means, based on intervals of data themselves. These intervals are specified via the boundaries variables.

1. Using the “whole” square. Default: from 10% to 90% of the square, to not take the edges into account)

2. Using the periphery. Default: from 5% to 10% and from 90% to 95%. Edges are still outside of these intervals, but strange behaviors might be found there, still.

3. Using the center. Default: from 45% to 55%.

Previously not mentioned is the variable “peak_detection_level”. Changing this impacts the way the single squares, also referred to as segments, are identified by the algorithm.
The next variable is the step size, which is the width of a single square in millimeters.
Lastly, the size of the header of the exported csv. Line scans needs to be specified. For this, open a line scan .csv and count the rows of metadata. The number of the first row containing real data - 1 should be used to define the variable “rows_to_skip”.

The next area of user input starts at code line 294. Here, the name of the intended file (the line scan) has to be specified. Once this is done, hit “F5”. The first doing so after VSC was launched, you also have to select “Python File” in the drop down window, which should appear. Then, a terminal will open at the bottom of the window. Also, a small slider (see right) will appear.

This indicates the script is running. Hitting the blue pause symbols pauses the execution of the code, the yellow arrow restarts it and the red square aborts it.
Once this slider disappears, the execution is finished.

If everything works out, output should be displayed in the terminal at the bottom.
First, a list of all found files in the current working directory is output. Then, the name of the current file. Following this, the assumed steps in the line scan and the by the algorithm automatically found steps are displayed. They should be roughly the same for everything to work properly.

During the execution of the script, if no errors occur, new windows displaying plots should pop open. In Figure 1 of these figures, the identified segments can be seen, as indicated by the vertical red lines.
Here, the position of each line should be checked. In general, the identified segments should always be seen as a suggestion by the code, rather than the final value.

For the case not all segments were identified correctly, if statements need to be added manually to the script, starting at code line 310. Each if statement has to be structured as follows:

if filename == *{"the name of the file"}***:** ← the name of the file here, don't forget " ".
    manual_list = *[x, x, x, x,]* ← the x coords, where the segments should be
    cutsites = main(filename, manual_list = manual_list)

Just copy the exemplary if-statement and fill in the cursive parts. The easiest way to do so is to copy the filename from terminal (Note: use "Ctrl" + "Shift" + "C", since the standard "Ctrl" + "C" will abort every running code!). Also, copy the list of the identified segments into the just-created if-statement. By comparing to Figure 1 of the opened figures, estimate where the segment's border should be. This of course should always be roughly in the middle of each trench. Then, exchange wrong values or add missing values in the list inside the if-statement. Restart the script (yellow arrow) for the changes to be implemented. Once all windows with images pop up, check the results. If everything seems fine, specify the next file in code line 299 and either abort the currently running script and subsequently start it again, or restart it directly.

It is best to create these if-statements even for line scans where all segments were identified correctly. This ensures a reproducibility of the results, even if the variable "peak_detection_level" was changed.

For each identified segment, the script creates a new .csv file containing the coordinates of this segment as well as the results of the linear function fitted to it. Additionally, an image of the original segment and the corrected segment is saved. Furthermore, the overview in Figure 1 of the opened figures is also saved. Most importantly, a file containing all segments' coordinates and calculated means and standard deviations is saved, too.

Examples of Errors

When it comes to errors in python, it is always a good idea to google them. Most errors are quite straight forward. Otherwise you could ask e.g. Lukas. It is usually always the same kind if errors occurring, so trying to understand them helps a lot.

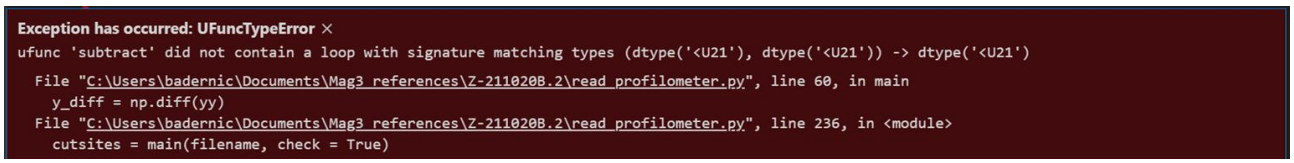

Figure 2: The number of rows to skip might be wrong (variable "rows_to_skip", code line 25

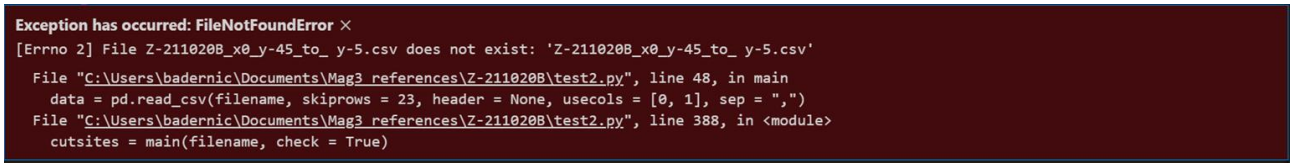

Figure 3: The file name you specified is most likely not correct

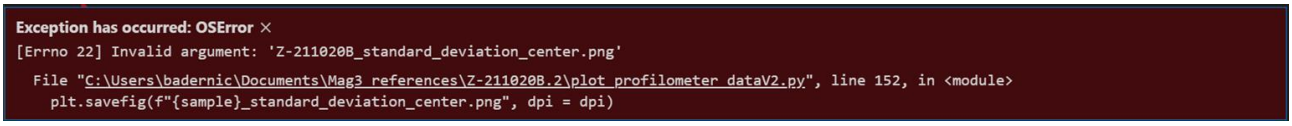

Figure 4: Python is trying to create a file (.csv or .png), but most likely you have this excat file still open. Close it and restart the script

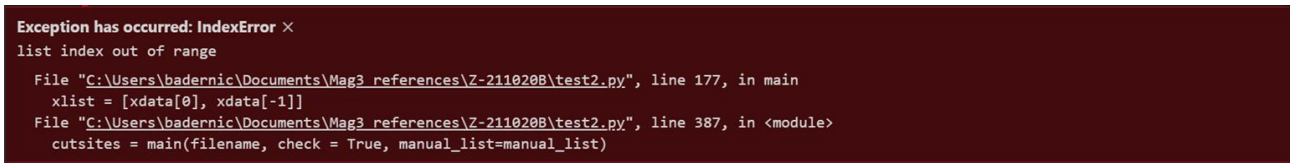

Figure 5: Essentially this error tells you, that the amount of assumed and identified/manually specified segments does not match somehow

Data Evaluation: Part II

In the sidebar in VSC, double click on “plot_profilometer_data.py”. Hit “F5”. There is nothing to specify in this script. If something is off, the terminal will either display this or you will see it in the resulting images. This script is written for the exact coordinate system displayed in Fig.1 above.

Last Comments

It is important to keep in mind that, once you run the script, the resulting files to be saved will without further warning overwrite your previous results, if existing. Previous results are not recoverable. This is why you should write if-statements for all line scans, because then you can quickly recreate the lost files.
Meddling with “.csv” files to much might also make it impossible for python to read them properly.