ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Simulation of Quantum Dots

Group project
Laboratory for Nanoelectronics
Prof. Vanessa Wood
Integrated Systems Laboratory
Department of Information Technology & Electrical Engineering
ETH Zurich

Handed in by
Christian Funck & Matthias Dittberner

Date
May 19, 2013

Supervisor: Prof. Vanessa Wood
Prof. Mathieu Luisier

Christian Funck
Plattenstrasse 71
8032 Zurich
09-920-177
cfunck@student.ethz.ch

Matthias Dittberner
Luegislandstrasse 269
8051 Zurich
09-913-450
mdittber@student.ethz.ch

Contents

Chapter 1

Simulations with TOM

TOM was programmed to simulate spherical nanocrystals and to visualize the results of the simulations. Though we focused on spherical structures, the code is hold abstract and slim to make extending to other structures, such as nanowires, fairly easy. The main aim of TOM is to offer OMEN users a toolbox, which includes the following 3 main tasks:

1. Automatization of the OMEN simulation process
   When it comes to simulating quantum dots with different parameters users do not want to spend a lot of time on writing command files for OMEN by themselves and start each OMEN task via the shell, but rather enter the main parameters into a Graphical User Interface (GUI) and let TOM do the rest (see section 1.2.1). This makes overnight simulations for large parameter sets possible.
2. Overview of all simulations done in the past
   All information respectively parameters of past simulations can be displayed in a GUI. Exporting selected simulations and visualizing them is possible as well (see section 1.2.2)
3. Visualization of simulation data
   All different kinds of plots are available within the toolbox, such as visualizing band gaps, wave functions or quantum dot structures (see section 1.2.3).

1.1 Installing the Software

****
****** TO DO !!! ******
****

1.2 Using TOM

1.2.1 Running a simulation

Type gui_simulate in the MATLAB command window and hit enter. The window as in figure 1.1 will open. A simulation set is defined as one row of the table, i.e one material with all kinds of sweeps. You can add more simulation sets by using the *Add rows to the table* panel. It is possible to copy and paste single cells of the table using the appropriate short cuts of your MATLAB default keyboard setup (`CTRL+C & CTRL+V` Windows setting, `ALT+W & CTRL+Y` Emacs setting). The columns are filled as follows:

1. Material
   Enter the number for the material you would like to simulate, according to the *Material / Compound* list.

2. Geometry
   Enter the number of the geometry given in the *Geometry* list. Very important in the case of materials with shells is, that you have to enter the geometry type of the core and the shell. The geometry types are separated with a comma.

   Example. For a spherical CdS-CdSe quantum dot the cell would look like this: 1, 1

3. Radius
   The radius has to be entered in a specific way. The syntax is:

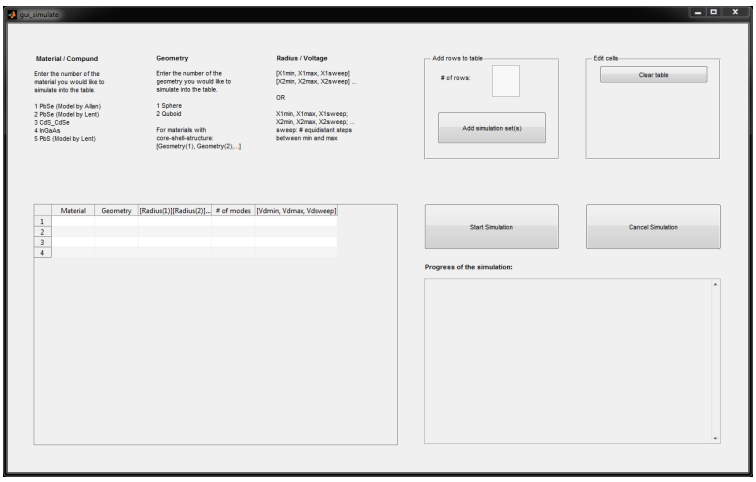

Figure 1.1: The gui_simualte window

`[Rmin(1), Rmax(1), Rsweep(1)][Rmin(2), Rmax(2), Rsweep(2)]...`

`Rmin(i)`	smallest radius of the i^{th} material to be simulated
`Rmax(i)`	largest radius of the i^{th} material to be simulated
`Rsweep(i)`	number of equidistant points between `Rmin(i)` & `Rmax(i)`

Example. Simulation a single spherical PbS quantum dot with radius 3.5 nm you would enter: [3.5,3.5,1]

Example. For a spherical CdS-CdSe quantum dot the cell would look like this: [1,4,4][5,6,2] Looking at the radius parameters, TOM will do all the permutations and generate therefore 8 quantum dots:

Quantum dot	1	2	3	4	5	6	7	8
Core radius in nm	1	1	2	2	3	3	4	4
Shell radius in nm	5	6	5	6	5	6	5	6

4. # of modes
   Enter a number of modes you would like to calculate.

5. Voltage
   The Voltage sweep is entered in the same way as the radius. You find more information under the following remark.

Remark (Vectors & Matrices). As the MATLAB GUIs do not accept vectors or matrices as it is known from the command window, the parameters have to be entered as a string and are converted to matrices later on.
There are different ways how to enter the parameters. Use the one, that is the most clear for you.

```
 a,b,c,...,d                 or
[a,b,c,...,d]                becomes a double vector

V = [ a b c ... d]


[a,b,c][d,e,f]...[g,h,j]     or
 a,b,c; d,e,f;...;g,h,j      or
[a,b,c; d,e,f;...;g,h,j]     becomes a double matrix

M = [ a b c
      d e f
      . . .
      g h j ]

where a,b,...,j are doubles as strings.
```

These input styles can be applied to column *Geometry*, *Radius* and *Voltage*.

You can use as many spaces as you want in between. The first and the last brace are not necessary if only a vector is entered or if rows are separated by semicolons.

After entering all the necessary parameters proceed by clicking `Start Simulation`. The *Progress of the simulation* panel will keep you informed about the warnings, wrong entered parameters and the current status of the simulation.

1.2.2 Display simulation information

There are two ways of displaying simulation information, either you can see the whole database (all simulations that are stored in the Simulations folder) or only specific data, for example only PbS simulation data.

Remark. TOM uses objects (called *qdotObj*) to store all important data of simulations such as the simulation parameters, date of simulation etc. (futher information section 1.3.1). All operations are done using these objects. If you would like to display only certain simulation data, you would need an *qdotObj* array. How to create such an array please read section 1.2.4.

Displaying the whole database can be done by typing `gui_db` into the MATLAB command window. A window as in figure 1.2 will appear showing all parameters and technical information of each simulation. Please note, that according to the size of the database, it might take some seconds to load all data. If you only want to display a set of simulation data, you can also call `gui_db(qdotObj_ARRAY)`, where `qdotObj_ARRAY` is an array of qdotQbj objects.

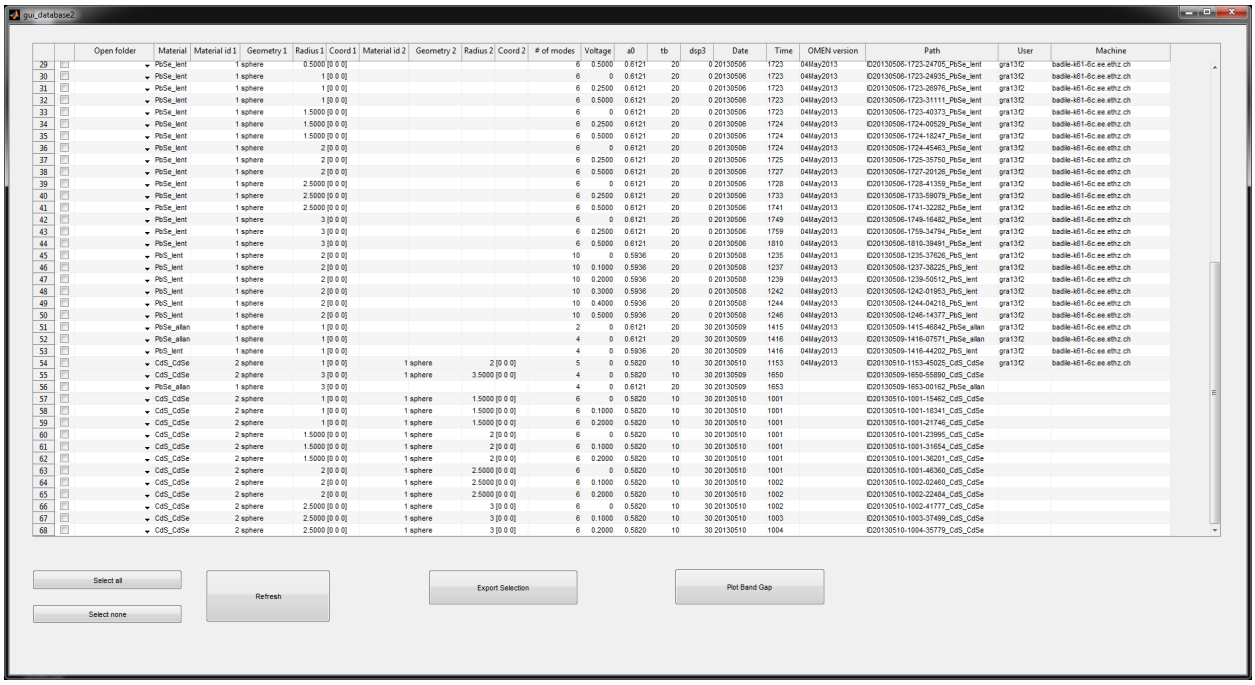

Figure 1.2: The gui_database window

Within the GUI you can sort the simulations with the column header, select simulations and let plot them, open the directory of a simulation or even export a selection, which is available as an qdotObj array in the main workspace entitled *ExportedDB*.

1.2.3 Visualization

As explained in the previous section, simulation data can be visualized through the *gui_ database* window, but also manually. In this section the plotting functions are explained.

```
1 plotBandGap(DB)
```

1.2.4 Additional tools

1.3 Maintaining TOM

1.3.1 General Structure

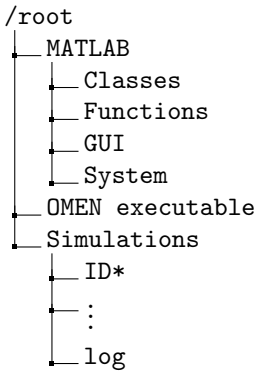

Figure 1.3: The TOM structure by default

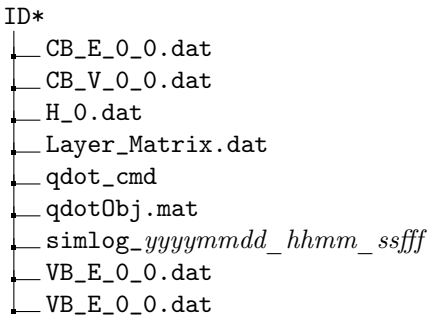

Figure 1.4: Structure of a simulation set

```
IDyyyymmdd-hhmm-ssfff_MATNAME
```

```
struct global config {
  config.root        = '';
  config.system      = '';
  config.log         = '';
  config.user        = '';
  config.machine     = '';
  config.OMEN        = '';
  config.simulations = '';
  config.vOMEN       = '';
  config.cancelSim   = '';
}
```

Chapter 2

MATLAB

2.1 Install

2.2 User

2.3 Maintainer

2.3.1 Basic concepts

Parameters for the simulation are stored and passed as arguments to functions as objects of the class *Qdot*, further refered to as QDO. The class provides properties for all parameters neccessarry for the simulation, as well as properties for administrative purposes, such as the specific simulation folder. Parameters relating to the geometry are stored in an object of subclass *Geometry*. Since more than one material is possible, the geometry property is often an array of objects of class Geometry. To instantiate a QDO, one can call the constructor with no arguments, which will create an empty QDO. Alternatively, a string with the material name can be passed as an argument, in which case the new QDO object will be constructed based on parameters defined in an external file, located in the folder *Classes*. The class Qdot provides some methods for basic displaying of some selected parameters, such as *getSelParams*.

Example. Creating a *Qdot* object based on default parameters and set the radius to 4

```
myQdot = Qdot('CdS_CdSe');
myQdot.geometry(2).radius = 4;
```

2.3.2 GUI

2.3.3 Simulation

The simulation parameters from the GUI are passed to the functions which will then start the simulation with OMEN. The parameters are stored in QDOs. Since it is often desirable to simulate over a range of parameters, some parameters (i.e. the radii and e-field) are at this stage not scalars, but a vectors, containing starting value, end value and number of steps in between. Such a generic QDO, is further refered to as QDOG.

The QDOG is passed to the function *simAll.m*, which will simulate all desired parameter combinations, by performing the following steps:

- An array of QDOs (QDOA) with all combinations of the parameter is created in the function *sweep.m* from QDOG. Supported sweep parameters are radius and electric field. This could easily be extended to other parameters by modifying the function *sweep.m*.
- The elements in the QDOA are then simulated one after the other, and all data saved in seperate folders, called ID*timestamp_material*:
- The OMEN command-file *qdot_cmd* is written, using the function *writeCmdFile.m*.
- The OMEN simulation will be started using the MATLAB function *unix*, which calls the operating system to execute the specified command.

- A logfile is written, recording the duration and the success of the simulation as well as the console output of OMEN. The success of the simulation is checked by inspecting whether the desired files were created.

After the simulation of all elements in the QDOA, an additional logfile is written, and saved in the folder *log*. It gives information about the success of all simulations, and thus provides an easy way to check if and which simulations failed.

Returned to the calling function is the QDOA as well as a vector indicating the success of every simulation.

Note that it is not strictly necessary to start the simulations using the GUI. The QDOG containing the desired parameters can also be created with standard MATLAB syntax, and passed subsequently to the function *simAll.m*, as is shown in the following example.

Example.

```
myQdot = Qdot('PbS_lent');
myQdot.geometry.radius = [1.5 3 4];   changing the parameter radius
simAll(myQdot);                       4 simulations with radii 1.5, 2, 2.5, 3 will be performed
```

2.3.4 Database

In order to know which parameter sets already have been simulated, there are some useful tools, which can be found in the folder *Functions/ QdotUtils*. The basic principle is to get all all parameters which were simulated, which can then be displayed in the GUI, filtered, deleted and so on.

Getting the parameters from all simulations

This is done by loading the QDOs of all performed simulations from their folders, and storing them in a QDOA.

function *getQDOA*()

Filtering

function *filtered = filterQDOA(QDOA, propertyName, value, mode, tol)*

The filtering can be applied to any QDOA, and returns a subset of this array, matching specified criteria. The argument *propertyName* specifies the *Qdot* property which is compared to *value*. The filter criteria are specified by selecting a mode of filtering. The following filtering modes are available:

- the property exactly matches *value*.
- the property lies within a range of values, specified by a vector: *value = [min max]*
- the property approximately matches *value*. For numeric properties this is specified using a tolerance *tol*. For string properties the *value* should be a substring of the property.
- filter for a constant difference between two properties. The property names are specified in a cell array: *propertyName = {propertyName1, propertyName2}*
- filter for a constant ratio between two properties.

The last two modes are especially interesting for selecting QDOs with two or more materials, to find the objects with a specified shell-thickness.

Example.

myQDOA = getQDOA;
filter for radius = 3.5
filtered = filterQDOA(QDOA, 'geometry(1).radius', 3.5, 1, 0);
filter for constant difference of 0.7nm between radius of material 1 and radius of material 2, within a tolerance of +/- 10%
filtered = filterQDOA(QDOA, {'geometry(1).radius', 'geometry(2).radius'}, 0.7, 4, 0.1);
filter for material containing Pb:
filteres = filterQDOA(QDOA, 'mat_name', 'Pb', 2, 0);

2.3.5 Plotting

Here follows a short description of functions which can be used for basic visualisation of the data obtained by the OMEN simulation. For a more detailed description, please refer to the code.
Note: These functions are based on the directory structure created by the simulation function *simAll.m*, i.e. to work properly, the simulation data, as well as the corresponding QDO must be located in their own folder, with the folder name specified in Qdot.path property.

Plotting the eigenenergies

Plotting the wavefunction

These functions all take an array of *Qdot* objects (QDOA) as an input. Additionally, the number of eigenmodes to be displayed has to be specified, as well as the band (conduction or valence band). The visualisation will then be done for every one these objects, and for all specified eigenmodes.

function ***plotEV3D****(QDOA, band, NMod)*
Plot the atoms of the quantum dot, their color indicating the probability density of an electron or hole. Red corresponds to high, blue to low probability.

function ***plotEV3DcrossSection****(QDOA, NMod)*
This function produces similar plots to the above, but it plots two cross sections of the quantum dot, for valence and conduction band respectively, in one window.

function ***plotEV3Dmax****(QDOA, band, probLim, NMod)*:
Again very similar to the *plotEV3D*, but the color code is simplified. The atoms with very high probability densities are red, the ones with high probability yellow, the rest transparent. The color is determined in the following way: The sum of the probabilities of all red atoms is smaller than a probability value specified in *probLim*. An analogous argument is applied for the yellow marked atoms.
This function makes it a lot easier to see how the wavefunction roughly looks like and changes from one mode to the next.

function ***plotEVAlongAxis****(QDOA, propertyName, startPoint, direction, plotGrid, tolerance, NMod, band)*
Plot the probability density along an arbitrary axis through the crystal. The data for all elements of QDOA is plotted in the same plot, thus making it easier to compare quantum dots with different parameters. The axis is specified through *startPoint* and *direction*, including a tolerance, which is the maximum distance which an atom can deviate from the specified line. Depending on the direction, the tolerance has to be adjusted to include a sufficient number of atoms. To check this, it is useful to specify the input argument gridPlot, which will plot the atoms, the chosen axis, and highlight the atoms on the line in red. However, this function is probably only suitable for large quantum dots. Furthermore an averaging over neighbouring atoms would be recommendable.

function ***compareEV****(QDOA, band, NMod, tol, propertyName, showGrid)*: plots the same as *plotEVAlongAxis*, but for three different directions (x,y,z axis), and arranges them in subplots in one figure.

Appendix A

Q & A

Why is opening gui_db is not possible? It might be that there are failed simulations in the database. Please delete the according folders and try again.

Why does the simulation not stop, when I hit *Cancel Simulation*? MATLAB is blocked during the time it has send a command to the shell. You might have to abort the process by pushing `CTRL+C` a couple of times.

Declaration of Originality

We hereby declare that the written work we have submitted entitled

Simulation of Quantum Dots

is original work which we alone have authored and which is written in our own words.

Authors

Last name	First name
Dittberner	Matthias
Funck	Christian

Supervisors

Last name	First name	Degree
Luisier	Mathieu	Professor
Wood	Vanessa	Professor

With the signature we declare that we have been informed regarding normal academic citation rules and that we have read and understood the information on *Citation etiquette* (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected. The above written work may be tested electronically for plagiarism. [1]

Place and date	Christian Funck	Place and date	Matthias Dittberner

[1] Based on the official Declaration of ETH Zurich: http://www.ethz.ch/faculty/exams/plagiarism/confirmation_en.pdf